

# noSQL

Mestrado Integrado em Engenharia Informática

<https://hpeixoto.me/class/nosql>

Hugo Peixoto  
[hpeixoto@di.uminho.pt](mailto:hpeixoto@di.uminho.pt)

2020/2021

# noSQL

PL08 - Introdução às Bases de Dados de Grafos

# → Sumário

---

 Introdução às bases de dados de grafos

 Instalação de container do Neo4J

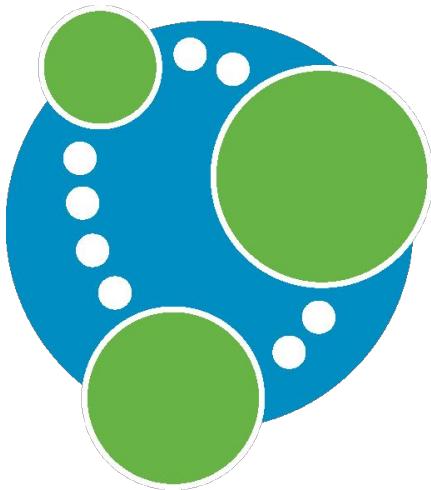
 Cypher

 Lab + FE06 - Ficha de exercícios 06

# Introdução às bases de dados de grafos



# Introdução às bases de dados de grafos



# neo4j

# 🔗 Introdução às bases de dados de grafos

Propriedades do Modelo de Grafos:

- Contém **nós** e **relações**
- Nós podem conter **propriedades** (par chave:valor)
- Nós podem ser identificados com um ou mais **labels**
- Relações podem ser nomeadas (**label**) e direcionadas e têm sempre um nó de início e um nó de fim
- Relações podem conter propriedades (par chave:valor)

# Share Introdução às bases de dados de grafos

**Nós:** (São equivalentes a vértices na teoria dos grafos) São a estrutura de dados de referência e estão interligados pelas relações. Um nó pode conter mais do que uma descrição e tem propriedades.

title = 'Forrest Gump'

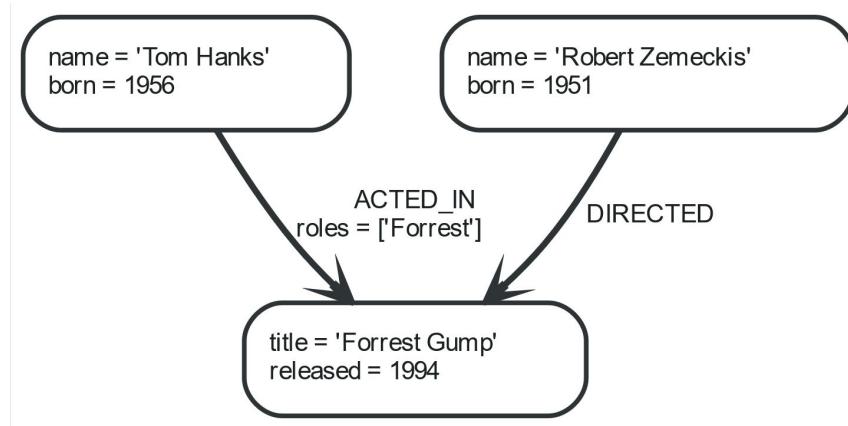
name = 'Tom Hanks'  
born = 1956

title = 'Forrest Gump'  
released = 1994

name = 'Robert Zemeckis'  
born = 1951

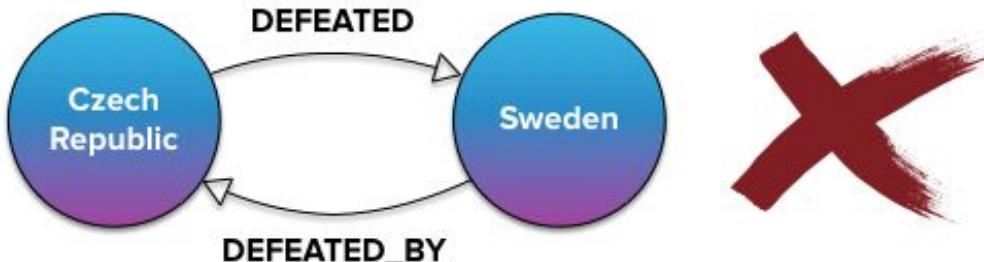
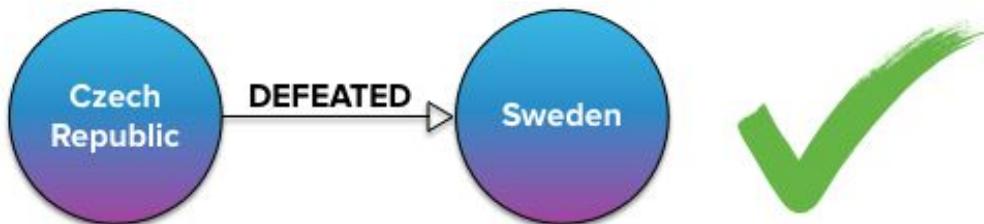
# Introdução às bases de dados de grafos

**Relações:** (equivalentes às arestas na teoria dos grafos) Uma relação liga dois nós, que por sua vez podem ter múltiplas relações. Uma relação pode ter uma ou mais propriedades.



# 🔗 Introdução às bases de dados de grafos

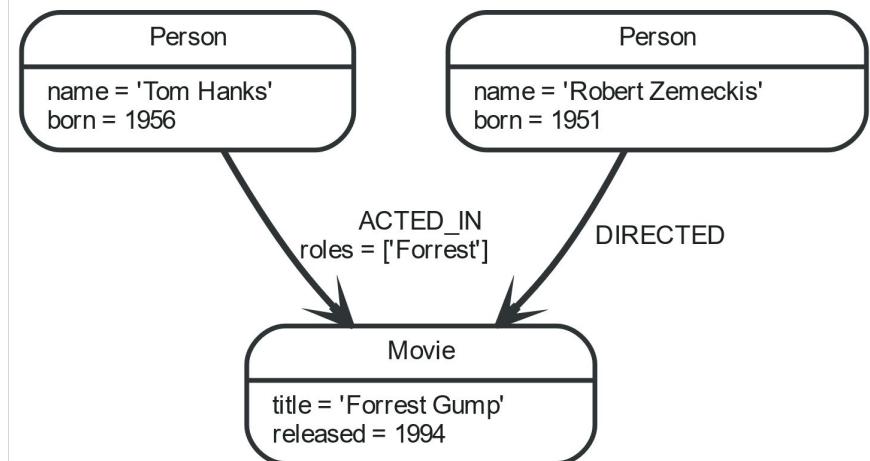
## Relações



# 🔗 Introdução às bases de dados de grafos

**Labels:** São usadas para agrupar nós. A cada nó podem ser atribuído vários labels. Os labels são indexados para melhorar a performance de pesquisa de nós.

É uma forma de agrupar nós. Todos os nós com o mesmo label pertencem ao mesmo conjunto.



# 🔗 Introdução às bases de dados de grafos

**Propriedades:** São aplicadas quer a nós quer a relações. São do tipo chave:valor o que significa que as propriedades podem assumir diferentes tipos (string, número, booleano).

## Types:

Number (integer, float)

String

Boolean

Spatial type: Point

Temporal types: Date, Time, LocalTime, DateTime, LocalDateTime, Duration

O tipo **null** não é válido.

