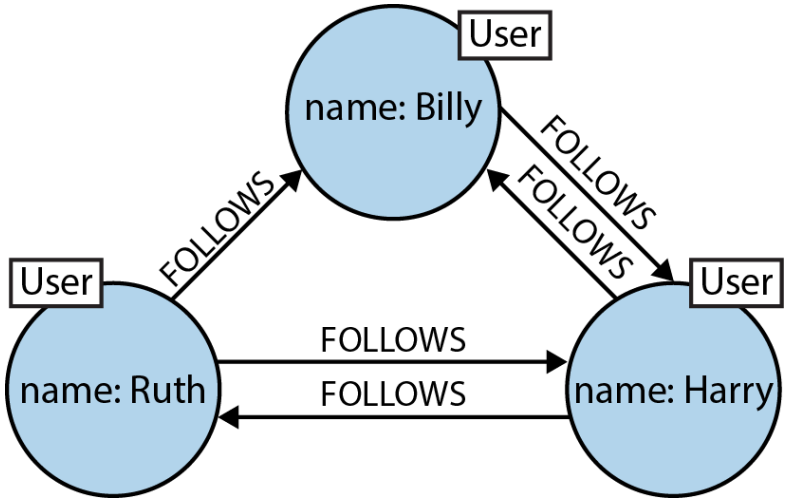
NoSql DBs
Graph Model DBs

P. Rullo

Modello dei dati

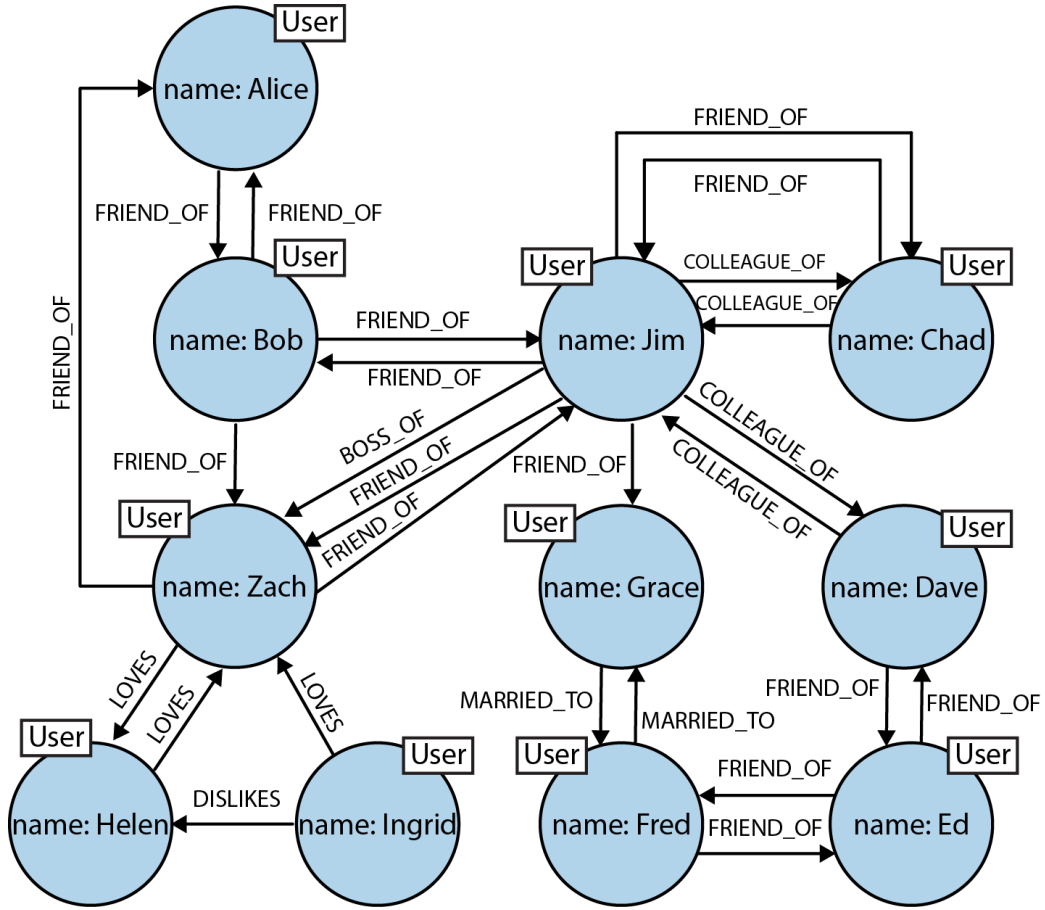
- Un *graph DB* (GDB) è un insieme di dati organizzati secondo una struttura logica a grafo.
- Più precisamente, un GDB è un grafo orientato, cioè, una coppia <N,A>, dove N è un insieme di nodi (o vertici) e A è un insieme di archi orientati ed etichettati, cioè, triple del tipo <n,m,e>, dove n e m sono nodi ed e è una etichetta.
- I nodi rappresentano oggetti del mondo reale, e gli archi relazione tra i nodi. Un grafo descrive quindi come gli oggetti si relazionano tra di loro.

