CS 5/7330

Graph Databases

Graphs

- Graphs
  - Vertices and edges
  - Vertices (may) have (multiple) labels/attributes
  - An edge connects a pair of vertices
    - Can be directed (or not)
    - Can have (multiple) labels/attributes
    - May have multiple edges between any pairs of vertices (not common, but can be useful)
- Graphs can be used to model:
  - Relationships
  - Maps
  - Networks

Graphs

Example:

From: Eifrem, Robinson, Webber, "Graph Databases", 2nd edition

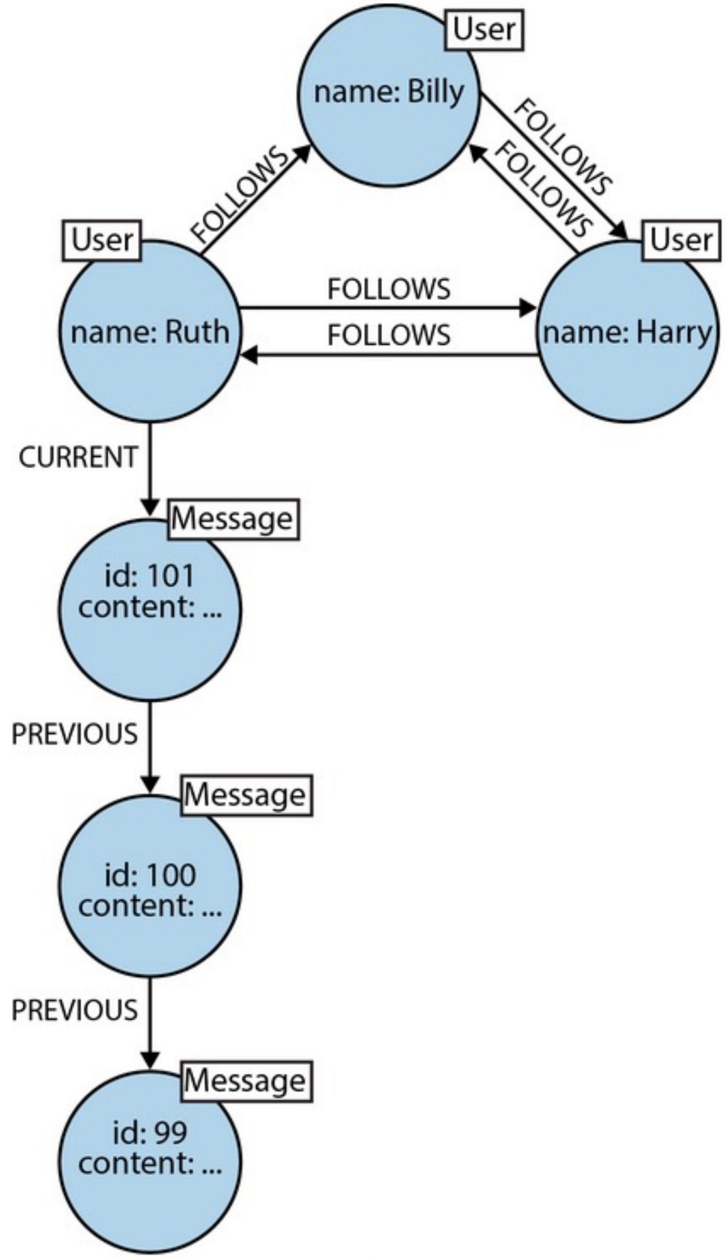


Figure 1-2. Publishing messages

Graphs

Example:

