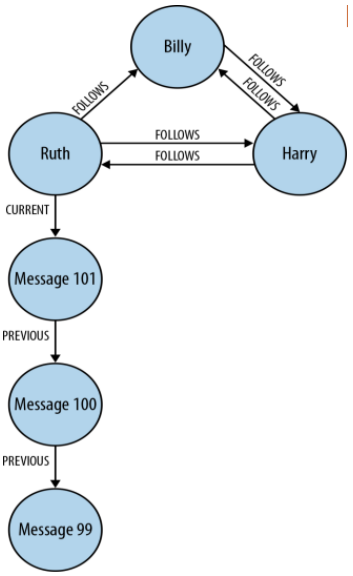
Lezione 6 03/04/2024

Graph DB

Ci interessa il grafo come modello di rappresentazione delle informazioni, perché considero i nodi come concetti, e gli archi sono le relazioni tra i concetti.

Anche il modello a grafo è un modello non relazionale, quindi rispetta il concetto di essere schemeless, quindi possiamo riutilizzare il nodo sia per rappresentare gli utenti che i messaggi.



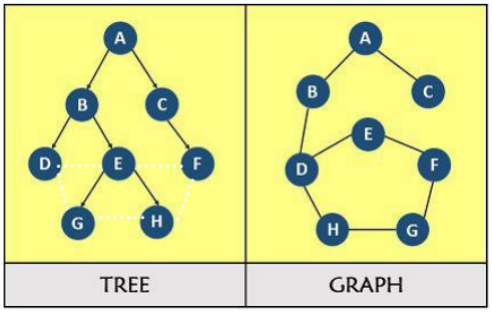
Bisogna fare una distinzione tra storage nativo e non nativo: il native graph storage è ottimato per la gestione dei dati salvati come grafi, mentre invece nel non-native graph storage i dati sono salvati in modelli non basati su grafi (relazionale, object oriented, wide column...), ma che supportano le query su grafi.

Non ci sono sistemi basati sui grafi che permettono la distribuzione dei dati, perchè non è efficiente, dato che il grafo è connesso e quindi un'interrogazione può aver bisogno di collegarsi ad altri nodi distribuiti. Infatti nel modello relazionale facevamo le divisioni di modo da mantenere le query il più locali possibili.

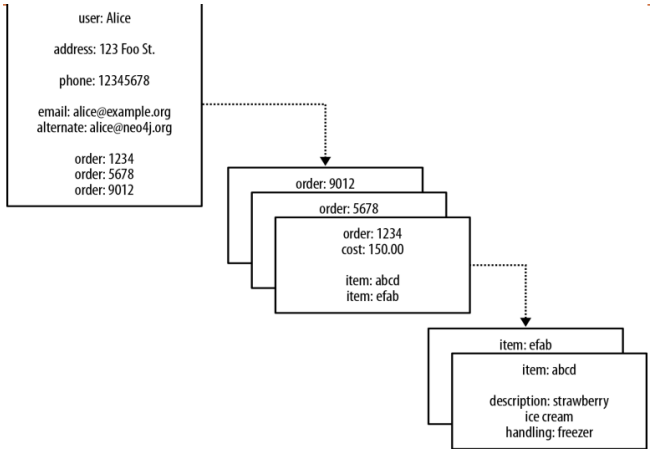
Il concetto di storage nativo viene chiamato **index free adjacency**. Vengono salvati vicini nel file i nodi che sono vicini semanticamente, per renderlo più efficiente. Grafi molto complicati ci mettono tanto tempo ad essere salvati ma poi sono molto efficienti.

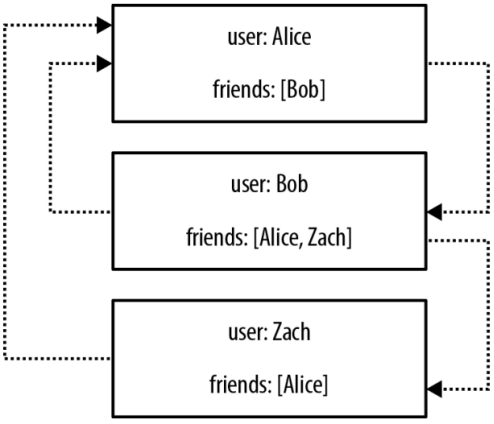
La differenza con un modello relazionale è che lì i collegamenti tra concetti sono fatti tramite i join, nel modello con documenti viene fatto tramite i sottodocumenti, mentre qui il collegamento tra concetti è fatto con gli archi.

Potrei modellare un grafo anche con un sistema a documenti, dal punto di vista teorico però non potrei rappresentare un grafo come un albero (sistema a documenti), dovrei anche fare referencing andando a tagliare il grafo e facendo referencing (puntatore) ad un altro nodo. Questo funziona bene quando ho chiaro l'use case, perché decido io in che punto andare a tagliare il grafo, perché so che il referencing non è efficiente. Per esempio mongodb può usare i grafi, con storage non nativo, e quindi diventa non efficiente per grafi complessi.