Modello dei dati



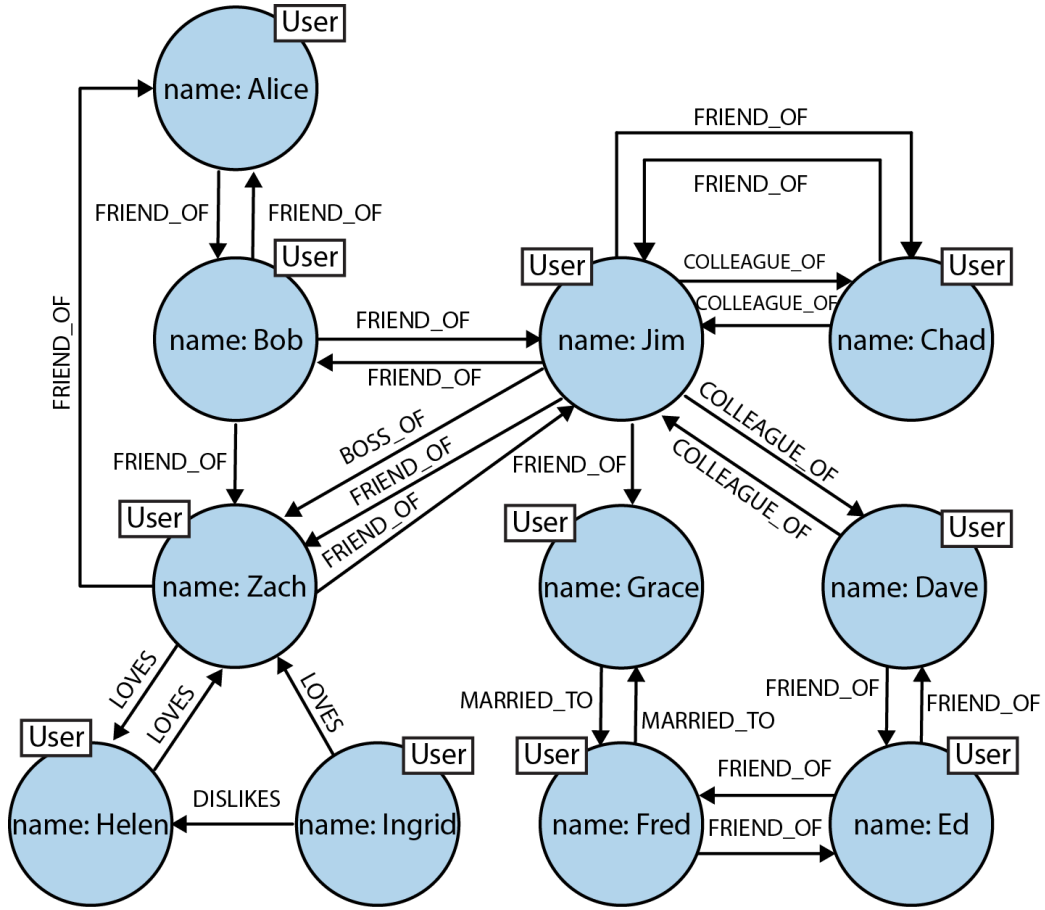
- Il grafo rappresenta un frammento di rete Twitter
- Ogni nodo rappresenta uno specifico utente, e gli archi rappresentano la relazione “follows”.

