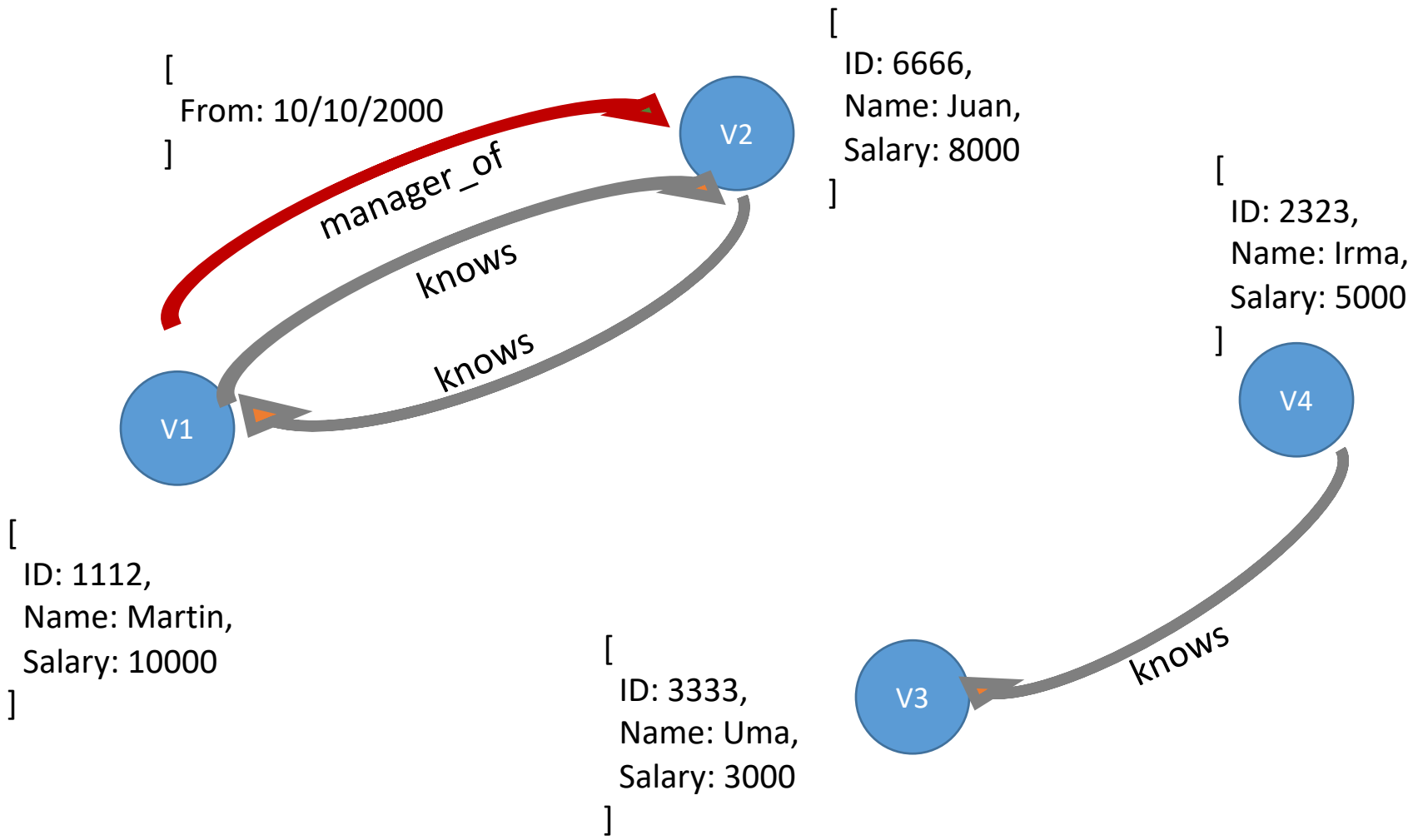
Introduction to Graph Databases

Neo4j

Alejandro Vaisman
avaisman@itba.edu.ar

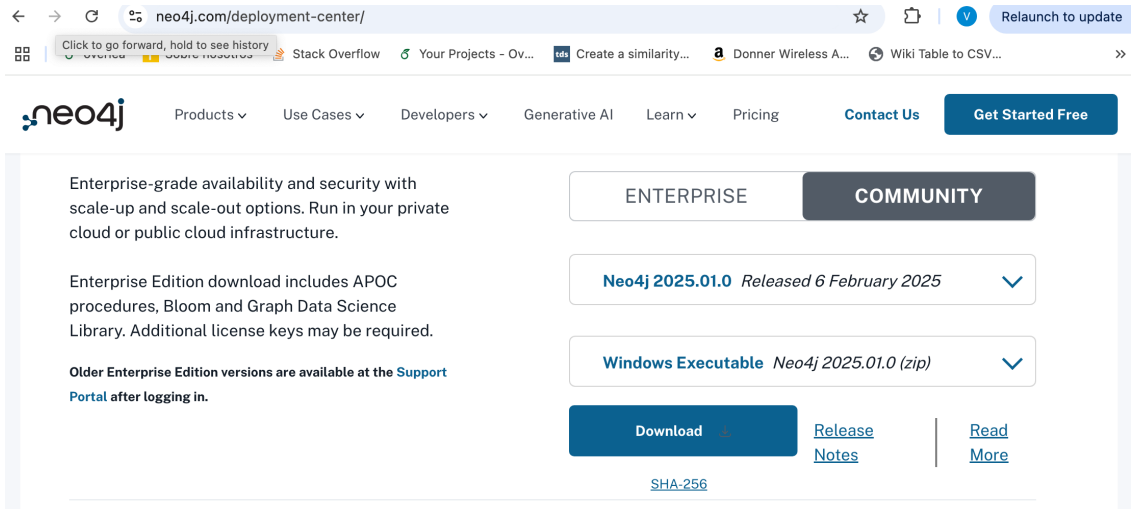
Querying Neo4j Databases with Cypher

Property graphs revisited

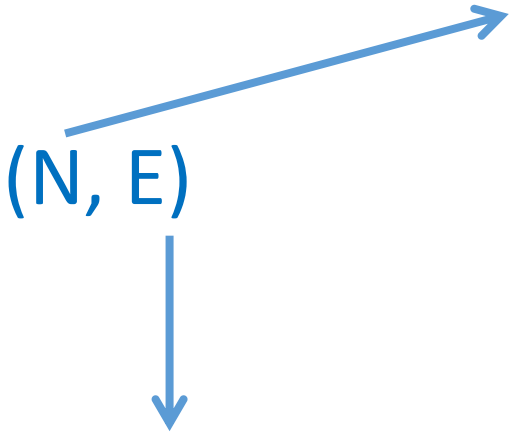


GDBMS: Neo4j www.neo4j.com

- Open Source
- Versions for Linux, Win, Mac. Implemented in Java
- High-level query language: Cypher



A Neo4j graph



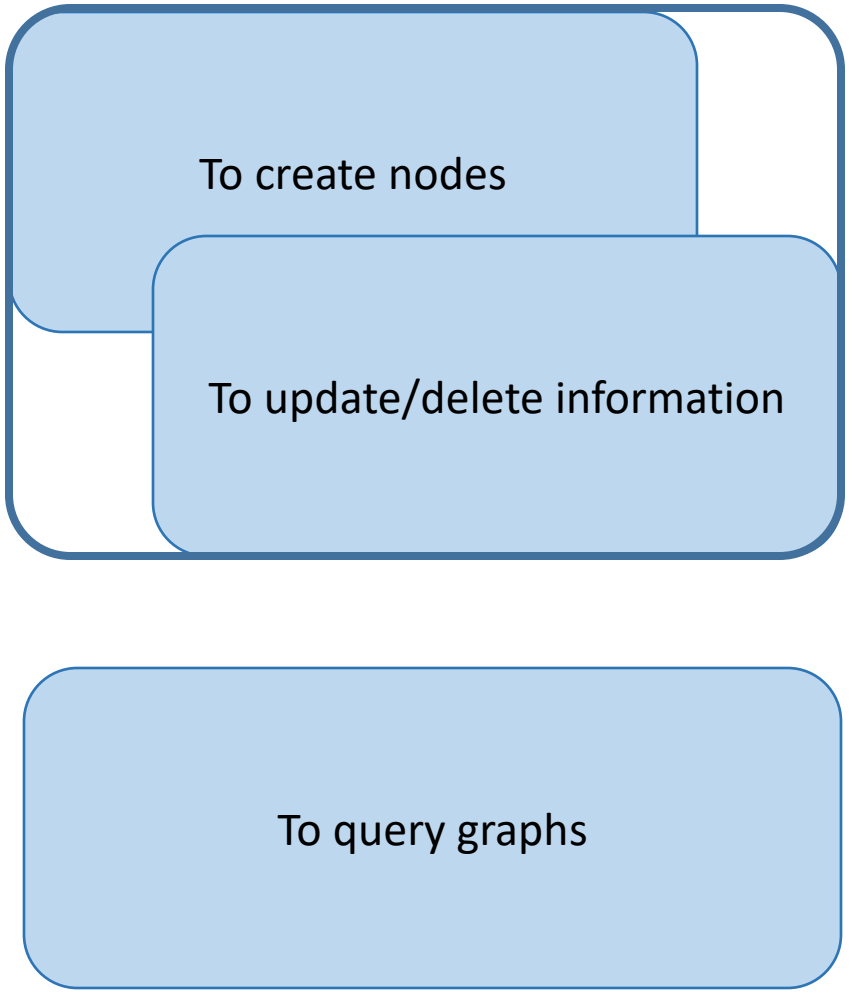
A set of nodes

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

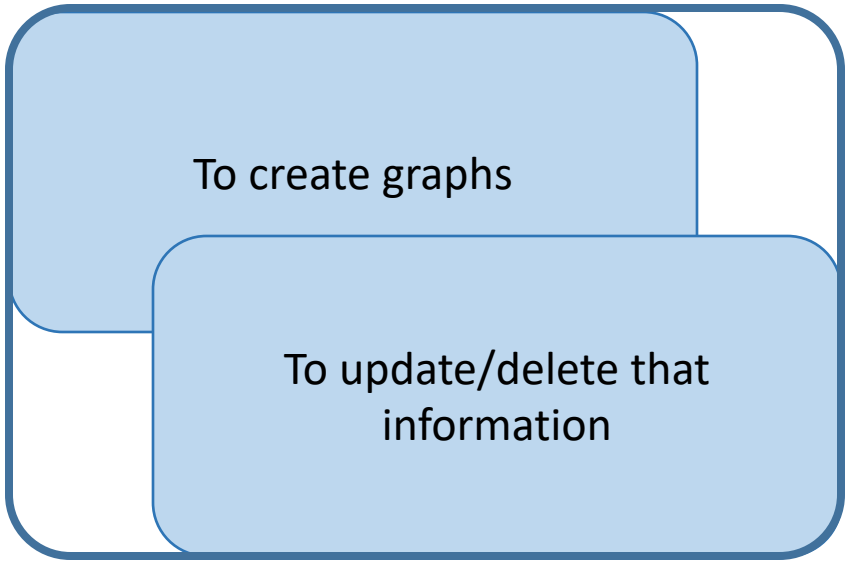
Directed edges (probably with cycles)

