NoSQL and Graph Databases: Principles

Agenda

- Graph Databases: Mission, Data, Example
- A Bit of Graph Theory
  - Graph Representations
  - Improving Data Locality (efficient storage)
  - Graph Partitioning and Traversal Algorithm
  - Types of Queries
- Graph Databases
- Neo4j
  - Data model
  - Traversal of the graph
  - Cypher query language

RDBMS recap

- RDBMS are predominant database technologies
  - Since 1970

- Data modeled as relations (tables)
  - object = tuple of attribute values
  - tables contain objects of the same type
  - tables interconnected via foreign keys

- Use SQL query language

Advantages of Relational Databases

- A (mostly) standard data model
- Many well developed technologies
  - physical organization of the data
  - search indexes: B^+-Trees, hash indexes
  - query optimization, search operator implementations
- Reliable concurrency control (ACID)
  - transactions: atomicity, consistency, isolation, durability
- Many reliable integration mechanisms
  - “shared database integration” of applications

NoSQL Databases

- What is "NoSQL"?
  - term used in late 90s for a different type of technology: Carlo Strozzi: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/
  - "Not Only SQL"?
    - but many RDBMS are also "not just SQL"

"NoSQL is an accidental term with no precise definition"

  - first used at an informal meetup in 2009 in San Francisco (presentations from Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB, and MongoDB)

[Sadalage & Fowler: NoSQL Distilled, 2012]

NoSQL Databases (cont.)

- NoSQL: Database technologies that are (mostly):
  - Not using the relational model (nor the SQL language)
  - Designed to run on large clusters (horizontally scalable)
  - No schema - fields can be freely added to any record
  - Open source
  - Based on the needs of 21st century web estates

[Sadalage & Fowler: NoSQL Distilled, 2012]

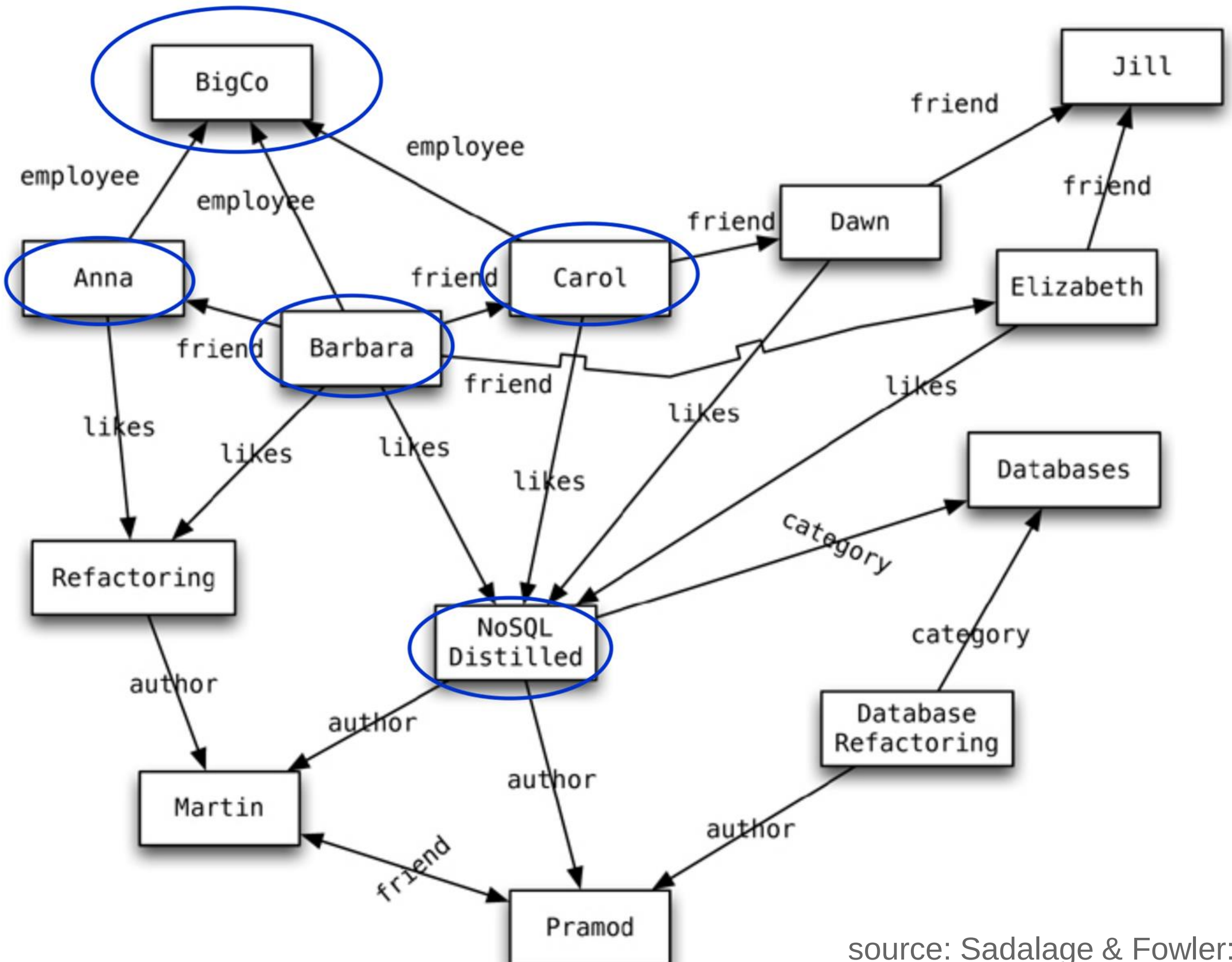
- Other characteristics (often true):
  - easy replication support (fault-tolerance, query efficiency)
  - simple API
  - eventually consistent (not ACID)

