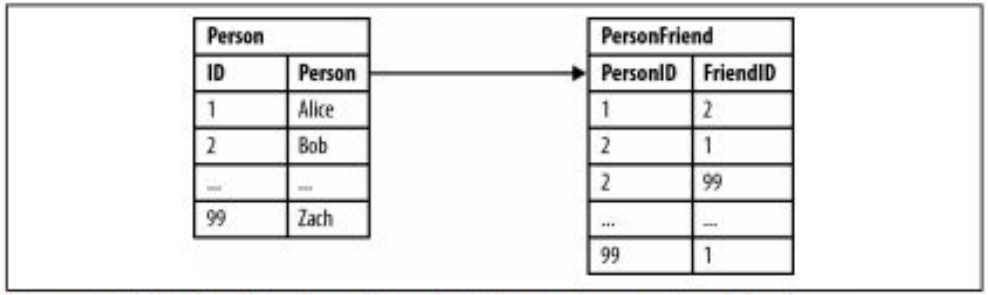
Graph Databases

Илья Малиновский, 444 гр.

27.09.2016

Relational Databases Lack Relationships



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Figure 2-2. Modeling friends and friends-of-friends in a relational database

Example 2-1. Bob's friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-3. Alice's friends-of-friends

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

1,000,000 users having ~50 friends each

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

NoSQL Databases Also Lack Relationships

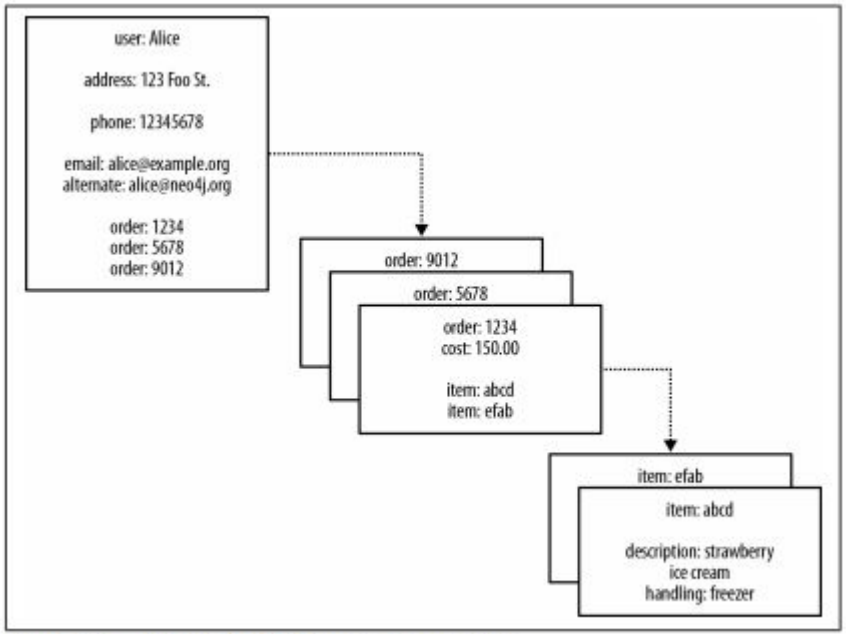
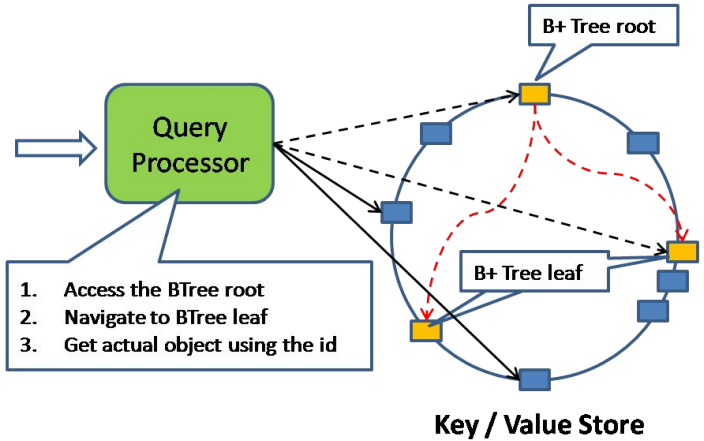


Figure 2-3. Reifying relationships in an aggregate store

Pitfalls

- No index-free adjacency



- No backward relationships

The Labeled Property Graph Model

- *Graph* = (nodes, relationships)
- *Node* = ([properties, labels])
- *Relationship* = (name, start -> end[, properties])
- *Property* = (key, value)