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

Cypher


To create nodes
To update/delete information
To query graphs

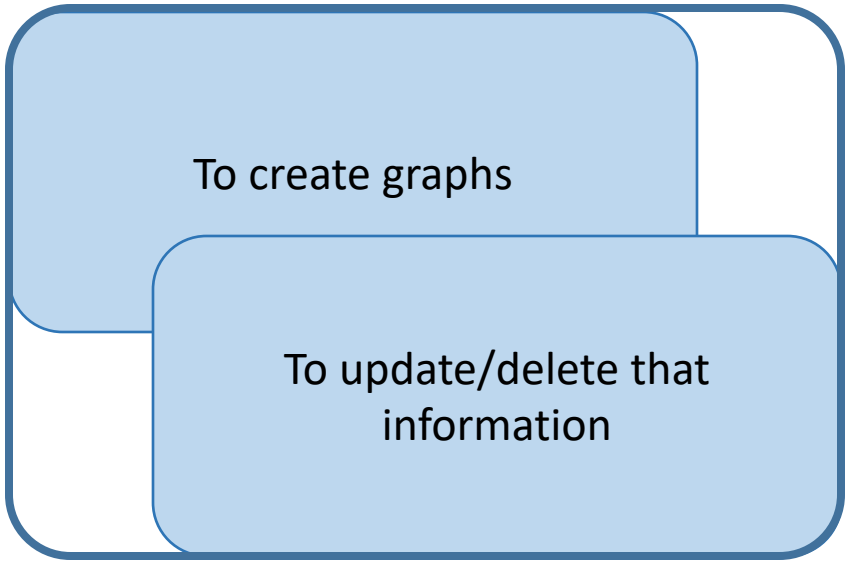
Cypher



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.

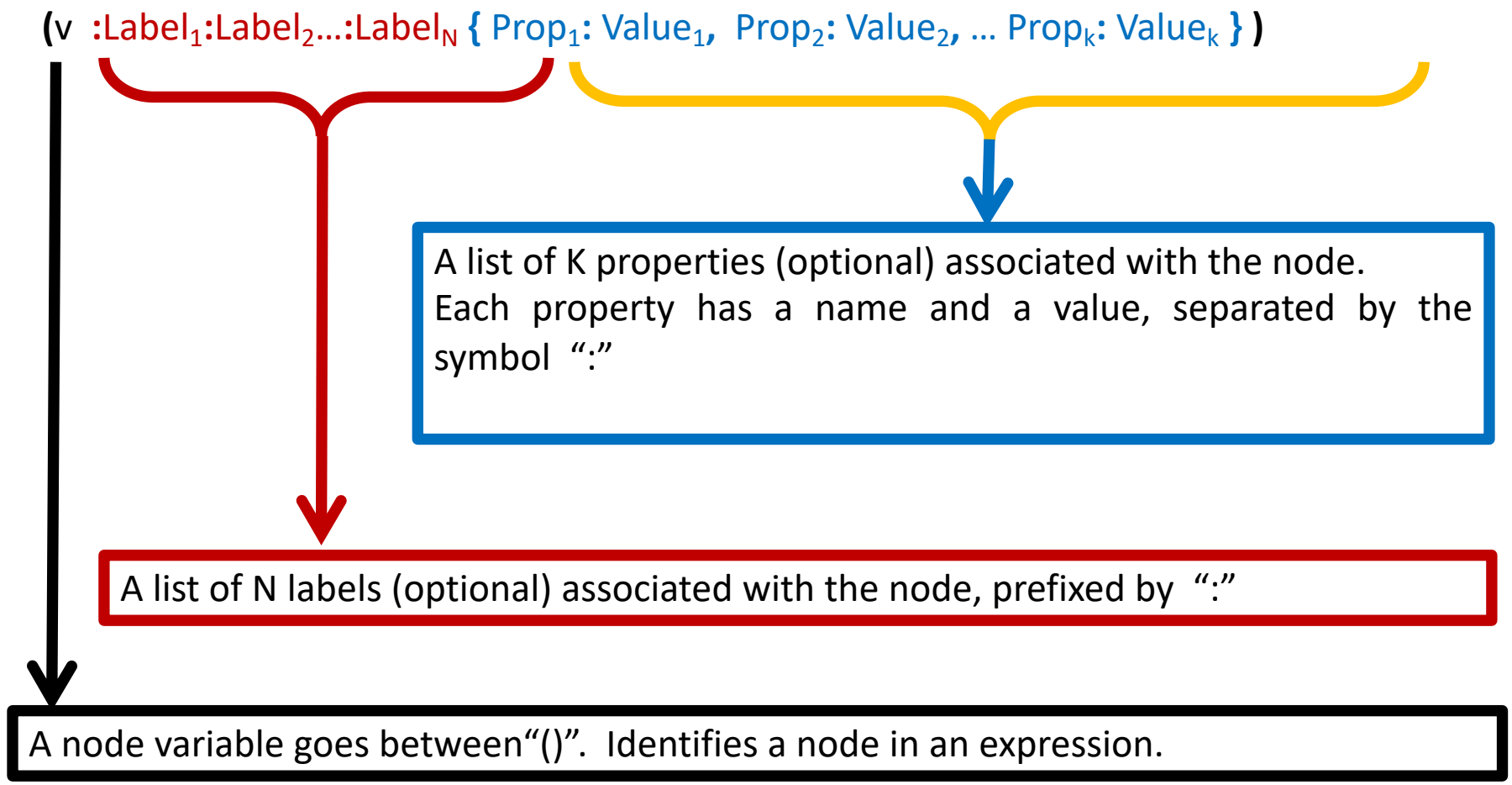
Cypher



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.

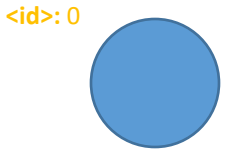
Cypher - nodes



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:

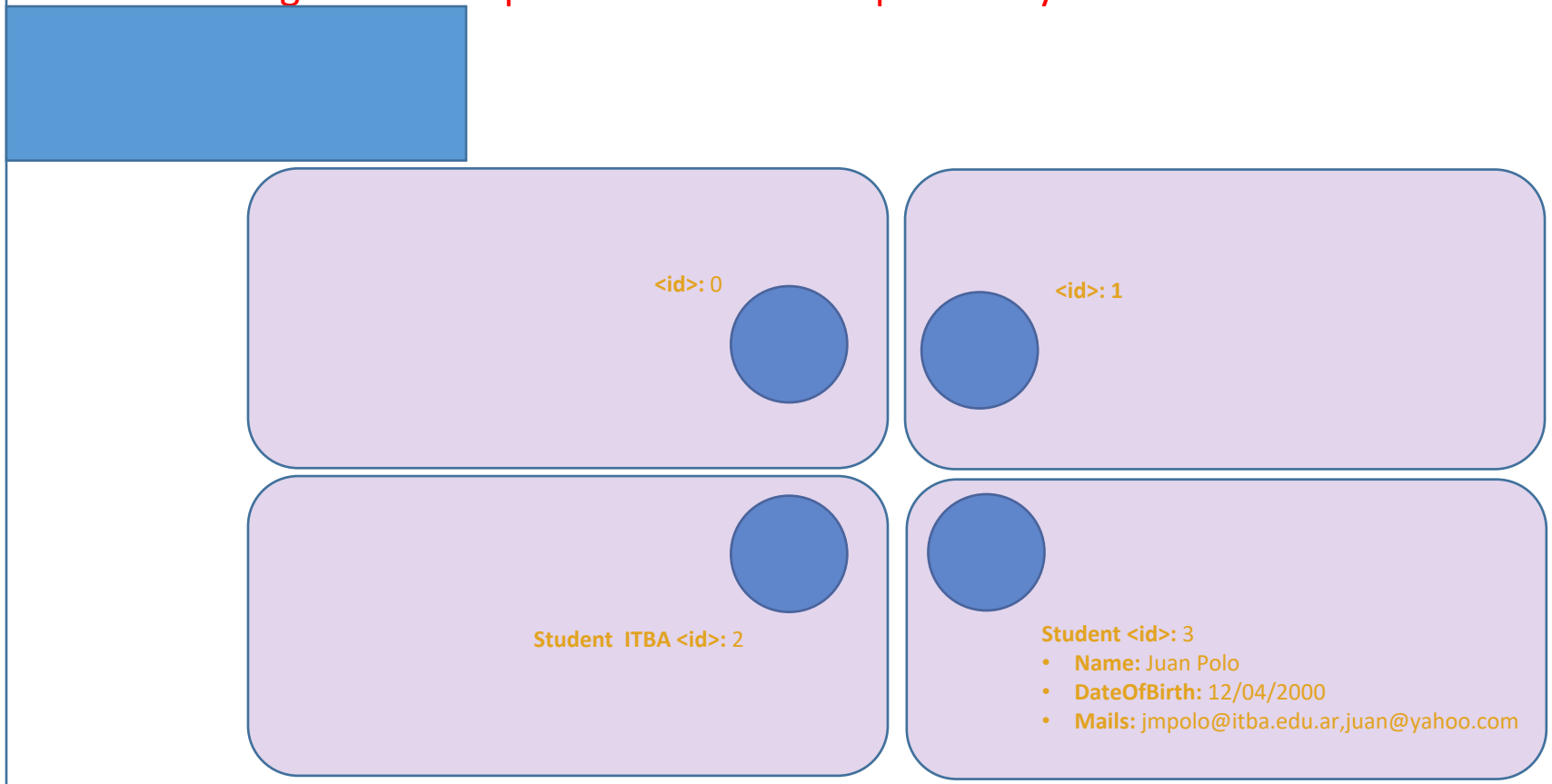
```
$ CREATE (n  :Student {Name: 'Juan Polo',
                       DateOfBirth: '12/04/2000',
                       Mails: ['jmpolo@itba.edu.ar', 'juan@yahoo.com']   })
 RETURN n;
```

