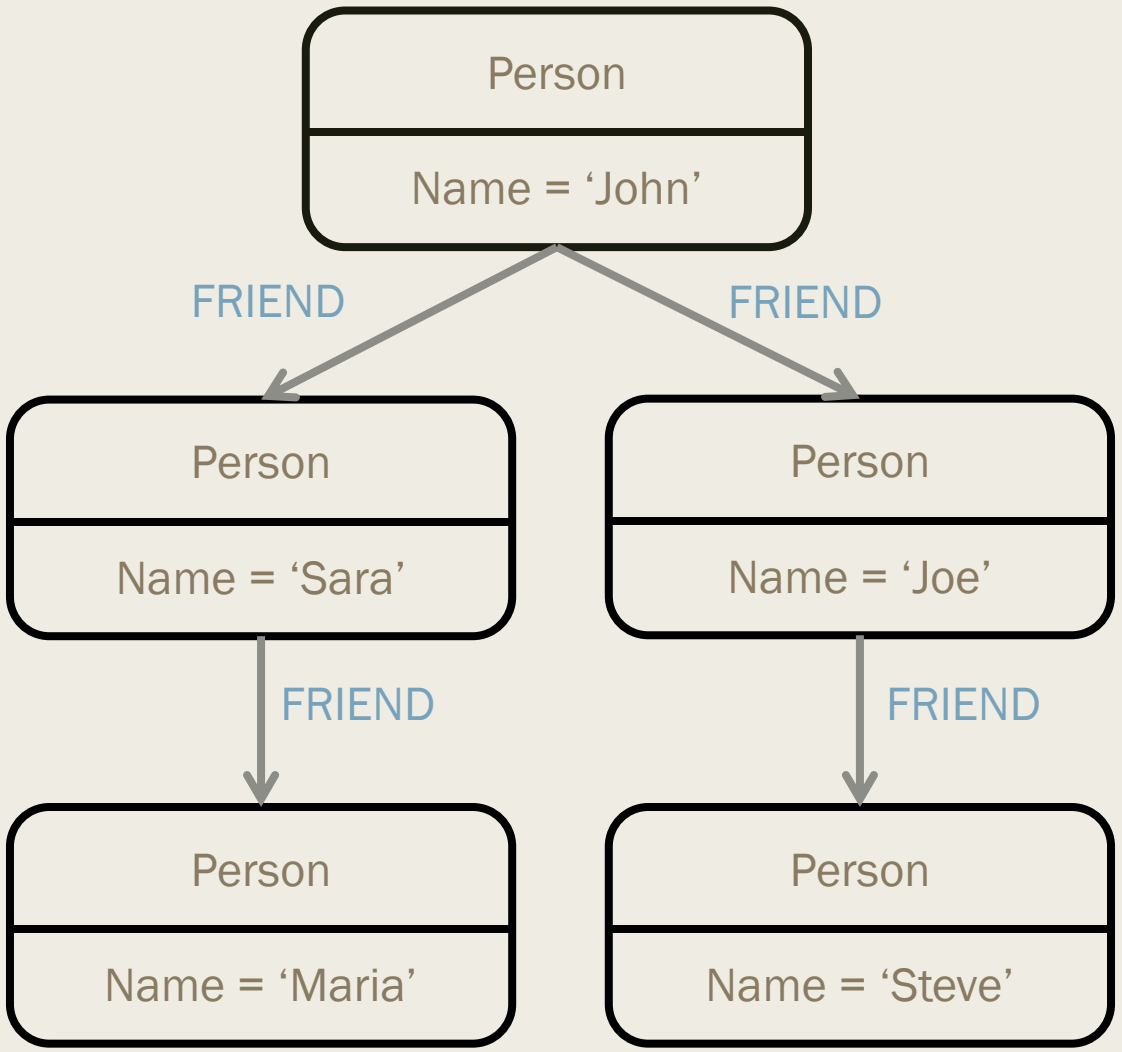
INFORMATION RETRIEVAL

HANDLING GRAPHS WITH NEO4J

lorenzo.bellomo@di.unipi.it - Lorenzo Bellomo

Paolo Ferragina



What is a Graph?

A graph **G = (V, E)** is defined with two elements:

- **V**: set of **nodes.** They are the **entities** we model
- **E**: **relations** between nodes: They model the kind of **interaction** they share

Both the elements of V and E can have attached **properties**

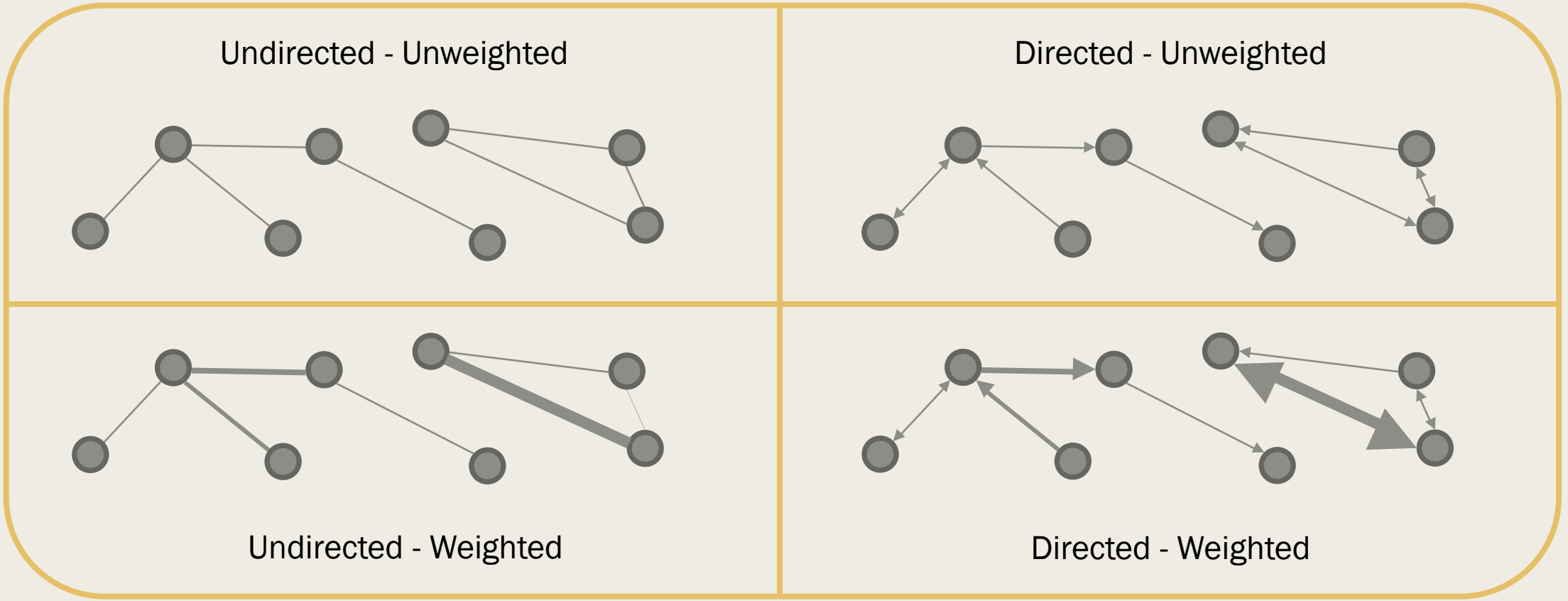
“By 2025, graph technologies will be used in 80% of data and analytics innovations, up from 10% in 2021, facilitating rapid decision making across the enterprise.”

