NoSQL Databases

PL06 – Introduction to Graph Databases

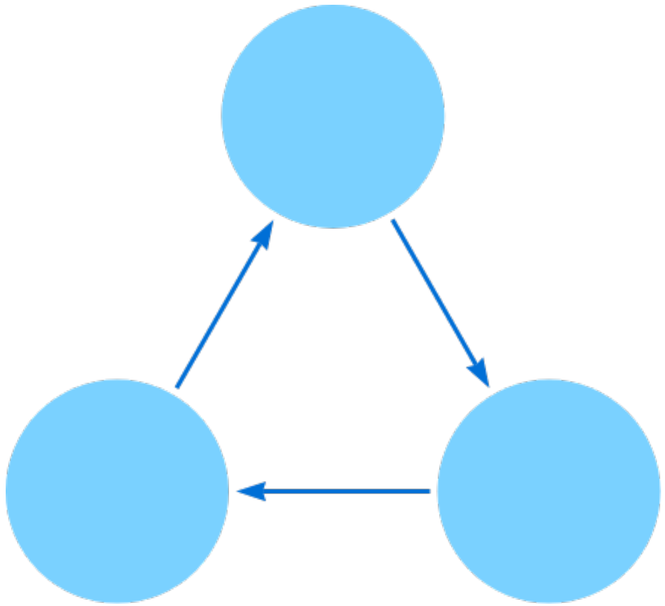
Teacher: Cristiana Neto
Email: cristiana.neto@algoritmi.uminho.pt

Office hours:
Friday 10h-11h

Summary

Introduction to graph databases

- **Data Model:**
  - Nodes and Relationships



- **Examples:**

Neo4j, OrientDB, InfiniteGraph, AllegroGraph

Introduction to graph databases

Graph Model Properties

- Contains **nodes** and **relationships**
- Nodes can contain **properties** (key:value pair)
- Nodes can be identified with one or more **labels**
- Relationships can be named (label) and have **direction**. They always have a start node and an end node
- Relationships can contain properties (key:value pair)

Introduction to graph databases

Nodes

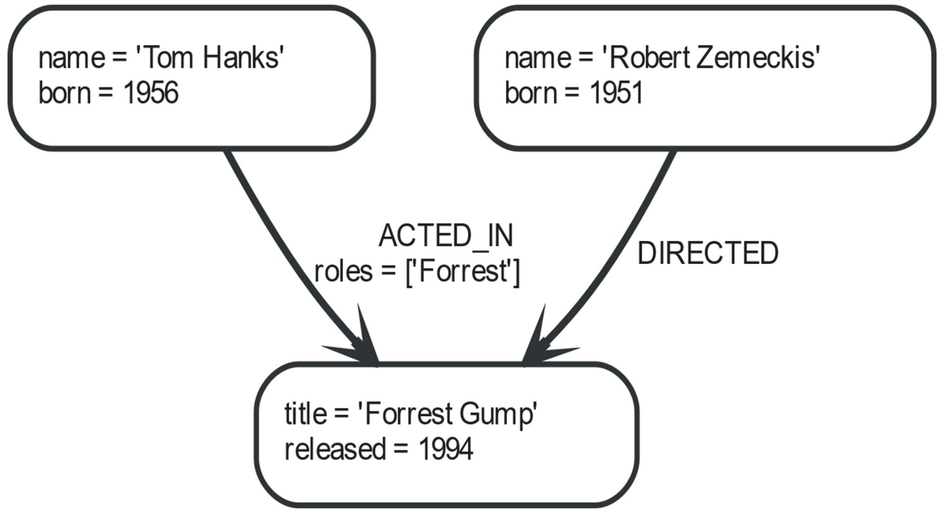
- They are equivalent to vertices in graph theory;

- They are the reference data structure and are interconnected by relationships. A node can contain more than one description and has properties.

Introduction to graph databases

Relationships

- Equivalent to edges in graph theory;
- A relationship connects two nodes, which in turn can have multiple relationships. A relationship can have one or more properties.



Introduction to graph databases

Properties

- They are applied both to nodes and to relationships.
- They are of type *key:value* which means that properties can take on different types.

Types

Number (integer, float)
String
Boolean
Spatial type: Point
Temporal types:
- Date
- Time
- LocalTime
- DateTime
- LocalDateTime
- Duration

Introduction to graph databases

Advantages

Performance: Graph databases excel at handling complex queries with interconnected data, offering fast query response times even as data scales.

Flexibility: With flexible or schema-less structures, graph databases allow for dynamic addition or modification of relationships and properties without disrupting existing data.

Agility: Graph databases enable rapid development by providing a natural representation of relationships, allowing for quick modeling and implementation of complex connections without extensive normalization processes.

Introduction to graph databases

Disadvantages

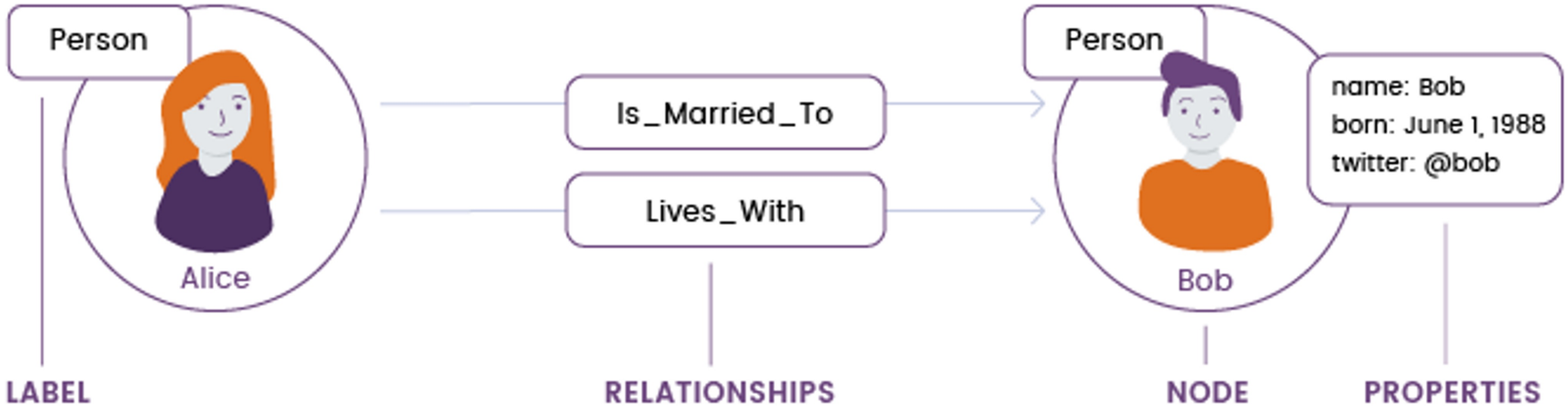
Sharding Challenges: Sharding, or distributing data across multiple machines, can be complex in graph databases due to the intricate relationships between nodes.

