UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

Relazione

Le relazioni tra le Accademie Italiane tra il 1525 e il 1700

Candidato :*Jalalledin Taavoni*

Relatore : *Prof. Vittore Casarosa*

Prof.ssa Maria Simi

Prof.ssa Enrica Salvatori

Anno Accademico 2017-2018

UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

Relazione

Le relazioni tra le Accademie Italiane tra il 1525 e il 1700

Candidato :*JALALLEDIN TAAVONI*

Relatore : *Prof. VITTORE CASAROSA*

Prof.ssa MARIA SIMI

Prof.ssa ENRICA SALVATORI

Anno Accademico 2017-2018

INDICE

INTRODUZIONE **8**

1. Capitolo 1 **9**

1.1 Il mondo delle Accademie 9

1.2 Un po' di Storia 9

1.3 Accademie italiane 10

1.4 Accademie e ambiente politico 11

1.5 Accademie e religione 12

2. Capitolo 2 **13**

2.1 Il mondo dei Grafi 13

2.2 La Teoria dei Grafi 13

2.3 Definizione di Grafo 14

2.4 Grafi orientati / non orientati 14

2.5 Modelli di dati a grafo 15

2.5.1 Grafo delle proprietà 15

2.5.2 Grafo RDF 17

2.5.3 Ipergrafo 18

2.6 I database orientati ai grafi 19

2.6.1 Graph DBMS 20

2.6.1.1 Storage sottostante 21

2.6.1.2 Processing Engine 21

2.6.2 Graph compute Engine 21

2.7 Storia dei database NoSQL 21

2.7.1 Tipi di database 23

2.7.1.1 Key-Value store 23

2.7.1.2 Document-oriented 23

2.7.1.3 Column Family store 23

2.7.1.4 Graph DBMS 23

2.8 Il database grafo Neo4j 24

2.8.1 Architettura 25

2.9 Cypher 26

2.9.1 Pattern Matching 27

2.9.2 Le Clausole MATCH e RETURN 27

2.9.3 La clausola WHERE 27

2.9.4 Le Clausole ORDER BY, LIMIT e SKIP 28

2.9.5 La Clausola WITH 28

2.9.6 La Clausola UNION 28

2.9.7 Le Clausola CREATE, MERGE, SET, DELETE 28

2.9.8 Il Cypher Query 29
2.9.8.1 Vincoli di proprietà uniche 29
2.9.8.2 indice di database 29
2.9.8.2.1 Esempio di un query dalla nostra database per Academia : 30
2.9.8.2.2 Esempio di un query dalla nostra database per libro : 31
2.9.8.2.3 Esempio di un query dalla nostra database per Autore: 31
2.9.8.2.4 Esempio di un query dalla nostra database per Member : 31
2.9.8.2.5 Esempio di un query dalla nostra database per le relazioni tra autori e sue opere: 31
2.9.8.2.6 Esempio di un query dalla nostra database per le relazioni tra le accademie e sue pubblicazione: 32
2.9.8.2.7 Esempio di un query dalla nostra database per le relazioni tra autori e member: 32
2.9.8.2.8 Esempio di un query dalla nostra database per le relazioni tra accademie e member: 32
2.9.8.2.9 Esempio di un query dalla nostra database per le relazioni tra autori e accademie: 32

3. Capitolo 3 **32**
3.1 Relazioni delle Accademie Italiane 32
3.1.2 La banca dati delle accademie italiane 33
3.2 Obiettivo 34
3.3 Fasi del procedimento e strumenti 34
3.3.1 Architettura dell'informazione 34
3.3.2 User experience design 35
3.3.3 Fase 1 35
3.3.3.1 MySQL 35
3.3.3.2 PhpMyAdmin 36
3.3.3.3 Arrows - graph diagram library 38
3.3.3.4 Importazione di dati da un database relazionale 39
3.3.3.4.1 Carico CSV 39
3.3.3.4.2 Impostazioni di configurazione per gli URL dei file 40
3.3.3.4.2.1 Dbms.security.allow_csv_import_from_file_urls 40
3.3.3.4.2.2 dbms.directories.import 40
3.3.3.4.3 Formato file CSV 40
3.3.3.5 Importa i dati da un file CSV 40
3.3.3.6 Importa i dati da un file CSV contenente intestazioni 41
3.3.3.7 ETL Tool 42
3.3.3.8 Neo4j-etl 42
3.3.3.8.1 ETL Tool gestione della struttura 42
3.3.3.8.2 non dimenticare le chiavi 42
3.3.3.8.3 cosa costituisce una chiave? 43
3.3.3.8.4 JoinTables - Sono JOIN o tabelle? 45

3.3.3.8.5 Entità intermedie 46
3.3.3.8.6 Applicazione delle regole di mappatura durante l'esportazione 48
3.3.3.8.7 Diagramma di architettura 48
3.3.4 Fase 2 51
3.3.4.1 Bootstrap 51
3.3.4.2 D3 52
3.3.4.3 Vis 52
3.3.4.4 Git 52
3.3.4.5 Visual Studio Code 52
3.3.4.6 Live-server 52
3.4 Outcome 55
Conclusione e sviluppi futuri **57**
Bibliografia e sitografia **58**

Ringraziamenti

Vorrei ringraziare il Prof. Casarosa e la Prof.ssa Simi e anche la Prof.ssa Salvatori, per la supervisione della mia tesi, consulenza preziosa e collaborazione utile sulla valutazione dei risultati.

Vorrei ringraziare la mia fidanzata per essere stata gentile e solidale quando ho trascorso il tempo sulla mia tesi e non con lei.

INTRODUZIONE

"Prima furono le selve, poi i campi colti, e i tuguri, appresso le piccole case, e le ville, quindi le città, finalmente le Accademie."

Le "Accademie italiane 1525-1700" è un progetto di ricerca che per quattro anni è stato finanziato dal "Consiglio per le scienze umanistiche del Regno Unito (2010-2014)", che prevedeva una collaborazione tra tre principali istituzioni di ricerca: Royal Holloway University of London; the University of Reading; the British Library.
Uno dei risultati principali del progetto, a cui ha largamente contribuito Simone Testa, è un database completo di informazioni sulle accademie di tutta la penisola italiana, con dettagli sulla loro appartenenza e pubblicazioni.Obiettivo di questa tesi è quello di analizzare il database "Le accademie italiane 1525-1700" e visualizzare le relazioni che riguardano le accademie in grafi, per poter rispondere a questa domanda:

Quali sono le relazioni tra le accademie, gli autori e le loro opere tra il 1525-1700?

Questa tesi cerca di conciliare due aspetti, il primo umanistico e il secondo informatico. La parte umanistica riguarda le accademie italiane tra 1525-1700, invece la parte informatica presenta uno strumento per la gestione di grafi, Neo4j.

La Tesi è organizzato a come segue:

Il primo capitolo presenta le accademie italiane nel periodo 1520-1700, al fine di consentire al lettore di comprendere il database per poi seguire meglio il progetto.
Il secondo capitolo ha l'obiettivo di fornire informazioni sul il mondo dei Grafi, Neo4j, le sue caratteristiche principali e il suo linguaggio.
Il terzo capitolo presenta lo sviluppo di un'applicazione Web per visualizzare i dati e illustra i modelli che abbiamo adottato. Questo capitolo è diviso in altre parti: una prima parte per mostrare l'architettura interna del database su cui si è effettuata la modellazione per mezzo di grafi e gli strumenti utilizzati. Una seconda parte, che spiega l'interfaccia utente e le sue funzionalità. La terza parte spiega cosa succede internamente nell'applicazione quando la usiamo.
L'ultimo capitolo spiega cosa abbiamo conseguito e i modi in cui la nostra applicazione potrebbe essere ulteriormente ottimizzata ed estesa.

1. Capitolo 1

1.1 Il mondo delle Accademie

In questo capitolo viene introdotta la storia dell'Accademia, il suo inizio e suoi significati nel tempo. Successivamente vedremo gli aspetti politici e religiosi e le principali tipologie, e alla fine parleremo del database usato per il progetto.

1.2 Un po' di Storia

"Accademia" è una parola ambigua, in primo luogo perché è difficile trovare un'unica definizione, esistono tanti diversi esempi per un lungo periodo di tempo, e secondo perché il significato del termine indica sia un luogo che una riunione di persone.
Bruno Migliorini tracciò una storia della parola e scoprì che il suo uso iniziò all'inizio del XV secolo come descrizione di un luogo vicino ad Atene dove Platone si riuniva con i suoi allievi. Il significato della parola come luogo appartato appare per la prima volta in una lettera di Poggio Bracciolini in cui descrive la sua casa come "Accademia Valdarnina" (1434) . In seguito, la parola viene a definire il raduno delle persone. Ciò è visibile nella lettera di Donato Acciaiuoli che descrive la riunione di persone che conoscono l'umanista greco Argyropoulos o ancora più nel raduno di Marsilio Ficino a Careggi, vicino a Firenze. Nel 1612 troviamo la prima definizione della parola nel Vocabolario degli Accademici della Crusca: "Accademia, Lat. *Academia*, Setta di filosofi, dal luogo dove primieramente si adunò. Oggi: adunanza d'huomini studiosi. "
"Academia" che significa una riunione o un incontro pubblico o privato della gente è nel verbale dell'Accademia dei Ricovrati (*The Sheltered Ones*) il 12 marzo 1600.
Alla fine del diciassettesimo secolo, nei verbali dell'Accademia Risvegliati (*The Awakened Ones*) a Pistoia, troviamo l'espressione "fare accademia", con il significato di "Riunirsi". In un altro contesto, Giacinto Gimma definisce l'andirivieni di studiosi nella casa di Magliabechi una "continua Accademia", un incontro continuo e informale di persone.
Riassumendo, nell'Italia del sedicesimo e diciassettesimo secolo, "accademia" definisce uno spazio e una riunione più o meno formale di persone che si riuniscono allo scopo di condividere le proprie conoscenze, ma potrebbe anche significare un rappresentazione teatrale eseguita da un gruppo; Per lungo tempo questa parola ha definito uno spazio sociale in cui un gruppo di persone si riuniva per condividere conoscenze, esperienze e informazioni senza la necessità di redigere regole di adesione o statuti.

1.3 Accademie italiane

Oltre 800 accademie, o società colte, fiorirono in Italia tra il XVI e il XVII secolo, formando un aspetto significativo e influente della cultura sociale e intellettuale.
Multidisciplinari nei loro interessi, che collegavano le arti e le scienze, operavano all'esterno, ma erano spesso interconnesse con istituzioni ufficiali come università, tribunali, organismi politici e religiosi e offrivano una forma di associazione più flessibile, apparentemente "libera" e "uguale".

Membri o affiliati potevano a volte includere figure socialmente marginali come donne e artigiani. Le accademie attrassero anche intellettuali stranieri con le loro reti estese in tutta Europa, fornendo un modello di associazione per altre nazioni da emulare, adattare o rifiutare.
Tuttavia, i documenti storici delle Accademie italiane sono attualmente frammentati in molti archivi e biblioteche diversi e sono spesso di scarsa visibilità, in parte a causa della quantità e della varietà dei dati sopravvissuti e della loro dispersione geografica, ma anche perché molte accademie sono esistite solo per brevi periodi di tempo.
Le accademie rappresentano una dimensione vitale e caratteristica della prima cultura moderna. C'erano circa 600 Accademie in Italia nel periodo 1525-1700. Nei secoli XVI e XVII le Accademie italiane erano responsabili della promozione del dibattito e della discussione in molte discipline diverse, dalla lingua e letteratura, attraverso le arti visive e dello spettacolo, fino alla scienza, tecnologia, medicina e astronomia. Questa vasta gamma di interessi è stata accompagnata da una vasta gamma di concetti e modelli diversi di Accademia. Alcune erano formalmente costituite, con regole pubblicate e liste di membri; altre erano gruppi molto più informali di individui con idee simili, spesso giovani uomini, con interessi comuni; alcune hanno condotto ricerche su argomenti particolari, altre hanno spaziato nell'intero spettro delle arti e delle scienze; alcune godevano del patronato e della partecipazione delle autorità politiche, altre operavano più in privato e persino clandestinamente.
Sperimentazione scientifica, dibattiti politici, recitazioni di poesia, recite di drammi sono tutte da scoprire tra le attività delle Accademie. Le Accademie funzionavano come istituzioni alternative alle università e ai tribunali e contavano tra i loro membri scienziati, scrittori, artisti, pensatori politici e rappresentanti di entrambi i sessi e di tutte le classi sociali. Le Accademie avevano anche un aspetto più giocoso, ideando per l'accademia e per ogni membro nomi divertenti che erano spesso rappresentati visivamente in illustrazioni e dispositivi punitivi. Internazionali in appartenenza e in corrispondenza con studiosi di tutta Europa, furono fondamentali per lo sviluppo delle reti intellettuali più tardi definite come la République des Lettres e per la diffusione di idee nella prima Europa moderna. La gamma di interessi e l'enorme numero di Accademie e le loro pubblicazioni rendono queste istituzioni centrali per lo studio della prima cultura europea moderna.
Le caratteristiche più importanti e più comuni delle accademie dal 1525 al 1700, oltre alla promozione stessa della cultura che coltivavano al loro intero, erano in realtà le reti, la socialità, la circolazione della conoscenza e la *seria ludere* - o giocosa serietà - che prendeva forma tramite comunicazione orale, manoscritti, supporti cartacei, rappresentazioni teatrali, concerti o tornei.
Uno degli aspetti secondo cui dovremmo interpretare le accademie italiane è la socialità. Come ci ha insegnato Maurice Agulhon, si deve studiare la socialità sia storicamente che geograficamente, e le accademie italiane rappresentano un modo privilegiato per osservare con precisione la cultura della socialità, perché la loro organizzazione variava in base al tempo e allo spazio. Dopo l'Accademia Platonica e Accademia Grande di Siena, la creazione di accademie in Italia divenne una tendenza che si diffuse attraverso la penisola dalla fine del XV secolo. Comprendeva sia incontri formali che meno formali e ha influenzato gruppi simili in altri paesi europei come la Francia e Spagna. Quindi credo che il fenomeno delle accademie italiane dovrebbe essere paragonato ai movimenti sociali contemporanei e alle reti e possono quindi essere concepite come reti di individui e gruppi che hanno facilitato la diffusione di idee e conoscenze sotto forma di interazioni sociali e pubblicazioni.

1.4 Accademie e ambiente politico

Mentre le accademie spesso si autodefinivano "repubbliche di lettere" o "società" con le loro regole e le loro convenzioni, inevitabilmente dovevano impegnarsi a un certo livello con le istituzioni locali. Nella penisola italiana politicamente frammentata degli inizi del 1500, queste istituzioni potevano andare dalle poche repubbliche che ancora rimanevano ai tribunali e ai centri principeschi governati da monarchi stranieri, con una varietà di istituzioni civiche intermedie.

Le accademie potrebbero aver svolto un ruolo utile generando propaganda culturale per le autorità politiche (e connesse autorità religiose) e servire come ambiente adatta a favorire la rete politica. Il caso dell'Accademia degli Umidi (*The Moist Ones*) a Firenze è stato spesso visto come emblematico del modo in cui alcune accademie si trasformavano nel tempo da istituzioni private a istituzioni pubbliche o comunque fortemente legate al potere politico, per sopravvivere. Negli anni successivi alla sua fondazione (1540), a partire da undici cittadini di modesta estrazione sociale e appartenenza, i programmi culturali e gli obiettivi dell'Accademia degli Umidi sono stati continuamente riconfigurati, sempre più in linea con la politica ducale. L'Accademia, con i suoi elementi più sovversivi neutralizzati, divenne così parte integrante del regime Mediceo come Accademia Fiorentina. Altrove, le autorità locali potevano essere ostili e timorose nei confronti delle accademie. A Napoli tutte le accademie furono temporaneamente chiuse per ordine del viceré, Pedro de Toledo, nel 1547, con reiterazioni di divieti per tutto il secolo, fino a quando la "pubblica" Accademia degli Oziosi (The Lazy Ones) fu fondata nel 1611.

La politica del governo dello stato dove si trovava la singola Accademia - era spesso esplicitamente vietata come argomento di diretta discussione negli statuti accademici. Tuttavia, queste istituzioni svolsero ugualmente un ruolo importante nel riunire élite potenti ma a volte senza diritti politici diretti, nella promozione di rapporti di mecenatismo e promozione delle loro idee a diversi livelli della società, attraverso conferenze pubbliche, dibattiti, spettacoli teatrali, schemi iconografici e pubblicazioni a stampa, e potevano spesso fornire opportunità per un coinvolgimento politico più o meno diretto. Allo stesso tempo, le accademie spesso adottavano internamente pratiche di anonimato, con il coinvolgimento di "segreti", velati e spesso non riconosciuti; spesso discutevano di idee problematiche, di autori come Lucrezio, Machiavelli, Erasmus (Brown, Faini, Pallini, Moroncini) o anche di idee scientifiche che sfidavano le vedute cattoliche ortodosse (Bottari). La loro relazione, a volte ambivalente, con la cultura e la stampa ufficiali li rendono - lontani da ossequiosi produttori di encomi cortigiani e versi frivoli - spazi contestatori portatori di una cultura contraria e persistente di idee "repubblicane", legate all'ideale di libertà intellettuale e aperte al dibattito nella repubblica delle lettere.

Teatro e performance, che sono stati un aspetto chiave delle attività culturali delle accademie, allo stesso modo possono fornire un sottile indicatore delle affiliazioni politiche di un'accademia e del suo livello di eterodossia religiosa e culturale, come nel caso dell'Accademia degli Innominati di Parma (The Unnamed Ones), come discusso nel saggio di Lisa Sampson. Questa istituzione prosperò nel periodo in cui la dinastia dei Farnese cominciava ad affermare con più forza la sua politica di controllo sul loro nuovo stato di Parma e Piacenza e sulla scena internazionale, legittimata dalla sua autorità all'interno della Chiesa cattolica post-tridentina. I Farnese riconobbero fin dall'inizio l'importanza delle attività culturali degli Innominati per rafforzare il loro regime, tenuto conto anche della mancanza di una università locale e delle relazioni ambivalenti con l'aristocrazia feudale locale.

Il potenziale di questa accademia per l'autonomia e il dibattito culturale sembra essere stato

influenzato considerevolmente dallo stile individuale di dominio e dalle ambizioni dei vari Duchi Farnese e dalla sua apertura ai membri "stranieri". Come le relazioni tra i Farnese e le élite locali - in particolare gli aristocratici feudali - divennero di più problematiche, le creazioni teatrali drammatiche varie e talvolta audaci dell'accademia divennero sempre più controllate.

1.5 Accademie e religione

Le accademie si sono dimostrate arene importanti anche per la negoziazione di relazioni all'interno di un sistema di istituzioni tra cui organizzazioni religiose, favorendo anche una eterodossia di tipo religioso e filosofico. Questa eterodossia poteva essere espressa in modo indiretto attraverso un uso creativo della letteratura e delle arti visive, ma anche in termini di apertura delle accademie a idee che sfidavano le dottrine della Chiesa, come nell'emblematico esempio di Galileo Galilei. Queste tendenze si notano particolarmente nelle accademie di tutto il Veneto, a partire dagli esempi della metà del XVI secolo - un periodo segnato dalla circolazione continua di riformisti o evangelici. Da notare, al volgere del diciassettesimo secolo, le cosiddette "guerre culturali" tra la Chiesa romana, in particolare rappresentata dai gesuiti, e lo stato veneziano con la sua maggiore apertura alle idee "libertine", che erano particolarmente coltivate all'Accademia degli Incogniti di Venezia.

A volte le accademie potevano essere più apertamente opposte alle autorità politiche o religiose, come dimostra Sangalli in un saggio che esplora il ruolo delle nell'esplorazione del ruolo cercato dalle accademie padovane verso la fine del sedicesimo secolo, nell'ambito dell'istruzione, quindi un campo principale di battaglia ideologica tra Chiesa e Stato. Questo saggio si concentra su Paolo Beni, ex-gesuita espulso dall'ordine nel 1593, nonché professore all'Università di Padova e membro di due accademie locali: l'Accademia degli Animosi (The Courageous Ones, 1573-76) e dal 1600 l'Accademia dei Ricovrati (The Recovered Ones) a cui appartenevano sia Cesare Cremonini che Galileo Galilei. Le varie le varie istituzioni a cui Beni è stato associato nel corso della sua vita si sono talvolta intrecciate in modi conflittuali e hanno plasmato la sua particolare prospettiva umanistica cristiana sincretista, che combinava aristotelismo e platonismo e idee fastidiose sulla grazia e sul libero arbitrio.

Significativamente, fu il modello di una "accademia ibrida" che ispirò la audace proposta di Beni nel 1619 di riformare le pratiche educative e la società nell'area padovana istituendo una "Congregazione per l'ottimo governo dell'università". Come aveva già' fatto per quanto riguardava una precedente proposta di "Accademia di Lettere" che combinava elementi di un'accademia secolare e di un collegio gesuita, Beni proponeva per la sua Congregazione un nuovo linguaggio vernacolare e letterario, metodi multidisciplinari, umanistici, nonché ruoli pastorali, disciplinari e professionali - ma senza il contributo delle autorità veneziane. Sebbene alla fine ignorate, queste proposte di istituzioni che potevano integrare le pratiche dell'accademia, del collegio dei Gesuiti e delle università mostrano il potenziale di un'accademia per promuovere riforme innovative di istruzione e cultura, prefigurando le riforme educative dell'Illuminismo.

