

NOSQL – GRAPH DATABASE

MARCIO JASINSKI

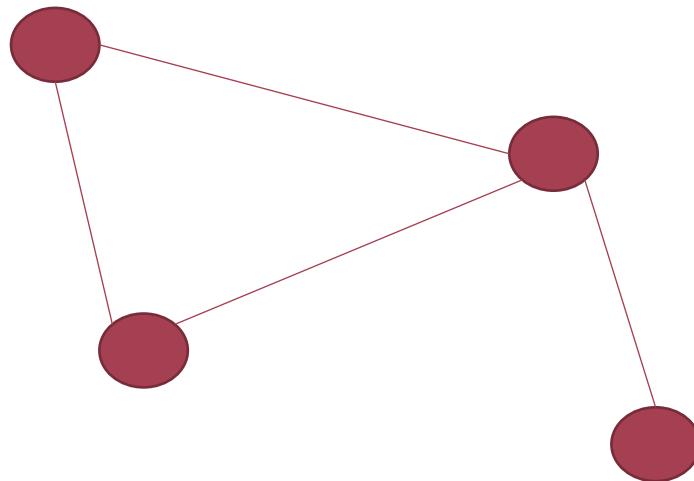
FURB

1. INTRODUÇÃO

Conceitos básicos de grafos

INTRODUÇÃO

- **Teoria dos Grafos** – Ramo da matemática que utiliza modelos para estudar relações entre os objetos de um conjunto.



Exemplo de um Grafo



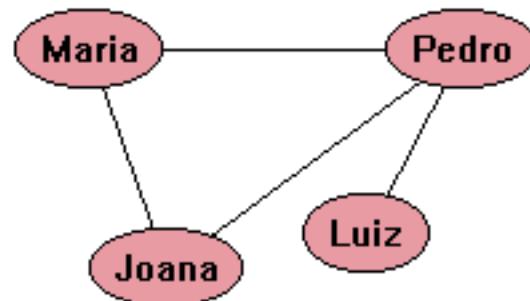
INTRODUÇÃO

- **Problemas comumente representados por grafos**
 - Trajetos entre cidades
 - Roteamento de veículos
 - Redes de computadores
 - Máquina de estados finite
 - Redes Sociais
- Grafos é uma estrutura que tem muito valor para computação como ferramenta elegante para resolver problemas complexos do mundo real.



INTRODUÇÃO

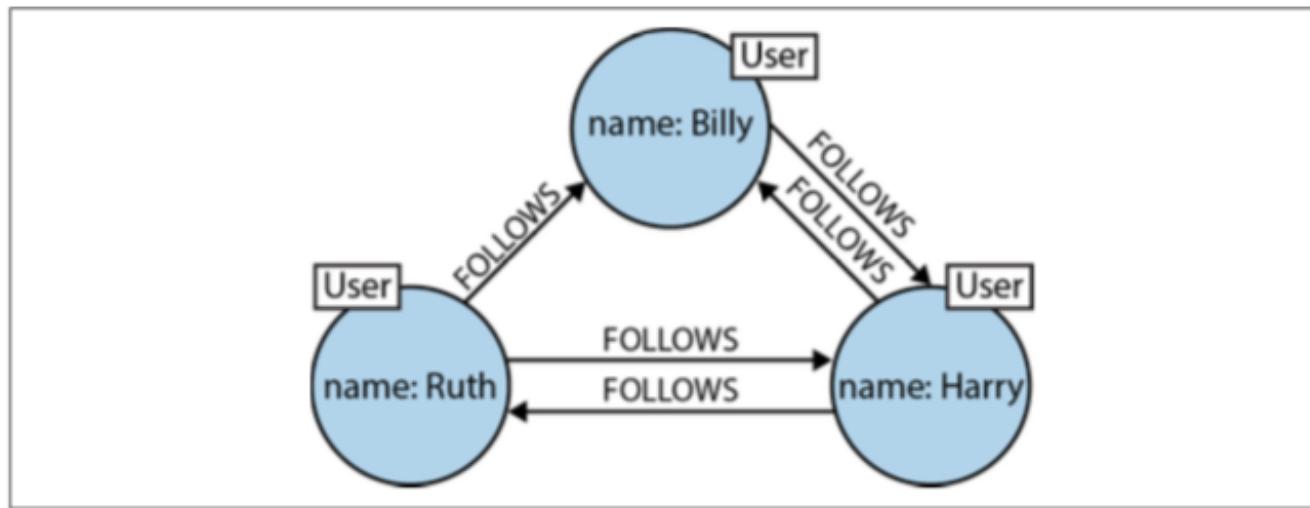
- Um grafo $G(V,A)$ é definido pelo par de conjuntos V e A , onde:
 - V - conjunto não vazio: os **vértices** ou **nodos** do grafo;
 - A - conjunto de pares ordenados $a=(v,w)$, v e $w \in V$: as **arestas** do grafo.
- Seja, por exemplo, o grafo $G(V,A)$ dado por: $V = \{ p \mid p \text{ é uma pessoa} \}$
 $A = \{ (v,w) \mid < v \text{ é amigo de } w > \}$



Nesse caso, a relação $< v \text{ é amigo de } w >$ é uma relação **simétrica**, ou seja, se $< v \text{ é amigo de } w >$ então $< w \text{ é amigo de } v >$. Como consequência, as arestas que ligam os vértices não possuem qualquer orientação

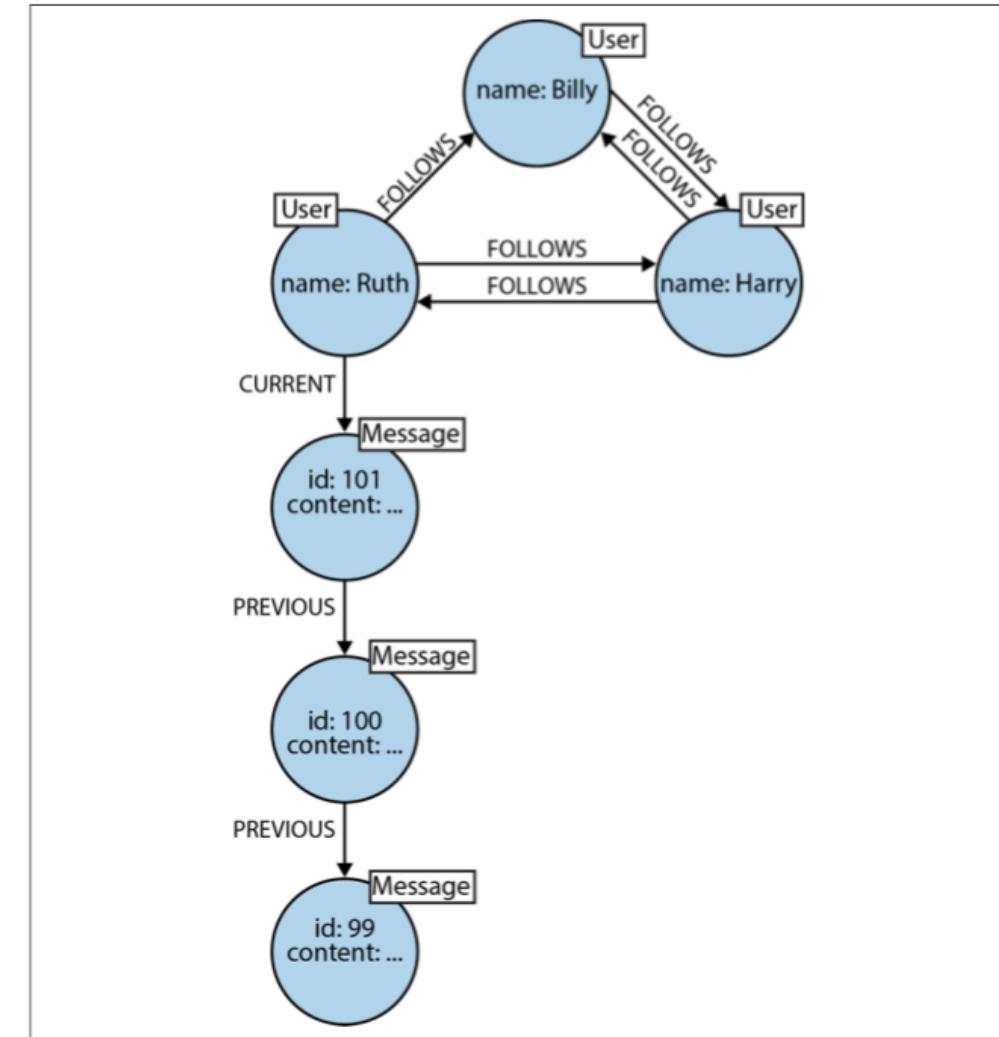
INTRODUÇÃO

- Um grafo pode ser a forma mais natural de se representar algumas estruturas
- Ou pelo menos, mais simples que um modelo relacional
- Por exemplo, uma rede social como o Twitter:



INTRODUÇÃO

- Uma das vantagens de um grafo, é a flexibilidade para conectar novas informações
- O schema é flexível para adotar novas informações e relações.
- No exemplo, temos um grafo rotulado onde:
 - Os vértices tem propriedades (chave/valor)
 - Os vértices tem um ou mais rótulos
 - As arestas são direcionadas e rotuladas



INTRODUÇÃO – BANCO DE GRAFOS

- Um banco de dados de grafos é basicamente um sistema de gerenciamento de dados com operações de CRUD sobre estruturas de grafos.
- Um banco de dados de grafo é um sistema que expõe métodos CRUD sobre uma estrutura de grafos;
- Diferente dos demais bancos de dados, os relacionamentos são “first class citizens” no modelo de grafos;
- Algumas vantagens de Bancos de Grafos sobre Bancos Relacionais
 - Performance
 - Flexibilidade
 - Agilidade
 - Clareza/Semântica



