Introduction to Graph Databases

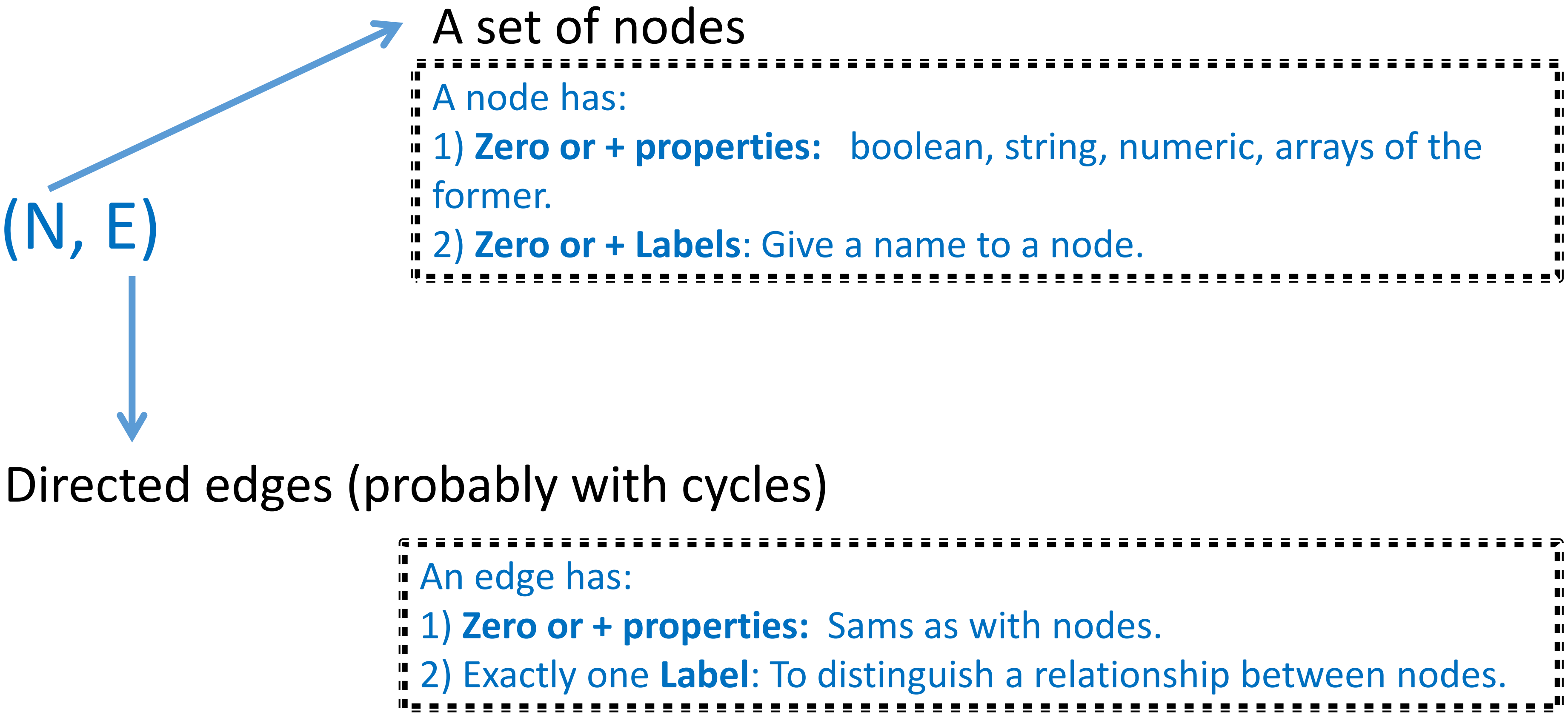
Neo4j

Alejandro Vaisman
avaisman@itba.edu.ar

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



(N, E)

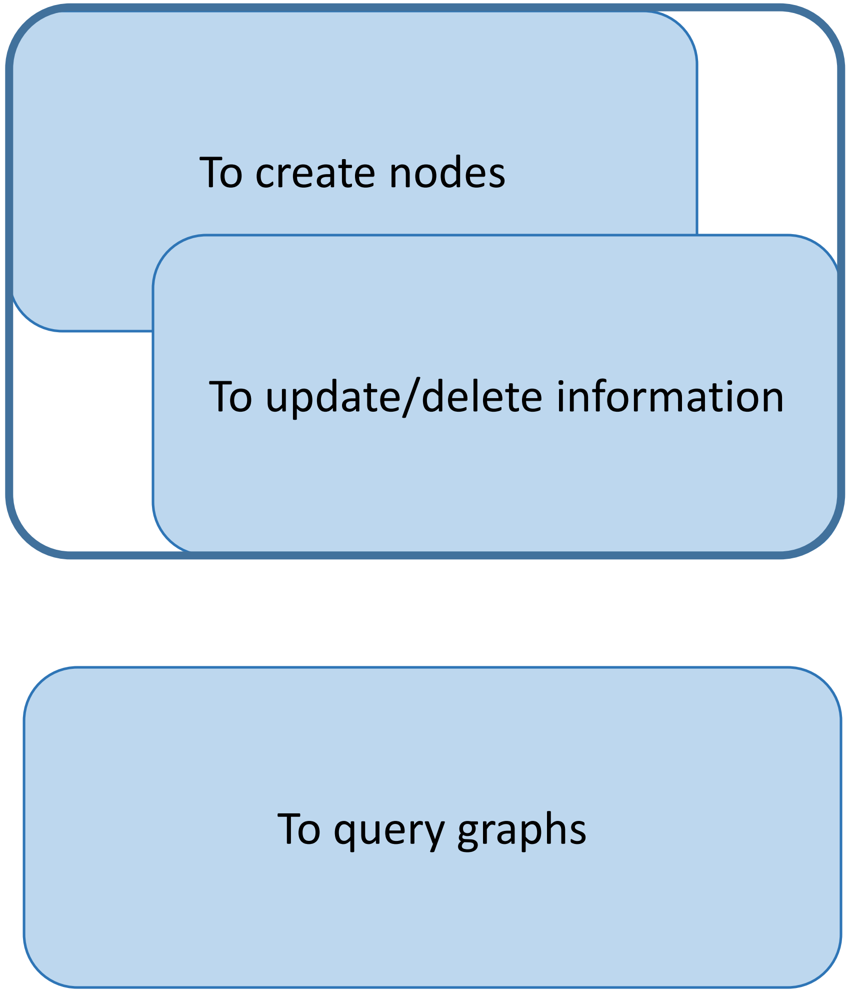
A set of nodes

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.