Student <id>: 3

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

Cypher - nodes

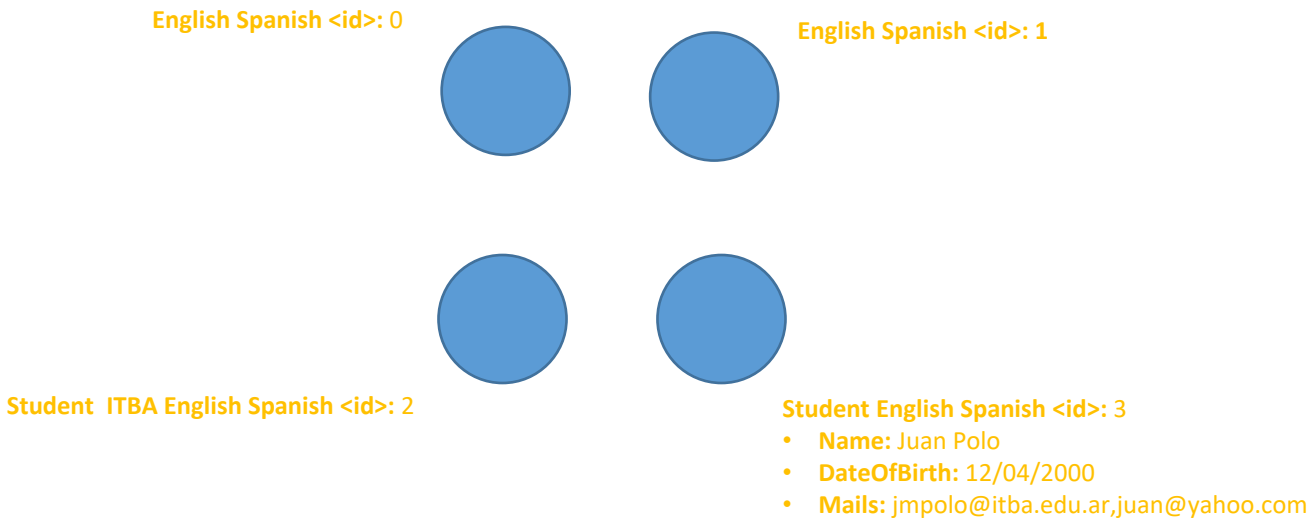
Add labels “English” and “Spanish” to all nodes previously created.



Cypher - nodes

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

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

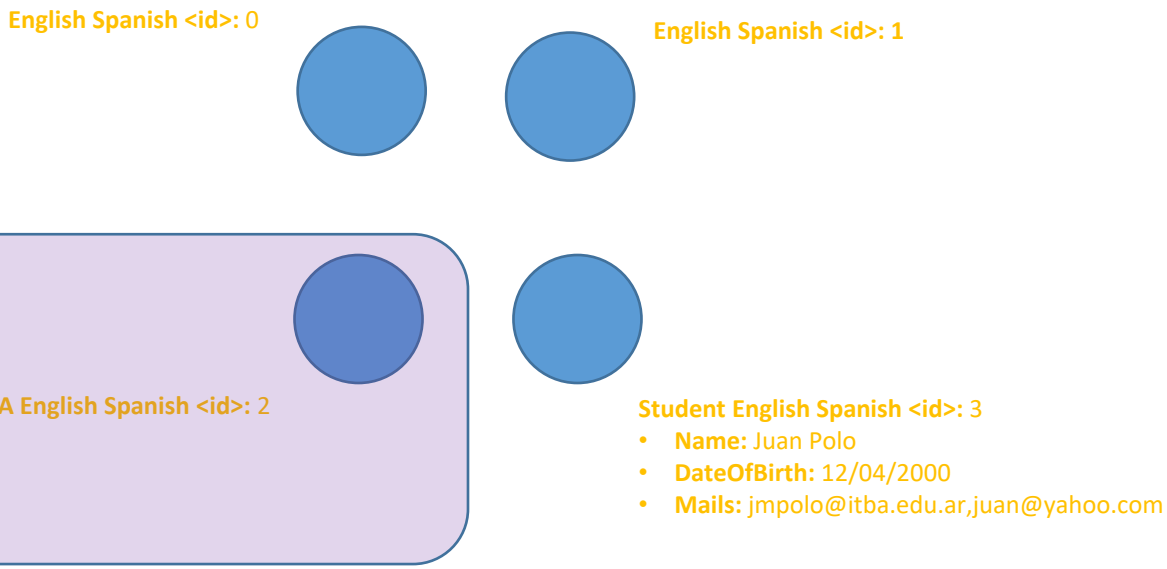


Cypher - nodes

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

$

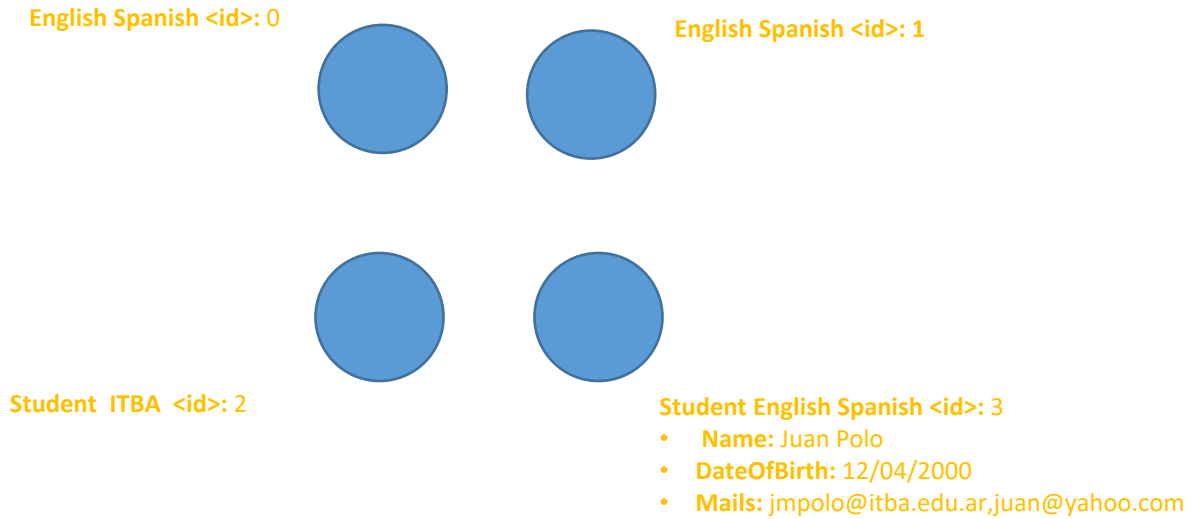
MATCH (n :ITBA)



Cypher - nodes

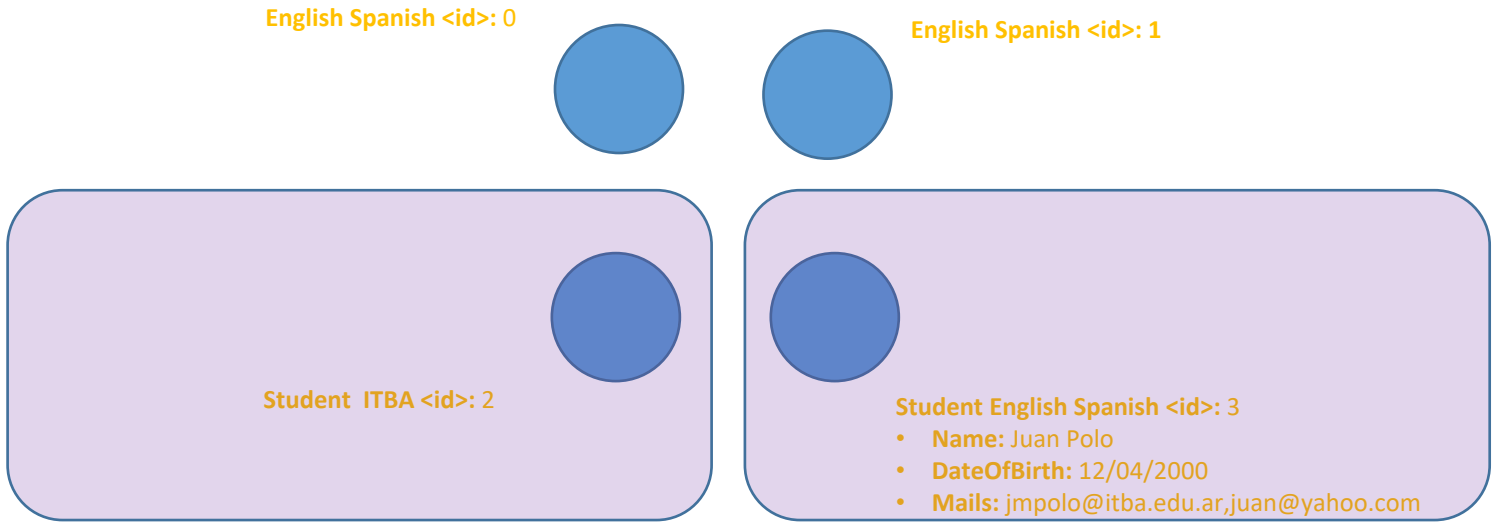
Delete labels English and Spanish from the node labelled “ITBA”