# Share Introdução às bases de dados de grafos

## Vantagens:

**Performance:** Nas bases de dados relacionais a performance sobe quando o número de relações aumenta. Nas bases de dados de grafos, o objetivo é manter a performance mesmo quando o número de relações aumenta.

**Flexibilidade:** A estrutura e o esquema das bases de dados de grafos podem ser facilmente ajustadas conforme as mutações de uma determinada aplicação. Adicionalmente é possível alterar a estrutura de dados sem danificar as funcionalidades existentes.

**Agilidade:** A estrutura de dados é fácil de atualizar, desta forma o armazenamento de dados pode evoluir conforme a aplicação evolui.

# 🔗 Introdução às bases de dados de grafos

## Bases de dados relacionais:

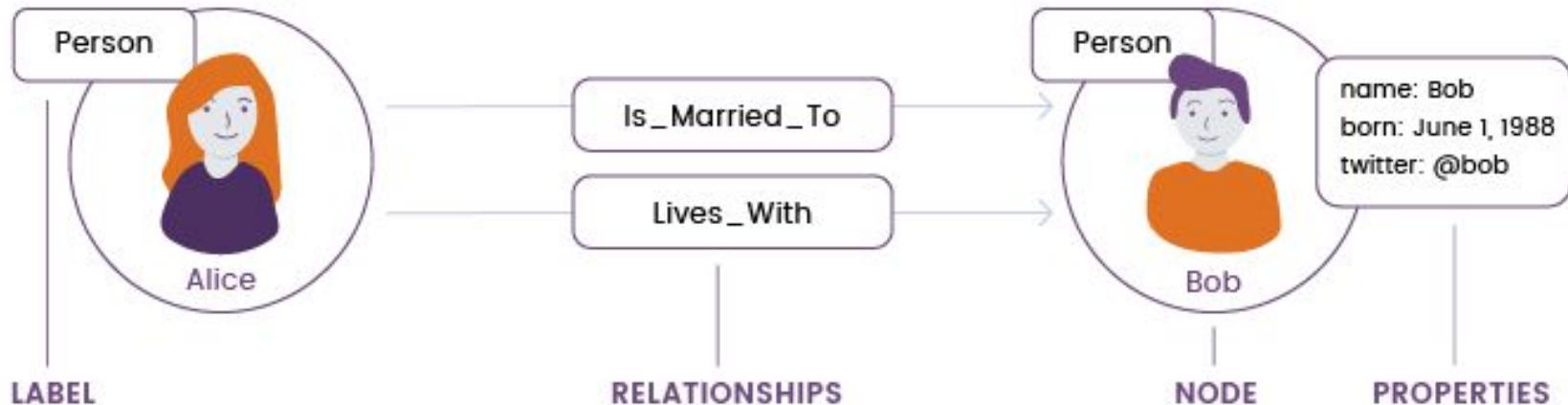
As relações existem em bases de dados Relacionais, porém esta estrutura é mais vincada no momento da modelação, com os joins de tabelas.

Porém com o aumentar dos dados, com o aumento da complexidade e com a falta de uniformização as bases de dados relacionais começam a ficar mais inundadas de joins complexos e de tabelas com nulos e validações necessárias.

Tabelas de join adicionam complexidade. Combinam dados da aplicação com metadados. Chaves estrangeiras fazem com que exista maior complexidade no desenvolvimento e manutenção para garantir que a base de dados continua funcional.

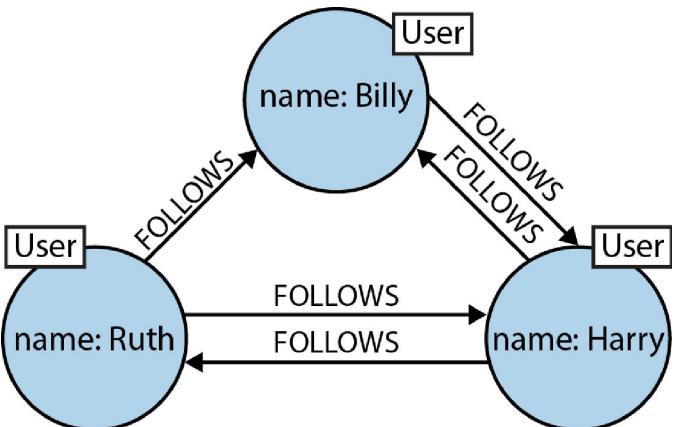
# Share Introdução às bases de dados de grafos

- As bases de dados de grafos são uma estrutura de dados genérica capaz de representar de forma simples, elegante e altamente acessível qualquer tipo de dados. Exemplo:

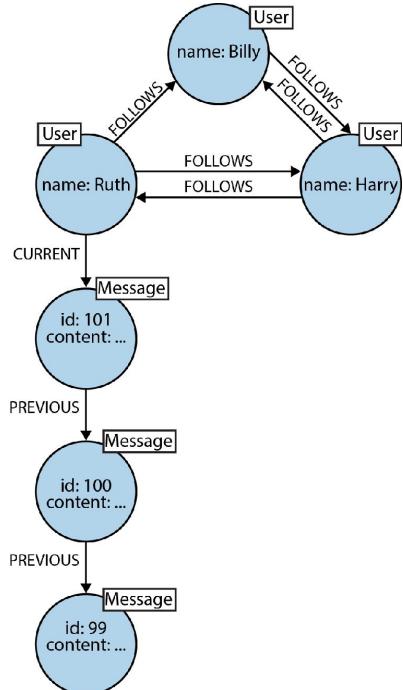


# Introdução às bases de dados de grafos

Exemplo, rede social Twitter:



Rede de utilizadores e relações



Timeline de posts

# Introdução às bases de dados de grafos

## Bases de dados relacionais:

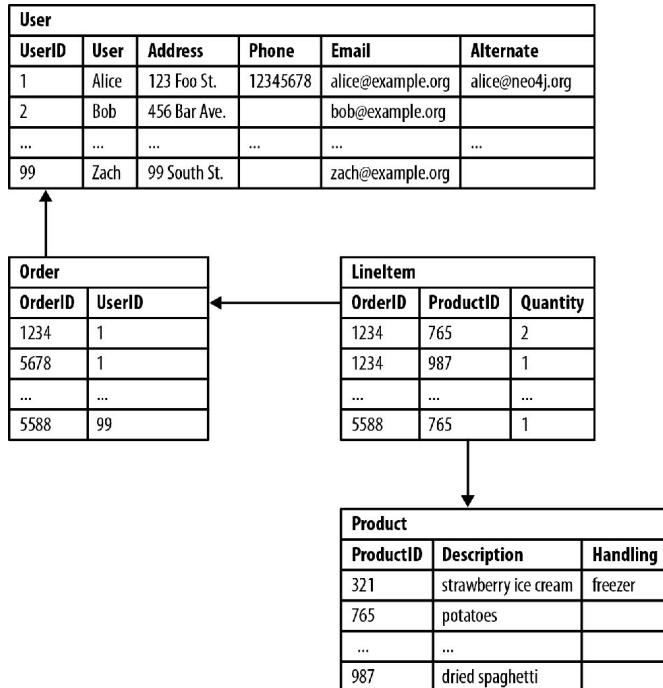
Necessários vários joins para conseguir descobrir que produtos um cliente comprou. Ainda mais complexo se a query for inversa.