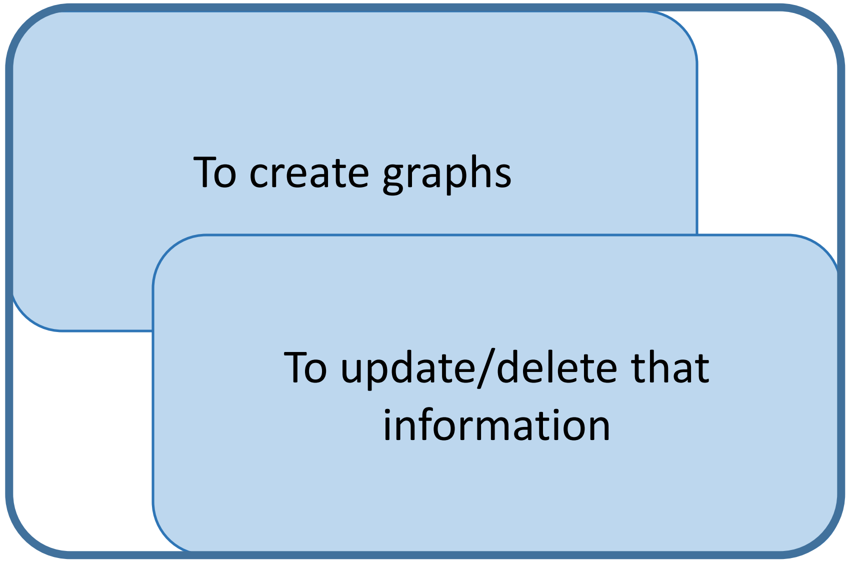
Cypher


To create nodes

To update/delete information

To query graphs

Cypher



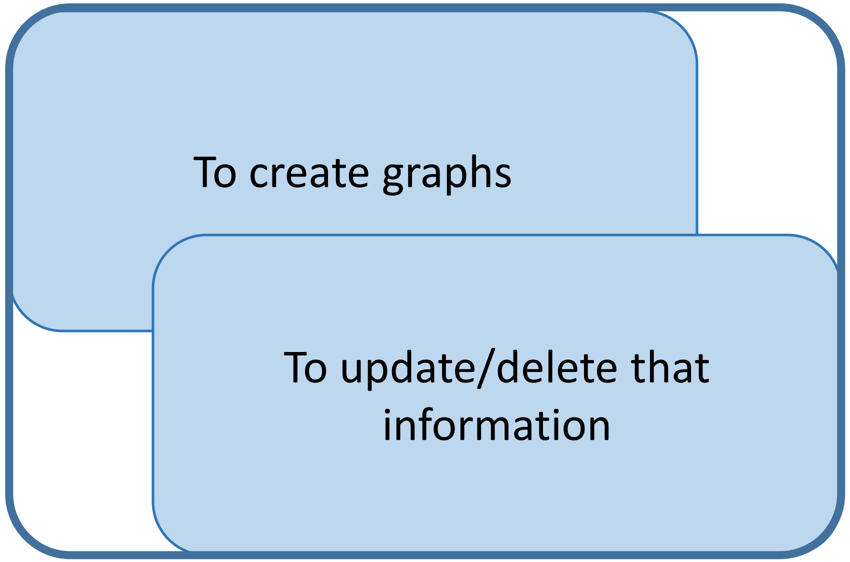
To create graphs

To update/delete that information

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.

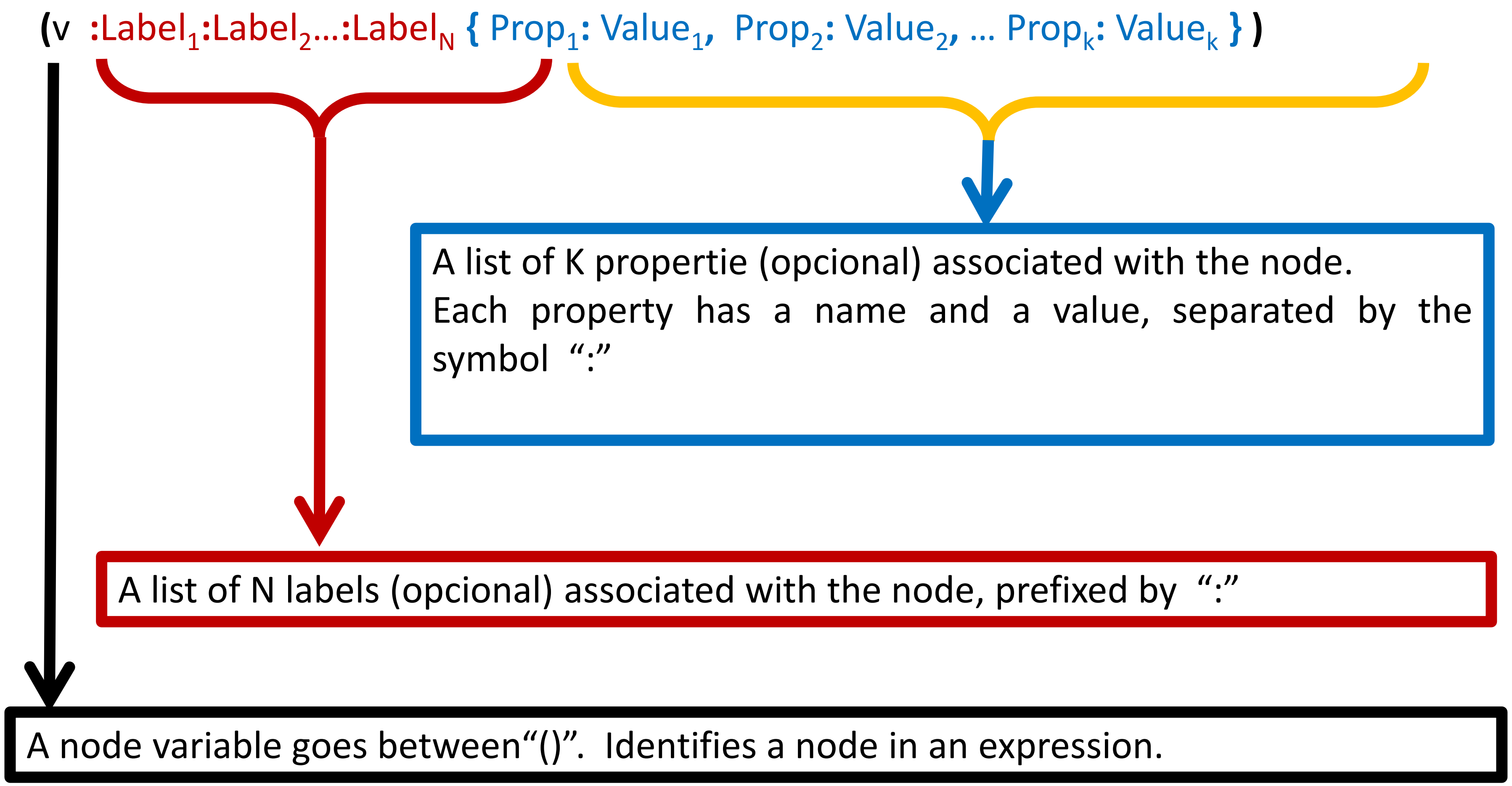
Cypher



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



(v :Label$_1$:Label$_2$…:Label$_N$ { Prop$_1$: Value$_1$, Prop$_2$: Value$_2$, … Prop$_k$: Value$_k$ })

A list of K propertie (opcional) associated with the node.
Each property has a name and a value, separated by the symbol “:”

A list of N labels (opcional) associated with the node, prefixed by “:”

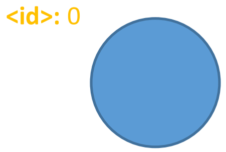
A node variable goes between“()”. Identifies a node in an expression.

Cypher - nodes

Create a node with no properties/labels:

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

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



Create another one.