From: Eifrem, Robinson, Webber, "Graph Databases", 2nd edition

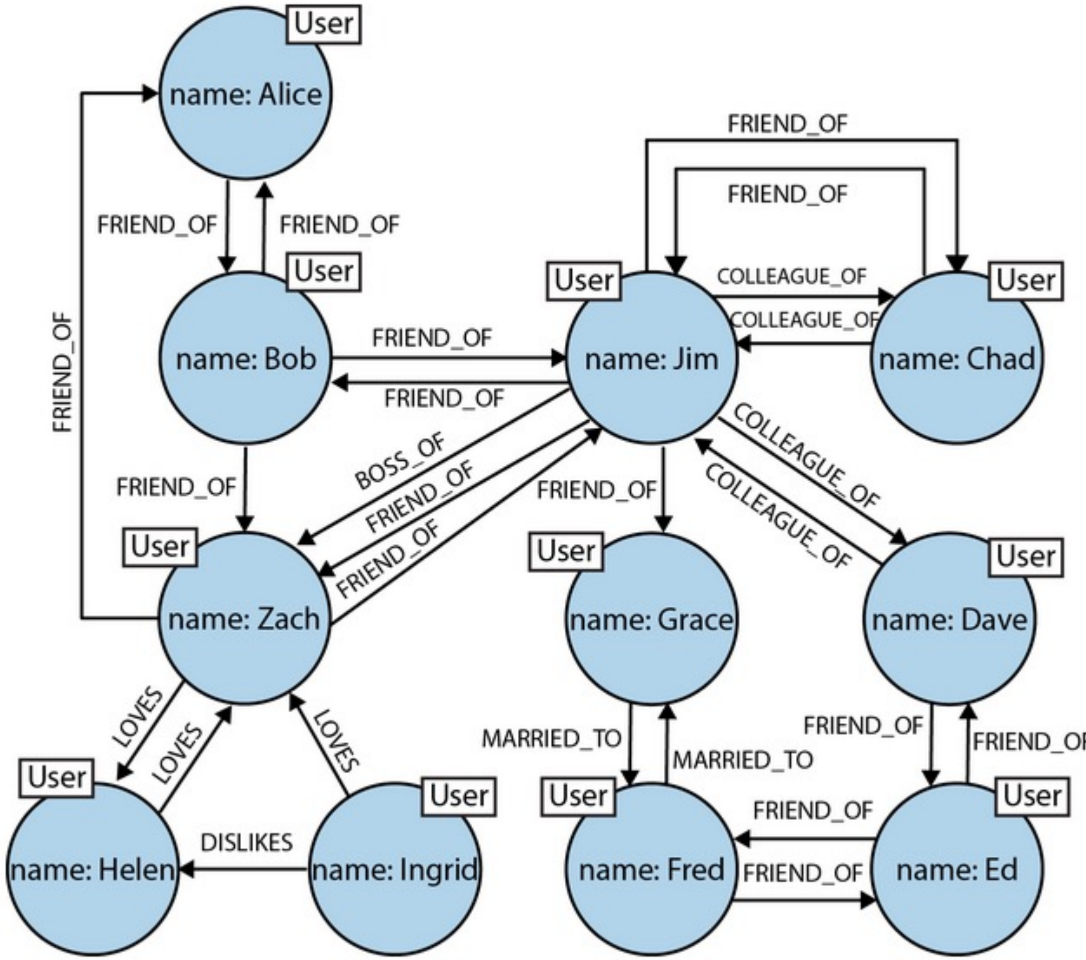


Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

Different types of graphs

Table 1. Various Graph Structures

Graph structure	Description
Undirected/directed graphs	All relationships in an undirected graph are symmetric. On the other hand, in a directed graph, each edge *e* from vertex *src* to vertex *des* is a directed tuple, such that (1) edge *e* is an *in-edge* of *des* and an *out-edge* of *src*, (2) vertex *des* is the *src*'s *out-neighbor* and vertex *src* is the *in-neighbor* of *des*, and (3) the number of *incoming/outgoing* edges of a vertex determines its *(in/out)* degree.
Labeled graphs	Vertices and edges are tagged with scalar values (labels or types) that may represent either their roles in different application domains or some attached metadata.
Attributed graphs	A variable list of attributes as (*key*, *value*) pairs are attached to vertices and edges, representing their properties. It is suitable for social networking sites that involve social interaction of individuals.
Multigraphs	Multiple edges (even with the same labels) between the same two vertices as well as self-loops are allowed.
Hypergraphs	These graphs can represent N-ary relationships through hyperedges that can connect any number of vertices (Bretto 2013). An undirected hyperedge can be represented by a subset of nodes (vertices) (Berge 1973); and a directed hyperedge can be represented by a tuple (ordered set) of