Merv Adrian e Afraz Jaffri (Gartner)

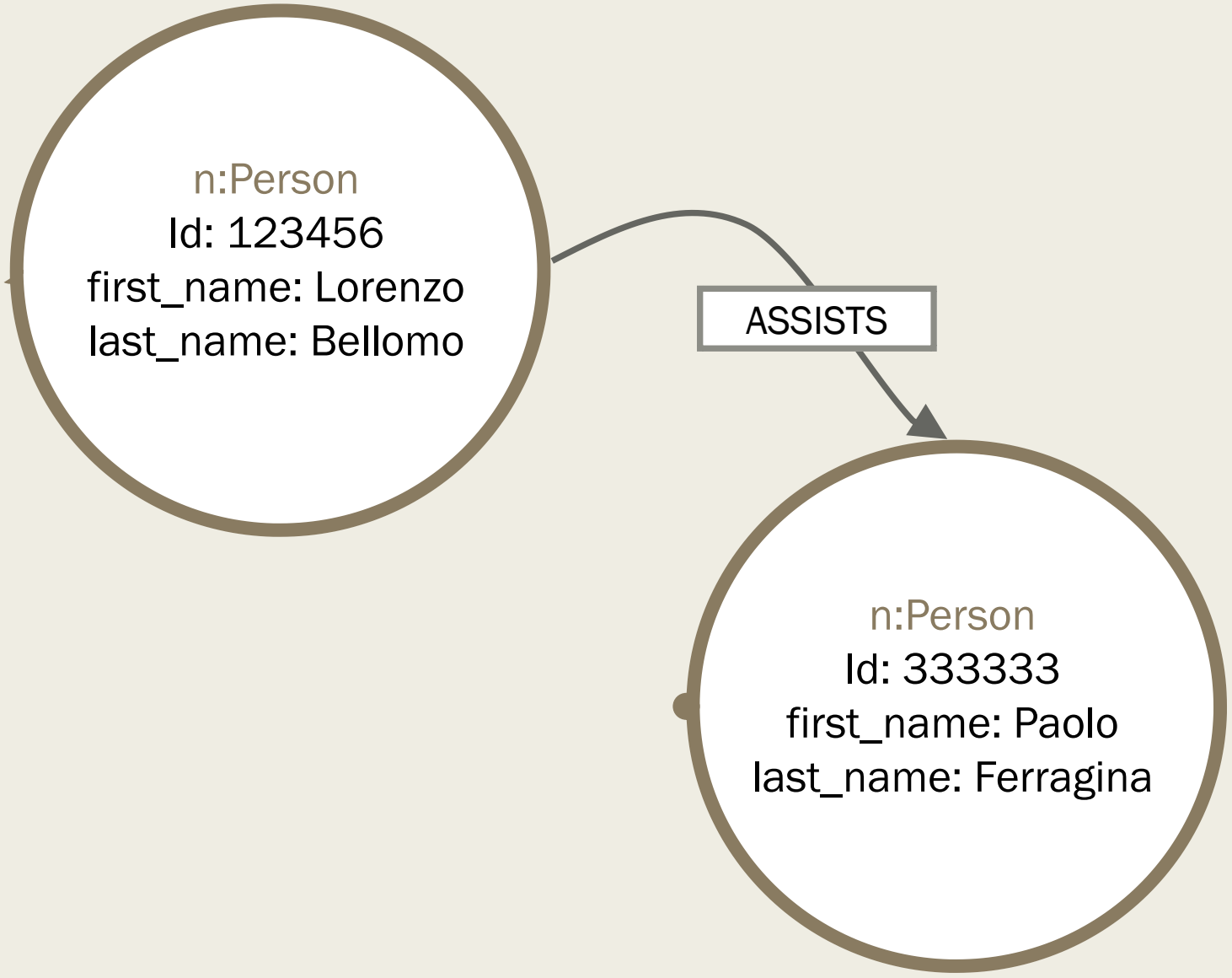
"Market Guide: Graph Database Management Solutions"

(30 Aug 2022)

Why Graphs?



Most common graph types



What is a Graph DB

Relationships are stored natively alongside the data elements (the nodes) in a flexible format.

Everything about the system is optimized for traversing through data quickly

When to use Graphs and GraphDBs

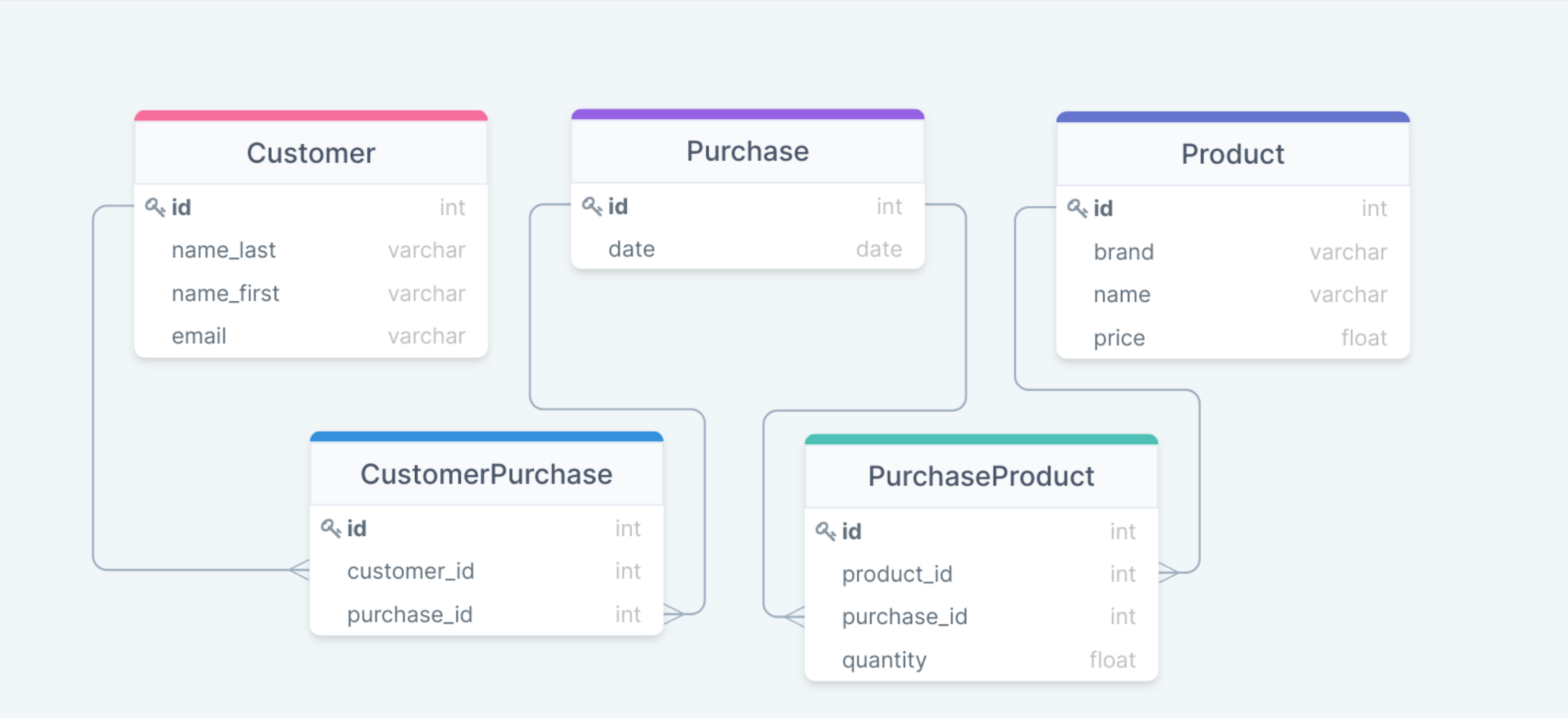
GRAPH STRENGHTS

- Graphs are great when the key interest are relationships
- Graph DBs are stronger than classic DBs when the focus is on links between entities of the same type
- Graph DBs are built for mining info at unknown depth
- Graphs are great for modelling routing problems