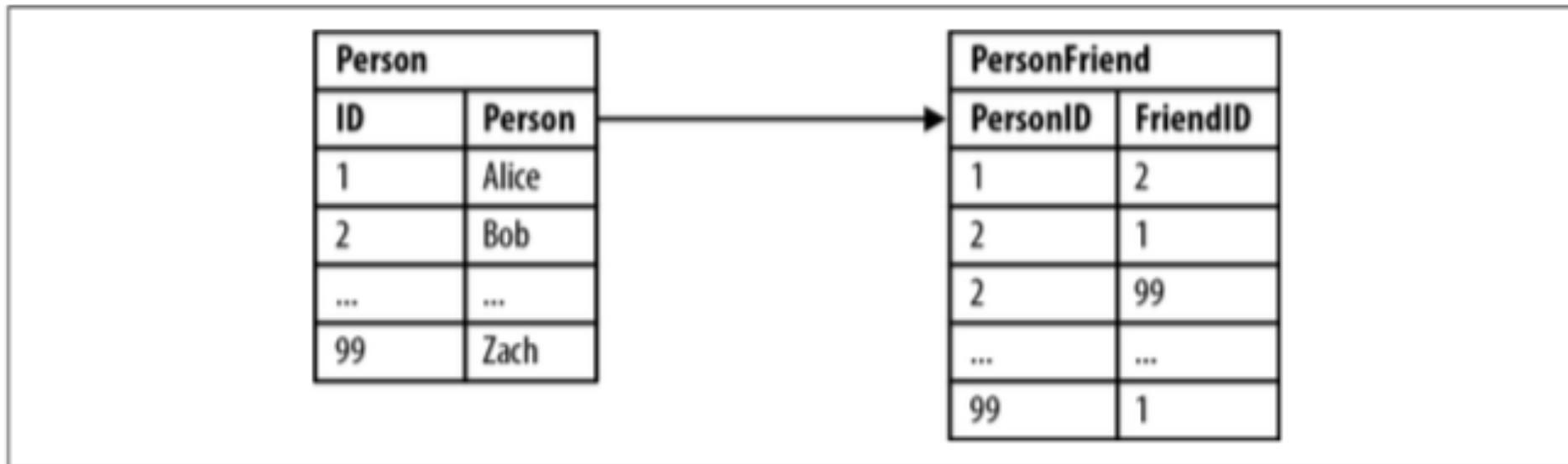
INTRODUÇÃO

- A principal vantagem de bancos de grafos é simplificar operações sobre relacionamentos e entidades;
- Em um banco relacional, os relacionamentos são indiretos, trazendo várias consequências:
 - A compreensão é difícil, pois há pouca clareza ao olhar os dados
 - São necessários códigos de referências entre tabelas
 - É preciso criar constraints para garantir a integridade
 - A construção de queries tende a ter alta complexidade
 - Algumas operações não tem bom desempenho



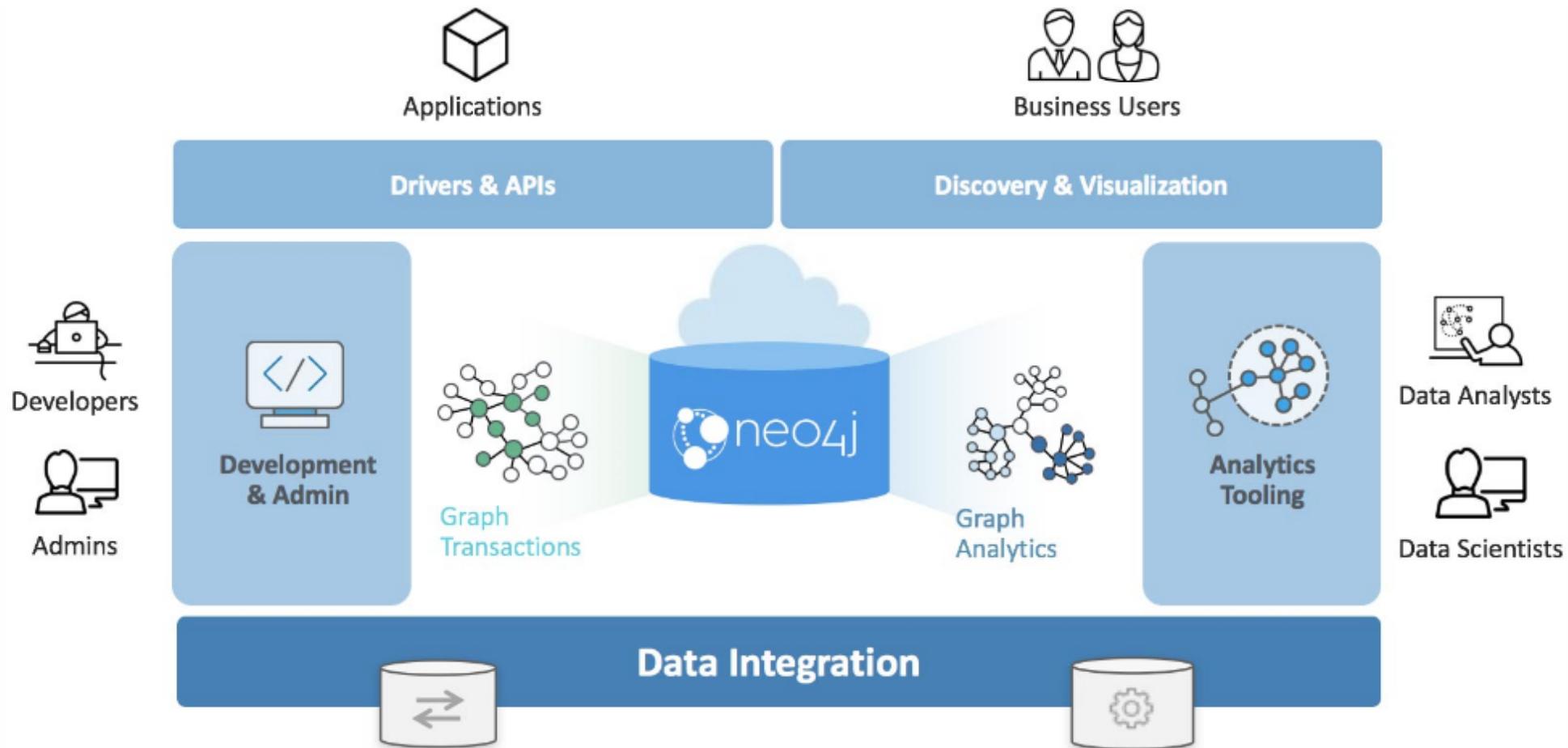
INTRODUÇÃO

- Exemplo



Neo4j Platform

NEO4J DATABASE PLATFORM



NEO4J DATABASE PLATFORM

- A plataforma é composta pelos seguintes elementos :
 - **O banco de dados (core)**: permite o armazenamento de dados como grafos, transações e estatísticas
 - **Linguagem Cypher**: Forma de interagir com o banco. O SQL do Neo4j
 - **Ferramentas de visualização e descoberta**: Neo4j Desktop permite interagir com os grafos de forma visual e exploratória.
 - **Analytics**: Ferramentas para extração de dados e análise
 - **Integração**: Ferramentas para importação de dados
- Possui versão community edition ☺



NEO4J

- Grafos no Neo4j são compostos de dois elementos centrais:

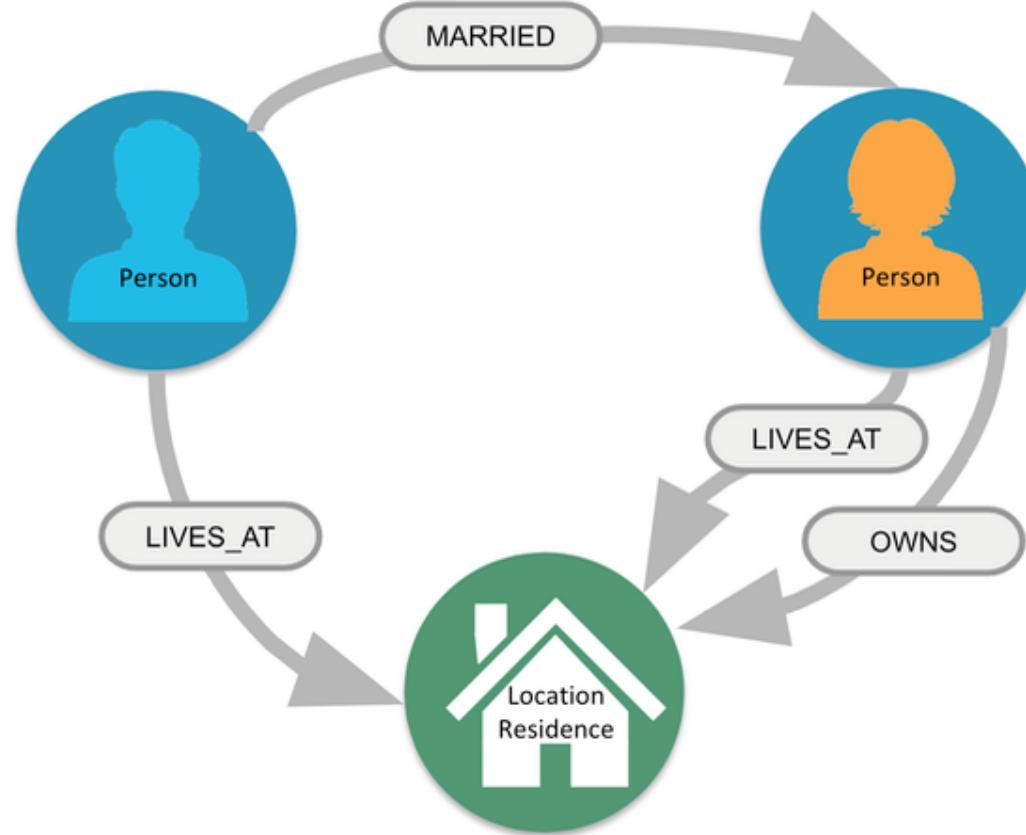
- **Nodes** – Vértices
- **Relationships (edges)** – Arestas