Four Basic Types of NoSQL Databases

- Key-value stores
- Document databases
- Column-family stores
- Graph databases

In this course we will discuss only graph databases in details

Graph Databases: Example



source: Sadalage & Fowler: NoSQL Distilled, 2012

Graph Databases: Mission

- To store entities and relationships between them
  - Nodes are instances of objects
  - Nodes have properties, e.g., name
  - Edges connect nodes and have directional significance
  - Edges have types e.g., likes, friend, …

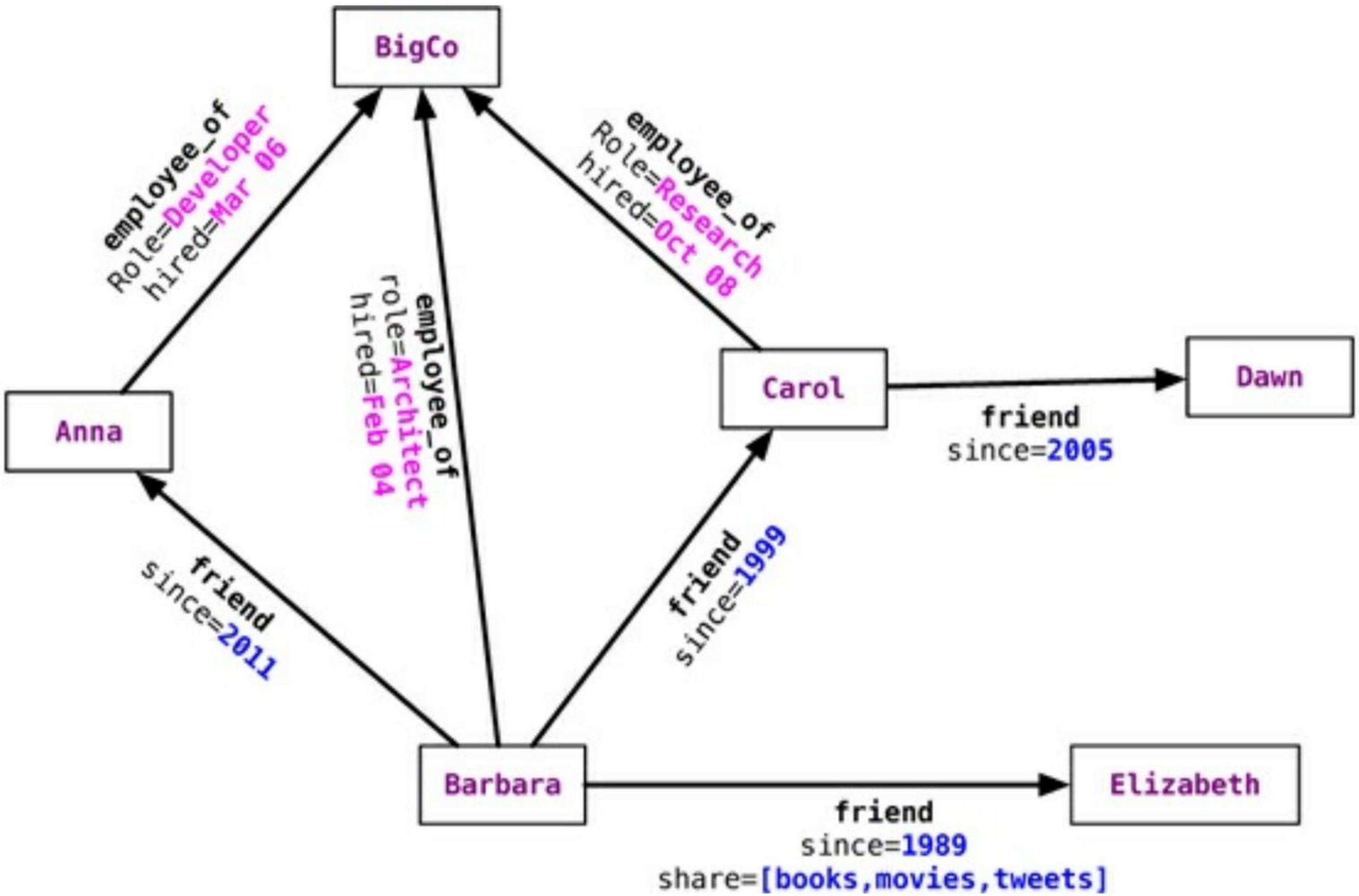
- Nodes are organized by relationships
  - Allow to find interesting patterns
  - **example:** Get all nodes that are “employee” of “Big Company” and that “likes” “NoSQL Distilled”

Basic Characteristics

- Different types of relationships between nodes
  - To represent relationships between domain entities