Por exemplo:

Que produtos um cliente comprou?

Vs

Que clientes compraram este produto?



# 🔗 Introdução às bases de dados de grafos

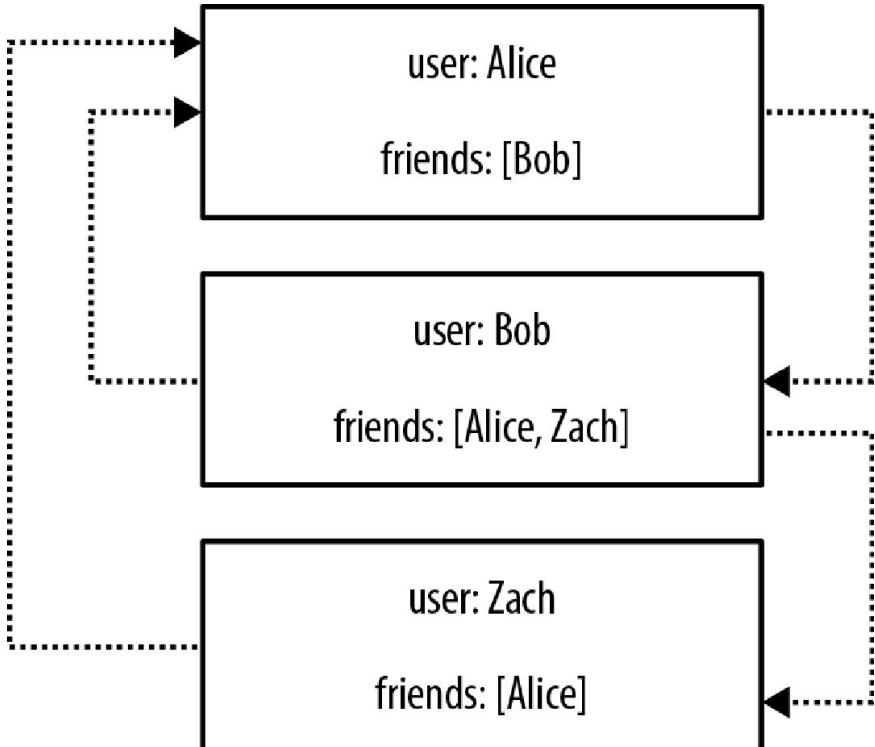
Bases de dados Documentais:

"Quem são os amigos do BOB?"

nem sempre é similar a:

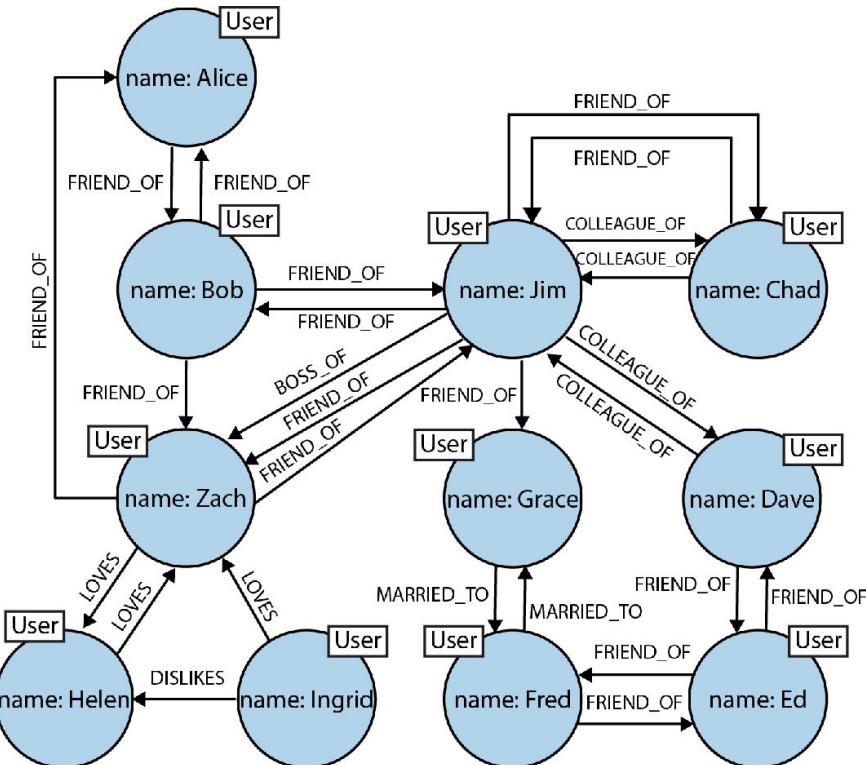
"De que o Bob é amigo?"

Apenas um scan a toda a coleção poderá navegar por todos os documentos à procura do atributo "Friends" que contenha o Bob, para responder à segunda questão.



# Introdução às bases de dados de grafos

Bases de dados de grafos:



# Share Introdução às bases de dados de grafos

	Neo4j	Relational databases	NoSQL databases
Data storage	Graph storage structure	Fixed, predefined tables with rows and columns	Connected data not supported at the database level
Data modeling	Flexible data model	Database model must be developed from a logical model	Not suitable for enterprise architectures
Query performance	Great performance regardless of number and depth of connections	Data processing speed slows with growing number of joins	Relationships must be created at the application level
Query language	Cypher: native graph query language	SQL: complexity grows as the number of joins increases	Different languages are used but none is tailored to express relationships
Transaction support	Retains ACID transactions	ACID transaction support	BASE transactions prove unreliable for data relationships
Processing at scale	Inherently scalable for pattern-based queries	Scales through replication, but it's costly	Scalable, but data integrity isn't trustworthy

# Introdução às bases de dados de grafos

## Casos de uso:

- Deteção e análise de fraudes
- Monitorização de redes e estruturas de base de dados
- Sistemas de recomendação
- Redes Sociais
- Gestão de identidades e acessos