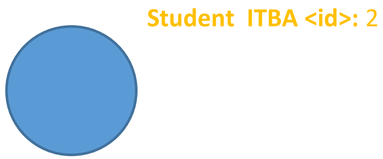
```
$ CREATE ();
```

If RETURN is not written, nodes are not displayed

Cypher - nodes

Create a node with two labels:

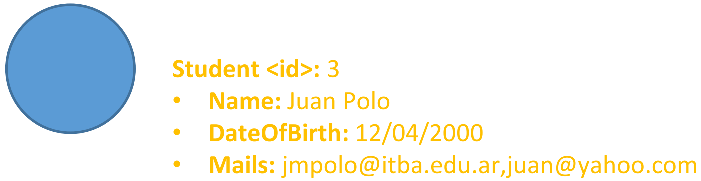
```
$ CREATE (v  :Student:ITBA)
   RETURN v;
```



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;
```

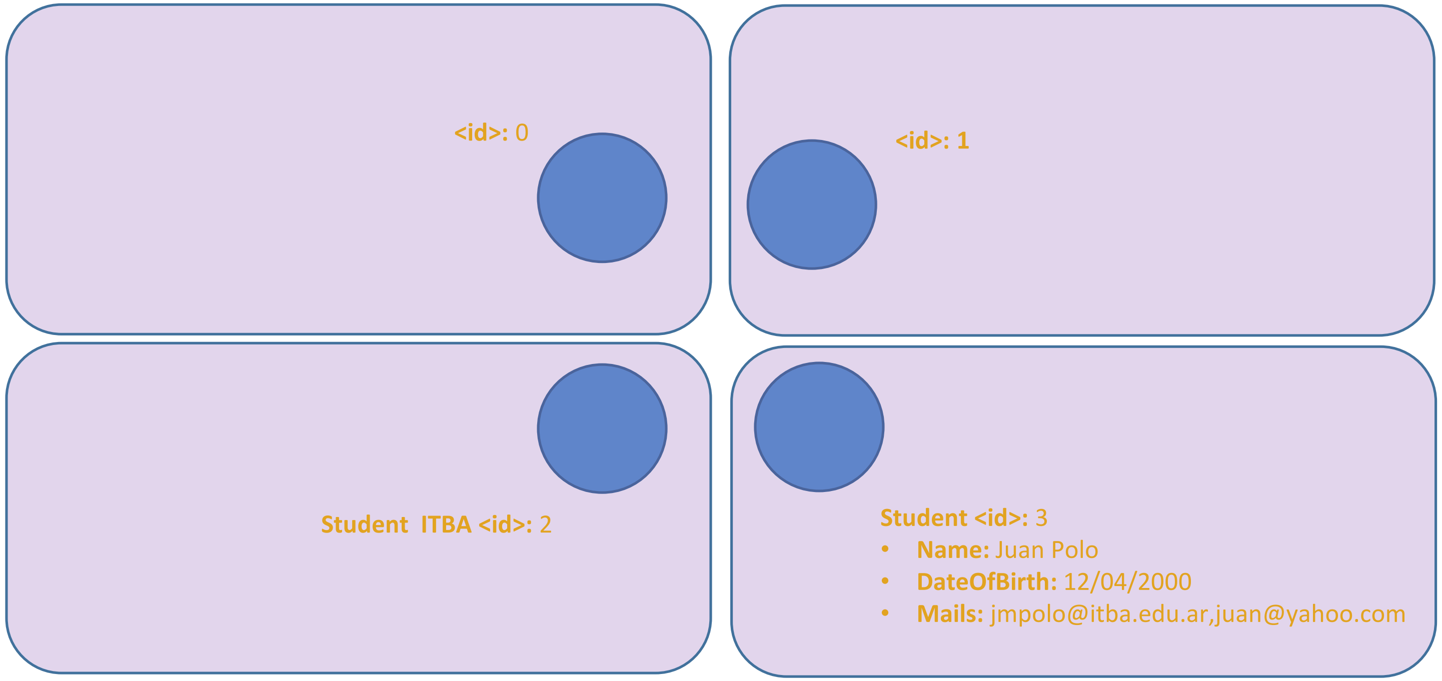


Student <id>: 3

- **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.



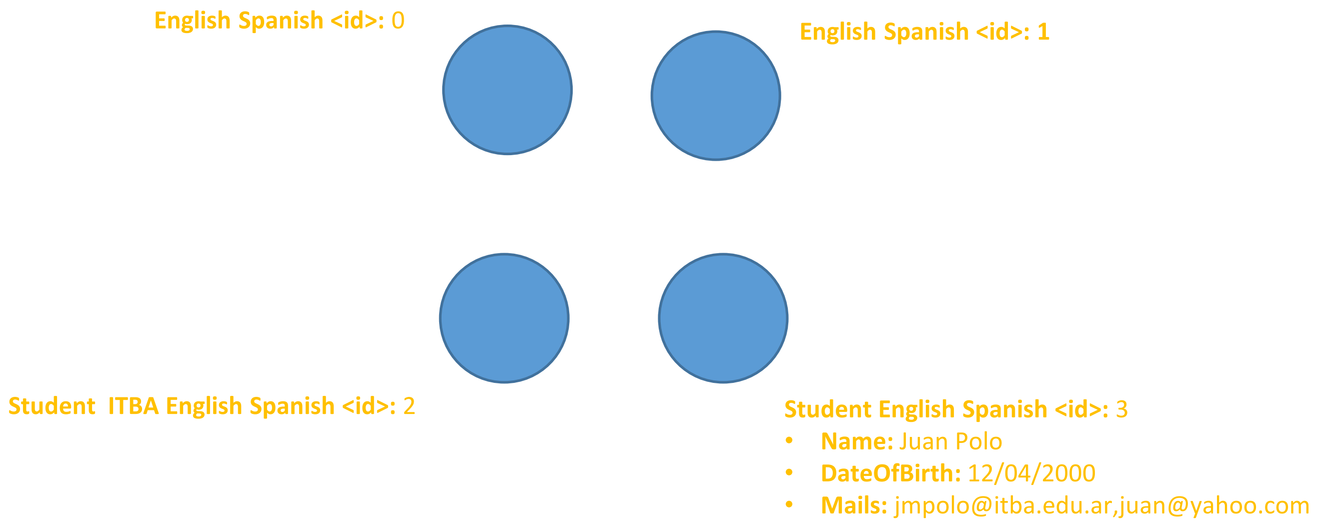
Student <id>: 3

- **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.

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



English Spanish <id>: 0

English Spanish <id>: 1

Student ITBA English Spanish <id>: 2

Student English Spanish <id>: 3

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

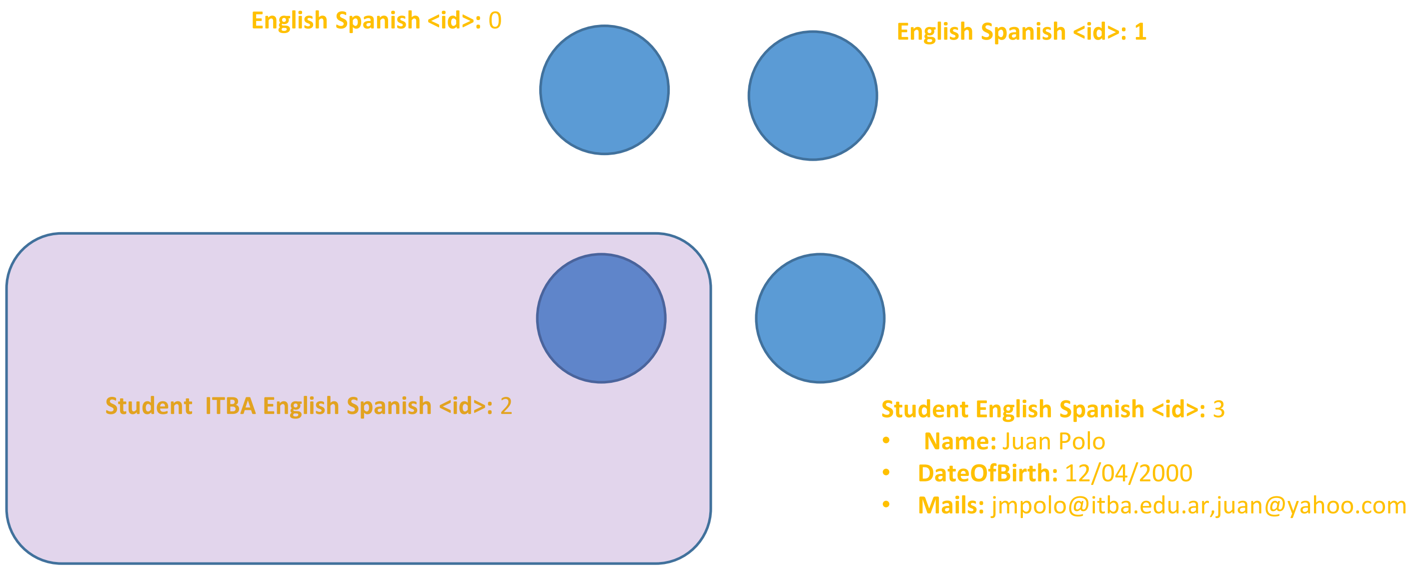
Cypher - nodes

Delete labels English and Spanish from the node labelled "ITBA"

$

MATCH (n :ITBA)

[redacted]



English Spanish <id>: 0

English Spanish <id>: 1

Student ITBA English Spanish <id>: 2

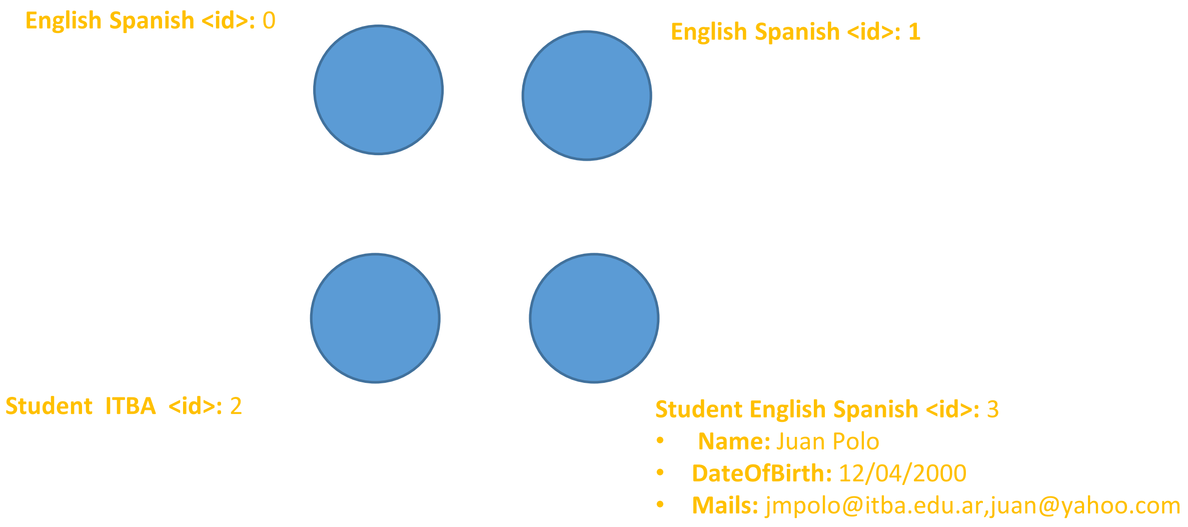