  - Or to model any kind of secondary relationships
    - Category, path, time-trees..
- No limit to the number and kind of relationships

- Relationships have: type, start node, end node, own properties
  - e.g., “since when” did they become friends

Relationship Properties: Example



source: Sadalage & Fowler: NoSQL Distilled, 2012

A Bit of a Theory

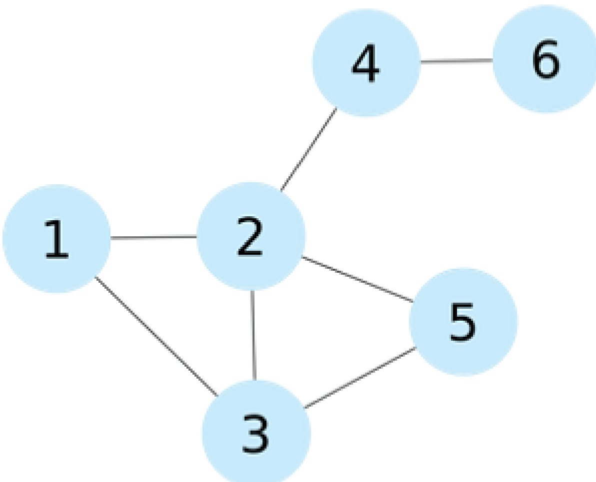
- Data: a set of entities and their relationships
  - => we need to efficiently represent graphs
- Basic operations:
    - finding the neighbours of a node,
    - checking if two nodes are connected by an edge,
    - updating the graph structure, …
  - => we need efficient graph operations
- graph $G = (V, E)$ is commonly modelled as
  - set of nodes (vertices) V
  - set of edges E
  - $n = |V|$, $m = |E|$
- Which data structure to use?