# ✖ Instalação de container do Neo4J



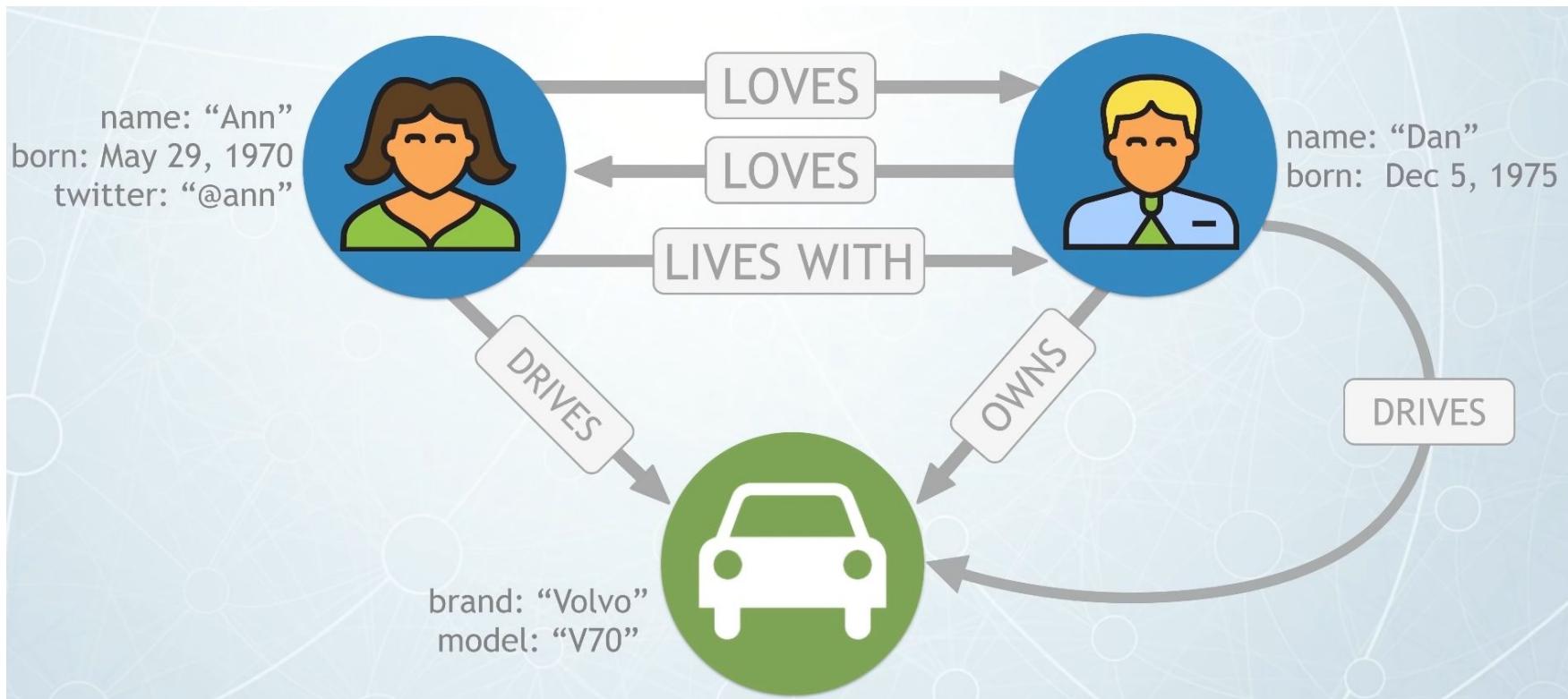
# 🔧 Instalação de container do Neo4J

```
neo4j:  
  image: neo4j:latest  
  container_name: neo4j  
  ports:  
    - "7687:7687"  
    - "7474:7474"  
  volumes:  
    - ./data/neo4j/data:/data  
    - ./data/neo4j/logs:/logs  
    - ./data/neo4j/import:/var/lib/neo4j/import  
    - ./data/neo4j/plugins:/plugins  
  environment:  
    - NEO4J_AUTH=neo4j/test
```

A small icon of a database server with three cylinders and a network connection symbol.

# Cypher



 Cypher

 Cypher

Who drives a car owned by a lover?

```
MATCH
```

```
  (p1:Person) - [:DRIVES] -> (c:Car) - [:OWNED_BY] -> (p2:Person) <- [:Loves]
  ] - (p1)
```

```
RETURN
```

```
p1
```



## Componentes da Query:

```
MATCH (p:Person) - [:ACTED_IN] -> (:Movie)
RETURN p
```

`MATCH` and `RETURN` are Cypher keywords

`p` is a variable

`:Movie` is a node label

`:ACTED_IN` is a relationship



# Cypher

## Case sensitive:

Labels dos nós

Tipos de relação

Chaves das propriedades

## Case insensitive:

Keywords do Cypher (return, match, where, etc)

 Cypher

AsciiArt para os nós:

Nós são representados por parênteses:

() ou (p)

Labels, ou tags, começam com : e agrupam nós por roles ou tipos:

(p:Person:Mammal)

Nós têm propriedades:

(p:Person {name : 'Veronica'})



# Cypher

## AsciiArt para os nós:

```
()  
(matrix)  
(:Movie)  
(matrix:Movie)  
(matrix:Movie {title: "The Matrix"})  
(matrix:Movie {title: "The Matrix", released: 1997})
```

) representa um nó descaracterizado e anónimo. Para ser utilizado em outra parte da query deve ser adicionada uma variável. Por exemplo, (matrix). A variável é apenas utilizável numa execução.

Ao utilizar o label "Movie" em conjunto com os : é declarado o tipo do nó. Assim é possível filtrar os nós que irão ser selecionados pela query.



# Cypher

## AsciiArt para as relações:

Relações são representadas através de hifens ou parênteses retos:

--> or - [h:HIRED] ->

A direção das relações é representada pelo sinal de maior ou menor:

(p1) - [:HIRED] -> (p2) or (p1) <- [:HIRED] - (p2)

As relações também têm propriedades:

- [:HIRED {type: 'full-time'}] ->



## AsciiArt para as relações:

```
-->  
-[role]->  
-[:ACTED_IN]->  
-[role:ACTED_IN]->  
-[role:ACTED_IN {roles: ["Neo"]}]->
```

O Cypher usa um par de hífens para representar uma relação não direcionada. As relações direcionadas são representadas através do sinal de menor ou maior. E os parênteses retos são usados para definir os detalhes da relação.



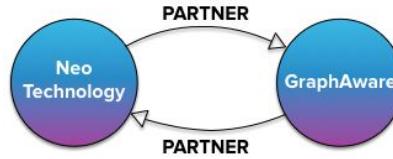
# Cypher

AsciiArt para as relações:

```
MATCH (neo) - [:PARTNER] - (partner)
```

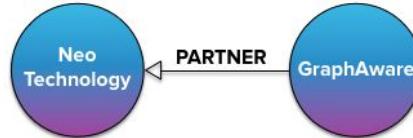
O resultado é igual a executar as duas seguintes operações:

```
MATCH (neo) - [:PARTNER] -> (partner) and MATCH (neo) <- [:PARTNER] - (partner)
```



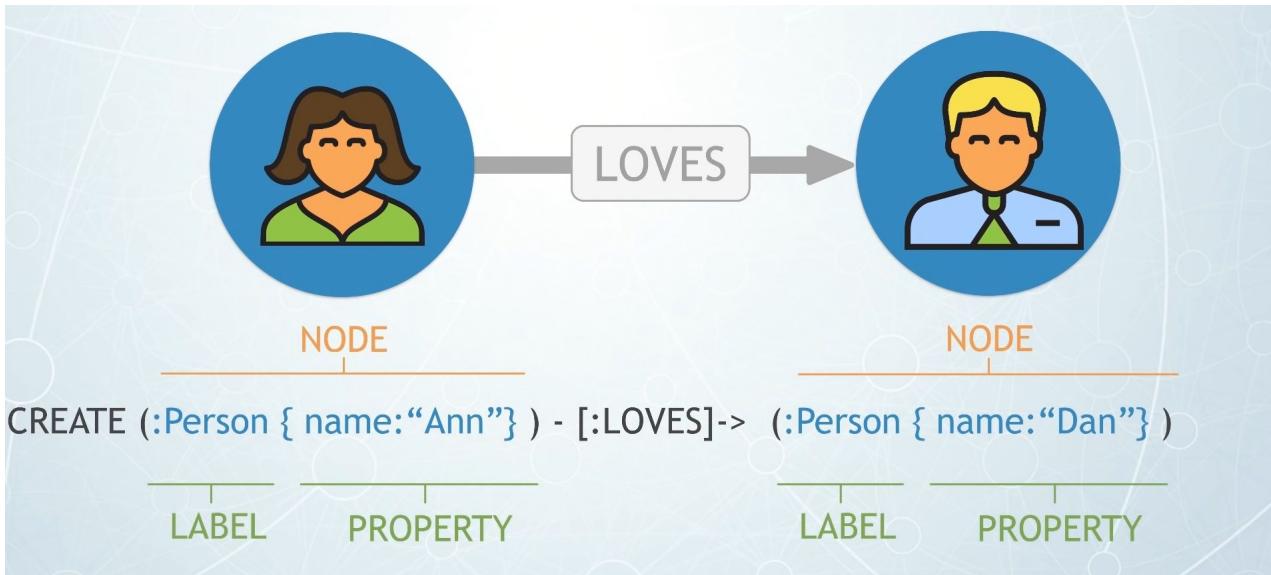
A forma mais eficiente é representar a relação de partner