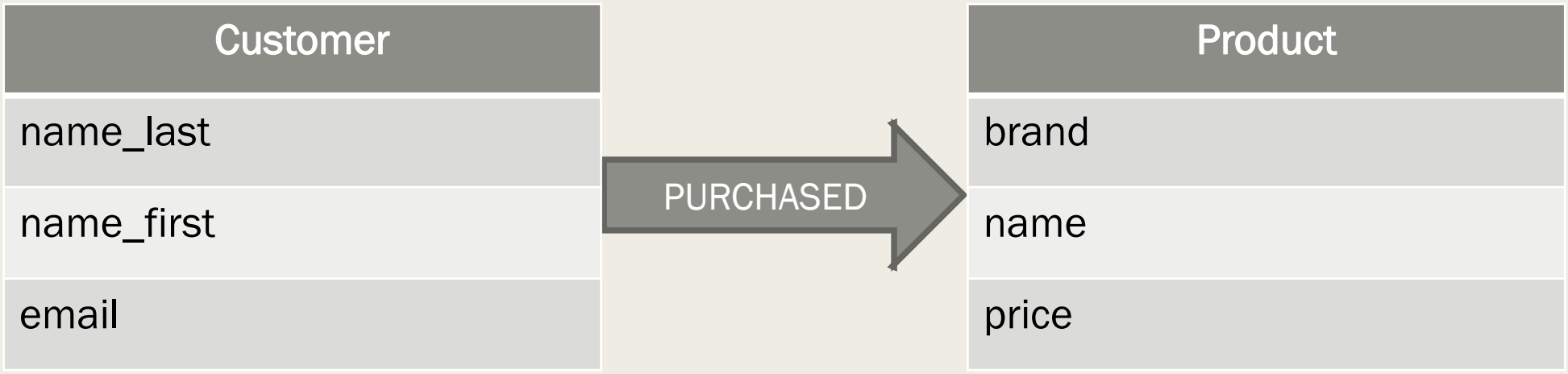
COMPARISON WITH SQL

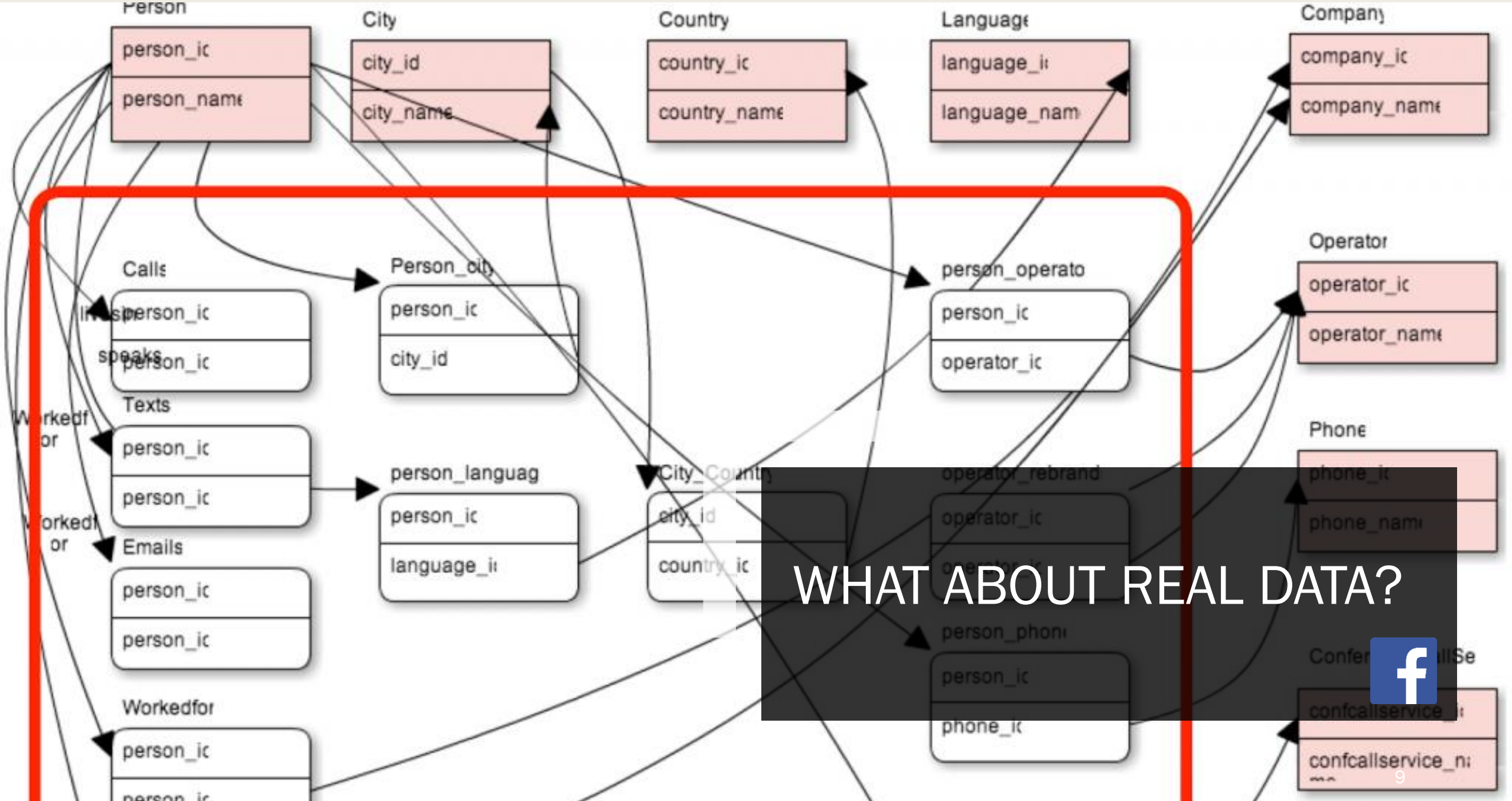
- Relationships in SQL-like DBs are obtained by JOINs
- SQL is great at modelling extremely constrained data
- Multi-level JOINs for complex relations

The SQL Model



The Graph Model




Person
person_ic
person_name
City
city_id
city_name
Country
country_ic
country_name
Language
language_i
language_nam
Company
company_ic
company_name
Calls
person_ic
person_ic
Person_city
person_ic
city_id
persøn_operato
person_ic
operator_ic
Operator
operator_ic
operator_name
Texts
person_ic
person_ic
person_languag
person_ic
language_i
City_Country
city_id
country_ic
operator_rebrand
operator_ic
Phone
phone_ic
phone_nam
Emails
person_ic
person_ic
person_phon
person_ic
phone_ic
Workedfor
person_ic
confcallservice_ni
WHAT ABOUT REAL DATA?