Data Structure: Adjacency Matrix

- Two-dimensional array A of $n \times n$ Boolean values
  - Indexes of the array = node identifiers of the graph
  - Boolean value A_{ij} indicates whether nodes i, j are connected

- Variants:
  - (Un)directed graphs
  - Weighted graphs…

Adjacency Matrix: Example



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

- Pros:
  - Adding/removing edges
  - Checking if 2 nodes are connected
- Cons:
  - Quadratic space: $O(n^2)$
  - We usually have sparse graphs
  - Adding nodes is expensive
  - Retrieval of all the neighbouring nodes takes linear time: $O(n)$

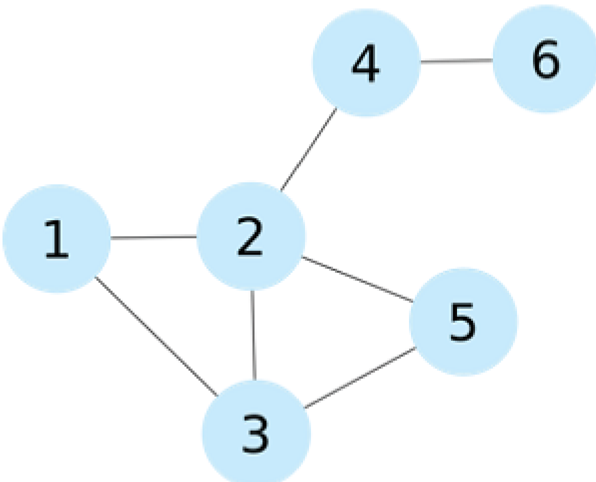
Data Structure: Adjacency List

- A set of lists, each enumerating neighbours of one node
  - A vector of *n* pointers to adjacency lists

- Undirected graph:
  - An edge connects nodes *i* and *j*
  - => the adjacency list of *i* contains node *j* and vice versa

- Often compressed
  - Exploiting regularities in graphs, difference from other nodes, ...

Adjacency List: Example



N1 → {N2, N3}

N2 → {N1, N3, N5}

N3 → {N1, N2, N5}

N4 → {N2, N6}

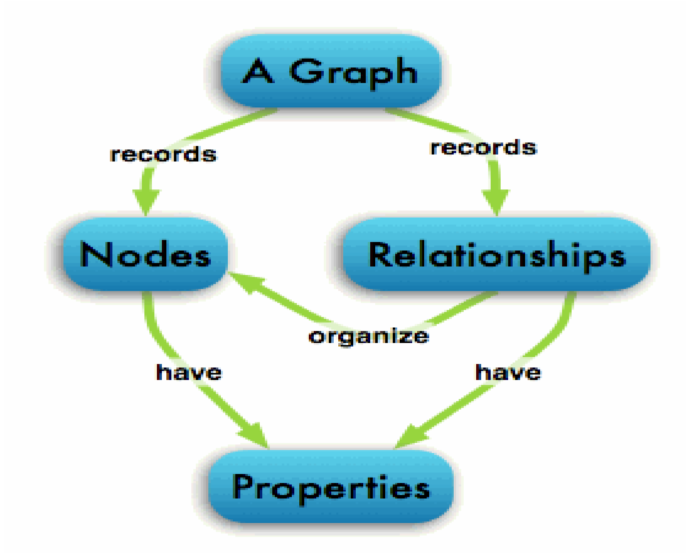
N5 → {N2, N3}

N6 → {N4}

- Pros:
  - Getting the neighbours of a node
  - Cheap addition of nodes
  - More compact representation of sparse graphs
- Cons:
  - Checking if there is an edge between two nodes

