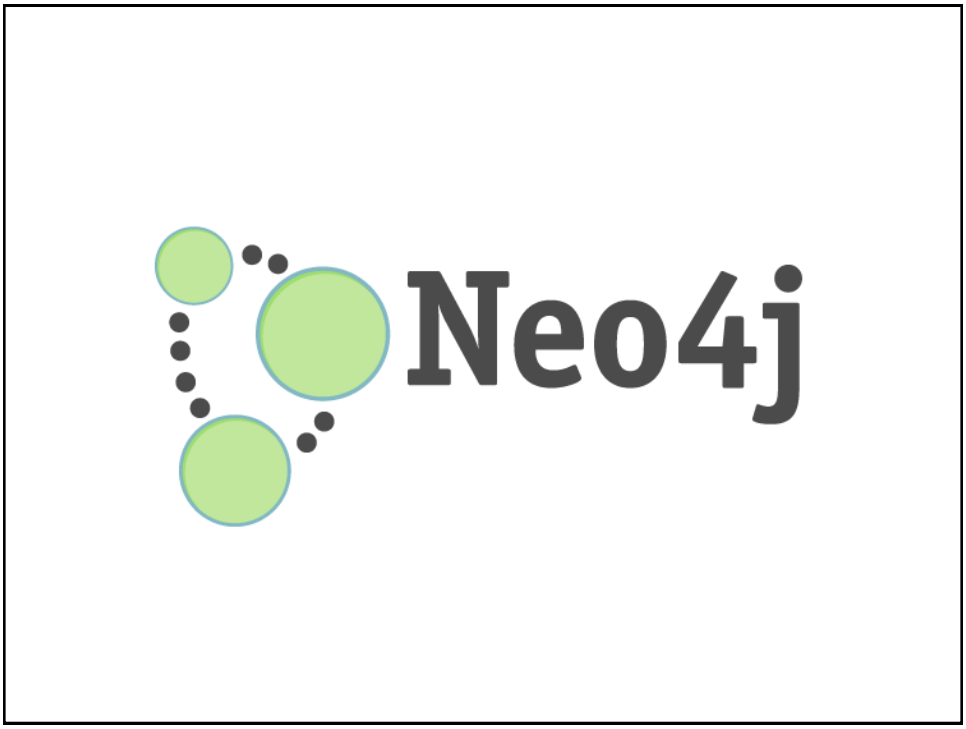
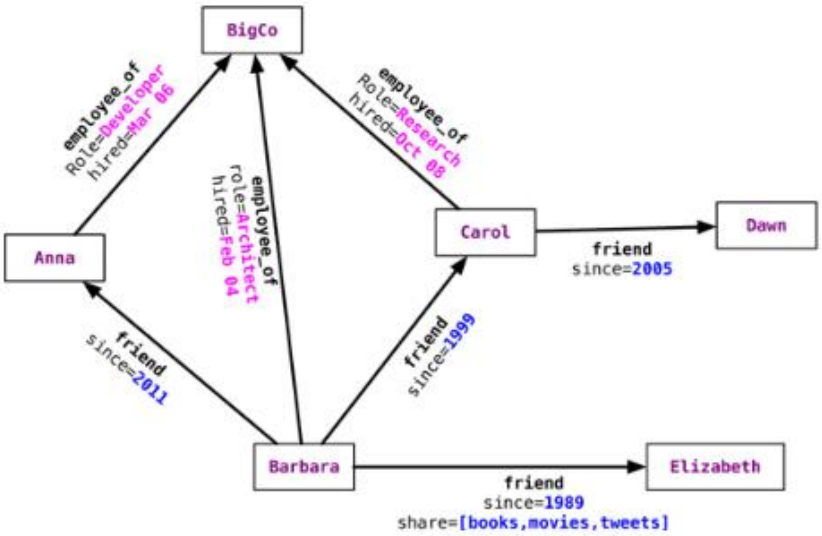