Complexity of relationship-heavy query

SQL

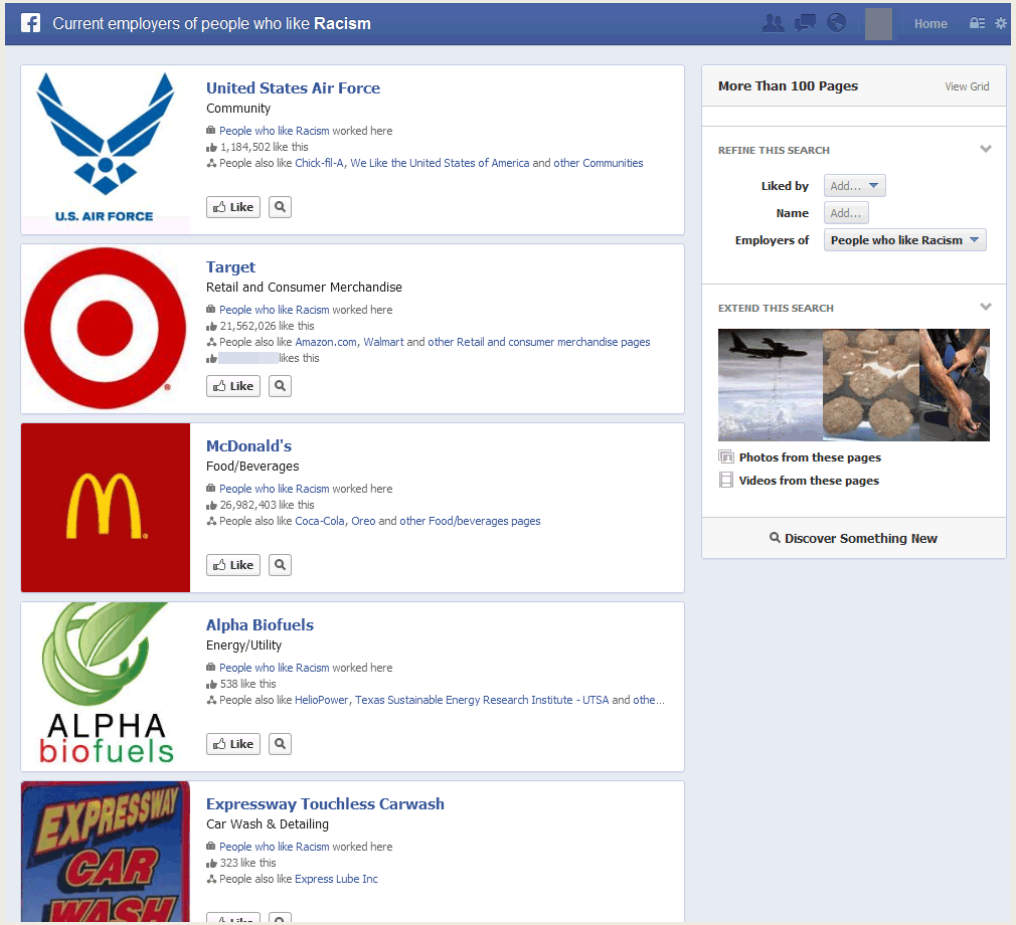
```
SELECT person.name
FROM person mario
  JOIN person AS customer ON
customer.person_id = mario.id
  JOIN purchases ON
purchases.person_id=
customer.person_id
JOIN items ON items.id =
purchases.item_id
WHERE mario.name = 'Mario Rossi'
GROUP BY customer.name
```

Cypher

```
MATCH (x:Person {name: 'Mario Rossi'})-
[:PURCHASED]->(o:Item)
RETURN o.name
```

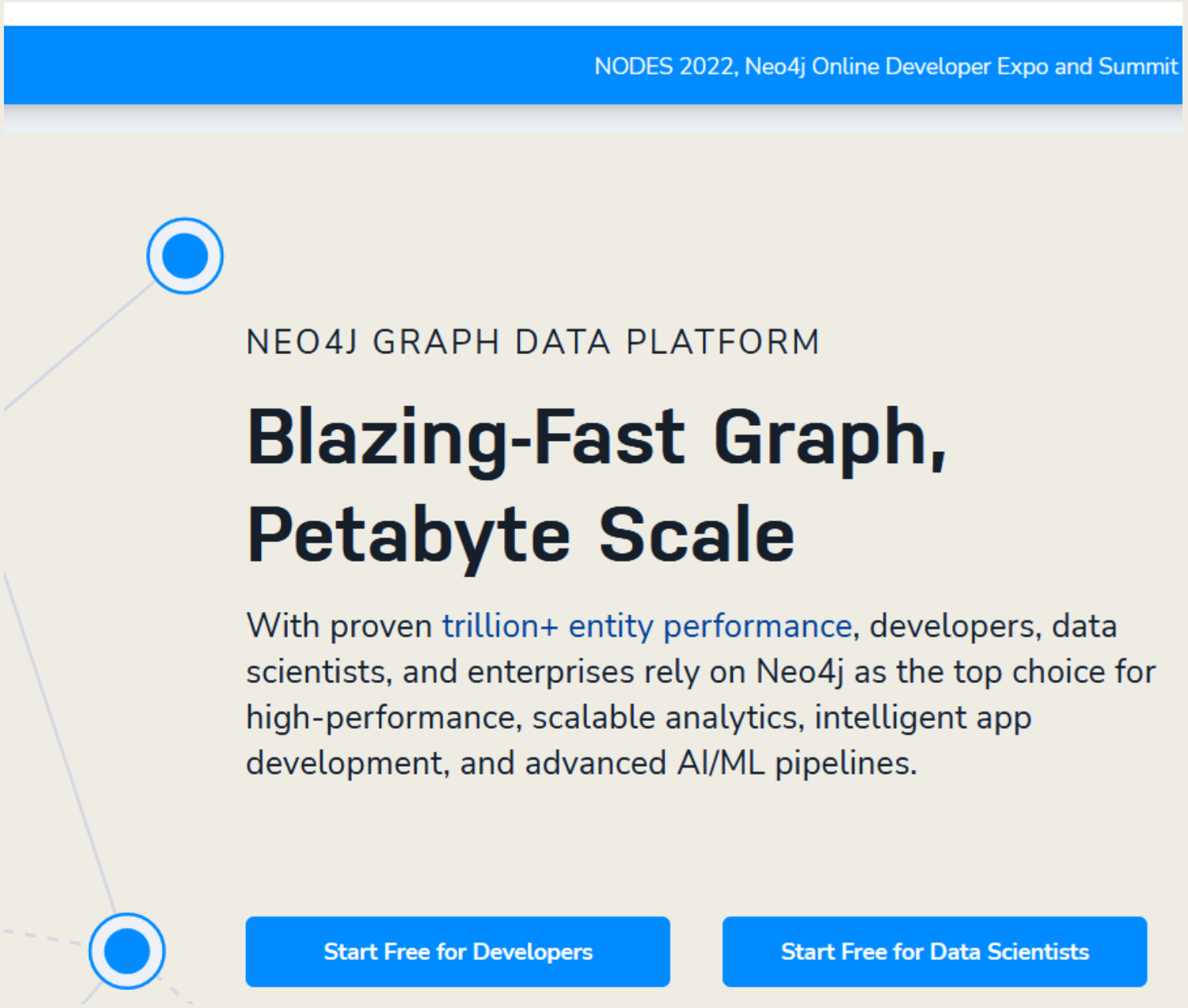
List of items purchased by Mario Rossi

Employers of people who like ‘Racism’



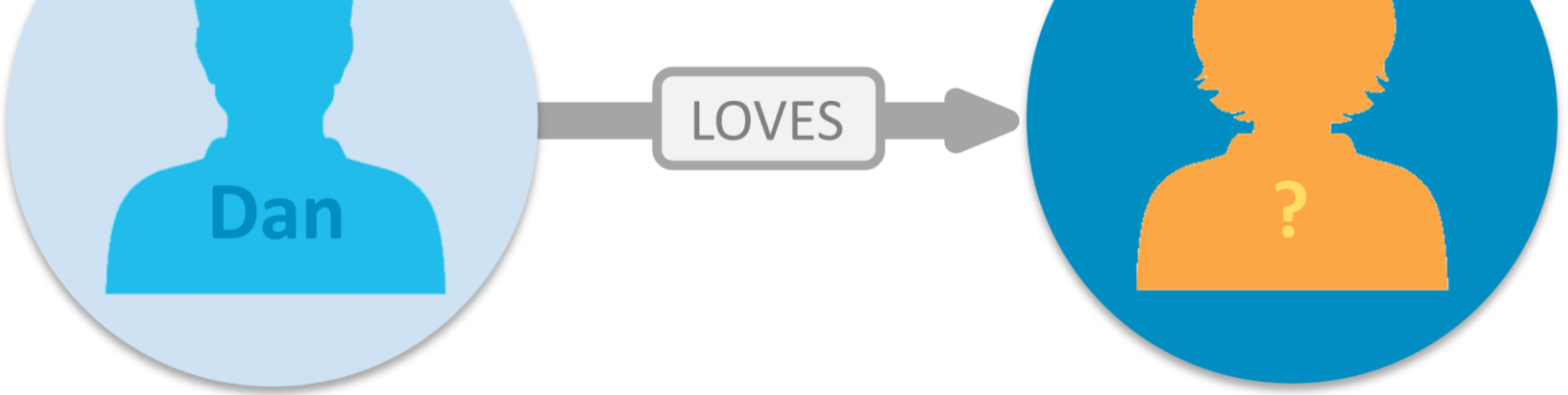
THE POWER OF ANSWERING UNEXPECTED QUERIES

https://actualfacebookgraphsearches.tumblr.com/



neo4j

*"With more than 800 customers, including **UBS, Comcast, eBay, Adobe, Levi Strauss & Co., Volvo Cars, Orange, and Airbus**, Neo4j is the world's most widely deployed graph database, enabling connected data applications for **more than 75% of the Fortune 100.**"*