- Os grafos são sempre rotulados (label);
- O rótulo de um vértice é chamado de **label** e se aplica a um ou mais **vértices (node)**
- O relacionamento define a conexão entre dois vértices
- O relacionamento pode ser direcional ou bidirecional

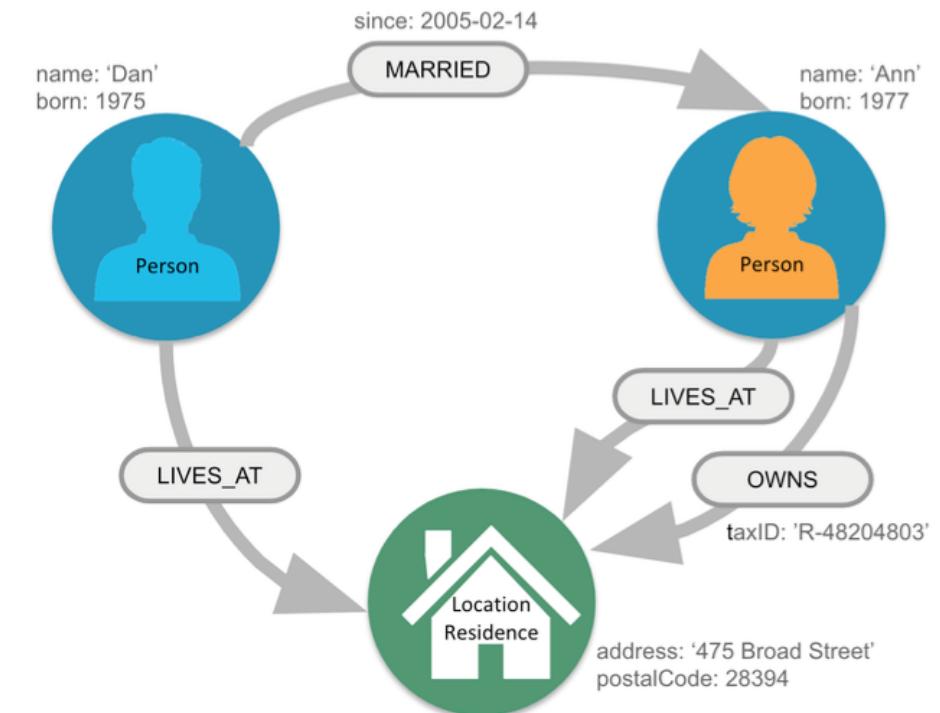


NEO4J



NEO4J

- Outro elemento importante no Neo4j são as propriedades. Dizemos que é um banco que implementa um grafo de propriedades.
- Isso significa que vértices (nodes) e arestas (relationships) podem ter propriedades adicionais.
- Propriedades são basicamente pares chave->valor de informação adicional



NEO4J

- Como compreender os elementos de um banco de grafos para quem conhece o modelo relacional?

Relational	Graph
Rows	Nodes
Joins	Relationships
Table names	Labels
Columns	Properties

NEO4J

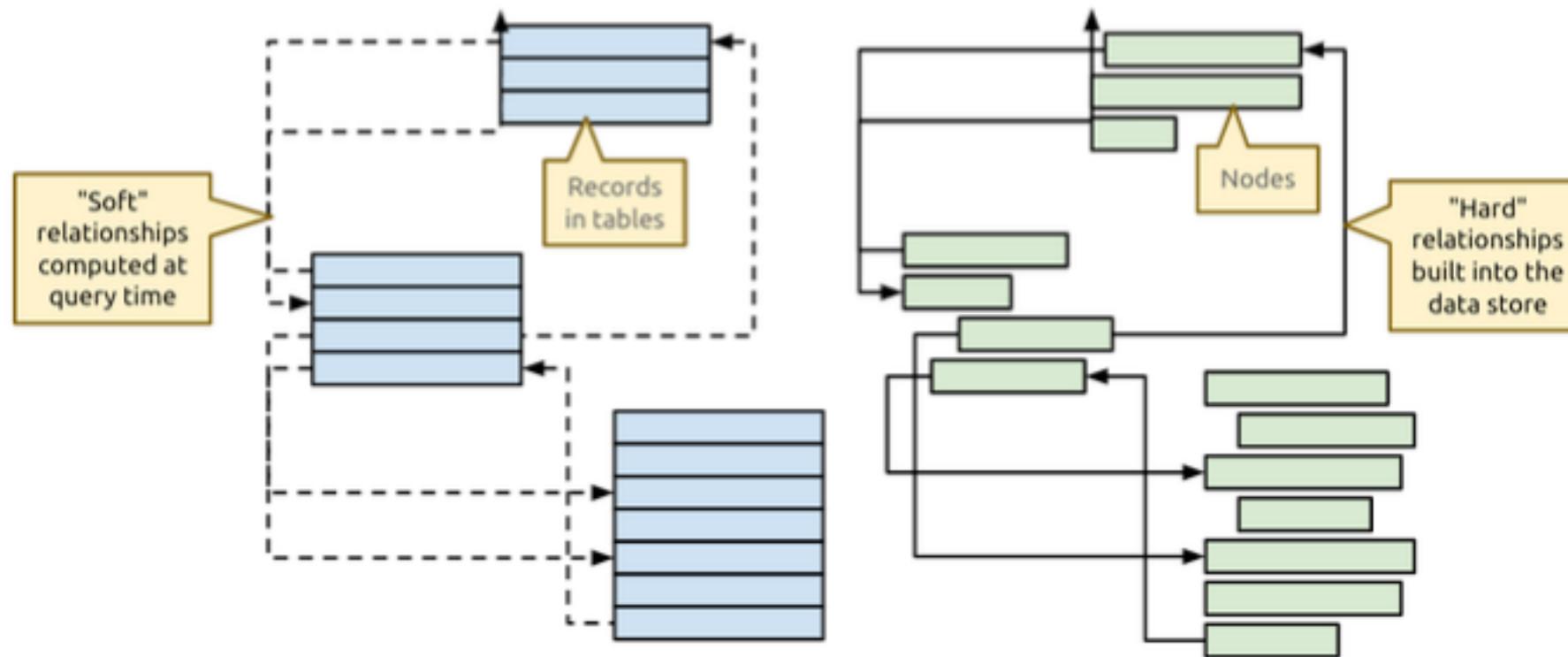
- Mas existem muitas diferenças significativas entre os dois modelos

Relational	Graph
Cada coluna precisa ter um valor que segue o tipo da coluna.	Vértices com o mesmo rótulo (label) não precisam ter o mesmo conjunto de propriedades.
Junções (joins) são calculadas em tempo de execução	Relacionamentos são armazenados em disco quando criados e podem ser percorridos em tempo de execução
Uma linha pertence a uma tabela	Um vértice (node) pode ter múltiplos rótulos (labels)



NEO4J

- A busca dos dados muda significativamente entre um modelo relacional e de grafos:



NEO4J

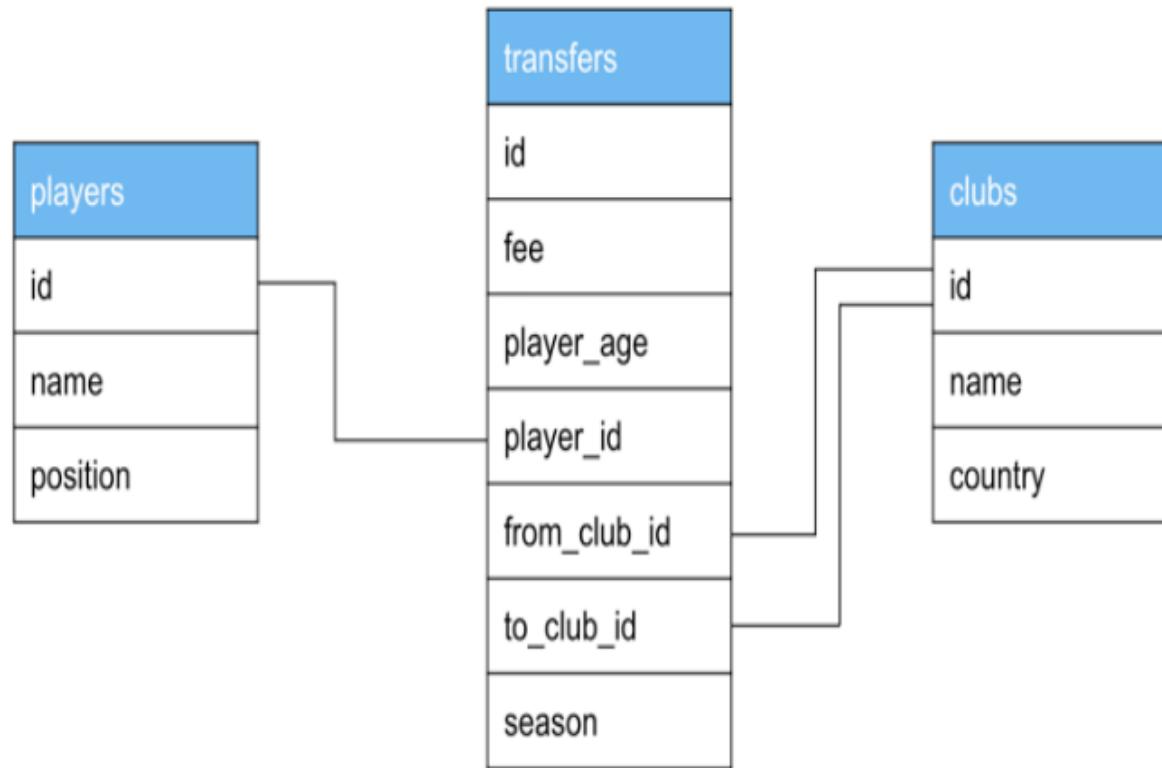
- Diferenças na modelagem dos dados

Relational	Graph
Tenta-se desenhar o schema no planejamento e o ideal é evitar mudanças depois disso.	Há mais flexibilidade para evoluir o schema (pelo menos no banco de dados)
Maior foco na abstração (Classe > Objetos)	Pode ser modelado à partir dos dados existentes.