2. Capitolo 2

2.1 Il mondo dei Grafi

In questo capitolo vengono introdotti i grafi e una panoramica sulle loro caratteristiche principali e loro tipi. Successivamente vengono introdotti Graph DBMS, illustrando le caratteristiche e i concetti di base di questo nuovo modo di strutturare ed organizzare le informazioni, e poi presentati i DBMS NoSQL, fornendo al lettore una panoramica sulle loro caratteristiche principali, e le principali tipologie, alla fine arriviamo a Neo4j e suoi aspetti.

2.2 La Teoria dei Grafi

La teoria dei grafi ha una data di nascita precisa: il 1736. In quella data, il matematico svizzero Leonhard Euler risolse il problema noto come i sette ponti di Königsberg.

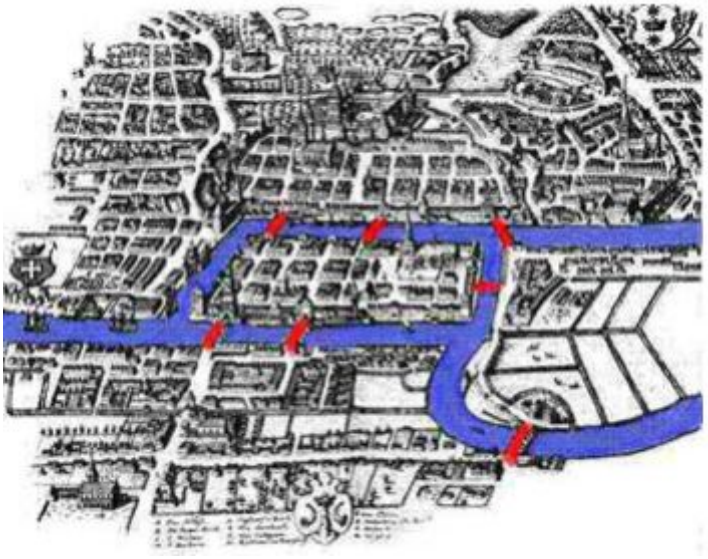
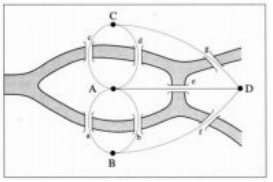


figura 2.2-1 Königsberg

Ci si chiedeva se fosse possibile fare una passeggiata in città, che partisse e arrivasse allo stesso punto, in modo da attraversare tutti i ponti esattamente una volta.



2.3 Definizione di Grafo

I grafi sono oggetti discreti che permettono di schematizzare una grande varietà di situazioni e di processi e spesso di consentirne delle analisi in termini quantitativi e algoritmici. Molti problemi di calcolo sono riconducibili a problemi su grafi, e possono essere risolti con tecniche sperimentate. La

struttura del grafo e costituita da vertici o nodi ed i collegamenti tra i nodi, i collegamenti si dividono in orientati e non orientati. Si dice grafo una coppia di insiemi G= (N, A) , con N insieme finito e non vuoto di nodi, ed A insieme finito degli Archi, tali che gli elementi di A sono coppie di elementi di N. I nodi i e j sono detti estremi dell'arco (i,j).
Esempio: Le informazioni di Twitter possono essere rappresentate come un grafo, l'immagine sottostante rappresenta una piccola rete di followers.

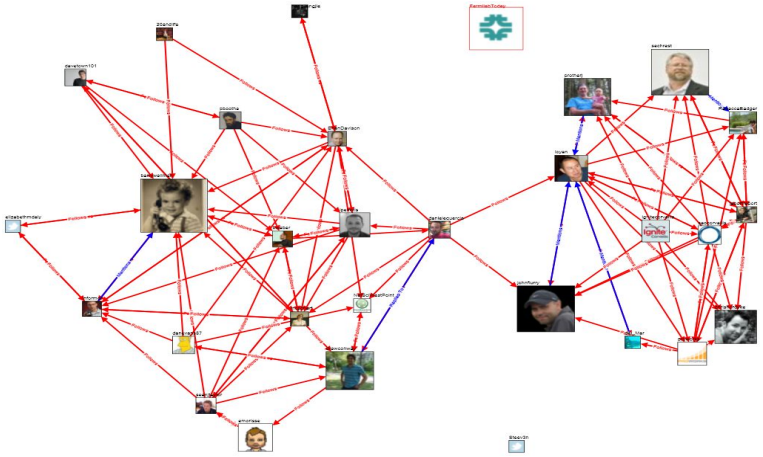


figura 2.3-1 rete

2.4 Grafi orientati / non orientati

I grafi sono orientati quando gli archi sono dotati di una direzione, in questi grafi si usano frecce per collegare fra loro i nodi . In particolare un arco è costituito da una testa che congiunge un nodo in entrata e da una coda che lascia un nodo in uscita.
Nei grafi non orientati invece gli archi non hanno un verso, in questi grafi i collegamenti si chiamano spigoli (edges). Di solito si rappresentano i nodi con circoletti e gli archi con linee per evidenziare le relazioni fra i nodi.

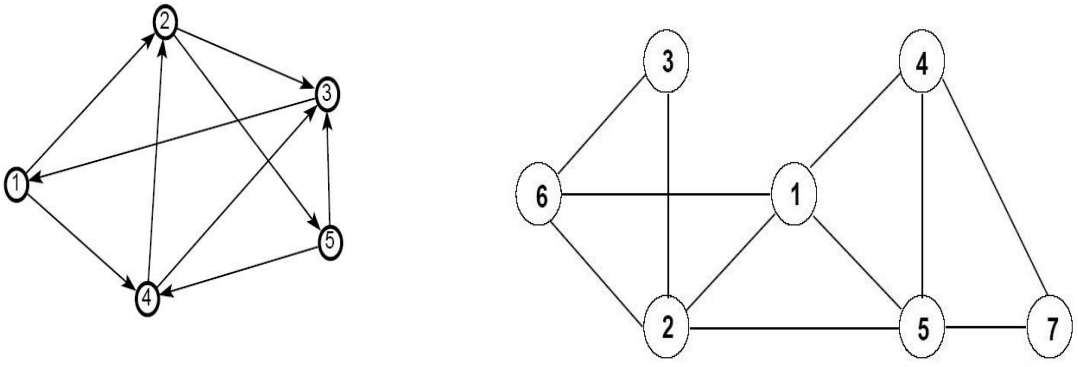


figura 2.4-1 Grafo orientato, non orientato

2.5 Modelli di dati a grafo

Quando si parla di modelli dei dati basati su grafo è inevitabile fare riferimento alla teoria dei grafi. In origine, i modelli di riferimento per l'implementazione dei database a grafo erano due: il grafo di proprietà e il Resource Description Framework graph (RDF). Il primo fa riferimento principalmente al progetto Tinkerpop, mentre il secondo è il modello di riferimento del Web semantico. I database a grafo che utilizzano il modello RDF sono anche noti come Triple Store, Quad Store, o RDF Store.

I due modelli non sono del tutto coincidenti, anche se solitamente il passaggio da uno all'altro è molto intuitivo. Per entrambi esistono dei linguaggi di interrogazione specifici, ma solo per RDF esiste uno standard riconosciuto in SPARQL. Successivamente ne sono stati sviluppati altri, di seguito i modelli ad oggi esistenti,

- Modello dei dati di grafo basico: grafo diretto con nodi e archi etichettati da qualche vocabolario (Es. Gram)
- Modello dei dati a ipernodo: Si basa sulla generalizzazione di grafo: con ipernordi e iperarchi, permette la creazione di oggetti complessi,dipendenze funzionali e eredità strutturale multipla
- Modello dei dati a ipernodo con grafi annidati: modello in cui ad un ipernodo può essere esso stesso un grafo
- Modello dei dati RDF: modello raccomandato dal W3C per rappresentare metadati
- Modello dei dati del grafo di Proprietà: modello di multigrafo diretto, etichettato, con attributi (proprietà) e con archi multipli tra i nodi (Neo4j, Sparksee/DEX, InfiniteGraph)

Un modo per caratterizzare i database a grafo è osservare il modello di dati che implementano. Tre sono i principali modelli: 1. grafi con proprietà 2. grafi RDF 3. ipergrafo.

2.5.1 Grafo delle proprietà

La definizione del grafo delle proprietà è semplice e molto intuitiva.

Un grafo di proprietà è un grafo che presenta le seguenti caratteristiche:

- È costituito da nodi, archi, proprietà ed etichette (o forme di classificazione simili).
- I nodi contengono un numero arbitrario di proprietà, che in genere sono coppie chiave-valore.
- Gli archi connettono i nodi e strutturano il grafo. Ogni arco ha una direzione, un solo nome, e un nodo iniziale e un nodo finale.
- I nodi e gli archi possono essere classificati in una o più categorie (per mezzo di classi o etichette). La categorie raggruppano i nodi insieme e indicano i ruoli che giocano all'interno di set di dati. Anche gli archi possono essere classificati. Insieme, la direzione dell'arco e il tipo (categoria) aggiungono significato alla struttura della rete.
- Anche le relazioni possono avere proprietà. La possibilità di aggiungere loro proprietà è particolarmente utile per migliorare la semantica del grafo e per limitare i risultati a query-time.

Questo tipo di modello ha ottenuto un buon successo e la maggior parte dei famosi grafi DBMS sul mercato utilizzano questo modello di dati. Inoltre, con il progetto Apache TinkerPop, c'è stato un tentativo di stabilire uno standard per questa forma di organizzazione dei dati, che sta guadagnando sempre più e più adozione nel campo. Apache Tinker Pop si definisce come un framework di calcolo di grafi open source, vendor-agnostic, sia per i database di grafi On Line Transaction Processing

(OLTP) che per l'analisi di grafi On Line Analytical Processing (OLAP); ed è distribuito sotto la licenza commerciale Apache v2.

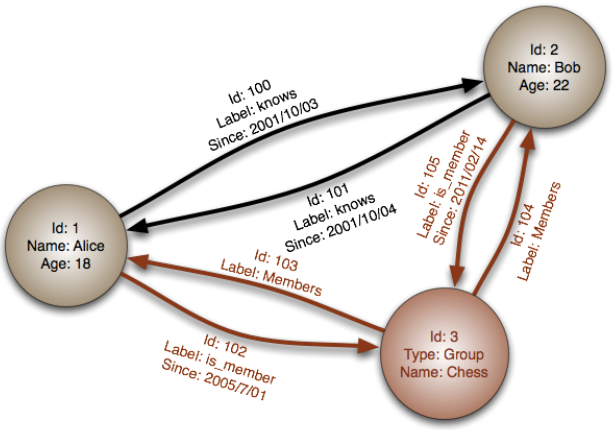


figura 2.5.1-1 Visualizzazione di un grafo di proprietà

Per i sistemi di analisi dei grafi, TinkerPop consente di esprimere algoritmi di analisi dei grafi, mentre per i database di grafi consente di eseguire modifiche ai dati del grafico e di esprimere query di attraversamento.

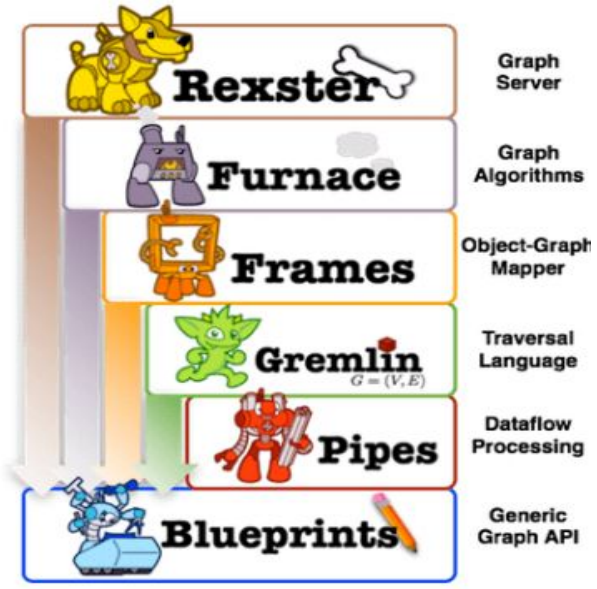


figura 2.5.1-2 Architettura di sistema TinkerPop

Gremlin è il linguaggio di query di navigazione fornito con TinkerPop ed è implementato in Groovy. Permette di esprimere un attraversamento in modo imperativo e dichiarativo. Un attraversamento imperativo dice ai pionieri come procedere ad ogni passo; mentre un attraversamento dichiarativo consente a ciascun attraversamento di selezionare un modello scelto da un insieme di modelli. In entrambi i casi, la richiesta dell'utente viene in seguito riscritta sulla base di un insieme di strategie di attraversamenti che fanno del loro meglio per determinare il piano di esecuzione ottimale, basato su una comprensione dei costi di accesso ai dati dei grafi e delle capacità dei sistemi di dati sottostanti.
Tuttavia, non tutti i database grafici sono basati su Gremlin come loro linguaggio di interrogazione; piuttosto, la maggior parte di essi ha sviluppato il proprio linguaggio di query dichiarativo. Le ragioni per questo è che il linguaggio dichiarativo in genere offre più spazio per consentire al server di

eseguire la pianificazione dell'analisi delle query e loro ottimizzazione; ed è più facile creare linguaggi progettati per invocazioni remote o semplificazioni di scrittura di query.

2.5.2 Grafo RDF

La definizione formale di RDF, fornita da W3C, è "Resource Description Framework (RDF) è un modello standard per lo scambio di dati sul Web. RDF ha caratteristiche che facilitano la fusione dei dati anche se gli schemi sottostanti differiscono, e supporta specificamente l'evoluzione degli schemi nel tempo senza richiedere agli utenti di cambiare tutti i dati." In altre parole, RDF estende la struttura di collegamento del Web per utilizzare gli URI per nominare le relazioni tra le cose e le due estremità del collegamento. La struttura principale di questo modello è un insieme di triple, ognuna composta da un soggetto, un predicato e un oggetto. Un insieme di tali triple è chiamato un grafo RDF; o un triple store.
Ci possono essere tre tipi di nodi in un grafo RDF: nodi URI, nodi letterali e nodi vuoti . URI sta per Universal Resource Identifier ed è una stringa di caratteri utilizzata per identificare univoca una risorsa. Il tipo più comune di URI è URL (Uniform Resource Locator), che viene utilizzato identificare le risorse Web. I nodi letterali sono usati per contenere valori come stringhe, numeri, e date. I nodi vuoti invece rappresentano risorse anonime, cioè quelli per i quali un URI o il valore letterale non è dato. Secondo lo standard, un nodo vuoto può essere usato solo come soggetto o oggetto di una tripla RDF.
Il soggetto è in genere identificato da un URI, mentre l'oggetto può essere un URI o un nodo letterale.
I predicati sono inoltre tipicamente identificati dagli URI e possono essere interpretati come una relazione tra i due nodi o come definizione di un valore di attributo (nodo oggetto) per un nodo soggetto. Il fatto che sia i verbi sia le risorse possano essere identificati dagli URI consente a chiunque di definire un nuovo concetto (sia verbo che dati), semplicemente definendo un URI per esso da qualche parte sul Web. Questo nuovo concetto acquisirà significato a livello mondiale e (possibilmente) visibilità e consentirà chiunque lo usi per dichiarare il suo grafo e magari collegarlo ad altri. Come conseguenza della sua architettura, quando viene creato un grafo RDF, da un punto di vista di basso livello si può vedere come archi che collegano entità con i loro attributi e altre entità. Tuttavia, da a punto di vista di alto livello, il grafo RDF realizza un grafo con etichette orientate fatto di spigoli tra entità. Inoltre, utilizzando questo semplice modello, RDF consente a dati strutturati e semistrutturati di essere mescolati, esposti e condivisi tra diverse applicazioni.

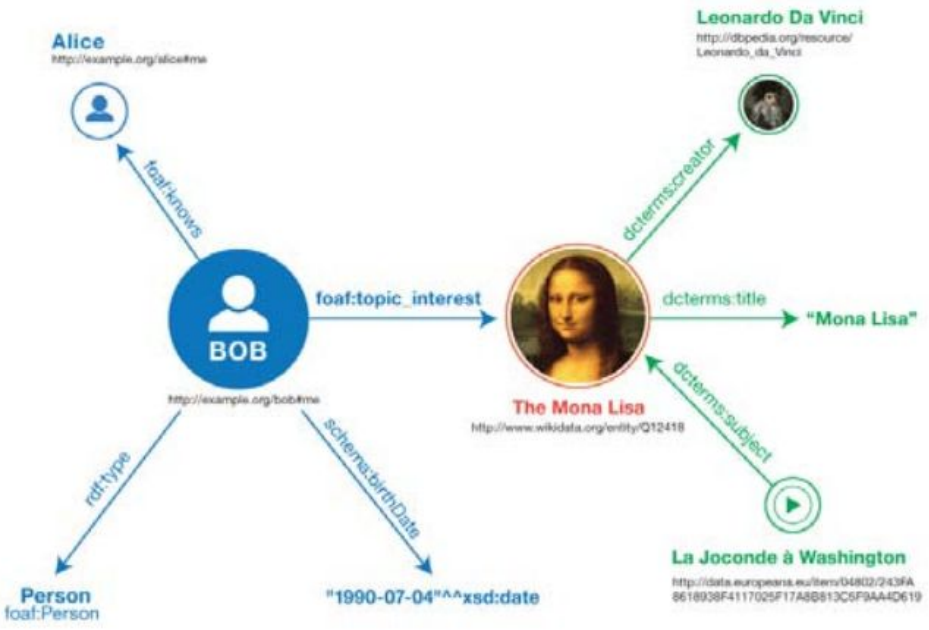


figura 2.5.2-1 Visualizzazione di un grafo RDF

2.5.3 Ipergrafo

Un altro modo per descrivere i dati del grafo è con un ipergrafo. Gli ipergrafi possono essere utili quando il set di dati include un gran numero di relazioni molti-a-molti. Tuttavia, con tali ipernodi, è semplice perdere la possibilità di specificare alcuni dettagli del arco a grana fine per le relazioni rappresentate dagli archi. Vediamo un esempio: nell'aerogramma (diretto) mostrato in figura vediamo che Alice e Bob sono i proprietari di tre veicoli; possiamo esprimere questo fatto solo usando un singolo ipernodo. In un grafo di proprietà, invece, dovremmo utilizzare sei relazioni per esprimere lo stesso fatto. Utilizzando sei relazioni anziché una, tuttavia, ci sono due vantaggi:

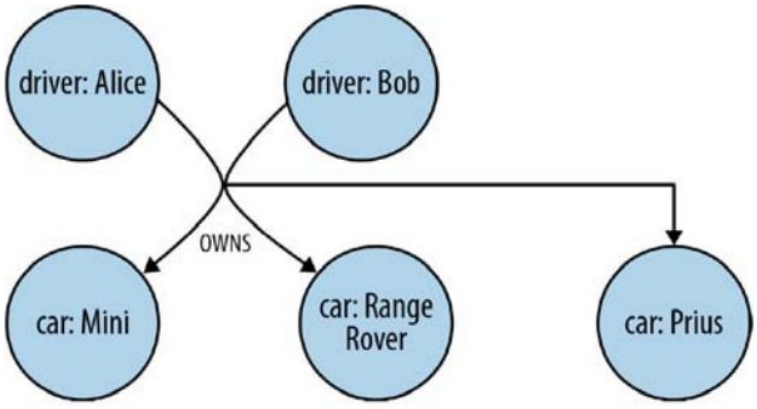


figura 2.5.3-1 Esempio di ipergrafo diretto

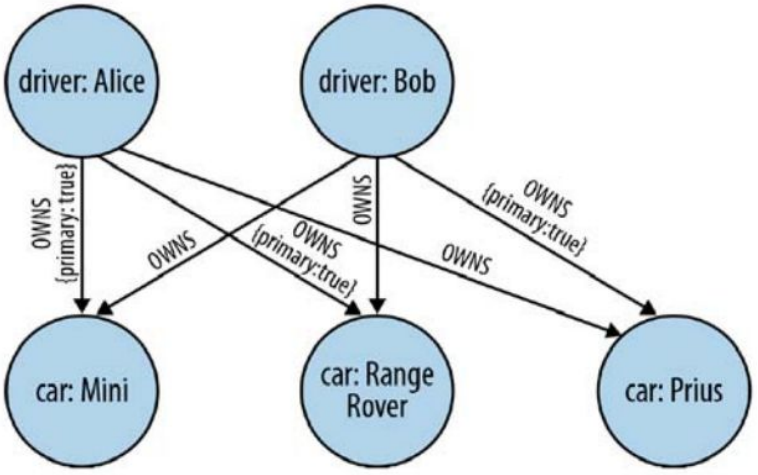


figura 2.5.3-2 Ipergrafo tradotto in un grafo di proprietà

- In primo luogo, stiamo utilizzando una tecnica di modellazione dei dati più familiare ed esplicita (che si traduce in meno confusione per un team di sviluppo).
- In secondo luogo, possiamo anche mettere a punto il modello con proprietà sulle relazioni, come, per esempio, una proprietà "driver primario" utile a fini assicurativi. La stessa cosa non può essere fatto se si usa un singolo ipernodo.

A causa della multidimensionalità dell'ipernodo, i modelli di ipergrafo sono più generali di grafi di proprietà. Tuttavia, i due sono isomorfi, quindi puoi sempre rappresentare un ipergrafo come a grafo delle proprietà (anche se con più relazioni e nodi). Tuttavia, la conversione opposta non è così immediato e dipende dalle informazioni memorizzate.

