Introduction to Graph Databases

Neo4j

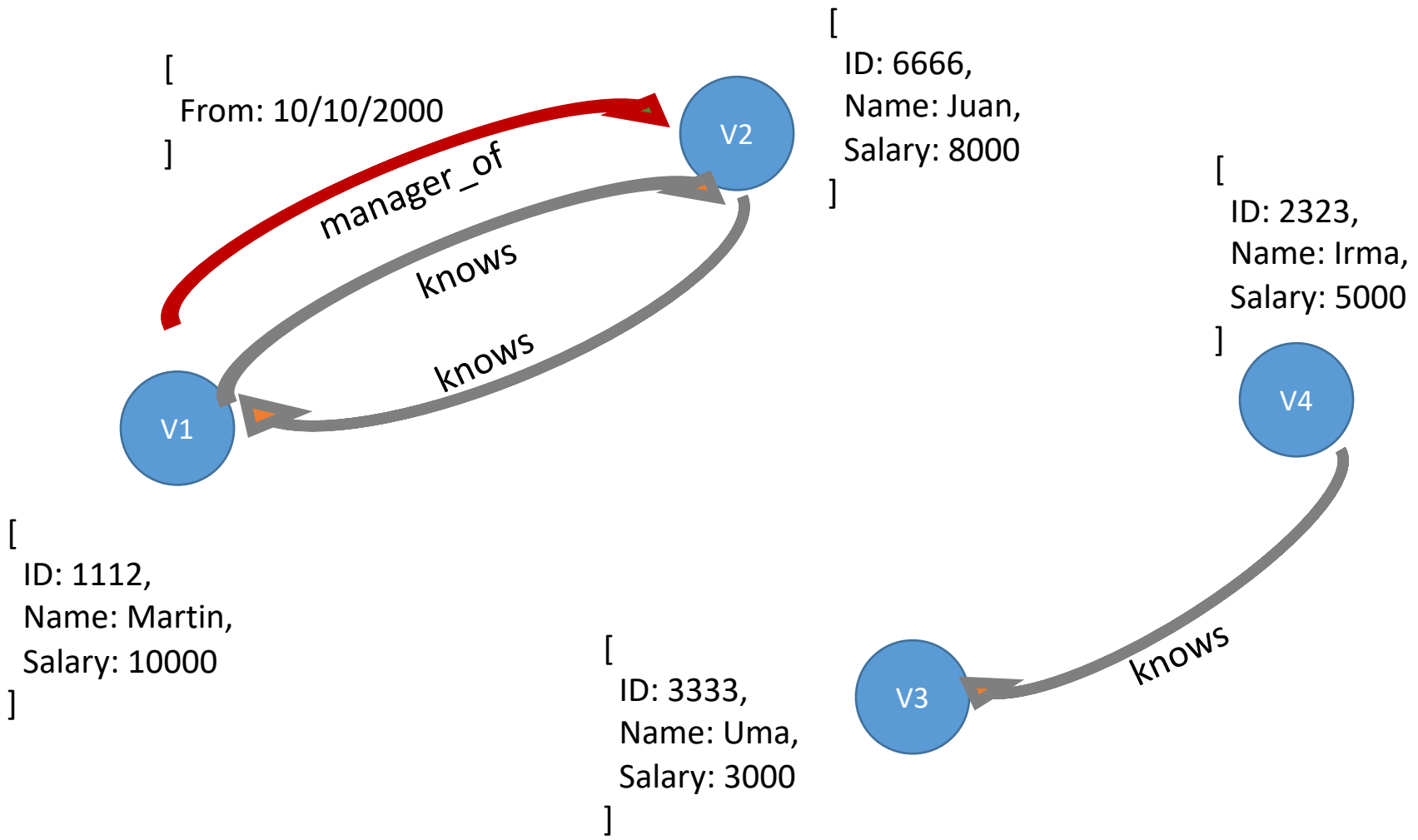
Alejandro Vaisman
avaisman@itba.edu.ar

Agenda

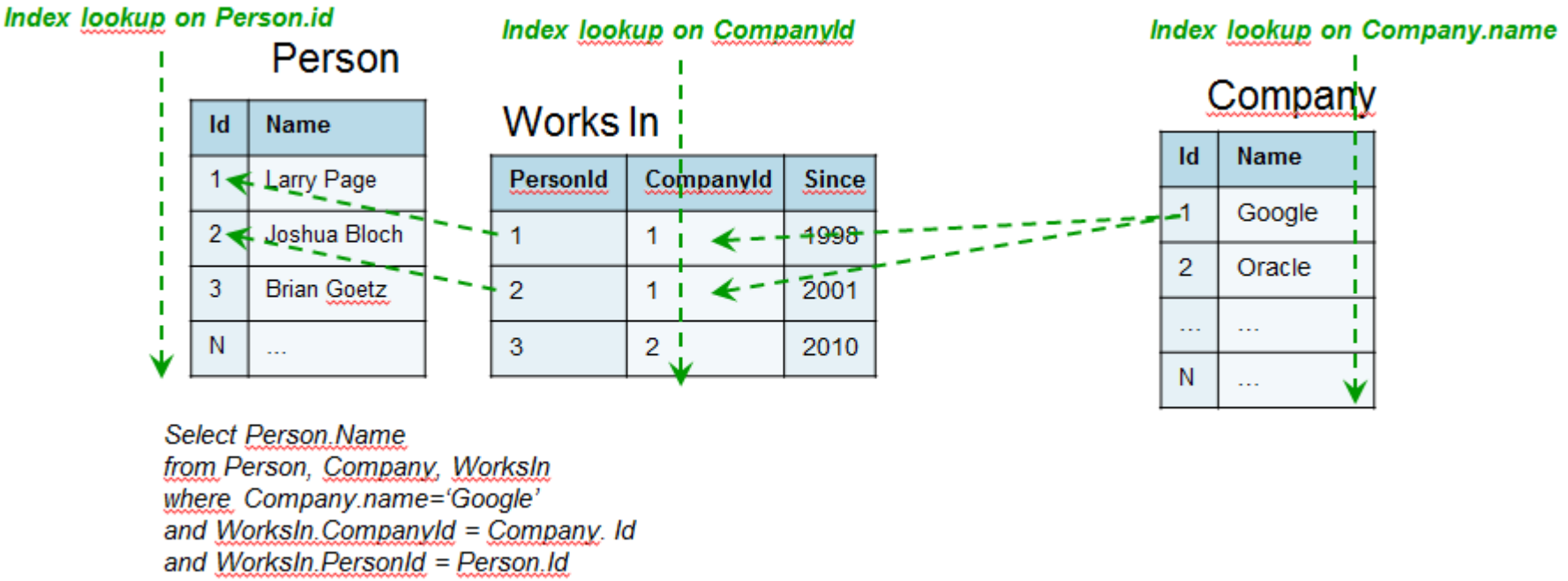
- 10.10.22. Introduction – Graph data models
- 13.10.22. Graph DB internals. Introduction to Neo4j
- 17.10.22. **Querying Neo4j databases**
- 20.10.22. Assignment 1. Graphs in relational databases
- 24.10.22. Assignment 2. Basic Cypher queries
- 27.10.22. Assignment 3. Advanced Cypher queries

Querying Neo4j Databases

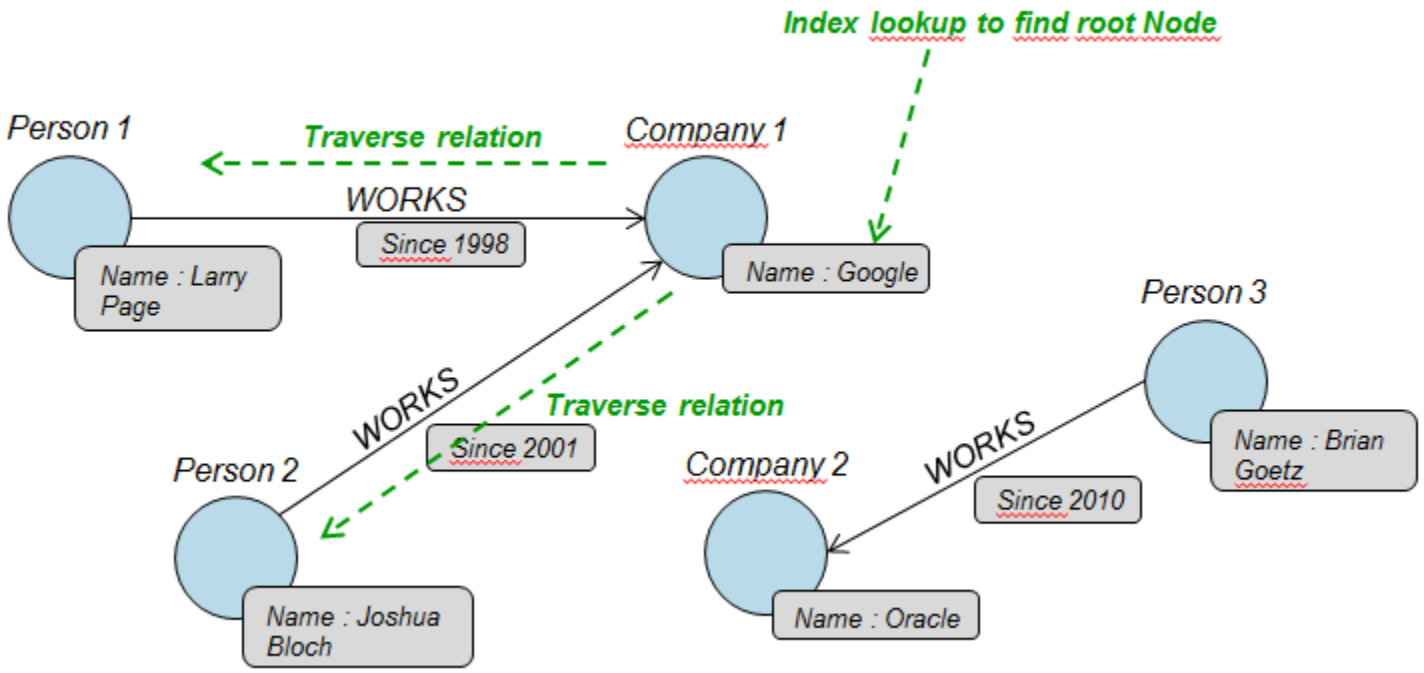
Property graphs revisited



Typical SQL query



Same query on graphs

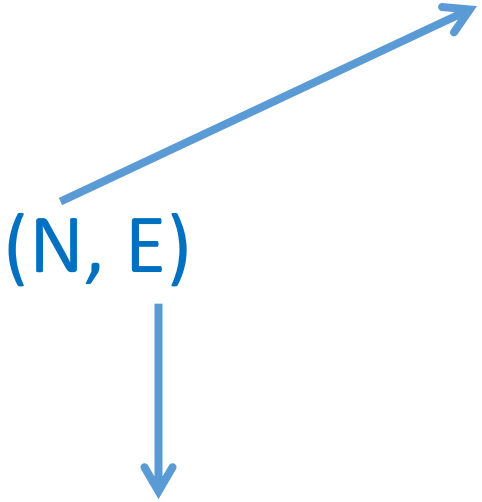


The deepest the navigation, the largest the difference with RDBs

GDBs: Neo4j www.neo4j.com

- Open Source.
- Versions for Linux, Win, Mac. Implemented in Java.
- High-level query language: Cypher.
- Customers: Lufthansa, Linkedin, InfoJobs, gameSys, eBay, FiftyThree, Accenture, National Geographic, CISCO, HP, Telenor, etc.

A Neo4j graph



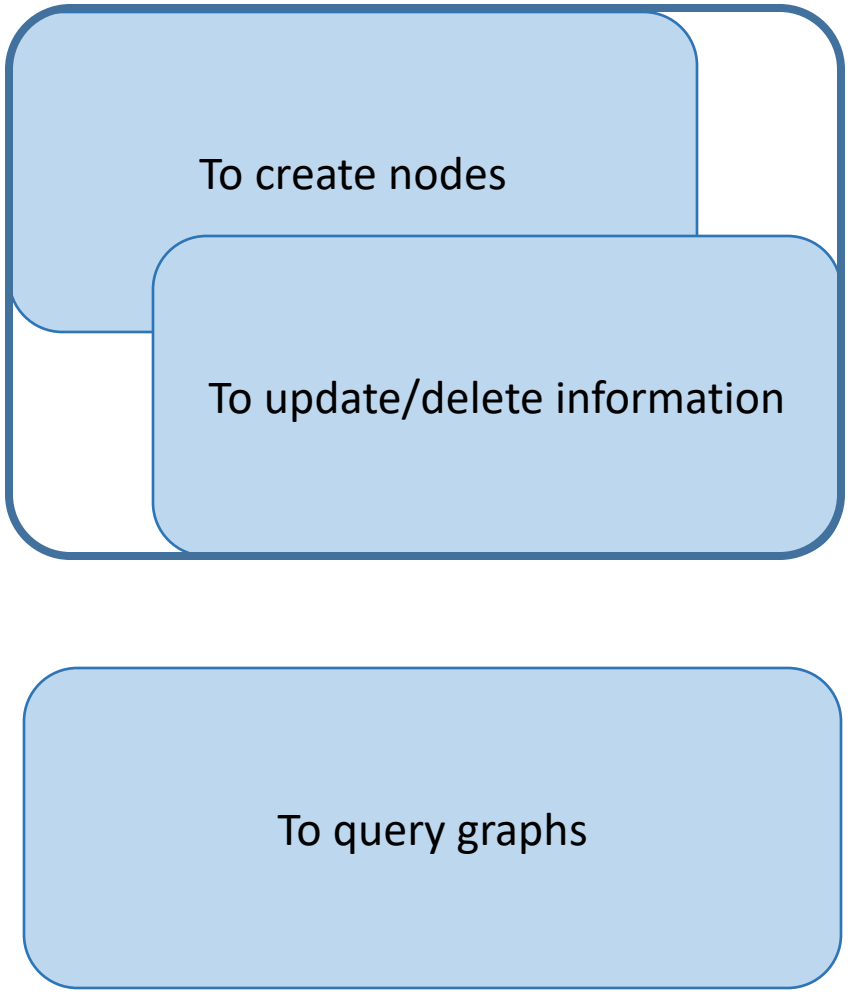
A set of nodes

A node has:
1) **Zero or + properties:** boolean, string, numeric, arrays of the former.
2) **Zero or + Labels**: Give a name to a node.