NODE Relationship NODE

```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```

LABEL PROPERTY VARIABLE

CYPHER

Neo4j's graph query language. It is like SQL for graphs, and was inspired by SQL

https://neo4j.com/docs/cypher-manual/4.0/

Neo4j Cypher Refcard 4.4

Legend

- Read
- Write
- General
- Functions
- Schema
- Performance
- Multidatabase
- Security

Syntax

Read query structure

```
[USE]
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

MATCH

RETURN

`RETURN *`
Return the value of all variables.

`RETURN n AS columnName`
Use alias for result column name.

`RETURN DISTINCT n`
Return unique rows.

`ORDER BY n.property`
Sort the result.

`ORDER BY n.property DESC`
Sort the result in descending order.

`SKIP $skipNumber`
Skip a number of results.

`LIMIT $limitNumber`
Limit the number of results.

`SKIP $skipNumber LIMIT $limitNumber`
Skip results at the top and limit the number of results.

`RETURN count(*)`
The number of matching rows. See Aggregating functions for more.

WITH

```
MATCH (user)-[:FRIEND]-(friend)
WHERE user.name = $name
```

Operators

General	`DISTINCT`, `.`, `[]`
Mathematical	`+`, `-`, `*`, `/`, `%`, `^`
Comparison	`=`, `<>`, `<`, `>`, `<=`, `>=`, `IS NULL`, `IS NOT NULL`
Boolean	`AND`, `OR`, `XOR`, `NOT`
String	`+`
List	`+`, `IN`, `[x]`, `[x .. y]`
Regular Expression	`=~`
String matching	`STARTS WITH`, `ENDS WITH`, `CONTAINS`

null

- `null` is used to represent missing/undefined values.
- `null` is not equal to `null`. Not knowing two values does not imply that they are the same value. So the expression `null = null` yields `null` and not `true`. To check if an expression is `null`, use `IS NULL`.
- Arithmetic expressions, comparisons and function calls (except `coalesce`) will return `null` if any argument is `null`.
- An attempt to access a missing element in a list or a

https://neo4j.com/docs/cypher-refcard/current/

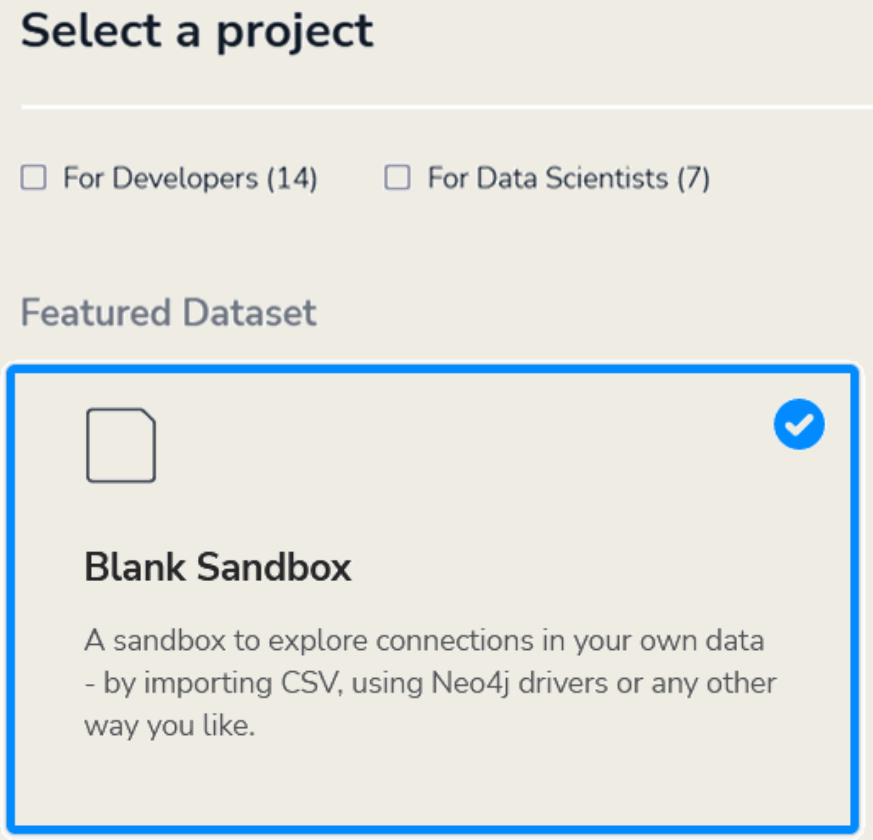
GQL

Some syntax

Description	Node	Edge
Generic	()	-- -> -[]- -[]->
With a **reference**	(n)	-[r]-
With a **label / edge type**	(:Person)	-[:ACTED_IN]-
With a label / edge type and an **inline property**	(:Person {name: 'Bob'})	-[:ACTED_IN {role: 'Dave'}]-
With a **reference**, label / edge type and an inline property	(p:Person {name: 'Bob'})	-[r:ACTED_IN {role: 'Dave'}]-

The Neo4j sandbox

- https://sandbox.neo4j.com/
- Free Sandbox environment, with remotely handled installation and browser integration
- No technical setup, just create an account at the link
- After the login, select the **Blank Sandbox** template project