nodes (Boley 1992) or head-and-tail sets (Gallo et al. 1993; Nguyen and Pallottino 1989). Hypergraphs are used to represent complex relationships in areas such as Artificial Intelligence. HyperGraphDB (Iordanov 2010) is based on hypergraphs and supports N-ary relationships by representing a link as a tuple (ordered set) of nodes. Figure 4 shows a simple hypergraph.
Nested graphs	Each vertex, in turn, can be a graph. At this time, no store supports nested graphs.

Typical Operations for graph databases

- On-line graph navigations:
  - Explore a (typically small) subset of vertices
  - Require fast response time
- Two types
  - Path query
    - Whether a path exists between two given vertices
    - Conditions on the edges along the path
    - Can also be given only one vertex and find all the other vertices that can be reached/can reach it from the query vertex

Typical Operations for graph databases

- On-line graph navigations:
  - Explore a (typically small) subset of vertices
  - Require fast response time
- Two types
  - Pattern matching query
    - Find all subgraphs that is isomorphic to the query subgraph
    - General graph isomorphism is NP-complete
    - Tree isomorphism has polynomial time solutions

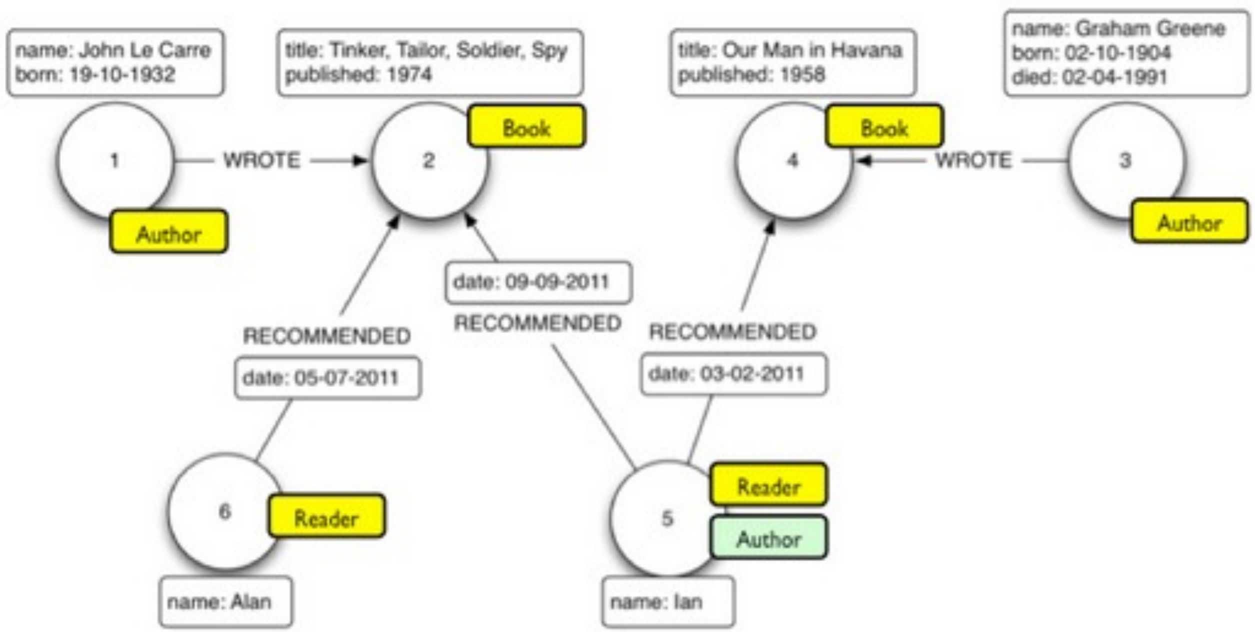
Typical Operations for graph databases

- Off-line analytical graph computation
  - Analyze large (number of) graphs
  - Typically iterative computation (not map-reduce)
  - Require highly tuned graph packages
  - Aim for high throughput, not low latency