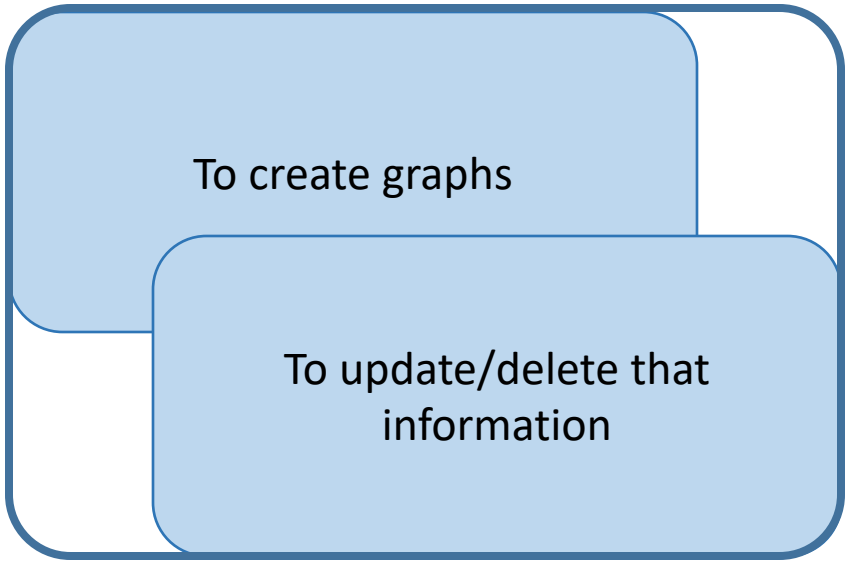
Directed edges (probably with cycles)

An edge has:
1) **Zero or + properties:** Sams as with nodes.
2) Exactly one **Label**: To distinguish a relationship between nodes.

Cypher


To create nodes
To update/delete information
To query graphs

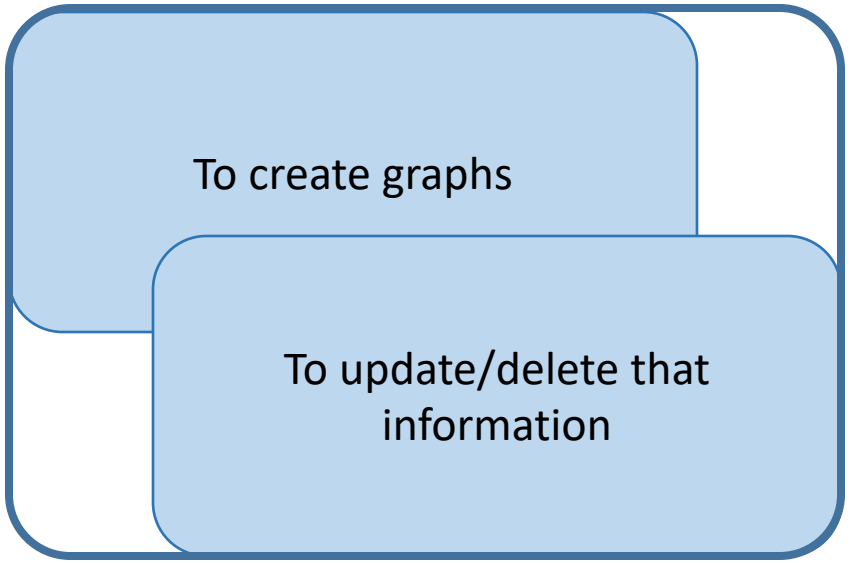
Cypher



Different from the relational model where:

1) First, the structure is created, to store tuples.
2) FKs are defined at the structural level.
3) Then, tuples are inserted/updated/deleted, and must conform to the structure.

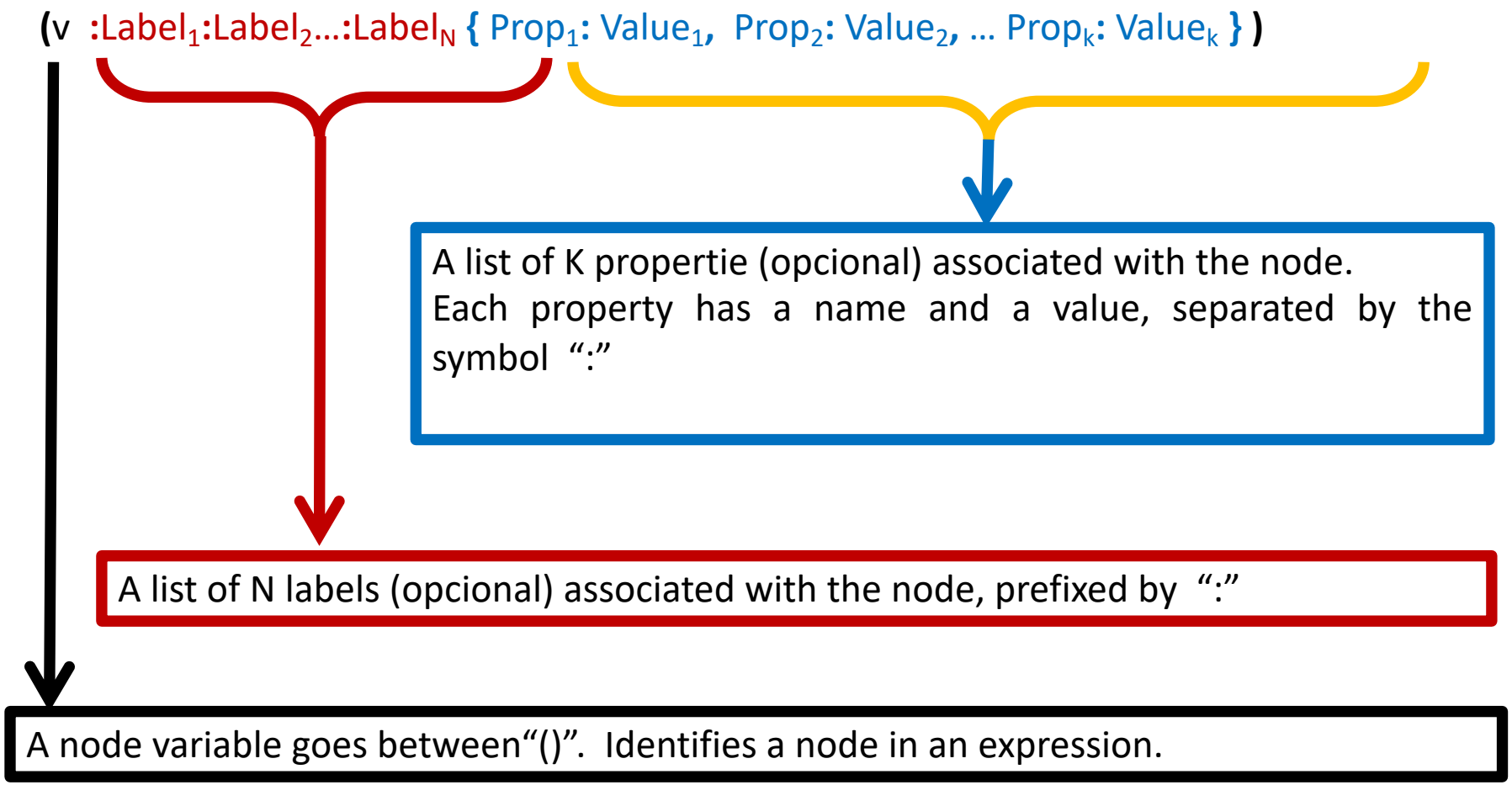
Cypher



Nodes and edges are created. Properties, labels, types, are the informational structure, but no schema is defined.

Topology can be thought as analogous to the FK in the relational model. Defined at the instance level.

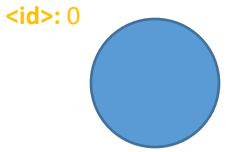
Cypher - nodes



Cypher - nodes

Create a node with no properties/labels:

```
$ CREATE (v)
  RETURN v;
```



ID assigned internally, with a different number each time. Can be reused by the system. Do not use it in applications.

Create another one.

```
$ CREATE ();
If RETURN is not written, nodes are not displayed
```

Cypher - nodes

Create a node with two labels:

```
$ CREATE (v  :Student:ITBA)
  RETURN v;
```

Student ITBA <id>: 2

Create a node with one label and 3 properties:

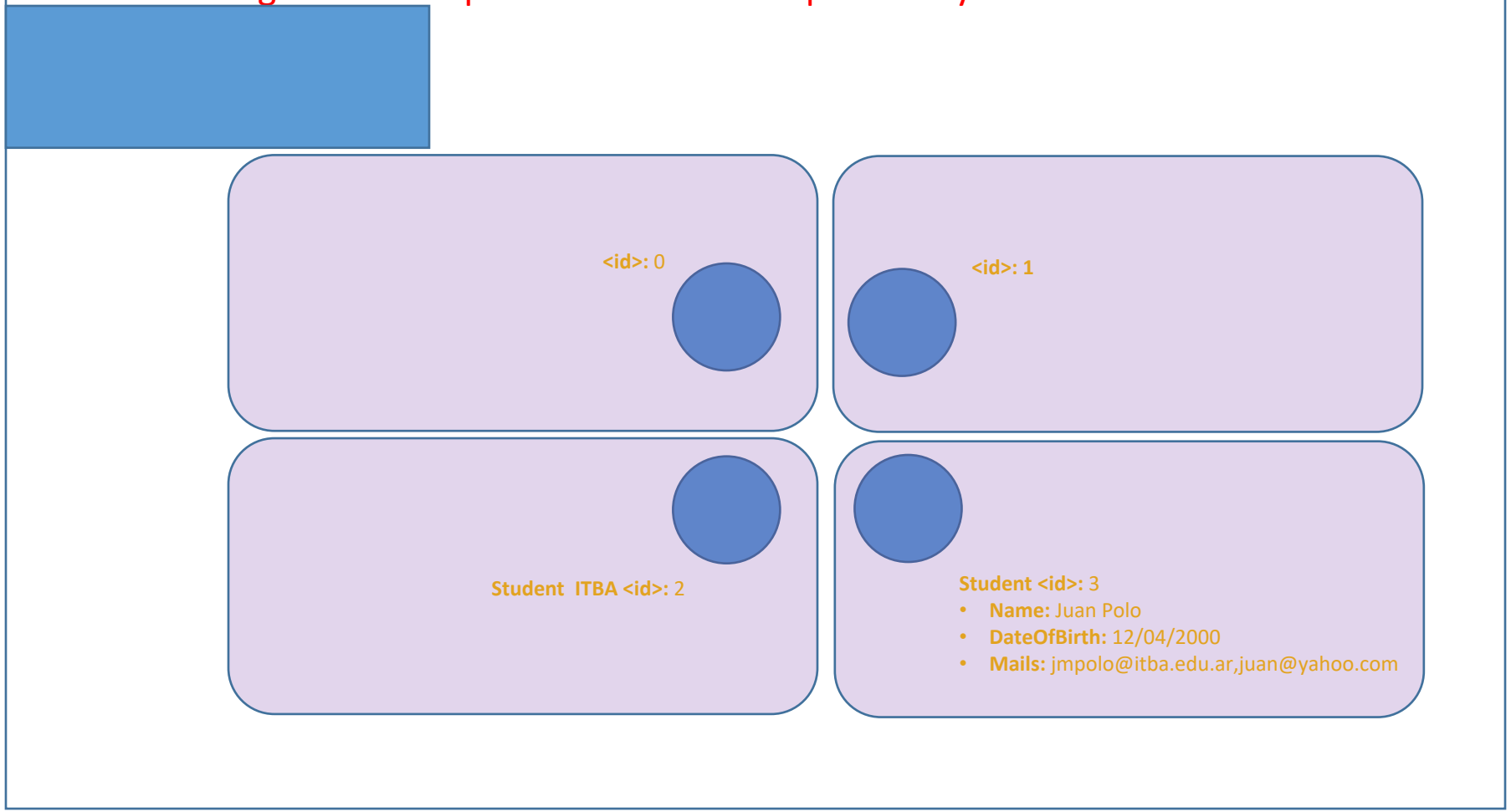
```
$ CREATE (n  :Student {Name: 'Juan Polo',
                       DateOfBirth: '12/04/2000',
                       Mails: ['jmpolo@itba.edu.ar', 'juan@yahoo.com']   })
 RETURN n;
```

Student <id>: 3

- **Name:** Juan Polo
- **DateOfBirth:** 12/04/2000
- **Mails:** jmpolo@itba.edu.ar,juan@yahoo.com