Scaling Up: While graph databases scale reasonably well, they may face limitations in scaling up to extremely large datasets or high-throughput workloads compared to other databases.

Learning Curve: Adopting a graph database often requires a shift in mindset and understanding of how to model and query data in a graph-oriented manner.

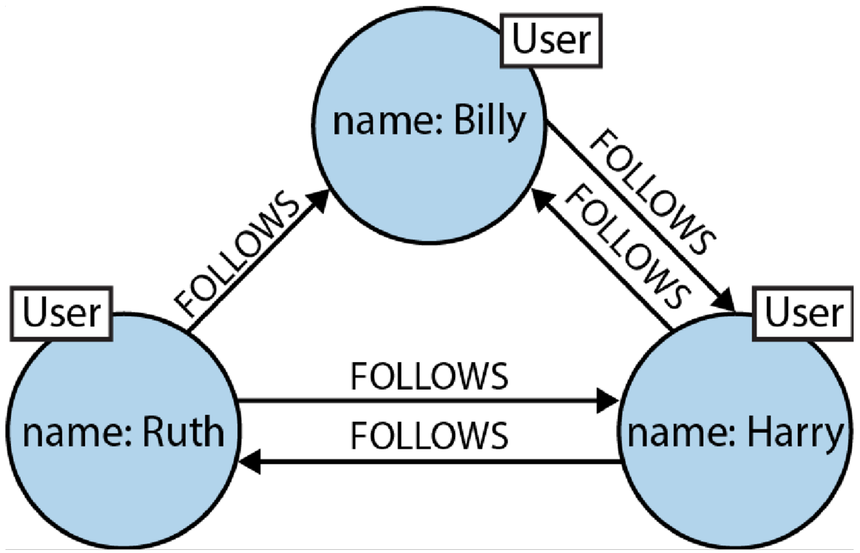
Introduction to graph databases

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.

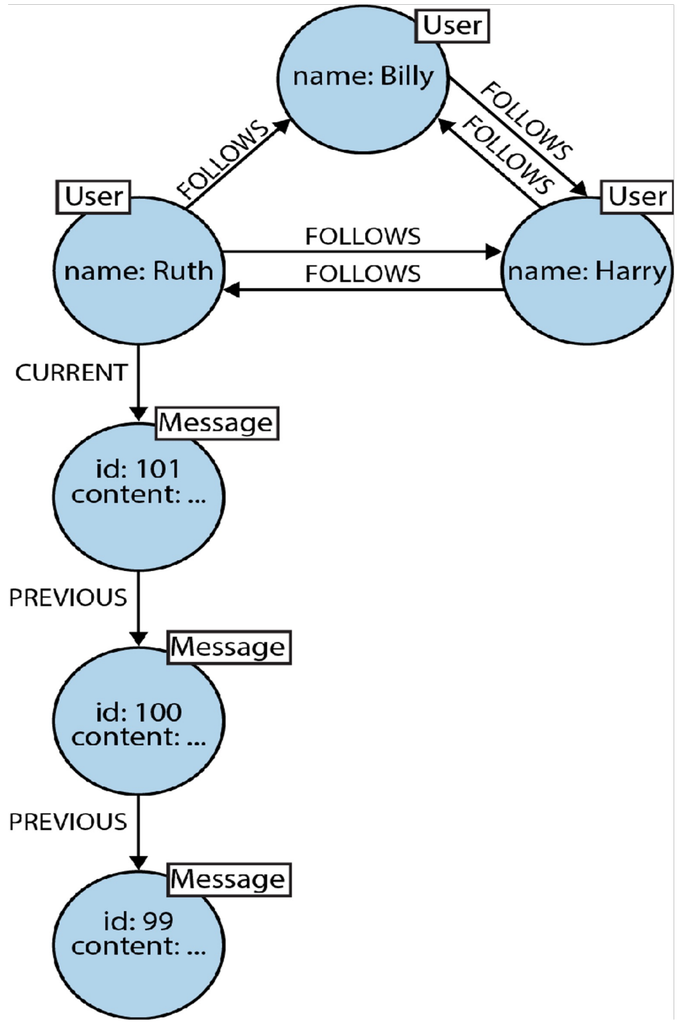


Introduction to graph databases

Example: X (Twitter)