Student English Spanish <id>: 3

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

Cypher - nodes

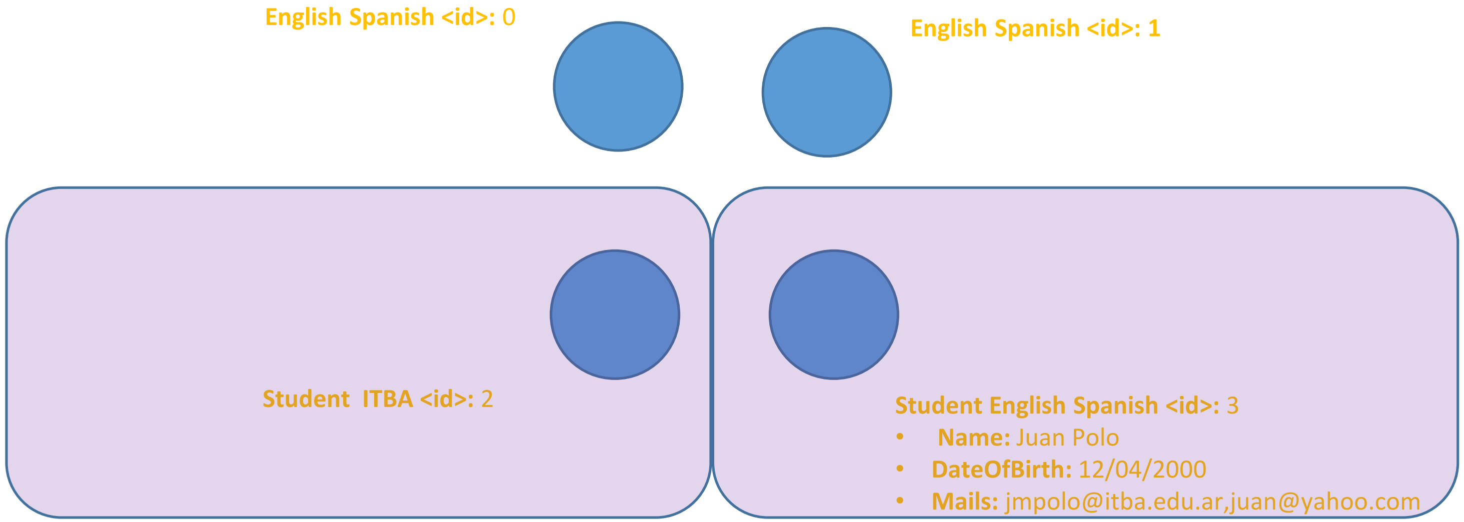
Delete labels English and Spanish from the node labelled “ITBA”

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


English Spanish <id>: 0
English Spanish <id>: 1
Student ITBA <id>: 2
Student English Spanish <id>: 3
- **Name:** Juan Polo
- **DateOfBirth:** 12/04/2000
- **Mails:** jmpolo@itba.edu.ar,juan@yahoo.com

Cypher - nodes

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

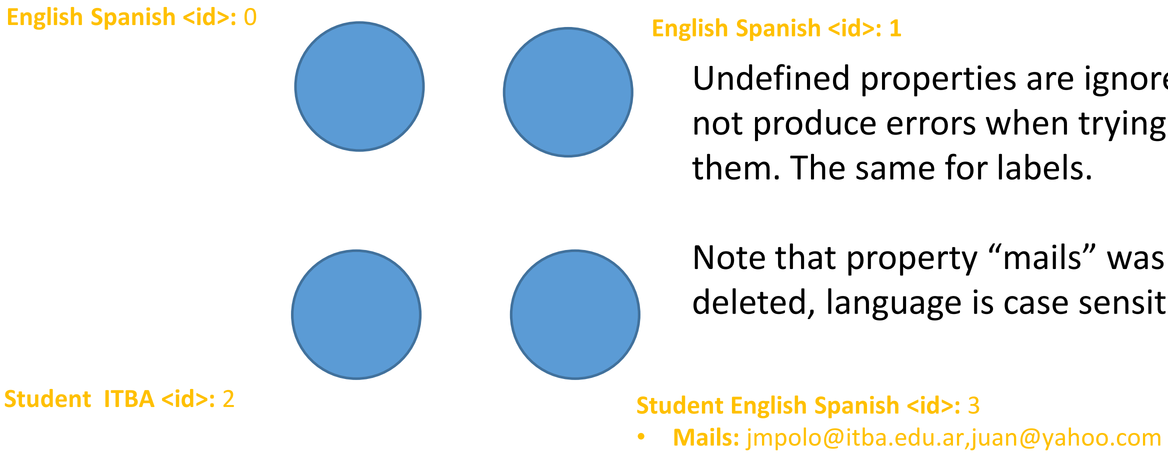


Cypher - nodes

Delete properties DateOfBirth, Name and Age from the nodes labelled “Student”.

Properties are referred to as: node.propertyName

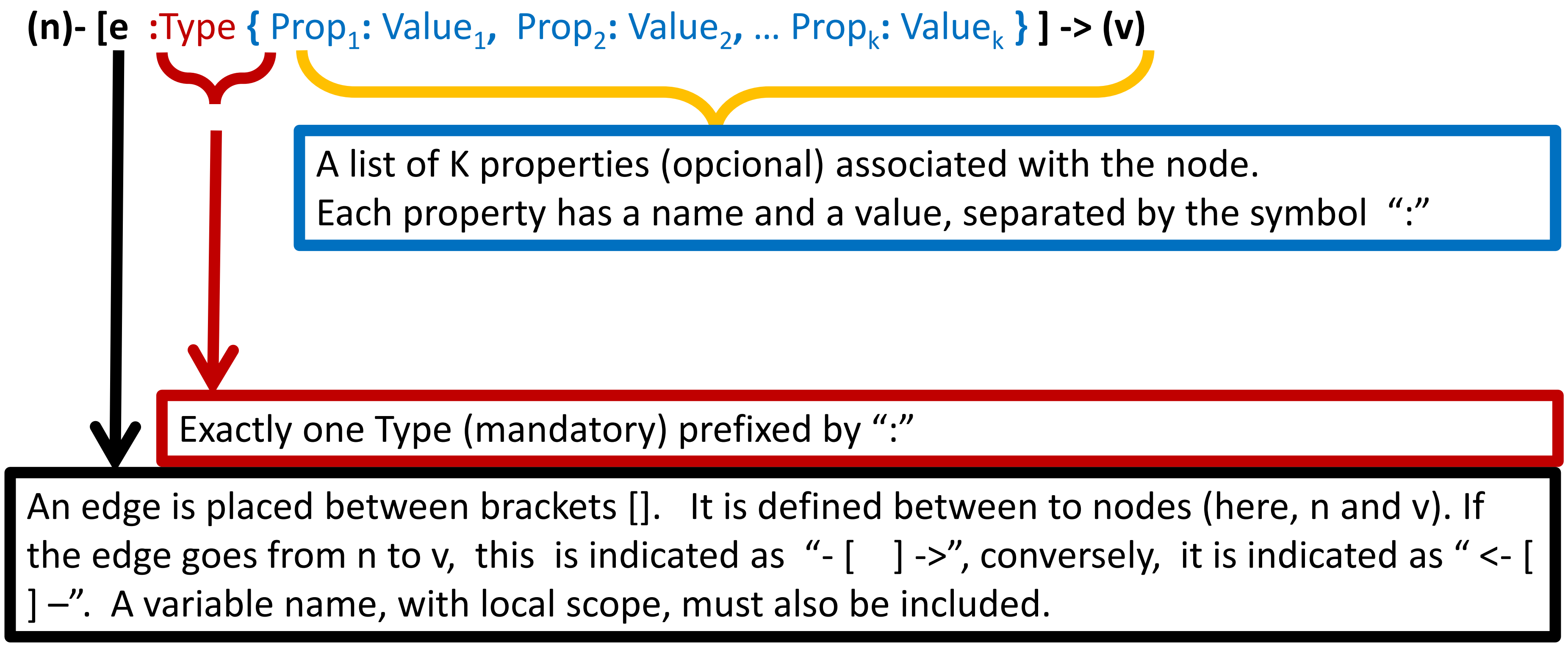
```
$ MATCH (n  :Student)
  REMOVE n.DateOfBirth, n.Name, n.mails, n.Age
  RETURN n
```


English Spanish <id>: 0
English Spanish <id>: 1
Student ITBA <id>: 2
Student English Spanish <id>: 3
• Mails: jmpolo@itba.edu.ar,juan@yahoo.com

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