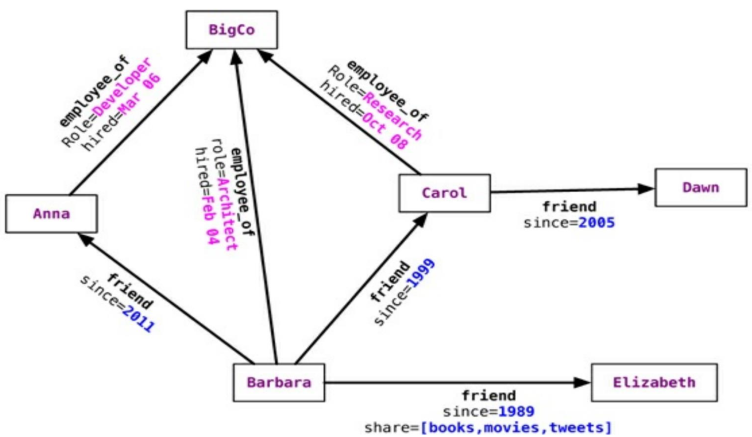
Graphs relationships

- Single-relational graphs
  - Edges are homogeneous in meaning
    - e.g., all edges represent friendship



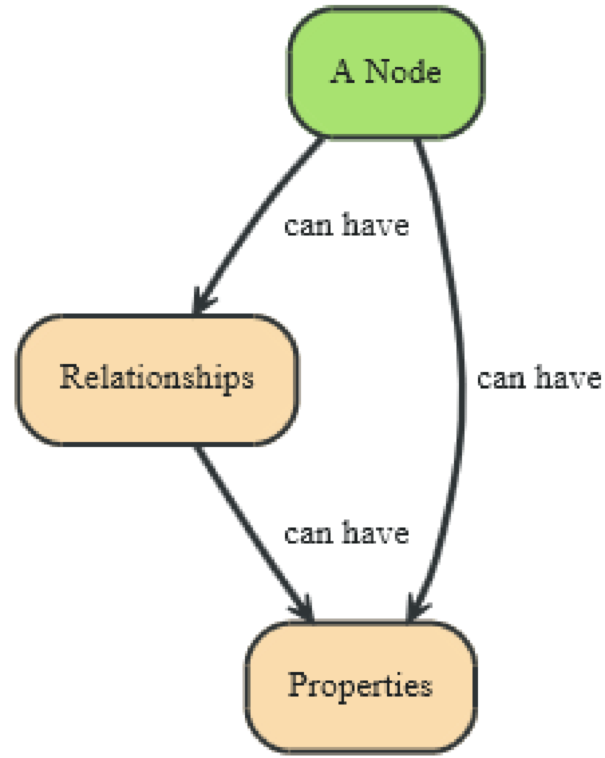
Graphs Relationships

- Multi-relational (property) graphs
  - Edges are typed or labeled
    - e.g., friendship, business, communication
  - Vertices and edges maintain a set of key/value pairs
    - Representation of non-graphical data (properties)
    - e.g., name of a vertex, the weight of an edge



Neo4j: Data Model

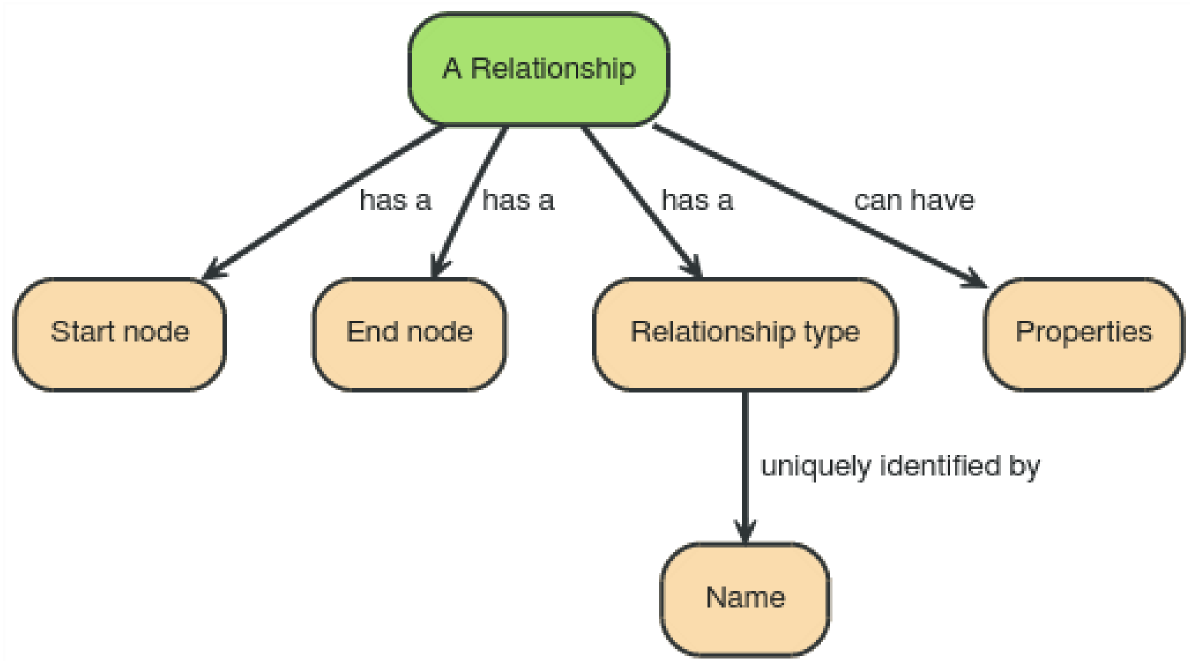
- Fundamental units: nodes + relationships
- Both can contain properties
  - Key-value pairs
  - Value can be of primitive type or an array of primitive type
  - null is not a valid property value
    - nulls can be modelled by the absence of a key