User Network and Relationships



Post Timeline

Introduction to graph databases

Relational databases

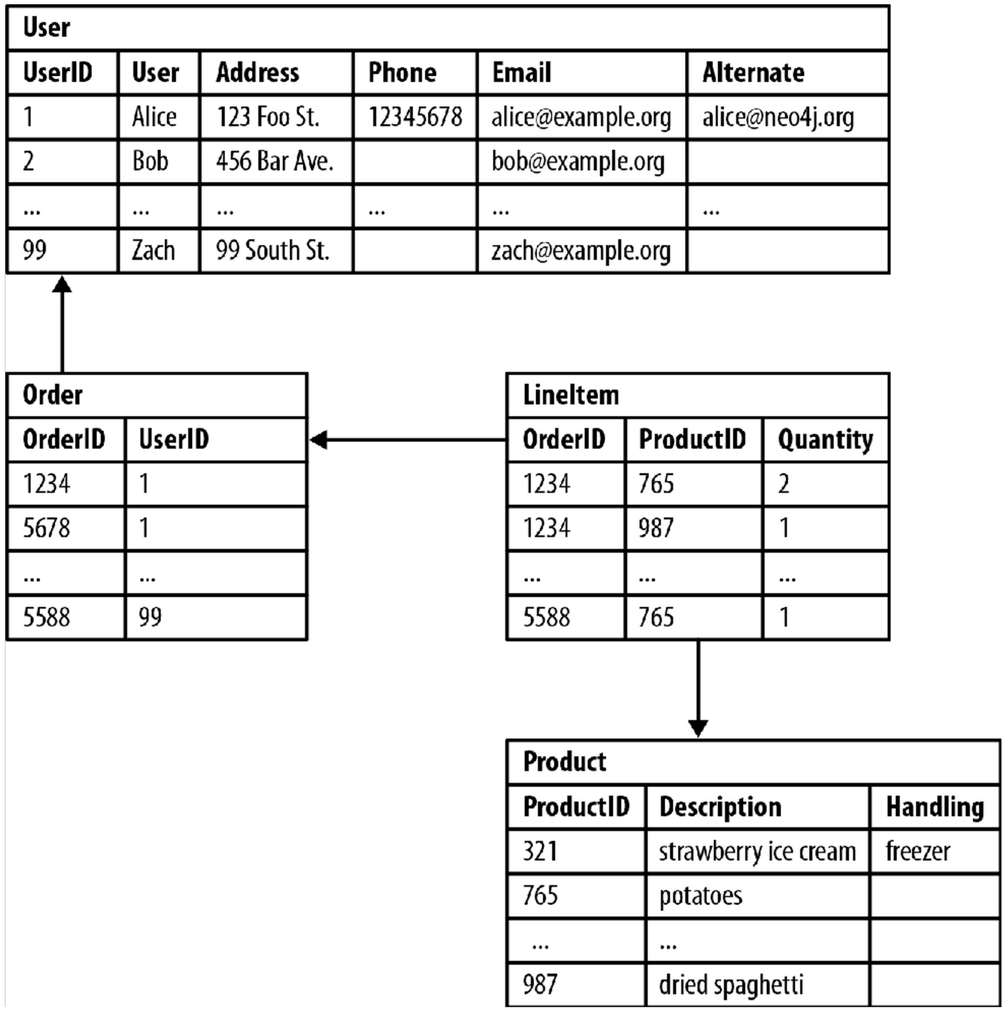
It takes several joins to be able to figure it out what products a customer bought. Even more complex if the query is reversed.

For instance:

What products did a customer buy?

vs

Which customers bought this product?



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Introduction to graph databases

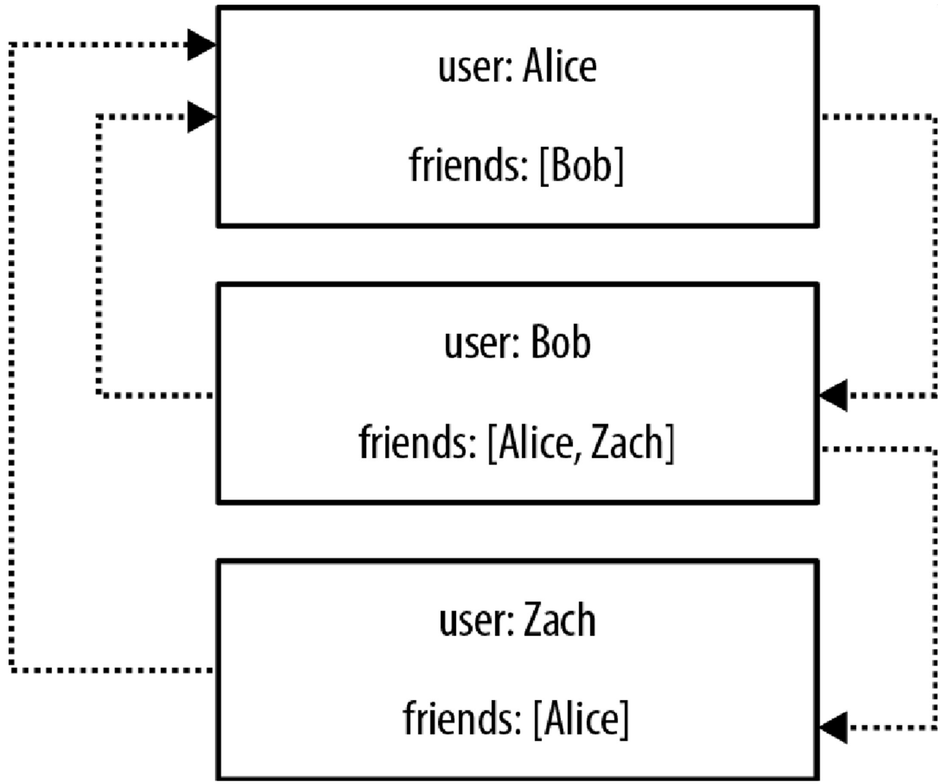
Document Databases

"Who are BOB's friends?"