Cypher - nodes

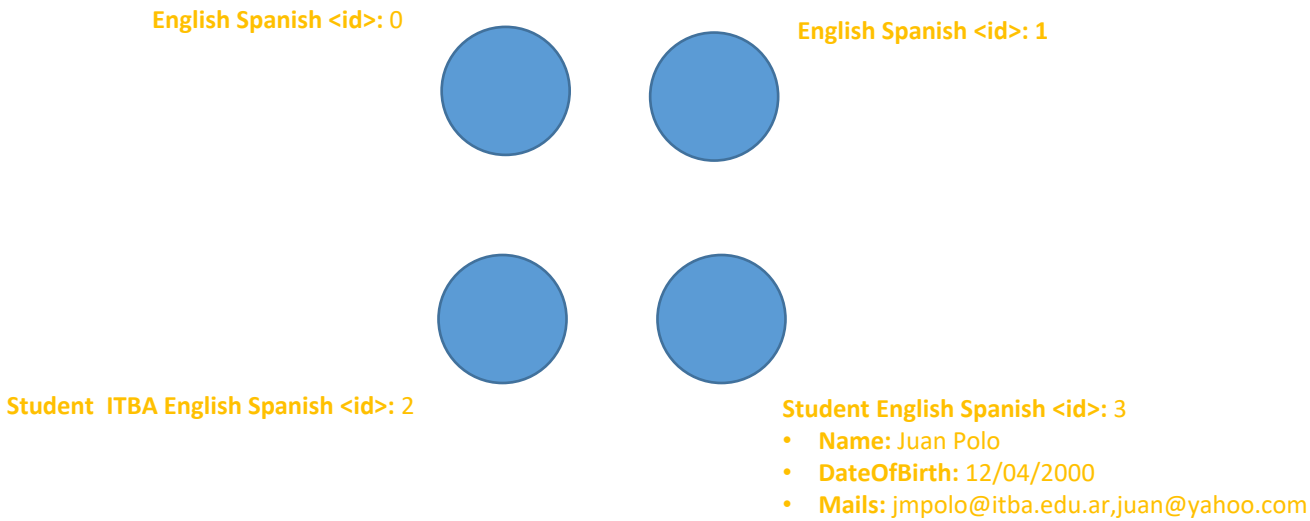
Add labels “English” and “Spanish” to all nodes previously created.



Cypher - nodes

Add labels “English” and “Spanish” to all nodes previously created.

```
$ MATCH (n)
  SET n  :English:Spanish
  RETURN n;
```

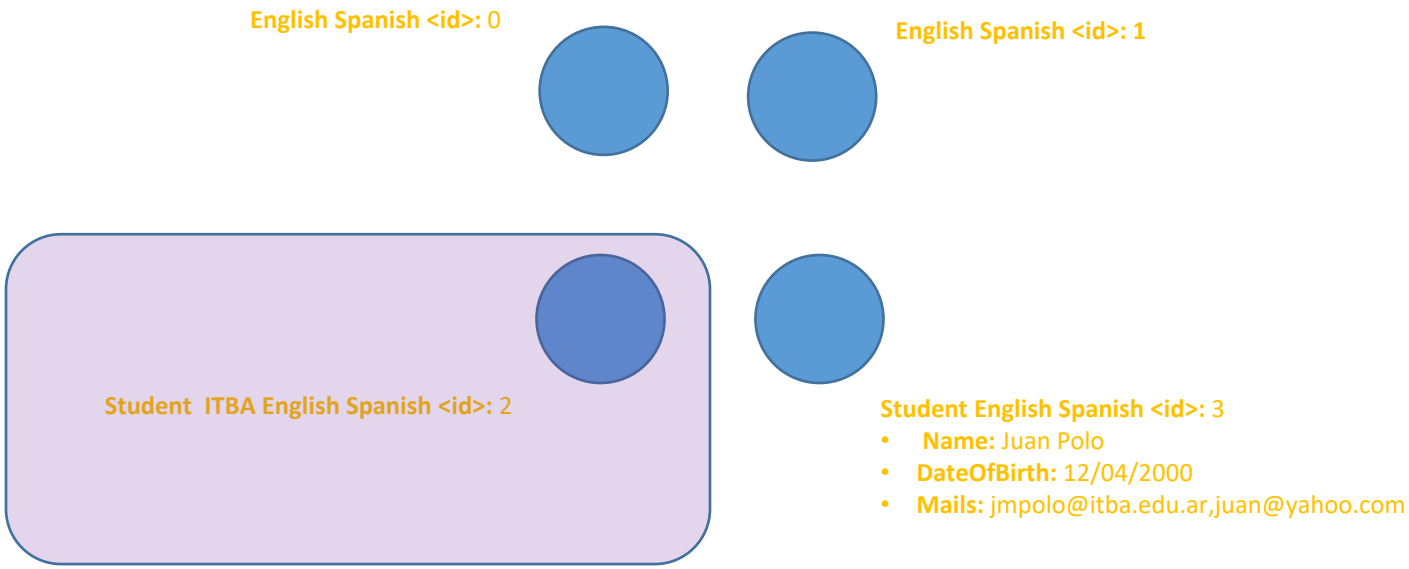


Cypher - nodes

Delete labels English and Spanish from the node labelled “ITBA”

$

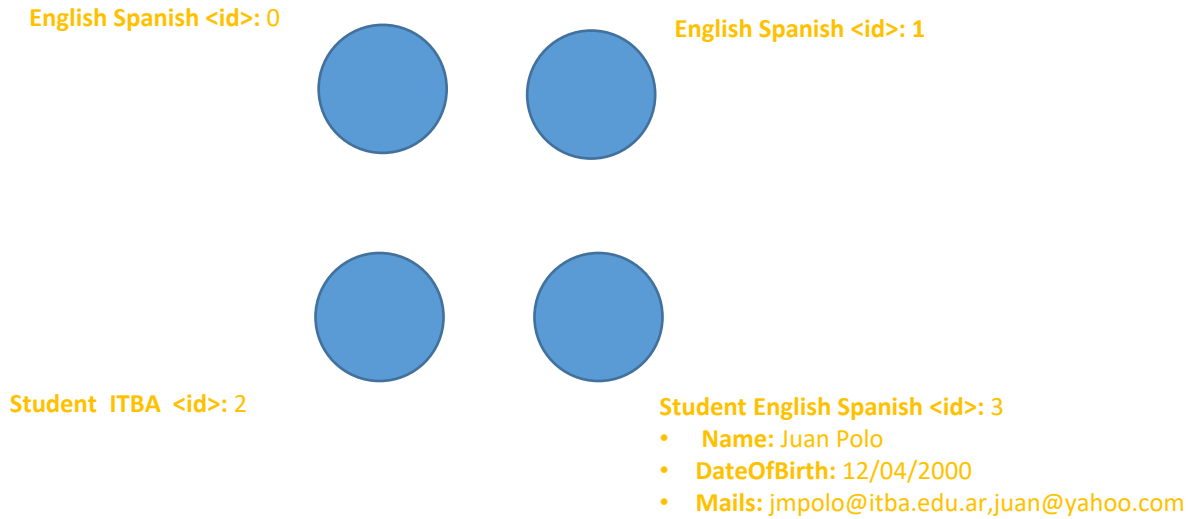
MATCH (n :ITBA)



Cypher - nodes

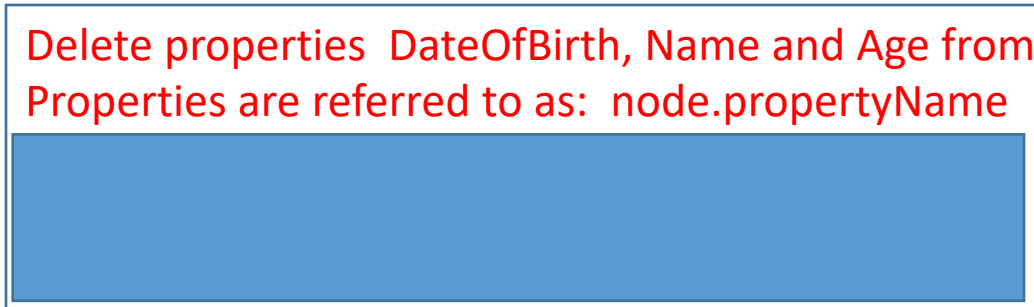
Delete labels English and Spanish from the node labelled “ITBA”

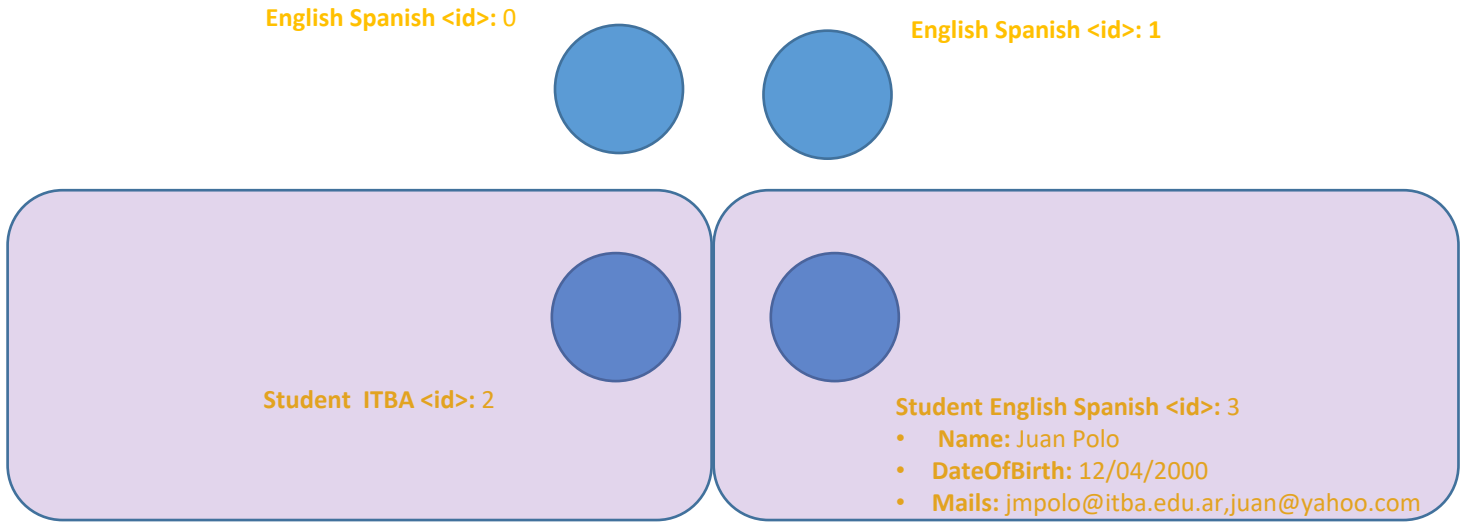
```
$ MATCH (n :ITBA)
  REMOVE n :English:Spanish
```



Cypher - nodes

Delete properties DateOfBirth, Name and Age from the nodes labelled “Student”.
Properties are referred to as: node.propertyName

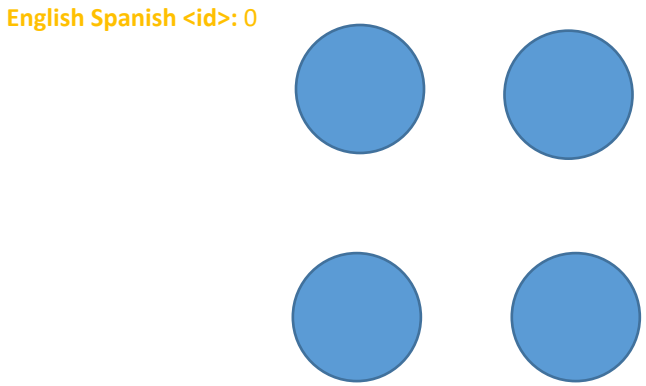




Cypher - nodes

Delete properties DateOfBirth, Name and Age from the nodes labelled “Student”.
Properties are referred to as: node.propertyName

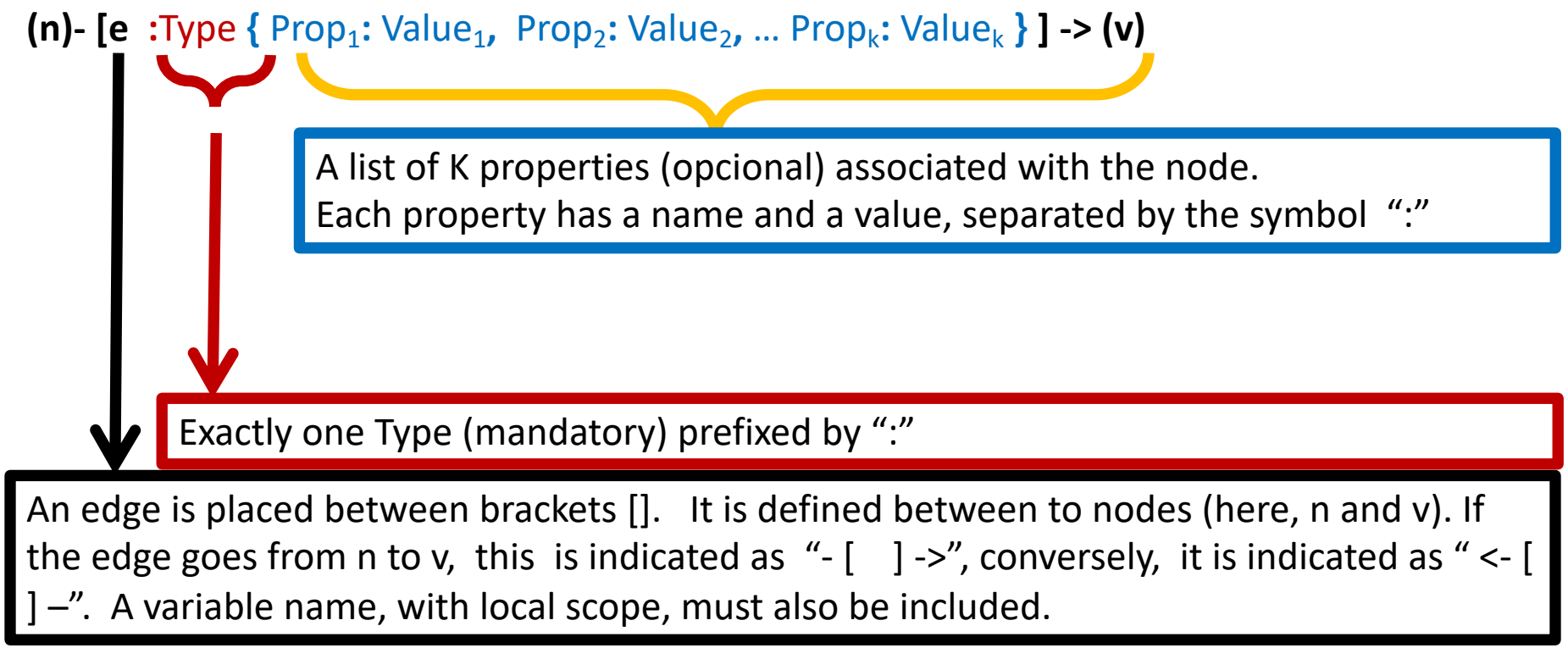
```
$ MATCH (n  :Student)
  REMOVE n.DateOfBirth, n.Name, n.mails, n.edad
  RETURN n
```



Undefined properties are ignored, the do not produce errors when trying to delete them. The same for labels.