Mentre i grafi di proprietà sono ampiamente considerati per avere il miglior equilibrio di pragmatismo e l'efficienza della modellazione, ipergrafi mostrano la loro particolare forza nel catturare il meta-intento. Ad esempio, se è necessario qualificare una relazione con un'altra, quindi gli ipergrafi in genere richiedono meno primitive rispetto ai grafi di proprietà.

Il modello di dati dell'ipergrafo non ha generato la stessa adozione degli altri due modelli di dati mostrato prima, e pochissimi DBMS grafi gestiscono i dati secondo questo modello; un esempio per loro è ipergrafo DB.

2.6 I database orientati ai grafi

Non è necessario comprendere l'arcana stregoneria matematica della teoria dei grafi per comprendere la tecnologia dei database grafici. Al contrario, sono più intuitivi da comprendere rispetto ai database relazionali (RDBMS). Un grafo è composto da due elementi: un nodo e una relazione.

Un database grafico chiamato anche database basato sui grafi, è un tipo di database NoSQL che utilizza la teoria dei grafi per archiviare, mappare e interrogare le relazioni. Un database grafico è essenzialmente una raccolta di nodi e spigoli. Ogni nodo rappresenta un'entità (come una persona o un'azienda) e ogni spigolo rappresenta una connessione o una relazione tra due nodi. Ogni nodo in un database grafico è definito da un identificativo univoco, un insieme di archi in uscita e / o archi in entrata e un insieme di proprietà espresse come coppie chiave / valore. Ogni spigolo è definito da un identificatore univoco, un nodo di partenza e / o di fine-fine e un insieme di proprietà. Il mantra degli appassionati del database grafico è "Se puoi disegnarlo sulla lavagna, puoi tracciare il grafo". I

database a grafo sono adatti per l'analisi delle interconnessioni, motivo per cui c'è stato un grande interesse nell'utilizzo di database a grafo per estrarre i dati dai social media. I database di grafi sono anche utili per lavorare con i dati nelle discipline aziendali che implicano relazioni complesse e schemi dinamici, come la gestione della supply chain, l'identificazione della fonte di un problema di telefonia IP e la creazione di raccomandazioni "clienti che hanno acquistato questi prodotti anche visitata un altra ...".

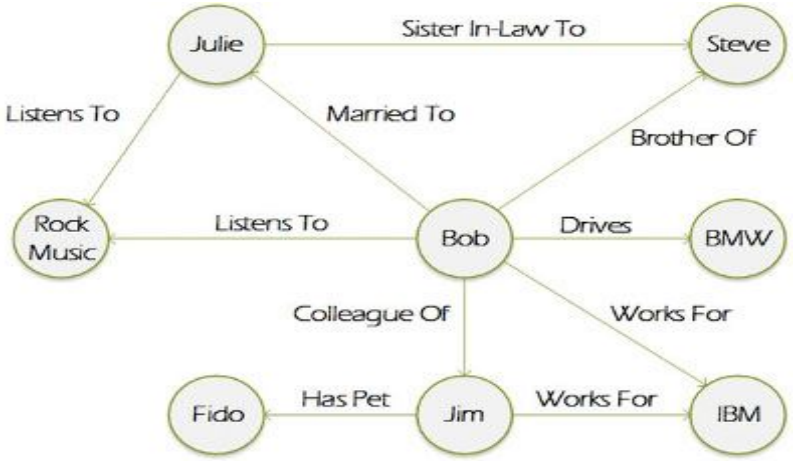


figura 2.6-1 Grafi

Semplicemente un database grafo è un database progettato per trattare le relazioni tra i dati come ugualmente importanti per i dati stessi. È inteso per contenere i dati senza costringerli a un modello predefinito. Invece, i dati vengono archiviati come se lo avessimo inizialmente estratto, mostrando come ogni singola entità si connette o è in relazione con gli altri.
Numerosi progetti e prodotti per la gestione, l'elaborazione e l'analisi dei grafi sono apparsi negli ultimi anni. Questa grande quantità di tecnologie rende difficile tener traccia di questi strumenti e come essi si differenziano, anche per coloro che da tempo lavorano in questo campo.
Tuttavia, il mondo dei Grafi, se visto dall'alto, è possibile dividerlo in due macro categorie:

1. Tecnologie impiegate principalmente per "transactional online graph persistence", tipicamente accedute per mezzo di applicazioni real time. Queste tecnologie vengono chiamate Graph DBMS. Esse sono l'equivalente dei sistemi OLTP del mondo relazionale. Questi sistemi sono caratterizzati da numerose ma semplici e veloci transazioni eseguite, spesso, in maniera concorrenziale.
2. Tecnologie impiegate principalmente per l'analisi Offline dei grafi. Solitamente eseguite come una serie di batch step. Queste tecnologie vengono chiamate Graph Compute Engine.

2.6.1 Graph DBMS

Un Graph Database Management System è un sistema di gestione online che sottopone un modello dati a grafo, a metodi di Creazione, Lettura, Aggiornamento e Cancellazione (Create, Read, Update e Delete : CRUD). I Graph DBMS sono generalmente costruiti per sistemi transazionali OLTP. Di conseguenza vengono progettati in modo da ottimizzare le prestazioni e l'integrità delle operazioni transazionali.
Vi sono due componenti da tenere in mente quando si si vuole analizzare una tecnologia di questo genere:

2.6.1.1 Storage sottostante

Sebbene sia scontato pensare che questi sistemi posseggano ogni loro componente proiettata verso il mondo dei grafi, in realtà solo qualche Graph DBMS utilizza dei native graph storage, ovvero, possiede una piattaforma di salvataggio delle informazioni sottostante nata ed ottimizzata per salvare i dati sotto forma di grafo. Diversi graph DBMS, in effetti, traducono e salvano le informazioni in modi differenti, ovvero all'interno di un database relazionale, di un database orientato agli oggetti, o qualche altro tipo di data store.

2.6.1.2 Processing Engine

Le definizioni fornite dal mondo dei grafi richiedono che, per essere considerati tali, i Graph DBMS debbano utilizzare l'index-free adjacency (questo significa che ogni elemento contiene un puntatore diretto ai suoi elementi adiacenti rendendo così le ricerche via indice non necessarie). Tuttavia, come detto in precedenza, è possibile espandere la definizione di Graph Database Management System a tutti quei DBMS che permettano di eseguire delle operazioni CRUD su un modello dati a grafo. Ciò significa che, possiamo distinguere i graph DBMS in due categorie, la prima tutti quelli che sfruttano l'index-free adjacency , processing engine nativo (più performante); la seconda quelli che non la usano, processing engine non nativo.

2.6.2 Graph compute Engine

Un Graph Compute Engine è una tecnologia che permette di eseguire Algoritmi Computazionali su Grafi Globali sopra grandi dataset. I graph compute engine sono progettati per eseguire operazioni come identificare clusters all'interno dei dati, oppure rispondere a delle domande come "Qual è la media della relazioni che posseggono gli utenti di una social network?" .
A causa dell'incapacità delle interrogazioni, i graph compute engine sono ottimizzati per scandire e processare enormi quantità di blocchi di informazione.

2.7 Storia dei database NoSQL

L'acronimo NoSQL può essere interpretato come una combinazione di due parole: No e SQL. Gli autori volevano sottolineare che il database non è RDBMS o No relazionale. Più tardi, altri cercarono di salvare il termine originale con una nuova espansione di acronimo in Not Only SQL. Qualunque sia il significato letterale, NoSQL è, oggi, un termine generico comprendente banche dati e archivi di dati che non seguono i principi RDBMS e di solito si riferiscono a tecnologie che consentono di elaborare insiemi di dati di grandi dimensioni e la loro distribuzione.
L'approccio del database non relazionale non è nuovo; il primo concetto è emerso all'inizio quando è stato inventato il primo set di macchine informatiche. I database NoSQL hanno trovato il loro posto nel contesto di domini specializzati e specifici come le directory gerarchiche per l'archiviazione delle credenziali di autorizzazione. Il concetto non relazionale è diventato un punto di riferimento essenziale nel tempo delle applicazioni Internet scalabili.
I NoSQL DBMS sono inoltre contraddistinti dal fatto che non utilizzano un sistema transazionale ACID, il quale garantisce che ogni sua transazione soddisfi le seguenti proprietà:

1. Atomicity - una transazione è un'unità di elaborazione atomica, indivisibile. Ciò significa che dovrà essere eseguita totalmente oppure per niente, senza scendere in parti più piccole.
2. Consistency - quando viene lanciata, una transazione trova il database in uno stato consistente, al suo completamento il Database dovrà ancora godere di questa proprietà.
3. Isolation - una transazione dovrà essere isolata completamente dalle altre. In caso di fallimento non dovrà interferire con le altre transazioni in esecuzione.
4. Durability - gli effetti di una transazione che ha terminato correttamente la sua esecuzione devono essere persistenti nel tempo.

Le principali categorie di DBMS NoSQL sono:

- Key-Value store.
- Document-oriented.
- Column Family store.
- Graph DBMS.

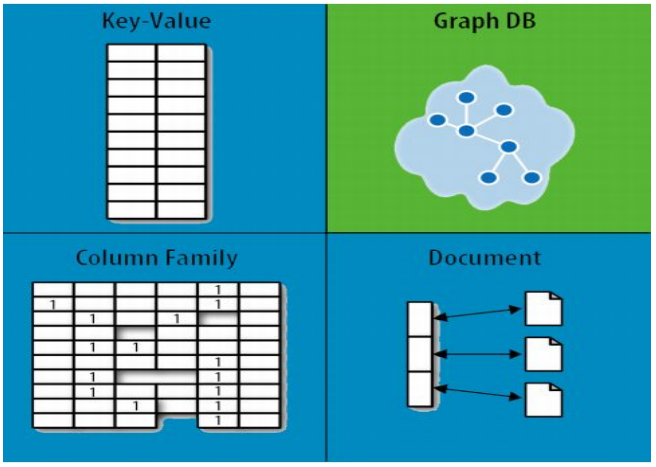


figura 2.7-1 NoSQL Databases

Google ha creato un'infrastruttura estremamente scalabile per il suo motore di ricerca e altre applicazioni. anche ha pubblicato la sua esperienza e conoscenza in molti articoli; i più importanti dei quali sono:

- Il file system di Google di Sanjay Ghemawat, Howard Gobioff e Shun-Tak Leung. 2003.[1]
- MapReduce: Elaborazione dati semplificata su cluster di grandi dimensioni di Jeffrey Dean e Sanjay Ghemawat. Del 2004.[2]
- Bigtable: un sistema di archiviazione distribuito per dati strutturati di Fay Chang et al. 2006.[3]

The release of Google's papers to the public stirred significant interest in open-source community. Doug Cutting and Mike Cafarella from University of Washington started their better open-source search engine. The project began in 2002 and was called Nutch. The released papers continuously affected Nutch development. Nutch generalized some papers' ideas into a framework that automated all data processing steps. In 2006 Cutting started working with Yahoo, the storage and processing parts were forked and became a foundation for Apache Hadoop. Hadoop is open-source framework written in Java and it is focused on distributed storage and distributed processing of very large data

[1] Il documento del file system di Google è disponibile all'indirizzo http://research.google.com/archive/gfs.html.

[2] La carta MapReduce è disponibile all'indirizzo http://research.google.com/archive/mapreduce.html.

[3] La carta più grande disponibile a http://research.google.com/archive/bigtable.html.

sets. Hadoop origins in Google's papers and contains the following main modules Hadoop Distributed File System and Hadoop MapReduce.
The emergence of Hadoop laid the groundwork for a rapid growth of NoSQL. In the next 5 years, NoSQL and similar concepts for managing big data have become widespread. Many well-known companies, including Facebook, Netflix, Yahoo, EBay, Hulu, IBM, and many more, found use-cases for it and contributed to their development.

2.7.1 Tipi di database

NoSQL riconosce 4 gruppi di base di database NoSQL. Ognuno di essi unisce un insieme di database con un comportamento simile, anche se alcuni di essi sono al limite e possono cadere in due gruppi.

2.7.1.1 Key-Value store

Gli archivi di valori-chiave sono i tipi NoSQL più semplici. La loro idea principale è rendere i dati rapidamente disponibili accedendo con la loro chiave univoca. Il problema comune è che non è possibile interrogare i dati né con chiave esterna né con altri dati memorizzati nel valore. Il valore dei dati memorizzati non ha schema e la chiave univoca offre ricerche molto veloci ed efficienti. Gli archivi a valore-chiave sono utilizzabili per la cache del contenuto, la memorizzazione dei registri e così via. Redis, RocksDB (di Facebook), Riak, Amazoon SimpleDB, Oracle BDB sono tra i rappresentanti.

2.7.1.2 Document-oriented

I database dei documenti sono molto simili memorizza valori chiave. Il modello è composto da documenti con versione che formano una raccolta di altre raccolte di valori-chiave. Il valore memorizzato non ha uno schema rigido. Il valore (documento) viene solitamente memorizzato nel formato JSON. I database di documenti consentono query avanzate per indici secondari. Il tipico caso d'uso è l'archiviazione dei dati per le applicazioni Web. I rappresentanti sono CouchDB e MongoDB.

2.7.1.3 Column Family store

Nel database NoSQL orientato alle colonne, i dati vengono archiviati in celle raggruppate in colonne di dati anziché come righe di dati. Le colonne sono raggruppate logicamente in famiglie di colonne. Le famiglie di colonne possono contenere un numero virtualmente illimitato di colonne che possono essere create durante il runtime o la definizione dello schema. La lettura e la scrittura vengono eseguite utilizzando le colonne anziché le righe. In confronto, la maggior parte dei DB relazionali memorizzano i dati in righe, il vantaggio di archiviare i dati in colonne, è la ricerca / accesso veloce e l'aggregazione dei dati. I database relazionali memorizzano una singola riga come una voce continua del disco. Diverse righe sono archiviate in punti diversi sul disco mentre i database Columnar memorizzano tutte le celle corrispondenti a una colonna come una voce continua del disco, rendendo così la ricerca / accesso più veloce. Cassandra (di Facebook) e Apache HBase (database Hadoop) sono alcuni dei rappresentanti.

2.7.1.4 Graph DBMS

In un database NoSQL Graph Base, non troverai il formato rigido di SQL o la rappresentazione di tabelle e colonne, ma viene utilizzata una rappresentazione grafica flessibile che è perfetta per risolvere i problemi di scalabilità. Le strutture del grafico vengono utilizzate con spigoli, nodi e proprietà che forniscono un adiacenza senza indice. I dati possono essere facilmente trasformati da un modello all'altro utilizzando un database NoSQL Graph Base.
I database grafi utilizzano un modello di grafo flessibile che può essere scalato su più macchine. Tipici di questo tipo di applicazioni sono i social network e le raccomandazioni, la gestione della rete e del cloud, la gestione dei dati master, geospaziale, bioinformatica e controllo della sicurezza e degli accessi. In generale, i database relazionali non sono adatti per l'archiviazione dei dati di relazione, soprattutto perché le query di relazione in RDBS sono complesse, lente e richiedono molti join; sono quindi inutilizzabili per attraversare in profondità. I database Graph differiscono maggiormente dagli altri gruppi NoSQL poiché sono basati su prerequisiti diversi. I rappresentanti sono Neo4j, InfoGrid, Infinite Graph, Apache Giraph, DEX, OrientDB.

2.8 Il database grafo Neo4j

Neo4j è un Graph DBMS open source transazionale, prodotto dalla software house Neo Technology. Possiede processing engine e underlying storage nativi ed è sviluppato completamente in Java. É robusto, scalabile e ad alte prestazioni. È dotato di:

- Transazioni ACID
- High Availability
- può memorizzare miliardi di nodi e relazioni
- alta velocità di interrogazione tramite attraversamenti
- linguaggio di interrogazione dichiarativo e grafo.

È un DBMS schema-less, ciò sta a significare che i suoi dati non devono attenersi alla alcuna struttura di riferimento prefissata, inoltre non possiede una politica di accesso controllata .La index-free adjacency è alla base delle sue alte prestazioni di attraversamento, di interrogazione e di scrittura, ed è uno degli aspetti chiave della sua architettura. L'index-free adjacency è una lista (o tabella), ove ogni suo elemento è composto da un nodo del grafo e dai puntatori ai nodi connessi ad esso.
Neo4j salva i dati dentro di una serie di store file, contenuti all'interno di un'unica cartella. Ognuno di questi file contiene al suo interno le informazioni relative ad una singola parte del grafo (e.g. nodi, relazioni, proprietà). Questa separazione della struttura del grafo facilita il suo attraversamento.
Neo4j è multi-relazionale e utilizza il modello del grafo delle proprietà: funziona con nodi, relazioni e proprietà. Nodi e relazioni sono analogici ai vertici del grafo e archi.Il database Neo4j è scritto in linguaggio Java - è possibile incorporare Neo4j nella propria applicazione Java. Se non si utilizza l'incorporamento Neo4j, Neo4j fornisce l'API REST. Neo4j arriva con un proprio linguaggio di query dichiarativo chiamato Cypher. Sia l'API Java che l'API REST consentono interrogazione tramite linguaggio Cypher. In Java, è possibile utilizzare l'API di attraversamento, che è meno leggibile e comodo (richiede una comprensione più profonda degli algoritmi del grafo) sebbene sia di più potente e consente un'elaborazione dei dati più rapida. Neo4j può essere facilmente installato su Windows, Linux e Mac OS; l'installazione è molto semplice e non richiede né ulteriori configurazioni né dipendenze.Neo4j fornisce un'interfaccia browser per l'interrogazione manuale del database con

Cypher. l'interfaccia presenta i risultati restituiti dalla visualizzazione grafica interattiva (mostrata nella figura). È possibile definire un colore e una dimensione per specifiche etichette di nodi e relazioni. Il principale lo svantaggio della visualizzazione interattiva è la questione della prestazione - la visualizzazione diventa molto lenta o non risponde quando visualizza più di 300 nodi. I risultati possono anche essere presentati in un tabulato. La query Cypher è scritta nella casella di input che supporta l'evidenziazione della sintassi e la cronologia delle query di navigazione.

figura 2.8-1 Neo4j

I nodi Neo4j rappresentano entità applicative; i nodi possono essere etichettati (digitati) e possono essere archiviati con le proprietà. Le etichette consentono una migliore corrispondenza di nodi particolari (analogia con le tabelle RDBM). Le proprietà sono memorizzate come coppie chiave-valore. Le chiavi sono testuali; i valori possono essere memorizzati come diversi tipi di dati. Neo4j supporta i seguenti tipi di dati: boolean, byte, short, int, long, float, double, char, String. Ogni tipo scalare disponibile può essere usato come una matrice. Neo4j non supporta valori NULL; possono essere modellati dall'assenza della chiave. Le relazioni collegano due nodi insieme. Ogni relazione deve essere diretta, avere a tipo, e un inizio e un nodo finale. Le relazioni possono iniziare e finire nello stesso nodo (una relazione riflessiva). Il tipo di relazione è l'etichetta definita dall'utente. Neo4j usa lo stile di codifica dove le etichette delle relazioni sono scritte in maiuscolo. Le relazioni possono memorizzare proprietà nello stesso modi come i nodi.

2.8.1 Architettura

Tutti i dati e le informazioni del grafo che il server storicizza e gestisce vengono salvate all'interno di una serie di file che prendono il nome di Store File, i quali vengono memorizzati all'interno di un'unica cartella, detta Cartella di Database. Ogni database o grafo possiede una propria Database Directory, e un server può gestire una sola di queste cartelle per volta. Prima di avviare il server è possibile definire da quale cartella caricare il grafo, modificando un dei file di configurazione presenti nell'albero cartelle del server (conf/Neo4j-server.conf).
Gli Store File di un grafo sono innumerevoli, ma le informazioni che ne descrivono la struttura e i dati che esso contiene sono essenzialmente tre:

- ➔ neostore.relationshipstore.db per le relazioni;
- ➔ neostore.propertystore.db per le proprietà;
- ➔ neostore.nodestore.db per i nodi.

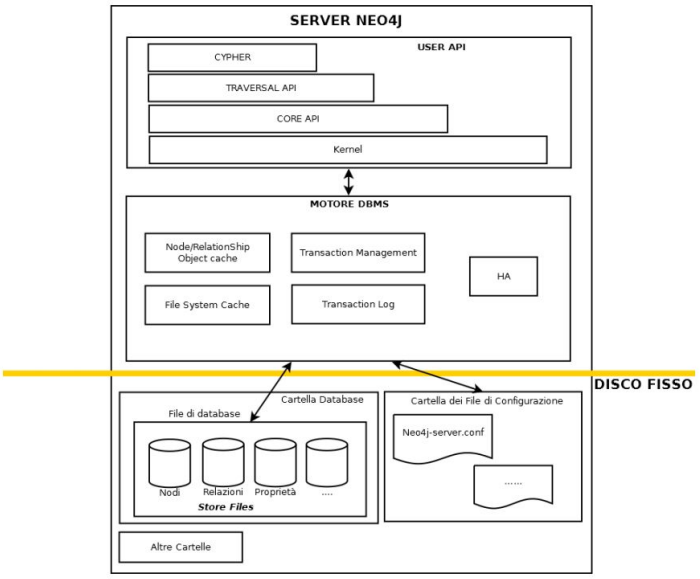


figura 2.8.1-1 Neo4j Architettura

Le caratteristiche principali di Neo4j sono:

- Native Graph storage, la memorizzazione dei dati avviene con una piattaforma ottimizzata e progettata per la memorizzazione e gestione dei grafi. Ciò assicura la consistenza dei dati e notevoli performance.
- Native Graph Processing, ogni nodo del grafo possiede un puntatore diretto ai nodi adiacenti (index-free adjacency) e ciò permette di effettuare milioni di salti da un nodo all'altro al secondo, in real-time.
- Integrità dei dati Supporta transazioni ACID.
- Modellazione "Amica della lavagna" Il modello dei dati è molto simile alla realtà esaminata.
- Un linguaggio di interrogazione potente ed espressivo La quantità di codice richiesto è da 10 a 100 volte minore del SQL.
- Scalabile e High Availability Scalabilità verticale e orizzontale ottimizzata per i grafi.
- Gestione ETL incorporata Importazione diretta dagli altri database.
- Integrazioni Driver e API disponibili per i linguaggi più popolari.