It doesn't always look like the following:

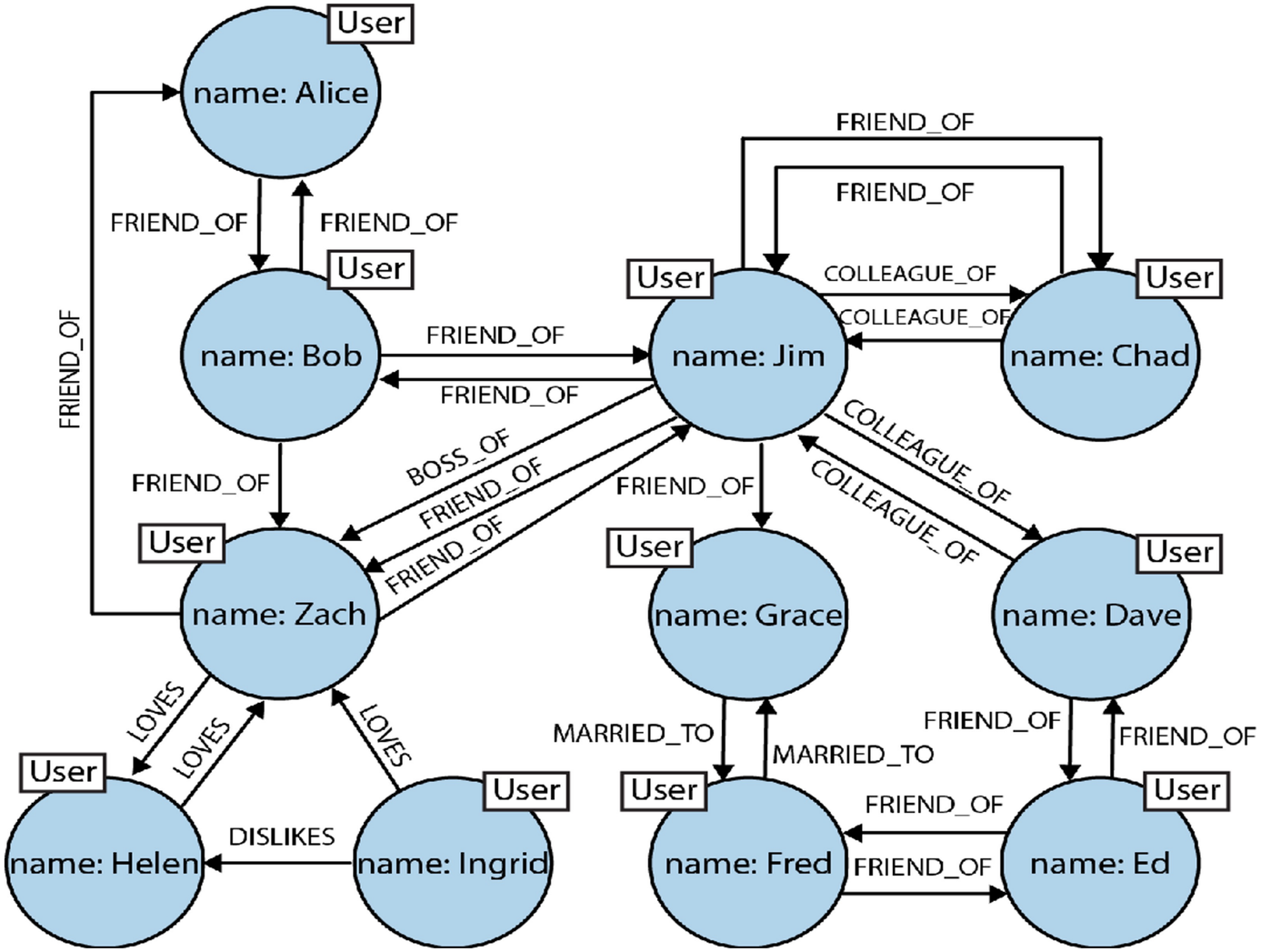
"What is Bob friends with?"

Just a scan of the entire collection will be able to browse through all the documents looking for the "Friends" attribute that contains Bob, to answer the second question.



Introduction to graph databases

Graph databases



Introduction to graph databases

Graph databases

	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

Introduction to graph databases

Use Cases

Fraud

Recommender systems

Social Media

Monitoring of networks and database structures

Identity and access management

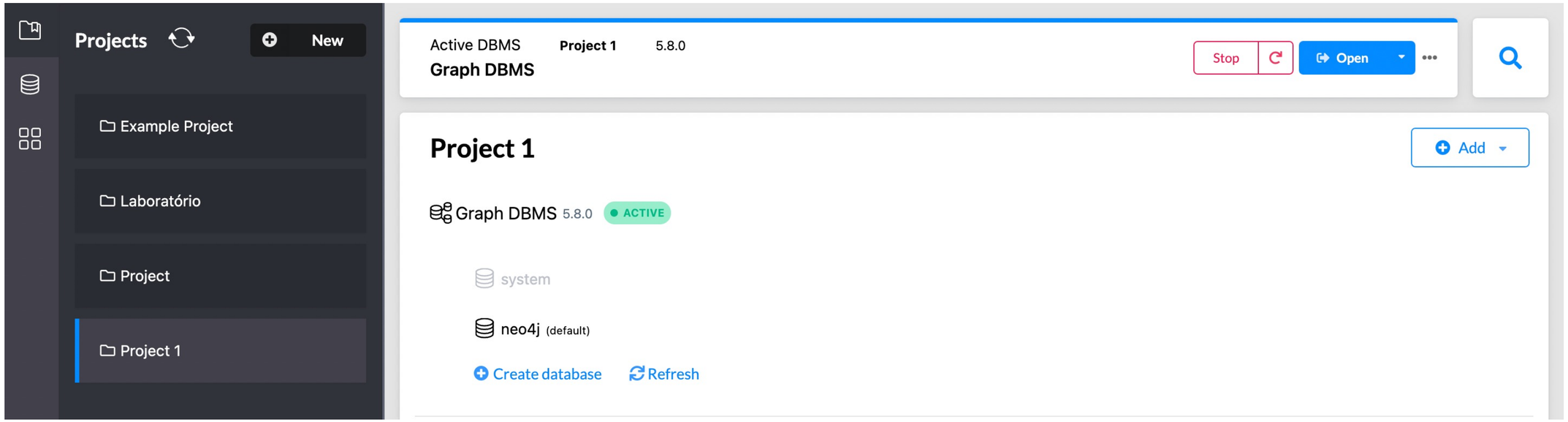
Neo4J Container Installation

```
> docker pull neo4j

> docker run --name=neo4j \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    neo4j
```

Abrir no browser: http://localhost:7474/

Neo4J Desktop



https://neo4j.com/download/

Cypher

Cypher is Neo4j's declarative graph query language. It was created in 2011 by Neo4j engineers as an SQL-equivalent language for graph databases. Similar to SQL, Cypher lets users focus on *what* to retrieve from graph, rather than *how* to retrieve it.