Escolhendo de forma aleatória a direção



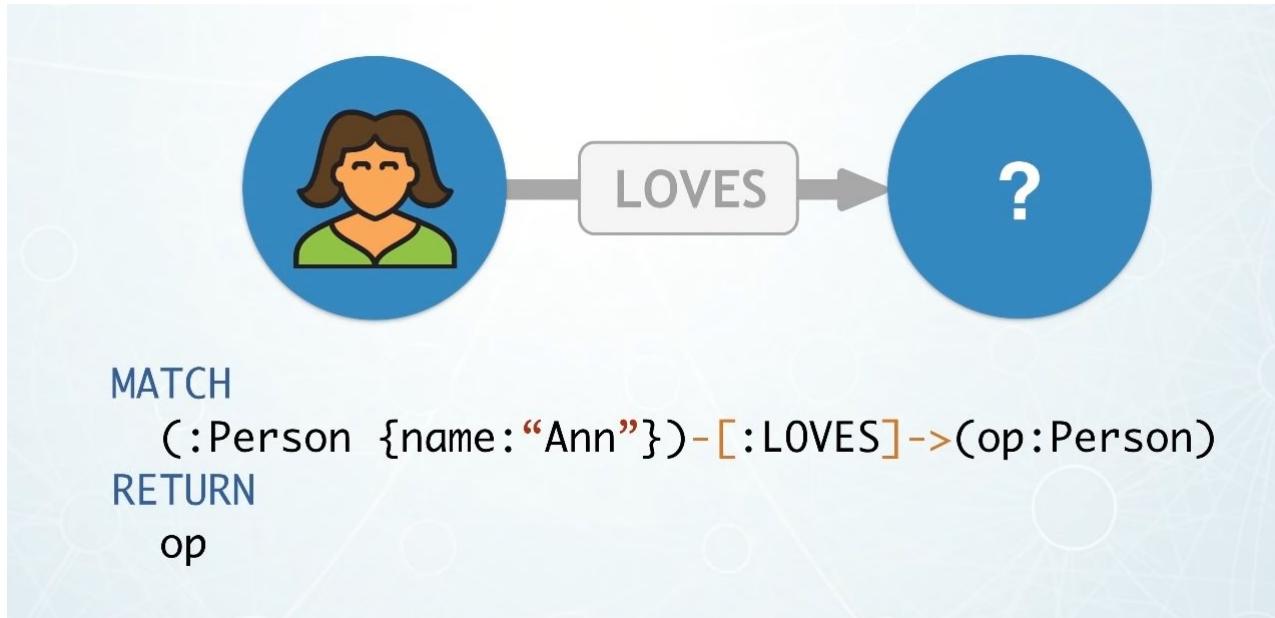
 Cypher

## Criar dados



 Cypher

"De quem a Ann gosta?"



 Cypher

Qual o carro da Ann?

```
MATCH
(:Person {name: 'Ann'})-[:DRIVES]->(c:Car)
RETURN
c
```





Qual o carro da Ann?

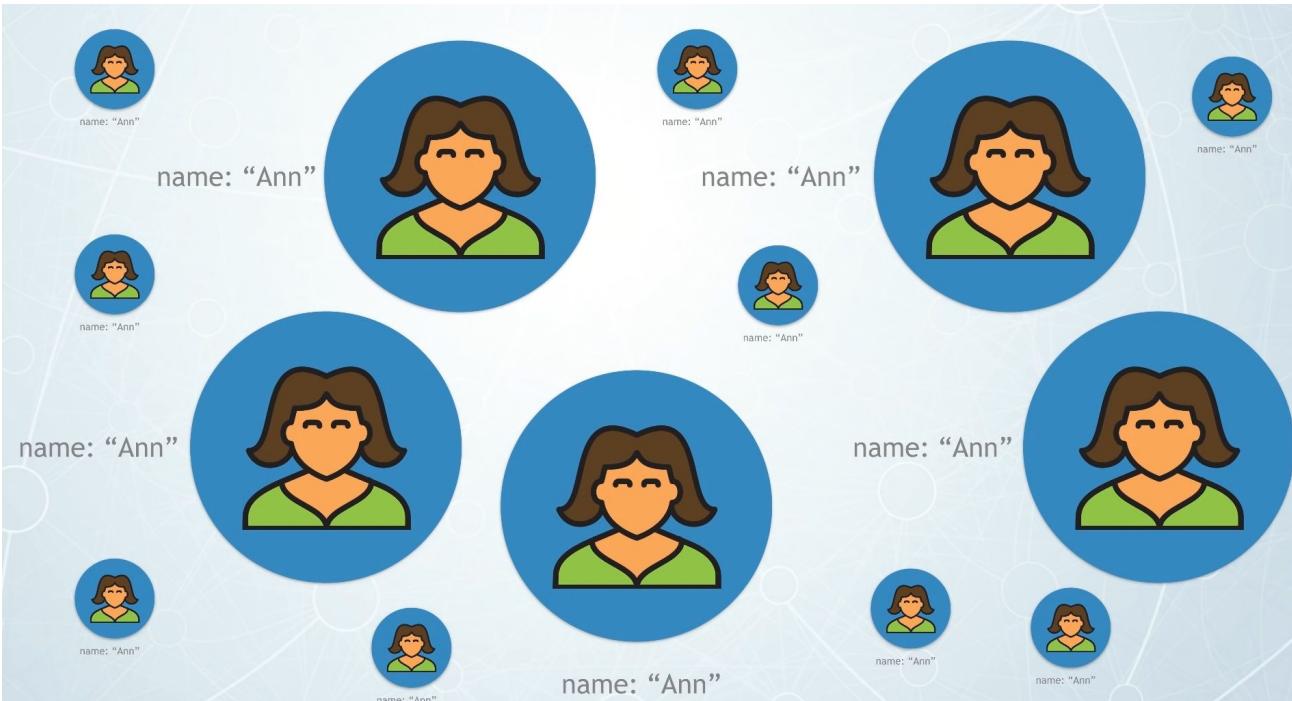
```
MATCH
  (a:Person)-[:DRIVES]->(c:Car)
WHERE
  a.name='Ann'
RETURN
  c
```