2.9 Cypher

Cypher è un linguaggio di query a grafo aperto indipendente dal fornitore utilizzato nell'ecosistema grafo. La sintassi in stile ASCII dell'arte di Cypher fornisce un modo familiare e leggibile per associare i modelli di nodi e relazioni all'interno dei set di dati del grafo.
Come SQL, Cypher è un linguaggio di query dichiarativo che consente agli utenti di indicare quali azioni devono essere eseguite (come corrispondenza, inserimento, aggiornamento o eliminazione) sui loro dati del grafo senza richiedere loro di descrivere (o programmare) esattamente come farlo . È possibile eseguire Cypher query tramite CLI (Neo4j Shell), API Java o API REST. Cypher prende in prestito la sua struttura da SQL: le query vengono create utilizzando varie clausole.

2.9.1 Pattern Matching

Cypher usa ASCII-art per descrivere i modelli per la corrispondenza. I nodi sono circondati da parentesi, le relazioni sono circondate da parentesi. Una stringa interna è un nome di variabile in cui a il nodo / arco sarà disponibile seguito da etichette e proprietà opzionali per una corrispondenza del modello. I nodi sono collegati insieme da --, --> o <-- caratteri. La freccia indica la direzione dell'arco, i trattini senza freccia lo ignorano. Le etichette e le proprietà delle relazioni possono essere abbinato anche a scelta; il loro schema è messo nel mezzo di due trattini.
Un modello può anche abbinare nodi e relazioni con etichette specifiche. Le etichette sono scritte dopo che il nome della variabile è stato separato con due punti. Le etichette sono opzionali, quindi sono le variabili nomi. È possibile aggiungere più etichette aggiungendo un altro punto e un nome etichetta; loro sono valutato dalla congiunzione logica (AND).
I nodi e le relazioni possono essere associati alle loro proprietà. Proprietà sono scritte in parentesi come coppie chiave-valore. Più proprietà sono separate da una virgola. Cypher offre l'opportunità di passare separatamente i valori delle proprietà come parametri, principalmente per evitare la sicurezza di SQL Injection problemi. I parametri sono circondati da un'altra coppia di parentesi e inviati a Neo4j separatamente.

```
(cat:User { name: 'Cat' })-[rel:KNOWS]->(ned:User { name: 'Ned' })
(author:Author)-[:TRANSLATED { from: 'cs', to: 'en' }]->(:Book)
(city:City:Capital)
(user:User { name: {user_name} } // parameter "user_name" is sent separately
```

2.9.2 Le Clausole MATCH e RETURN

La corrispondenza di un modello può essere eseguita dalla clausola MATCH. Un set di risultati corrispondenti può essere elaborato dalla clausola RETURN che può eseguire una proiezione sul set di risultati. Nodo e proprietà dell'arco sono accessibili con la notazione dot. La clausola OPTIONAL MATCH è simile a outer join in SQL se il modello opzionale (nodo o relazione) non corrisponde, viene prodotto il valore NULL.

```
MATCH (node) RETURN node.property;
MATCH (nodeA)-[edgeB]->() RETURN nodeA.propertyA, nodeB.propertyB;
MATCH (nodeA)--(nodeC) RETURN nodeA.propertyA, nodeC.propertyC;

MATCH (a:Show { title: 'Kings' })
OPTIONAL MATCH (a)-->(x:Actor)
RETURN x, x.name;
```

2.9.3 La clausola WHERE

Nodi e relazioni possono essere filtrati in base alle condizioni; possono essere inseriti nella clausola WHERE, o direttamente in un modello di corrispondenza. Le proprietà possono essere confrontate per l'uguaglianza con altre variabili e costanti; possono anche essere testati contro le espressioni regolari. La seconda query nel seguente codice sorgente restituisce tutti i titoli di libri che sono stati graditi da un utente, ma quell'utente non deve gradisce come il libro dei Kings.

```
MATCH (book:Book) WHERE book.pagesCount > 200 RETURN book.title;

MATCH (user:User)-[:LIKES]->(book:Book)
WHERE NOT (user)-[:LIKES]->(:Book { title: 'Kings' })
RETURN book.title;
```

2.9.4 Le Clausole ORDER BY, LIMIT e SKIP

Cypher consente l'ordinazione e l'impaginazione nelle clausole ORDER BY, LIMIT e SKIP. Tutti le clausole accettano l'espressione e si comportano in base ai risultati della valutazione.

```
MATCH (n) RETURN n ORDER BY n.name SKIP 10 LIMIT 5;
```

2.9.5 La Clausola WITH

La clausola WITH consente il concatenamento di parti di query; ciò significa piping i risultati da uno clausola come input per altre clausole. La clausola WITH è simile l'aggregazione in SQL, ma potrebbe essere usato ripetutamente La seguente query Cypher limita i percorsi corrispondenti al nodo m fino all'ultimo uno e quindi restituisce tutti i nodi adiacenti.

```
MATCH (n { name: 'Ned' })--(m)
WITH m ORDER BY m.name DESC LIMIT 1
MATCH (m)--(o) RETURN o.name;
```

2.9.6 La Clausola UNION

La clausola UNION consente di unire più risultati delle query. Rimuove le righe duplicate; La clausola UNION ALL viene utilizzata per lasciare duplicati nel set di risultati.

```
MATCH (n:Actor) RETURN n.name AS name
UNION ALL
MATCH (n:Movie) RETURN n.title AS name;
```

2.9.7 Le Clausola CREATE, MERGE, SET, DELETE

Le clausole CREATE, MERGE, SET e DELETE servono per conservare i grafici. La clausola CREATE crea nuovi nodi, relazioni e interi schemi. La clausola di MERGE corrisponde ai nodi, alle relazioni e ai modelli esistenti e crea le parti mancanti. Il seguente codice sorgente mostra due query: la prima crea una nuova relazione tra due nodi; se i nodi non corrispondono, non viene creata alcuna

relazione; la seconda query corrisponde a un nodo esistente e lo restituisce, se il nodo non esiste, viene creato.

```
MATCH (a:Person { name: 'Ygritte' }),(b:Person { name: 'Jon' })
CREATE (a)-[r:LOVES { since: '432' }]->(b);

MERGE (bran:Person { name: 'Brandon' })
RETURN bran;
```

La clausola SET consente di aggiornare particolari proprietà dei nodi e delle relazioni corrispondenti. L'impostazione della proprietà su NULL rimuove la proprietà. La clausola SET può essere attivata dalle clausole ON CREATE e ON MATCH che sono rilevanti nel contesto della clausola MERGE. La clausola ON CREATE viene attivata quando viene creato un nodo; La clausola ON MATCH viene attivata ogni volta che un nodo è abbinato.

```
MATCH (n:Person { firstname: 'Robert' })
SET n.nick = 'Rob'
RETURN n;

MERGE (arya:Person { name: 'Arya' })
ON CREATE SET arya.born = timestamp()
ON MATCH SET arya.lastSeen = timestamp()
RETURN arya;
```

La clausola DELETE rimuove i nodi e le relazioni passati dal grafo. Non è possibile rimuovere nodi che hanno relazioni; devono essere rimossi nella stessa query al più tardi. La seguente query rimuove i nodi con il nome di proprietà Sam e tutte le relazioni dei nodi.

```
MATCH (n { name: 'Sam' })-[r]-()
DELETE n, r;
```

2.9.8 Il Cypher Query

Qui vediamo qualche linea dalla nostra cypher query per IAD database. In questo modo vediamo come possiamo creare i nostri dati con cypher.

2.9.8.1 Vincoli di proprietà uniche

I vincoli di proprietà univoci assicurano che i valori delle proprietà siano univoci per tutti i nodi con un'etichetta specifica.

```
create constraint on (a:`Academy`) assert a.`id` is unique;
create constraint on (b:`Book`) assert b.`id` is unique;
create constraint on (m:`Member`) assert m.`id` is unique;
create constraint on (p:`Person`) assert p.`recordId` is unique;
```

2.9.8.2 indice di database

Un indice di database è una copia ridondante di alcuni dati nel database allo scopo di rendere più efficienti le ricerche di dati correlati. Ciò comporta un ulteriore spazio di archiviazione e scritture

più lente, quindi decidere cosa indicizzare e cosa non indicizzare è un compito importante e spesso non banale.

```
CREATE INDEX ON :Academy(name)
CREATE INDEX ON :Book(shortTitle)
CREATE INDEX ON :Person(name)
CREATE INDEX ON :Member(id)
```

2.9.8.2.1 Esempio di un query dalla nostra database per Academia :

```
create(a:`Academy`{`alternativename`:"Academia
Gelatorum",`cityEnglishName`:"Bologna",`cityId`:1,
`cityItalianName`:"Bologna",`cityLatinName`:"Bononiae",`dateText`:"1588-
ca.1799",`emblemDescription`:"A wood of trees covered with ice",
`endDate`:"ca.1799", `id`:1, `motto`:"Nec longum tempus", `name`:"Gelati
(Accademia dei)", `notes`:"List of members of the Academy is in Memorie
Imprese e Ritratti de' Signori Accademici Gelati di Bologna (Bologna,
1672) and Leggi dell'Accademia de' Sig[nori] Gelati di Bologna (Bologna,
1683). Biographical information are in Fantuzzi, Notizie degli scrittori
bolognesi (1784). --- Giorgio Stabile refers to a letter mentioning the
Accademia dei Gelati as early as 1558, see Stabile, `Betti Claudio,
detto Betto il Giovane`, in DBI. --- Explanation of the Academy`s
emblem, is in Zoppio, Consolatione di Melchiorre Zoppio, filosofo
morale, nella morte della moglie, Olimpia Luna Z. (1603). --- For the
Gelati`s interest in theatre, see Marina Calore, `La biblioteca
drammatica delgi Accademici Gelati di Bologna`, in Atti della Accademia
delle scienze dell`Istituto di Bologna , Classe di scienze morali,
Rendiconti, 1992-1993, pp. 61-82. --- For a history of the Accademia in
the late XVII c., see Bergamini, `Dai Gelati alla Renia (1670-1698).
Appunti per una storia delle accademie letterarie bolognesi`, in La
Colonia Renia: profilo documentario e critico dell'Arcadia bolognese,
ed. Mario Saccenti 2 vols (Modena: 1988).
Publications related to the Academy, but not in British Library
catalogue, include: Encomia iurisprudentiae solemnia Paulus Maccius
Acad. Gelato. In Archigymnasio Bononiae dicebat. Et Illustrissimo D.
Augustino Marsilio Senat. dicabat... (1633); Nerei vaticinium de raptu
Helenae Apellaea Guidonis Rheni arte depicto. Paulus Maccius in Academia
Gelatorum Defessus describebat, Emanuelique Vizanio adolescenti
Latinarum, Graecarumque litterarum studiosissimo, ... donabat (Bononiae
: typis Clementis Ferronij, 1633 (Bononiae, Kalen. Februarij [1 II]
1633); Cruentum divae Martinae virginis et martyris in Verba Christi
Iuramentum Paullus Maccius in Academia Gelatorum defessus scribebat, et
illistrissimis Bononiensis Reipublicae Senatoribus dicabat (Bononiae:
typis Clementis Ferronij, 1635); Lo sposalizio della terra col Cielo per
l`Immacolata Concezione della Verg. Orazione panegirica di Paolo Emilio
Fantucci senatore bolognese, recitata dal medesimo nel tempio di S.
```

```
Francesco di Bologna nell`occasione della solita accademia
dell`Immacolata Concezione (Bologna: presso Gio. Battista Ferroni,
1658); Accademia dei Gelati, Componimenti dispensati nella publica
adunanza de gli Academici Gelati per le nozze del sig.r Tolomeo Duglioli
il Dedito e della S.ra Maria Barberini (In Bologna: per Bartolomeo
Cochi, 1618?)",`recordId`:"021-000000001",`roles`:"Dedicatee|Editor",
`startDate`:"1588"})
```

2.9.8.2.2 Esempio di un query dalla nostra database per libro :

```
create (b:`Book`{`academyIds`:107,`artistId`:0,`authorIds`:0,
`contributorIds`:"18128|18095",`dedicationPlace`:"Venice 22 march
1561",`editorIds`:"18122",`formats`:"8°",`id`:642,`languages`:"Italian",
`libraryLocation`:"British Library", `longTitle`:"di diversi huomini
grandi et chiari et begli ingegni. raccolte per M. Dionigi Atanagi.
Libro primo. Riveduto, scelto et corretto dal medesimo, et con somma
diligenzastampato.",`marginalia`:0,`pagination`:"438",`printerId`:18245,
`printerOrnament`:"Z563",`publicationPlaceEnglishNameId`:35,`publication
PlaceItalianNameId`:35,`publicationPlaceLatinNameId`:35,`publicationYear
`:"1601",`publisherId`:0,`recordId`:"023-000005232",`shelfmarks`:"1084.d
.10",`shortTitle`:"De le lettere facete et piacevoli",
`subjects`:"Letters"})
```

2.9.8.2.3 Esempio di un query dalla nostra database per Autore:

```
create(p:`Person`{`birthDate`:"fl.1613",`dateText`:"fl.1613",`forename`:
"Tomaso",`gender`:"Male",`id`:19265, `name`:"Dralli, Tomaso, , fl.1613",
`nationality`:"of Varese",`personalTitles`:"
",`recordId`:"022-000006448", `surname`:"Dralli"})
```

2.9.8.2.4 Esempio di un query dalla nostra database per Member :

```
create(m:`Member`{`academyId`:516,`id`:31748,`personId`:20474})
```

2.9.8.2.5 Esempio di un query dalla nostra database per le relazioni tra autori e sue opere:

```
create (b:Book)-[:`WRITTEN_BY`]->(p:Person)
```

2.9.8.2.6 Esempio di un query dalla nostra database per le relazioni tra le accademie e sue pubblicazione:

```
create (b:Book)-[:`PUBLISHED_IN`]->(a:Academy)
```

2.9.8.2.7 Esempio di un query dalla nostra database per le relazioni tra autori e member:

```
create (p:Person)-[:`MEMBER_AT`]->(m:Member)
```

2.9.8.2.8 Esempio di un query dalla nostra database per le relazioni tra accademie e member:

```
create (m:Member)-[:`MEMBER_AT`]->(a:Academy)
```

2.9.8.2.9 Esempio di un query dalla nostra database per le relazioni tra autori e accademie:

```
create (p:Person)-[:`MEMBER`]->(a:Academy)
```

3. Capitolo 3

3.1 Relazioni delle Accademie Italiane

In questo capitolo viene introdotto l'obiettivo del progetto, quali sono le fasi del progetto, Successivamente vediamo gli strumenti che si sono usati per configurarlo, alla fine si dimostra con alcuni screenshots la funzionalità del progetto ed il suo outcome.
Per fare il progetto Le relazioni tra "Le Accademie Italiane tra il 1525 e il 1700" ho studiato un altro precedente progetto, "Le accademie italiane 1525-1700", che era svolto da una collaborazione di Royal Holloway University of London; The University of Reading; The British Library.
Il progetto "Le accademie italiane 1525-1700" rappresenta le prime reti intellettuali dell'Europa moderna (2010-2014), e nasce da un precedente progetto che entro l'estate 2009 aveva completato un ampio catalogo di libri depositati presso British Library pubblicati dalle accademie italiane nelle città di Bologna, Napoli, Padova e Siena.
La Collezione tematica di Accademie Italiane fornisce un database ricercabile dettagliato per l'individuazione di materiale stampato relativo alle Accademie scientifiche italiane attive ad Avellino, Bari, Benevento, Bologna, Brindisi, Caltanissetta, Catania, Catanzaro, Enna, L'Aquila, Lecce,

Mantova, Napoli, Padova, Palermo, Roma, Salerno, Siena, Siracusa, Trapani e Venezia nel periodo 1525-1700 e ora custodite nelle collezioni di British Library. si può cercare nel database informazioni su:

- Accademie - i loro nomi, attivita, interessi, posizione, date, emblemi e motti
- Libri - titoli, autori, dettagli di pubblicazione, illustrazioni, censori
- Nomi di persone, soprannomi, date, nazionalità, motti, emblemi, ritratti e ruoli

È infatti molto peculiare che un fenomeno culturale pervasivo come il movimento accademico fiorito in Italia nel tardo Rinascimento, che ha coinvolto e collegato migliaia di persone attraverso la penisola in spazi privati e pubblici, che ha promosso la pubblicazione di centinaia di libri, e che ha spianato la strada verso le successive reti intellettuali europee, possa essere disprezzato e quasi cadere nel dimenticatoio.
È anche molto interessante vedere che il lavoro di mappatura della presenza di accademie in tutto il territorio italiano e la preparazione di una valutazione rigorosa della loro importanza è dovuto a Michele Maylender, avvocato di formazione che viveva ai confini della lingua italiana. Sebbene alcuni importanti contributi siano venuti da studiosi italiani del XX secolo, nessun altro lavoro dopo la Storia delle accademie d'Italia di Maylender (Bologna: Cappelli, 1926-30) ha aggiornato la sua ricerca sul movimento accademico italiano come un fenomeno globale.
“Il database delle accademie italiane (IAD)”, che è l'output principale del progetto di database “IAD”, ha finora registrato 585 accademie in 44 città e oltre 7000 persone coinvolte in tutte le accademie o la pubblicazione dei 905 libri pubblicati sotto gli auspici delle accademie.

3.1.2 La banca dati delle accademie italiane

Lo IAD è una risorsa digitale che consente il collegamento di persone (nel loro vari ruoli accademico, tipografo, autore, dedicatario, collaboratore, incisore, censore, editore), pubblicazioni, accademie e città. Poiché una rete così vasta e complessa di riferimenti incrociati non è possibile sulla pagina stampata, lo strumento digitale rappresenta la geografia e la storia delle accademie italiane in un modo che dura da tempo influenzato e continua a influenzare la migliore ricerca sulla cultura italiana, storia, vale a dire, la necessità di studiare la geografia, così come la storia, di Letteratura italiana. L'attitudine al cambiamento nella dottrina che i BigData e le discipline umanistiche digitali stanno portando alla ribalta è stata commentata in un articolo del 2010 nel New York Times. Patricia Cohen ha commentato la nuova tendenza in Umanistica: "La prossima grande idea nel linguaggio, nella storia e nelle arti?" Dati.
I membri di una nuova generazione di umanisti esperti di tecnologia digitale sostengono che è giunto il momento di smettere di cercare ispirazione nel prossimo "ISMO" politico o filosofico e iniziare ad esplorare come la tecnologia sta cambiando la nostra comprensione delle arti liberali. Quest'ultima frontiera riguarda il metodo, dicono, usando potenti tecnologie e vasti depositi di materiali digitalizzati che i precedenti studiosi di scienze umanistiche non avevano.
Se c'è un fenomeno storico e sociale che necessita di essere quantificato e reinterpretato alla luce dei nuovi dati acquisiti e dei nuovi media con cui sono archiviati i dati, è il "Movimento accademico italiano" per usare la definizione di Richard Samuels - in modo indicativo, ha adottato anche Daniel Roche la stessa definizione per la sua ricerca pionieristica nella provincia francese accademie e accademici del XVIII secolo.

3.2 Obiettivo

Questo progetto si basa su “IAD (Italian Academies Database)”, secondo database, esistono scrittori, i loro libri, le accademie ed i suoi membri, etc e ovvio ci sono relazioni tra loro. Il lavoro principale di questa tesi è visualizzare queste relazioni, per esempio tra scrittori e loro opere, in modo grafici.
Quindi se vogliamo sapere quali sono i libri dello scrittore X, e lui a quale accademia appartiene, possiamo osservare tutto questo in grafi. Si possono considerare le opzioni sotto elencate come vantaggi del progetto:

- accessibilità a più dati in meno tempo;
- scoprire relazioni che si trovano velocemente;
- osservare relazioni invece di studiare tanta informazione nel database.

3.3 Fasi del procedimento e strumenti

In questo progetto ci sono due parti Information architecture (IA) ed User experience design (UX), ho usato strumenti come neo4j e web server nella parte IA, e nella parte UX ho usato strumenti come HTML, CSS, JAVAScript e altre librerie che in seguito si continua a discuterne.