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



Cypher - nodes

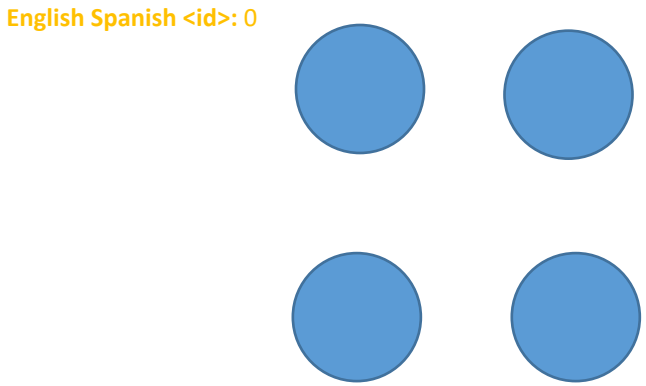
Delete properties DateOfBirth, Mails and Name from the nodes labelled “Student”.
Properties are referred to as: node.propertyName



Cypher - nodes

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

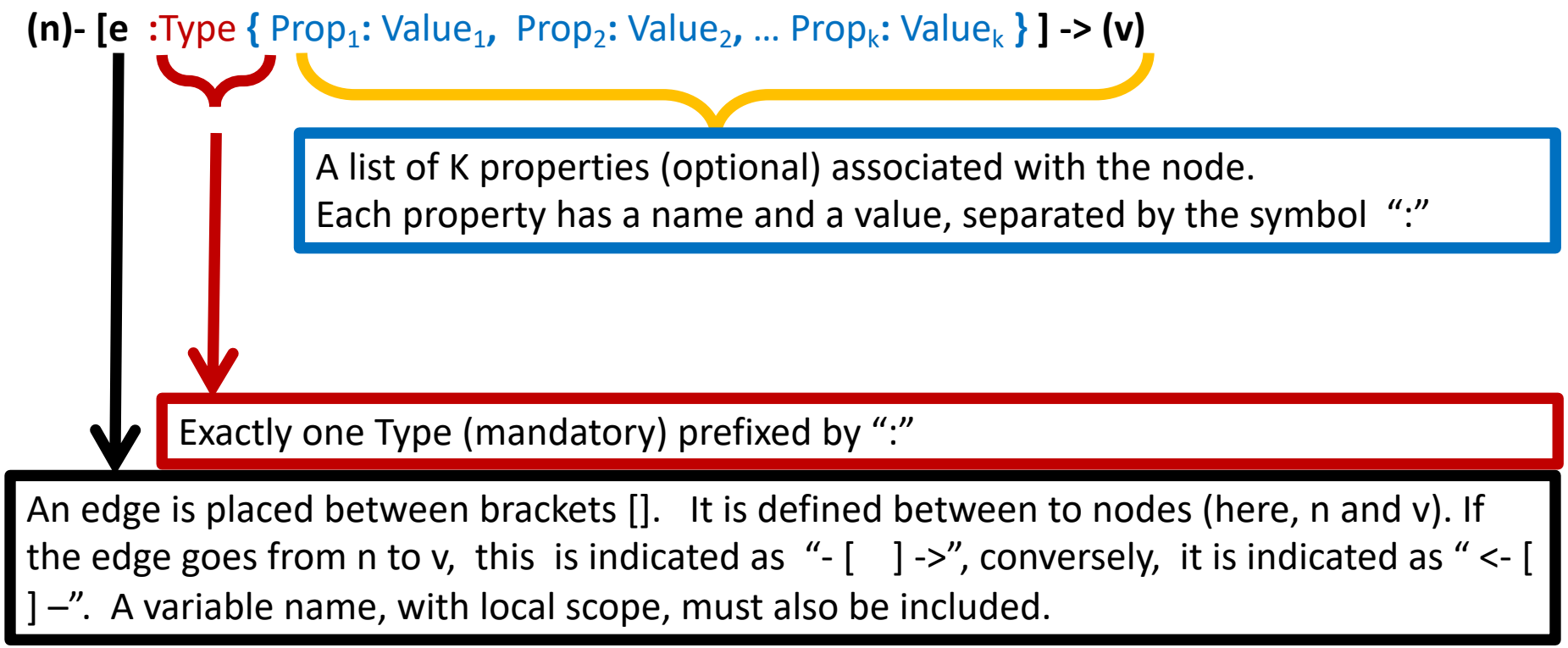
```
$ MATCH (n :Student)
  REMOVE n.DateOfBirth, n.Name, n.mails
  RETURN n
```



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



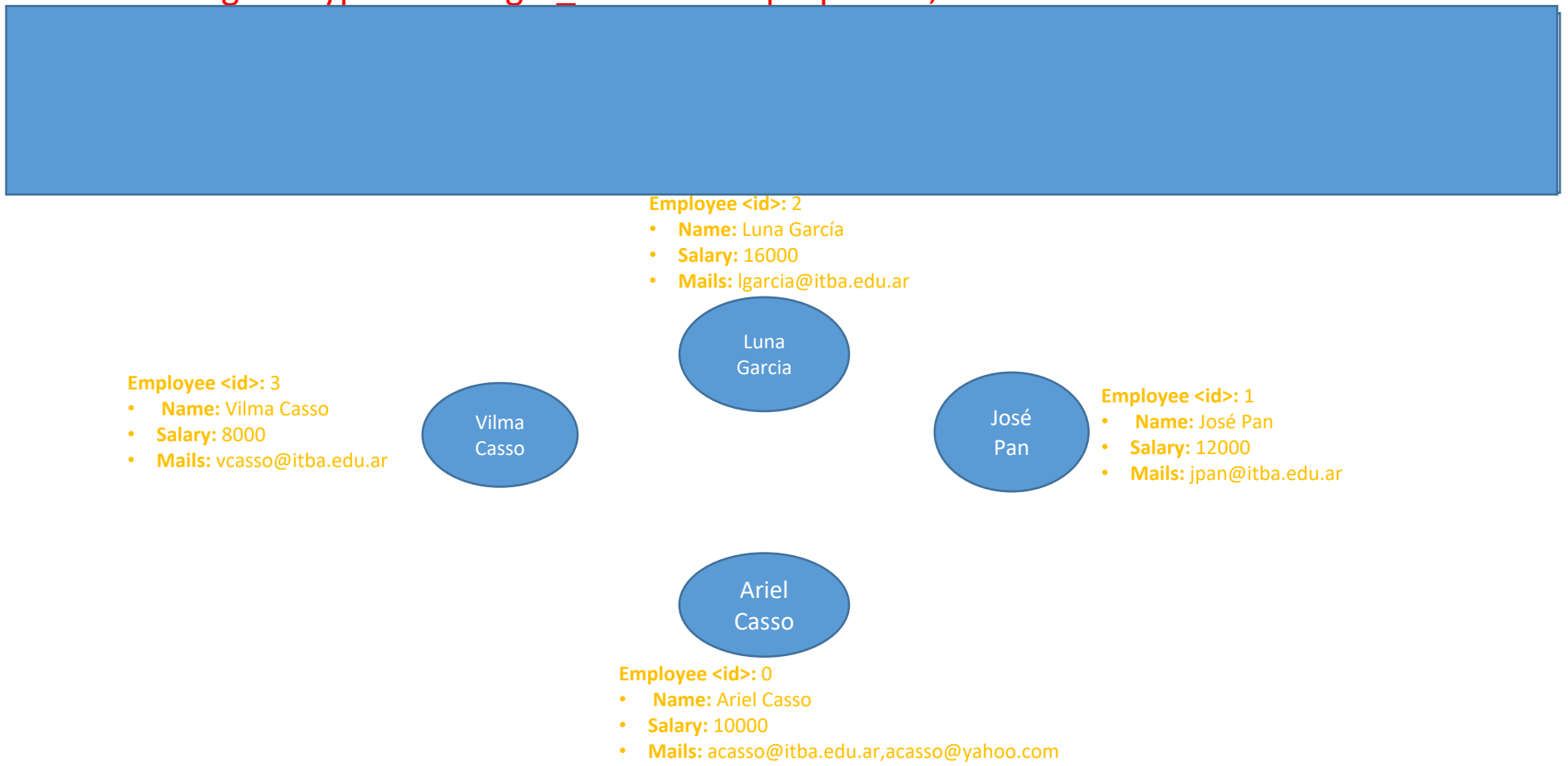
Cypher - Edges

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

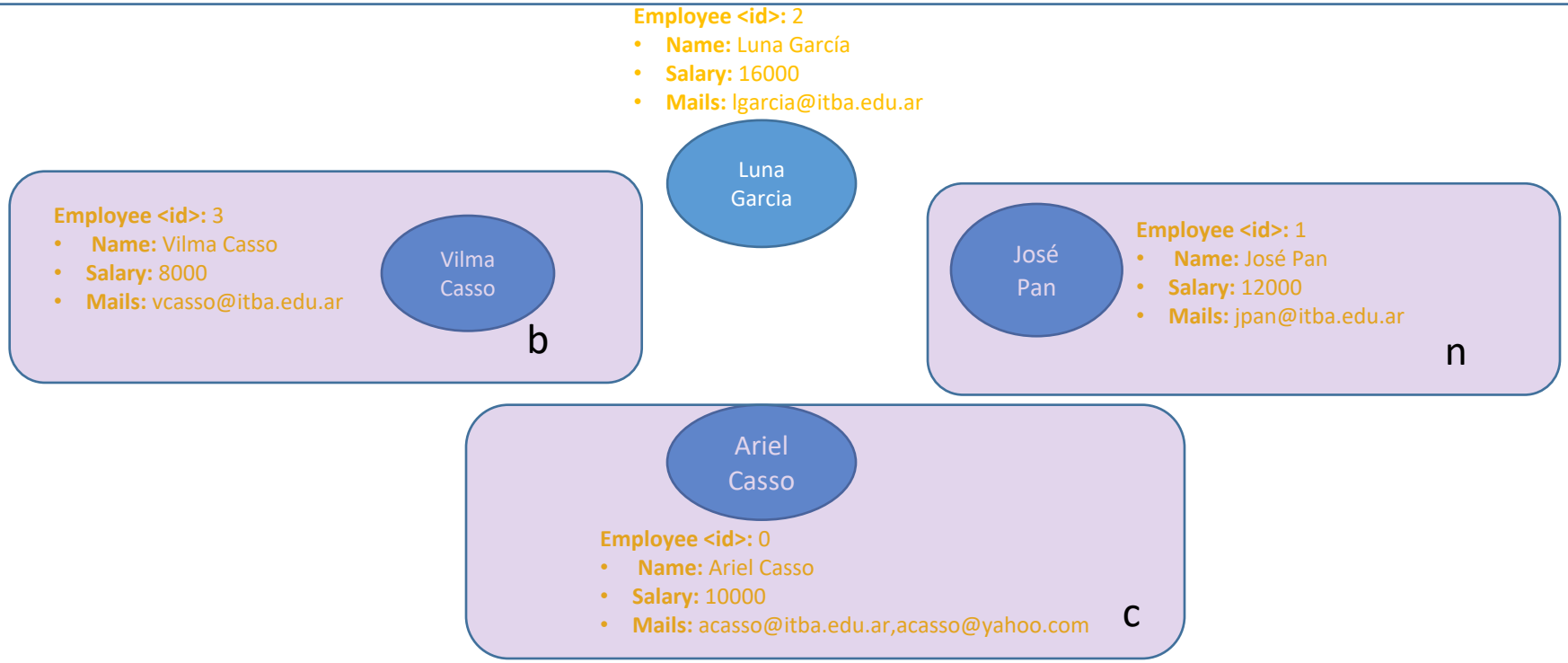
Create an edge of type «manager_of» with no properties, from José Pan to Vilma and Ariel Casso:



Cypher - Edges

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

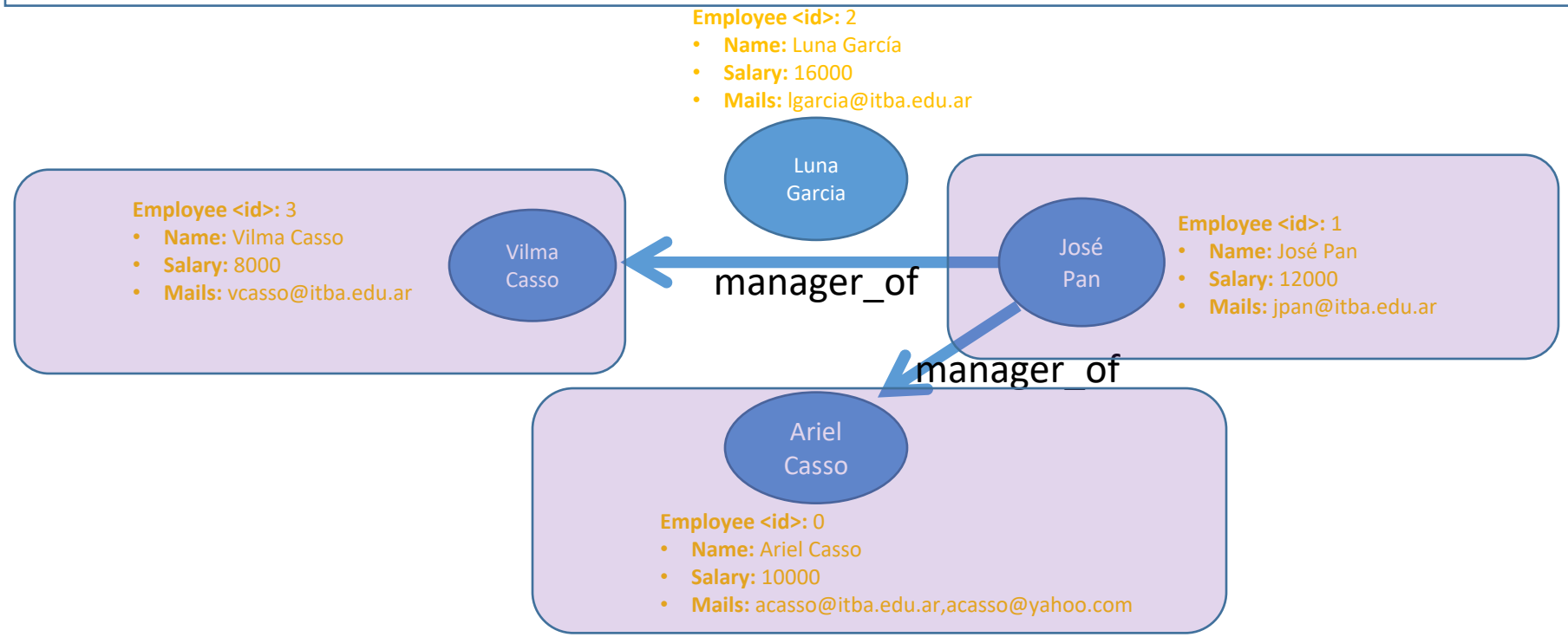
```
$ MATCH ( n  :Employee  {Name: 'José Pan'} ), ( b  :Employee  {Name: 'Vilma
  Casso'} ), ( c  :Employee  {Name: 'Ariel Casso'} )
 CREATE (b) <- [r1  :manager_of]  - (n)  - [r2  :manager_of] -> (c)
 RETURN r1, r2
```



Cypher - Edges

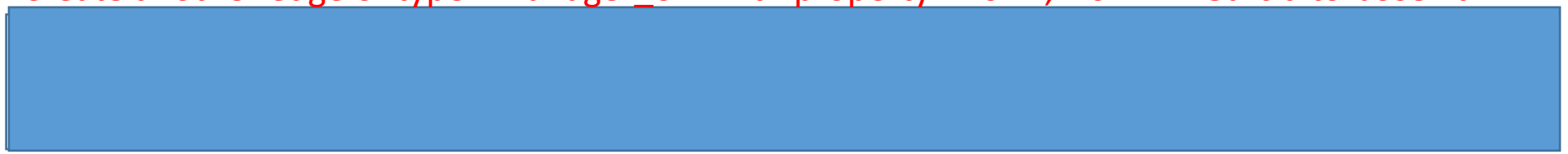
Create an edge of type «manager_of» with no properties, from José Pan to Vilma and Ariel Casso:

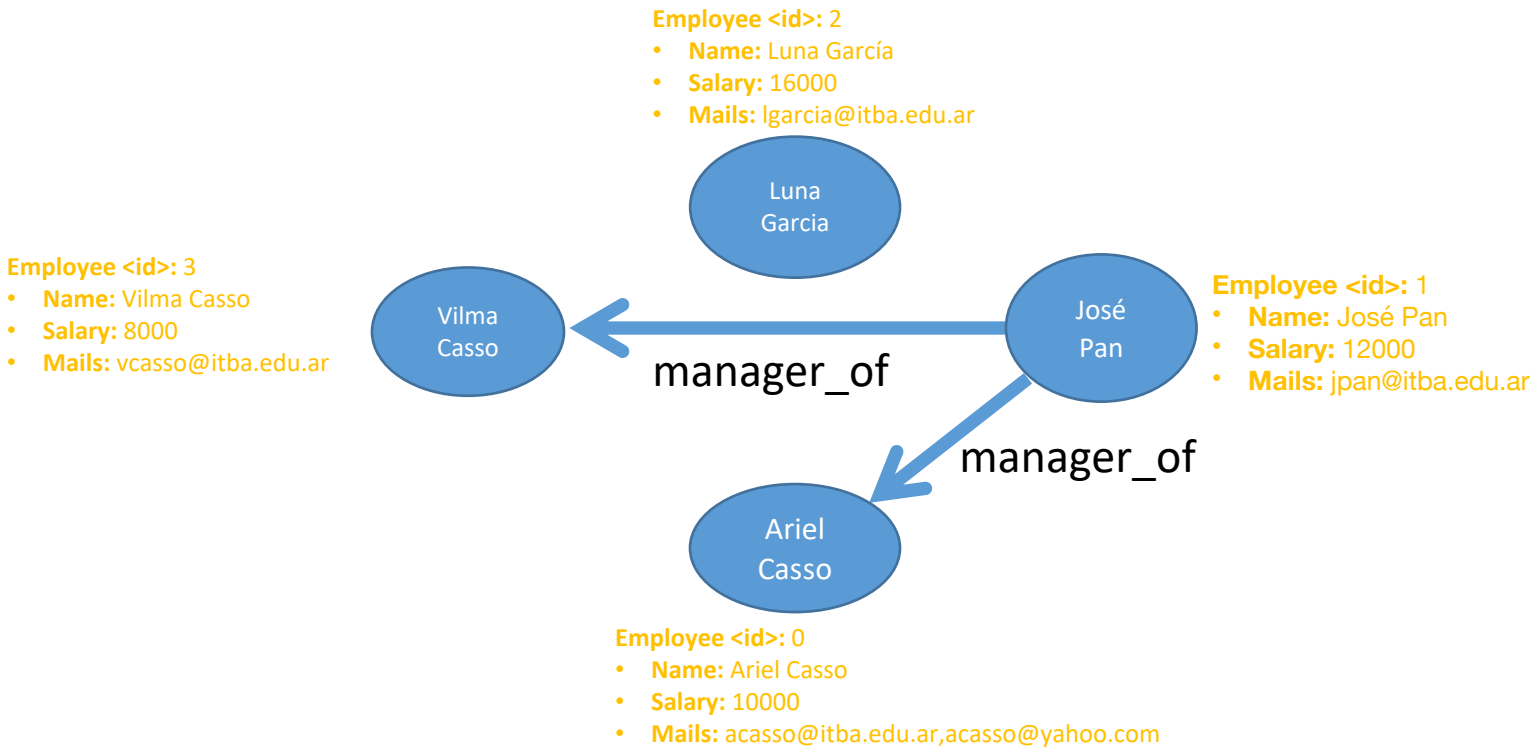
```
$ MATCH ( n  :Employee  {Name: 'José Pan'} ), ( b  :Employee  {Name: 'Vilma
  Casso'} ), ( c  :Employee  {Name: 'Ariel Casso'} )
 CREATE (b) <- [r1  :manager_of]  - (n)  - [r2  :manager_of] -> (c)
 RETURN r1, r2
```