Cypher provides a visual way of matching patterns and relationships. It relies on the following ascii-art type of syntax:

```
(nodes)-[:CONNECT_TO]→(otherNodes)
```

Cypher

AsciiArt for the nodes

Nodes are represented by parentheses:

`()` or `(p)`

Labels, or tags, start with : and group nodes by roles or types:

```
(p:Person:Mammal)
```

Nodes have properties:

```
(p:Person {name : ‘Veronica’})
```

Cypher

AsciiArt for the nodes

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

() represents an uncharacterized, anonymous node. To be used elsewhere in the query, a variable must be added, for instance: (matrix). The variable is only usable in one run.

By using the "Movie" label in conjunction with the : the type of the node is declared. Thus, it is possible to filter the nodes that will be selected by the query.

Cypher

AsciiArt for the relations

Relationships are represented by hyphens or square brackets:

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

The direction of the relationships is represented by the < or > sign:

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

Relationships also have properties:

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

Cypher

AsciiArt for the relations

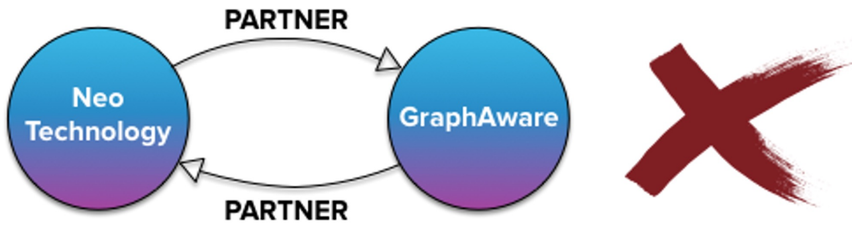
```
-->
-[role]->
-[:ACTED_IN]->
-[role:ACTED_IN]->
-[role:ACTED_IN {roles: ["Neo"]}]->
```

Cypher uses a pair of hyphens to represent an undirected relationship. Directed relationships are represented by the sign of > or <. And the square brackets are used to define the details of the relationship.

Cypher

➡ AsciiArt for the relations

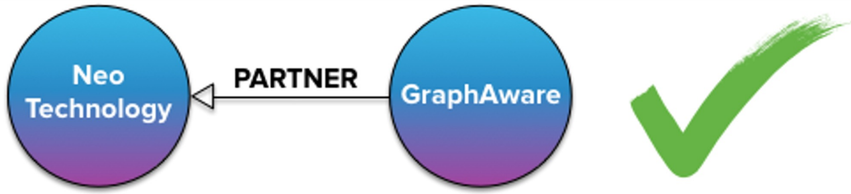
```
MATCH (neo)-[:PARTNER]-(partner)
```



The result is equal to performing the following two operations:

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

The most efficient way is to represent the partner relationship randomly choosing the direction.



Cypher

AsciiArt

Case sensitive:

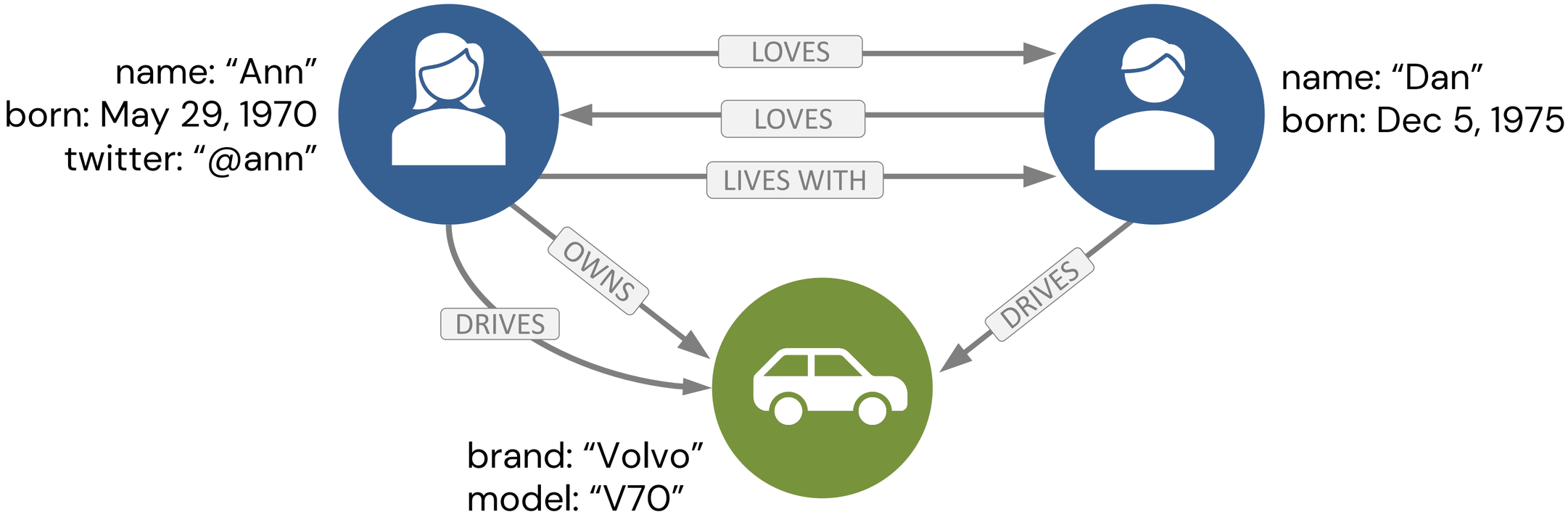
- Node labels
- Relationship types
- Property keys

Case insensitive:

- Cypher keywors (return, match, where, etc)

Cypher

Example



Cypher

➡ Example

Who drives a car owned by a lover?


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

MATCH and RETURN are Cypher keywords
p1, p2 , c are variables
:Person, :Car are node labels
:DRIVES, :OWNED_BY, :LOVES are relationships

Cypher

Keywords

MATCH: Used to specify patterns to match in the graph.