3.3.1 Architettura dell'informazione

È la struttura organizzativa logica e semantica delle informazioni, dei contenuti, dei processi e delle funzionalità di un sistema o ambiente informativo. L'architettura dell'informazione è il cuore di un qualsiasi progetto di interaction design. Applicabile anche al di fuori dell'ambito informatico, questa struttura è l'anima fondamentale di un qualsiasi insieme di contenuti e dati destinati alla fruizione, integrando informazioni e processi, svolge un ruolo chiave nel definire il reale grado di fruibilità e di usabilità di un sistema per l'utente finale. L'architettura dell'informazione comprende l'analisi, la scelta e la progettazione degli strumenti tecnici e culturali per l'organizzazione, la catalogazione, la ricerca, la navigazione e la presentazione di contenuti e dati nei vari formati disponibili (digitali e non).
Secondo un noto schema proposto da Peter Morville e Louis Rosenfeld, lo scopo fondamentale dell'architettura delle informazioni è mettere in relazione utenti e contenuti, con una finalità di business. Dell'architettura dell'informazione sono state date via via varie definizioni, più generali (cioè non legate essenzialmente al digitale) o più specifiche (riferite al web). Queste sono quelle fornite dall Information Architecture Institute, riprese anche nell'ultima edizione del libro di Rosenfeld e Morville:

- il design strutturale di ambienti informativi condivisi;
- l'arte e la scienza di organizzare ed etichettare siti web, intranet, comunità online e software con l'obiettivo di favorire l'usabilità e la trovabilità (findability);
- una emergente comunità di pratica focalizzata nell'applicare all'ambiente digitale i principi del design e della architettura.

Come si vede soltanto le ultime due definizioni fanno esplicito riferimento al web o alla sfera digitale. Infatti, *ubiquitous computing, internet of things* e dispositivi mobili hanno reso ormai internet sempre

più pervasivo, presente quindi in una quantità sempre più ampia di oggetti e ambienti quotidiani. Ciò ha reso i confini tra online e offline, fisico e digitale sempre più labili: ecco il motivo per cui la prima definizione parla di ambienti informativi condivisi in generale.

3.3.2 User experience design

Lo *user experience design* (UX, UXD, UED o XD) è il processo volto ad aumentare la soddisfazione e la fedeltà del cliente migliorando l'usabilità, la facilità d'uso e il piacere fornito nell'interazione tra il cliente e il prodotto. La progettazione dell'esperienza utente comprende la tradizionale progettazione interazione uomo-macchina e la estende indirizzandosi a tutti gli aspetti del prodotto o del servizio come percepito dagli utenti. L'esperienza utente è qualsiasi aspetto di un'interazione della persona con un dato sistema IT, includendo l'interfaccia, la grafica, la progettazione industriale, l'interazione fisica e manuale.

3.3.3 Fase 1

Prima di tutto per mostrare le relazioni dobbiamo capire cosa c'è nel database IAD, in cui si possono trovare informazione su:

- le Accademie - i suoi nomi, attività, interessi, posizione, date, emblemi e motti;
- libri - titoli, autori, dettagli di pubblicazione, illustrazioni, censori;
- persone: nomi, soprannomi, date, nazionalità, motti, emblemi, ritratti e ruoli.

Quindi ho portato database su mySQL con accesso tramite phpmyadmin, In seguito si parla degli strumenti utilizzati per il progetto.

3.3.3.1 MySQL

MySQL o Oracle MySQL è un Relational database management system (RDBMS) composto da un client a riga di comando e un server. Entrambi i software sono disponibili sia per sistemi Unix e Unix-like sia per Windows; le piattaforme principali di riferimento sono Linux e Oracle Solaris.MySQL è un software libero rilasciato a doppia licenza, compresa la GNU General Public License, ed è sviluppato per essere il più possibile conforme agli standard ANSI SQL e ODBC SQL.I sistemi e i linguaggi di programmazione che supportano MySQL sono molto numerosi: ODBC, Java, Mono, .NET, PHP, Python e molti altri.Le piattaforme LAMP e WAMP incorporano MySQL per l'implementazione di server per gestire siti web dinamici, inoltre molti dei Content Management System di successo come WordPress, Joomla e Drupal nascono proprio con il supporto predefinito a MySQL.Il codice sorgente di MySQL era inizialmente di proprietà della società MySQL AB, veniva però distribuito con la licenza GNU GPL oltre che con una licenza commerciale. Fino alla versione 4.0, una buona parte del codice del client era licenziato con la GNU LGPL e quindi poteva essere utilizzato anche per applicazioni proprietarie. Dalla versione 4.1 in poi, anche il codice del client è distribuito sotto GNU GPL. Esiste peraltro una clausola estensiva che consente l'utilizzo di MySQL con una vasta gamma di licenze libere.Nel luglio 2007 la società svedese MySQL AB aveva 385 dipendenti in numerosi paesi. I suoi principali introiti provenivano dal supporto agli utilizzatori di MySQL tramite il pacchetto Enterprise, dalla vendita delle licenze commerciali e dall'utilizzo da parte di terzi del marchio MySQL.

3.3.3.2 PhpMyAdmin

phpMyAdmin è un'applicazione web scritta in PHP, distribuita con licenza GPL, che consente di amministrare un database MySQL o MariaDB tramite un qualsiasi browser. L'applicazione è indirizzata sia agli amministratori del database, sia agli utenti. Gestisce i permessi prelevandoli dal database. phpmyadmin permette di creare un database da zero, creare le tabelle ed eseguire operazioni di ottimizzazione sulle stesse. Presenta un feedback sulla creazione delle tabelle per evitare eventuali errori. Sono previste delle funzionalità per l'inserimento dei dati (popolazione del database), per le query, per il backup dei dati, ecc. L'amministratore ha anche a disposizione un'interfaccia grafica per la gestione degli utenti: l'interfaccia permette l'inserimento di un nuovo utente, la modifica della relativa password e la gestione dei permessi che l'utente ha sul database.

IAD è accessibile sul sito di British Library in tanti modi ma tutto in formato XML. Il lavoro con xml è molto complicato in database a grafo. avevo ricevuto un versione finale da un'altra lavoro che risulta database IAD in SQL. In seguito vediamo diagramma e struttura del database:

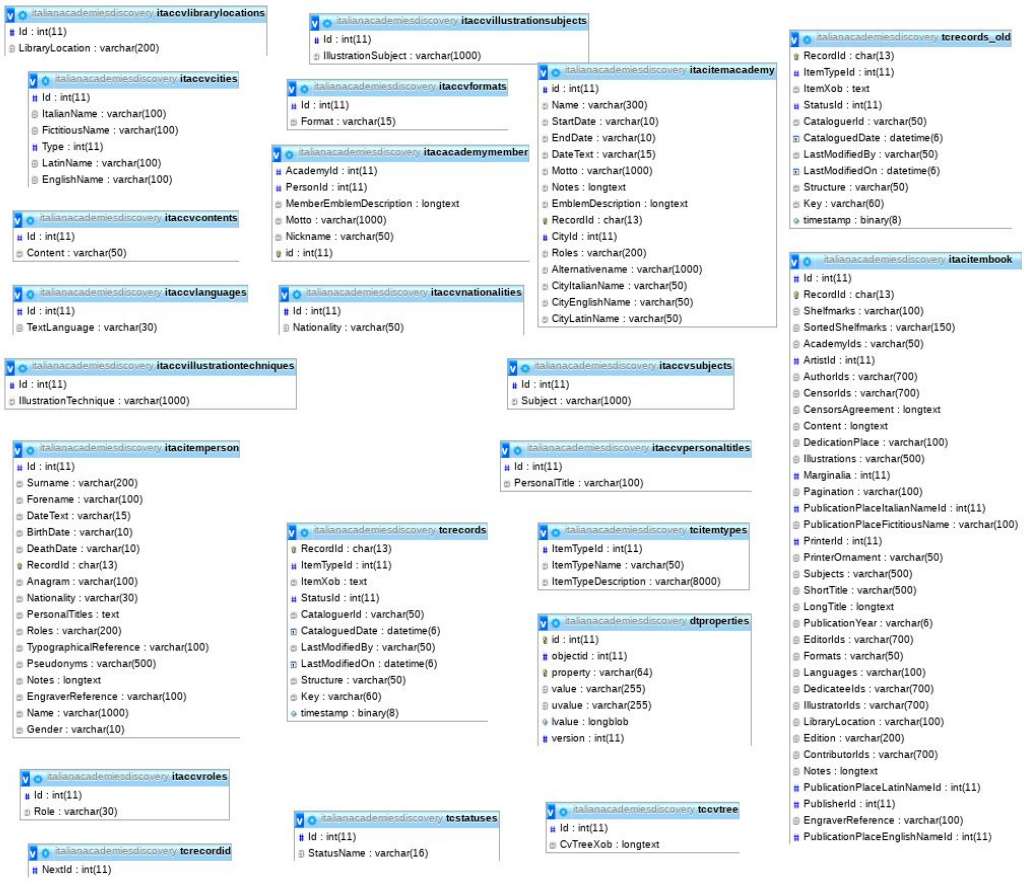


figura 3.3.3.2-1 Diagramma Database

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	dtproperties	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	16 KiB	-
☐	itacacademymember	Browse Structure Search Insert Empty Drop	5,386	InnoDB	utf8_general_ci	800 KiB	-
☐	itaccvcities	Browse Structure Search Insert Empty Drop	87	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvcontents	Browse Structure Search Insert Empty Drop	47	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvformats	Browse Structure Search Insert Empty Drop	16	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvillustrationsubjects	Browse Structure Search Insert Empty Drop	140	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvillustrationtechniques	Browse Structure Search Insert Empty Drop	27	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvlanguages	Browse Structure Search Insert Empty Drop	38	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvlibrarylocations	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvnationalities	Browse Structure Search Insert Empty Drop	357	InnoDB	utf8_general_ci	48 KiB	-
☐	itaccvpersonaltitles	Browse Structure Search Insert Empty Drop	464	InnoDB	utf8_general_ci	48 KiB	-
☐	itaccvroles	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8_general_ci	16 KiB	-
☐	itaccvsubjects	Browse Structure Search Insert Empty Drop	328	InnoDB	utf8_general_ci	16 KiB	-
☐	itacitemacademy	Browse Structure Search Insert Empty Drop	589	InnoDB	utf8_general_ci	304 KiB	-
☐	itacitembook	Browse Structure Search Insert Empty Drop	776	InnoDB	utf8_general_ci	1.8 MiB	-
☐	itacitemperson	Browse Structure Search Insert Empty Drop	7,100	InnoDB	utf8_general_ci	7.3 MiB	-
☐	tccvtree	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	16 KiB	-
☐	tcitemtypes	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8_general_ci	16 KiB	-
☐	tcrecordid	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8_general_ci	16 KiB	-
☐	tcrecords	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	64 KiB	-
☐	tcrecords_old	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	16 KiB	-
☐	tcstatuses	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8_general_ci	16 KiB	-
	22 tables	Sum	15,375	InnoDB	utf8_general_ci	10.5 MiB	0 B

figura 3.3.3.2-2 Strumenti di database

Database di IAD è veramente dettagliato e completo e i dati non erano del tutto puliti, Quindi ho iniziato a pulire il database, ma si può accedere a tabelle escluse in caso di bisogno.

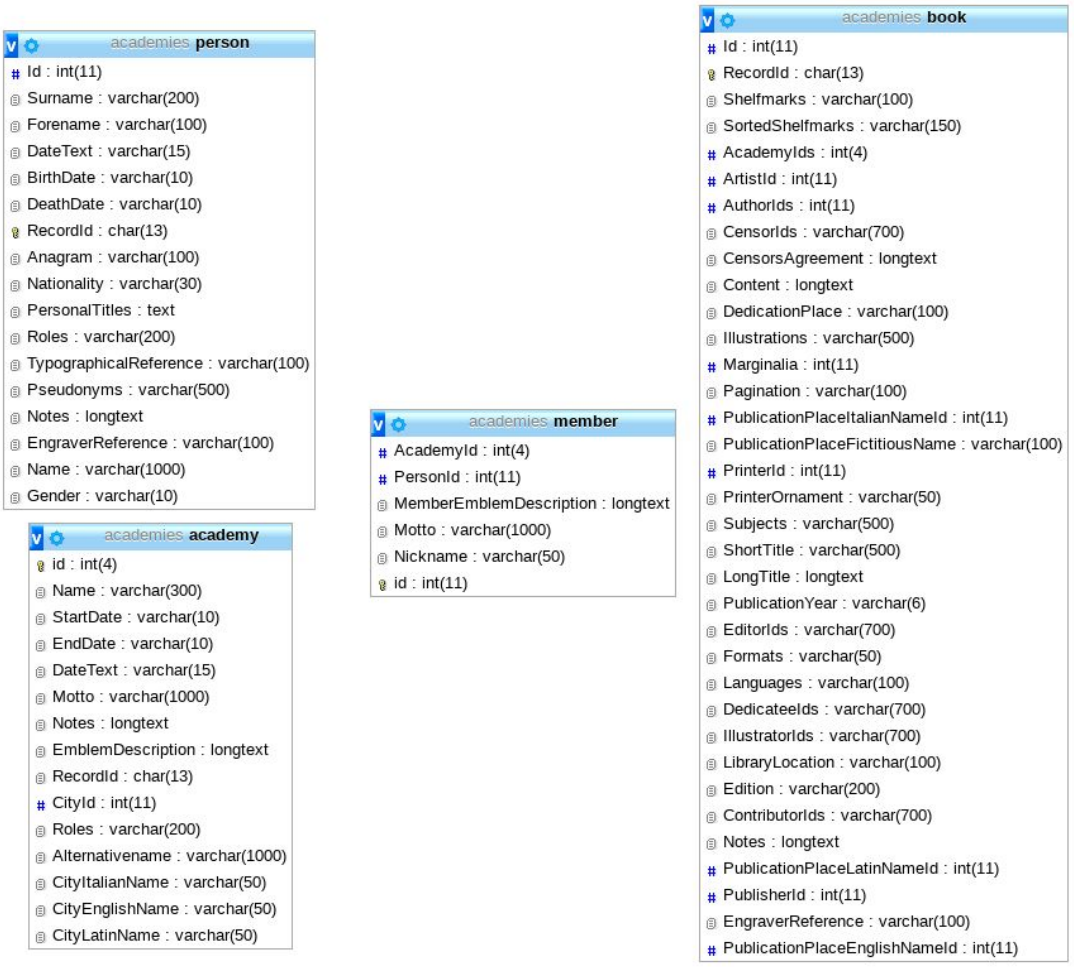


figura 3.3.3.2-3 Diagramma del progetto

Table	Action	Rows	Type	Collation	Size	Overhead
academy	Browse Structure Search Insert Empty Drop	589	InnoDB	utf8_general_ci	320 KiB	-
book	Browse Structure Search Insert Empty Drop	776	InnoDB	utf8_general_ci	1.8 MiB	-
member	Browse Structure Search Insert Empty Drop	5,386	InnoDB	utf8_general_ci	784 KiB	-
person	Browse Structure Search Insert Empty Drop	7,100	InnoDB	utf8_general_ci	7.3 MiB	-
4 tables	Sum	13,851	InnoDB	utf8_general_ci	10.1 MiB	0 B

figura 3.3.3.2-4 Strumenti del progetto

In questo momento abbiamo i dati puliti, ho cominciato a creare il modello di dati (Data Model) per creare la nostra relazione tra nodi, in questo progetto obiettivo e dimostrare relazioni tra Persone e Accademie, Libri con le Persone, Libri con le Accademie.

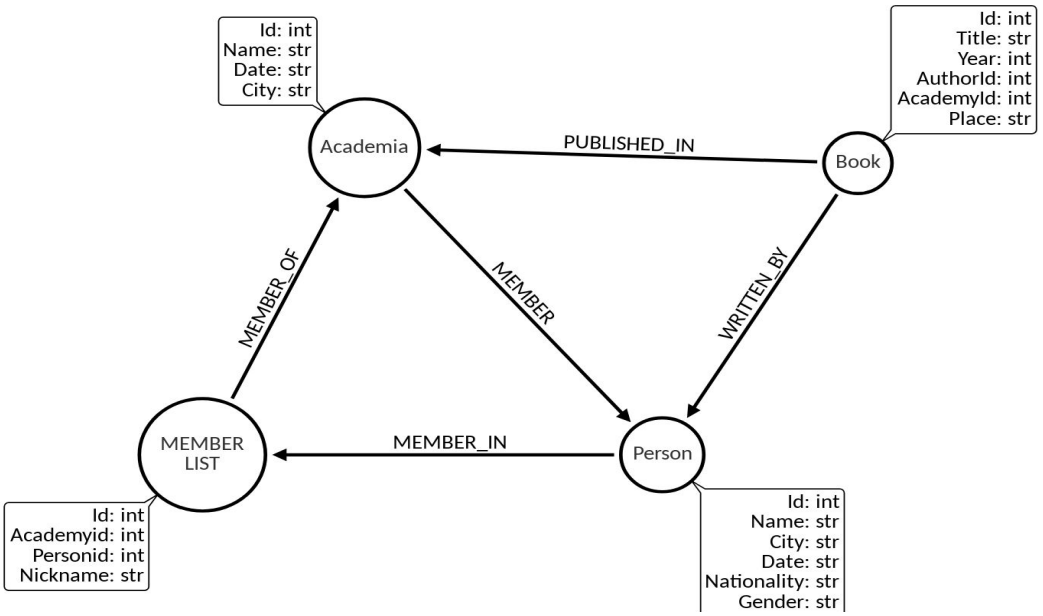


figura 3.3.3-1 Data model

3.3.3.3 Arrows - graph diagram library

Libreria JavaScript per disegnare diagrammi di piccoli graffi, usando D3 per generare SVG. Utile per spiegare i concetti di modellazione grafica Neo4j in presentazioni e blog.

Dopo che abbiamo il nostro modello, importeremo i dati su neo4j per creare la nostra relazione, Neo4j preferisce import LOAD CSV, abbiamo tanti modelli per fare import, qui vediamo due modelli a fare import, prima manuale che dobbiamo scrivere tutto lo schema e mettere i nodi e proprietà o possiamo usare ETL Tool per fare importo.

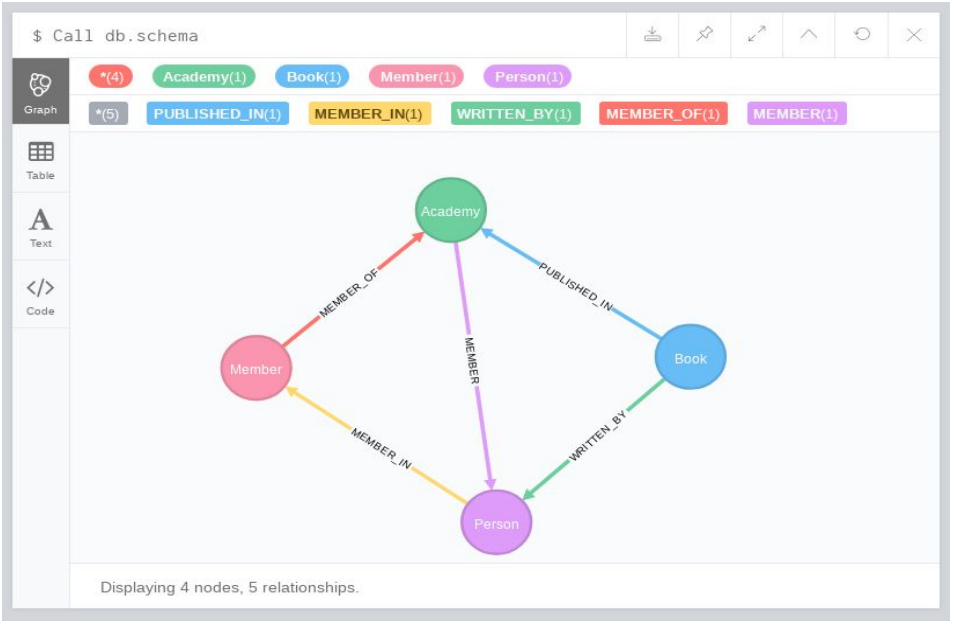


figura 3.3.3-2 Data Model Progetto nell Neo4j

3.3.3.4 Importazione di dati da un database relazionale

Quando hai una buona conoscenza del modello dei dati del tuo grafo e ritieni di rappresentare accuratamente i dati, pronto per trasferire i dati dalla struttura relazionale a un grafo!
Esistono 3 principali approcci per spostare i dati relazionali in un grafo.

1. LOAD CSV: probabilmente il modo più semplice per importare i dati dal tuo database relazionale. Richiede un dump di singole entità tabelle e join tabelle formattate come file CSV.
2. APOC: Awesome Procedures on Cypher. Creato come libreria di estensione per fornire procedure e funzioni comuni agli sviluppatori. Le procedure utili includono apoc.load.jdbc, apoc.load.csv, apoc.load.json e altri.
3. Strumento ETL: strumento di interfaccia utente costruito internamente che converte relazionale in grafico da una connessione JDBC. Consente l'importazione di dati in blocco per set di dati di grandi dimensioni con prestazioni rapide e un'esperienza utente semplice.

3.3.3.4.1 Carico CSV

L'URL del file CSV viene specificato utilizzando FROM seguito da un'espressione arbitraria che valuta l'URL in questione.