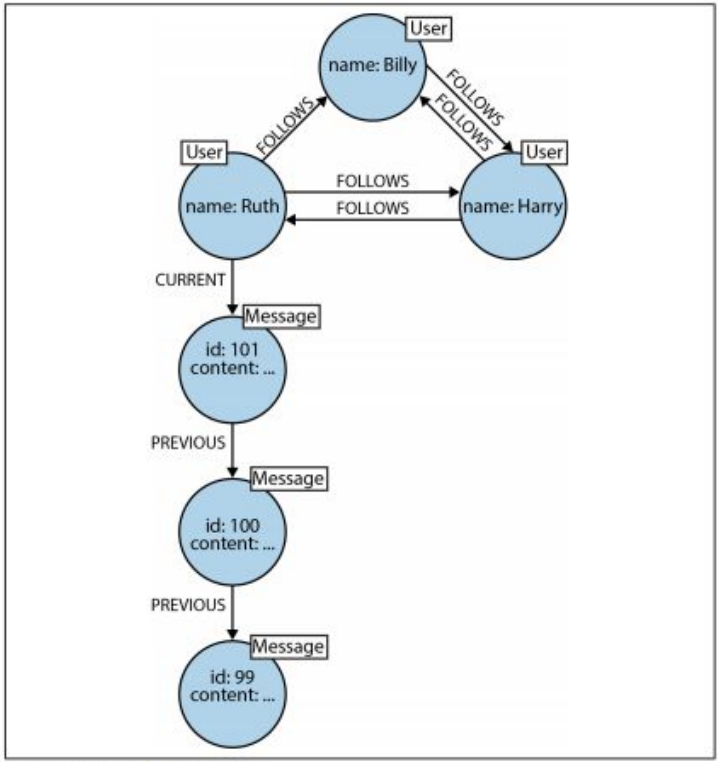
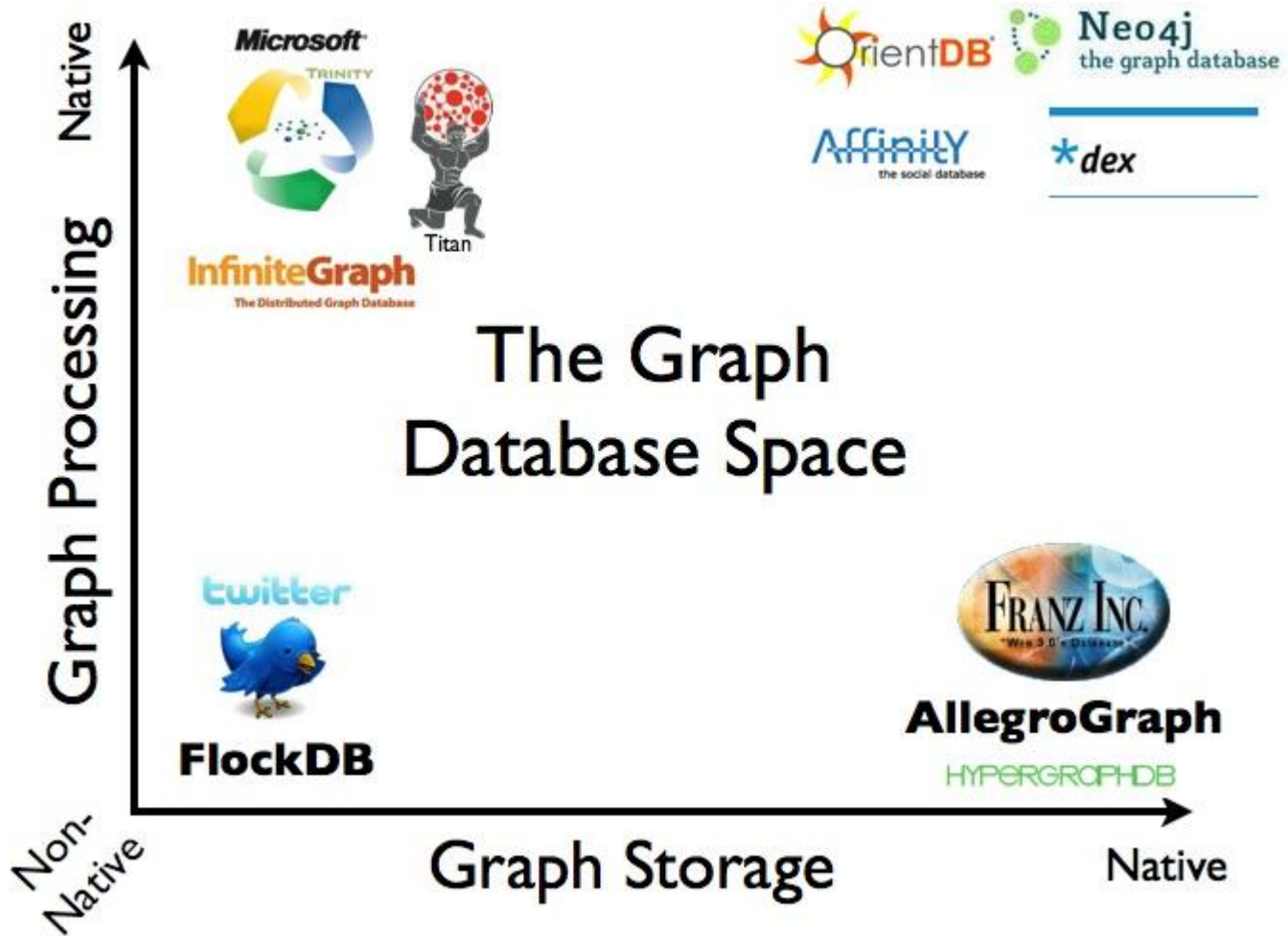


Figure 1-2. Publishing messages


Native
Graph Processing
Non-Native
Graph Storage
Native
Microsoft
TRINITY
Titan
InfiniteGraph
The Distributed Graph Database
OrientDB
Neo4j
the graph database
Affinity
the social database
*dex
The Graph
Database Space
twitter
FlockDB
FRANZ INC.
AllegroGraph
HYPERGRAPHDB