NEO4J

- Transferência entre jogadores – modelo relacional



NEO4J

- Transferência entre jogadores – modelo de grafos

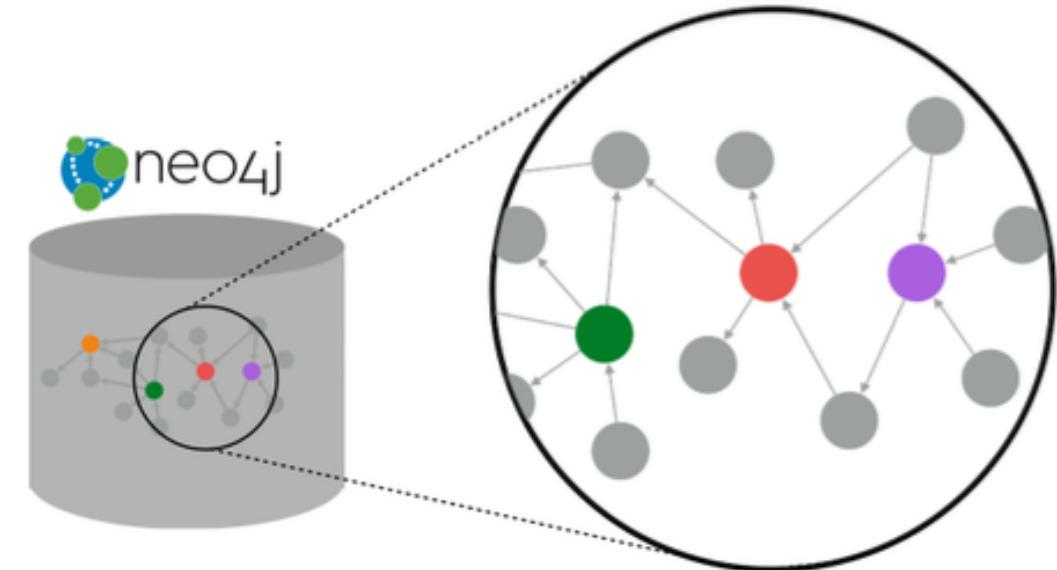


Neo4j

Database

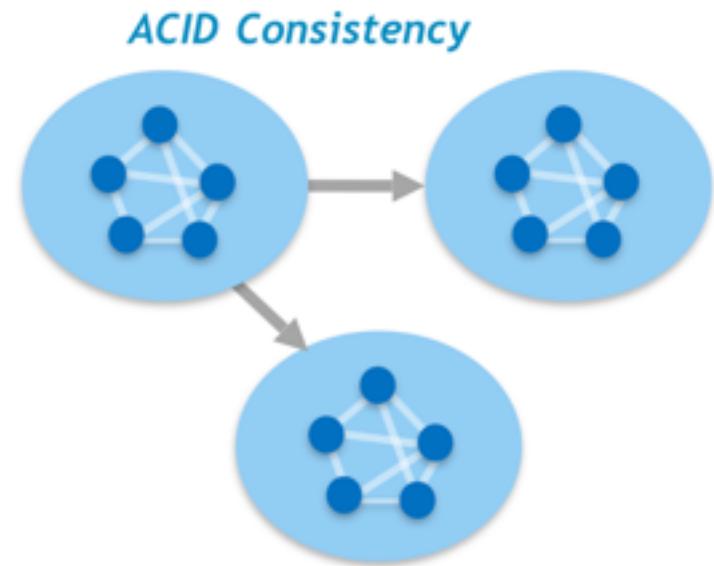
NEO4J DATABASE

- Index-free adjacency
 - Quando um novo vértice ou aresta é adicionada ao grafo, será armazenado como uma conexão aos demais dados.
 - Assim, qualquer acesso subsequente é feito por navegação direcionada.
 - Mesmo grafos grandes podem ser armazenados e acessados em tempo constante sem necessidade de índices
- **Em um banco relacional, qual é a estrutura utilizada para armazenar um índice?**



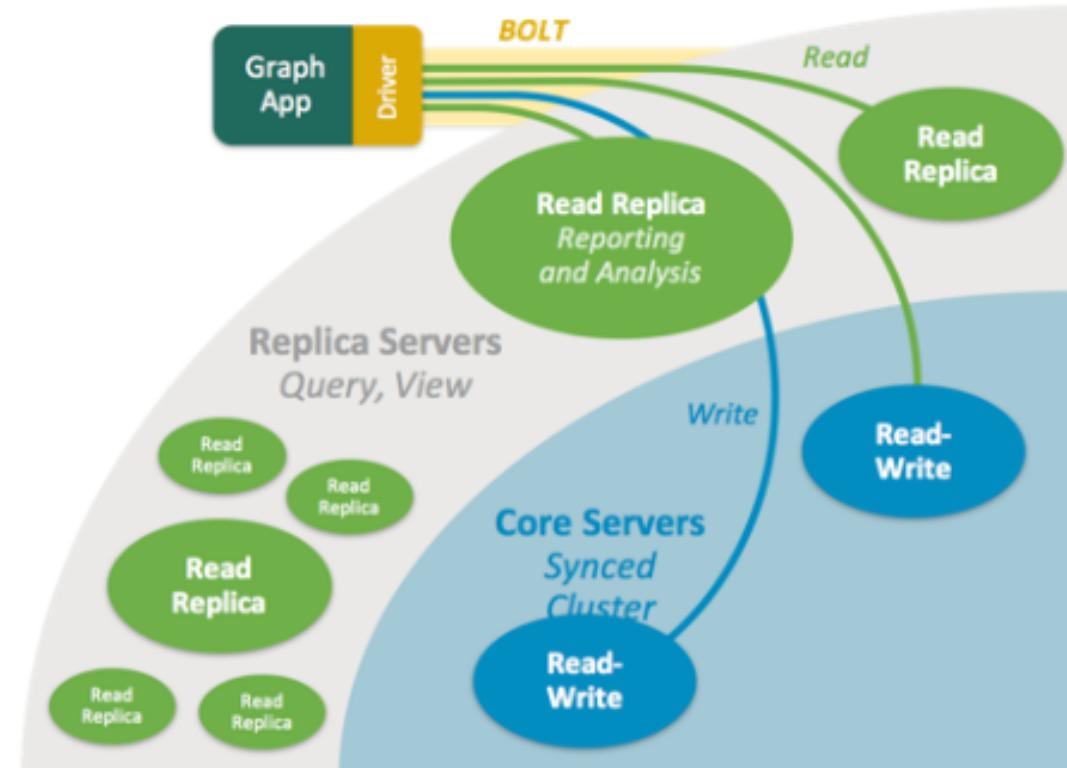
NEO4J DATABASE

- Consistência ACID
 - Se uma aresta entre vértices é criada, não apenas a alteração sobre a aresta, como a alteração sobre os vértices precisa ser consistente. Assim, ou toda operação será feita, ou nada será alterado.
 - Suporte à transações para operações sobre o grafo, índices e schema
 - Modelo de leitura padrão é READ_COMMITTED
 - É possível utilizar locks sobre operações com maior isolamento
 - Locks são restritos à vertices e arestas
 - Detecção de deadlocks.