Modello dei dati



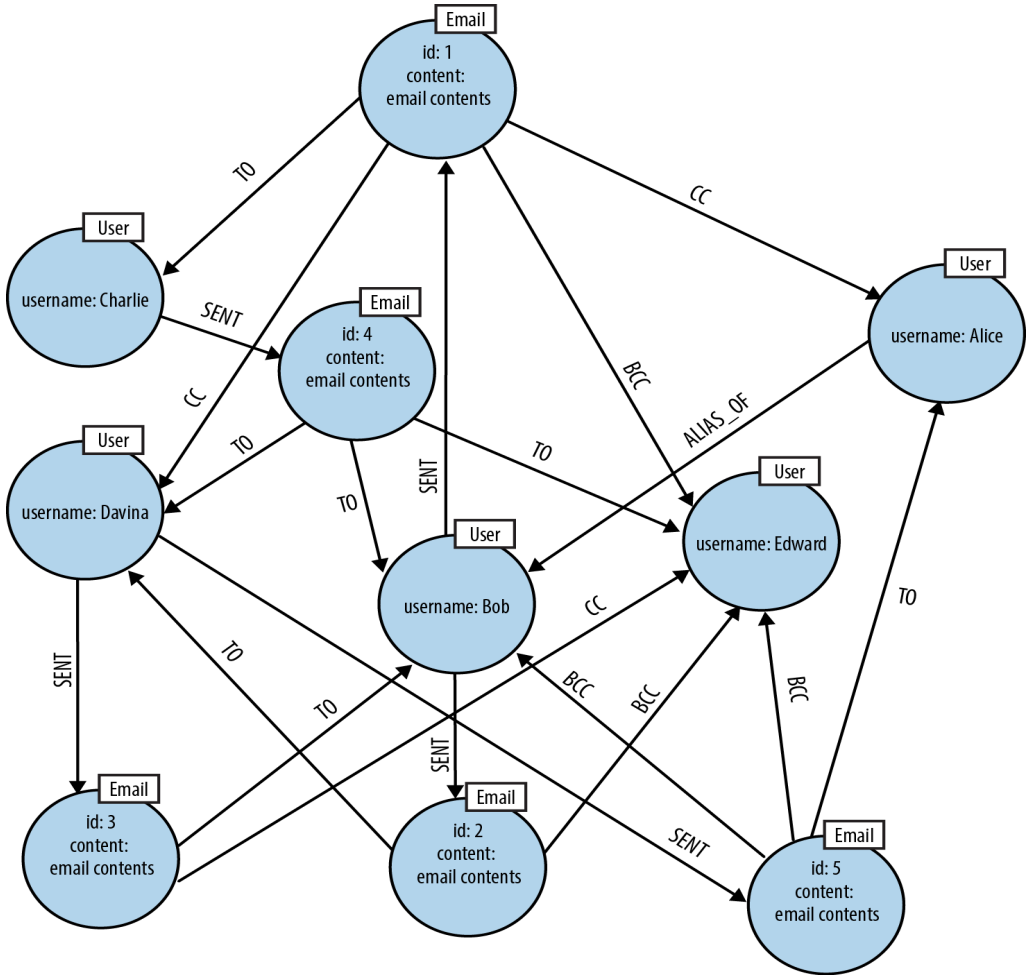
- Il grafo descrive un insieme di utenti di una rete sociale ed i loro rapporti di amicizia, colleganza, amore, ecc.
- Le relazioni tra le entità non presentano uniformità in tutto il dominio - la rete è strutturata in modo variabile, in quanto non tutti i nodi sono collegati allo stesso modo agli altri nodi

Modello dei dati



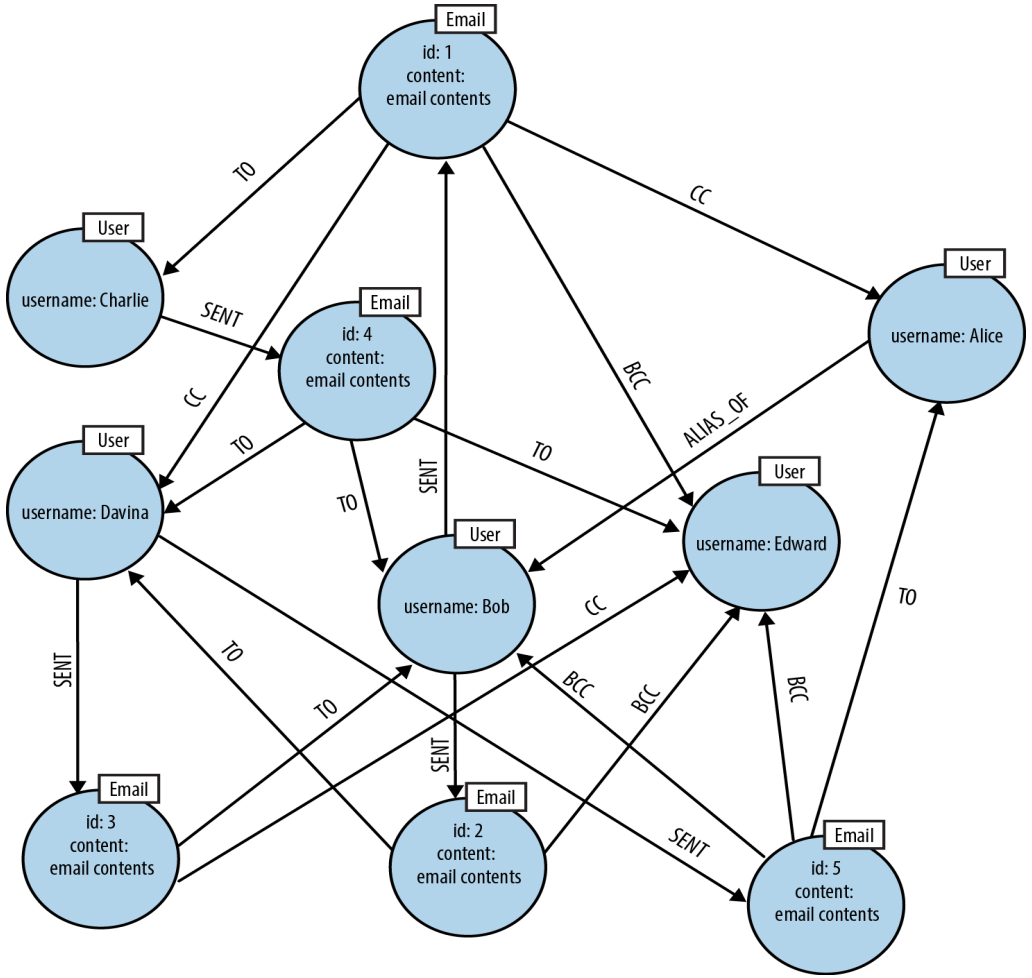
- Quindi non è possibile in linea di principio definire uno *schema fisso* che vale per tutti i “pezzi” della rete.
- La flessibilità del modello a grafo permette di aggiungere nuovi nodi e nuove relazioni senza compromettere la rete esistente