Query Languages

Cypher

```
START n=node(1)
MATCH (n)<-[:KNOWS]-(x)-[:HAS]->()
RETURN x
```

Gremlin

```
g.v(1).in('KNOWS').out('HAS')
.uniqueObject.toList()
```

https://github.com/jadell/neo4jphp/wiki/Cypher-and-gremlin-queries

Cypher. Example 1

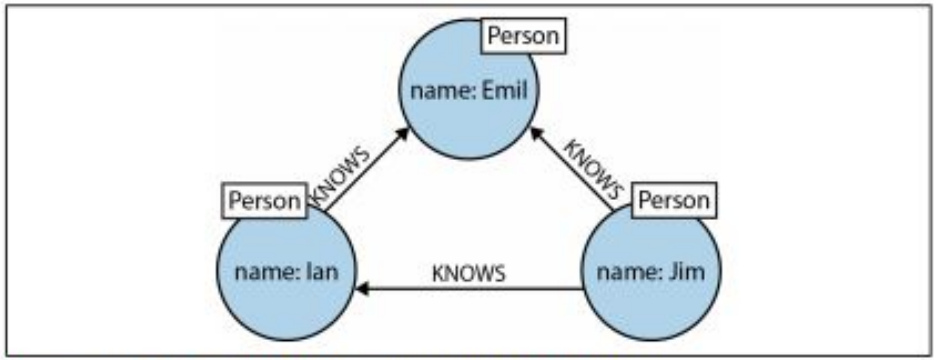


Figure 3-1. A simple graph pattern, expressed using a diagram

```
CREATE (emil:Person {name:'Emil'})
      <-[:KNOWS]-(jim:Person {name:'Jim'})
      -[:KNOWS]->(ian:Person {name:'Ian'})
      -[:KNOWS]->(emil)
```

(specification by example)

Cypher. Example 1. Match

```
MATCH (a:Person)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
WHERE a.name = 'Jim'
RETURN b, c
```