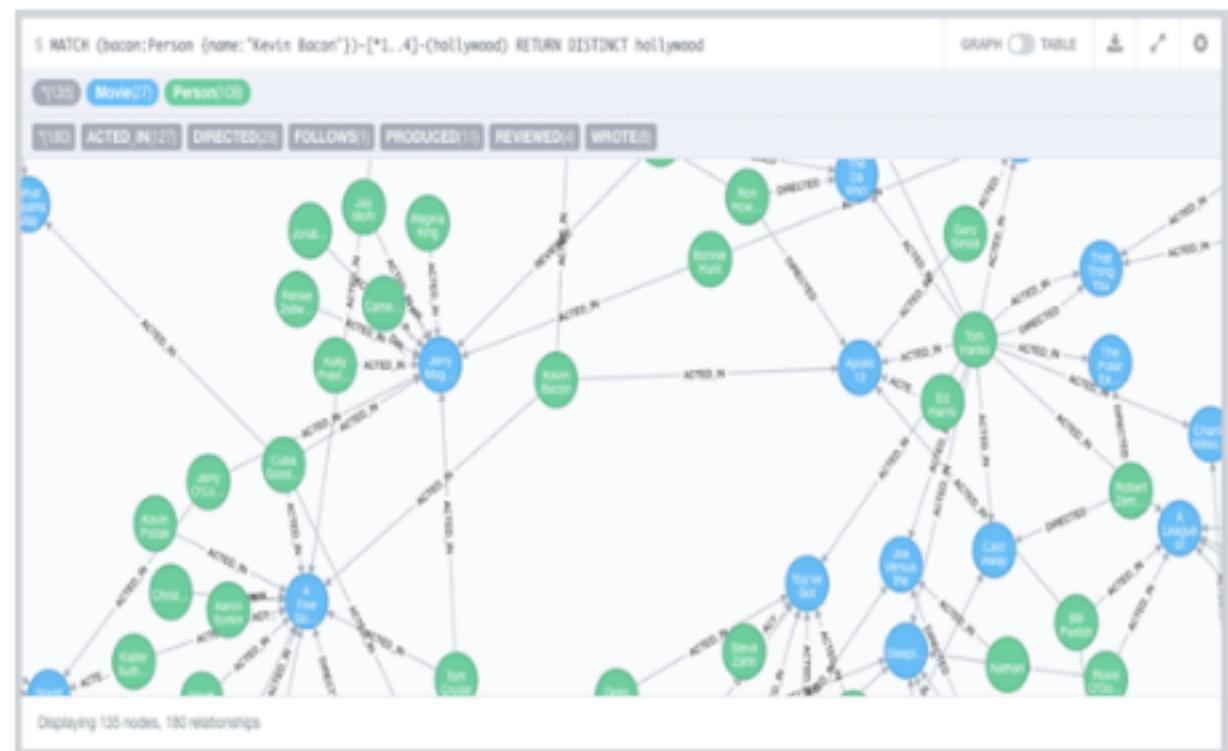
NEO4J DATABASE

- Suporte a cluster
 - Maneira de obter escalabilidade e disponibilidade
 - Modelo mais focado em read replica
 - Servidores de escrita precisam ser sincronizados



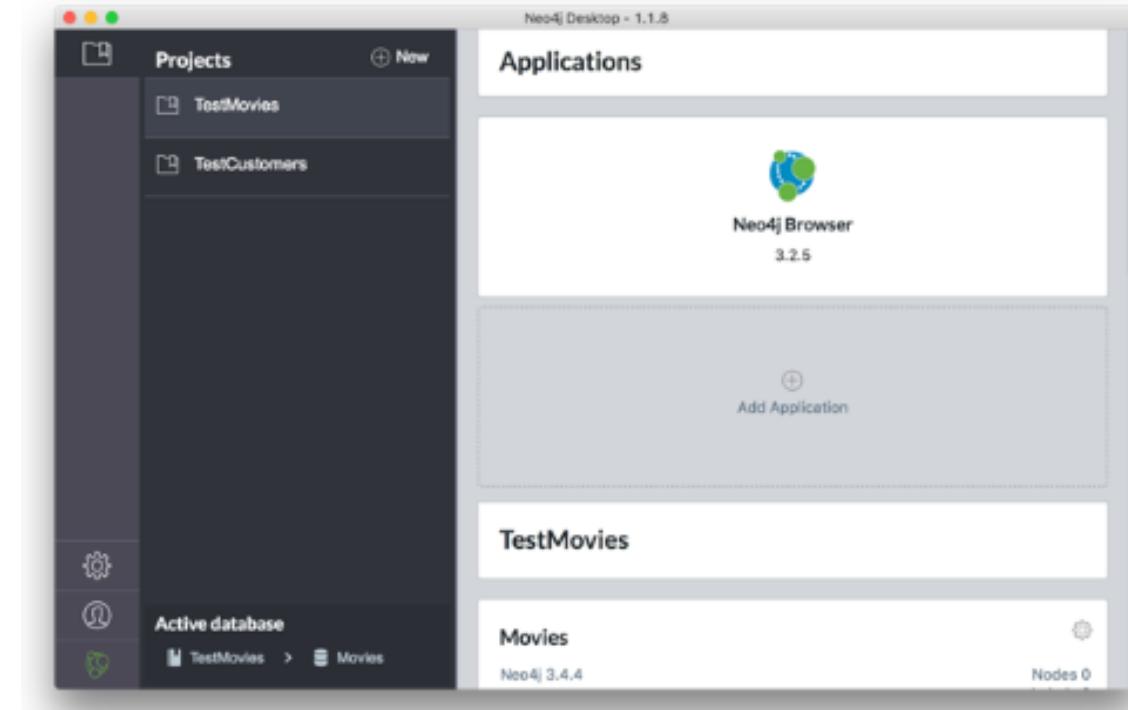
NEO4J DATABASE

- Neo4j Desktop - Ferramenta para acessar e interagir com o Neo4j
- A ferramenta já vem com o Neo4j
- Permite
 - Gerenciar schemas
 - Executar Queries
 - Visualizar os grafos
 - Importar/Exportar dados



NEO4J DATABASE

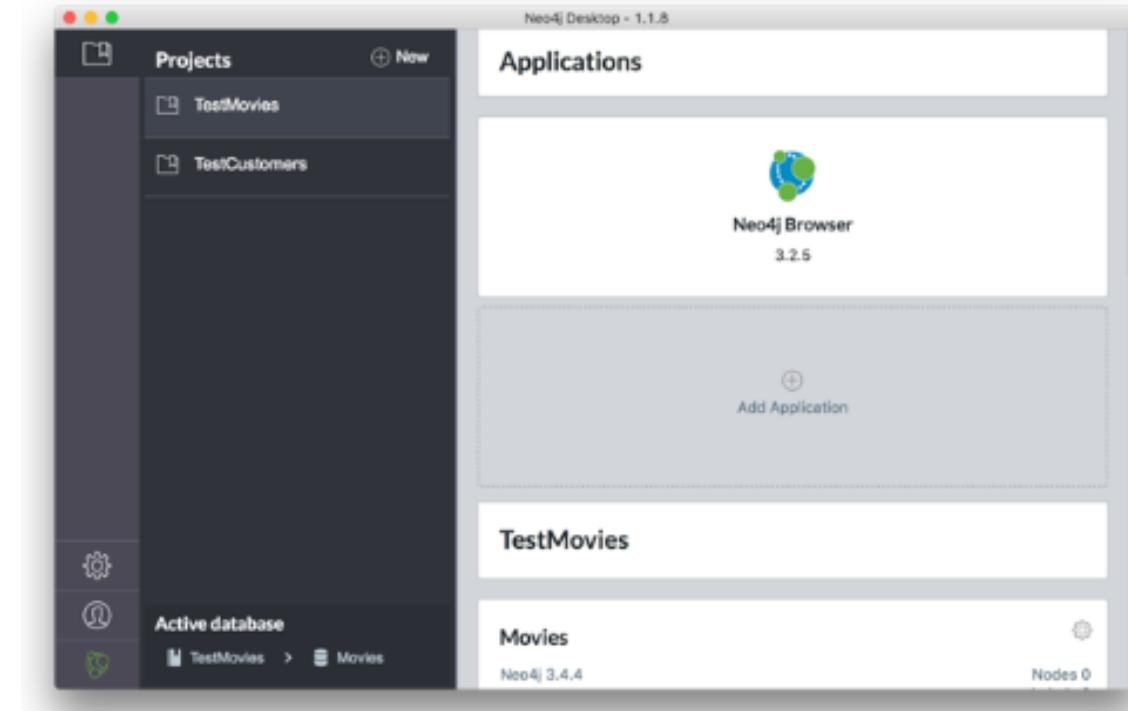
- Baixando e instalado o Neo4j Desktop
- <https://neo4j.com/download-center/#desktop>
- Instale o Neo4j Desktop
- Em projeto, crie um novo Banco de Grafo
- Inicie o banco
- Clique em Neo4j Browser



NEO4J DATABASE

- Exercícios:
 - Abrir o Neo4j Browser

:play movie graph



Cypher

O seu novo “SQL”

CYPHER

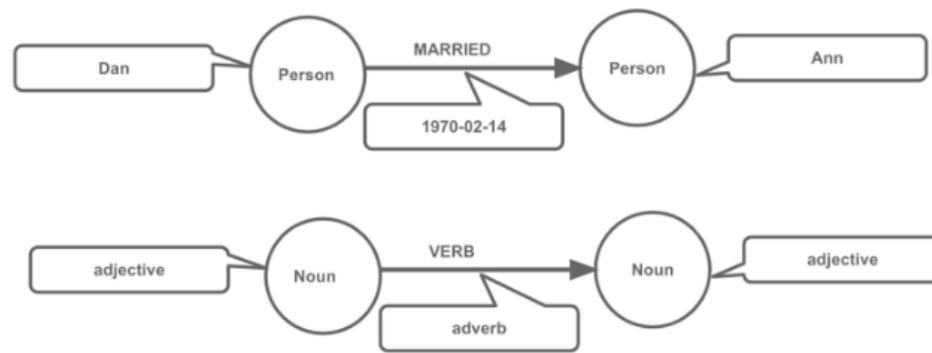
- Linguagem declarativa para operar sobre o banco de dados (grafo);
- Desenhada para lembrar um desenho, tem uma inspiração ASCII Art
- A ideia é ser simples de ler para pessoas com queries auto explicáveis
- A filosofia é sempre expressar O QUE deve ser extraído do grafo não COMO os dados devem ser extraídos

```
(A) - [:LIKES] -> (B) , (A) - [:LIKES] -> (C) , (B) - [:LIKES] -> (C)
```

```
(A) - [:LIKES] -> (B) - [:LIKES] -> (C) <- [:LIKES] - (A)
```