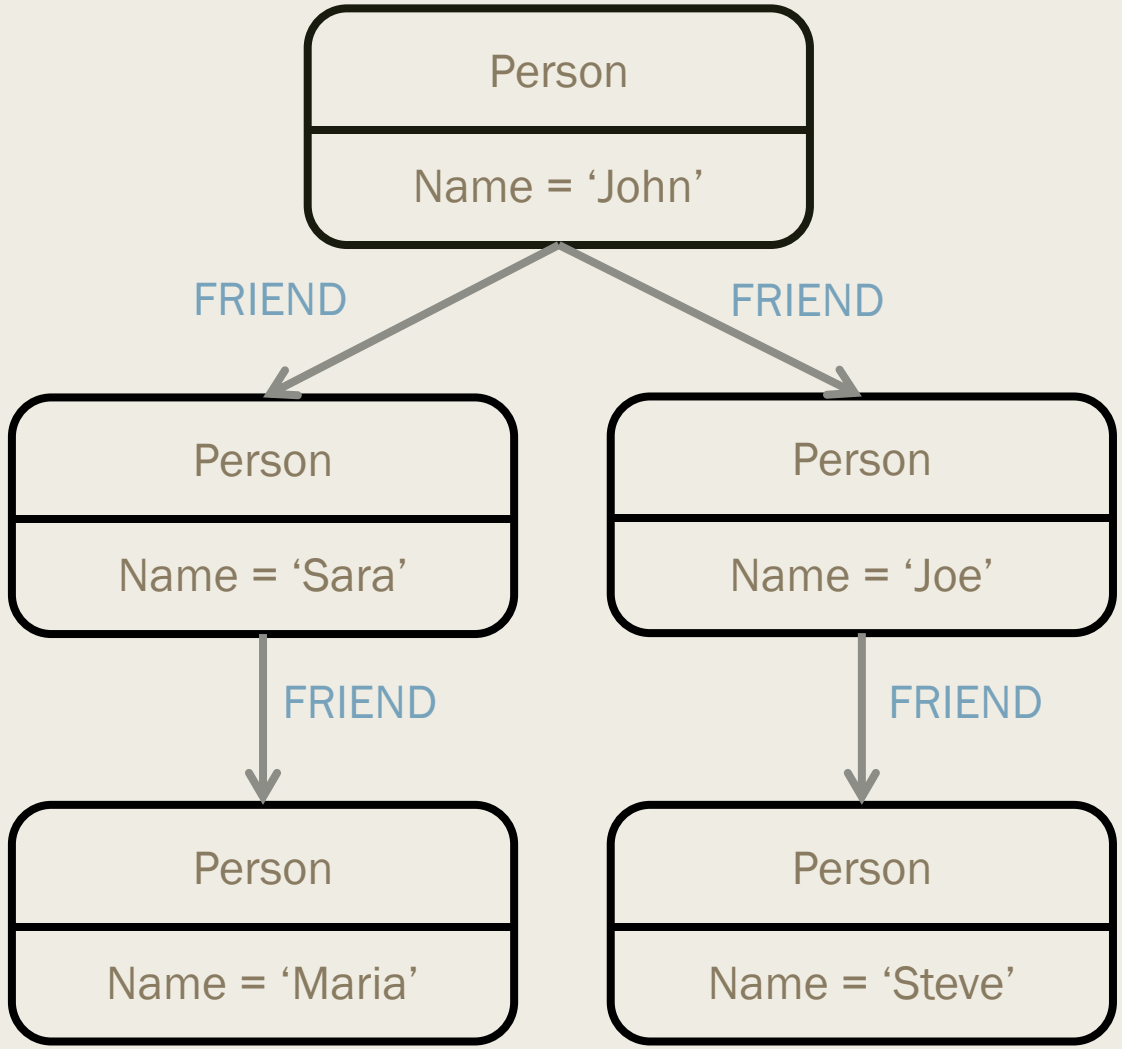


Creating a graph

Variable Node Type Properties

```
CREATE (john:Person {name: "John"})
CREATE (joe:Person {name: "Joe"})
CREATE (steve:Person {name: "Steve"})
CREATE (sara:Person {name: "Sara"})
CREATE (maria:Person {name: "Maria"})
CREATE (john)-[:FRIEND]->(joe)-[:FRIEND]->(steve)
CREATE (john)-[:FRIEND]->(sara)-[:FRIEND]->(maria)
```

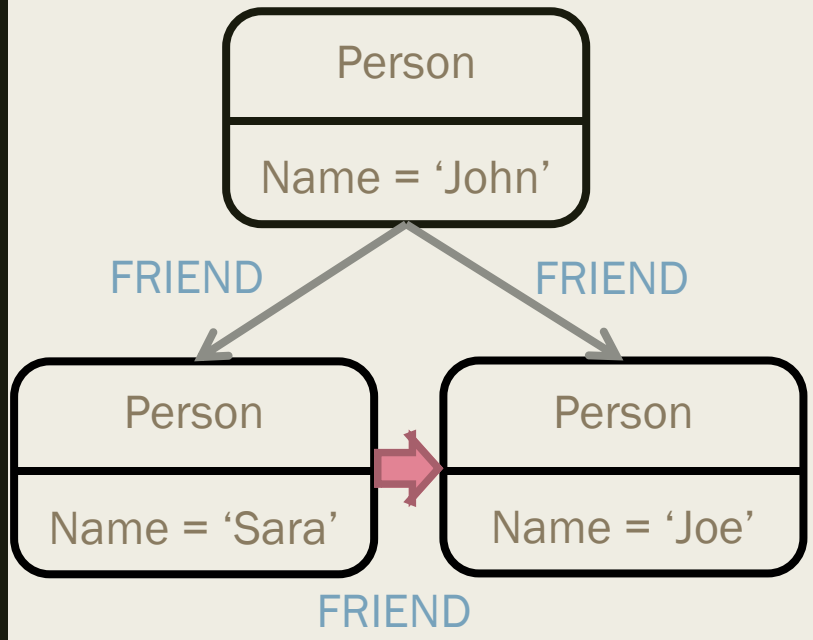
Relationship label



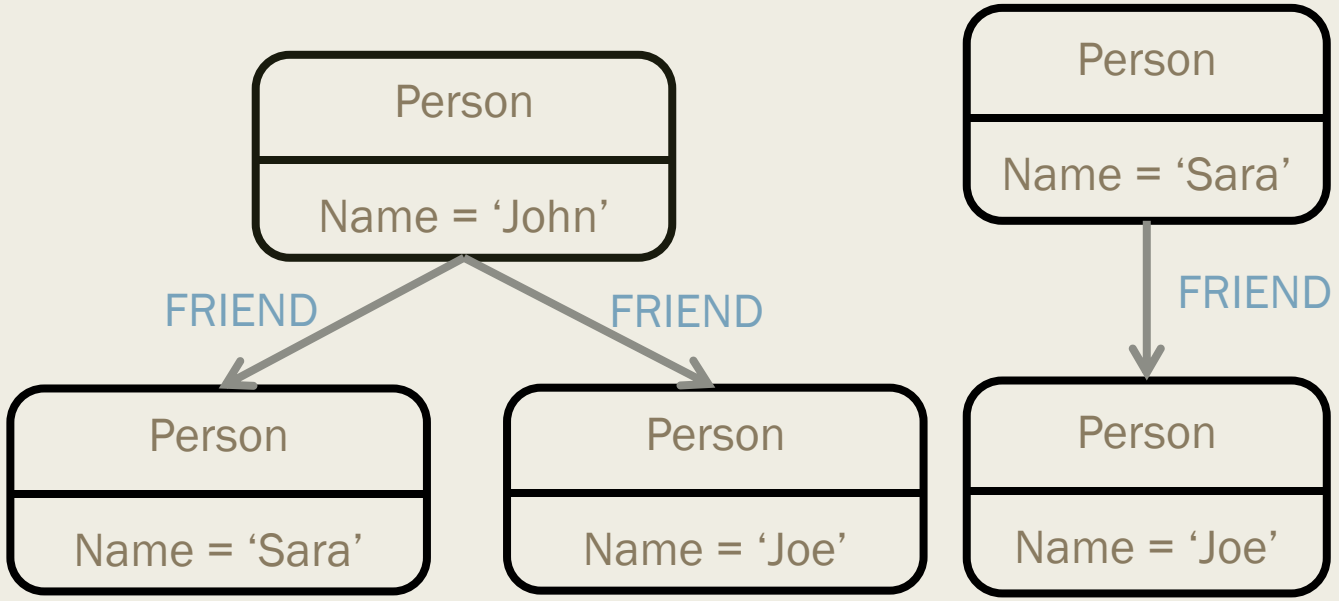
Careful – edge additon

CREATE (j:Person {name: "Sara"})-[rel:FRIEND]->(m:Person {name: "Joe"})