(n)- [e :Type { Prop$_1$: Value$_1$, Prop$_2$: Value$_2$, … Prop$_k$: Value$_k$ }] -> (v)

A list of K properties (opcional) associated with the node.
Each property has a name and a value, separated by the symbol “:”

Exactly one Type (mandatory) prefixed by “:”

An edge is placed between brackets []. It is defined between to nodes (here, n and v). If the edge goes from n to v, this is indicated as “- [ ] ->”, conversely, it is indicated as “ <- [ ] –”. A variable name, with local scope, must also be included.

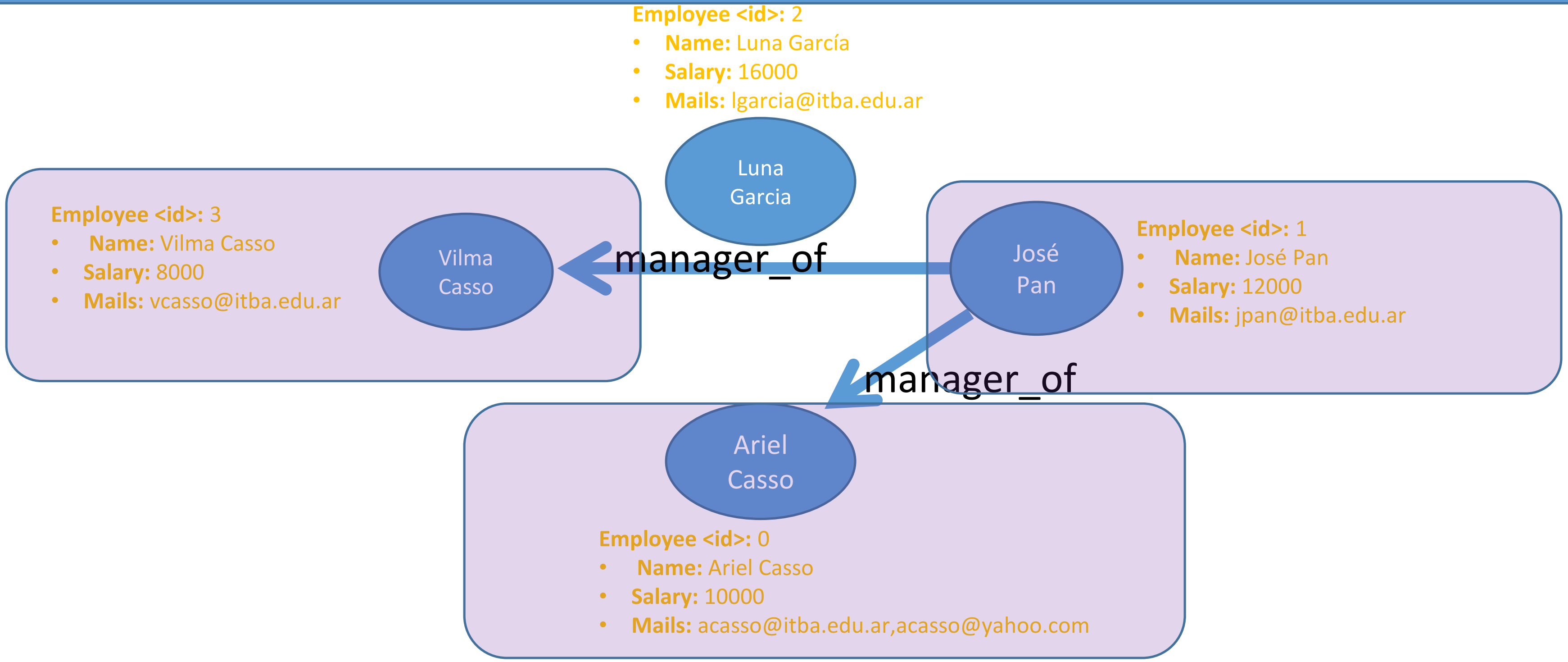
Cypher - Edges

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:



Employee <id>: 2
- **Name:** Luna García
- **Salary:** 16000
- **Mails:** lgarcia@itba.edu.ar

Employee <id>: 3
- **Name:** Vilma Casso
- **Salary:** 8000
- **Mails:** vcasso@itba.edu.ar

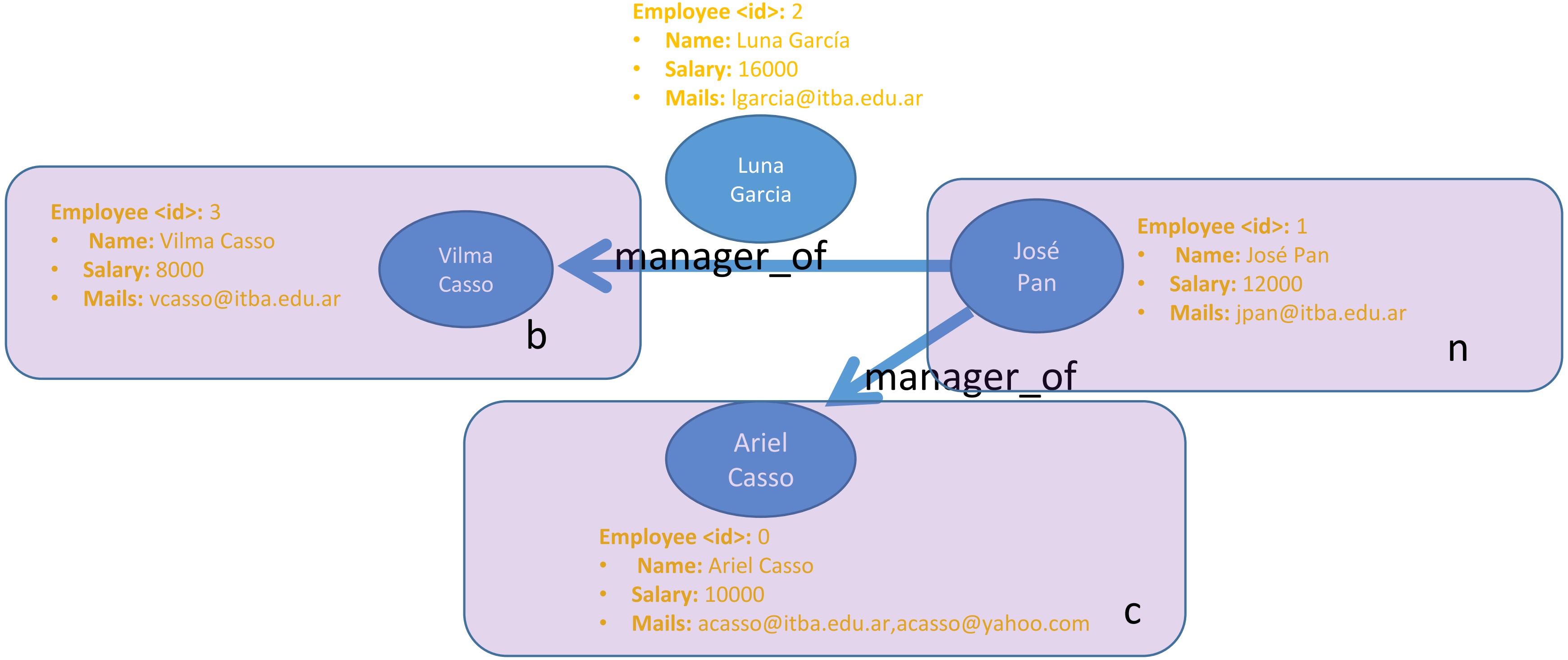
Employee <id>: 1
- **Name:** José Pan
- **Salary:** 12000
- **Mails:** jpan@itba.edu.ar

Employee <id>: 0
- **Name:** Ariel Casso
- **Salary:** 10000
- **Mails:** acasso@itba.edu.ar,acasso@yahoo.com

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
```