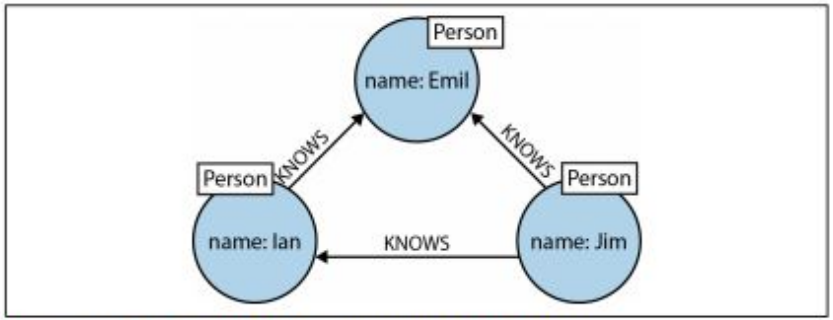


Figure 3-1. A simple graph pattern, expressed using a diagram

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c),
      (a)-[:KNOWS]->(c)
RETURN b, c
```

More on Database Projecting. Relational DBs

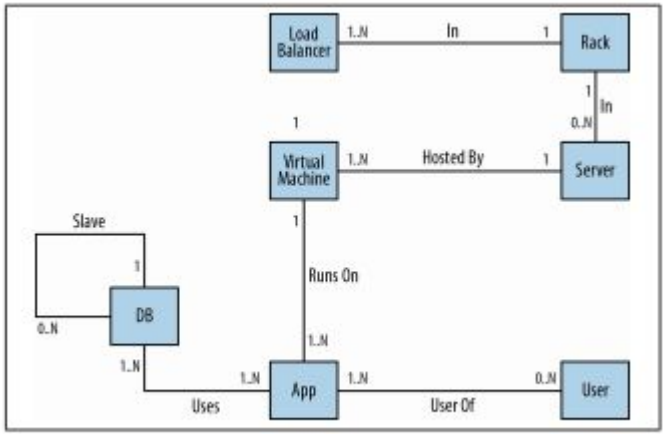


Figure 3-3. An entity-relationship diagram for the data center domain

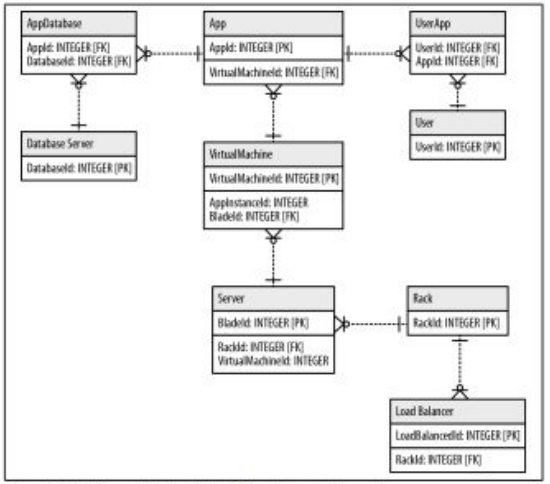
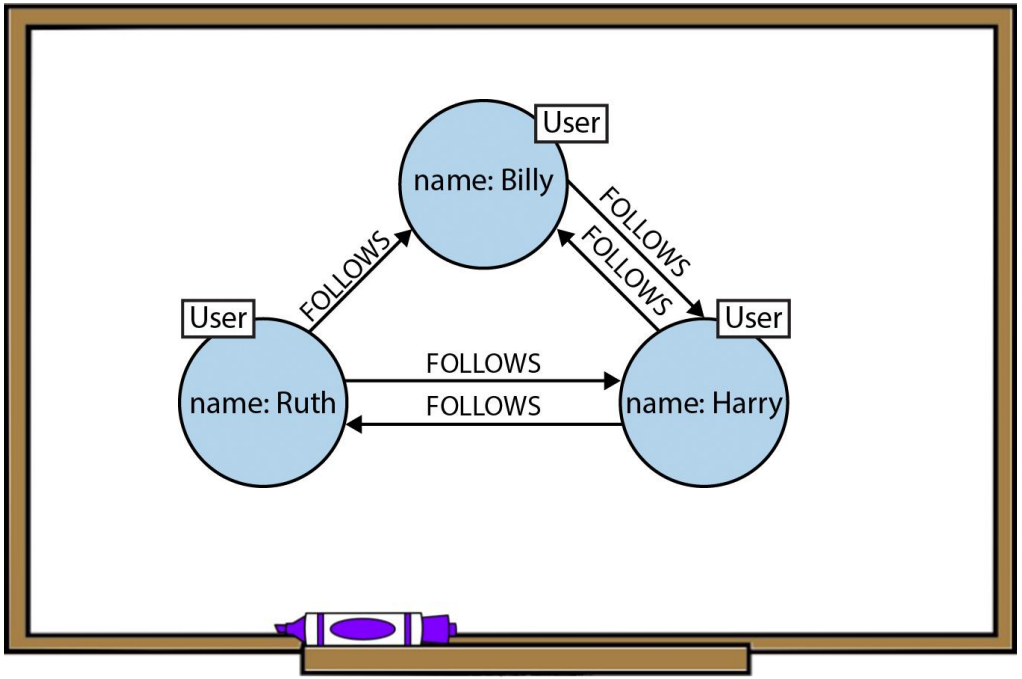


Figure 3-4. Tables and relationships for the data center domain

Normalization

Denormalization

More on Database Projecting. Graph DBs



“what you sketch on the whiteboard is typically what you store in the database”

✓ *Design for queryability*

Cypher. Example 2

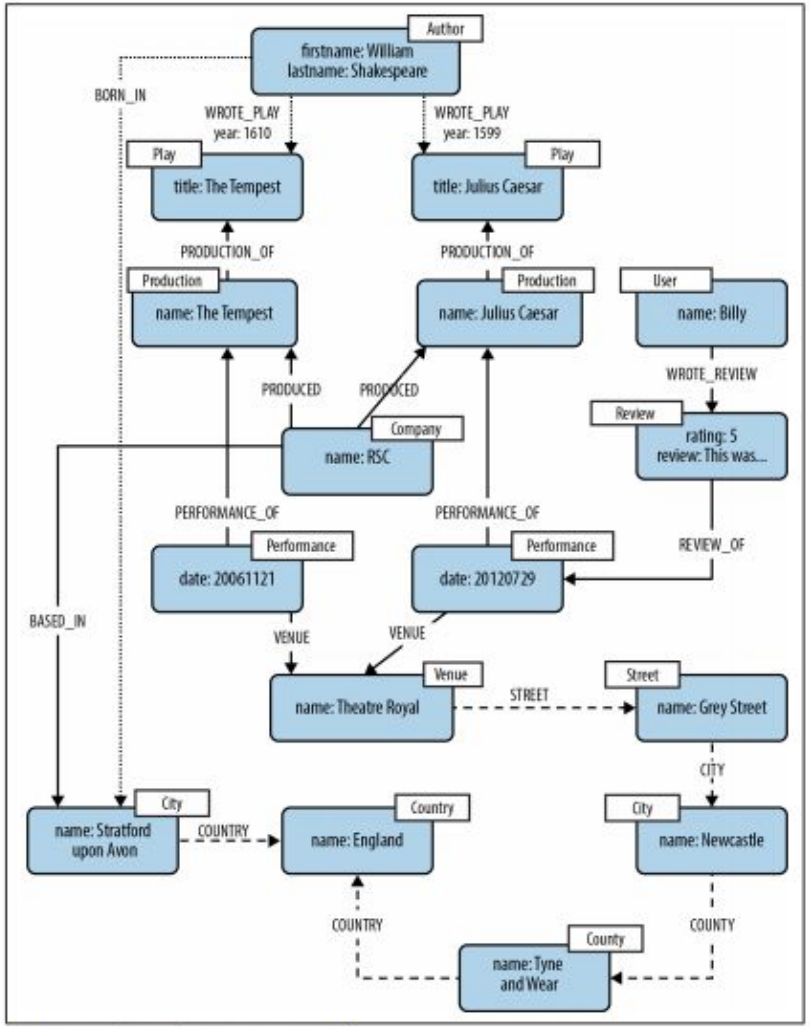


Figure 3-6. Three domains in one graph

Cypher. Example 2. Query