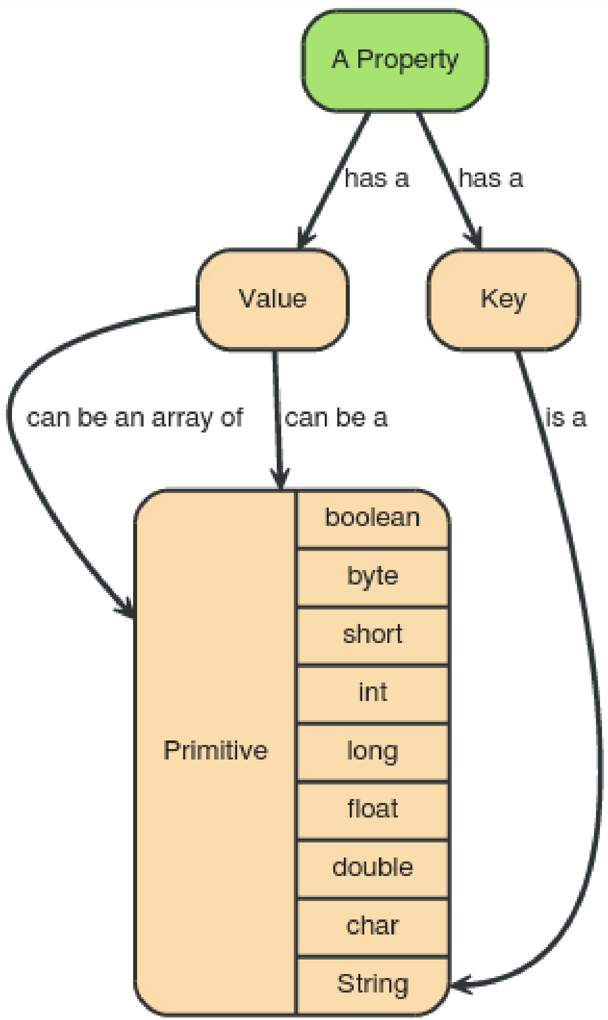
http://db-engines.com/en/system/Neo4j

Data Model: Relationships

- Directed relationships
  - Incoming and outgoing edge
    - Equally efficient traversal in both directions
    - Direction can be ignored
      when not needed by applications
  - Always have start
    and end node
    - Can be recursive



Data Model: Properties



Type	Description
boolean	true/false
byte	8-bit integer
short	16-bit integer
int	32-bit integer
long	64-bit integer
float	32-bit IEEE 754 floating-point number
double	64-bit IEEE 754 floating-point number
char	16-bit unsigned integers representing Unicode characters
String	sequence of Unicode characters