Modello dei dati



- Il grafo descrive un flusso di email
- Due tipi di nodi: User (username) e Email, connessi attraverso le seguenti relazioni:
- user-email
  - SENT: per specificare il mittente di una email
  - TO: per specificare i destinatari diretti di una email
  - CC e BCC: per specificare i destinatari in copia carbone
- user-user: ALIAS-OF per specificare che un certo indirizzo utente è un alias di un altro

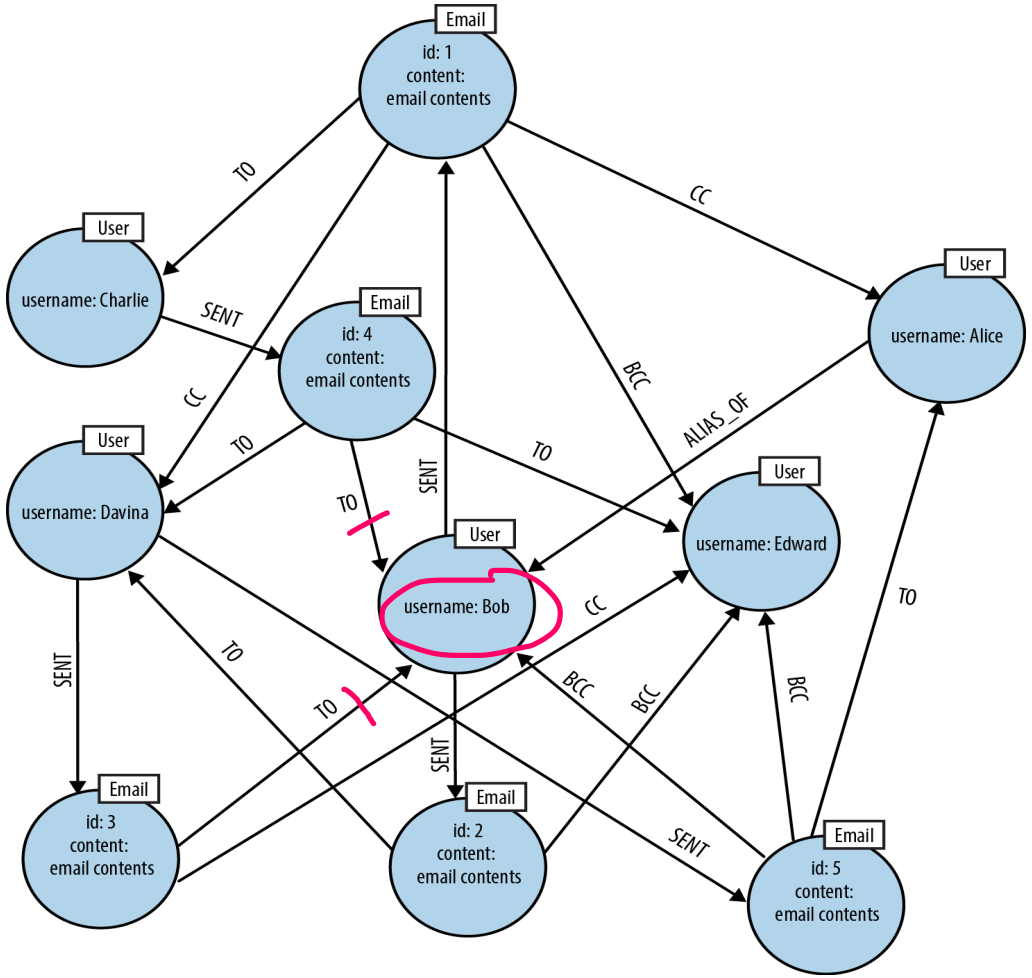
Creazione e interrogazione di un GDB: il linguaggio Cypher



- **CREATE** (alice:User {username:'Alice'}),
  - (bob:User {username:'Bob'}),
  - (charlie:User {username:'Charlie'}),
  - (davina:User {username:'Davina'}),
  - (edward:User {username:'Edward'}),
  - (alice)-[:ALIAS_OF]->(bob)