```
MATCH (theater:Venue {name:'Theatre Royal'}),
      (newcastle:City {name:'Newcastle'}),
      (bard:Author {lastname:'Shakespeare'}),
      (newcastle)<-[:STREET|CITY*1..2]-(theater)
      <-[:VENUE]-()-[p:PERFORMANCE_OF]->()
      -[:PRODUCTION_OF]->(play)<-[:WROTE_PLAY]-(bard)
RETURN play.title AS play, count(p) AS performance_count
ORDER BY performance_count DESC
```

Cypher. Example 3

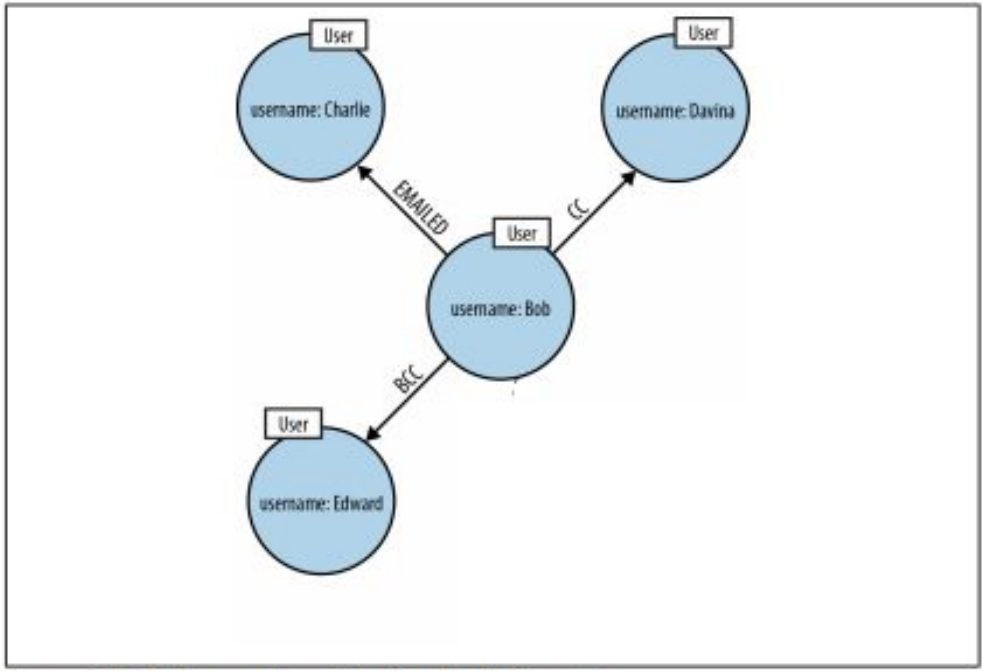


Figure 3-8. Missing email node leads to lost information

Cypher. Example 3, fixed

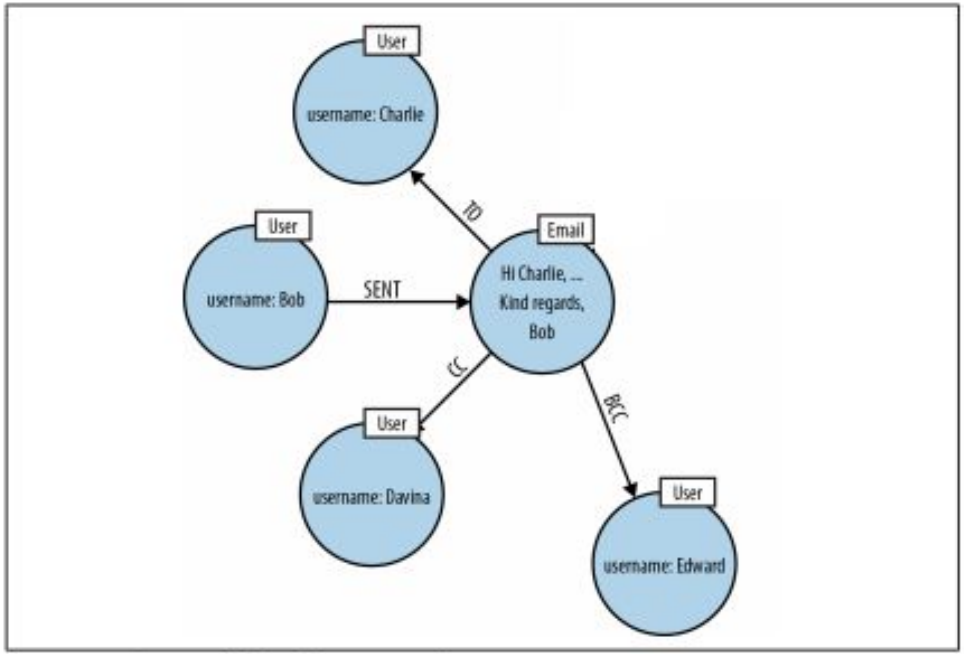


Figure 3-9. Star graph based on an email

Nouns = Nodes
Verbs = Relationships

Avoid *verbing*, i.e. “emailed”, “CCed”, etc.

More on Data Modeling

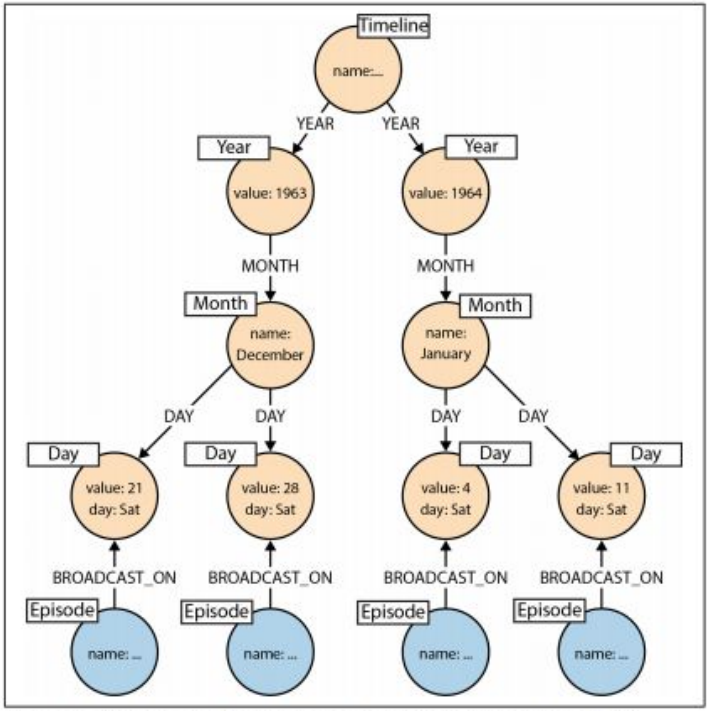