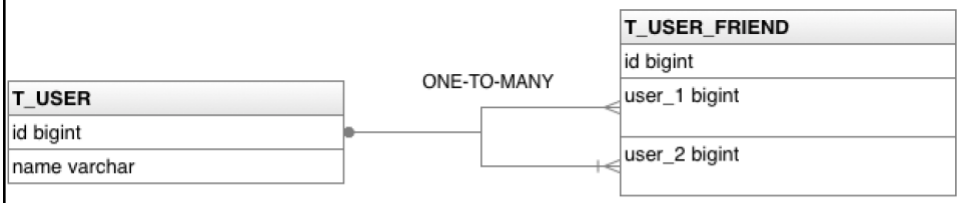
Modelo em Grafos

- Dados organizados em nós e vértices.
- Modelo onde o foco é o relacionamento.

Modelo em Grafos



Uma pequena rede social



T_USER

id	name
1	John S
2	Kate H
3	Aleksa V
4	Jack T
5	Jonas P
5	Anne P

T_USER_FRIEND

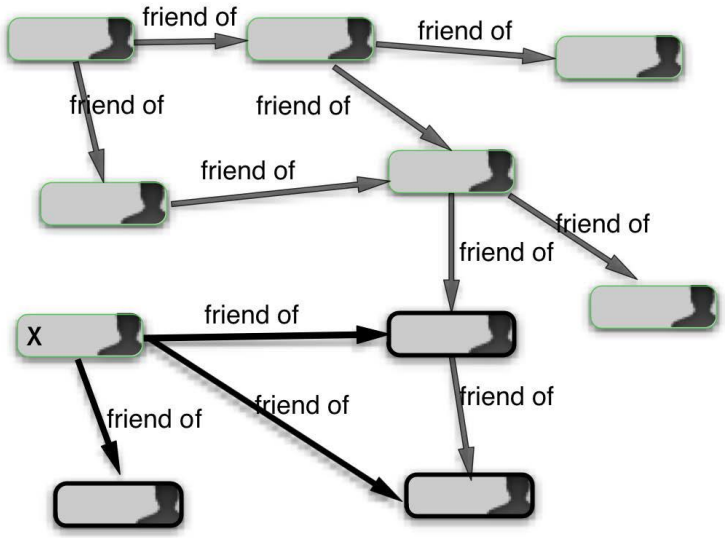
id	user_1	user_2
1000	1	2
1001	3	5
1002	4	1
1003	6	2
1004	4	5
1005	1	4

Busca de Amigos Relacional

- Amigos:
  - `select distinct uf.* from t_user_friend uf where uf.user_1 = ?`
- Amigos dos Amigos:
  - `select distinct uf2.* from t_user_friend uf1 inner join t_user_friend uf2 on uf1.user_1 = uf2.user_2 where uf1.user_1 = ?`
- Amigos dos Amigos dos Amigos:
  - `select distinct uf3.* from t_user_friend uf1 inner join t_user_friend uf2 on uf1.user_1 = uf2.user_2 inner join t_user_friend uf3 on uf2.user_1 = uf3.user_2 where uf1.user_1 = ?`

Fonte: Partner 2013

Rede Social Grafo



Busca de Amigos Grafo

- ```
  TraversalDescription
  traversalDescription =
  Traversal.description().
  relationships("IS_FRIEND_OF",
  Direction.OUTGOING).evaluator(Evaluators
  .atDepth(2)).uniqueness(Uniqueness.NODE_
  GLOBAL);
  ```
- ```
  Iterable<Node> nodes =
  traversalDescription.
  traverse(nodeById). nodes();
  ```

Busca de Amigos

- 1.000.000 usuários com 50 amigos
- MySql:

Depth	Execution time (seconds) for 1 million users	Records returned
2	0.016	~2500
3	30.267	~125,000
4	1543.505	~600,000
5	Not finished	—

Fonte: Partner 2013

Busca de Amigos

- 1.000.000 usuários com 50 amigos
- Neo4j:

Depth	Execution time (seconds) for 1 million users	Records returned
2	0.01	~2500
3	0.168	~110,000
4	1.359	~600,000
5	2.132	~800,000

Fonte: Partner 2013

Por que o tempo do Relacional?