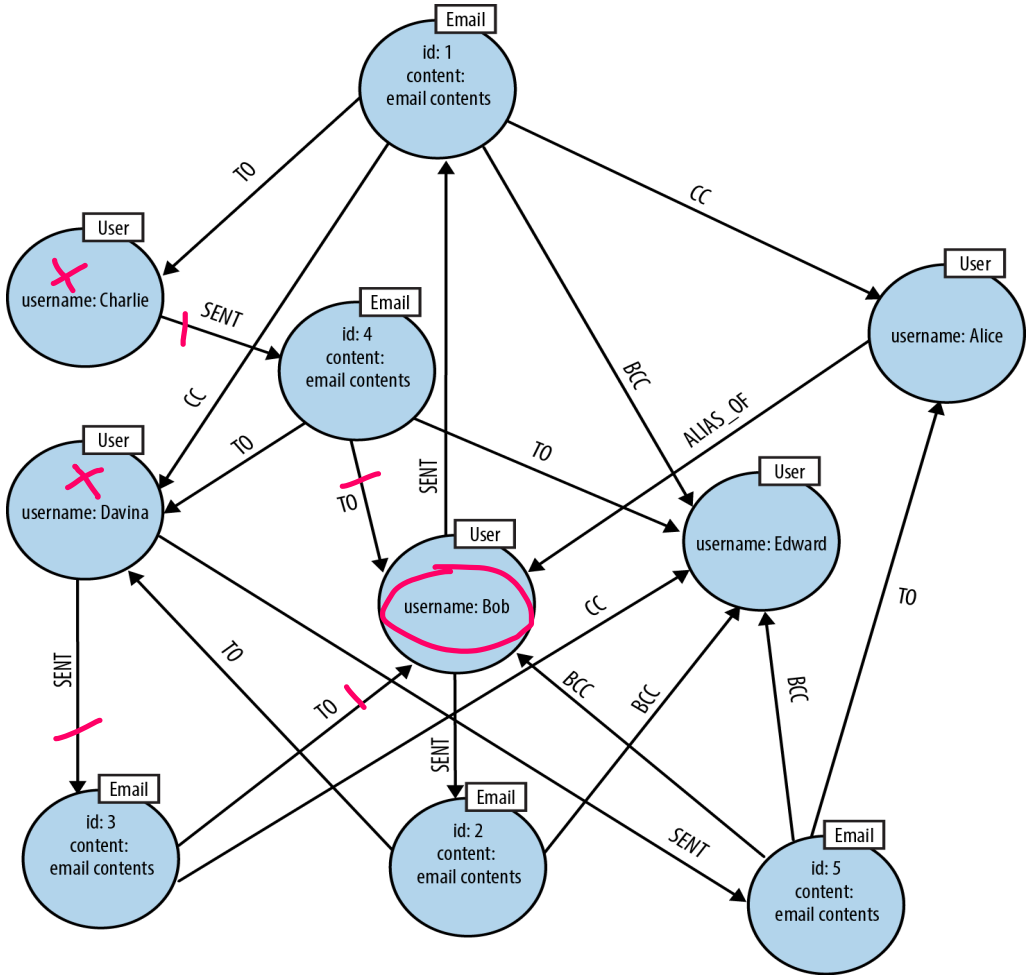
- **CREATE** (email_1: email {id:'1', content:'email1 contents'}),
  - (bob)-[:SENT]->(email_1),
  - (email_1)-[:TO]->(charlie),
  - (email_1)-[:CC]->(davina),
  - (email_1)-[:CC]->(alice),
  - (email_1)-[:BCC]->(edward);
- **CREATE** ...