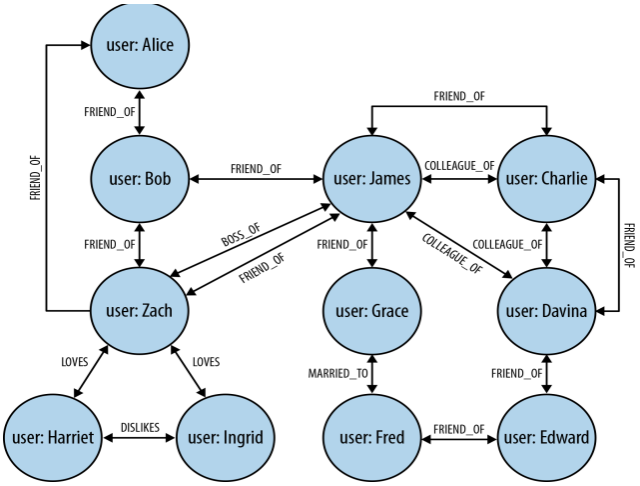


Per esempio in un use case dove ho un prodotto, i commenti, i commenti dei commenti etc, posso andare a tagliare questi grafi di modo da andare a mostrare per esempio i primi 10 commenti, mentre gli altri li posso recuperare nel caso che l'utente li richieda, andando a fare una query. Questo diventa un metodo più efficiente di usare i grafi.





Le frecce tratteggiate sono i referencing, che però non sono efficienti. Vanno bene se usate occasionalmente, non in modo sistematico.

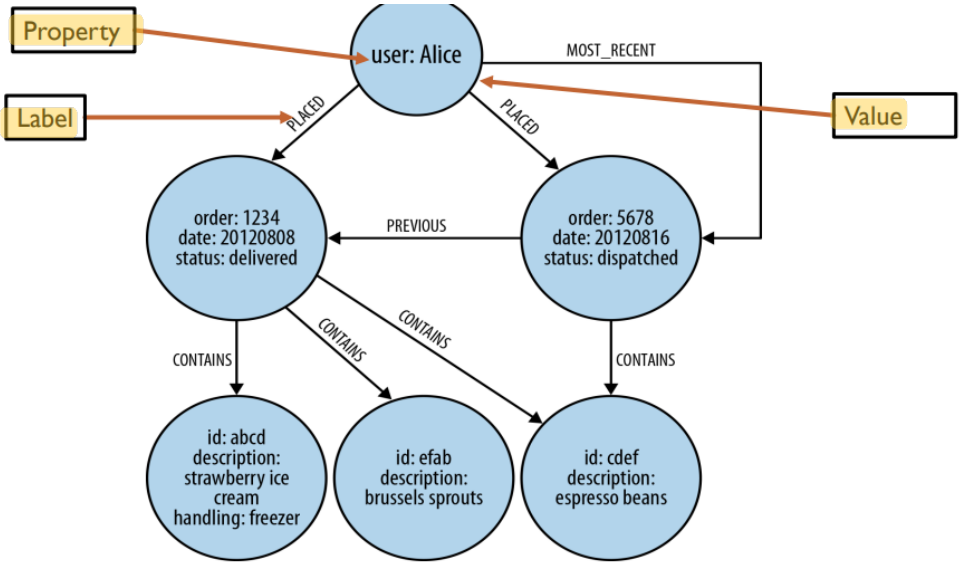


Come posso rappresentare queste relazioni “loves” “dislikes” etc? Posso andare semplicemente a mettere un 1 in una matrice. Questa è la potenza e la semplicità.

La relazione friend of a friend nel dbms relazionale si ferma a livello 5 perchè ci mette troppo, con il metodo a grafi invece funziona e ci mette qualche minuto.

Proprietà

Posso dare per esempio le etichette sugli archi. Posso andare a rappresentare cose complicate con coppie proprietà valore.



è improprio parlare di tipizzazione, user non è un tipo, è un’etichetta.

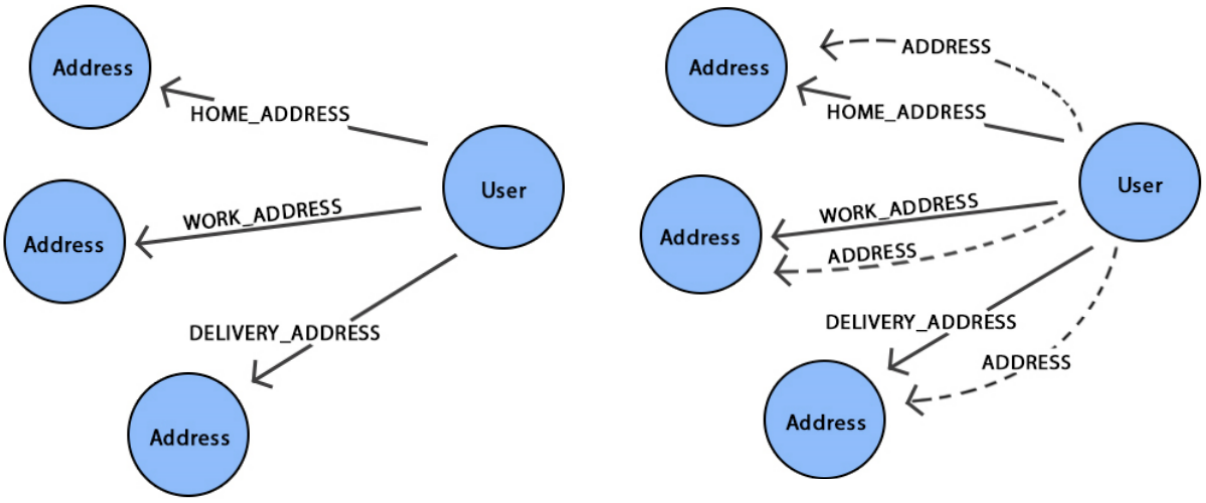
Per esempio devo modellare questo problema dove l’utente ha più indirizzi.

DELIVERY_ADDRESS

VS

ADDRESS{type : 'delivery'}

Qui posso usare un'etichetta diversa, ovvero DELIVERY_ADDRESS, oppure posso usare il secondo metodo.



I due casi sono entrambi corretti, dipende dalla situazione.