Employee <id>: 2
- **Name:** Luna García
- **Salary:** 16000
- **Mails:** lgarcia@itba.edu.ar

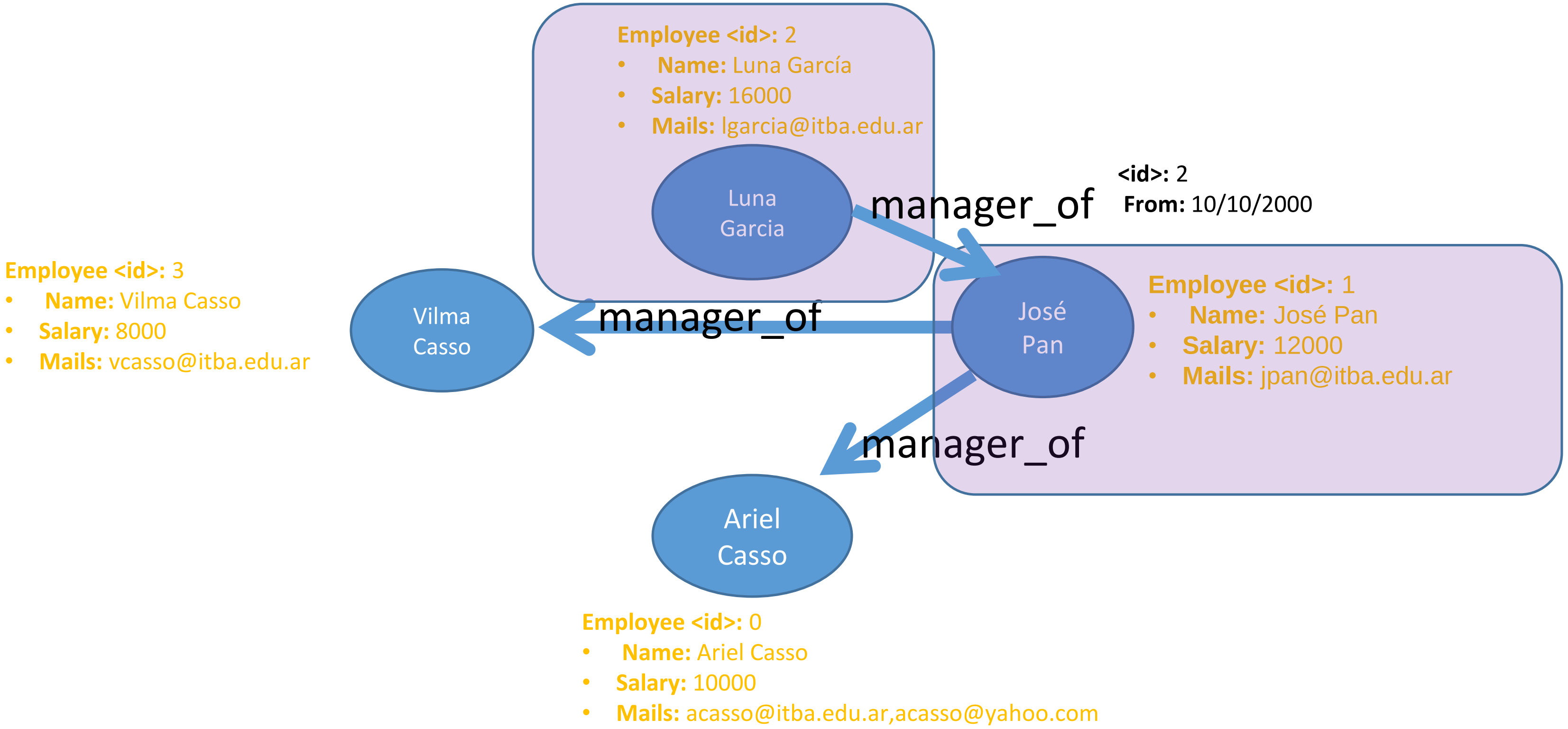
Employee <id>: 3
- **Name:** Vilma Casso
- **Salary:** 8000
- **Mails:** vcasso@itba.edu.ar