CYPHER

- Na estrutura do grafo, é possível pensar como o mesmo tivesse uma estrutura de linguagem onde
 - Os vértices são os substantivos
 - As arestas indicam verbos
 - As propriedades de vértices são adjetivos para os substantivos
 - As propriedades de arestas são advérbios para os verbos



CYPHER

- Vértices (nodes)
 - Representados por parêntesis

`()` – vértice anônimo

`(n)` – vértice representado pela variável `n`

CYPHER - NODE

- Rótulos (Labels) – Os grafos são rotulados e os filtros se aplicam aos rótulos para otimizar buscas
- Exemplo de rótulos são Pessoa, Filme, Cidade, etc
- Para referenciar os vértices e rótulos, a seguinte sintaxe é usada :Rotulo em um vértice

```
( )  
(variable)  
(:Label)  
(variable:Label)  
(:Label1:Label2)  
(variable:Label1:Label2)
```



CYPHER – NODE E LABELS

```
()          // anonymous node not be referenced later in the query
(p)        // variable p, a reference to a node used later
(:Person)   // anonymous node of type Person
(p:Person)  // p, a reference to a node of type Person
(p:Actor:Director) // p, a reference to a node of types Actor and Director
```

CYPHER - MATCH

- Para obter informações, utilizamos a declaração **MATCH**
- Durante a execução, a engine do banco passa pelos grafos procurando o padrão definido pelo **MATCH**
- Como parte da query, sempre é necessário usar o **RETURN** para obter os vértices da busca

```
MATCH (variable) RETURN variable
```

```
MATCH (variable:Label)RETURN variable
```

CYPHER - MATCH

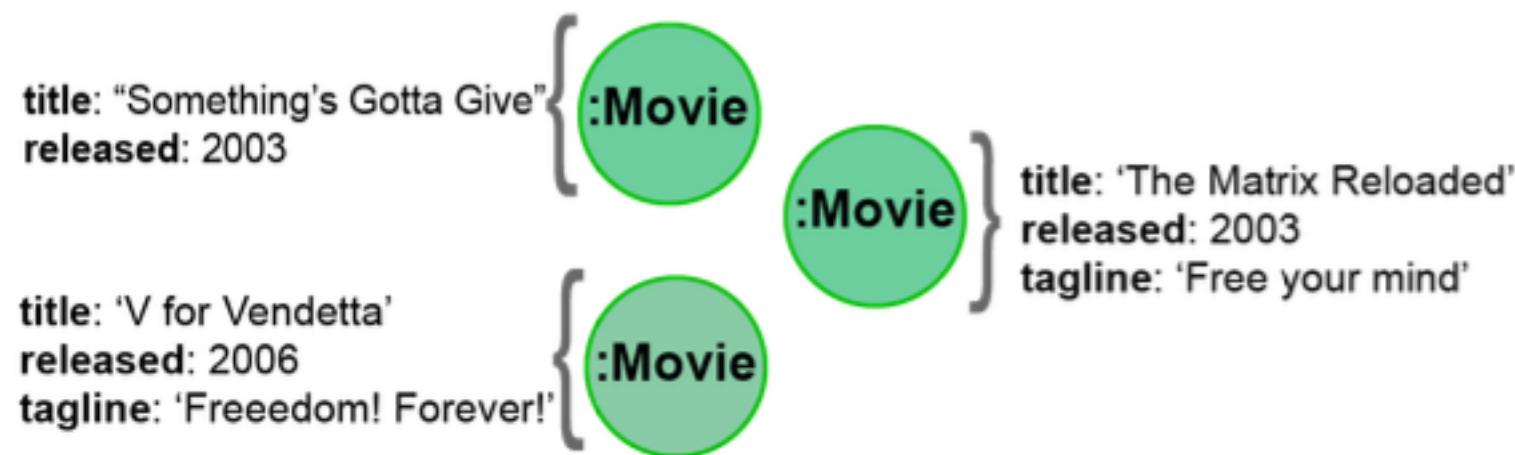
- Para obter informações, utilizamos a declaração **MATCH**
- Durante a execução, a engine do banco passa pelos grafos procurando o padrão definido pelo **MATCH**
- Como parte da query, sempre é necessário usar o **RETURN** para obter os vértices da busca

```
MATCH (n)          // returns all nodes in the graph
RETURN n
```

```
MATCH (p:Person)  // returns all Person nodes in the graph
RETURN p
```

CYPHER – NODE PROPERTIES

- Vértices e arestas podem ter propriedades no modelo chave/valor



CYPHER – NODE PROPERTIES

- Identificando as propriedades existentes no banco

```
$ CALL db.propertyKeys
```

propertyKey
"title"
"released"
"tagline"
"name"
"born"
"roles"
"summary"
"rating"
"id"
"share_link"
"favorite_count"
"display_name"

Started streaming 12 records in less than 1 ms and completed in less than 1 ms.

CYPHER – NODE PROPERTIES

- Filtro com propriedades do vértice

```
MATCH (variable {propertyKey: propertyName})
RETURN variable
```

```
MATCH (variable:Label {propertyKey: propertyName})
RETURN variable
```

```
MATCH (variable {propertyKey1: propertyName1, propertyKey2: propertyName2})
RETURN variable
```

```
MATCH (variable:Label {propertyKey: propertyName, propertyKey2: propertyName2})
RETURN variable
```

CYPHER – NODE PROPERTIES

- Filtro com propriedades do vértice

```
MATCH (p:Person {born: 1970})  
RETURN p
```

```
MATCH (m:Movie {released: 2003, tagline: 'Free your mind'})  
RETURN m
```

CYPHER – NODE PROPERTIES

- Retornando apenas valores de uma propriedade

```
MATCH (variable {prop1: value})
RETURN variable.prop2
```

```
MATCH (variable:Label {prop1: value})
RETURN variable.prop2
```

```
MATCH (variable:Label {prop1: value, prop2: value})
RETURN variable.prop3
```

```
MATCH (variable {prop1:value})
RETURN variable.prop2, variable.prop3
```

CYPHER – NODE PROPERTIES

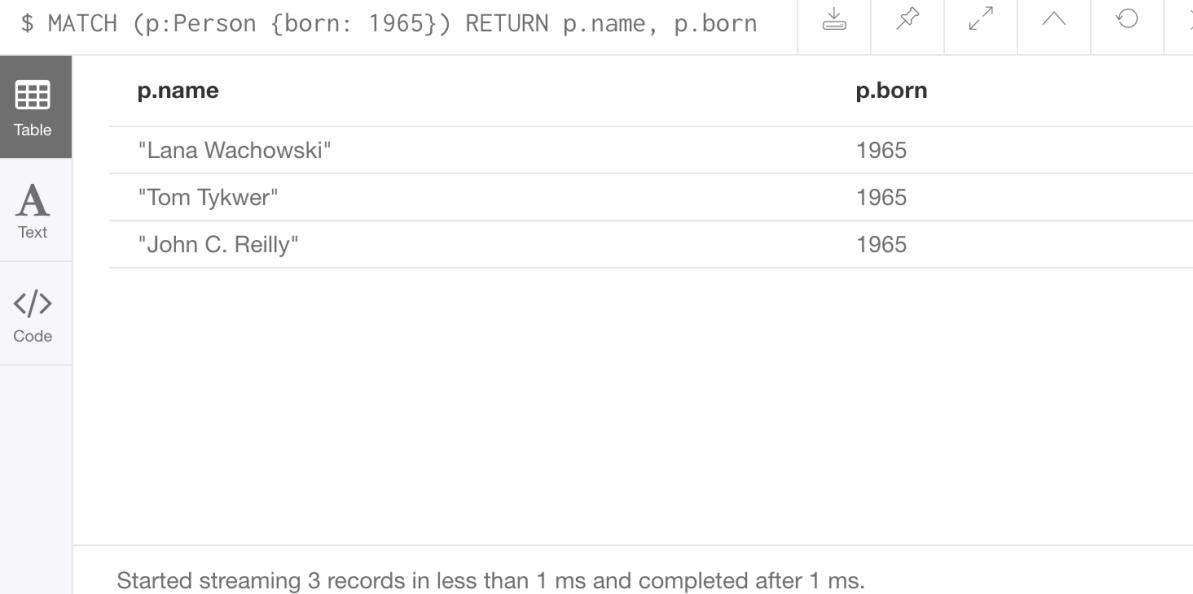
- Retornando apenas valores de uma propriedade

```
MATCH (p:Person {born: 1965})  
RETURN p.name, p.born
```

\$ MATCH (p:Person {born: 1965}) RETURN p.name, p.born

p.name	p.born
"Lana Wachowski"	1965
"Tom Tykwer"	1965
"John C. Reilly"	1965

Started streaming 3 records in less than 1 ms and completed after 1 ms.



The screenshot shows the Neo4j browser interface. On the left, there's a sidebar with three tabs: 'Table' (selected), 'Text', and 'Code'. The main area displays a table with two columns: 'p.name' and 'p.born'. The table contains three rows with the following data: 'Lana Wachowski' and '1965'; 'Tom Tykwer' and '1965'; and 'John C. Reilly' and '1965'. At the bottom of the table, a message says 'Started streaming 3 records in less than 1 ms and completed after 1 ms.' Above the table, a command line shows the Cypher query: '\$ MATCH (p:Person {born: 1965}) RETURN p.name, p.born'.

EXERCÍCIOS

A screenshot of a terminal window titled ':play intro-neo4j-exercises'. The window lists 16 exercises, each consisting of a blue button labeled 'Exercise' followed by a number and a brief description. The first exercise is highlighted with a red border. Navigation arrows are visible on the left and right sides of the list.