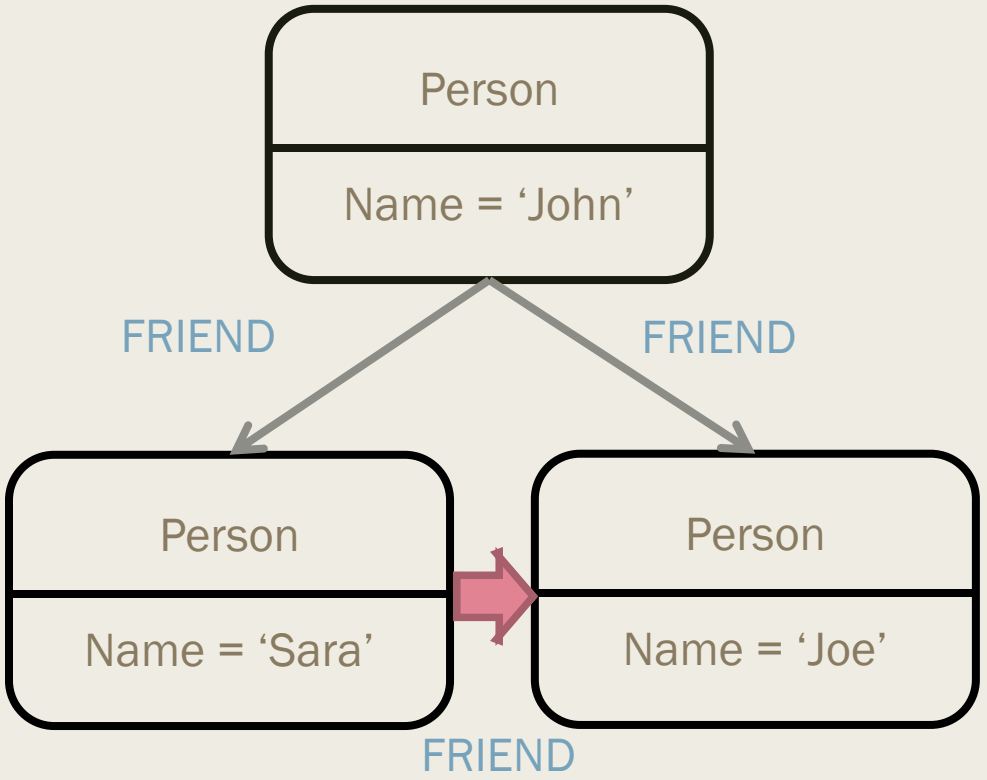
EXPECTED OUTCOME



REAL OUTCOME



How to add edges (properly)



To delete the two newly created nodes, click on them and check their Ids

MATCH (n) where ID(n)=<your_id>
DETACH DELETE n

```
MATCH (j:Person {name: "Sara"})
MATCH (m:Person {name: "Joe"})
MERGE (j)-[r:FRIEND]->(m)
RETURN j, r, m
```

Deletions

To delete (purge) the entire graph

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

To delete one node (by id)

```
MATCH (n) where elementId(n)=<your_id> DETACH DELETE n
```

To delete all the outgoing edges from a node with a property

```
MATCH (n: {property_name: "x"})-[r:REL_NAME]->() DELETE r
```

Let’s add some locations

CREATE (aus:Country {name: "Australia"})

CREATE (ger:Country {name: "Germany"})

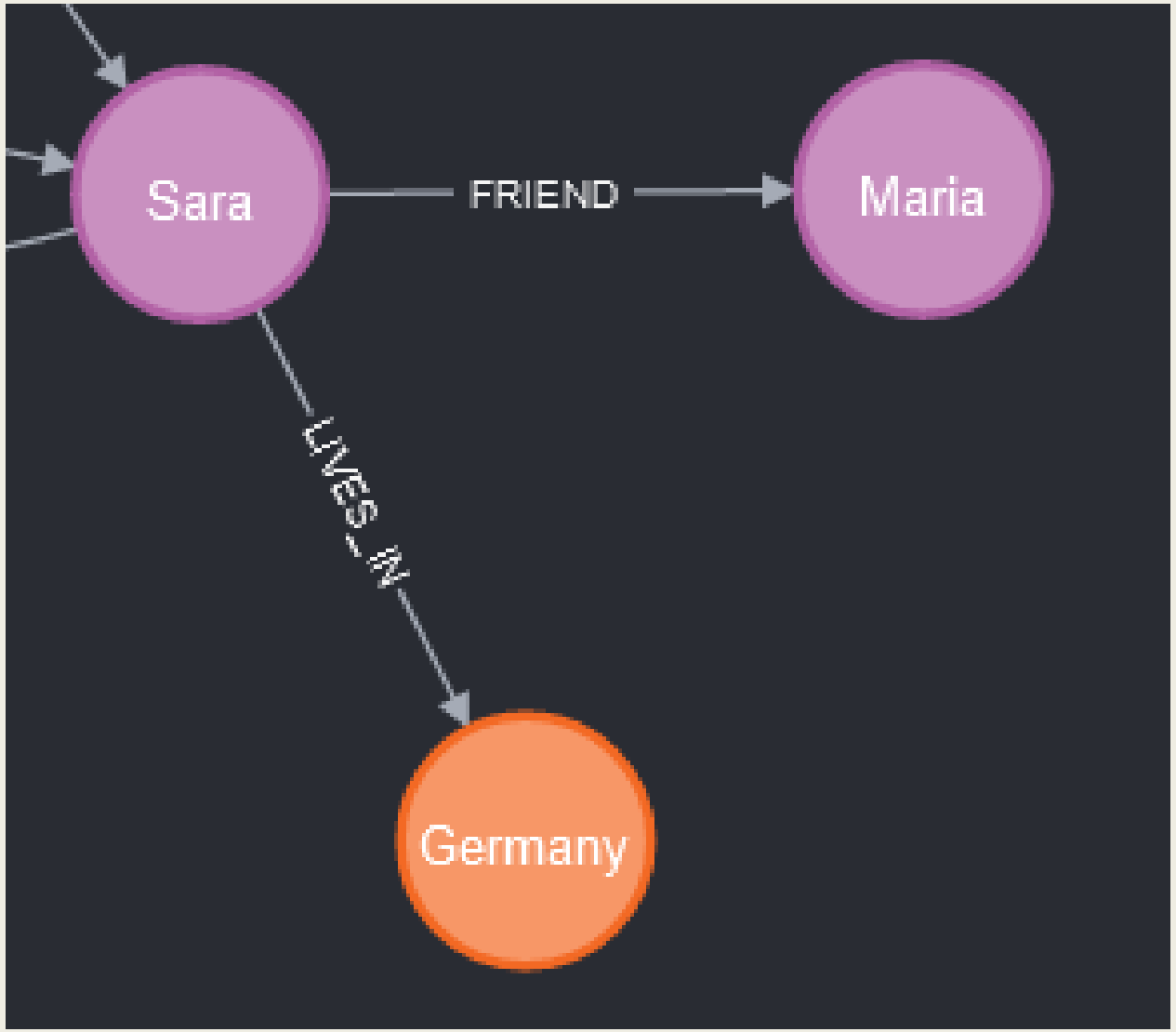
MATCH (j:Person {name: "Sara"})

MATCH (l:Country {name: "Germany"})

MERGE (j)-[r:LIVES_IN]->(l)

RETURN j, l

And add a state for each of the nodes in the graph



Some simple queries

You can use regex in your queries

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

Use the keyword "exists"

```
MATCH (p:Person)-[r:FRIEND]->(m:Person)
WHERE exists((m)-[:FRIEND]->(p))
RETURN m, p
```

Aggregate values

```
MATCH (p1:Person)-[:FRIEND]->(p2:Person)
RETURN p1.name, collect(p2.name), count(*) as numberOfFriends
```

https://regex101.com/

Exercise

Count the number of people who have **at least** one friend that lives in Australia

```
MATCH (p1:Person)-[:FRIEND]->(p2:Person)
WHERE exists((p2)-[:LIVES_IN]->(:Country {name: "Australia"}))
RETURN count(*) as value
```

Print the names of people who have **at least** one friend that lives in Australia

Some useful queries

Find all things related to a node

```
MATCH (p:Person {name: "Maria"})-[relatedTo]-(x)
return x.name, type(relatedTo), Labels(x)[0]
```

Labels[0] because we can add more than one label to a node

Limit hop size

```
MATCH (p:Person {name: "Maria"})-[*1..2]-(x)
return DISTINCT x
```

Try also adding an arrow (->) before (x)