Example : Neo4j

- Based on the Labelled Property Graph Model
- 4 major component of a graph
  - Nodes – generally for entity
  - Relationships (edges) – generally for relationship between entities
  - Properties – key/value pairs that is associated with nodes and/or relationships
    - Like document model, no need for fixed schema
  - Labels: Assigned to nodes (notice that, just labels, not key/value pairs)

Example : Neo4j



The labeled property graph model

From, Jerome Baton, “Learning Neo4j 3.x”, 2nd edition

Neo4j Data Model

- Work best for binary relationship between entities
- May need extra node for n-ary relationships
- Challenge of whether making something a property or a node
  - Make graphs simpler vs. query speed
  - Depends on whether it is faster to traverse a graph vs. search node/edge based on properties

Neo4j Data Model

- General mapping from E-R model to Graph Model
  - Entity -> Nodes (with labels serve as names)
  - (Binary) Relationship -> Edges
  - Attributes -> Properties

Cypher

- Common database language for graph databases
- Based on patterns and pattern matching
- Basic ideas:
    - Database : a set of nodes and edges
    - Commands to create those nodes and edges
    - Queries: specifying a pattern of edge/node/paths/subgraphs and try to match it.

Cypher

Creating a node:

- CREATE (u: User {username: “greg”, age: “33”})
  - Return a User (label) node (u) that has a property (name/”greg”, age/33)
  - The variable u is used as a reference to the node
  - You can create multiple nodes (separate each node with a comma) in a statement
- CREATE (u: User:Admin {username: “greg”, age: “33”})
  - A node can have multiple labels
  - This node has two labels, User, Admin.
    - No notion of sub-labels.

Cypher

Finding a node

- MATCH (a:User), (b:System)

  WHERE a.username=“greg” AND b.sysname = “UNIX”

  RETURN a, b
- Notice that if multiple nodes matches they will all be returned (like a Cartesian Product)

Projections

- MATCH (a:User {name: “John Doe”})”

  RETURN a.age, a.gender

Cypher

Patterns used for queries

- (a)
  - matching any node.
  - a is the variable name that use to refer to the node in the rest of the query
- (a) - - > (b)
  - Match any pair of nodes that go from a to b
- (a) < - - (b)
  - Match any pair of nodes that go from b to a
- (a) - - > (b) - - > (c)
  - Match three nodes that form a path from a to b to c

Cypher

Patterns used for queries

- (a: User)
  - matching any node with the label user.
  - a is the variable name that use to refer to the node in the rest of the query
- (a: User:Admin)
  - Matching nodes with BOTH labels
- (a:User|Admin)
  - Matching nodes with either label

Cypher

Patterns used for queries

- (a: {name:"John", age:25)
  - matching any node with the following properties.
- (a) -> [{length : 30}]-> (b)
  - Matching a path where the edge has the property specified

Cypher

Building an edge

- MATCH (a:User), (b:System)

  WHERE a.username="greg" AND b.sysname = "UNIX"

  CREATE (a)-[pr:LOGIN]->(b)

  SET pr.date ="11/11/2020"

  RETURN pr

  - Create an edge from node a to node be, with the label "LOGIN" and key/value pair (date/"11/11/2020")

Cypher

Constraints:

- Uniqueness Constraint:
  - CREATE CONSTRAINT ON (a:USER) ASSERT a.username IS UNIQUE
  - Ensure uniqueness of username for all USER nodes
- Property Existence Constraint:
  - CREATE CONSTRAINT ON (a.User) ASSERT exists(a.username)
  - Ensure every node of User have username property