Creazione e interrogazione di un GDB: il linguaggio Cypher



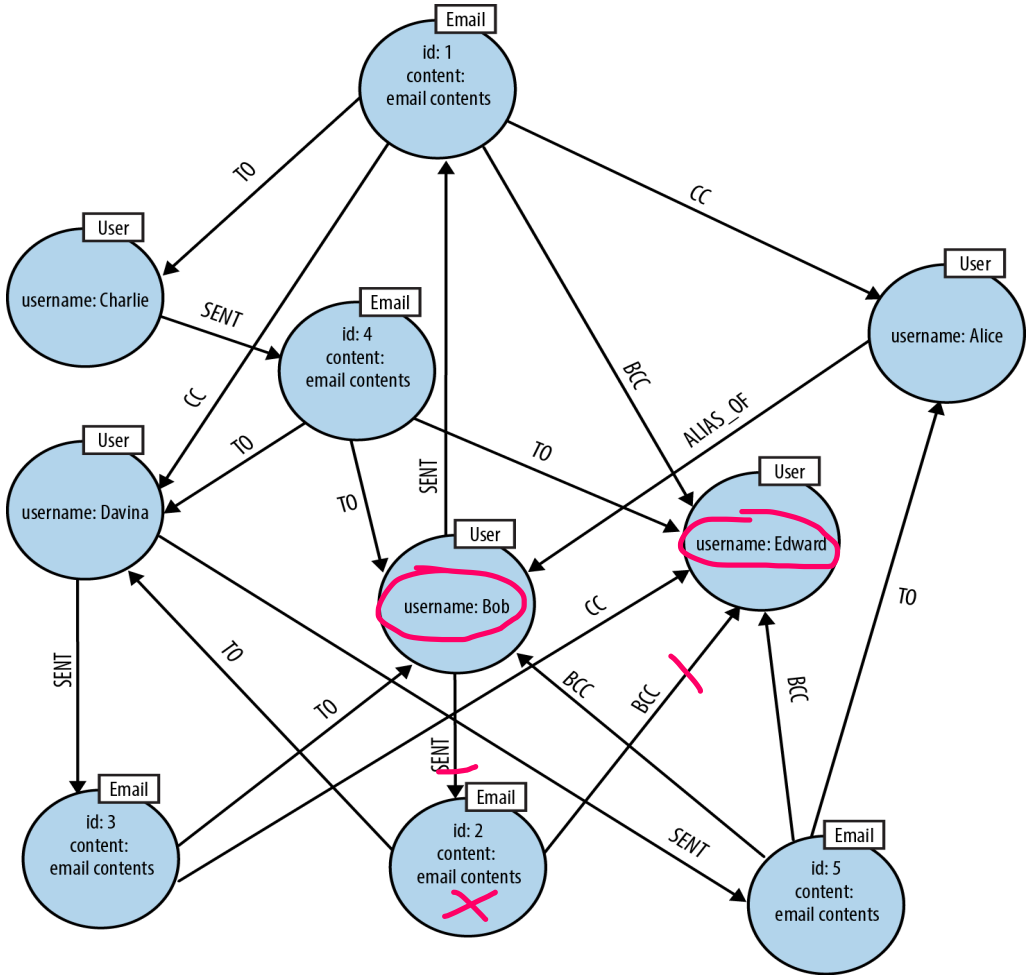
- Email ricevute da bob come diretto destinatario
  - **MATCH** (x: email) - [:TO] ->(a: User)
  - **WHERE** a.username = Bob
  - **RETURN** x.id, x.content
- In questo esempio abbiamo usato 3 clausole fondamentali di Cypher:
  - **Match**: consente di definire un percorso sul grafo. In particolare, (email) - [:TO] -> (a: User) descrive il percorso che dalla generaica email "email" porta all'utente *a* con username "Bob" attraverso la relazione TO
  - **Where**: specifica che il nodo utente è quello con username "Bob"
  - **Return**: specifica i dati restituiti dalla query - equivale alla Select di SQL.

Creazione e interrogazione di un GDB: il linguaggio Cypher



- Mittenti delle email ricevute da bob
  - **MATCH** (a: User) <- [:TO] - (:email) <- [:SENT] – (x:User)
  - **WHERE** a.username = Bob
  - **RETURN** x.username

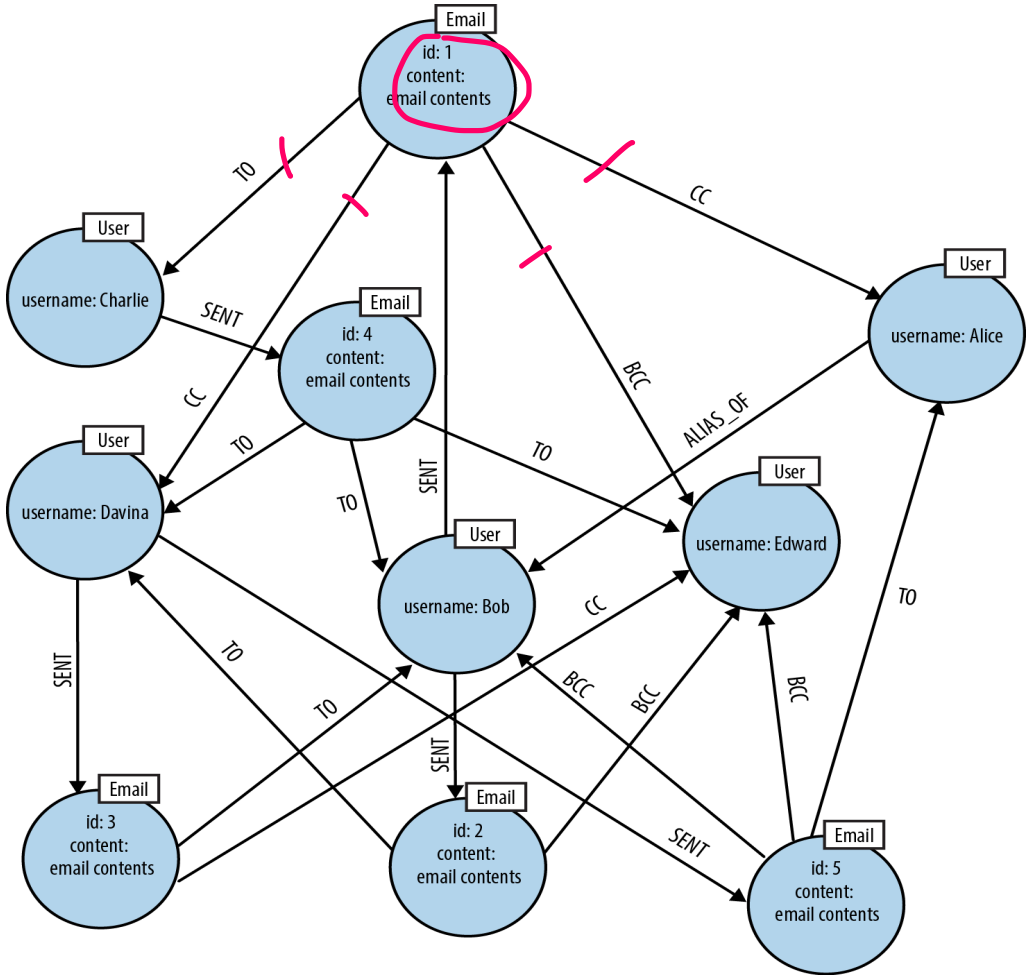
Creazione e interrogazione di un GDB: il linguaggio Cypher