Cypher - Edges

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

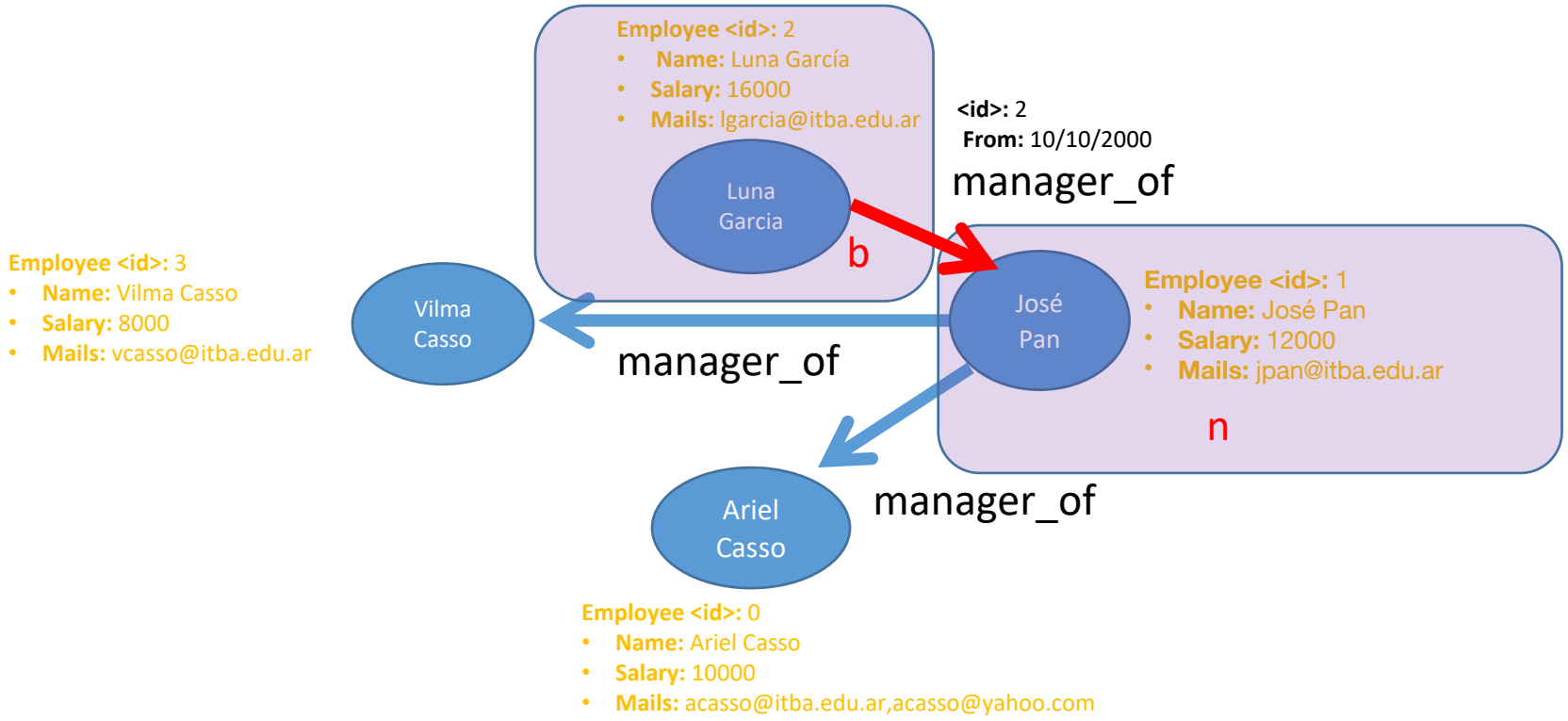




Cypher - Edges

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

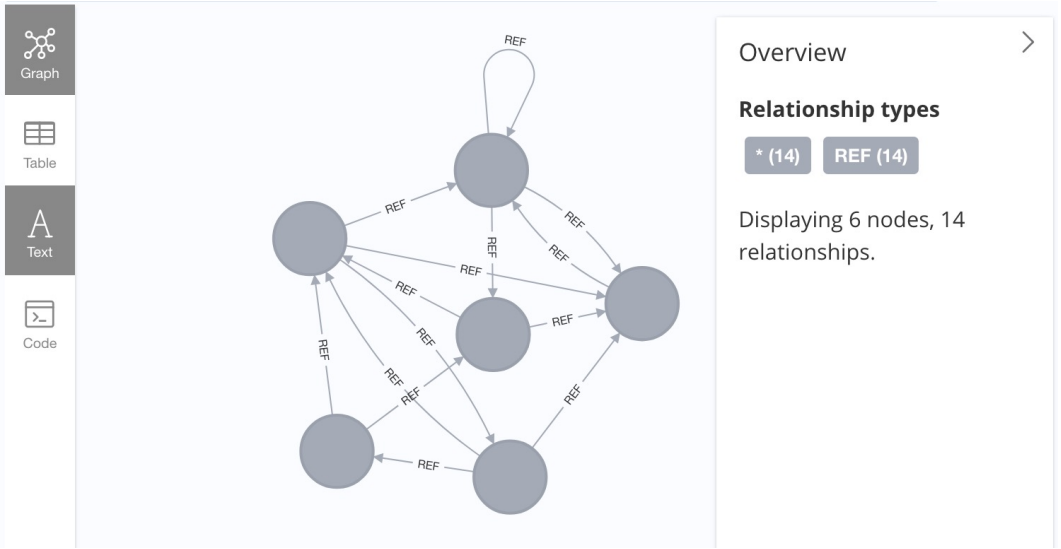
In addition to the above:

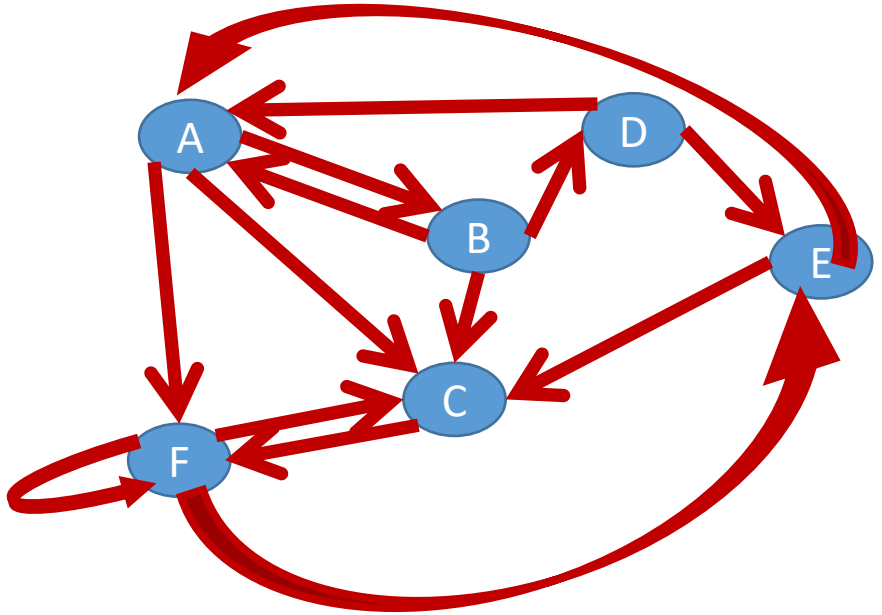
1) If we don't need to refer to a node, we can use "()", with no variable, e.g., ( ) – [:manager_of] -> ( )
2) If we don't need to refer to an edge, we can omit it, e.g.: (a) --> (b) indicates an edge between a and b.
3) If we don't need to consider the direction of the edge, just use "- -" (without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

Cypher – Example

Consider the query:

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





Cypher – Example

Consider the query:

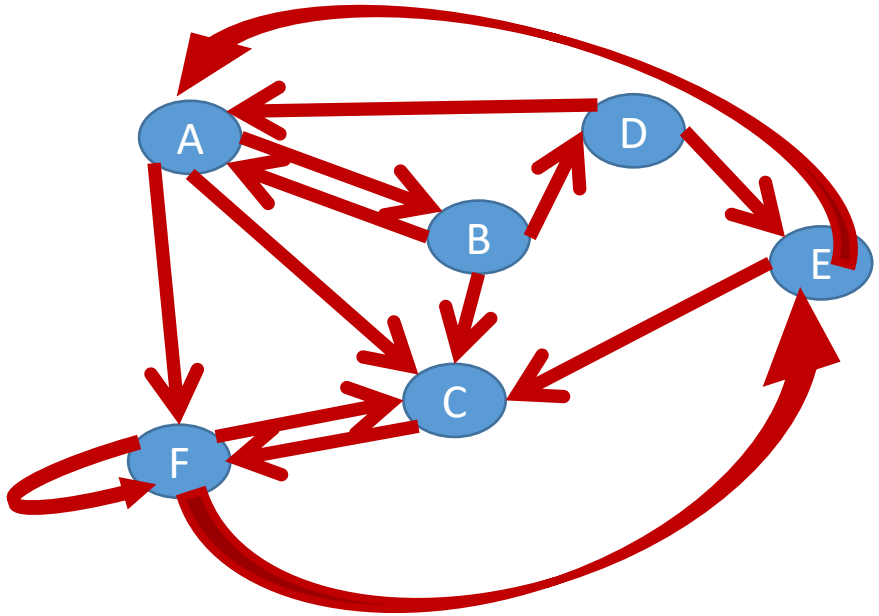
```
$ MATCH (p)-[]->(s)-[]->(x)
   RETURN Count(p), s.URL, Count(x)
```

Returns the following. Why???