Note that property “mails” was not deleted, language is case sensitive.

Cypher - Edges



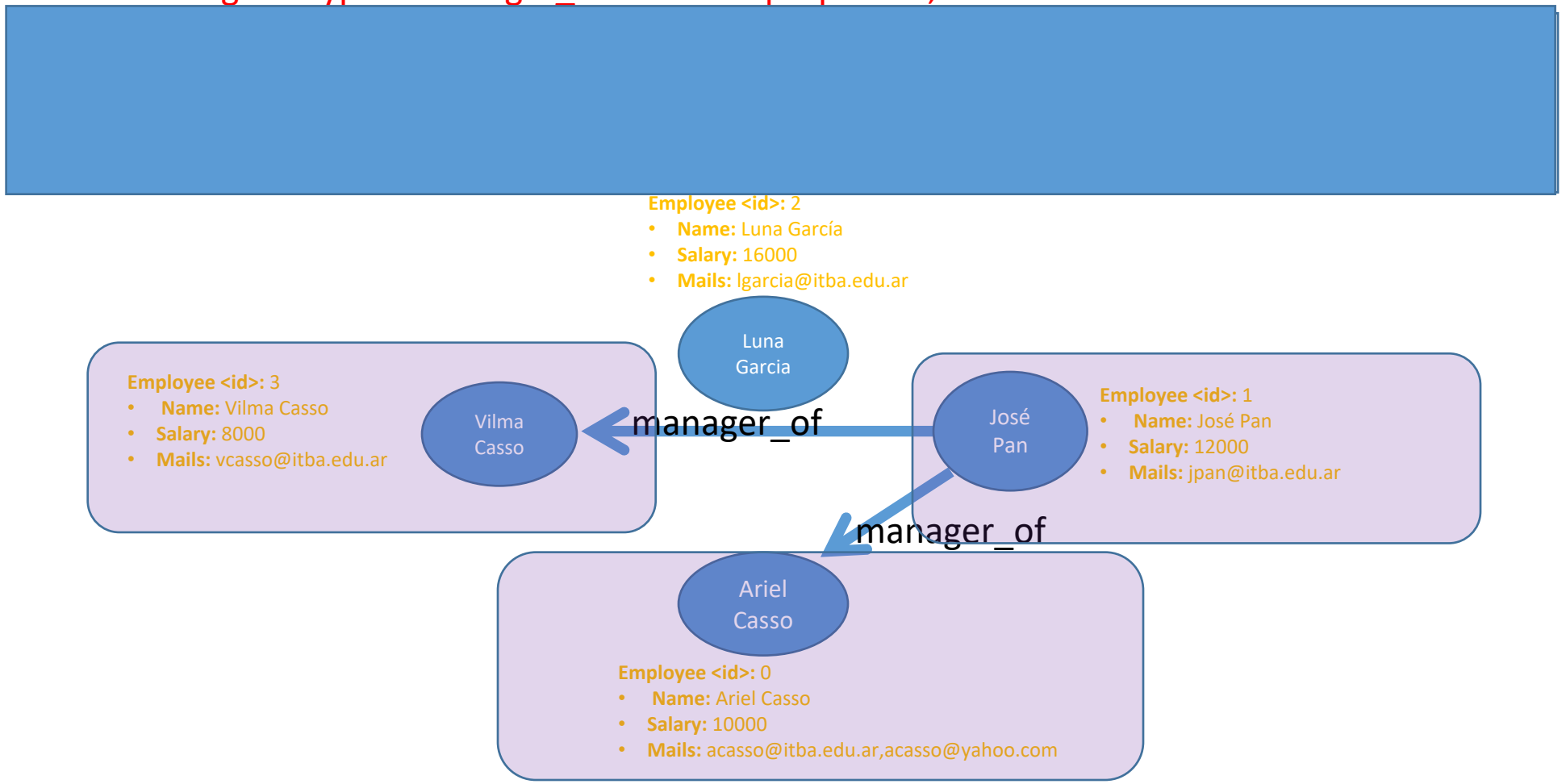
Cypher - Edges

Consider a Neo4j database. The nodes already created are:

```
$ CREATE (n :Employee { Name: 'Ariel Casso',
  Salary: 10000,
  Mails: ['acasso@itba.edu.ar', 'acasso@yahoo.com'] });

 CREATE (n :Employee { Name: 'José Pan',
 Salary: 12000,
 Mails: ['jpan@itba.edu.ar'] });

 CREATE (n :Employee { Name: 'Luna García',
 Salary: 16000,
 Mails: ['lgarcia@itba.edu.ar'] });

CREATE (n :Employee { Name: 'Vilma Casso',
Salary: 8000,
Mails: ['vcasso@itba.edu.ar'] });
```

Cypher - Edges

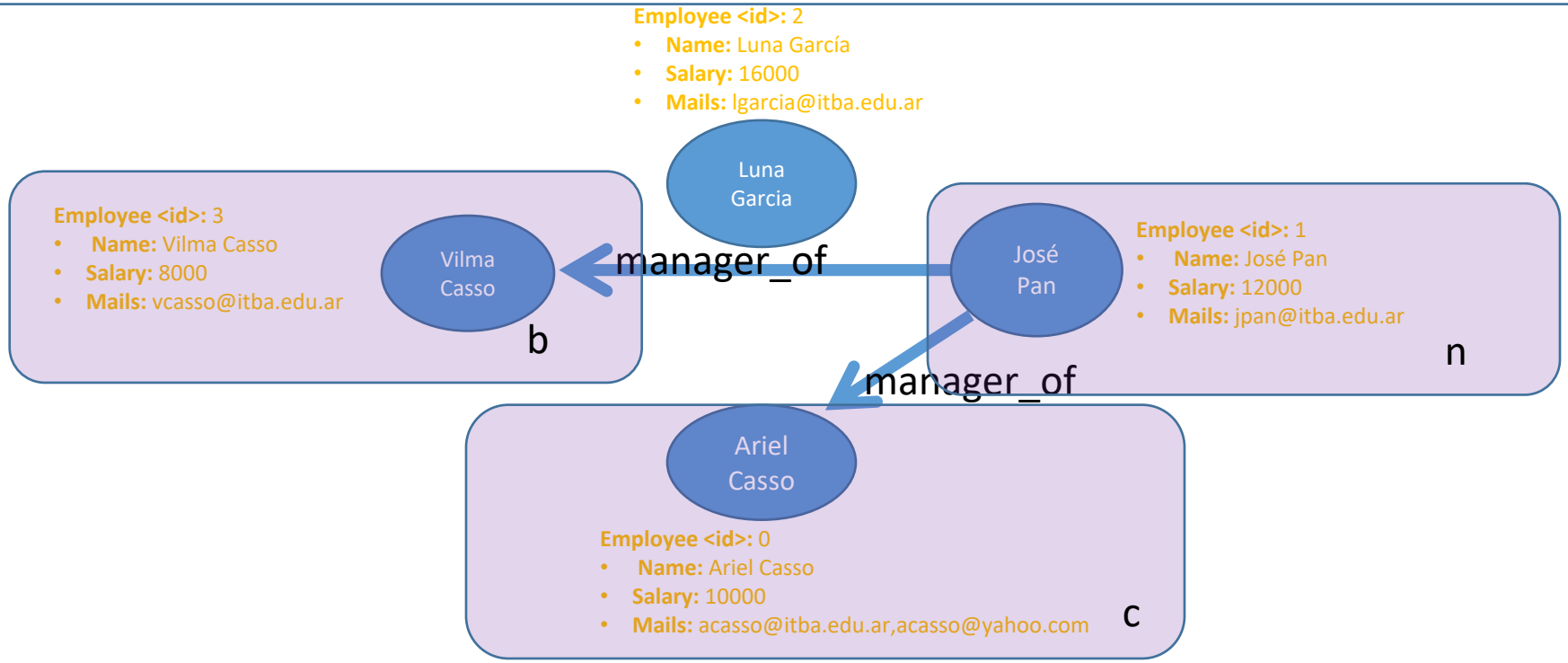
Create an edge of type «manager_of» with no properties, from José Pan to Vilma and Ariel Casso:



Cypher - Edges

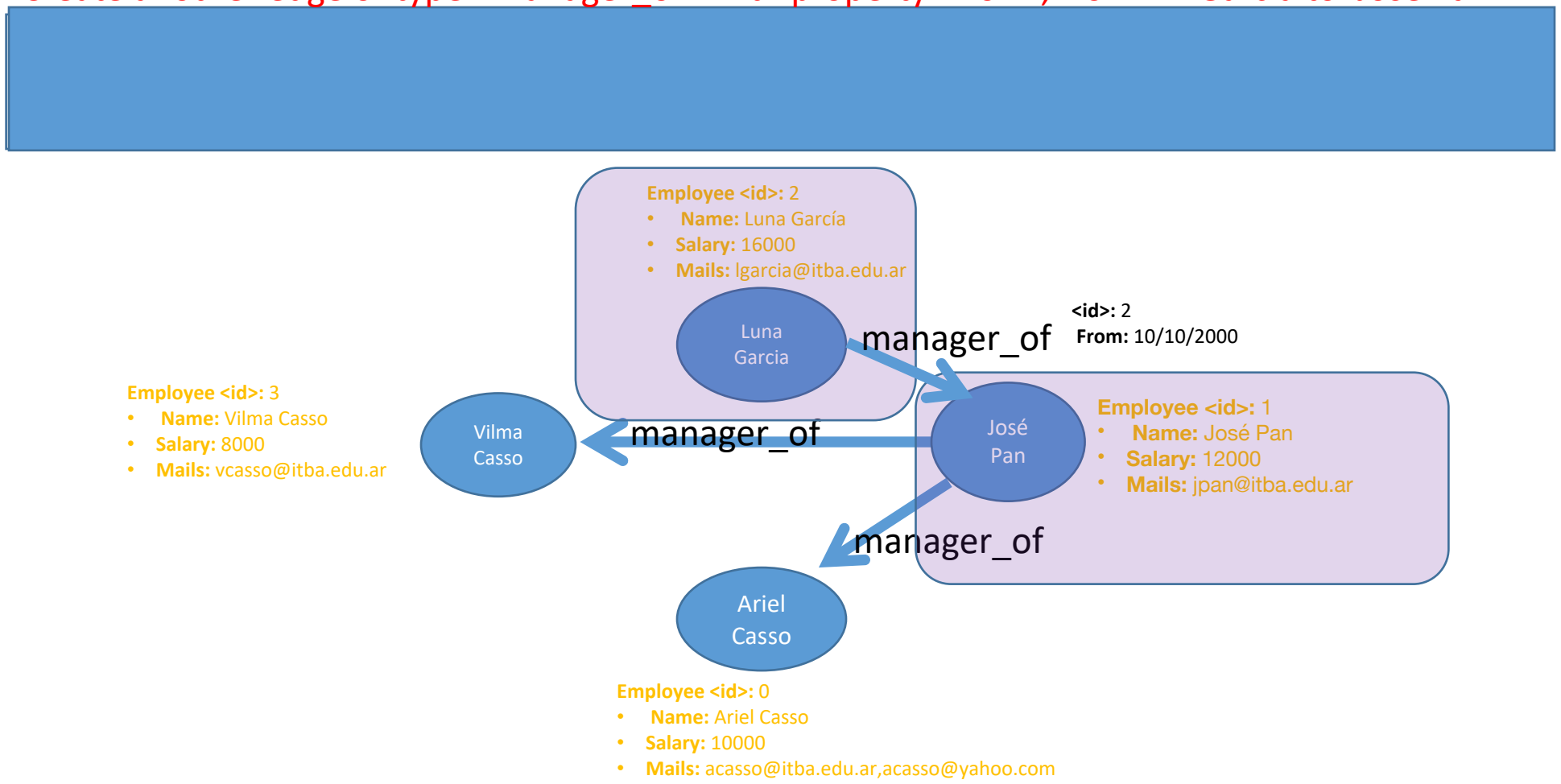
Create an edge of type «manager_of» with no properties, from José Pan to Vilma and Ariel Casso:

```
$ MATCH ( n  :Employee  {Name: 'José Pan'} ), ( b  :Employee  {Name: 'Vilma
  Casso'} ), ( c  :Employee  {Name: 'Ariel Casso'} )
 CREATE (b) <- [r1  :manager_of]  - (n)  - [r2  :manager_of] -> (c)
 RETURN r1, r2
```