È necessario specificare una variabile per i dati CSV utilizzando AS.
LOAD CSV supporta le risorse compresse con gzip , Deflate e gli archivi ZIP.
I file CSV possono essere archiviati sul server del database e quindi accessibili tramite un file:/// URL.
In alternativa, LOAD CSV supporta anche l'accesso ai file CSV tramite HTTPS , HTTP e FTP .
LOAD CSV seguirà i reindirizzamenti HTTP ma per motivi di sicurezza non seguirà i reindirizzamenti che modificano il protocollo, ad esempio se il reindirizzamento sta passando da HTTPS a HTTP .
LOAD CSV è spesso usato insieme al suggerimento di query PERIODIC COMMIT;

3.3.3.4.2 Impostazioni di configurazione per gli URL dei file

3.3.3.4.2.1 Dbms.security.allow_csv_import_from_file_urls

Questa impostazione determina se Cypher consentirà l'uso di file:/// URL durante il caricamento dei dati utilizzando LOAD CSV. Tali URL identificano i file sul filesystem del server del database. L'impostazione predefinita è vera .
L'impostazione dbms.security.allow_csv_import_from_file_urls=false disabilita completamente l'accesso al file system per LOAD CSV.

3.3.3.4.2.2 dbms.directories.import

Imposta la directory root per gli file:/// URL utilizzati con la LOAD CSV clausola Cypher . Questo deve essere impostato su una singola directory sul filesystem del server del database e farà caricare tutte le richieste dagli file:/// URL relativi alla directory specificata (simile a come chroot funziona Unix). Il valore predefinito è import . Questa è una misura di sicurezza che impedisce al database di accedere ai file al di fuori della directory di importazione standard . L'impostazione dbms.directories.import di essere vuoto rimuove questa misura di sicurezza e consente invece l'accesso a qualsiasi file sul sistema. Questo non è raccomandato.

Gli URL dei file verranno risolti rispetto alla dbms.directories.import directory. Ad esempio, un URL di file sarà in genere simile file:///myfile.csv o file:///myproject/myfile.csv.

- Se dbms.directories.import è impostato sull'importazione del valore predefinito , utilizzando gli URL di cui sopra LOAD CSV si leggerà rispettivamente da <NEO4J_HOME> / import / myfile.csv e <NEO4J_HOME> import / myproject / myfile.csv .
- Se è impostato su / data / csv , l'utilizzo degli URL di cui sopra LOAD CSV verrebbe letto rispettivamente da /data/csv/myfile.csv e / data / csv / myproject / myfile.csv .

3.3.3.4.3 Formato file CSV

Il file CSV da utilizzare con LOAD CSV deve avere le seguenti caratteristiche:

- la codifica dei caratteri è UTF-8;
- la terminazione della linea di estremità dipende dal sistema, ad esempio, è \n su unix o \r\n su windows;
- il terminatore di campo predefinito è , ;
- il carattere del terminatore di campo può essere modificato utilizzando l'opzione FIELDTERMINATOR disponibile nel LOAD CSV comando;
- le stringhe tra virgolette sono consentite nel file CSV e le virgolette vengono eliminate durante la lettura dei dati;
- il carattere per la citazione della stringa è doppia citazione " ;
- se dbms.import.csv.legacy_quote_escaping è impostato sul valore predefinito di true, \viene utilizzato come carattere di escape;
- una virgoletta deve essere racchiusa tra virgolette e sfuggita, con il carattere di escape o una seconda virgoletta doppia.

3.3.3.5 Importa i dati da un file CSV

Per importare i dati da un file CSV in Neo4j, è possibile utilizzare LOAD CSV per ottenere i dati nella query. Quindi lo scrivi nel tuo database usando le normali clausole di aggiornamento di Cypher.
artists.csv.

```
1,ABBA,1992
2,Roxette,1986
3,Europe,1979
4,The Cardigans,1992
```

Query.

```
LOAD CSV FROM 'https://neo4j.com/docs/developer-manual/3.4/csv/artists.csv' AS
line
CREATE (:Artist { name: line[1], year: toInteger(line[2])})
```

Viene Artist creato un nuovo nodo con l' etichetta per ogni riga nel file CSV. Inoltre, due colonne del file CSV sono impostate come proprietà sui nodi.
Risultato.

```
+-------------------+
| No data returned. |
+-------------------+
Nodes created: 4
```

```
Properties set: 8
Labels added: 4
```

3.3.3.6 Importa i dati da un file CSV contenente intestazioni

Quando il tuo file CSV ha intestazioni, puoi visualizzare ogni riga del file come una mappa anziché come una serie di stringhe.
artisti-con-headers.csv.

```
Id,Name,Year
1,ABBA,1992
2,Roxette,1986
3,Europe,1979
4,The Cardigans,1992
```

Query.

```
LOAD CSV WITH HEADERS FROM
'https://neo4j.com/docs/developer-manual/3.4/csv/artists-with-headers.csv' AS
line
CREATE (:Artist { name: line.Name, year: toInteger(line.Year)})
```

Questa volta, il file inizia con una singola riga contenente i nomi delle colonne. Indica questo usando WITH HEADERS e puoi accedere a campi specifici con il loro nome di colonna corrispondente.
Risultato.

```
+-------------------+
| No data returned. |
+-------------------+
Nodes created: 4
Properties set: 8
Labels added: 4
```

3.3.3.7 ETL Tool

ETL è l'abbreviazione di extract, transform, load , tre funzioni di database che vengono combinate in un unico strumento per estrarre i dati da un database e inserirli in un altro database.Extract è il processo di lettura dei dati da un database. In questa fase, i dati vengono raccolti, spesso da più e diversi tipi di fonti.Trasforma è il processo di conversione dei dati estratti dalla forma precedente nella forma in cui deve essere inserito in modo che possa essere inserito in un altro database. La trasformazione avviene utilizzando regole o tabelle di ricerca o combinando i dati con altri dati.Il caricamento è il processo di scrittura dei dati nel database di destinazione.
neo4j-etl-tool è un progetto open source di Neo4j per colmare il divario tra il mondo del database relazionale e il database grafo mondo. Ora puoi facilmente importare il tuo database MySQL nella tua istanza Neo4j con componenti ETL che possono essere attivati da neo4j-etl .

3.3.3.8 Neo4j-etl

Lo strumento ETL Neo4j estrae i metadati da uno schema MySQL e quindi applicare alcune regole di mapping predefinite per un'esportazione CSV che deve essere utilizzata dallo neo4j-import

strumento per l'effettiva importazione bulk. Tutto ciò avviene tramite un singolo comando senza intervento da parte dell'utente. Così:

```
./bin/neo4j-etl mysql export --user  --parola d'ordine --database
northwind \
 --destination $ NEO4J_HOME / data / databases / graph.db /
 --import-tool $ NEO4J_HOME / bin \
 --csv-directory / tmp
```

3.3.3.8.1 ETL Tool gestione della struttura

ETL iniziato con:

1. Tutte le informazioni sono memorizzate in relazioni o **tabelle** => In Neo4j, memorizziamo le informazioni nei nodi con le proprietà (approssimativamente l'equivalente di righe di dati).
2. Relazioni tra gli elementi di informazione memorizzati (righe nelle **tabelle**) indicati da **JOIN** => In Neo4j, le relazioni tra i nodi vengono memorizzate esplicitamente come relazioni .

3.3.3.8.2 non dimenticare le chiavi

I passi:

- Identifica oggetti di prima classe: tabelle e JOIN
- Esporta in Neo4j

Interrogare per le **tabelle** : facile.
Interrogazione per **JOIN** : è qui che si trova il problema. I JOIN non sono cittadini di prima classe in database relazionali. Le chiavi che si uniscono ai tavoli sono.
Quindi abbiamo aggiornato le regole di conseguenza:

1. Tutte le informazioni sono memorizzate in **tabelle** => In Neo4j, memorizzare le informazioni nei nodi con proprietà .
2. Le relazioni tra gli elementi di informazione memorizzati (le righe nelle **tabelle**) sono indicate dalle **chiavi** che uniscono le tabelle => In Neo4j, le relazioni tra i nodi vengono memorizzate esplicitamente come relazioni .

3.3.3.8.3 cosa costituisce una chiave?

I passi:

- Identifica oggetti di prima classe: tabelle e chiavi
- Esporta in Neo4j

Interrogazione per le **tabelle** : già effettuata
Interrogazione per le **chiavi** : le chiavi sono espresse come vincoli in MySQL. I soliti sono **PrimaryKey** e **ForeignKey** .

Chiave primaria: identifica record univoci in una tabella
È possibile in MySQL e in molti altri database definire **PrimaryKey** non una singola colonna ma un gruppo di colonne, ovvero una chiave composta o scomposta. Ad esempio, un autore può essere identificato in modo univoco utilizzando il proprio nome e cognome, nel qual caso combiniamo queste due colonne per creare un identificativo univoco mentre importiamo i dati.

Chiave esterna: identifica le relazioni tra tabelle
Dolce, proprio quello che stavo cercando. Ogni tasto ha un **source** come tabella iniziale e **target** come tabella finale. In altre parole il **start** nodo e il **end** nodo di una relazione in Neo4j.
Ecco un esempio: in che modo i territori sono collegati alle regioni? Tutti i territori sono associati a una regione specifica. Confrontiamo la vista relazionale e grafica del mondo:

la vista del database relazionale in MySQL:
I territori hanno un **ForeignKey RegionId** che si riferisce al **PrimaryKey RegionId** nella **Region** tabella.

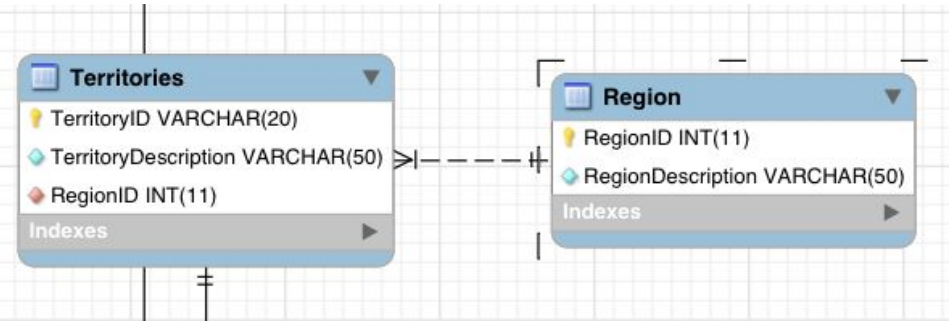


figura 3.3.3.8.3-1 la vista del database relazionale in MySQL

La vista del database grafo in Neo4j:
se la importiamo nel grafo, un Territory nodo iniziale è collegato al **Region** nodo finale in questo modo: **(Territory)-[:REGION]->(Region)**
puoi vedere che non abbiamo più bisogno delle chiavi **(Territories.RegionId, Region.RegionId)** per fornire la relazione tra di esse.

figura 3.3.3.8.3-2 La vista del database grafo in Neo4j

Che aspetto hanno i dati in MySQL:

TerritoryID	TerritoryDescription	RegionID	RegionID	RegionDescription
29202	Columbia	4	4	Southern
30346	Atlanta	4	4	Southern
31406	Savannah	4	4	Southern
32859	Orlando	4	4	Southern
33607	Tampa	4	4	Southern
72716	Bentonville	4	4	Southern
75234	Dallas	4	4	Southern
78759	Austin	4	4	Southern

figura 3.3.3.8.3-3 i dati in mysql

Che aspetto hanno i dati in Neo4j:

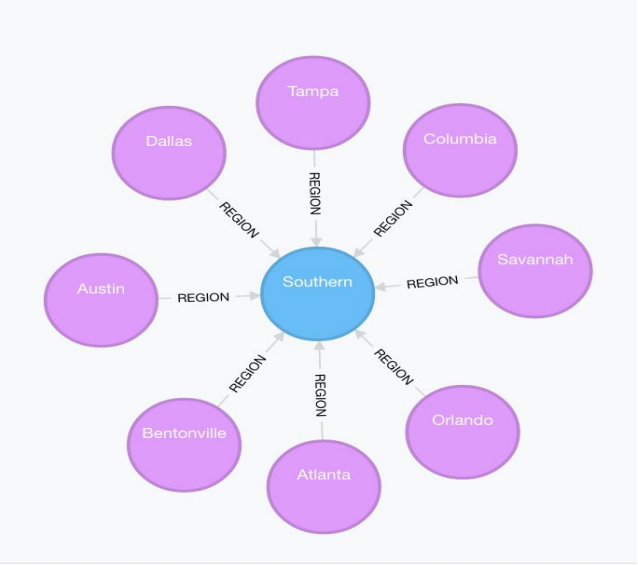


figura 3.3.3.8.3-4 i dati in neo4j

Tasti compositi: casi speciali
Come probabilmente sappiamo per esperienza, un **ForeignKey** vincolo può essere posto su un composito **PrimaryKey**, il che pone una sfida aggiuntiva.
CompositeKeys sono spesso composti da chiavi naturali rilevanti per il dominio. Il database Northwind utilizza invece tasti numerici artificiali (surrogati o sintetici). Ad esempio, **Suppliers** in Northwind potrebbe essere identificato da **TaxNumber** e **Country** invece di artificiale SupplierId. Queste due colonne faranno parte di a **CompositeKey**, che dovresti usare come **ForeignKey** a **Supplier**.
Lo **neo4j-etl** strumento gestisce automaticamente le chiavi composite in base alle meta informazioni del database relazionale.

È necessario un aggiornamento delle regole:
Tutte le informazioni sono memorizzate in **tabelle** => In Neo4j, memorizzare le informazioni nei nodi con proprietà .

Le relazioni tra gli elementi di informazione memorizzati (le righe nelle tabelle) sono indicate da **vincoli** che collegano le tabelle insieme => In Neo4j, le relazioni tra i nodi vengono memorizzate esplicitamente come relazioni .

3.3.3.8.4 JoinTables - Sono JOIN o tabelle?

Una volta identificati chiari oggetti di prima classe come tabelle e vincoli di chiavi primarie e straniere, il prossimo passo ovvio è:

- Esporta in Neo4j

Per esportare, passiamo attraverso l'elenco delle tabelle per identificare quali sono i diversi tipi di vincoli posti su di essi. Alcune tabelle hanno un **PrimaryKey UNIQUE** vincolo posto su di esse, quelle per lo più contengono entità del dominio. Questi sono quelli facili da convertire come nodi.
Mentre lo facevamo, scopriamo che ci sono delle eccezioni: **JoinTables** .

In MySQL ...
La struttura di alcune tabelle rappresenta una tabella JOIN, in cui una tabella viene unita a un'altra tabella attraverso una tabella provvisoria. Questo di solito viene fatto per rappresentare una relazione molti-a-molti o talvolta fatto come un esercizio di normalizzazione.
Ad esempio, **OrderDetails** è una tabella JOIN per indicare un join tra **Orders** e **Products**.

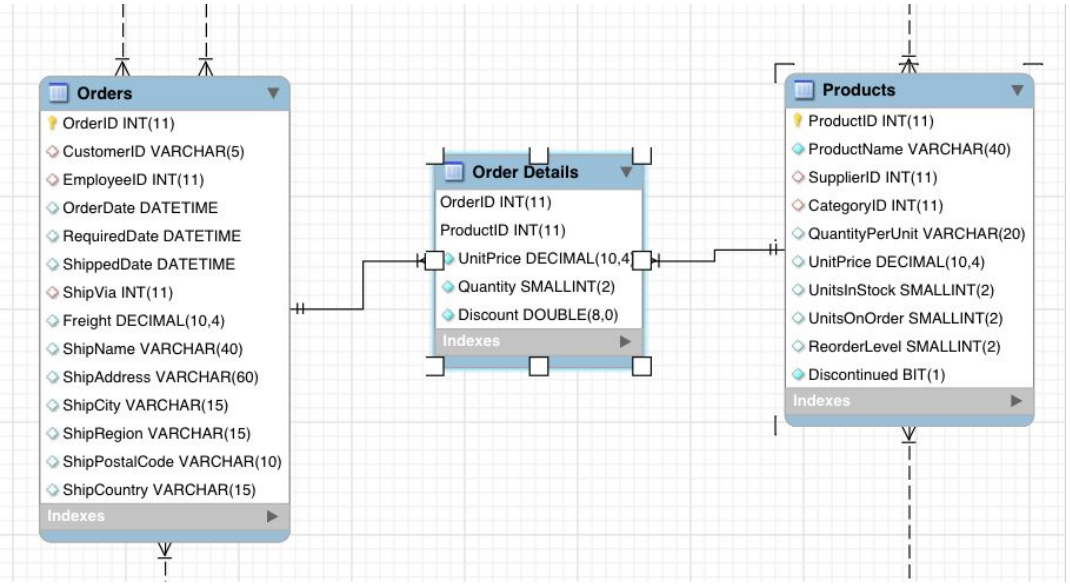


figura 3.3.3.8.4-1 una tabella JOIN in sql

... Tradotto in Neo4j:
Se dovessimo seguire le regole che avevamo precedentemente impostati, ci ritroveremo con tre nodi: **Orders**, **OrderDetails** e **Products**. Ma in Neo4j, le relazioni sono entità di prima classe in modo che possiamo ignorare il nodo intermedio e possono invece importarlo come: **(Order)-[:ORDER_DETAIL]->(Product)**.
A differenza di MySQL, non è necessario creare una tabella per archiviare dettagli su JOIN come **UnitPrice**, **Quantity** o **Discount**. Invece, è possibile memorizzare queste informazioni sulla relazione come una proprietà.

figura 3.3.3.8.4-2 una JOIN in neo4j

Nota: **OrderDetails** ha **OrderId** e **ProductId** come a **CompositePrimaryKey**. La **JoinTable** regola dovrebbe ancora essere applicata in questo scenario.

Si indovina ora, un altro aggiornamento delle regole:
Tutte le informazioni sono memorizzate in **tabelle** => In Neo4j, memorizzare le informazioni nei nodi con proprietà .
Le relazioni tra gli elementi di informazione memorizzati (le righe nelle **tabelle**) sono indicate da **vincoli** che collegano le tabelle insieme => In Neo4j, le relazioni tra i nodi vengono memorizzate esplicitamente come relazioni .
Interpretare i vincoli come **JOIN** o **JoinTables** :

- ➔ **JOINs** => Store JOINs come relazioni .
- ➔ **JoinTables** non sono tabelle; sono anche memorizzati come relazioni .

3.3.3.8.5 Entità intermedie

Tornare al lavoro:

- Identifica oggetti di prima classe: tabelle , vincoli , JOIN e JoinTables
- Esporta in Neo4j

Quando proviamo ad importare JoinTables, non è difficile notare alcune anomalie.

In MySQL ...
Ci sono tabelle che appaiono come tabelle JOIN ma JOIN più di due tabelle, cioè contengono più di due ForeignKeys.
In Northwind, non abbiamo una tabella simile, ma **Orders** potrebbe essere vista come una tabella intermedia che collega tutte le altre tabelle come **Employees**, **Customers** e **Shippers**.

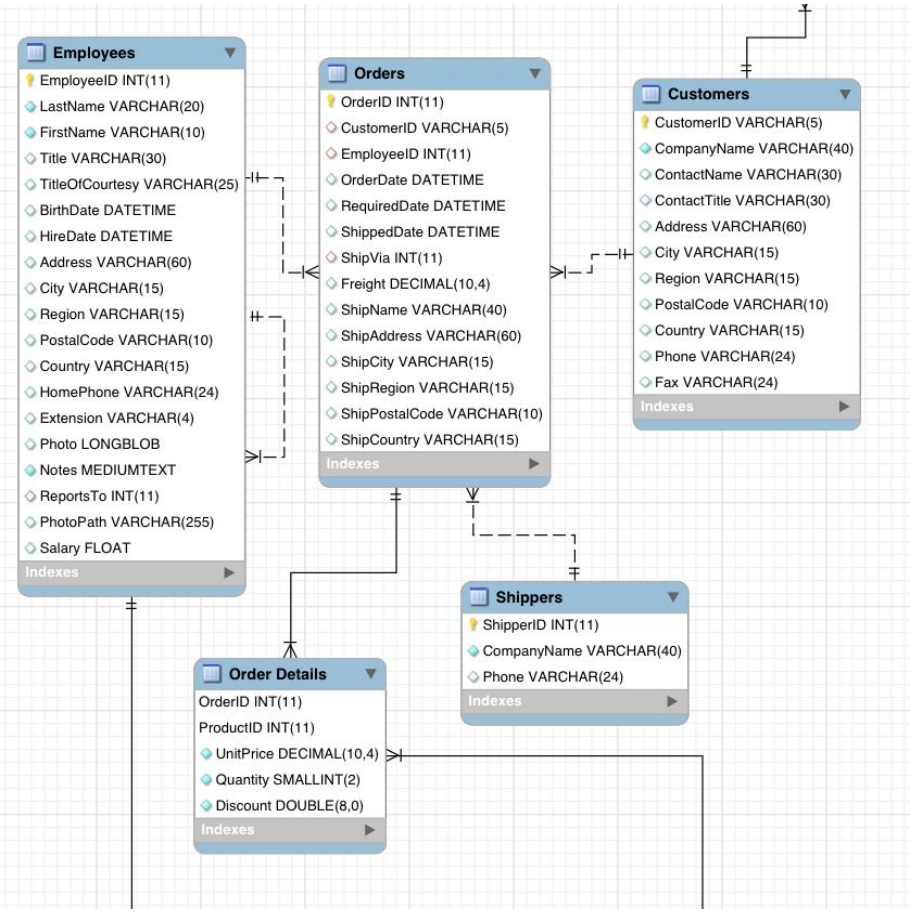


figura 3.3.3.8.5-1 Join piu di 2 tabelle in sql

... Tradotto in Neo4j:

In Neo4j, tale tabella viene importata come nodo. Spesso queste sono entità o concetti "mancanti" nel tuo dominio. La **ForeignKeys** si trasformano in relazioni come previsto e la tavola JOIN è importata come un nodo intermedio .