- Para encontrar todos os amigos de um usuário em profundidade 5, um motor de banco de dados relacional precisa gerar o produto cartesiano da tabela t_user_friend cinco vezes. Com 50.000 registros na tabela, o conjunto resultante terá 50.000^5 linhas ($102,4 \times 10^{21}$), o que leva bastante tempo e poder computacional para calcular. Em seguida, descartar mais de 99% dos dados, para retornar apenas 1000 registros!

Fonte: Partner 2013

Por que o tempo do Grafo?

- Devido a estrutura de dados.
- Se alguém lhe perguntar quantas pessoas estão sentados a 5 metros em torno de você, você vai se levantar e contá-los. Se o estádio está meio vazio, você vai contar com as pessoas ao seu redor tão rápido quanto você pode contar. Se o estádio está lotado, você ainda vai fazê-lo em um tempo semelhante.

Fonte: Partner 2013

Ranking Grafos

14 systems in ranking, October 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Neo4j	Graph DBMS	23.68	-0.54
2.	2.	Titan	Graph DBMS	2.32	+0.26
3.	3.	OrientDB	Multi-model	2.00	+0.07
4.	4.	Sparksee	Graph DBMS	0.82	-0.02
5.	5.	Giraph	Graph DBMS	0.41	+0.03
6.	6.	ArangoDB	Multi-model	0.23	-0.02
7.	7.	InfiniteGraph	Graph DBMS	0.20	+0.02
8.	8.	Sqrrl	Multi-model	0.18	+0.04
9.	9.	InfoGrid	Graph DBMS	0.12	+0.01

Fonte: db-engines.com

Neo4j

- Transações ACID
- Alta disponibilidade
- Escala para bilhões de nós e relacionamentos
- Busca de alta velocidade através de *transversals*
- Linguagem de busca declarativa (cypher)

Entidades

- Node
  - Geralmente representa uma entidade
- Relationship
  - Um relacionamento conecta dois nós
  - Pode ser *Incoming* ou *Outgoing*
- Label
  - Usado para agrupar nós
  - Mesmo label significa ser do mesmo grupo
- Propriedades
  - Par chave-valor
  - Chave é uma String
  - Valor é uma primitiva ou array

Tipos de dados

- Boolean
- Byte: 8 bits
- Short: 16 bits
- Int: 32 bits
- Long: 64 bits
- Float: 32 bits
- Double: 64 bits
- Char: 16 bits
- String: sequência de Unicode

Cypher

- Linguagem de query do Neo4j.
- Desenhada para grafos.
- Busca de padrões.