Cypher - Edges

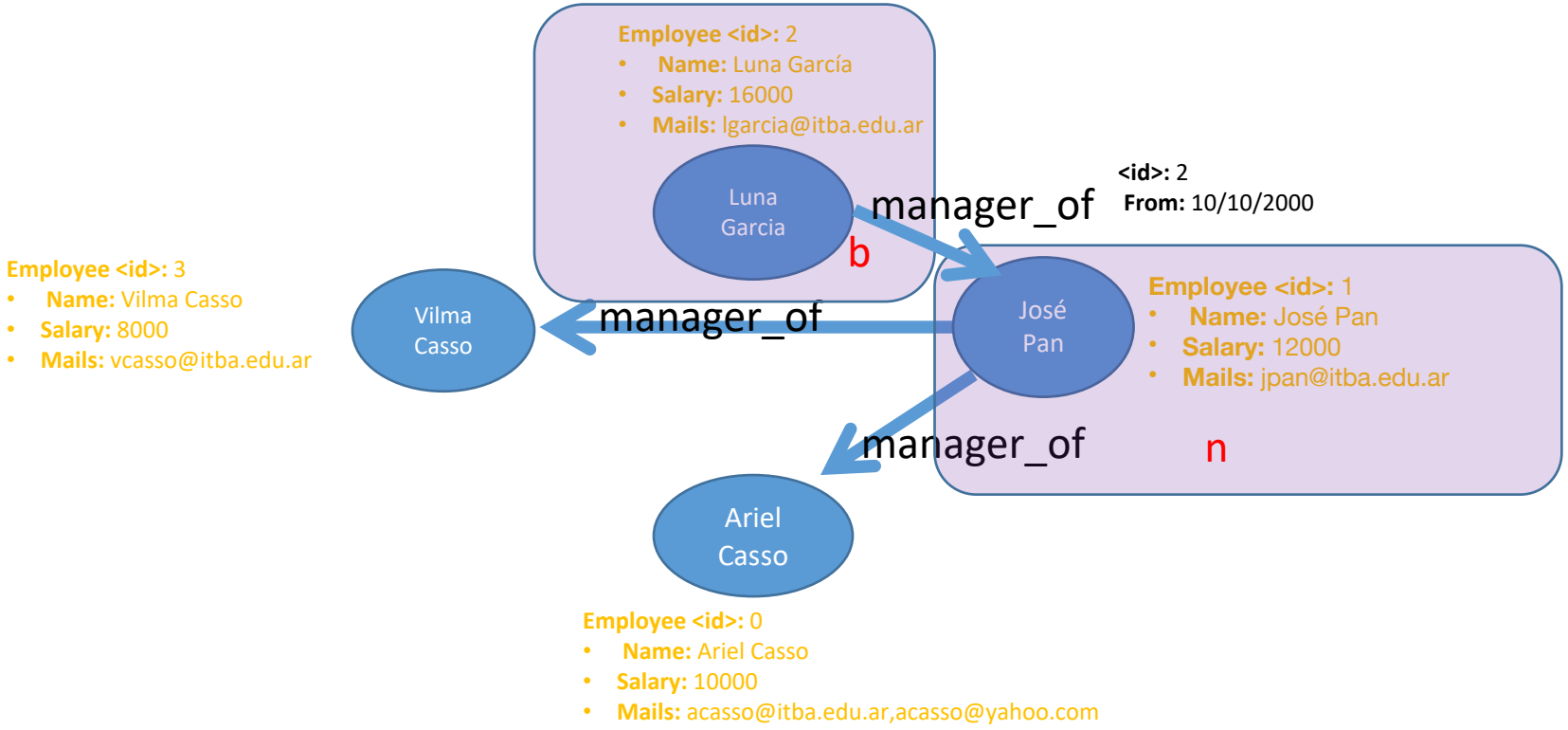
Create another edge of type «manager_of» with property “from”, from L. García to José Pan



Cypher - Edges

Create another edge of type «manager_of» with property “from”, from L. García to José Pan

```
$ MATCH  ( n  :Employee  {Name: 'José Pan'} ),( b  :Employee  {Name: 'Luna García'} )
  CREATE (n) <- [r  :manager_of  {From: '10/10/2000'} ]  - (b)
  RETURN n, r, b
```



Cypher – queries

High-level query language based on pattern matching

Query graphs expressing informational and/or topological conditions

Cypher – queries

MATCH

OPTIONAL MATCH

WHERE

«Match» expresses a pattern that DBMS will try to match. OPTIONAL MATCH Works like an «outer join», in SQL, i.e., if dores not find a match, puts nulls.
The WHERE clause is part of the «MATCH or OPTIONAL MATCH». No order can be assumed for the evaluation of the conditions in the WHERE clause, this is decided by the DBMS.

RETURN

ORDER BY

LIMIT

SKIP

LIMIT returns only part of the result. SKIP skips the first results. Unless ORDER BY used, no assumption can be done for the discarded results.

The evaluation produces subgraphs, and any portion of the match could be returned.
«RETURN DISTINCT» eliminates duplicates.

Cypher – queries

In addition to the above:

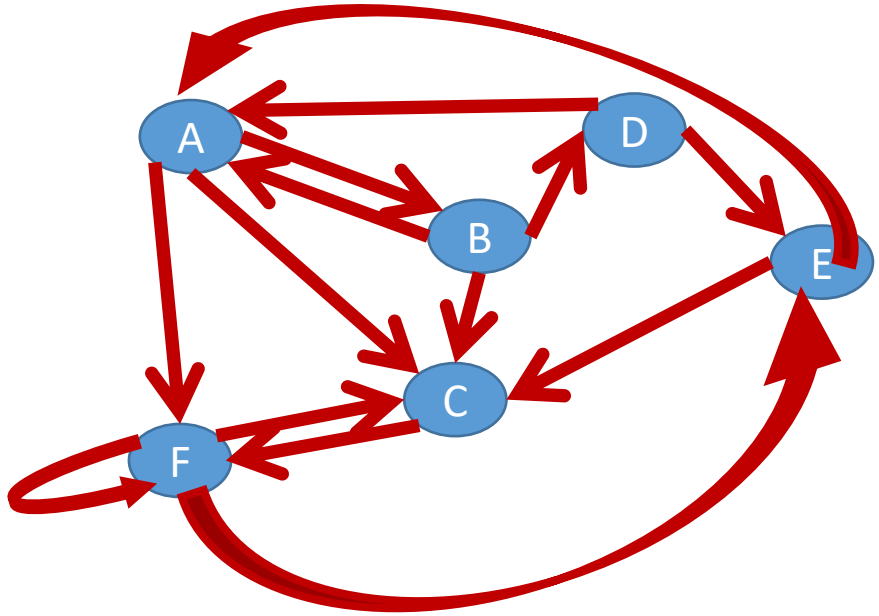
1) If we don’t need to refer to a node, we can use “()”, with no variable.
2) If we don’t need to refer to an edge, we can omit it, e.g.: (a) --> (b) indicates an edge between a and b.
3) If we don’t need to consider the direction of the edge, just use “- -” (without the arrow end)
4) If a mattern matches more tan one label, write the OR condition as, e.g., [:manager_of | :Student]
5) To express a path of any length, use [*]. For a fixed length, e.g., 3, use [*3]
6) To indicate boundaries to the length of a pathm use [*2..4] . To limit only one end, use : [*2 ..]

Cypher – Example

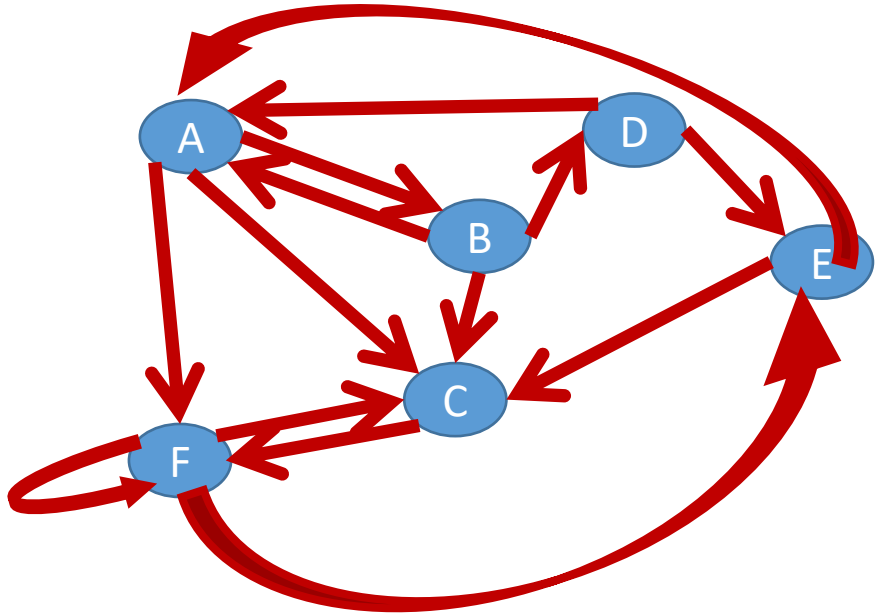
The query:

```
$ MATCH (p)-[]->(s)-[]->(x)
   RETURN Count(p), s.URL, Count(x)
```

Returns the following. Why???



Cypher – Example



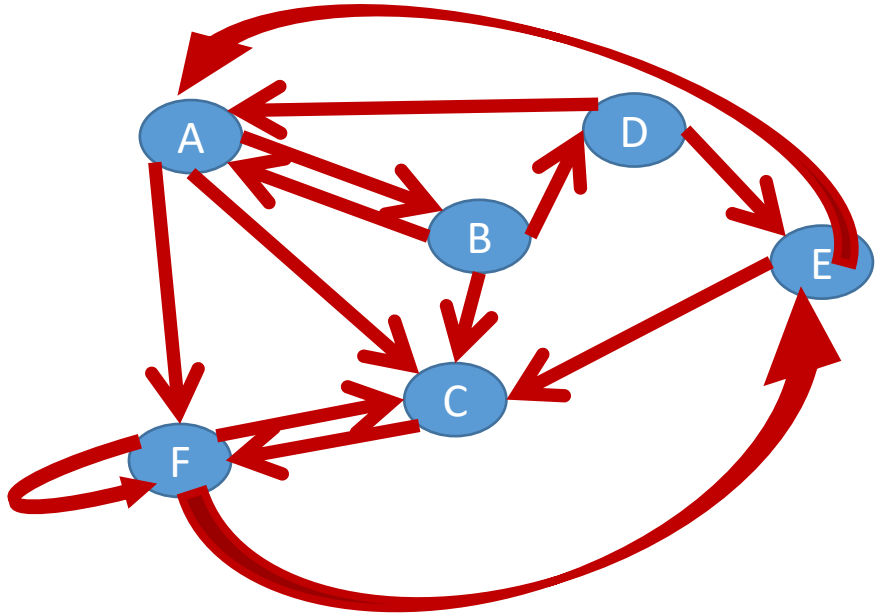
The query:

```
$ MATCH (p)-[]->(s)-[]->(x)
   RETURN Count(p), s.URL, Count(x)
```

Returns the following. Why???

«Count(p)»	«s»	«Count(x)»
9	A	9
3	B	3
4	C	4
2	D	2
4	E	4
8	F	8

Cypher – Example




```
$ MATCH (p)-[]->(s)-[]->(x)
   RETURN Count(p), s, Count(x)
```

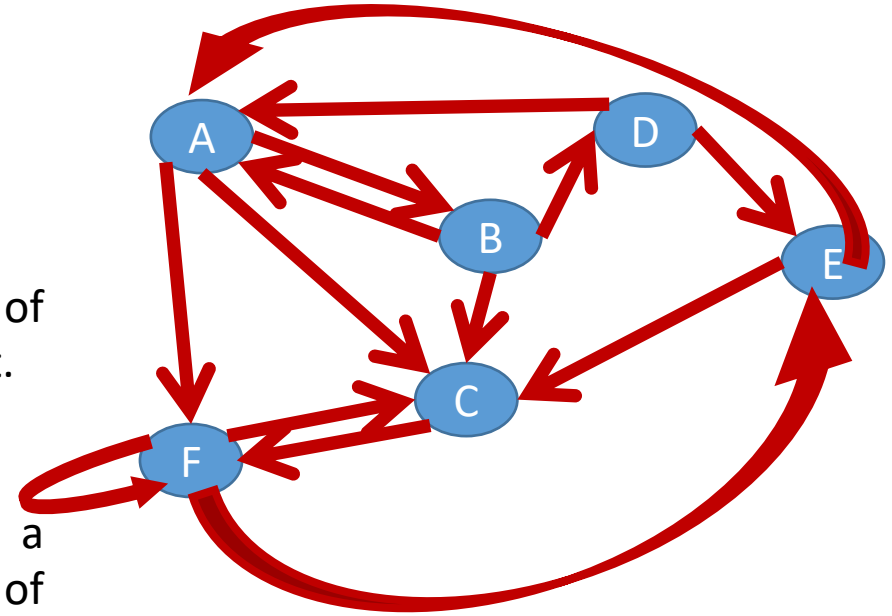
The first clause computes paths where a node (s) has an incoming and an outgoing edge.
E.g., for «c», these paths are:

(a) -- (c) –> (f)
(f) -- (c) --> (f)
(b) -- (c) --> (f)
(e) -- (c) --> (f)

The second clause groups these 4 paths and return how many nodes are connected on each side, to node (c)., and we obtain:

4 c 4

Cypher – Example



A page X gets a score computed as the sum of all votes given by the pages that references it.