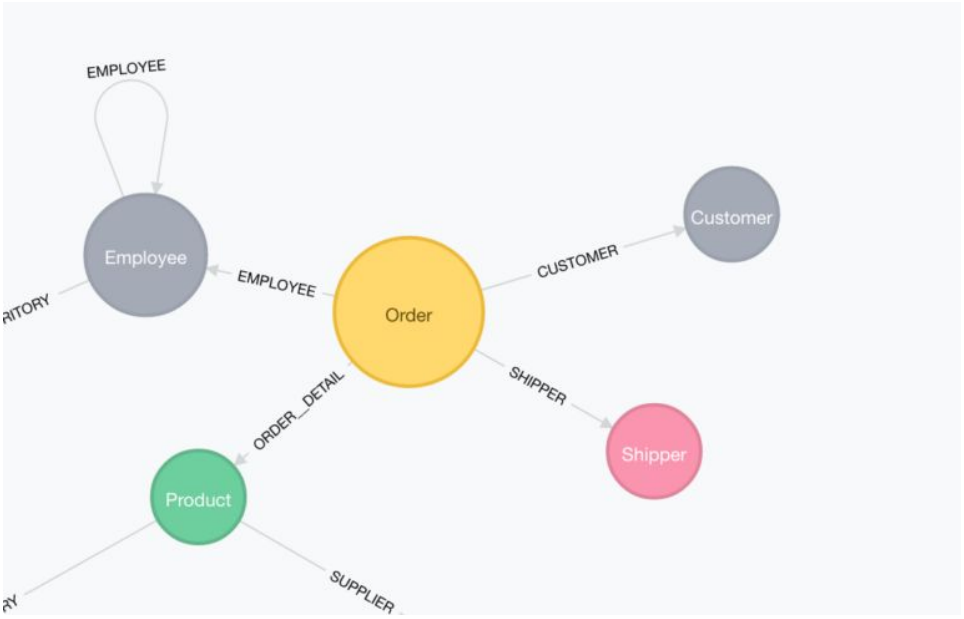


figura 3.3.3.8.5-2 JOINs in neo4j

Quindi aggiorniamo le regole al seguente:

1. Tutte le informazioni sono memorizzate in **tabelle** => In Neo4j, memorizzare le informazioni nei nodi con proprietà .
2. Le relazioni tra gli elementi di informazione memorizzati (le righe nelle **tabelle**) sono indicate da **vincoli** che collegano le tabelle insieme => In Neo4j, le relazioni tra i nodi vengono memorizzate esplicitamente come relazioni .
    A. Interpretare i vincoli come **JOIN** o **JoinTables** .
        a. **JOINs** => Store JOINs come relazioni .
        b. **JoinTables** che hanno esattamente due **ForeignKeys** sono memorizzati come relazioni.
        c. **Le tabelle** che corrispondono al caso del nodo intermedio (più di due **ForeignKeys**) vengono importate come nodi e le **JOIN** alle altre tabelle vengono memorizzate come relazioni .

3.3.3.8.6 Applicazione delle regole di mappatura durante l'esportazione

Una volta mappato l'intero schema su Tables, JOINs e JoinTables, abbiamo iniziato a lavorare sull'effettiva importazione dei dati basata su questi mapping. Abbiamo deciso di utilizzare lo strumento di importazione neo4j per eseguire la nostra importazione in blocco. Questo strumento accetta file CSV che rappresentano nodi e relazioni. Una nota sulla generazione di file CSV: Abbiamo scritto un generatore CSV che genera i file in base ai mapping che abbiamo già identificato interpretando lo schema relazionale ai nodi e alle relazioni .
Entrambe le funzionalità (**generate-mappings** e **export**) sono accessibili tramite lo **neo4j-etl** strumento da riga di comando. è possibile eseguire l'operazione completa in una sola volta, utilizzando il export comando. Abbiamo anche preso una decisione architettonica per avere la capacità di:

A. Genera solo il mapping in un formato JSON utilizzando il generate-mappings comando.
B. Esegui solo l'esportazione utilizzando un file di mappatura già generato passando csv-resources un'opzione al export comando.

Ciò è stato fatto in modo che gli utenti avessero più controllo sul processo, ad esempio per modificare la mappatura in base alla conoscenza del proprio dominio. Dopo tutto, conosci meglio il tuo dominio.

```
./bin/neo4j-etl mysql export --user  --parola d'ordine --database
northwind \
 --destination $ NEO4J_HOME / data / databases / graph.db /
--import-tool $ NEO4J_HOME / bin \
 --csv-directory / tmp
```

3.3.3.8.7 Diagramma di architettura

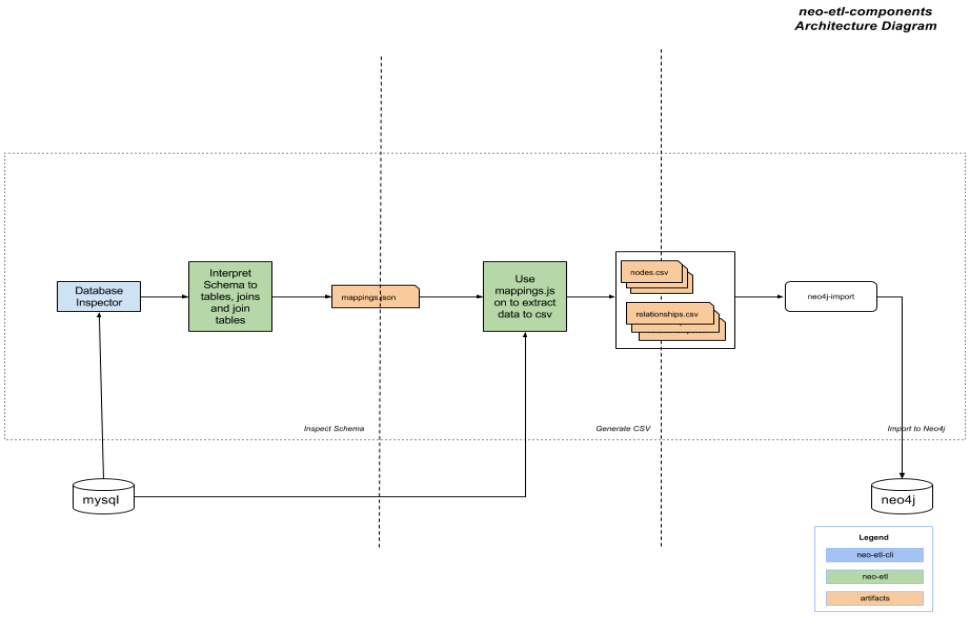


figura 3.3.3.8.7-1 Diagram di architettura

Esempio di mappatura di un modello relazionale a un grafo Iniziamo con il diagramma ER completo di un database Northwind:

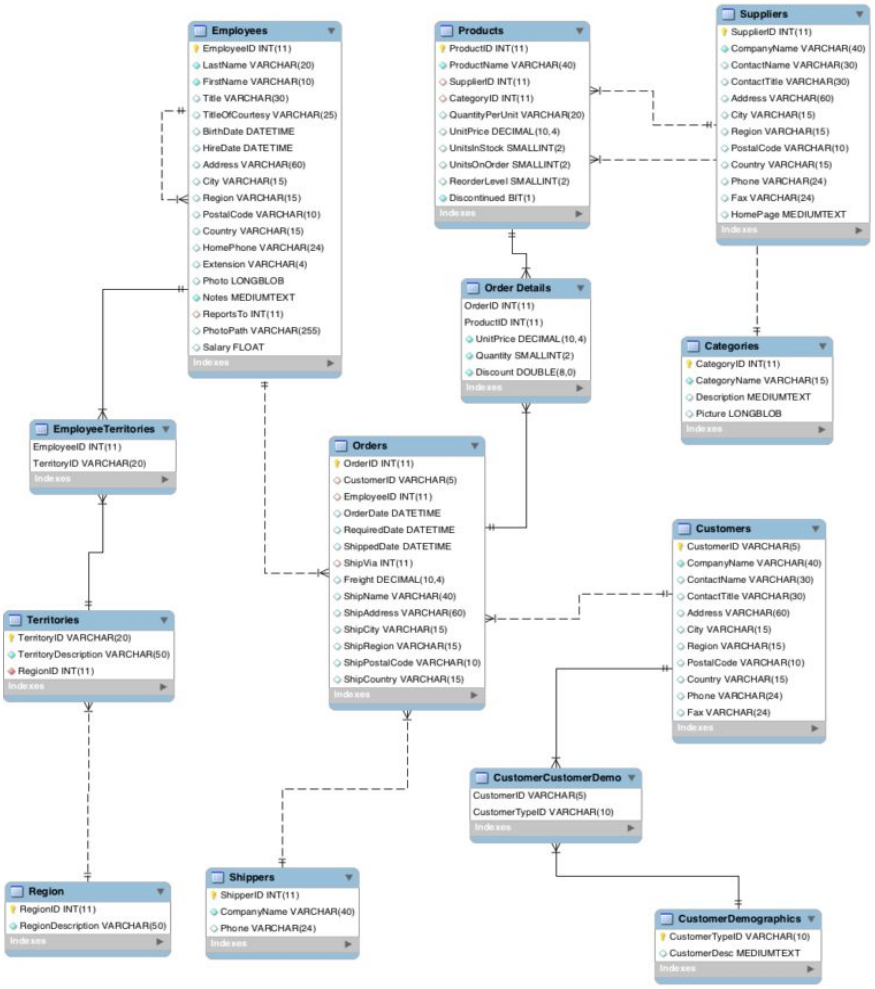


figura 3.3.3.8.7-2 il diagramma ER completo di un database Northwind

After you run the **neo4j-etl** tool against this Northwind database…

```
./bin/neo4j-etl mysql export  --user  --password  --database
northwind \
 --destination $NEO4J_HOME/data/databases/graph.db/ --import-tool
$NEO4J_HOME/bin  \
 --csv-directory /tmp
```

… Il risultato di modello di grafo in Neo4j appare così:

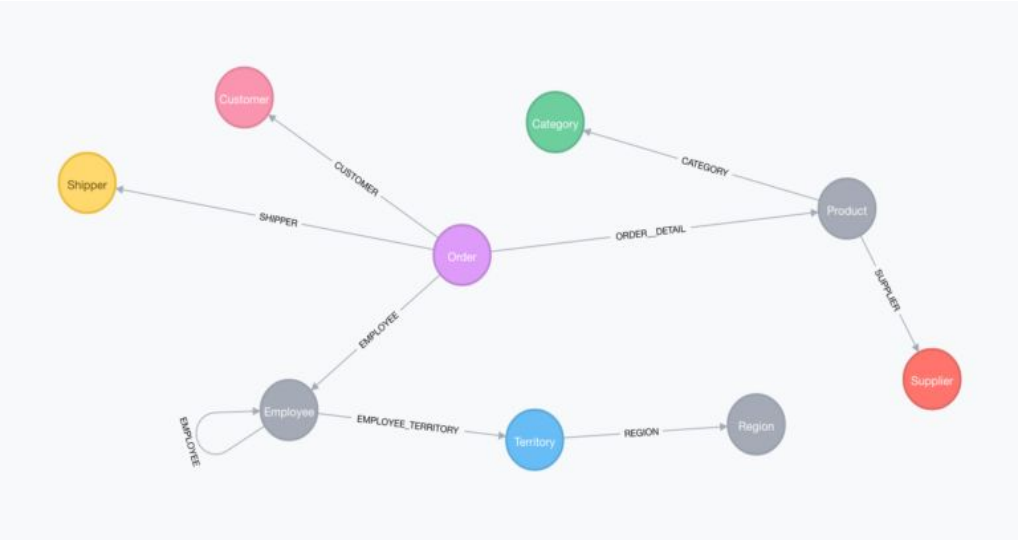


figura 3.3.3.8.7-2 modello di grafo in Neo4j database Northwind

Caratteristiche:

- Neo4j-ETL UI in Neo4j Desktop
- Gestire più connessioni RDBMS
- estrarre automaticamente i metadati del database dal database relazionale
- derivare il modello del grafo
- modifica visivamente etichette, tipi di relazione, nomi di proprietà e tipi
- visualizza il modello corrente come un grafo
- persistono la mappatura come json
- recuperare i dati CSV rilevanti dai database relazionali
- eseguire l'importazione tramite neo4j-import, bolt-connector, cypher-shell, neo4j-shell
- bundle MySQL, PostgreSQL, consente il driver JDBC personalizzato con Neo4j Enterprise

Anche ho trovato ELT Tool come una soluzione migliore quindi ho usato Neo4j-etl per caricare i dati a Neo4j, vediamo i nostri dati nel Neo4j-Browser.

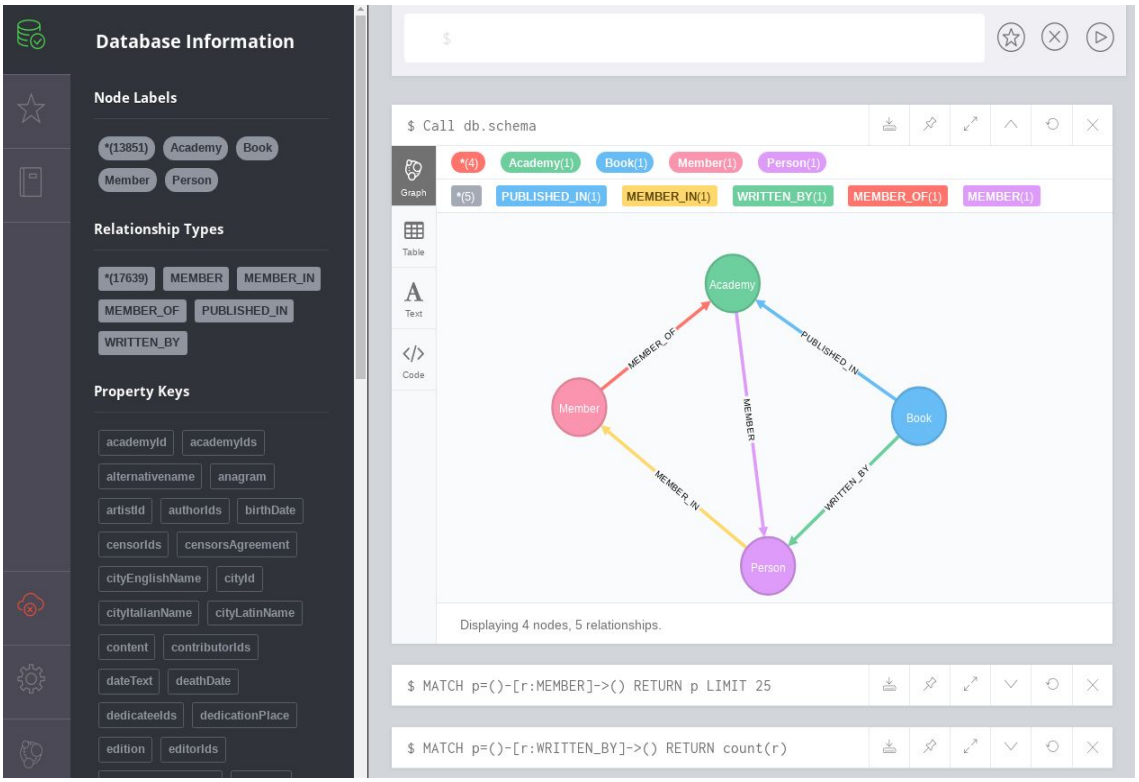


figura 3.3.3-3 Database schema nel Neo4j-Browser

A questo punto il nostro database a grafo è già pronto che include:

- ➔ 589 Academy
- ➔ 7100 Person
- ➔ 776 Book
- ➔ 5386 Member
  - ❖ 5422 MEMBER relationships between Academy to Person
  - ❖ 5424 MEMBER_IN relationships between Person to Member
  - ❖ 5384 MEMBER_OF relationships between Member to Academy
  - ❖ 776 PUBLISHED_IN relationships between Book to Academy
  - ❖ 633 WRITTEN_BY relationships between Book to Person

3.3.4 Fase 2

Questa sezione si dedica alle spiegazioni della parte dei clienti, in continuazione vedremo strumenti utilizzati e suoi componenti.

3.3.4.1 Bootstrap

Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript.

3.3.4.2 D3

D3.js (or just D3 for Data-Driven Documents) is a JavaScript library for producing dynamic, interactive data visualizations in web browsers. It makes use of the widely implemented SVG, HTML5, and CSS standards. It is the successor to the earlier Protovis framework. In contrast to many other libraries, D3.js allows great control over the final visual result. Its development was noted in 2011, as version 2.0.0 was released in August 2011.

3.3.4.3 Vis

vis.js Una libreria di visualizzazione dinamica basata su browser. La libreria è progettata per essere facile da usare, per gestire grandi quantità di dati dinamici e per consentire la manipolazione e l'interazione con i dati. La libreria è composta dai componenti DataSet, Timeline, Network, Graph2d e Graph3d.

3.3.4.4 Git

Git è un sistema di controllo della versione per tracciare i cambiamenti nei file del computer e coordinare il lavoro su quei file tra più persone. Viene utilizzato principalmente per la gestione del codice sorgente nello sviluppo del software, ma può essere utilizzato per tenere traccia delle modifiche in qualsiasi set di file. Come sistema di controllo di revisione distribuito, è finalizzato alla velocità, all'integrità dei dati e al supporto per flussi di lavoro distribuiti e non lineari. Git è stato creato da Linus Torvalds nel 2005 per lo sviluppo del kernel Linux, con altri sviluppatori del kernel che hanno contribuito al suo sviluppo iniziale. Il suo attuale manutentore dal 2005 è Junio Hamano. Come con la maggior parte degli altri sistemi di controllo delle versioni distribuiti, e diversamente dalla maggior parte dei sistemi client-server, ogni directory Git su ogni computer è un repository completo con una cronologia completa e capacità di tracciamento della versione complete, indipendentemente dall'accesso alla rete o da un server centrale. Git è un software gratuito e open source distribuito secondo i termini della GNU General Public License versione 2.

3.3.4.5 Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Esso include supporto per debugging, un controllo per Git integrato, Syntax highlighting, IntelliSense, Snippet e refactoring del codice. È anche personalizzabile: gli utenti possono cambiare il tema dell'editor, le scorciatoie da tastiera, e le preferenze. È un software libero, anche se la versione ufficiale è sotto una licenza proprietaria. Visual Studio Code è basato su Electron, un framework con cui è possibile sviluppare applicazioni Node.js.

3.3.4.6 Live-server

è un piccolo server di sviluppo con funzionalità live reload. Usalo per hackerare i tuoi file HTML / JavaScript / CSS, ma non per distribuire il sito finale. Ci sono due ragioni per l'utilizzo di questo:

1. Le richieste AJAX non funzionano con il protocollo file: // a causa di restrizioni di sicurezza, ad esempio se hai bisogno di un server se il tuo sito recupera il contenuto tramite JavaScript.

2. Avere la pagina caricata automaticamente dopo che le modifiche ai file possono accelerare lo sviluppo.

Si riguarda parti diversi del sito, in primo passo nella parte superiore della pagina abbiamo sezione About.

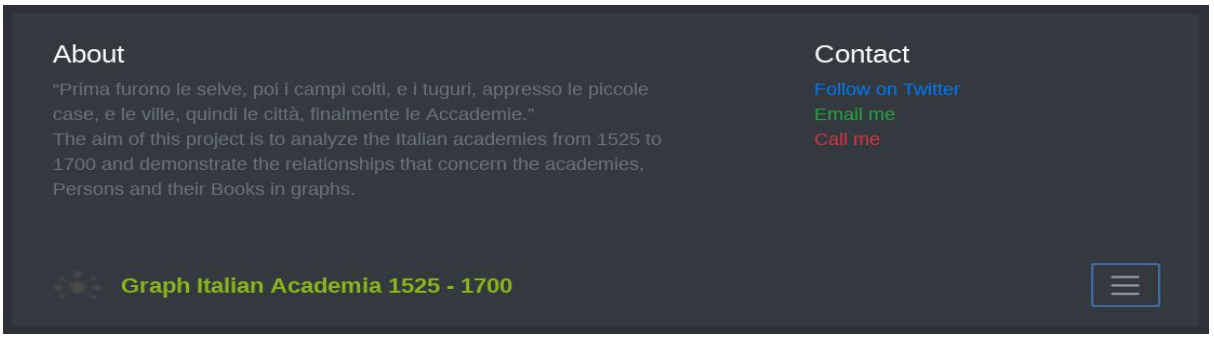


figura 3.3.4-1 About

Nella parte centrale si trova cruscotto dei grafi che include tre parti.