CREATE: Used to create nodes, relationships, or paths in the graph.

RETURN: Specifies what data to retrieve from the graph.

WHERE: Allows filtering of results based on specified conditions.

MERGE: Combines the functionality of MATCH and CREATE, ensuring that a pattern exists and creating it if not.

DELETE: Removes nodes, relationships, or paths from the graph.

SET: Updates properties of nodes or relationships.

WITH: Chains queries together, passing results from one part of the query to the next.

UNWIND: Expands a list or collection into a sequence of rows.

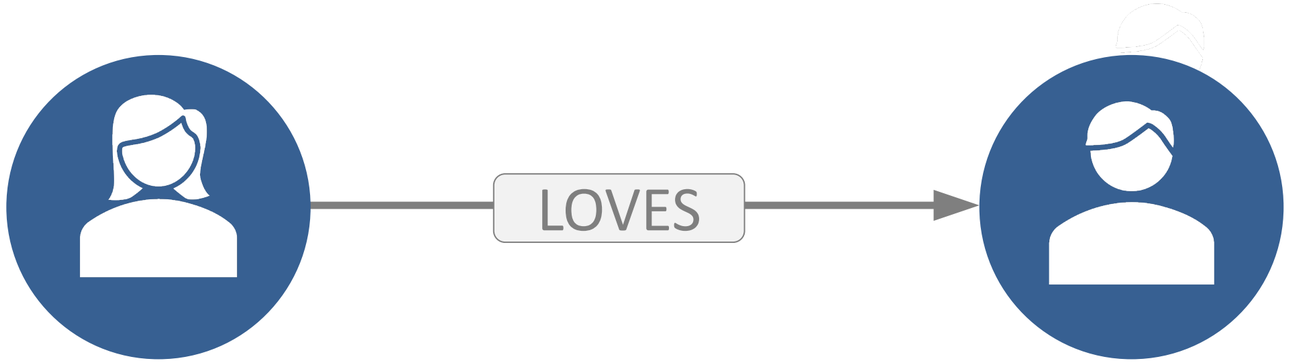
FOREACH: Executes a subquery for each element in a list or collection.

ORDER BY: Sorts the results based on specified criteria.

LIMIT: Limits the number of results returned by the query.

Cypher

Create data




```
CREATE (:Person {name: “Ann” }) - [:LOVES] -> (:Person {name: “Dan” })
```

NODE — (:Person {name: “Ann” })

LABEL — :Person

PROPERTY — {name: “Ann” }

NODE — (:Person {name: “Dan” })

LABEL — :Person

PROPERTY — {name: “Dan” }

Cypher

Create data – Singularity

{name: “Ann” }) {name: “Ann” })

```
CREATE (:Person {name: “Ann” })
```

```
CREATE CONSTRAINT ON (p:Person)
ASSERT p.name is UNIQUE
```

ERROR **Neo.ClientError.Schema.ConstraintValidationFailed**

```
Neo.ClientError.Schema.ConstraintValidationFailed:
Node(0) already exists with label `Person` and
property `name` = 'Ann'
```

Cypher

➡ Create data and search nodes

```
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
```

Cypher

➡ Operators

`=, <>, <, >, <=, >=, 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
```

Cypher

➡ Run 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
```

➡ Filter results:

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

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

Cypher

Add nodes:

Criar o filme Mystic River:

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

Cypher

Add properties:

Add a tagline to the movie 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
```

Update properties:

Update the movie's release date (the syntax is the same!!):

```
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
```

Cypher

Create relationships:

Find the movie "Mystic River" and actor Kevin Bacon and create a relationship between them.

```
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
```

Ou

```
MATCH (p:Person {name: "Kevin Bacon"})
MATCH (m:Movie {title: "Mystic River"})
MERGE (p)-[r:ACTED_IN]->(m)
ON CREATE SET r.roles = ["Sean"]
RETURN p, r, m
```

MERGE = MATCH + CREATE

Cypher

Criar uma pessoa na BD:

```
CREATE (me:Person {name:"Cristiana Neto"}) RETURN me.name
```

Classificar o filme "Mystic River":

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

Cypher

Delete nodes

To delete the node and the relationships that may exist, it is necessary to run:

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

The "OPTIONAL MATCH" clause is used to search for the described pattern. Whereas nulls are used for unknown parts of the pattern. We may not know what we were related to. The DETACH DELETE command is designed to perform exactly this type of operation:

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

Delete all nodes and relationships:

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

Cypher

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

Indexes

An index is a copy of some data in order to make searches faster. However, this entails the need for more storage and longer writing time (duplication of data). It's not always easy to decide what should be indexed.

Example of creating an index:

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

Exemplo para eliminar um índice:

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

Cypher

Indexes

An index is a copy of some data in order to make searches faster. However, this entails the need for more storage and longer writing time (duplication of data). It's not always easy to decide what should be indexed.

Example of creating an index:

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

Exemplo para eliminar um índice:

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

Laboratory

Platform-based e-commerce advertising

Objective:
Offer the most relevant products or services to customers

Data Import:
Copy the contents of the email_neo4j.txt file to the browser and run

Verification:

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

Laboratory

Platform-based e-commerce advertising

The database:

Node labels and properties:

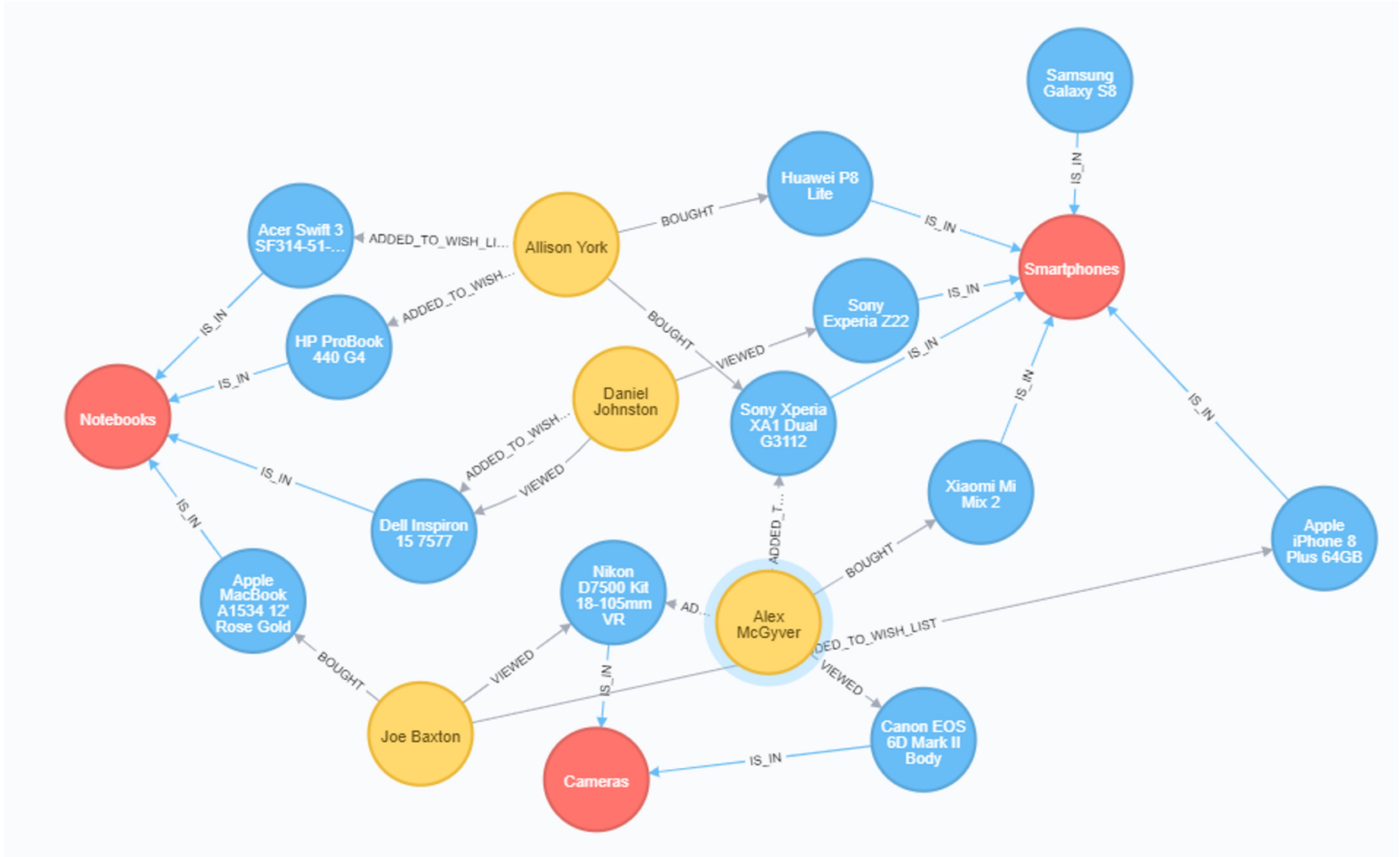
- Category (title)
- Product (title, description, price, availability, shippability)
- Customer (name, email, registration date)
- Promotional Offer (type, content)

Relations:

- Product is_in Category
- Customer added_to_wish_list Product
- Customer bought Product
- Customer viewed (clicks_count) Product

Laboratory

Platform-based e-commerce advertising



Laboratory

Platform-based e-commerce advertising

Example #1: Use neo4j to determine a particular customer's preferences. In this case, the goal is to look for products in the notebook category that can be included in a promotional proposal.

It is necessary to learn what the customers' preferences are in order to create a professional offer for a certain category, in this case notebooks.

Laboratory

Platform-based e-commerce advertising: Example #1

Step 1: List all the notebooks that users have viewed or added to their shopping list.

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

OR

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

Laboratory

Platform-based e-commerce advertising: Example #1

Step 2: Include the notebooks found earlier in a promotional proposal

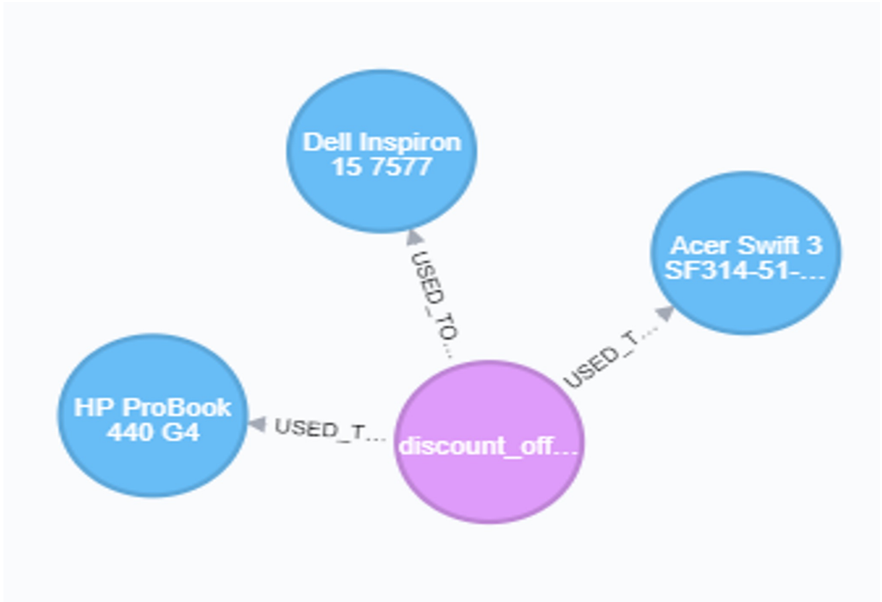
```
CREATE(offer:PromotionalOffer {type: 'discount_offer', content: 'Notebooks discount offer!!'})
WITH offer
MATCH (:Customer)-[:ADDED_TO_WISH_LIST|:VIEWED]->(notebook:Product)-[:IS_IN]->(:Category {title: 'Notebooks'})
MERGE(offer)-[:USED_TO_PROMOTE]->(notebook);
```

Laboratory

Platform-based e-commerce advertising: Example #1

Step 3: Validate the correct creation of the promotional offer

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

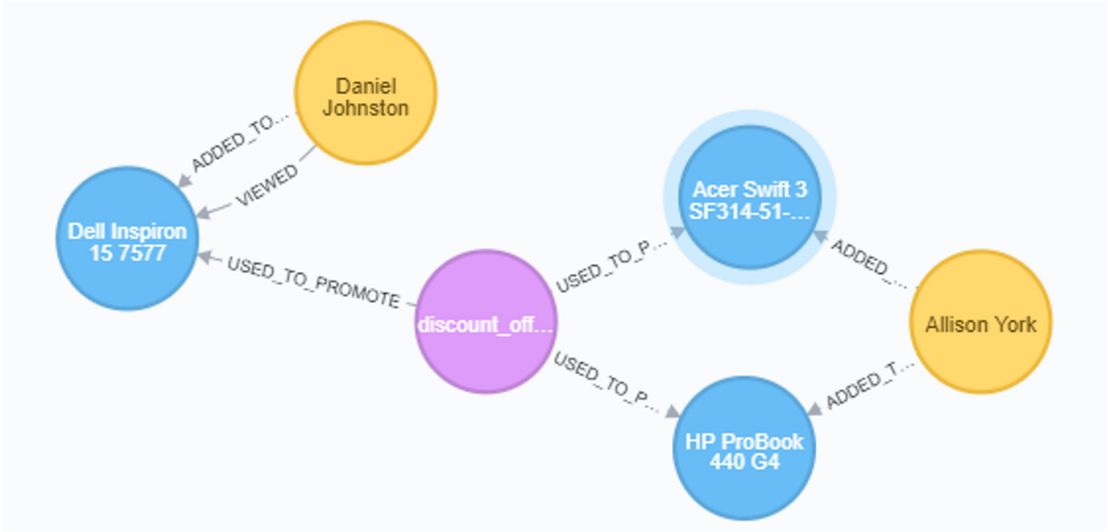


Laboratory

Platform-based e-commerce advertising: Example #1

Step 4: After creating the promotional offer, we will select which customers we will send it to:

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



Laboratory

Platform-based e-commerce advertising:

Example #2: It is necessary to develop a more efficient promotional campaign whose conversion rate is higher. To do this, an offer of alternative products must be made to customers. For example, if a customer showed interest in a product and didn't buy it, a promotional offer can be created that offers related products.

In this case, the promotional offer will be directed to the customer "Alex McGyver".

Laboratory

Platform-based e-commerce advertising: Example #2

a) All products that the customer "Alex McGyver" has not seen or purchased should be searched for:

ADDED_TO_WISH_LIST,
VIEWED,
BOUGHT.

b) We should also have a query that returns all the products that "Alex McGyver":

ADDED_TO_WISH_LIST,
VIEWED,
BOUGHT.

c) These queries should return products of the same category and a similar price range. To do this, only products that cost +-20% of a given item will be listed.

Laboratory

Platform-based e-commerce advertising: Example #2

a) All products not related to "Alex McGyver":

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

Result:

Sony Experia Z22 preço: $765.0, categoria: "Smartphones"

Samsung Galaxy S8 preço: $784.0, categoria: "Smartphones"

Apple iPhone 8 Plus 64GB preço: $874.2, categoria: "Smartphones"

Huawei P8 Lite preço: $191.0, categoria: "Smartphones"

~~Acer Swift 3 SF314-51-34TX preço: $595.0, categoria: "Notebooks"~~

~~HP ProBook 440 G4 preço: $771.3, categoria: "Notebooks"~~

~~Dell Inspiron 15 7577 preço: $1477.5, categoria: "Notebooks"~~

~~Apple MacBook A1534 12' Rose Gold preço: $1293.0, categoria: "Notebooks"~~

Laboratory

Platform-based e-commerce advertising: Example #2

b) All products related to "Alex McGyver":