Check the database schema

```
CALL db.schema.visualization()
```

Run Data Science Algorithms

To run data science algorithms, such as PageRank, we need to create an **in-memory projection** of the graph

```
CALL gds.graph.project('projection_name', ['Person', 'Country'], ['FRIEND', 'LIVES_IN'])

YIELD graphName AS graph, nodeProjection, nodeCount AS nodes, relationshipCount
AS rels
```

We can also keep the weights of the edges, if needed, by adding {relationshipProperties: 'weight'}

Verify that it worked

```
CALL gds.graph.list()
```

Generic Syntax

How a call to a graph data science algorithm looks like

```
CALL gds[.<tier>].<algorithm>.<execution-mode>[.<estimate>](
        graphName: String,
        configuration: Map
)
```

PageRank

```
CALL gds.pageRank.stream('projection_name')

YIELD nodeId, score

WITH gds.util.asNode(nodeId) AS n,score AS pageRank

RETURN n.name AS name, Labels(n)[0] AS type, pageRank

ORDER BY pageRank DESC
```

You can also write the scores of pagerank as properties of the graph

```
CALL gds.pageRank.write('projection_name',
    {writeProperty: 'pageRank'}
)
YIELD nodePropertiesWritten, ranIterations
```

Shortest paths

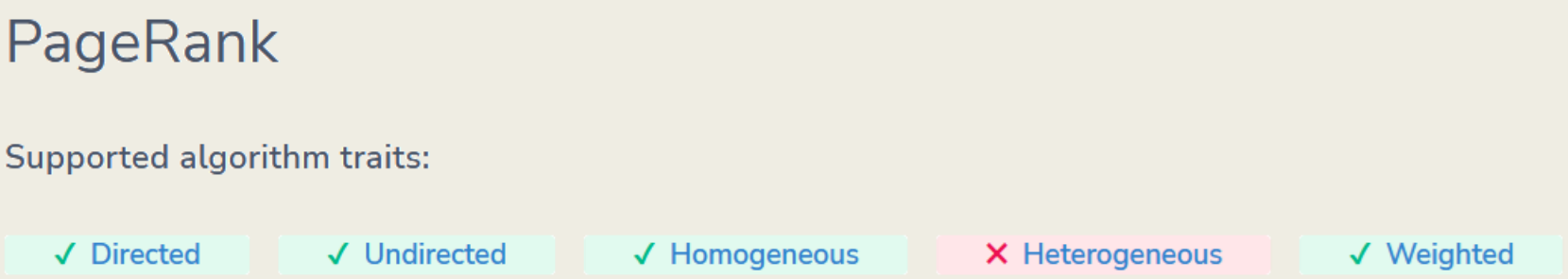
```
MATCH (source:Person {name: 'Maria'})
MATCH (target:Person {name: 'Joe'})
CALL gds.shortestPath.dijkstra.stream('projection_name', {
    sourceNode: source,
    targetNode: target
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
    index, gds.util.asNode(sourceNode).name AS SourceName,
    gds.util.asNode(targetNode).name AS targetNodeName,
    totalCost, [nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,
    costs, nodes(path) as path
ORDER BY index
```

To consider weighted graphs
relationshipWeightProperty: 'weight'

Which graph type is supported?

You can verify, for each algorithm, which graph type is supported, by visiting its documentation page:

https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/



Importing / Exporting Graphs

Write the graph to a csv

```
CALL apoc.export.csv.all('my_dataset.csv', {})
```

or on the sandbox: CALL apoc.export.csv.all(null, {stream:TRUE})

Now purge the graph

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

Now load the same graph through the csv

```
LOAD CSV FROM 'https://data.neo4j.com/bands/artists.csv' AS line
CREATE (:Artist {name: line[1], year: toInteger(line[2])})
```

Loading a file from the web

The Neo4J Sandbox does not allow us to load files from our computer, but we can use online accessible files.

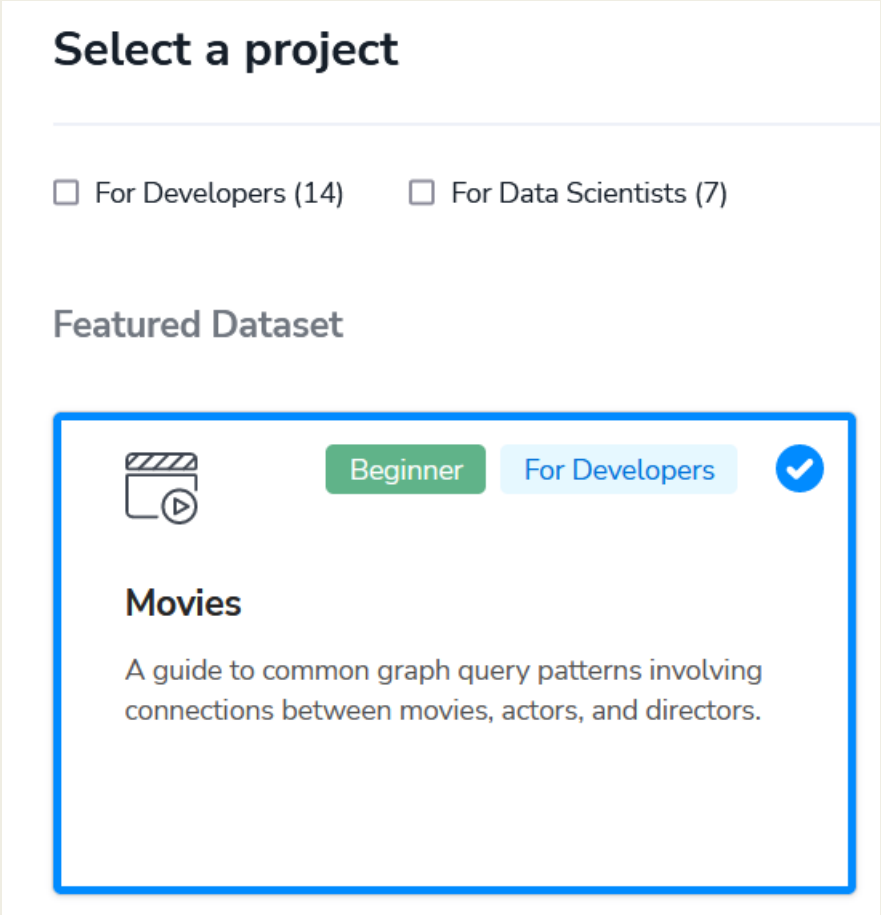
It is very common to have nodes and edges as two different csv files.

```
LOAD CSV WITH HEADERS FROM
"https://raw.githubusercontent.com/LorenzoBellomo/InformationRetrieval/main/data/nodes.csv" AS line

CREATE (:Person {name: line.name, lastName: line.last_name, my_id: line._id})
```

```
LOAD CSV WITH HEADERS FROM
"https://raw.githubusercontent.com/LorenzoBellomo/InformationRetrieval/main/data/edges.csv" AS line

MATCH (p1:Person {my_id: line.source})

MATCH (p2:Person {my_id: line.target})

MERGE (p1) –[:SUPERVISOR]->(p2)
```

Some Exercises

- Now let's go back to https://sandbox.neo4j.com/
- Select the **Movies** project
- Let's do some exercises on the dataset
- Remember to run:
  - CALL db.schema.visualization()



Exercises

- Which movies has Keanu Reeves starred in?
- Which directors directed the movies of Keanu Reeves

References

- Cypher introductory tutorial
  https://neo4j.com/developer/cypher/intro-cypher/
- Cypher docs
  https://neo4j.com/docs/cypher-manual/current/
- Data Science Library docs
  https://neo4j.com/docs/graph-data-science/current/
- Install Neo4j
  https://neo4j.com/docs/operations-manual/current/installation/
- Run Neo4j through Docker
  https://neo4j.com/docs/operations-manual/current/docker/