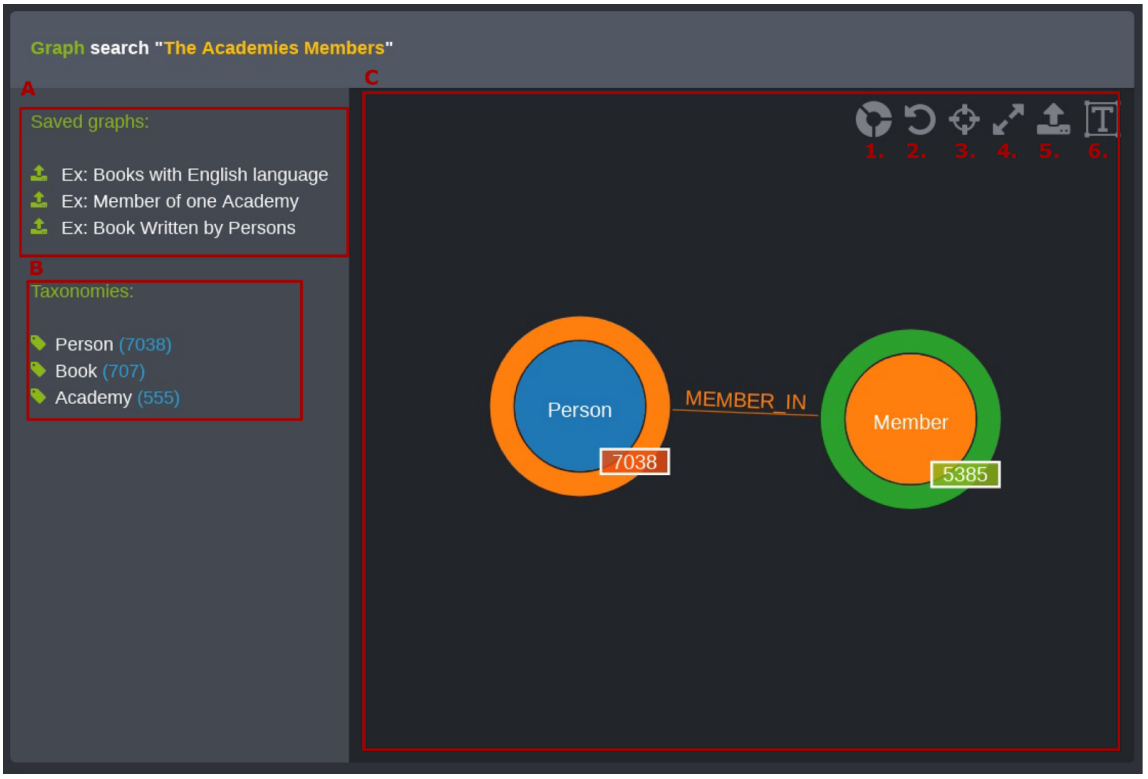


figura 3.3.4-2 Dashboard

A. Saved Graphs: Una tabella che ci da possibilità di visualizzare le modelli già creati anche possibilità di salvare modelli desiderati dell'utente.
B. Taxonomies: Una lista che prevede classificazione dei nodi nel database.
C. Un'interfaccia interattiva progettata per creare query per utenti non tecnici, il grafo è costituito da nodi selezionabili collegati tra loro da collegamenti, e un elenco di azioni disponibili nel contenitore del grafo.

1. le relazioni
2. Aggiorna
3. posizione primaria
4. schermo intero
5. salva il grafo
6. visualizza il testo

Inferiormente si trova sezione di visualizzazione del grafo che abbiamo scelto, che include parte scritto che conferme nostra scelta e parte visuale.

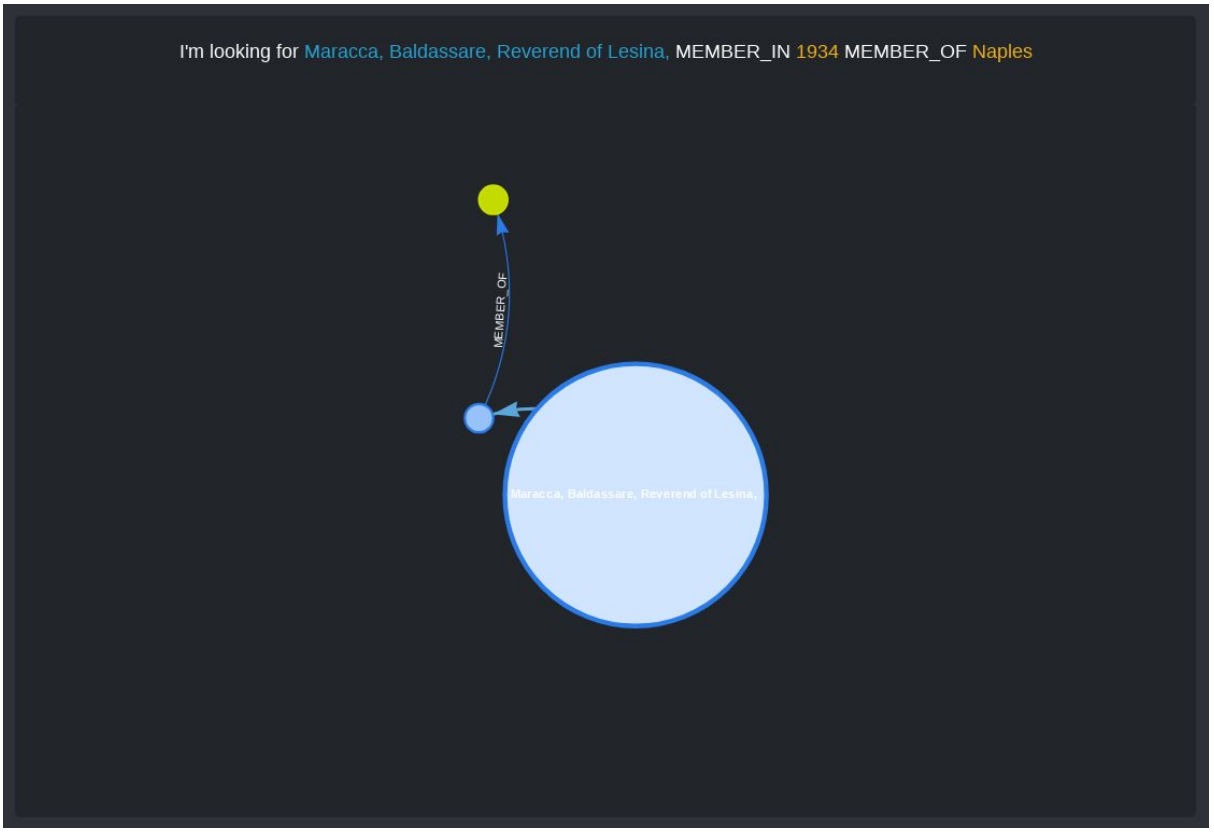


figura 3.3.4-3 Visual

Prossimamente abbiamo parte sviluppatore che genera cypher query dalla nostra scelta e c'è possibilità di aprire pagina Query bar per fare query direttamente a server Neo4j.

figura 3.3.4-4 Dev

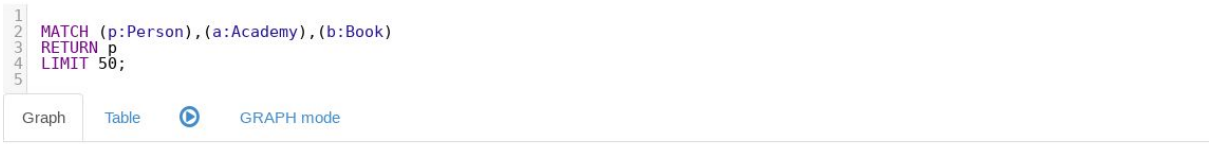


figura 3.3.4-5 Query Bar

Alla fine abbiamo risultato del nostro grafo come una lista.

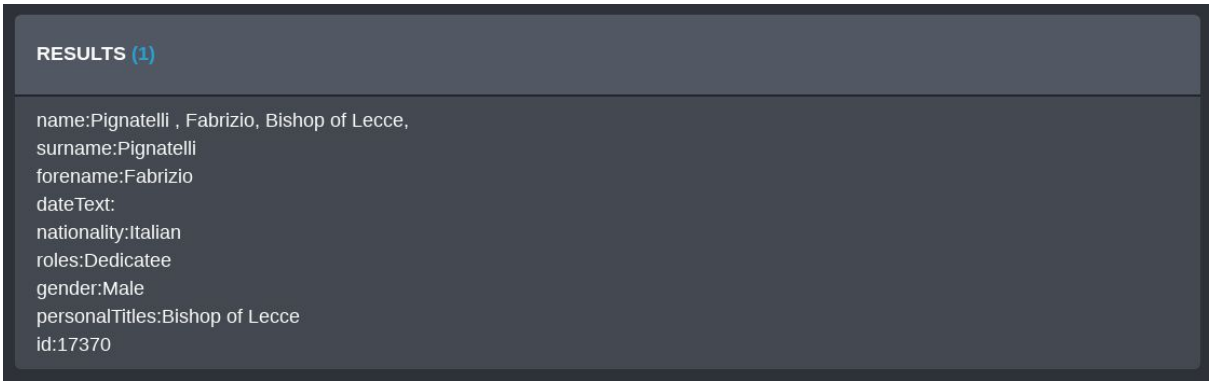


figura 3.3.4-6 Risultato

3.4 Outcome

Ho trasformato database IAD a database a grafo e ho scritto un interfaccia per permettere di accedere a tutte le possibilità di database, quindi il sito permette visualizzare relazioni tra nodi secondo database a grafo.

Nella parte di database a grafo ho create 13851 nodi e 17639 relazioni, per esempio abbiamo campione alcuni nodi, riportando i conteggi delle proprietà e delle relazioni per nodo.

figura 3.4-1 campione nodi

la proprietà per i nodi di Book sono:

```
["publisherId", "illustratorIds", "libraryLocation", "printerOrnament",
"shortTitle", "subjects", "publicationYear", "censorsAgreement",
"printerId", "academyIds", "recordId", "publicationPlaceItalianNameId",
"publicationPlaceEnglishNameId", "artistId", "id", "languages",
"editorIds", "formats", "dedicateeIds", "authorIds", "censorIds",
"illustrations", "notes", "shelfmarks", "longTitle", "pagination",
"marginalia", "publicationPlaceLatinNameId"]
```

la proprietà per i nodi di Person sono:

```
["name", "dateText", "surname", "recordId", "deathDate",
"personalTitles", "birthDate", "id", "roles", "nationality", "forename",
"gender"]
```

la proprietà per i nodi di Academy sono:

```
["name", "dateText", "cityLatinName", "endDate", "cityId", "recordId",
"id", "emblemDescription", "roles", "cityItalianName", "notes",
"cityEnglishName", "startDate", "motto", "alternativename"]
```

la proprietà per i nodi di Member sono:

```
["id", "personId", "academyId"]
```

vediamo quantità di relazioni secondo il tipo:

figura 3.4-2 quantità di relazioni

Come abbiamo già visto nella sezione precedente si può considerare il sito che è un visualizzatore dei grafi come outcome principale del mio progetto, un'interfaccia interrogativa con possibilità di creare modelli da informazione di database e visualizzare i risultati in modo grafico ed scritto.

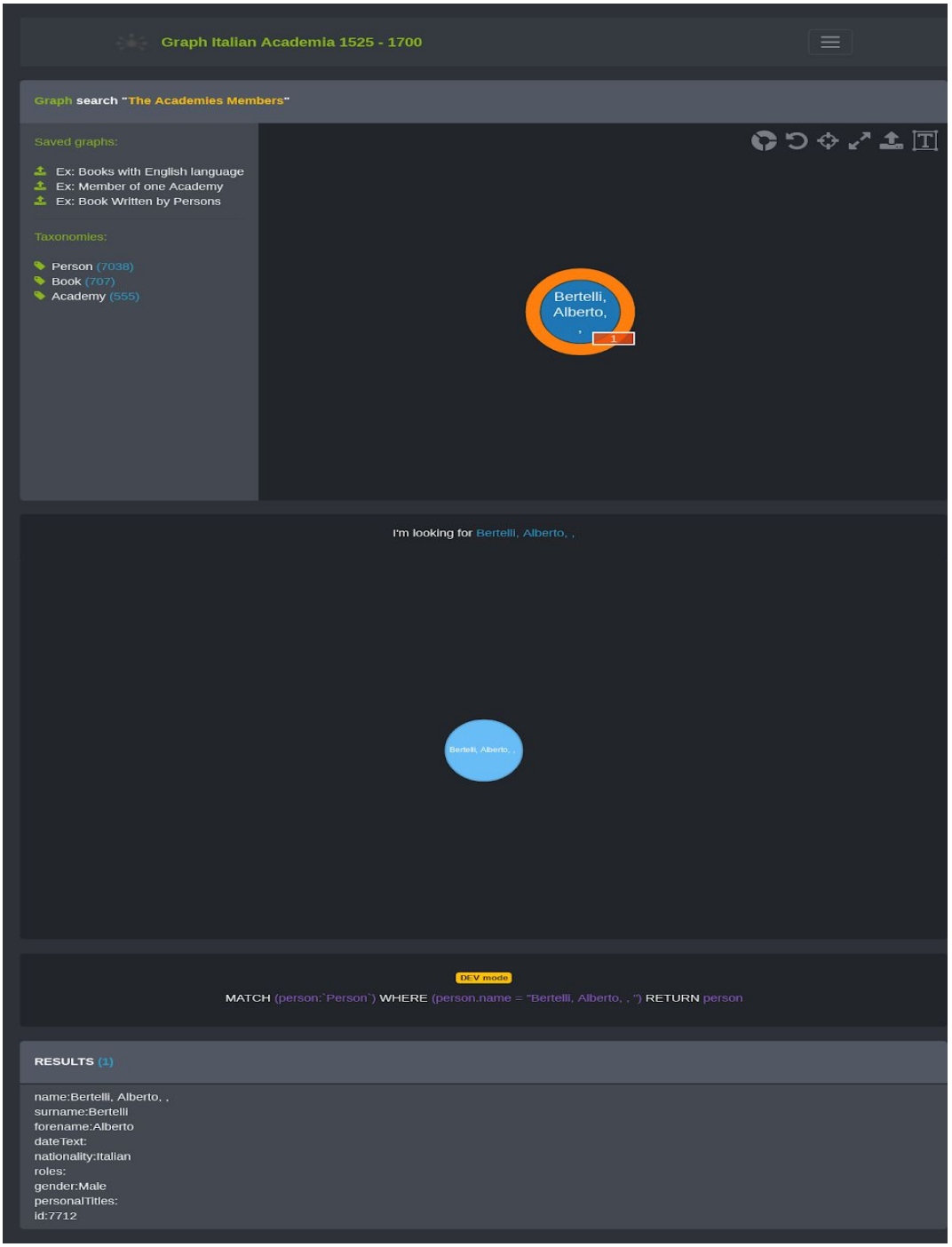


figura 3.4-3 SitoWeb

Conclusione e sviluppi futuri

Il termine accademia deriva dal greco e indicava la scuola filosofica di Platone, fondata nel 387 a.C. e situata in un luogo appena fuori le mura di Atene, chiamata così dal nome dell'eroe di guerra Academo che aveva donato agli ateniesi un terreno che divenne un giardino aperto al pubblico dove Platone filosofava con i suoi discepoli. Un'accademia è un'istituzione destinata agli studi più raffinati e all'approfondimento delle conoscenze di più alto livello. Il termine può indicare una società scientifica dedicata alla ricerca nel campo delle scienze naturali, della filosofia e delle belle arti. L'aura di prestigio associato all'origine del nome spinge molti istituti (soprattutto privati) a fregiarsi di questo appellativo, non sempre in maniera appropriata.

Abbiamo visto in questo lavoro, le potenzialità di una tecnologia emergente nel campo dei database: il Graph Database, il modo in cui questo nuovo modello di database può essere utile in alcuni domini di dati e perché possa essere preferito al modello relazionale vincente e consolidato.
Questo progetto è una versione preliminare, ma per il futuro è possibile aggiungere altre opzioni, per esempio un grafo che evidenzia le relazioni tra nodi in determinati periodi di tempo, o un'opzione per cercare un nodo speciale, anche si può utilizzare questo lavoro per altri database.
Aggiungerei il recupero dei dati che si sono persi e uno studio più approfondito sul tipo di query che sarebbero utili.

Bibliografia e sitografia

[1] “6. Key-Value Stores and Document Databases.” *Advanced Data Management*, doi:10.1515/9783110441413-010.

[2] Atzeni, Paolo. *Basi Di Dati: Modelli e Linguaggi Di Interrogazione*. McGraw-Hill, 2013.

[3] Batley, Sue. “Information Architecture: an Introduction.” *Information Architecture for Information Professionals*, 2007, pp. 1–11., doi:10.1016/b978-1-84334-232-8.50001-2.

[4] Breitman, K. K., et al. *Semantic Web Concepts, Technologies and Applications*. Springer, 2010.

[5] Di_Nunzio, Giorgio Maria. *Basi Di Dati: Manuale Di Esercizi per La Progettazione Concettuale v 2.0*. Esculapio, 2015.

[6] Duckett, Jon, et al. *JavaScript & JQuery: Interactive Front-End Web Development*. Wiley, 2015.

[7] Everson, J. E., et al. *The Italian Academies 1525-1700: Networks of Culture, Innovation and Dissent*. Legenda, 2016.

[8] Everson, Jane E. “The Italian Academies 1525-1700.” 2016, doi:10.4324/9781315559780.

[9] Guia, José, et al. “Graph Databases: Neo4j Analysis.” *Proceedings of the 19th International Conference on Enterprise Information Systems*, 2017, doi:10.5220/0006356003510356.

[10] Ian Robinson. Jim Webber. Emil Eifrem. *"Graph Databases, 2nd Edition"*. 2015.

[11] “Integration of Relational and NoSQL Databases.” *Bridging Relational and NoSQL Databases Advances in Data Mining and Database Management*, pp. 239–281., doi:10.4018/978-1-5225-3385-6.ch006.

[12] Jordan, Gregory. “Practical Neo4j.” 2014, doi:10.1007/978-1-4842-0022-3.

[13] Kemper, Chris. “Building an Application with Neo4j.” *Beginning Neo4j*, 2015, pp. 103–129., doi:10.1007/978-1-4842-1227-1_8.

[14] Kemper, Chris. “Querying Data in Neo4j with Cypher.” *Beginning Neo4j*, 2015, pp. 83–101., doi:10.1007/978-1-4842-1227-1_7.

[15] McFarland, David Sawyer. *JavaScript & JQuery: the Missing Manual: the Book That Should Have Been in the Box*. O'Reilly, 2014.

[16] "Modern JavaScript Programming." *Pro JavaScript™ Techniques*, pp. 3–16., doi:10.1007/978-1-4302-0283-7_1.

[17] Panzarino, Onofrio, and Jaroslaw Blaminsky. *Learning Cypher: Write Powerful and Efficient Queries for Neo4j with Cypher, Its Official Query Language*. Packt Publishing, 2014.

[18] Pasquini, Jacopo, and Simone Giomi. *Web Usability: Guida Completa Alla User Experience e All'usabilità per Comunicare e Vendere Online*. Hoepli, 2014.

[19] Pasquini, Jacopo, and Simone Giomi. *Web Usability: Guida Completa Alla User Experience e All'usabilità per Comunicare e Vendere Online*. Hoepli, 2014.

[20] Pearrow, Mark. *Web Usability*. Jackson Libri, 2001.

[21] Rosenfeld, Louis, et al. *Information Architecture for the Web and Beyond*. O'Reilly, 2015.

[22] "The Semantic Web." *Social Networks and the Semantic Web Semantic Web and Beyond*, pp. 3–26., doi:10.1007/978-0-387-71001-3_1.

[23] Stair, Ralph M., and George Walter Reynolds. *Fundamentals of Information Systems*. Cengage Learning, 2016.

[24] Testa, Simone. *Italian Academies and Their Networks: 1525-1700: from Local to Global*. Palgrave Macmillan, 2015.

[25] Zhang, Jie, and Robin Cohen. "A Trust Model for Sharing Ratings of Information Providers on the Semantic Web." *Semantic Web and Beyond Canadian Semantic Web*, pp. 45–61., doi:10.1007/978-0-387-34347-1_4.

[26] Spiegazione di test di usabilità. Disponibile a: https://www.techved.com/ux-design/ (visitato il 11 maggio 2018).

[27] Manuale Online di Neo4j v2.0M. Disponibile a: http://docs.neo4j.org/chunked/stable/ (visitato il 14 agosto 2018).

[28] NOSQL Databases. Disponibile a: http://www.nosql-database.org/ (visitato il 14 agosto 2018).

[29] Learning Cypher : Onofrio Panzarino. Disponibile a: https://neo4j.com/books/learning-cypher/ (visitato il 10 agosto 2018).

[30] Learning Neo4j 3 - Second Edition. Jérôme Baton, Rik Van Bruggen. Disponibile a: https://www.packtpub.com/big-data-and-business-intelligence/learning-neo4j-3x-second-edition (visitato il 17 maggio 2018).

[31] Information Architecture Institute, What is IA. Disponibile a: https://www.iainstitute.org/what-is-ia (visitato il 20 luglio 2018).

[32] Andrea Resmini, Luca Rosati, A Brief History of Information Architecture, Journal of Information Architecture. Disponibile a: http://journalofia.org/volume3/issue2/03-resmini/ (visitato il 12 luglio 2018).

[33] Graph Databases 2nd Edition. By Ian Robinson, Jim Webber, and Emil Eifrém. Disponibile a: (https://neo4j.com/graph-databases-book/ (visitato il 2 luglio 2018).

[34] Graph Databases: The New Way to Access Super Fast Social Data [online]. Disponibile a: http://mashable.com/2012/09/26/graph-databases/ (visitato il 2 agosto 2018).

[35] Release Notes: Neo4j 2.2.0 [online]. Neo Technology, Inc. [cit. 2015-04-13]. Disponibile a: http://neo4j.com/release-notes/neo4j-2-2-0/ (visitato il 1 maggio 2018).