Exercise	Description
Exercise 1	— Retrieving Nodes
Exercise 2	— Filtering queries using property values
Exercise 3	— Filtering queries using relationships
Exercise 4	— Filtering queries using the WHERE clause
Exercise 5	— Controlling query processing
Exercise 6	— Controlling results returned
Exercise 7	— Working with Cypher data
Exercise 8	— Creating Nodes
Exercise 9	— Creating Relationships
Exercise 10	— Deleting Nodes and Relationships
Exercise 11	— Merging Data in the Graph
Exercise 12	— Using Cypher parameters
Exercise 13	— Analyzing and monitoring queries
Exercise 14	— Managing constraints and node keys
Exercise 15	— Managing indexes
Exercise 16	— Importing data

Relacionamentos

CYPHER – NODE PROPERTIES

- Retorno com ALIAS

```
MATCH (variable:Label {propertyKey1: propertyName1})
RETURN variable.propertyKey2 AS alias2
```

```
MATCH (p:Person {born: 1965})
RETURN p.name AS name, p.born AS `birth year`
```

CYPHER – RELACIONAMENTOS

- Os relacionamentos através das arestas do grafo é que fazem do Neo4J uma ferramenta interessante como banco de dados com estruturas complexas...
- A notação de arestas é dada por --, --> e <--

```
( )          // a node
( )--( )    // 2 nodes have some type of relationship
( )-->( )  // the first node has a relationship to the second node
( )<--( )  // the second node has a relationship to the first node
```

CYPHER – RELACIONAMENTOS

- Executando queries com relacionamentos

```
MATCH (node1)-[:REL_TYPE]->(node2)  
RETURN node1, node2
```

```
MATCH (node1)-[:REL_TYPEA | :REL_TYPEB]->(node2)  
RETURN node1, node2
```

node1

Vértice de interesse da busca que pode ter labels e propriedades para filtro.

:REL_TYPE

Tipo do relacionamento (aresta) entre o node1 e node 2
is the type (name) for the relationship. For this syntax the
relationship is from node1 to node2.

node2

Vértice de interesse da busca que deve estar associado ao node1
pelo relacionamento especificado. Também pode ser filtrado com
labels e propriedades

CYPHER – RELACIONAMENTOS

- Executando queries com relacionamentos

```
MATCH (p:Person)-[rel:ACTED_IN]->(m:Movie {  
    title: 'The Matrix'  
})  
  
RETURN p, rel, m
```

CYPHER – RELACIONAMENTOS

- Vértice anônimo



```
MATCH (p:Person)-[rel:ACTED_IN]->(:Movie {
    title: 'The Matrix'
})
RETURN p
```

CYPHER – RELACIONAMENTOS

- Relacionamento anônimo



```
MATCH (p:Person)--->(m:Movie {  
    title: 'The Matrix'  
})  
  
RETURN p, m
```

CYPHER – RELACIONAMENTOS

- Retornando o tipo do relacionamento (rótulo da aresta)

```
MATCH (p:Person)-[rel]->(:Movie  
{title:'The Matrix'})  
  
RETURN p.name, type(rel)
```

CYPHER – RELACIONAMENTOS

- Filtrando pelas propriedades da aresta

```
MATCH (p:Person)-[:REVIEWED {rating: 65}]->(:Movie  
{title: 'The Da Vinci Code'})  
  
RETURN p.name
```

CYPHER – RELACIONAMENTOS

- Utilizando a direção do grafo...
- “Quero a lista de todos os atores com os filmes que eles atuaram, junto com o diretor de cada filme”

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)  
RETURN a.name, m.title, d.name
```

CYPHER – WHERE

- O WHERE ajuda a fazer filtros com mais flexibilidade (bem no estilo SQL)

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie  
{released: 2008})  
  
RETURN p, m
```

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)  
WHERE m.released = 2008  
  
RETURN p, m
```

CYPHER – WHERE

- Com o WHERE, é simples executar AND, OR e utilizar operadores =>,<=

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released = 2008 OR m.released = 2009
```

```
RETURN p, m
```

CYPHER – WHERE

- O WHERE pode ser usado até mesmo para filtrar pelo rótulo (label)

```
MATCH (p)
WHERE p:Person
RETURN p.name
```

```
MATCH (p)-[:ACTED_IN]->(m)
WHERE p:Person AND m:Movie AND m.title='The Matrix'
```

```
RETURN p.name
```

CYPHER – WHERE

- Outro benefício é permitir o teste se determinada propriedade existe (schemaless)

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name='Jack Nicholson' AND exists(m.tagline)

RETURN m.title, m.tagline
```

CYPHER – WHERE

- WHERE para buscas com strings e expressões regulares

```
MATCH (p:Person)-[:ACTED_IN]->()
WHERE toLower(p.name) STARTS WITH 'michael'

RETURN p.name
```

```
MATCH (p:Person)
WHERE p.name =~ 'Tom.*'

RETURN p.name
```

CYPHER – WHERE

- WHERE para buscas em listas

```
MATCH (p:Person)
WHERE p.born IN [1965, 1970]

RETURN p.name as name, p.born as yearBorn
```

CYPHER – WHERE

- WHERE para buscas em propriedades
- Considere que queremos saber no nome do ator que atuou em um filme... Mas só lembramos do nome do personagem e do filme.
- Sabemos que no nosso banco, o papel do personagem é um relacionamento com a propriedade roles

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
WHERE 'Neo' IN r.roles AND m.title='The Matrix'

RETURN p.name
```

CYPHER – PATHS

- Paths são caminhos no grafo, que servem muitas vezes para serem processados depois

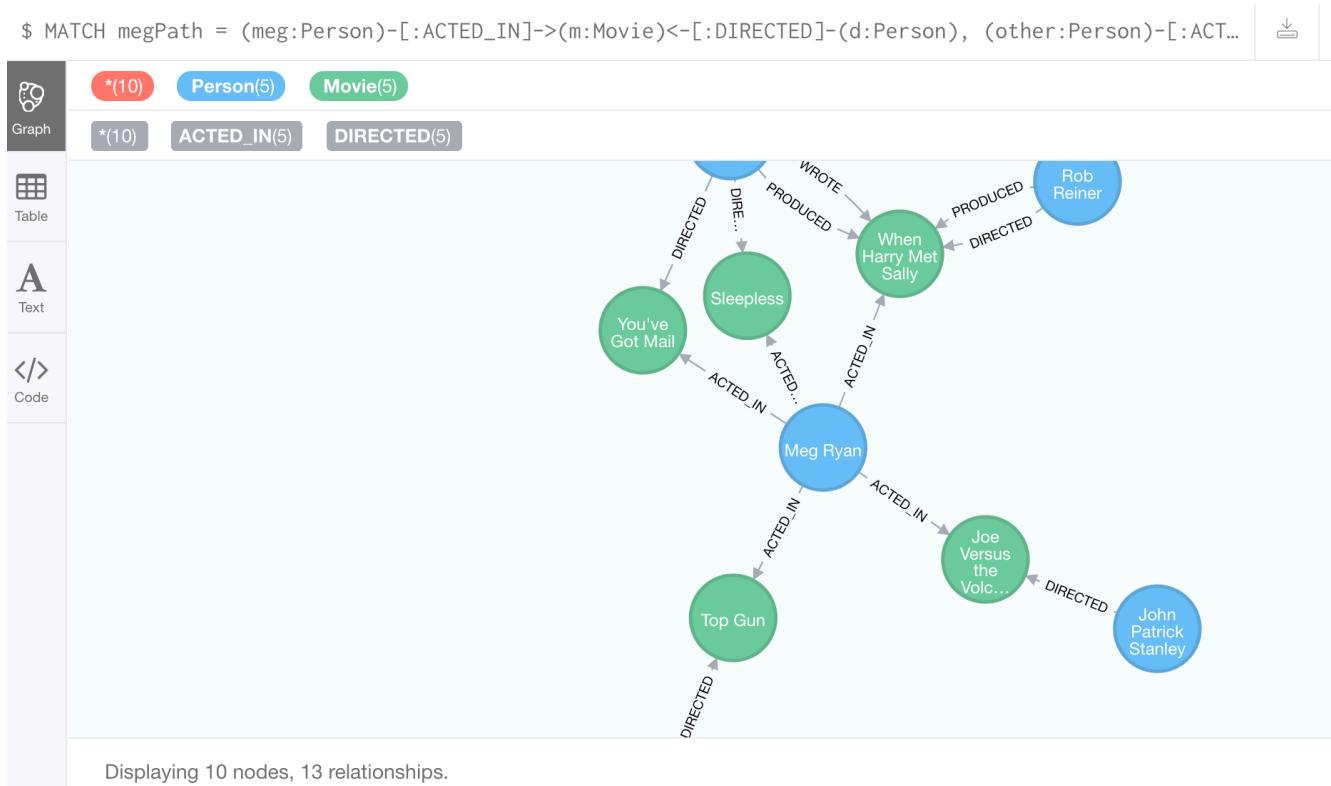
```
MATCH megPath = (meg:Person)-[:ACTED_IN]->(m:Movie)<-
[:DIRECTED]-(d:Person),
(other:Person)-[:ACTED_IN]->(m)
```

```
WHERE meg.name = 'Meg Ryan'
```

```
RETURN megPath
```

CYPHER – PATHS

- Paths são caminhos no grafo, que servem muitas vezes para serem processados depois



CYPHER – WHERE COM SALTOS (HOPS)

- A busca com saltos pode ser feito com o caractere * sobre o relacionamento
- É possível definir todos os caminhos possíveis :FOLLOWS*
- Um tamanho definido de tamanho do caminho :FOLLOWS*N
- E um tamanho variável como :FOLLOWS*N..M

```
MATCH (follower:Person)-[:FOLLOWS*2]->(p:Person)
WHERE follower.name = 'Paul Blythe'

RETURN p
```

CYPHER – SHORTEST PATH

- Uma função que já existe é o caminho mais curto

```
MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'A Few Good Men' AND
    m2.title = 'The Matrix'

RETURN p
```

CYPHER – AGREGADORES

- Quantas vezes o mesmo ator atuou em um filme com o mesmo diretor?

```
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)  
RETURN a.name, d.name, count(*)
```

CYPHER – OPTIONAL MATCH

- Faz o mesmo que o MATCH mas se não encontrar resultados, traz null
- É o equivalente ao OUTER JOIN

```
MATCH (p:Person) WHERE p.name STARTS WITH 'James'  
OPTIONAL MATCH (p)-[r:REVIEWED]->(m:Movie)  
  
RETURN p.name, type(r), m.title
```

\$ MATCH (p:Person) WHERE p.name STARTS WITH 'James' OPTIONAL MATCH (p)-[r:REVIEWED]->(m:Movie) RETU...