Employee <id>: 1
- **Name:** José Pan
- **Salary:** 12000
- **Mails:** jpan@itba.edu.ar

Employee <id>: 0
- **Name:** Ariel Casso
- **Salary:** 10000
- **Mails:** acasso@itba.edu.ar,acasso@yahoo.com

Cypher - Edges

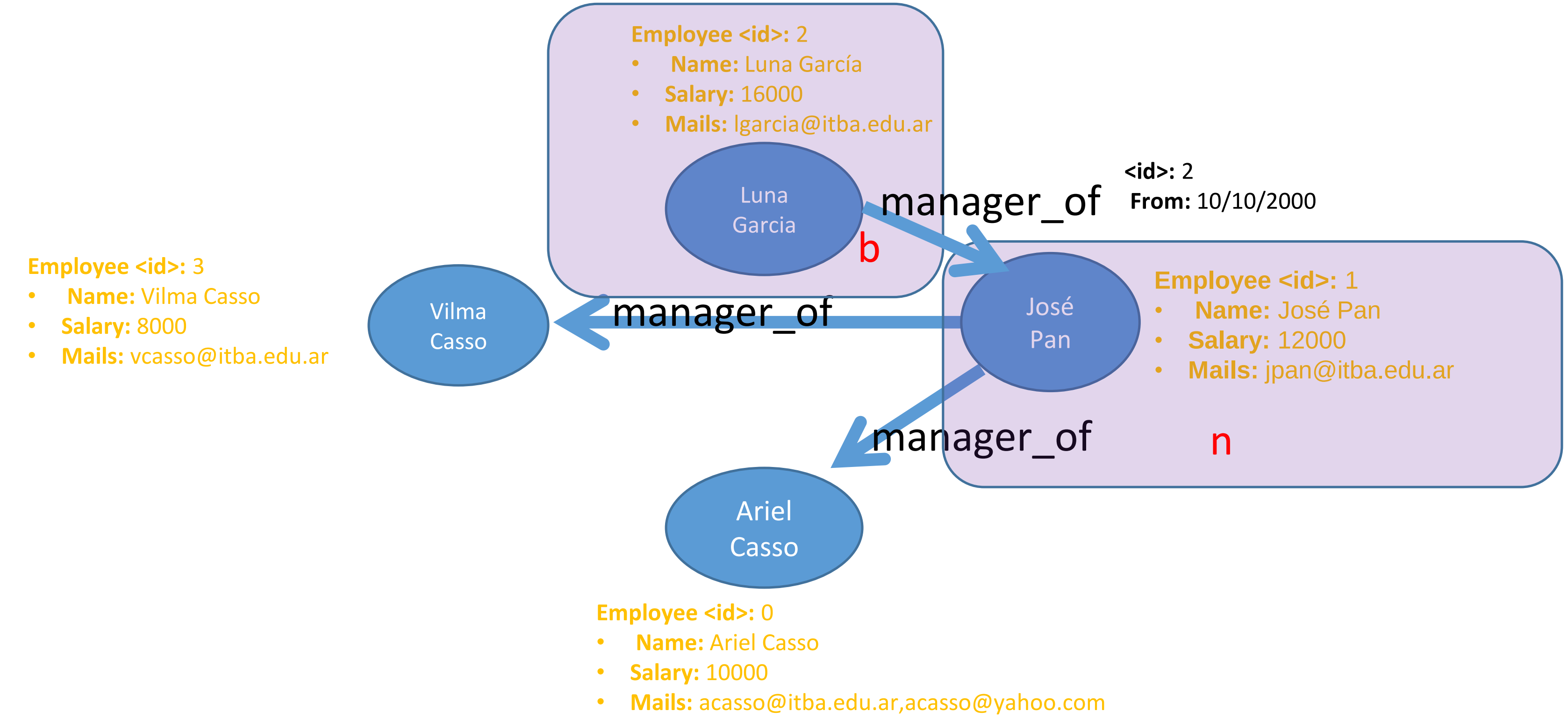
Create another edge of type «manager_of» with property “from”, from L. García to José Pan