# Cypher

## Singularidade

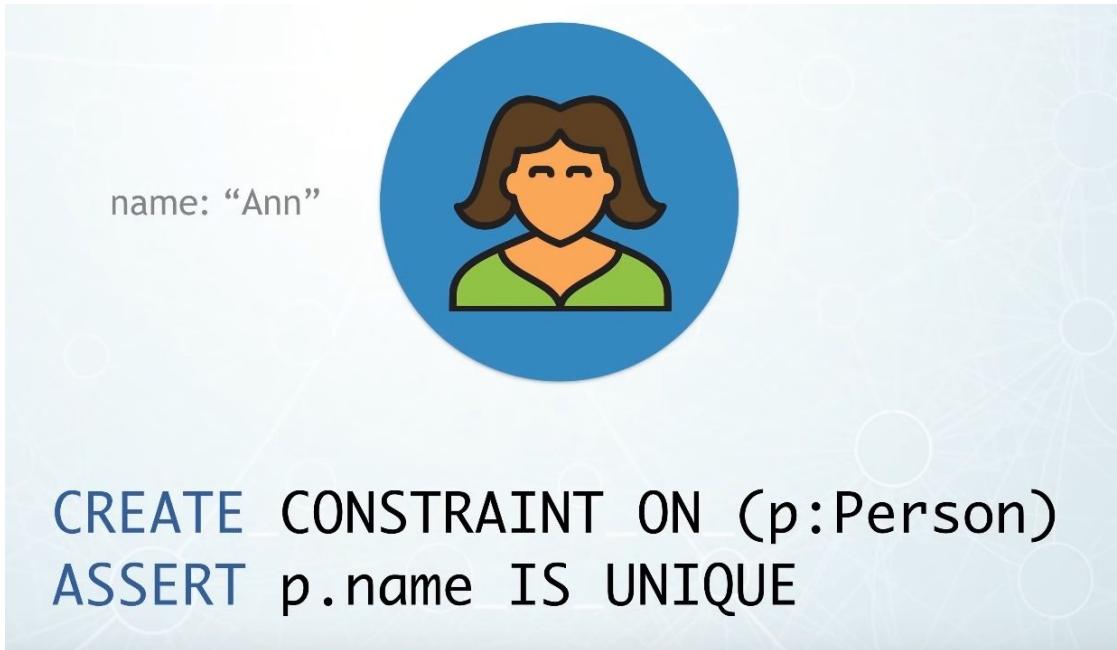
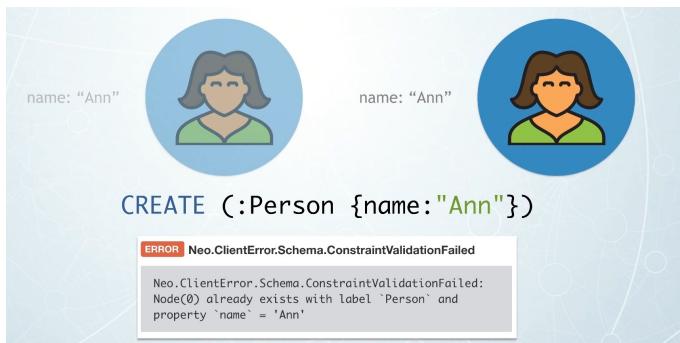




# Cypher

## Singularidade

Apenas pode existir uma Ann!





## Criar e pesquisar nós:

```
CREATE (me:Person {name: "My Name" })
RETURN me
```

```
MATCH (me:Person )
WHERE me.name="My Name"
RETURN me.name
```

## Short-hand syntax:

```
MATCH (me:Person {name:"My Name"})
RETURN me.name
```

A small icon of two stacked databases with a keyhole on the left side.

# Cypher

## Pesquisar todos os nós:

```
MATCH  (n)
RETURN n
```

Numa base de dados normal irá devolver muitos dados. Convém limitar a pesquisar a 100 nós por exemplo.

Esta pesquisa percorre todos os nós e verifica se fazem match com o padrão n. Dado que é apenas uma variável e não foi atribuída nenhuma propriedade ou label, irá devolver todos os nós na base de dados.



# Cypher

## Operadores:

=, <>, <, >, <=, >=, IS NULL, IS NOT NULL

## Expressões regulares:

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
WHERE p.name =~ "K.+"
      OR m.released >= 2000
RETURN p, r, m
```



## Executar queries:

```
:play movie-graph
```

```
MATCH (p:Person {name: "Tom Hanks"}) -[r:ACTED_IN|DIRECTED]-(m:Movie)
RETURN p,r,m
```

```
MATCH (p:Person {name: "Tom Hanks"}) -[r:ACTED_IN|DIRECTED]-(m:Movie)
RETURN p.name, type(r), m.title
```

## Filtrar resultados:

```
MATCH (m:Movie {title: "The Matrix"})
RETURN m
```

```
MATCH (m:Movie)
WHERE m.title = "The Matrix"
RETURN m
```



## Filtrar resultados comparando propriedades de diferentes nós:

Por exemplo, listar todos os atores que contracenaram com o Tom Hanks e que são mais velhos que ele.

```
MATCH (tom:Person)-[:ACTED_IN]->()-<-[ :ACTED_IN]-(actor:Person)
WHERE tom.name = "Tom Hanks"
AND actor.born < tom.born
RETURN actor.name AS Name
```



# Cypher

## Adicionar nós:

Criar o filme Mystic River:

```
CREATE (movie:Movie {title: "Mistic River", released:1993 })
```



# Cypher

## Adicionar propriedades:

Adicionar uma tagline ao filme Mystic River:

```
MATCH (movie:Movie)
WHERE movie.title = "Mystic River"
SET movie.tagline = "We bury our sins here, Dave. We wash them clean."
RETURN movie.title AS title, movie.tagline AS tagline
```

## Atualizar propriedades:

Atualizar a data de lançamento do filme (a sintaxe é a mesma!!):

```
MATCH (movie:Movie)
WHERE movie.title = "Mystic River"
SET movie.released = 2003
RETURN movie.title AS title, movie.released AS released
```



# Cypher

## Criar relações:

Encontrar o filme "Mystic River" e o Ator Kevin Bacon e criar uma relação entre eles.

```
MATCH (kevin:Person) WHERE kevin.name = "Kevin Bacon"
MATCH (mystic:Movie) WHERE mystic.title = "Mystic River"
CREATE (kevin)-[r:ACTED_IN {roles:["Sean"]}]>(mystic)
RETURN mystic, r, kevin
```

Criar uma pessoa na BD:

```
CREATE (me:Person {name:"Hugo Peixoto"}) RETURN me.name
```

Classificar o filme "Mystic River":

```
MATCH (kevin:Person), (movie:Movie)
WHERE me.name="Hugo Peixoto" AND movie.title="Mystic River"
CREATE (me)-[r:REVIEWED {rating:80, summary:"tragic character movie"}]->(movie)
RETURN me, r, movie
```



# Cypher

## Apagar nós:

Para apagar o nó e as relações que podem existir é necessário correr:

```
MATCH (p:Person {name:"Hugo Peixoto"})  
OPTIONAL MATCH (me)-[r]-()  
DELETE me, r
```

A cláusula "OPTIONAL MATCH" é usada para procurar o padrão descrito. Enquanto que os nulos são usados para partes desconhecidas do padrão. Podemos não saber com que nós estava relacionado. O comando DETACH DELETE foi criado para executar exatamente este tipo de operação:

```
MATCH (emil:Person {name:"Emil Eifrem"})  
DETACH DELETE emil
```

## Apagar todos os nós e relações:

```
MATCH (n)  
DETACH DELETE n
```



# Cypher

Ordenar, limitar e saltar resultados (Order, skip and limit):

```
MATCH (person:Person)
RETURN person.name, person.born
ORDER BY person.born
```

```
MATCH (actor:Person) -[:ACTED_IN]->(movie:Movie)
RETURN actor.name AS Actor, movie.title AS Movie
SKIP 10 LIMIT 10
```



# Cypher

## Índices

Um índice é uma cópia de alguns dados de forma a tornar as pesquisas mais rápidas. Porém isto comporta a necessidade e mais armazenamento e maior tempo de escrita (duplicação dos dados). Nem sempre é fácil decidir o que deve ser indexado.

Exemplo de criação de um índice:

```
CREATE INDEX ON :Movie(title)
```

Exemplo para eliminar um índice:

```
DROP INDEX ON :Movie(title)
```

# ✉ Laboratório



# Laboratório

Publicidade baseada em plataforma de e-commerce:

## Objetivo:

Oferecer os produtos ou serviços mais relevantes aos clientes

## Importação dos dados:

Copiar o conteúdo do ficheiro *email\_neo4j.txt* para o browser e correr

## Verificação:

```
MATCH (n)
```

```
RETURN n
```

# Laboratório

A base de dados:

## Labels dos nós e propriedades:

Category (title)

Product (title, description, price, availability, shippability)

Customer (name, email, registration date)

Promotional Offer (type, content)

## Relações:

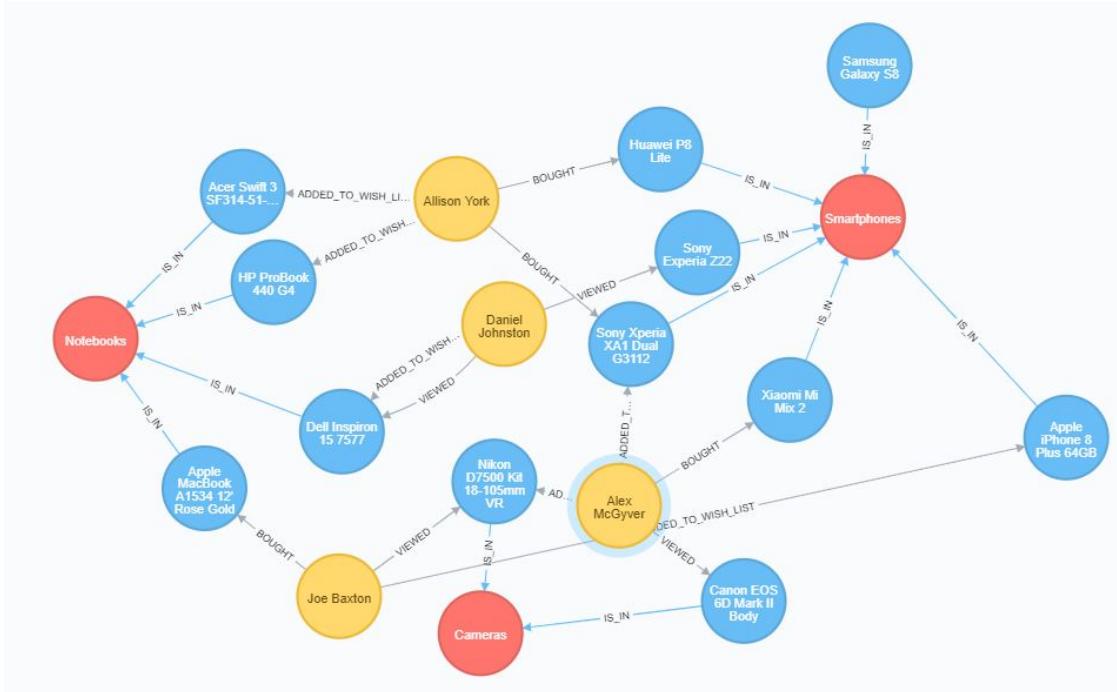
Product is\_in Category

Customer added\_to\_wish\_list Product

Customer bought Product

Customer viewed (clicks\_count) Product

# Laboratório



# Laboratório

**Exemplo #1:** Utilizar o neo4j para determinar as preferências de um determinado cliente. Neste caso o objetivo é procurar produtos da categoria notebook que sejam passíveis de ser incluídos numa proposta promocional.

É necessário aprender quais as preferências dos clientes de forma a criar uma oferta profissional para uma determinada categoria, neste caso **notebooks**.

# Laboratório

**Passo 1:** Listar todos os notebooks que os utilizadores viram ou adicionaram à lista de compras.

```
MATCH (:Customer)-[:ADDED_TO_WISH_LIST|:VIEWED]->(notebook:Product)-[:IS_IN]->(:Category
{title: 'Notebooks' })
RETURN notebook;
```

OU

```
MATCH
(:Customer)-[:ADDED_TO_WISH_LIST|:VIEWED]->(notebook:Product)-[:IS_IN]->(cat:Category)
WHERE cat.title = 'Notebooks'
RETURN notebook;
```

# Laboratório

**Passo 2:** Incluir os notebooks encontrados anteriormente numa proposta promocional

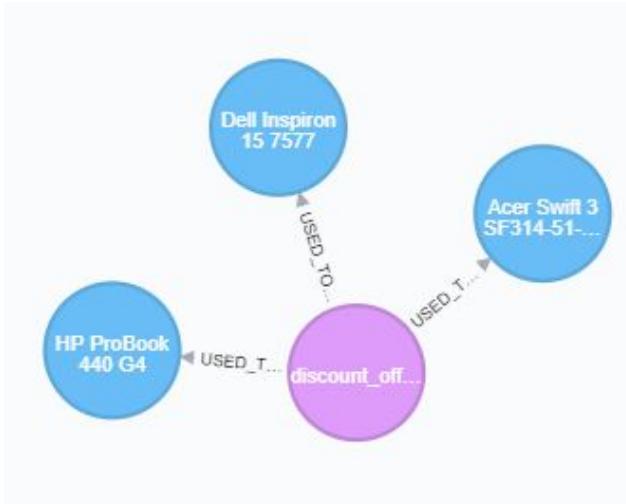
```
CREATE (offer:PromotionalOffer {type: 'discount_offer', content: 'Notebooks discount offer!!!'})  
WITH offer  
MATCH (:Customer)-[:ADDED_TO_WISH_LIST|:VIEWED]->(notebook:Product)-[:IS_IN]->(:Category  
{title: 'Notebooks'})  
MERGE (offer)-[:USED_TO_PROMOTE]->(notebook);
```

MERGE = MATCH + CREATE

# Laboratório

**Passo 3:** Validar a correta criação da oferta promocional

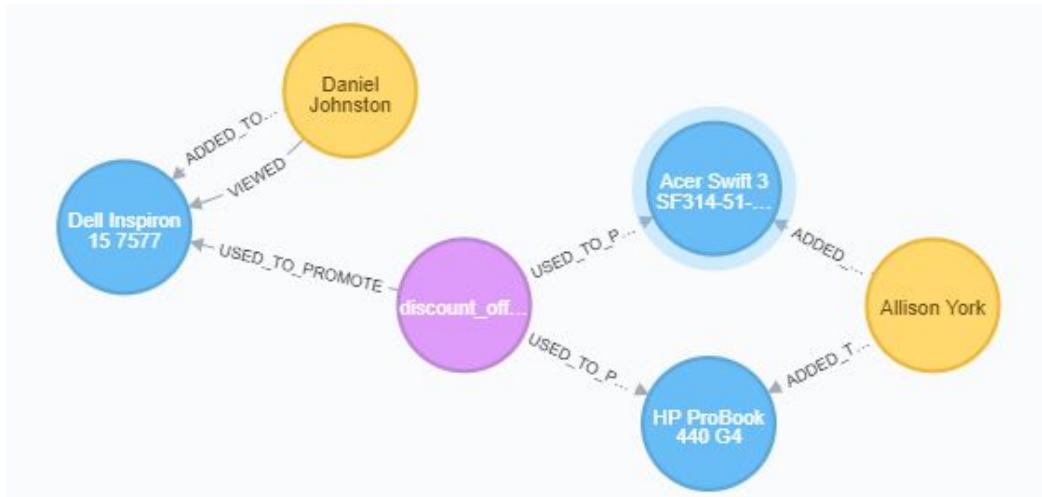
```
MATCH (offer:PromotionalOffer)-[:USED_TO_PROMOTE]->(product:Product)  
RETURN offer, product;
```



# Laboratório

**Passo 4:** Após a criação da oferta promocional vamos selecionar a que clientes vamos enviar a mesma:

```
MATCH (offer:PromotionalOffer {type:  
'discount_offer'})-[:USED_TO_PROMOTE]->(product:Product)<-[:ADDED_TO_WISH_LIST|:VIEWED]-  
(customer:Customer)  
RETURN offer, product, customer;
```



# Laboratório

**Exemplo #2:** É necessário desenvolver uma campanha promocional mais eficiente cuja taxa de conversão seja mais elevada. Deve para isso ser feita uma oferta de produtos alternativos aos clientes. Por exemplo, se um cliente mostrou interesse num produto e não o comprou, pode ser criada uma oferta promocional que ofereça produtos relacionados.

Neste caso a oferta promocional será direcionada ao cliente "Alex McGyver".

# Laboratório

- a) Devem ser procurados todos os produtos que o cliente "Alex McGyver" não:  
ADDED\_TO\_WISH\_LIST,  
VIEWED,  
BOUGHT.
  
- b) Devemos também ter uma query que devolva todos os produtos que o "Alex McGyver":  
ADDED\_TO\_WISH\_LIST,  
VIEWED,  
BOUGHT.
  
- c) Estas queries devem devolver produtos da mesma categoria e de uma gama de preço semelhante. Para tal só vão ser listados produtos que custem +-20% de um determinado item.

# Laboratório

## a) Todos os produtos não relacionados com o "Alex McGyver":

```
MATCH (alex:Customer {name: 'Alex McGyver'})  
MATCH (free_product:Product)  
WHERE NOT ((alex)-->(free_product))  
RETURN free_product;
```

### Resultado:

Sony Experia Z22 preço: \$765.0

Samsung Galaxy S8 preço: \$784.0

Apple iPhone 8 Plus 64GB preço: \$874.2

Huawei P8 Lite preço: \$191.0

Acer Swift 3 SF314-51-34TX preço: \$595.0

HP ProBook 440 G4 preço: \$771.3

Dell Inspiron 15 7577 preço: \$1477.5

Apple MacBook A1534 12' Rose Gold preço: \$1293.0

# Laboratório

## b) Todos os produtos relacionados com o "Alex Mcgyver":

```
MATCH (product:Product)
WHERE ((alex)-->(product))
RETURN product;
```

### Resultado:

Xiaomi Mi Mix 2 (preço: \$420.87): Preço para as recomendações de \$336.70 até \$505.04.

Sony Xperia XA1 Dual G3112 (preço: \$229.50): Preço para as recomendações de \$183.60 até \$275.40.

# Laboratório

a+b) Listar todos os produtos novos que compartilham a mesma categoria que os produtos com os quais o cliente já interagiu:

```
MATCH (alex:Customer {name: 'Alex McGyver'})  
MATCH (free_product:Product)  
WHERE NOT ((alex)-->(free_product))  
MATCH (product:Product)  
WHERE ((alex)-->(product))  
MATCH (free_product)-[:IS_IN]->(category)<-[:IS_IN]-(product)  
RETURN free_product;
```

## Resultado:

Apple iPhone 8 Plus 64GB (preço: \$874.20)

Huawei P8 Lite (preço: \$191.00)

Samsung Galaxy S8 (preço: \$784.00)

Sony Xperia Z22 (preço: \$765.00)

# Laboratório

c) Adicionar o filtro para listar apenas os produtos dentro da gama de preços definida (+-20% do valor):

```
MATCH (alex:Customer {name: 'Alex McGyver'})  
MATCH (free_product:Product)  
WHERE NOT ((alex)-->(free_product))  
MATCH (product:Product)  
WHERE ((alex)-->(product))  
MATCH (free_product)-[:IS_IN]->(category)<-[:IS_IN]-(product)  
WHERE  
((product.price - product.price * 0.20) >= free_product.price <= (product.price + product.price * 0.20))  
RETURN free_product;
```

**Resultado:**

Huawei P8 Lite (price: \$191.00)

# Laboratório

É possível agora criar uma oferta promocional, com:

- tipo: 'personal\_replacement\_offer'
- conteúdo: 'Personal replacement offer for ' + alex.name.

O que vai ser armazenado vai ser o email do cliente como propriedade da relação "USED\_TO\_PROMOTE" entre o cliente e o produto:

```
MATCH (alex:Customer {name: 'Alex McGyver'})  
MATCH (free_product:Product)  
WHERE NOT ((alex)-->(free_product))  
MATCH (product:Product)  
WHERE ((alex)-->(product))  
MATCH (free_product)-[:IS_IN]->()-<[:IS_IN]-(product)  
WHERE ((product.price-product.price*0.20)>=free_product.price<= (product.price+product.price*0.20))  
CREATE(offer:PromotionalOffer{type:'personal_replacement_offer', content: 'Personal replacement offer for ' + alex.name})  
WITH offer, free_product, alex  
MERGE(offer)-[rel:USED_TO_PROMOTE{email:alex.email}]->(free_product)  
RETURN offer, free_product, rel;
```

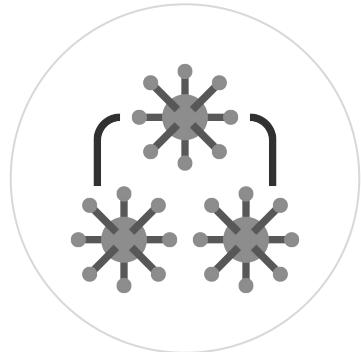
# Laboratório



**USED\_TO\_PROMOTE**

**<id>:** 286 **email:** mcgalex@example.com

# FE06 - Introdução às Bases de Dados de Grafos



# noSQL

Mestrado Integrado em Engenharia Informática

<https://hpeixoto.me/class/nosql>

Hugo Peixoto  
[hpeixoto@di.uminho.pt](mailto:hpeixoto@di.uminho.pt)

2020/2021