Table

A
Text

</>
Code

p.name	type(r)	m.title
"James Marshall"	null	null
"James L. Brooks"	null	null
"James Cromwell"	null	null
"James Thompson"	"REVIEWED"	"The Replacements"
"James Thompson"	"REVIEWED"	"The Da Vinci Code"

CYPHER – WITH

- Permite realizar processamento intermediário e/ou operações sobre o fluxo de dados
- Com o WITH, usa-se variáveis da parte anterior da query e novas para serem usadas na parte seguinte

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH a, count(a) AS numMovies, collect(m.title) as
movies
```

```
WHERE numMovies > 1 AND numMovies < 4
RETURN a.name, numMovies, movies
```

EXERCÍCIOS

\$:play intro-neo4j-exercises

\$:play intro-neo4j-exercises

Exercises:

- [Exercise 1](#) — Retrieving Nodes
- [Exercise 2](#) — Filtering queries using property values
- [Exercise 3](#) — Filtering queries using relationships
- [Exercise 4](#) — Filtering queries using the WHERE clause
- [Exercise 5](#) — Controlling query processing
- [Exercise 6](#) — Controlling results returned
- [Exercise 7](#) — Working with Cypher data
- [Exercise 8](#) — Creating Nodes
- [Exercise 9](#) — Creating Relationships
- [Exercise 10](#) — Deleting Nodes and Relationships
- [Exercise 11](#) — Merging Data in the Graph
- [Exercise 12](#) — Using Cypher parameters
- [Exercise 13](#) — Analyzing and monitoring queries
- [Exercise 14](#) — Managing constraints and node keys
- [Exercise 15](#) — Managing indexes
- [Exercise 16](#) — Importing data

4 / 4 < • • • • >

Controlando resultados

CYPHER – DISTINCT

- Igual em SQL, elimina duplicados

```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'

RETURN m.released, collect(DISTINCT m.title) AS movies
```

CYPHER – DISTINCT

- Pode ser utilizado com o WITH

```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'
WITH DISTINCT m

RETURN m.released, m.title
```

CYPHER – ORDER BY

- Ordenar os resultados (just like SQL)

```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'

RETURN m.released, collect(DISTINCT m.title) AS movies
ORDER BY m.released DESC
```

CYPHER – LIMIT

- Limitar a quantidade de resultados

```
MATCH (m:Movie)
RETURN m.title as title, m.released as year ORDER BY
m.released DESC LIMIT 10
```

CYPHER – COLLECT()

- collect() – transforma um resultado em uma lista

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH m, count(m) AS numCast, collect(a.name) as cast
RETURN m.title, cast, numCast ORDER BY size(cast)
```

CYPHER – DATE()

- Permite manipular datas pela query

```
MATCH (actor:Person)-[:ACTED_IN]->(:Movie)
WHERE exists(actor.born) // calculate the age
WITH DISTINCT actor, date().year - actor.born as age

RETURN actor.name, age as `age today`
ORDER BY actor.born DESC
```

EXERCÍCIOS

\$:play intro-neo4j-exercises

\$:play intro-neo4j-exercises

Exercises:

- [Exercise 1](#) — Retrieving Nodes
- [Exercise 2](#) — Filtering queries using property values
- [Exercise 3](#) — Filtering queries using relationships
- [Exercise 4](#) — Filtering queries using the WHERE clause
- [Exercise 5](#) — Controlling query processing
- [Exercise 6](#) — Controlling results returned
- [Exercise 7](#) — Working with Cypher data
- [Exercise 8](#) — Creating Nodes
- [Exercise 9](#) — Creating Relationships
- [Exercise 10](#) — Deleting Nodes and Relationships
- [Exercise 11](#) — Merging Data in the Graph
- [Exercise 12](#) — Using Cypher parameters
- [Exercise 13](#) — Analyzing and monitoring queries
- [Exercise 14](#) — Managing constraints and node keys
- [Exercise 15](#) — Managing indexes
- [Exercise 16](#) — Importing data

4 / 4 < • • • • >

Criando grafos no Neo4J

CYPHER – VÉRTICES (NODES)

- Criar novos vértices pode ser feito de várias formas:

```
CREATE (optionalVariable optionalLabels {optionalProperties})
```

```
CREATE (:Movie {title: 'Batman Begins'})
```

```
CREATE (:Movie:Action {title: 'Batman Begins'})
```

```
CREATE (m:Movie:Action {title: 'Batman Begins'})
```

```
CREATE (m:Movie:Action {title: ' Batman Begins'})  
RETURN m.title
```

CYPHER – VÉRTICES (NODES)

- É possível criar múltiplos vértices ao mesmo tempo

```
CREATE
```

```
(:Person {name: 'Michael Caine', born: 1933}),  
(:Person {name: 'Liam Neeson', born: 1952}),  
(:Person {name: 'Katie Holmes', born: 1978}),  
(:Person {name: 'Benjamin Melniker', born: 1913})
```

CYPHER – VÉRTICES (NODES)

- Adicionar ou remover labels (Update)

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m:Action

RETURN labels(m)
```

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
REMOVE m:Action

RETURN labels(m)
```

CYPHER – VÉRTICES (NODES)

- Adicionar ou remover propriedades (Update)

```
SET x.propertyName = value
SET x.propertyName1 = value1, x.propertyName2 = value2
SET x = {propertyName1: value1, propertyName2: value2}
SET x += {propertyName1: value1, propertyName2: value2}
```

```
REMOVE x.propertyName
SET x.propertyName = null
```



CYPHER – VÉRTICES (NODES)

- Adicionar ou remover propriedades (Update)

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m.released = 2005, m.lengthInMinutes = 140
RETURN m
```

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m.grossMillions = null
REMOVE m.videoFormat
RETURN m
```

CYPHER – RELACIONAMENTOS (ARESTAS)

- Adicionar relacionamentos

```
CREATE (x)-[:REL_TYPE]->(y)
CREATE (x)<-[:REL_TYPE]-(y)
```

```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Michael Caine' AND m.title = 'Batman Begins'
CREATE (a)-[:ACTED_IN]->(m)

RETURN a, m
```

CYPHER – RELACIONAMENTOS (ARESTAS)

- Remover relacionamentos

```
DELETE x
```

```
MATCH (a:Person)-[r]->(m:Movie)
WHERE a.name = 'Michael Caine' AND m.title = 'Batman Begins'
DELETE r

RETURN a, m
```

CYPHER – RELACIONAMENTOS (ARESTAS)

- Adicionar propriedades aos relacionamentos

```
SET r.propertyName = value
SET r.propertyName1 = value1, r.propertyName2 = value2
SET r = {propertyName1: value1, propertyName2: value2}
SET r += {propertyName1: value1, propertyName2: value2}
```

```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
CREATE (a)-[rel:ACTED_IN]->(m)
SET rel.roles = ['Bruce Wayne', 'Batman']
RETURN a, m
```

CYPHER – RELACIONAMENTOS (ARESTAS)

- Remover propriedades aos relacionamentos

```
MATCH (a:Person)-[rel:ACTED_IN]->(m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
REMOVE rel.roles

RETURN a, rel, m
```

EXERCÍCIOS

A screenshot of a terminal window titled ':play intro-neo4j-exercises'. The window displays a list of 16 exercises, each with a blue button labeled '(E) Exercise [number]' followed by a brief description. The exercises are arranged in two columns. The first column contains exercises 1 through 8. The second column contains exercises 9 through 16. Exercises 8 and 9 are highlighted with a red border. The terminal window has standard OS X-style window controls at the top and bottom.

Exercise	Description
(E) Exercise 1	— Retrieving Nodes
(E) Exercise 2	— Filtering queries using property values
(E) Exercise 3	— Filtering queries using relationships
(E) Exercise 4	— Filtering queries using the WHERE clause
(E) Exercise 5	— Controlling query processing
(E) Exercise 6	— Controlling results returned
(E) Exercise 7	— Working with Cypher data
(E) Exercise 8	— Creating Nodes
(E) Exercise 9	— Creating Relationships
(E) Exercise 10	— Deleting Nodes and Relationships
(E) Exercise 11	— Merging Data in the Graph
(E) Exercise 12	— Using Cypher parameters
(E) Exercise 13	— Analyzing and monitoring queries
(E) Exercise 14	— Managing constraints and node keys
(E) Exercise 15	— Managing indexes
(E) Exercise 16	— Importing data



Marcio.Jasinski@senior.com.br