Email inviate da Bob in cui Edward appare in BBC

- **MATCH** (a: user{username: “Bob”})-[:SENT]-> (x:email)-[:BBC]->(b: User{username: “edward”})
- **RETURN** x.id, x.content

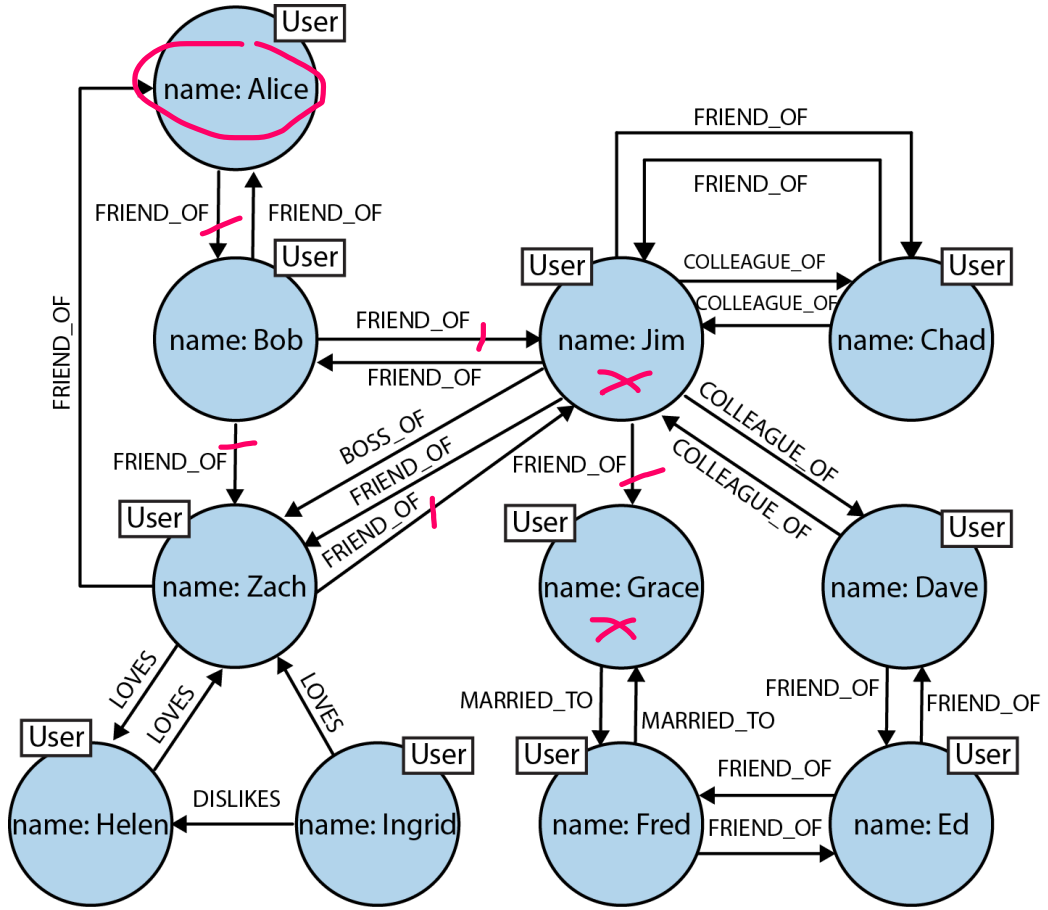
Creazione e interrogazione di un GDB: il linguaggio Cypher



Numero complessivo di destinatari della email id=1

- **MATCH** (a: email) - [:TO] -> (d1: user) , (a) - [:CC] –> (d2: user), (a) - [:BCC] –> (d3:user)
- **WHERE** a.id = “1”
- **RETURN** count(d1)+count(d2)+count(d3)