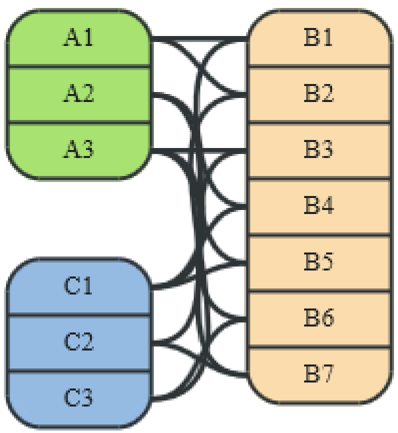
Graphs (Neo4j) vs. RDBMS

- RDBMS designed for a single type of relationship
  - “Who is my manager”
- Adding another relationship usually means a lot of schema changes
- In RDBMS we model the graph beforehand based on the traversal we want
  - If the traversal changes, the data will have to change
  - Graph DBs: the relationship is not calculated but persisted

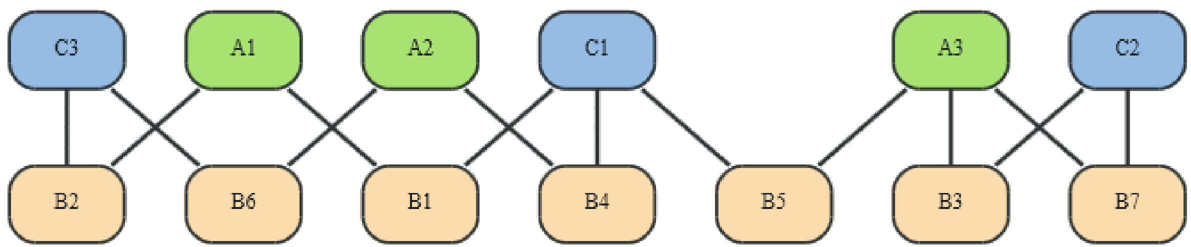
Graphs (Neo4j) vs. RDBMS (2)

- RDBMS is optimized for aggregated data
- Neo4j is optimized for highly connected data
  - It uses adjacency list as a data structure

Relational data



Graph data



source: http://neo4j.com/docs/stable/

Graph DBs: Suitable Use Cases

- Connected Data
  - Social networks
  - Any link-rich domain is well suited for graph databases
- Routing, Dispatch, and Location-Based Services
  - Node = location or address that has a delivery
  - Graph = nodes where a delivery has to be made
  - Relationships = distance
- Recommendation Engines
  - “your friends also bought this product”
  - “when buying this item, these others are usually bought”

Graph DBs: When Not to Use

- If we want to update all or a subset of entities
  - Changing a property on many nodes is not straightforward
    - e.g., analytics solution where all entities may need to be updated with a changed property

- Some graph databases may be unable to handle lots of data
  - Distribution of a graph is difficult

Neo4j: Basic Info

- Open source graph database
- Initial release: 2007
- Written in: Java
- OS: cross-platform
- Full transactions (ACID)
- Partitioning: None
- Replication: Master-slave
    - Eventual consistency

Neo4j in Server mode

- Two ways to use Neo4j:
  - Self-standing server + connections
  - Embeded: Used directly within a Java application

- Server mode:
  - download from http://neo4j.com/download/
  - extract `neo4j-community-X.Y.Z.tar.gz`
  - `./bin/neo4j start`
  - go to: http://localhost:7474/

Cypher: Clauses

- **MATCH**: The graph pattern to match
- **WHERE**: Filtering criteria
- **RETURN**: What to return
- **CREATE**: Creates nodes and relationships.
- **DELETE**: Remove nodes, relationships, properties
- **SET**: Set values to properties
- **WITH**: Divides a query into multiple parts
- **START**: Starting points in the graph
  - by explicit index lookups or by node IDs (both deprecated)

Cypher: Creating Nodes (Examples)

CREATE n;
*(create a node, assign to var **n**)*
`Created 1 node, returned 0 rows`

CREATE (a: Person {name : 'David'})
RETURN a;
(create a node with label 'Person' and 'name' property 'David')
`Created 1 node, set 1 property, returned 1 row`