Figure 4-6. A timeline tree showing the broadcast dates for four episodes of a TV program

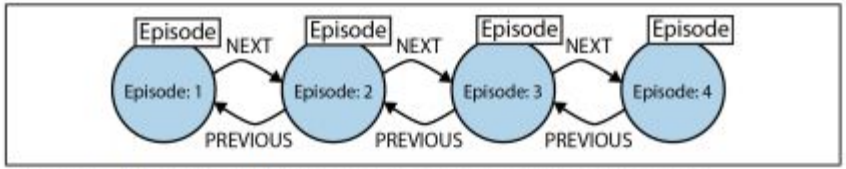


Figure 4-7. A doubly linked list representing a time-ordered series of events

What's inside?

Store files for *nodes, relationships, labels,* and *properties*

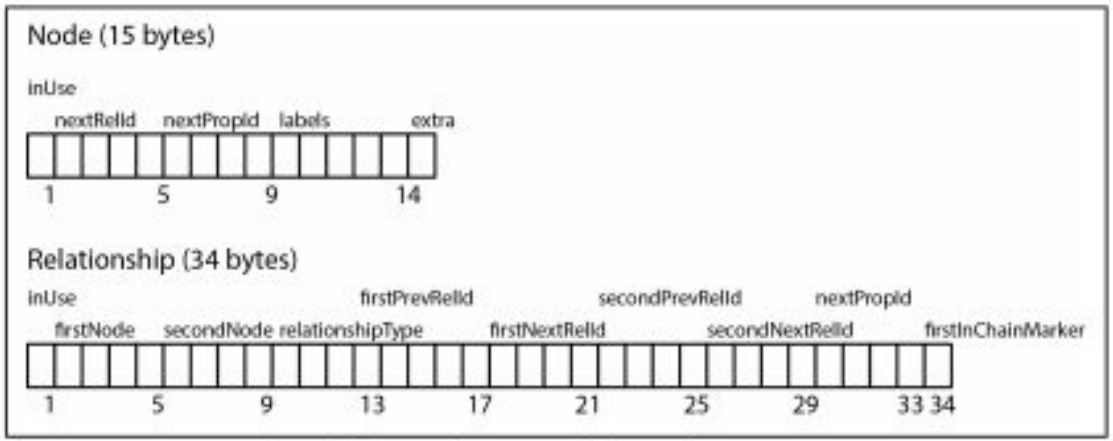


Figure 6-4. Neo4j node and relationship store file record structure

What's inside? (2)

Store files for *nodes, relationships, labels,* and *properties*

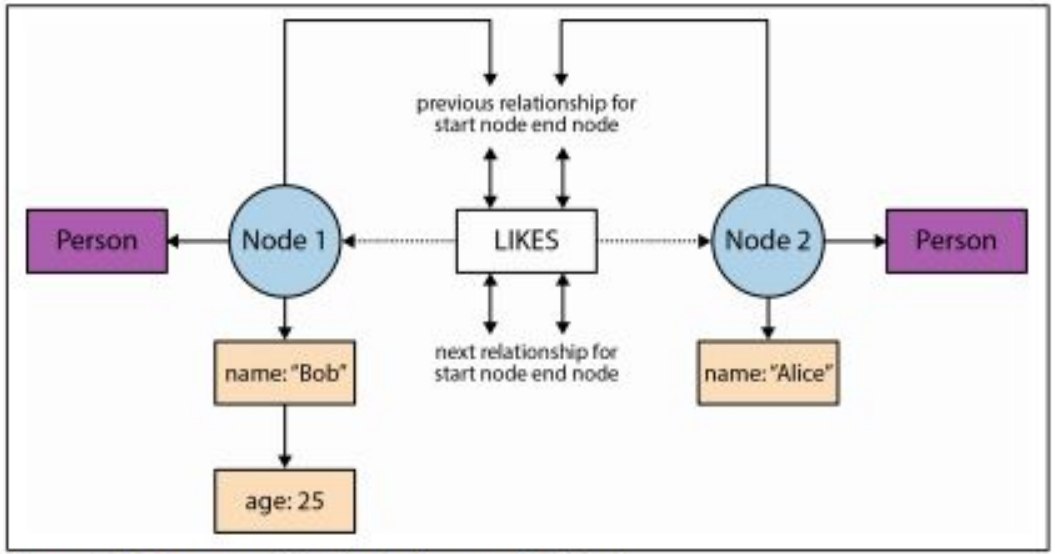


Figure 6-5. How a graph is physically stored in Neo4j

Optimizing O(1)...

- SSDs
- in-memory caching (least frequently used cache policy)

Least Frequently used (LFU) page-replacement algorithm