2 nós e 1 relacionamento

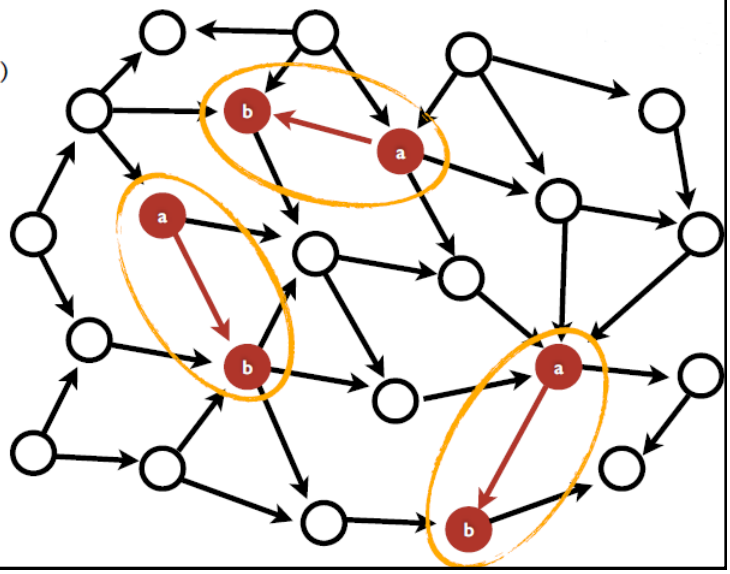
```
MATCH (a)--(b)
RETURN a, b;
```

2 nós e 1 relacionamento

```
MATCH (a)-->(b)
RETURN a, b;
```

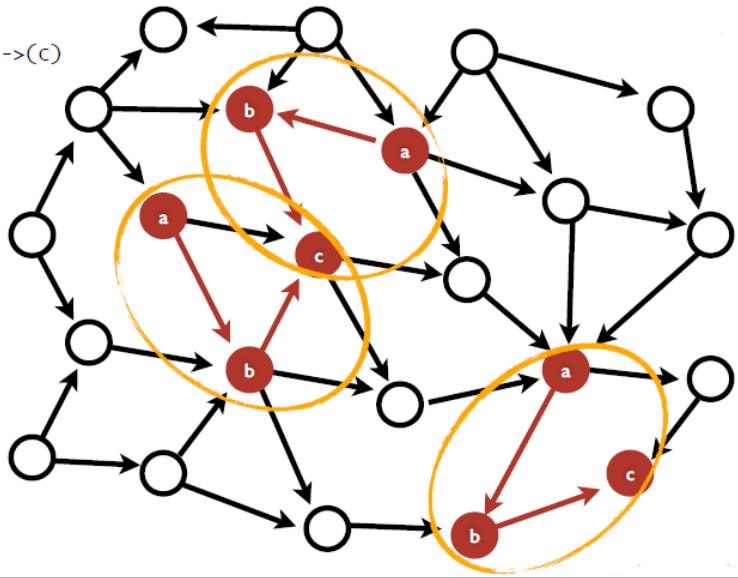
Busca dos nós no grafo

```
MATCH (a)-->(b)
  RETURN a, b;
```



Busca de nós no grafo

```
(a)-->(b)-->(c)
```



Executáveis

- Neo4j.bat
  - Inicia o servidor
- http://127.0.0.1:7474
  - Shell cliente

Criando um nó

```
CREATE (n:Actor {name:"Tom Hanks"})

CREATE (movie:Movie {title:'Sleepless IN Seattle'});
```

Buscando o nó

```
MATCH (actor:Actor {name: "Tom Hanks"})
RETURN actor;
```

Relacionando

```
MATCH (actor:Actor)
WHERE actor.name = "Tom Hanks"

MATCH (movie:Movie)
WHERE movie.title = "Sleepless IN Seattle"

CREATE (actor)-[:ACTED_IN]->(movie);
```

Buscar um Path

```
MATCH (actor) -[:ACTED_IN]-> (movie)
RETURN actor, movie;
```

Setar a propriedade de um Nó

```
MATCH (actor:Actor {name: "Tom Hanks"})
SET actor.DoB = 1944
RETURN actor.name, actor.DoB;
```

Remover um Nó

```
MATCH (actor:Actor {name: "Tom Hanks"})
OPTIONAL MATCH (actor)-[r1]-()
DELETE r1, actor;
```

Índices

```
CREATE INDEX ON :Person(name)

DROP INDEX ON :Person(name)
```

Neo4j em Java

```
GraphDatabaseFactory factory = new
GraphDatabaseFactory();
GraphDatabaseService db =
factory.newEmbeddedDatabase("/grafo");
```

Criando um Grafo em Java

```
public enum RelTypes implements
RelationshipType {
    KNOWS
}
```

Criando um grafo em Java

```
Transaction t = this.db.beginTx();
firstNode = graphDb.createNode();
firstNode.setProperty("message", "Hello, ");
secondNode = graphDb.createNode();
secondNode.setProperty("message", "World!");
relationship = firstNode.createRelationshipTo(
secondNode, RelTypes.KNOWS);
relationship.setProperty("message", "brave
Neo4j");
t.success();
t.finish();
```

Busca de caminhos em Java

```
PathFinder<WeightedPath> finder =
GraphAlgoFactory.dijkstra(
PathExpanders.forTypeAndDirection(
ExampleTypes.MY_TYPE, Direction.BOTH), "cost");

WeightedPath path = finder.findSinglePath(nodeA,
nodeB);

System.out.println(path.weight());
```

Livro Grátis! ☺

- http://graphdatabases.com