Cypher: Changing Properties

MATCH (n: Person {name: 'Andres'})
SET n.surname = 'Taylor'
RETURN n
(find a node with name 'Andres' and set it surname 'Taylor')

```
n
  Node[0]{name:"Andres",surname:"Taylor"}
1 row
Properties set: 1
```

Cypher: Delete

MATCH (n: Person {name: 'Andres'})
DELETE n
(delete all Persons with name ‘Andres’)

```
Nodes deleted: 2
TransactionFailureException: Unable to commit transaction
```

MATCH (n: Person {name: 'Andres'}), (n-[r]-())
DELETE r,n
(first, we must delete all relationships of node with name ‘Andres’)

```
Nodes deleted: 1
Relationships deleted: 1
```

Cypher: Queries

MATCH (p: Person)
WHERE p.age > 18 **AND** p.age < 30
RETURN p.name
(return names of all adult people under 30)

MATCH (user: Person {name: 'Andres'})-[:friend]->(follower)
RETURN user.name, follower.name
(find all ‘friends’ of 'Andres')

Cypher: Queries (2)

MATCH (andres: Person {name: 'Andres'})-[*1..3]-(node)
RETURN andres, node ;
(find all 'nodes' within three hops from 'Andres')

MATCH p=shortestPath(
 (andres:Person {name: 'Andres'})-[*]-(david {name:'David'})
)
RETURN p ;
(find the shortest connection between 'Andres' and 'David')

Guidelines on Data model Transformation (Relational -> graph)

- Each entity table is represented by a label on nodes
- Each row in a entity table is a node
- Columns on those tables become node properties.
- Replace foreign keys with relationships to the other table, remove them afterwards
- Remove data with default values, no need to store those
- Indexed column names, might indicate an array property (like email1, email2, email3)
- Join tables are transformed into relationships, columns on those tables become relationship properties

Table translation

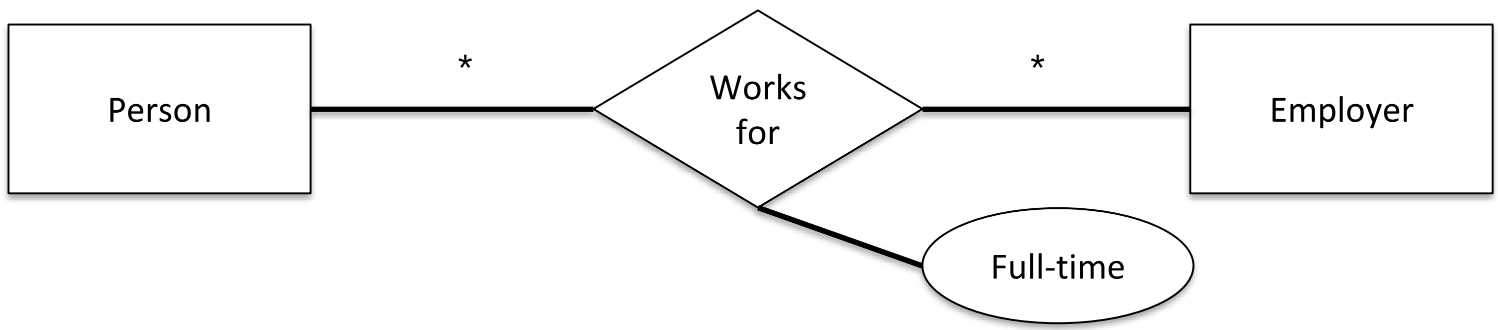
- You cannot translate tables directly.
- In a RDBMS you define the structure of the table.
- In a Graph DB you insert the data as nodes and you give them a <u>type</u>

Person			
name	address	job	married

- CREATE (p:Person{name :‘Jim Raynor’, address:’Mar Sara’,job:’Marshal’,married:false})
  RETURN p;

Join table translation

- In a RDBMS you model relationship as tables.
- The relationship is modelled as data with PK-FK connections.
- In a Graph DB you can type the edges and use them as relationships.



End

- Questions