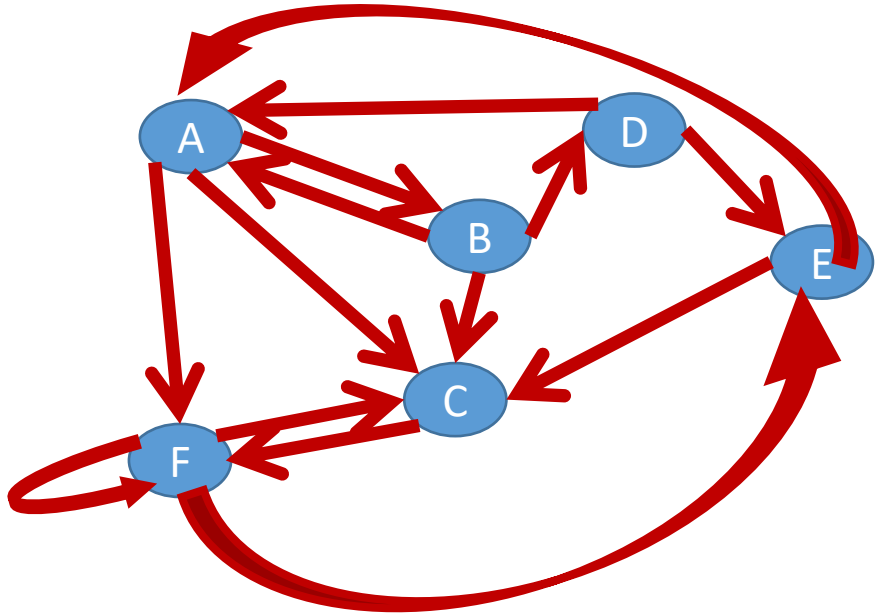
If a page Z references a page X, Z gives X a normalized vote computed as the inverse of the number of pages referenced by Z. To prevent votes of self-referencing pages, if Z references X and X references Z, Z gives 0 votes to X.

Compute the page rank for each web page.

Cypher – Example

Possible solution:

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  RETURN  x, SUM(vote) AS Rank
  ORDER BY x.URL
```



«p»	«vote»
A	0.333
B	0.333
C	1
D	0.5
E	0.5
F	0.333

The first MATCH - WITH pair computes, for each node, the inverse of the number of outgoing edges, and passes this number on to the next clause.

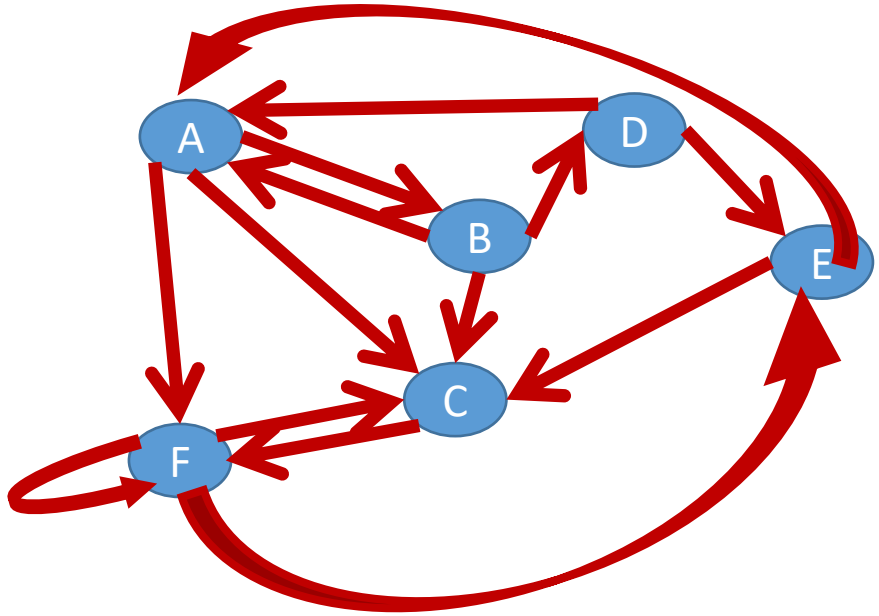
Cypher – Example

Possible solution:

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  RETURN  x, SUM(vote) AS Rank
  ORDER BY x.URL
```

«p»	«vote»
A	0.333
B	0.333
C	1
D	0.5
E	0.5
F	0.333

«p»	«x»
A	C
A	F
B	C
B	D
D	A
D	E
E	A
E	C
F	E

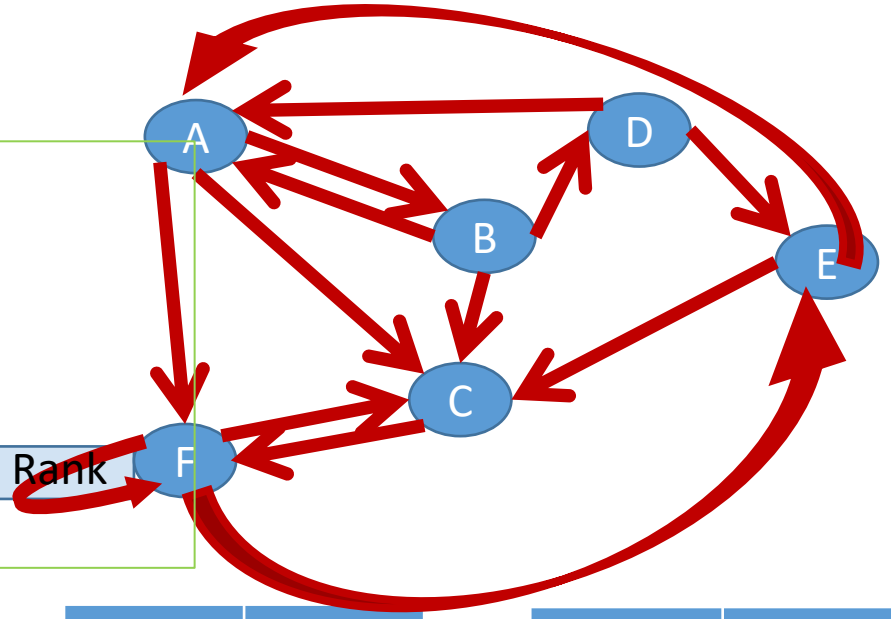


Now, for each of these 6 “p” nodes, look for the paths of length 1 where no reciprocity exists (e.g., delete A ->B and B -> A)

Cypher – Example

Possible solution

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  RETURN  x.URL, COLLECT (p.URL), SUM(vote) AS Rank
  ORDER BY x.URL
```



«p»	«vote»
A	0.333
B	0.333
C	1
D	0.5
E	0.5
F	0.333

«p»	«x»
A	C
A	F
B	C
B	D
D	A
D	E
E	A
E	C
F	E

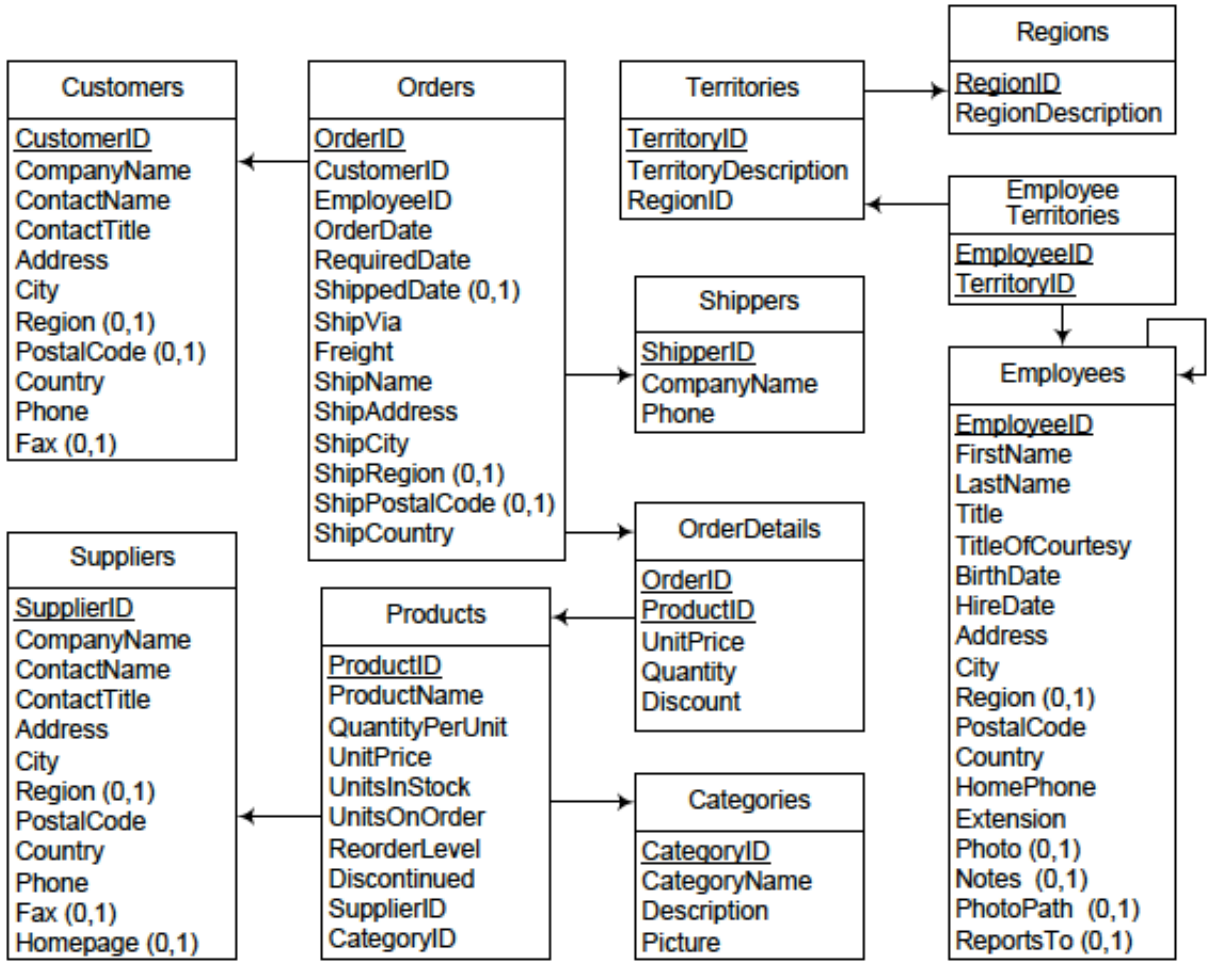
«x»	«p» grouped
A	D, E
C	A, B, E
D	B
E	D, F
F	A

«x»	«Rank»
A	½ + ½
C	1/3 + 1/3 + 1/2
D	1/3
E	½ + 1/3
F	1/3

Finally, groups results by the second component and sorts.

Neo4j - Practice

Neo4j Practice – The Northwind Database



Loading the graph

1. Using the LOAD CVS statement

```
LOAD CSV WITH HEADERS FROM "file:///territories.csv" AS row
CREATE (:Territory {territoryID: row.territoryid,
name: row.territorydescription});

============
LOAD CSV WITH HEADERS FROM "file:///employees.csv" AS row
CREATE (:Employee{employeeID: row.employeeid,
lastName: row.lastname,firstName: row.firstname, city:row.city,region:row.region,country:row.country});

==============

LOAD CSV WITH HEADERS FROM "file:///employeeterritories.csv" AS row
MATCH (t:Territory {territoryID: row.territoryid})
MATCH (e:Employee {employeeID: row.employeeid})
MERGE (e)-[:AssignedTo]->(t)
```

Loading the graph

2. Connecting to a Postgres DB

- Copy database driver to the “Plugins” folder
- APOC library must also be copied in the “Plugins” folder
- **Check the right APOC version for your Neo4j version!!!**

```
WITH "jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres" as url

%% NorthwindOLTP:  your database in the PostgreSQL instance
%% url: to be used in the procedure call
CALL apoc.load.jdbc(url,"select * from categories") YIELD row
% the query string can also mention just a table
% row: a “row variable” just as before
RETURN row.description,row.categoryname
```

This lists the table “categories” in Neo4j.
We can use this also for loading data into Neo4j.

Loading the graph

```
WITH "jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres" as url

CALL apoc.load.jdbc(url,"select * from products") YIELD row

CREATE (:Product {productID: row.productid,productName:row.productname, supplier: row.supplierid, category:row.categoryid,
qtyperunit:row.quantityperunit})

==================================

WITH "jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres" as url

CALL apoc.load.jdbc(url,"select * from suppliers") YIELD row

CREATE (:Supplier {supplierID: row.supplierid, supplierName:row.companyname, city:row.city, region:row.region, country:row.country})
```

Loading the graph

3. With Cypher

```
MATCH(s:Supplier)