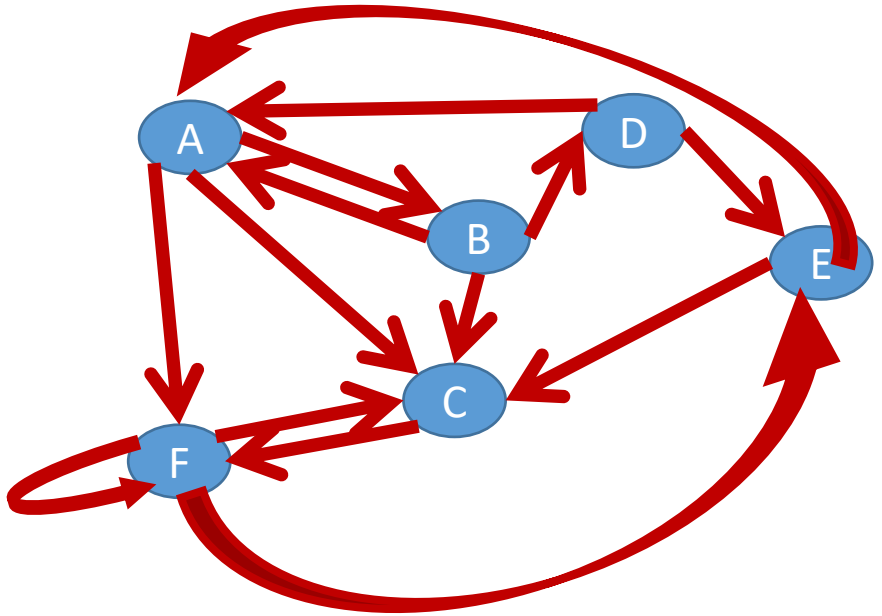
webdb.db$

Table | Text | Code

	Count(p)	s.URL	Count(x)
1	9	"A"	9
2	3	"B"	3
3	4	"C"	4
4	2	"D"	2
5	4	"E"	4
6	8	"F"	8



Cypher – Example




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

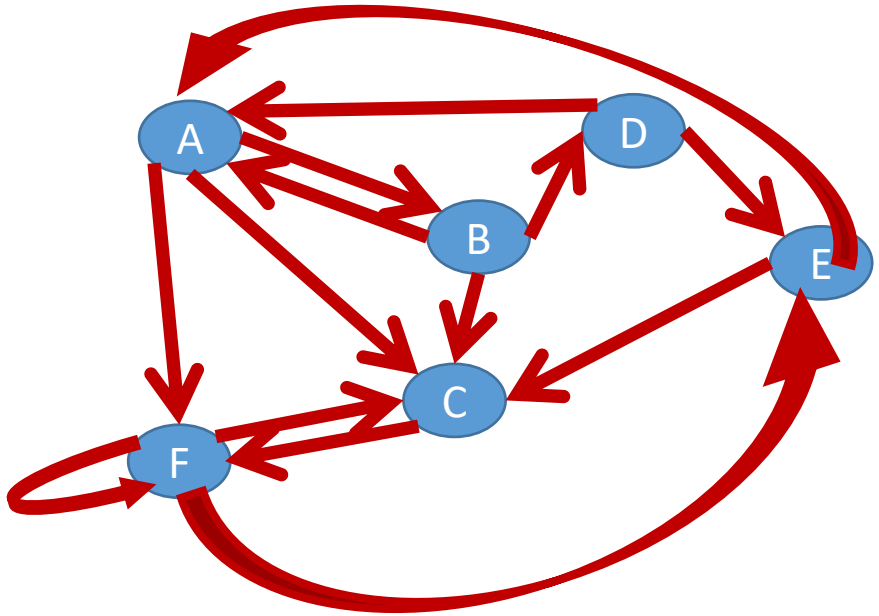
The first clause computes paths where a node (s) has an incoming and an outgoing edge.
E.g., for «c», these paths are:

(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.

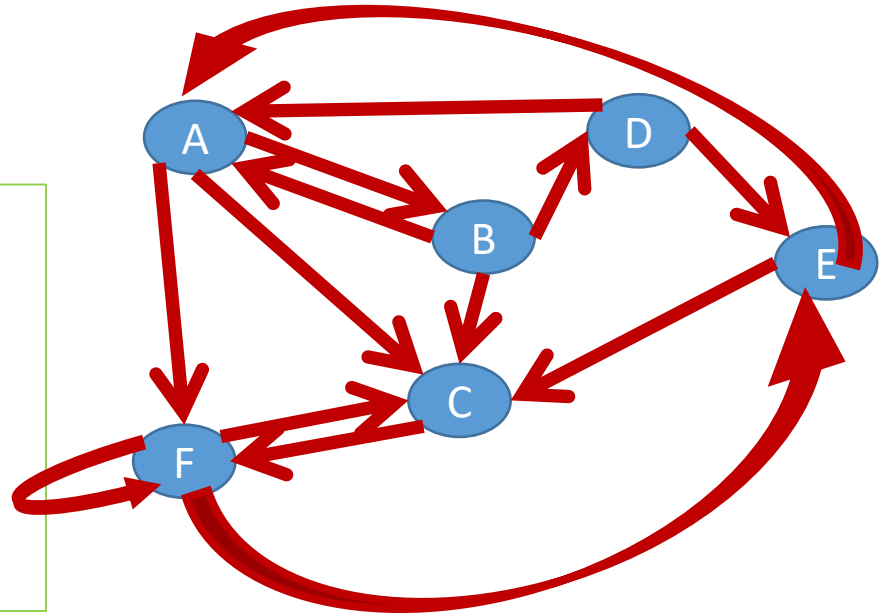
If a page Z references a page X, Z gives X a normalized vote computed as the inverse of the number of pages referenced by Z. To prevent votes of cross-referencing pages, if Z references X and X references Z, Z gives 0 votes to X.

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.URL, SUM(vote) AS Rank
ORDER BY x.URL
```



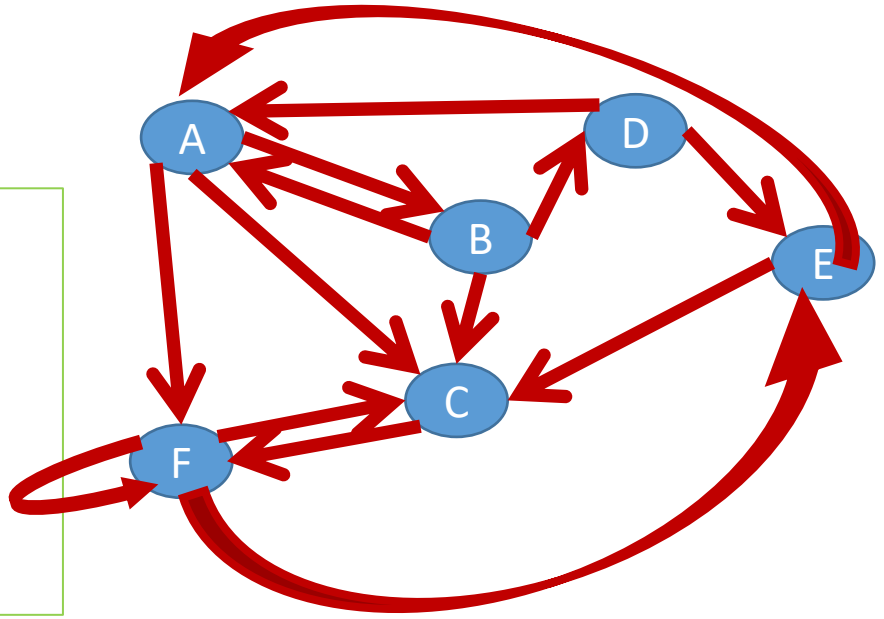
«p»	«vote»
A	0.333
B	0.333
C	1
D	0.5
E	0.5
F	0.333

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.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

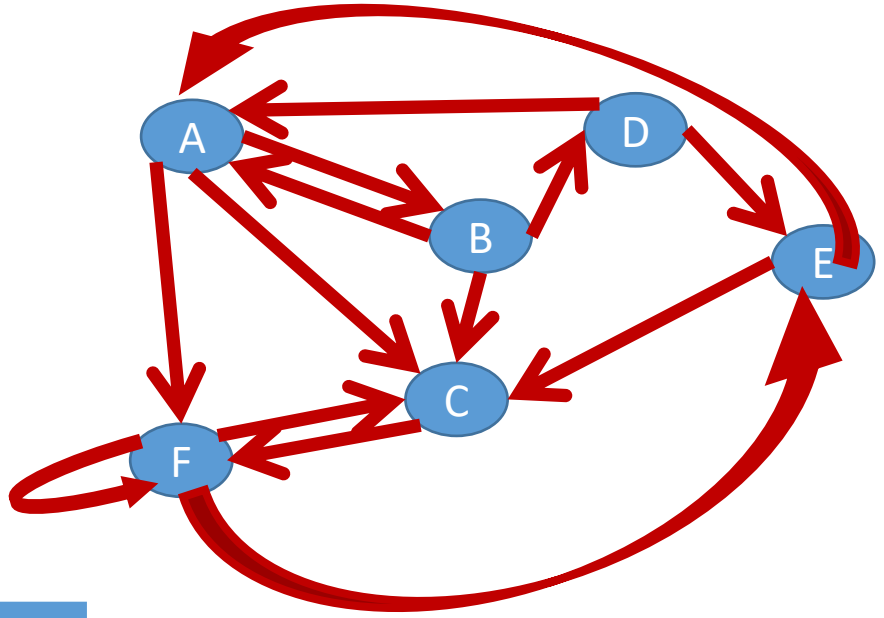
«p»	«x»
A	C
A	F
B	C
B	D
D	A
D	E
E	A
E	C
F	E

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, 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

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

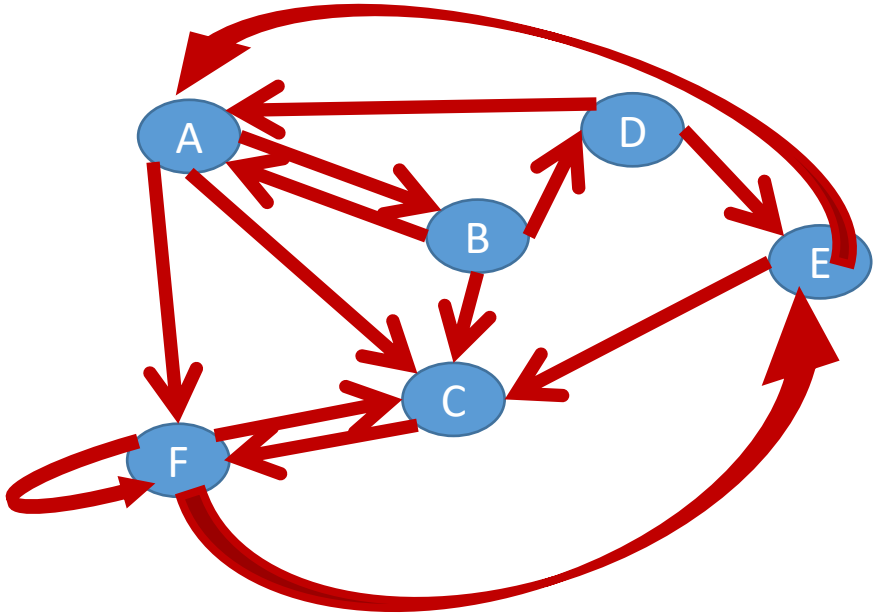
«x»	«Rank»
A	½ + ½
C	1/3 + 1/3 + 1/2
D	1/3
E	½ + 1/3
F	1/3

Finally, group results by the second component and sorts.

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

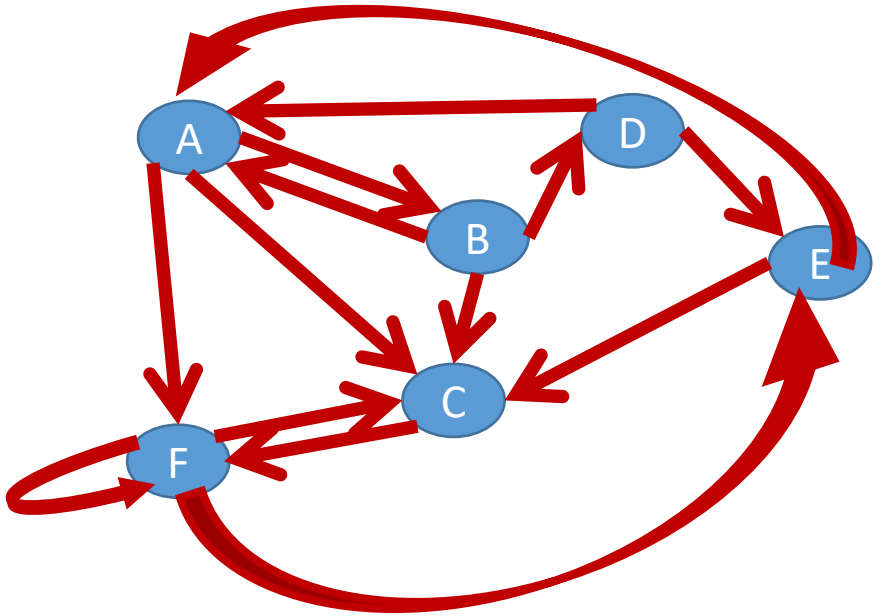
«x»	«Rank»
A	½ + ½
C	1/3 + 1/3 + 1/2
D	1/3
E	½ + 1/3
F	1/3

Finally, groups results by the second component and sorts.

Cypher – Example

Alternative solution

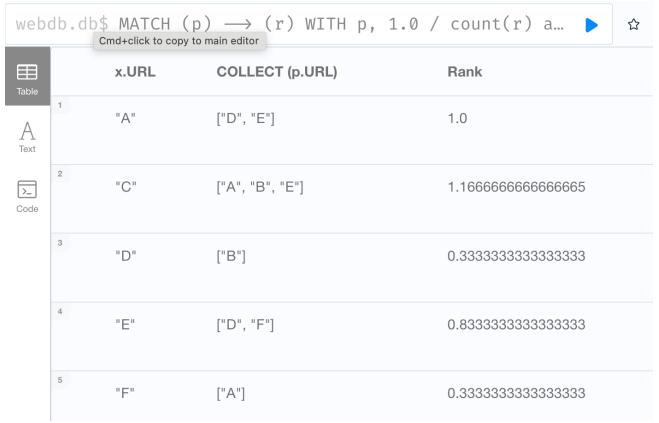
```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  RETURN  x.URL, COLLECT (p.URL), SUM(vote) AS Rank
  ORDER BY x.URL
```



«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

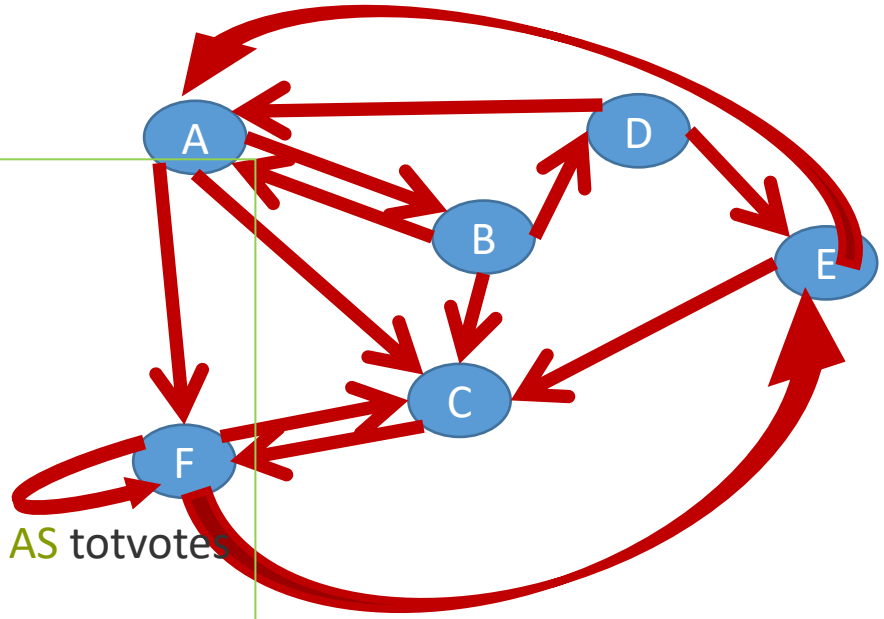
«x»	«p» grouped
A	D, E
C	A, B, E
D	B
E	D, F
F	A



Cypher – Example

Alternative solution

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  WITH x.URL as x, COLLECT(vote) as votelist
  RETURN x, reduce(totalvotes = 0.0, n IN votelist| totalvotes + n)  AS totvotes
  ORDER BY x
```