REF. String	7	0	1	2	0	3	0	4	2	3	0
	7	7	7	2	2	2	2	4	4	3	3
		0	0	0	0	0	0	0	0	0	0
			1	1	1	3	3	3	2	2	2
Count											
0		1	1	1	2	2	3	3	3	3	4
1			1	1	1	1	1	1	1	1	1
2				1	1	1	1	1	2	2	2
3						1	1	1	1	2	2
4								1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1

On Higher Levels

Exposes the graph primitives to the user

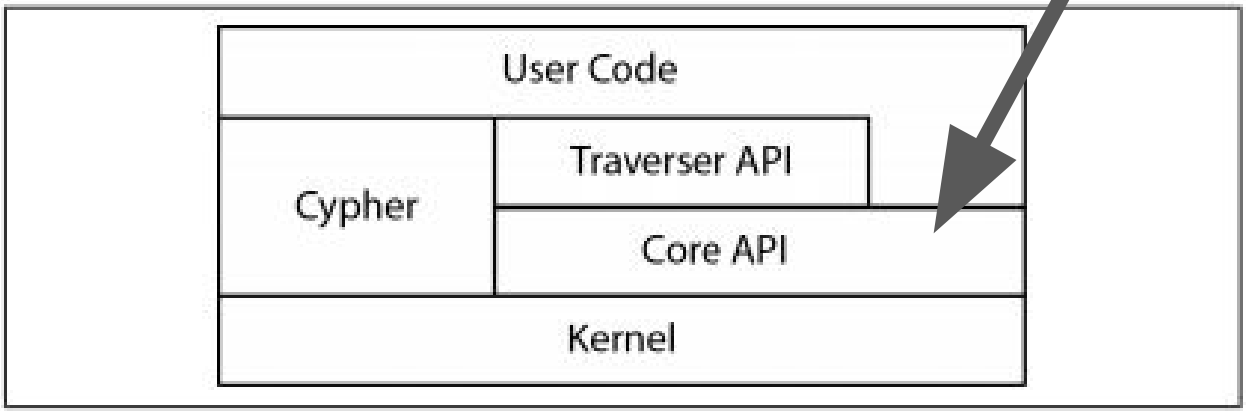
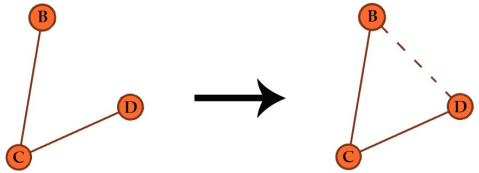


Figure 6-6. Logical view of the user-facing APIs in Neo4j

Harnessing Graph Structure

- Shortest paths (Dijkstra, A*)
- Triadic closures

(predict weak relationships)



- Local bridges

(useful for recommendations)

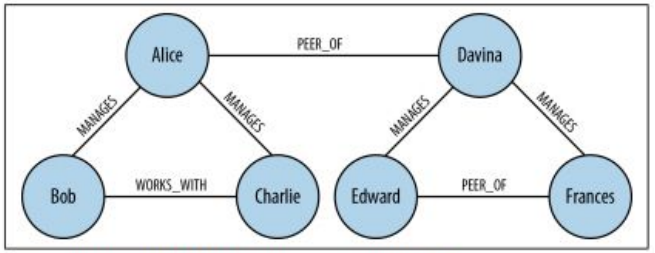
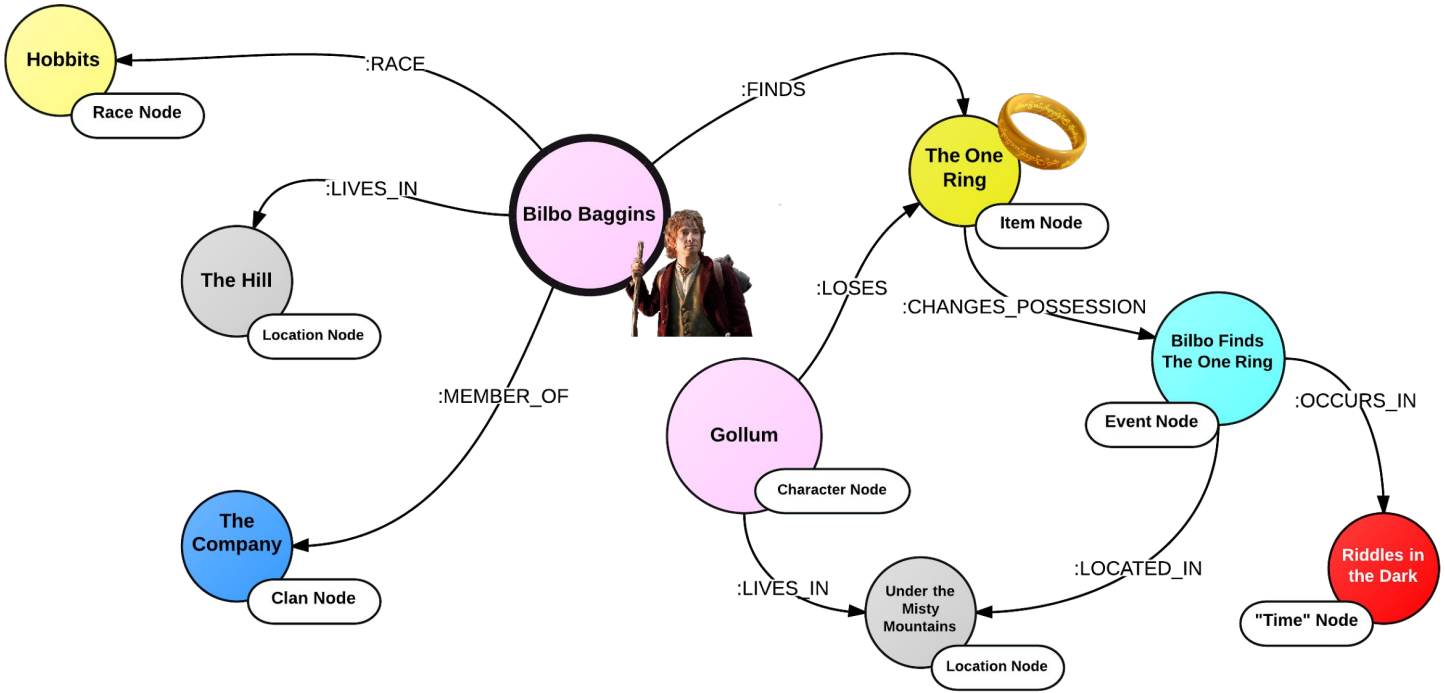