MATCH(p:Product) where p.supplier=s.supplierID

MERGE (s)-[:Supplies]->(p)
```

Loading the graph

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/city.csv" AS row
CREATE (:City {cityID:row.citykey,cityName: row.cityname});

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/territories.csv" AS row
CREATE (:Territory {territoryID: row.territoryID, name: row.territoryDescription});

...

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/employee-territories.csv" AS row
MATCH (territory:Territory{territoryID: row.territoryID})
MATCH (employee:Employee {employeeID: row.employeeID})
MERGE (employee)-[:AssignedTo]->(territory);
```

Loading the graph

-- Create a view to put together orders and order details

CREATE VIEW order1 AS (SELECT o.orderid AS orderID,o.orderdate AS
orderDate,o.shippeddate AS shippedDate,o.shipname AS shipName, sum(quantity)
AS totqty,sum(unitprice*quantity) AS totAmount FROM orders o,orderdetails o1
WHERE o.orderid=o1.orderid
group by o.orderid,o.orderdate,o.shippeddate,o.shipname
order by orderid asc)
SELECT * INTO ordershg FROM order1

COPY ordershg to 'C:\tmp\ordershg.csv' delimiter ',' CSV header USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:/NWdata/ordershg.csv" AS row
CREATE (:**Order** {orderID: row.orderid, orderDate: row.orderdate,
ShippedDate: row.shippeddate,shipName:row.shipname,totalQty:row.totqty, totalAmount:row.totamount});

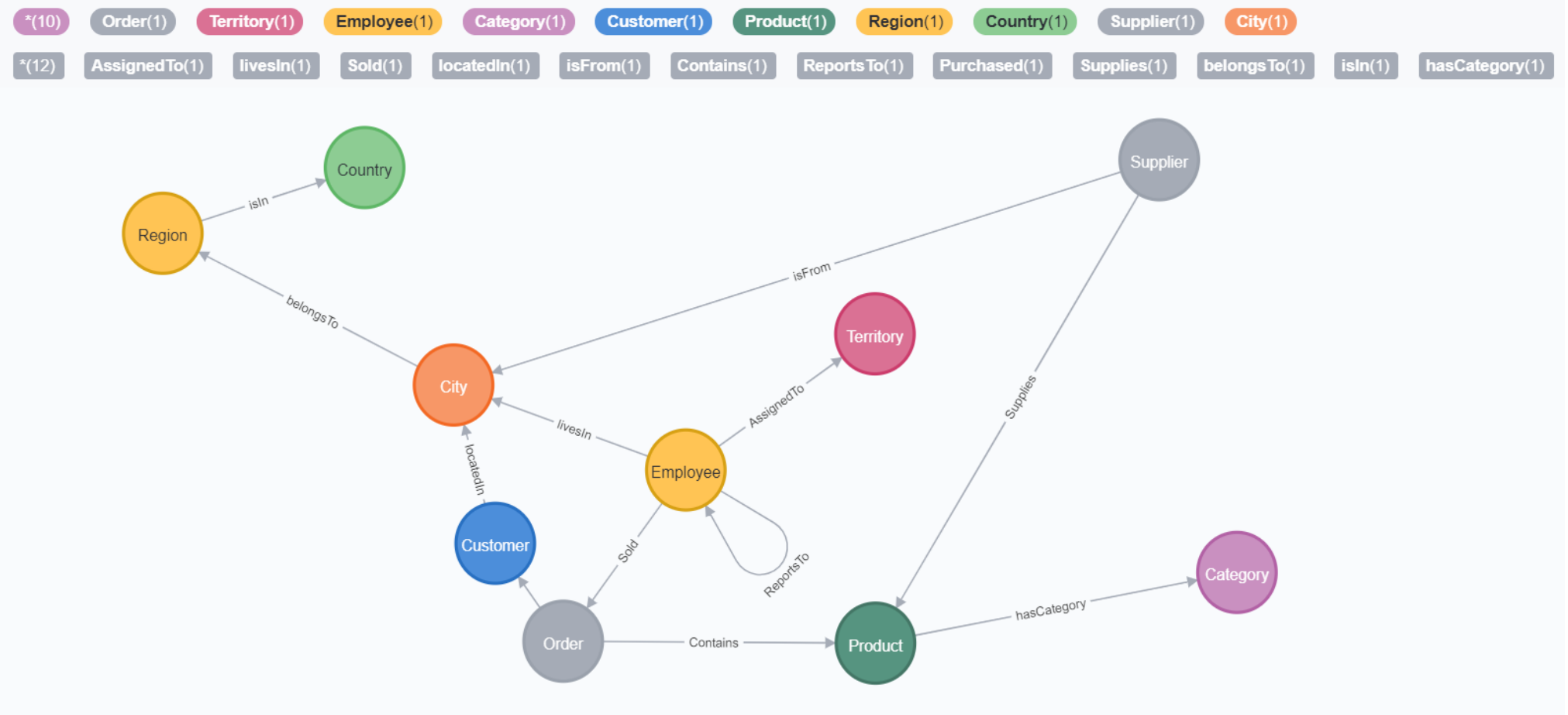
You can also connect to a PostgreSQL database

CALL apoc.load.jdbc('jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres','select * from ordershg') YIELD row
CREATE (:Order {orderID: row.orderid, orderDate: row.orderdate, ShippedDate:
row.shippeddate,shipName:row.shipname,totalQty:row.totqty, totalAmount:row.totamount});

Loading the graph

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/orders.csv" AS row
MATCH (order:Order {orderID: row.orderID})
MATCH (employee:Employee {employeeID: row.employeeID})
MERGE (employee)-[:Sold]->(order);

LOAD CSV WITH HEADERS FROM "file:/NWdata/order-details.csv" AS row
MATCH (order:Order {orderID: row.orderID})
MATCH (product:Product {productID: row.productID})
MERGE (order)-[:Contains{unitPrice:row.unitPrice,quantity:row.quantity,  discount:row.discount}]->(product);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/products.csv" AS row
MATCH (product:Product {productID: row.productID})
MATCH (supplier:Supplier {supplierID: row.supplierID})
MERGE (supplier)-[:Supplies]->(product);

CALL apoc.load.jdbc('jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres','select * from employees') YIELD row
MATCH (employee:Employee {employeeID: row.employeeid})
MATCH (employee1:Employee {employeeID: row.reportsto})
MERGE (employee)-[:ReportsTo]->(employee1);
```

Schema: Northwindhg database



Problem 1. Northwindhg database

- Query 1. List products and their unit price.

```
MATCH (p:Product)
RETURN p.productName, p.unitPrice
ORDER BY p.unitPrice DESC
```

- Query 2. List information about products 'Chocolade' & 'Pavlova'.

```
MATCH (p:Product)
WHERE p.productName IN ['Chocolade','Pavlova']
RETURN p
```

- Query 3. List information about products with names starting with a "C", whose unit price is greater than 50.

```
MATCH (p:Product)
WHERE p.productName STARTS WITH "C" AND tofloat(p.unitPrice) > 50
RETURN p.productName, p.unitPrice;
```

%% Try without ”tofloat”

- Query 4. Same as 3, but considering the sales price, not the product’s price.

```
MATCH (p:Product) <- [c:Contains] - (o:Order)
WHERE p.productName STARTS WITH "C" AND tofloat(c.unitPrice) > 50
RETURN distinct p.productName, p.unitPrice, c.unitPrice;
```

Problem 1. Northwindhg database

- Query 5. Total purchased by customer and product.

```
MATCH (c:Customer)
OPTIONAL MATCH (p:Product)<-[pu:Contains]-(:Order)-[:Purchased]->(c)
RETURN c.customerName, p.productName,tofloat(sum(pu.unitPrice) * pu.quantity) as volume
ORDER  BY  volume desc
```

- Query 6. Top 10 employees, considering the number of orders sold.

```
MATCH (:Order)<-[:Sold]-(e:Employee)
RETURN e.firstName,e.lastName, count(*) AS Ordenes
ORDER BY Ordenes DESC LIMIT 10
```

- Query 7. For each employee, list the assigned territories.

```
MATCH (t:Territory)<-[:AssignedTo]-(e:Employee)
RETURN e.lastName, COLLECT(t.name);
```

- Query 8. For each city, list the companies settled in that city.

```
MATCH (c:City)<-[:locatedIn]-(c1:Customer)
RETURN c.cityname, COLLECT(c1.customerName);
```

Problem 1. Northwindhg database

- Query 10. How many persons an employee reports to, either directly or transitively?

```
MATCH (report:Employee)
OPTIONAL MATCH (e)<-[rel:ReportsTo*]-(report)
RETURN report.lastName AS e1, COUNT(rel) AS reports
```

- Query 11. To whom do persons called “Robert” report to?

```
MATCH (e:Employee)<-[:ReportsTo*]-(sub:Employee)
WHERE sub.firstName = 'Robert'
RETURN e.firstName,e.lastName,sub.lastName
```

- Query 12. Who does not report to anybody?

```
MATCH (e:Employee)
WHERE NOT (e)-[:ReportsTo]->()
RETURN e.firstName as TopBossFirst, e.lastName as TopBossLast
```

- Query 13. Suppliers, number of categories they supply, and a list of such categories

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
WITH s.supplierName as Supplier, collect(distinct c.categoryName) as Categories
WITH Supplier, Categories, size(Categories) AS Cantidad ORDER BY Cantidad DESC
RETURN Supplier, Cantidad, Categories;
```

Problem 1. Northwindhg database

- Query 14. Suppliers who supply beverages

```
MATCH (c:Category {categoryName:"Beverages"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.supplierName as ProduceSuppliers;
```

- Query 15. Customer who purchases the largest amount of beverages

```
MATCH (cust:Customer)<-[:Purchased]-(:Order)-[o:Contains]->(p:Product), (p)-[:hasCategory]->
(c:Category{categoryName:"Beverages"})
RETURN cust.customerName as CustomerName, SUM(o.quantity)
LIMIT 1
```

- Query 16. List the 5 most popular products (considering the number of orders)

```
MATCH (c:Customer)<-[:Purchased]-(o:Order)-[o1:Contains]->(p:Product)
RETURN c.customerName, p.productName, count(o1) as orders
ORDER BY orders desc  LIMIT 5
```

Problem 1. Northwindhg database

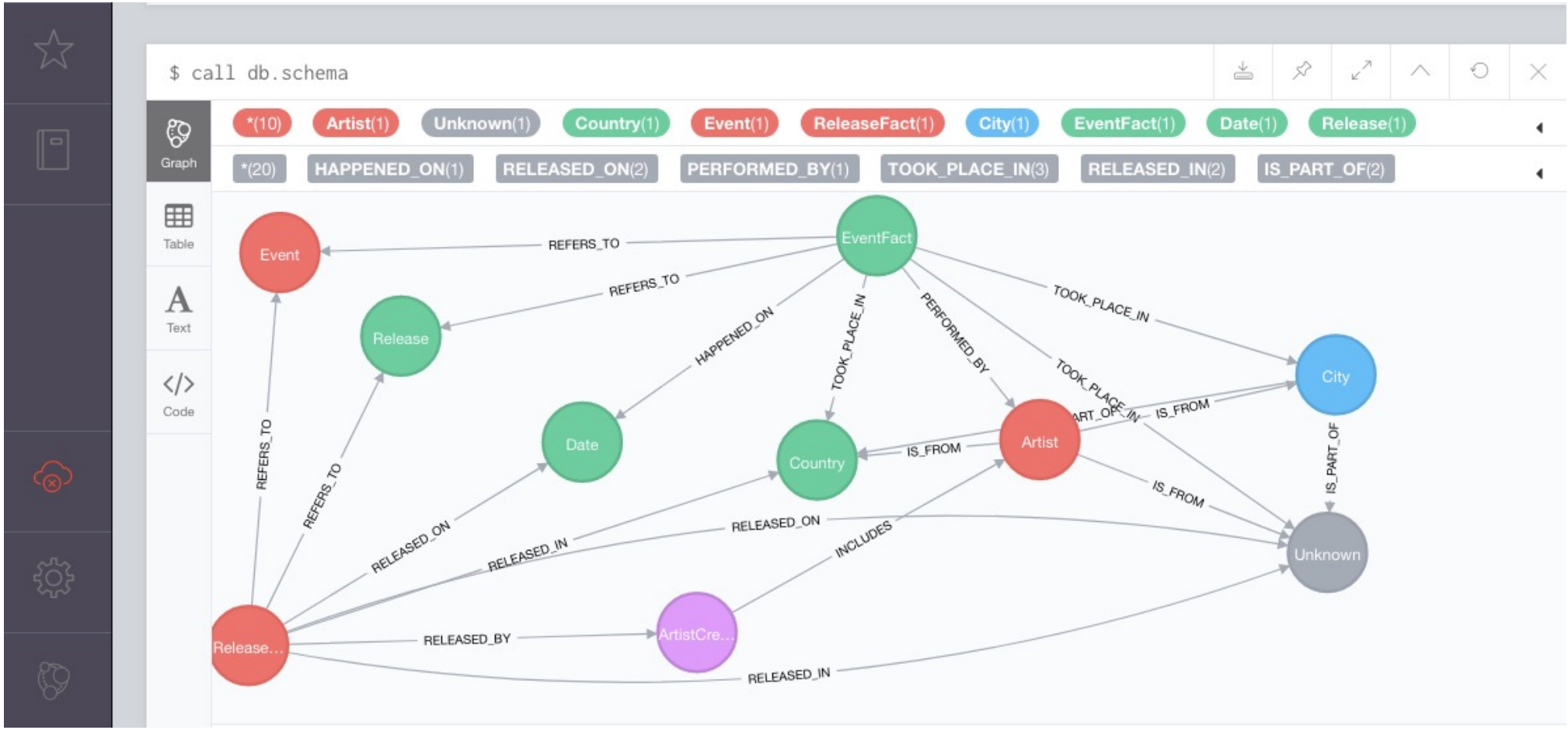
- Query 17. Products ordered by customers from the same country than their suppliers

```
MATCH (c:Customer) -[r:locatedIn]->(cy:City)-[:belongsTo]->(:Region)-[:isIn]->(co:Country)
WITH co, c MATCH (s:Supplier) WHERE co.countryname = s.country
WITH s, co, c MATCH(s)-[su:Supplies]-(p:Product)<-[:Contains]-(o:Order)-[:Purchased]->(c)
RETURN c.customerName,s.supplierName,co.countryname,p.productName
```

OR

```
MATCH (c:Customer) -[r:locatedIn]->(cy:City)-[:belongsTo]->(:Region)-[:isIn]->(co:Country)
WITH co, c
MATCH(s:Supplier)-[su:Supplies]-(p:Product)<-[:Contains]-(o:Order)-[:Purchased]->(c) WHERE co.countryname = s.country
RETURN c.customerName,s.supplierName,co.countryname,p.productName
```

Problem 2 – MusicBrainz database



Problem 2 – MusicBrainz database

Query 1. Compute the number of releases per artist.

```
MATCH (r:ReleaseFact)-[]->(a:ArtistCredit)-[]->(a1:Artist)
RETURN a1.name, a1.gender
```

Query 2. Compute the number of releases per artist and per year.