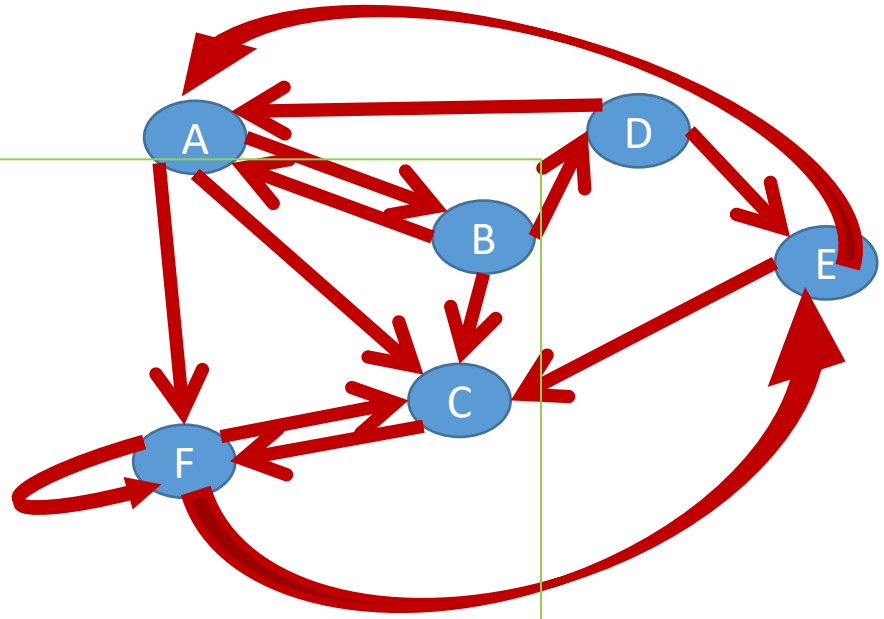


x	totvotes
"A"	1.0
"C"	1.1666666666666665
"D"	0.3333333333333333
"E"	0.8333333333333333
"F"	0.3333333333333333

Cypher – Example

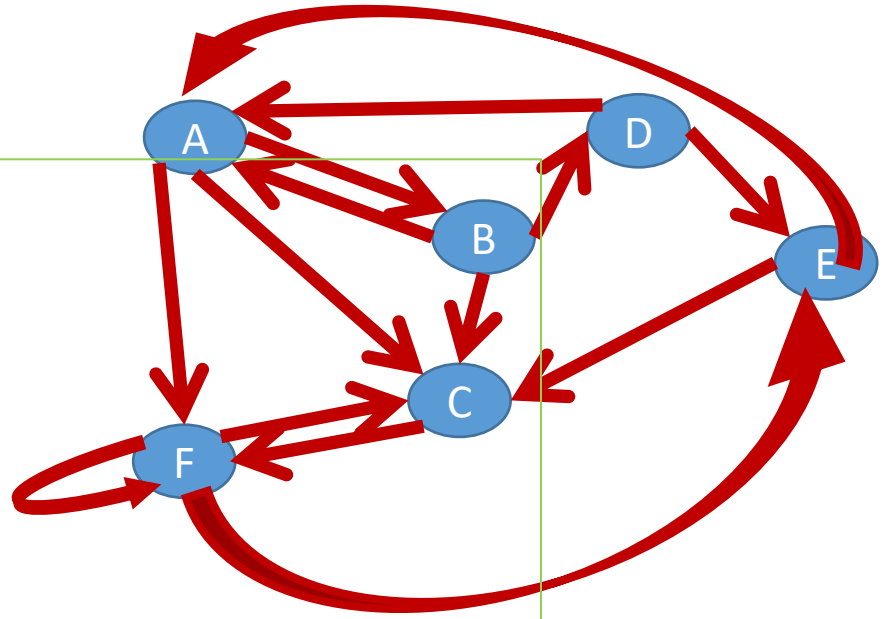
Alternative solution using the APOC library

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  WITH x.URL as x, COLLECT(vote) as votelist
  // APOC library
  RETURN x, votelist, apoc.coll.sum(votelist) as votes
ORDER BY x
```



x	ll	votes
"A"	[0.5, 0.5]	1.0
"C"	[0.3333333333333333, 0.3333333333333333, 0.5]	1.1666666666666665
"D"	[0.3333333333333333]	0.3333333333333333
"E"	[0.5, 0.3333333333333333]	0.8333333333333333

Cypher – Example



Query: Who gives "x" less than 0.5 votes?

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  WITH x.URL as x, COLLECT([p.URL,vote]) as votelist
  // COMPREHENSION LIST
 RETURN x, [p in votelist WHERE p[1] < 0.5|p[0]] AS lowvotes
 ORDER BY x
```

x	votelist
"A"	[["D", 0.5], ["E", 0.5]]
"C"	[["A", 0.3333333333333333], ["B", 0.3333333333333333], ["E", 0.5]]
"D"	[["B", 0.3333333333333333]]
"E"	[["D", 0.5], ["F", 0.3333333333333333]]

x	lowvotes
"A"	[]
"C"	["A", "B"]
"D"	["B"]
"E"	["F"]