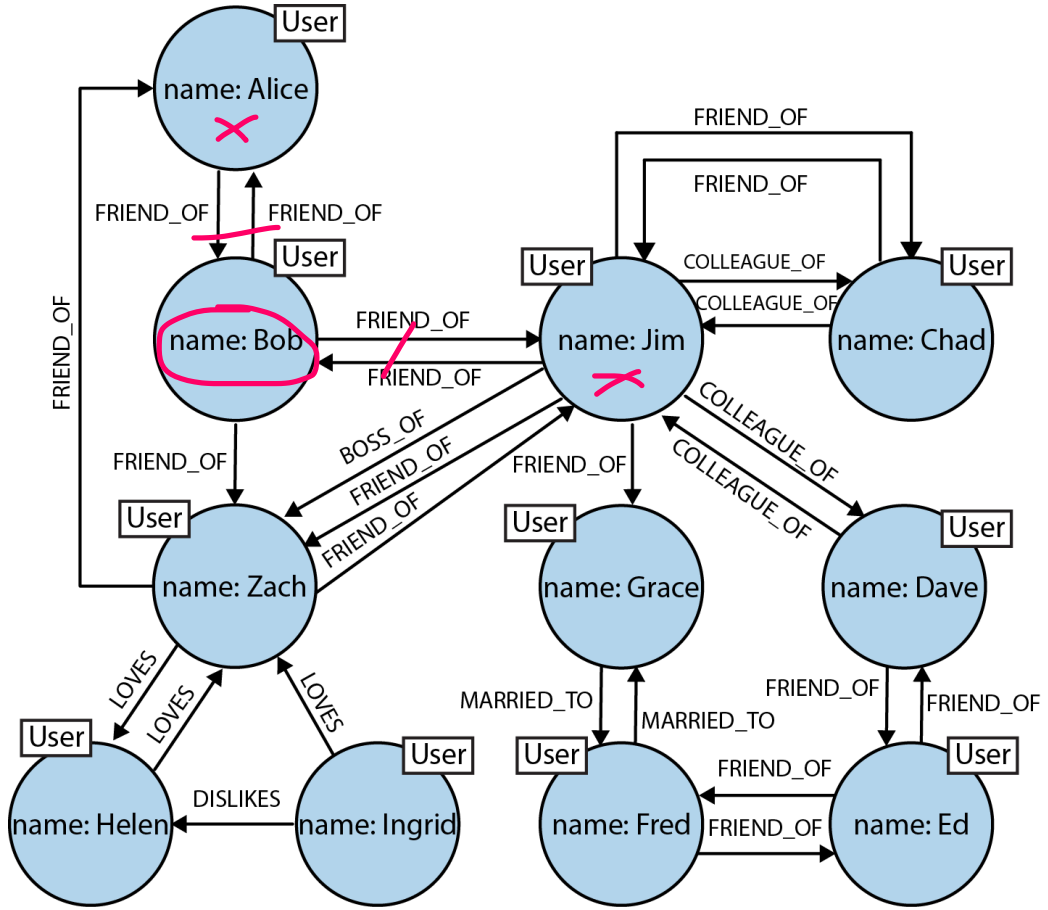
Creazione e interrogazione di un GDB: il linguaggio Cypher



Amici di livello 3 di alice

- **MATCH** (a: user)-[:FRIEND-OF]-> (b:user)-[: FRIEND-OF]->(c:user))-[: FRIEND-OF]->(d:user)
- **WHERE** a.username = “Alice”
- **RETURN** d.name

Creazione e interrogazione di un GDB: il linguaggio Cypher



Amici di Bob (di primo livello) che ricambiano l'amicizia

- **MATCH** (a: user)-[:FRIEND-OF]-> (b:user)-[: FRIEND-OF]->(a)
- **WHERE** a.username = "Bob"
- **RETURN** b.name

Amici di bob che NON ricambiano l'amicizia

- **MATCH** (a: user)-[:FRIEND-OF]-> (b:user)
- **WHERE** a.username = "Bob" and NOT ((b) -[: FRIEND-OF]->(a))
- **RETURN** b.name

Memorizzazione di grafi ed efficienza di un GDB

- Alcuni GDB usano strutture di memorizzazione native, cioè, ottimizzate per memorizzare e manipolare grafi
- Neo4J è uno di questi
- Altri GDB, invece, mappano i grafi nel modello relazionale. Ad esempio, usando il modello relazionale, la relazione FRIEND_OF (di tipo molti-a-molti) dell'esempio 2, viene rappresentata come segue:

  - **User(<u>username</u>, …)**
  - **FriendOf(user1, user2)**

Memorizzazione di grafi ed efficienza di un GDB

- E’ evidente che una tale rappresentazione penalizza molte interrogazioni dal punto di vista dell’efficienza.
- Ad esempio, per calcolare gli amici di livello n di un dato utente (vedi interrogazione 2.1, con n=3), con una rappresentazione relazionale è necessario eseguire n-1 join della relazione FriendOf con sé stessa.
- Considerato che la cardinalità di tale relazione è, in un caso reale, molto elevata (molti milioni di tuple), è facile immaginare che l’interrogazione avrà tempi di risposta molto elevati.

Memorizzazione di grafi ed efficienza di un GDB

Profondità	Tempo RDB (sec)	Tempo GDB (sec)	Nodi restituiti
2	0.016	0.01	2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Ricerca non terminata	2.132	~800,000

- Risultati della ricerca, fino ad una profondità massima di cinque, ottenuti da un GDB con *native graph storage* (in particolare, Neo4j) e un DB relazionale, per un social network contenente 1.000.000 di persone, ciascuno con circa 50 amici.