Query 3. Compute the number of events per artist.

```
MATCH (e:EventFact)-[r:PERFORMED_BY]->(a:Artist)
RETURN a.name, count(*)
```

Query 4. Compute the number of times the artist performed in each event.

Query 5. For each (event, artist, year) triple, compute the number of times the artist performed in an event on an year.

Problem 2 – MusicBrainz database

Query 6. Same as Query 5, for artists in the United Kingdom and events happened after year 2006.

Query 7. Compute the number of releases, per language, in the UK.

Query 8. Compute, for each pair of artists, the number of times they have performed together at least twice in an event.

Query 9. Compute the triples of artists, and the number of times they have performed together in an event, if this number is at least 3.

```
MATCH (a1:Artist)<-[]-(e:EventFact)-[]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1,a2,COLLECT(e) AS events WHERE SIZE(events) > 2
MATCH (a1:Artist)<-[]-(e1:EventFact)-[]->(a2:Artist)
MATCH (a3:Artist)<-[]-(e1) WHERE a2.id < a3.id
WITH a1.name as name1, a2.name as name2,a3.name as name3 ,
COUNT(e1.idEvent) as nbrTimes WHERE nbrTimes > 2
RETURN name1,name2,name3, nbrTimes ORDER BY nbrTimes DESC
```

ITBA
Instituto Tecnológico de Buenos Aires

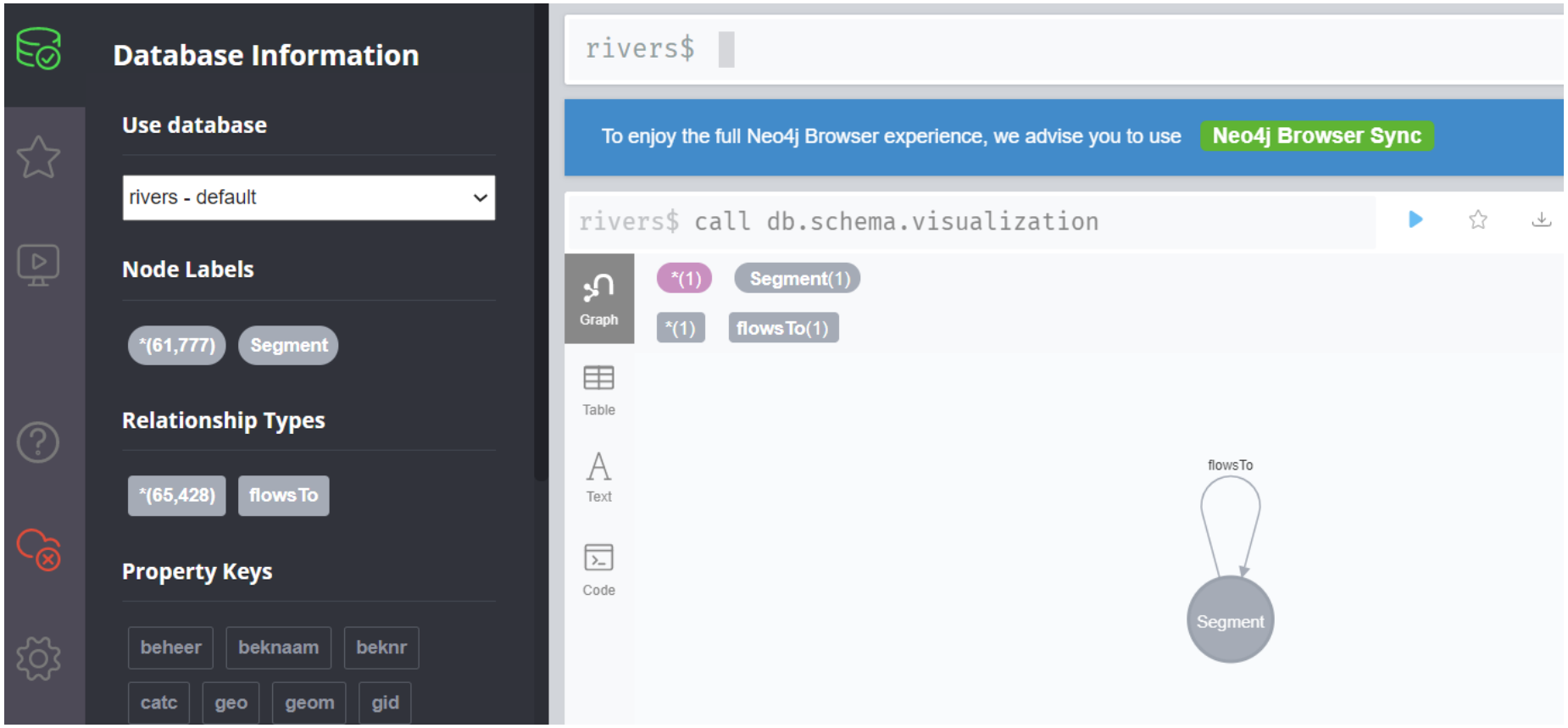
Problem 2 – MusicBrainz database

Query 10. Compute the quadruples of artists, and the number of times they have performed together in an event, if this number is at least 3.

Query 11. Compute the pairs of artists that have performed together in at least two events and that have worked together in at least one release, returning the number of events and releases together.

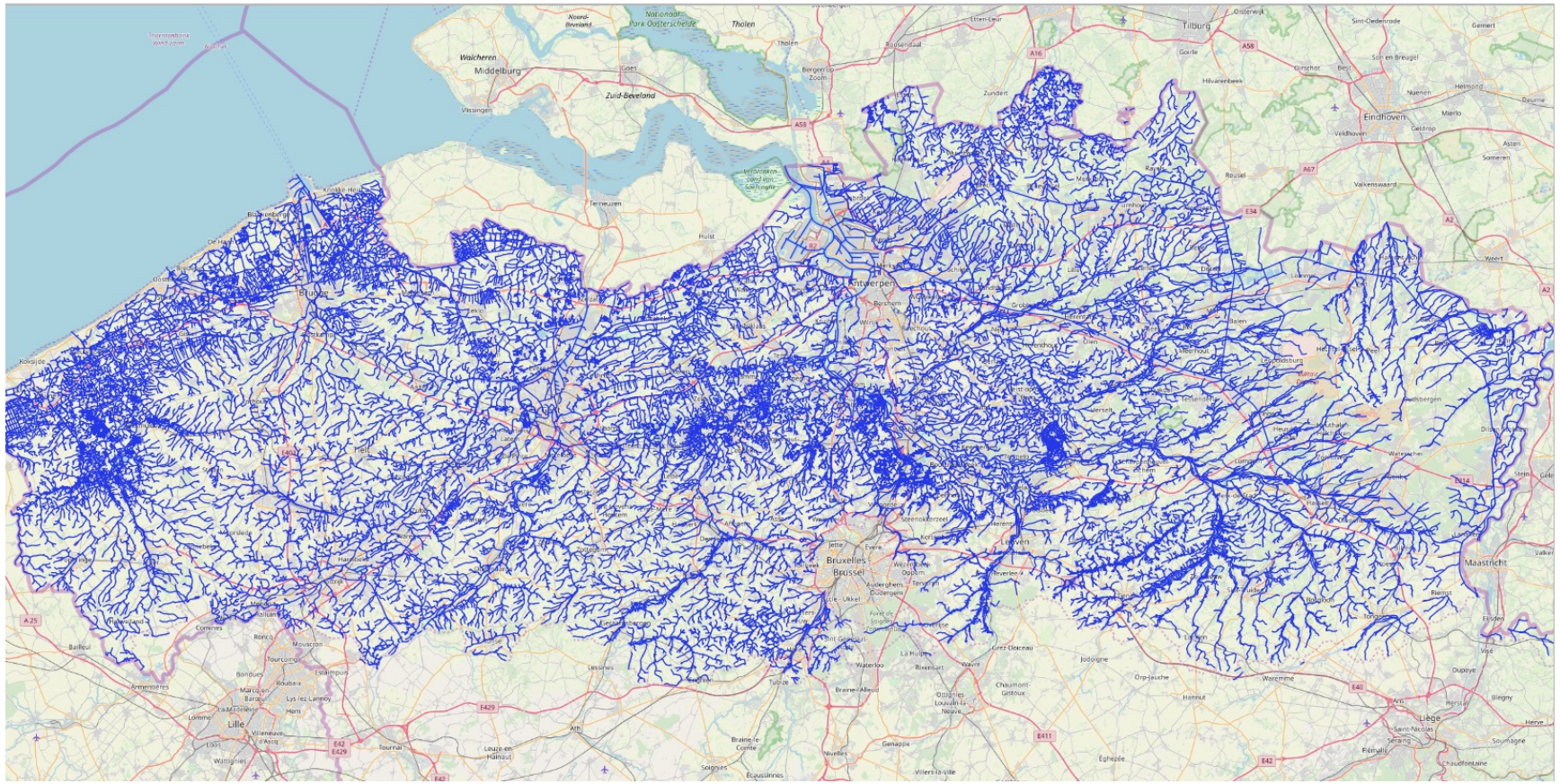
Query 12. Compute the of artists who released a record and performed in at least an event, and the year(s) this happened.

Problem 3 – Rivers



ITBA
Instituto Tecnológico de Buenos Aires

Problem 3 – Rivers



ITBA
Instituto Tecnológico de Buenos Aires

Problem 3 – Rivers

Query 6. Find the number of splits in the downstream path of segment 6020612

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path AS pp
UNWIND NODES(pp) as p
MATCH (p)-[:flowsTo]->(r:Segment)
WITH p, count(DISTINCT r) as co WHERE co > 1
RETURN count(p)
```

ITBA
Instituto Tecnológico de Buenos Aires

Problem 3 – Rivers

Query 10. Find the branches of downstream flow starting at a given position (identifed by a segment's vhas), together with the length and number of segments of each branch.

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path AS pp
UNWIND NODES(pp) as p
MATCH (p)-[:flowsTo]->(r:Segment)
WITH r, count(DISTINCT p) as co WHERE co = 1
WITH collect(r) as pc
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1,endNodes:pc}) YIELD path AS pp
WITH [p in NODES(pp) |p.vhas] AS nodelist,
reduce(longi= tofloat(0),n IN nodes(pp)|longi+n.lengte) AS segLen,
reduce(longi= 1,n IN nodes(pp)| longi + 1) AS nbrSeg
RETURN nodelist[size(nodelist)-1] as id, nbrSeg, segLen
```

UNION

……

ITBA
Instituto Tecnológico de Buenos Aires

Problem 3 – Rivers

Query 10. Find the branches of downstream flow starting at a given position (identifed by a segment's vhas), together with the length and number of segments of each branch.

…..

UNION

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n, {relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path AS pp
UNWIND NODES(pp) as p
MATCH (p)-[:flowsTo]->(r:Segment)
WITH r, count(DISTINCT p) as co WHERE co > 1
WITH collect(r) as pc
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.expandConfig(n,{relationshipFilter:"flowsTo>", minLevel:1,endNodes:pc}) YIELD path AS pp
WITH [p in NODES(pp) |p.vhas] AS nodelist,
reduce(longi = tofloat(0),n IN nodes(pp)|longi+n.lengte) AS segLen,
reduce(longi = 1,n IN nodes(pp)| longi + 1) AS nbrSeg
RETURN nodelist[size(nodelist)-1] as id, nbrSeg, segLen;
```

Problem 3 – Rivers

Query 15. Find all segments reachable from the segment closest to Antwerpen's Groenplaats

```
CALL apoc.spatial.geocodeOnce('Groenplaats Antwerpen Flanders Belgium') YIELD location as ini
MATCH (n:Segment)
WITH n, ini,distance(point({longitude:n.source_long, latitude:n.source_lat}),
point({longitude:ini.longitude, latitude:ini.latitude}) ) as d
WITH n, d order by d asc limit 1
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1})  YIELD path as pp
UNWIND NODES(pp) as p
 RETURN p.vhas;
```