```
MATCH (alex:Customer {name: 'Alex McGyver'})
MATCH (product:Product)
WHERE ((alex)-->(product))
RETURN product;
```

Result:

Xiaomi Mi Mix 2 (preço: $420.87, categoria:), categoria: "Smartphones": Preço para as recomendações de $336.70 até $505.04.

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

~~Canon EOS 6D Mark II Body (preço: $1794.0), categoria: "Cameras"~~

~~Nikon D7500 Kit 18-105mm VR (preço: $1612.35), categoria: "Cameras"~~

Laboratory

Platform-based e-commerce advertising: Example #2

a+b) List all new products that share the same category as the products the customer has already interacted with:

```
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;
```

Result:

Apple iPhone 8 Plus 64GB (price: $874.20)

Huawei P8 Lite (price: $191.00)

Samsung Galaxy S8 (price: $784.00)

Sony Xperia Z22 (price: $765.00)

Laboratory

Platform-based e-commerce advertising: Example #2

c) Add the filter to list only the products within the defined price range (+-20% of the value):

```
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;
```

Result:

Huawei P8 Lite (price: $191.00)

Laboratory

Platform-based e-commerce advertising: Example #2

It is now possible to create a promotional offer, with:

- Type: 'personal_replacement_offer'
- content: 'Personal replacement offer for' + alex.name.

What will be stored will be the customer's email as a property of the "USED_TO_PROMOTE" relationship between the customer and the product:

```
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;
```

FE05 – Graph Databases with Neo4j

University of Minho
Department of Computer Science

FE05

Course: MSc in Informatics/ MSc in Bioinformatics
U.C.: NoSQL Databases

Worksheet 05 - PL06	
Teacher:	António Abelha / Cristiana Neto
Theme:	Introduction to Graph Databases
Class:	Laboratory Practice
Academic Year:	2023-2024 – 2nd Semester
Duration:	2 hours

NoSQL Databases

PL06 – Introduction to Graph Databases

Teacher: Cristiana Neto
Email: cristiana.neto@algoritmi.uminho.pt

Office hours:
Friday 10h-11h