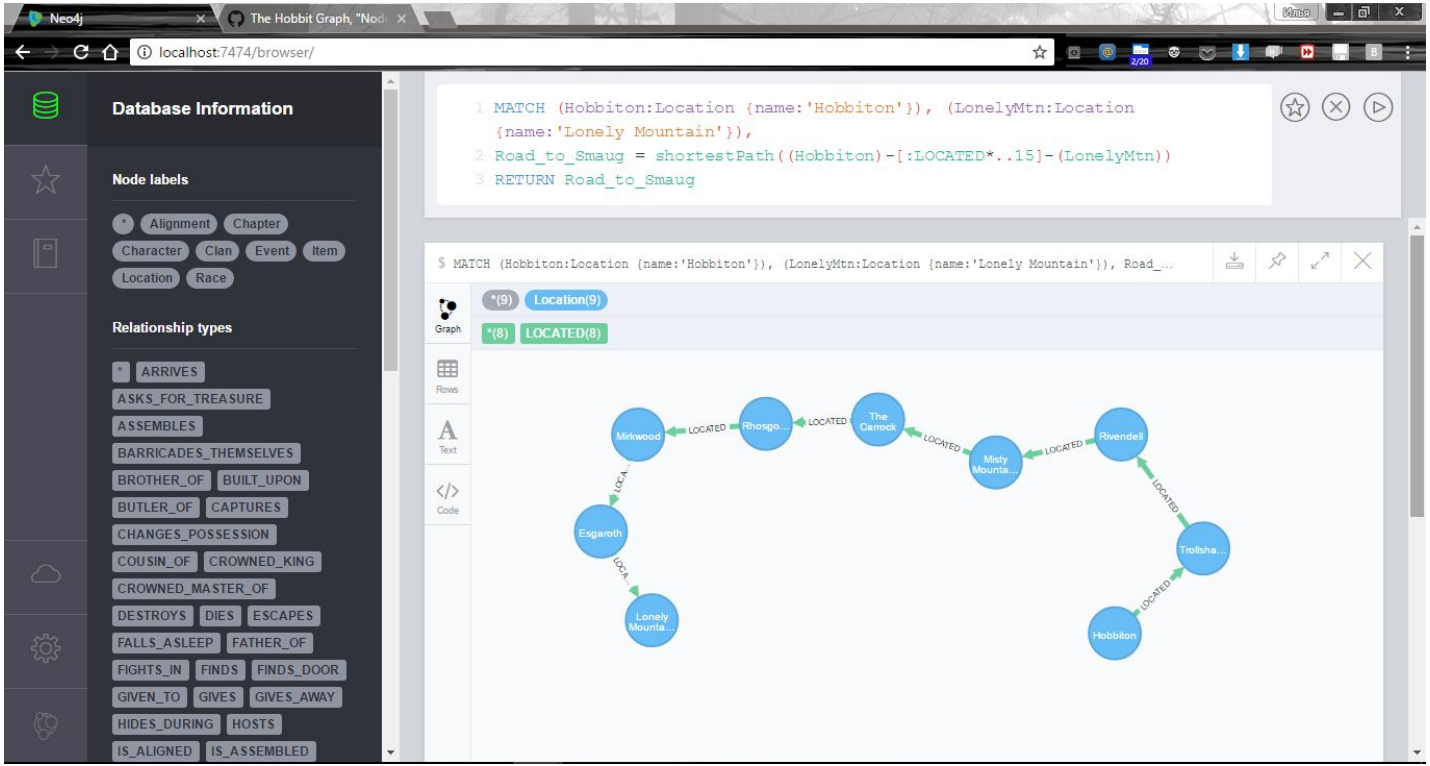
Figure 7-20. Alice and Davina are connected by a local bridge

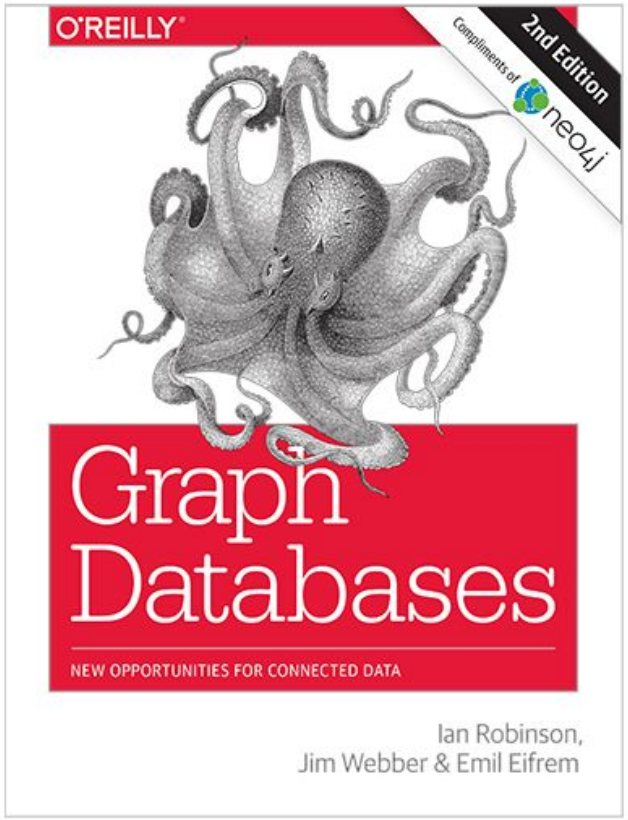
The Hobbit Graph, or To Nodes and Back Again



https://gist.github.com/kvangundy/c43ade7d259a77fe49a8

```
MATCH (Hobbiton:Location {name:'Hobbiton'}),
      (LonelyMtn:Location {name:'Lonely Mountain'}),
      Road_to_Smaug = shortestPath((Hobbiton)-[:LOCATED*..15]-(LonelyMtn))
RETURN Road_to_Smaug
```





O'Reilly *Graph Databases*

The Definitive Book on Graph Databases.

Download your FREE copy

* First Name

* Last Name

* Business Email

Get My Free Book

http://graphdatabases.com

Why Choose Graph DBs?

- Performance ≠ f(dataset size)
- Accelerated development cycles
- Extreme business responsiveness