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
```



Employee <id>: 2
- **Name:** Luna García
- **Salary:** 16000
- **Mails:** lgarcia@itba.edu.ar

<id>: 2
From: 10/10/2000

Employee <id>: 3
- **Name:** Vilma Casso
- **Salary:** 8000
- **Mails:** vcasso@itba.edu.ar

Employee <id>: 1
- **Name:** José Pan
- **Salary:** 12000
- **Mails:** jpan@itba.edu.ar

Employee <id>: 0
- **Name:** Ariel Casso
- **Salary:** 10000
- **Mails:** acasso@itba.edu.ar,acasso@yahoo.com

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:

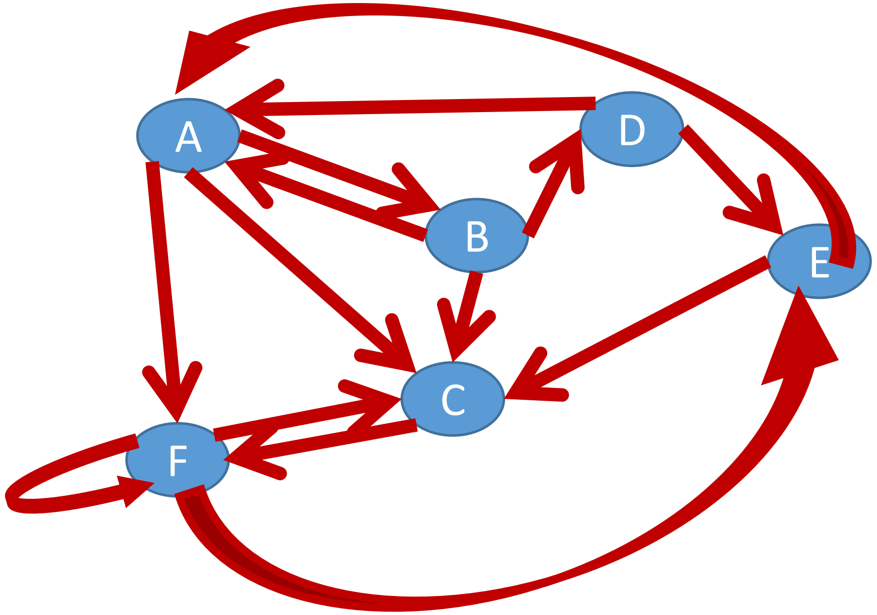
1) If we don’t need to refer to a node, we can use “()”, with no variable.
2) If we don’t need to refer to an edge, we can omit it, e.g.: (a) --> (b) indicates an edge between a and b.
3) If we don’t need to consider the direction of the edge, just use “- -” (without the arrow end)
4) If a mattern matches 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???



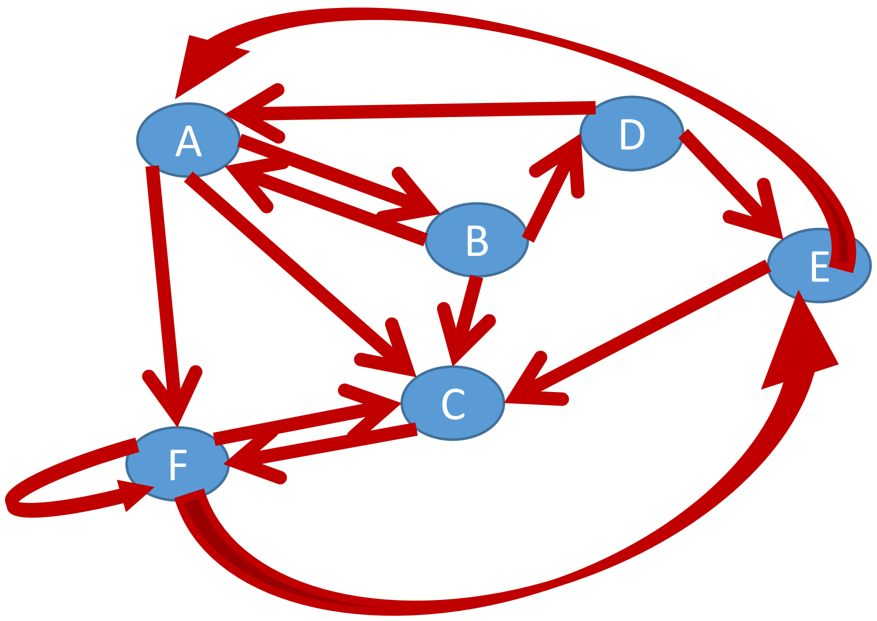
Cypher – Example

The query:

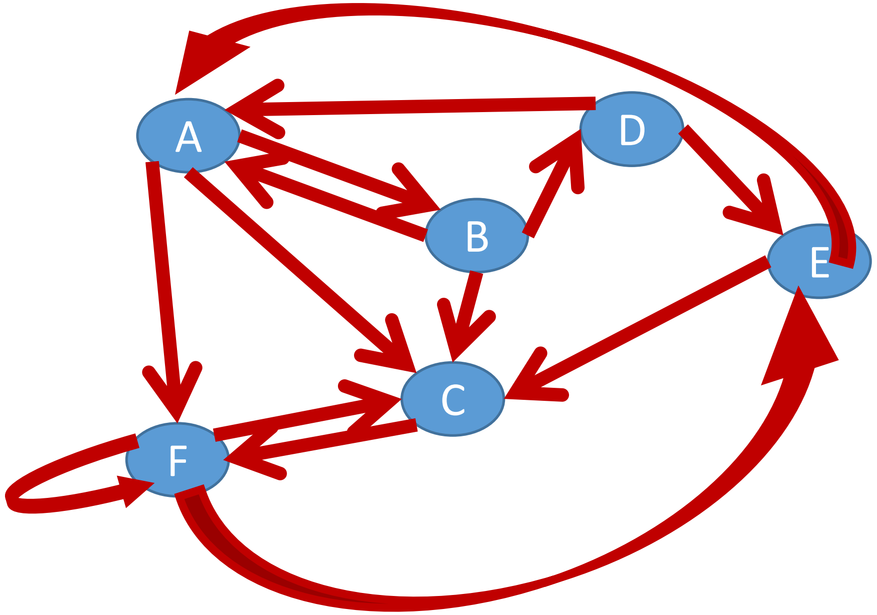
```
$ MATCH (p)-[]->(s)-[]->(x)
  RETURN Count(p), s.URL, Count(x)
```

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

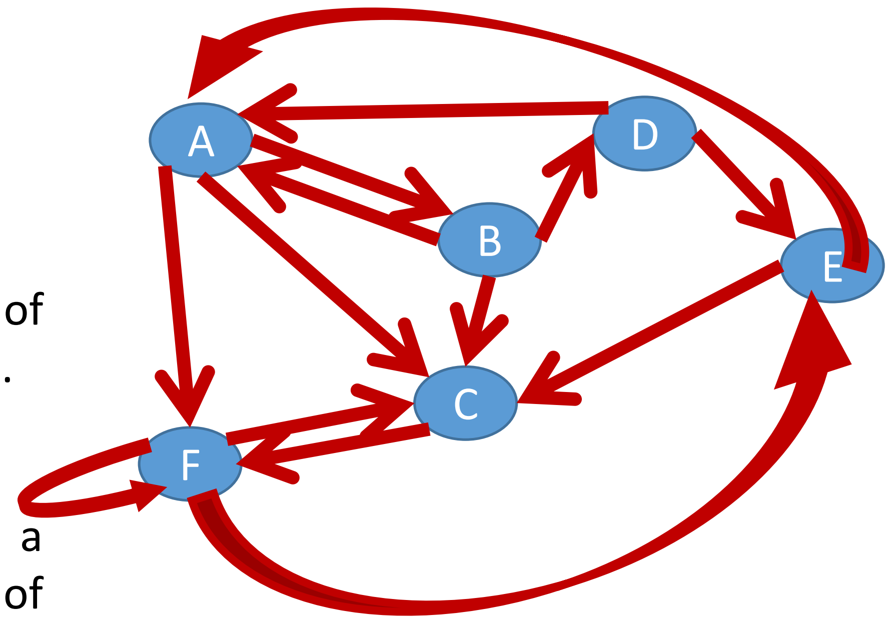


```
$ MATCH (p)-[]->(s)-[]->(x)
  RETURN Count(p), s, Count(x)
```

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 returm 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.

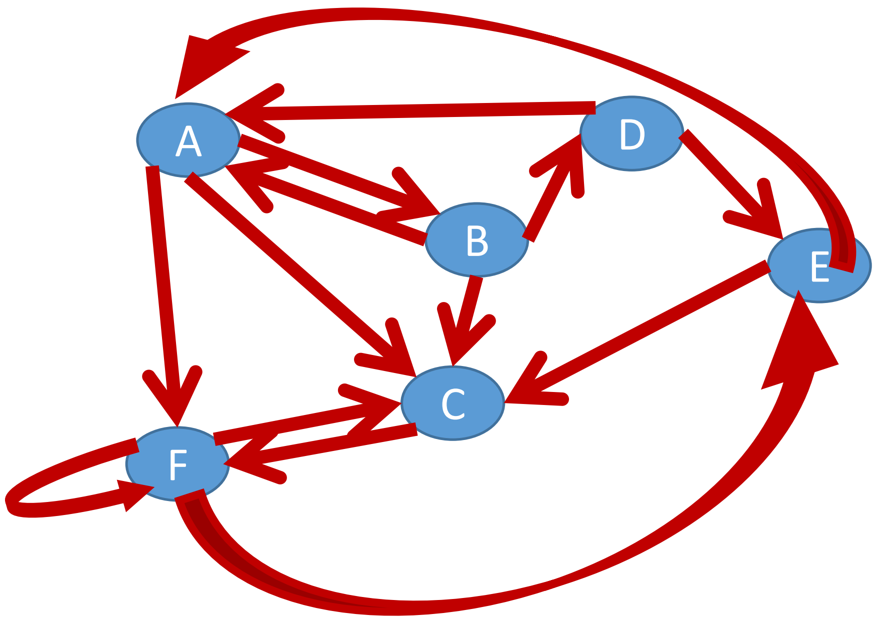
If a page Z references a page X, Z gives X a normalized vote computed as the inverse of the number of pages referenced by Z. To prevent votes of self-referencing pages, if Z references X and X references Z, Z gives 0 votes to X.

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.

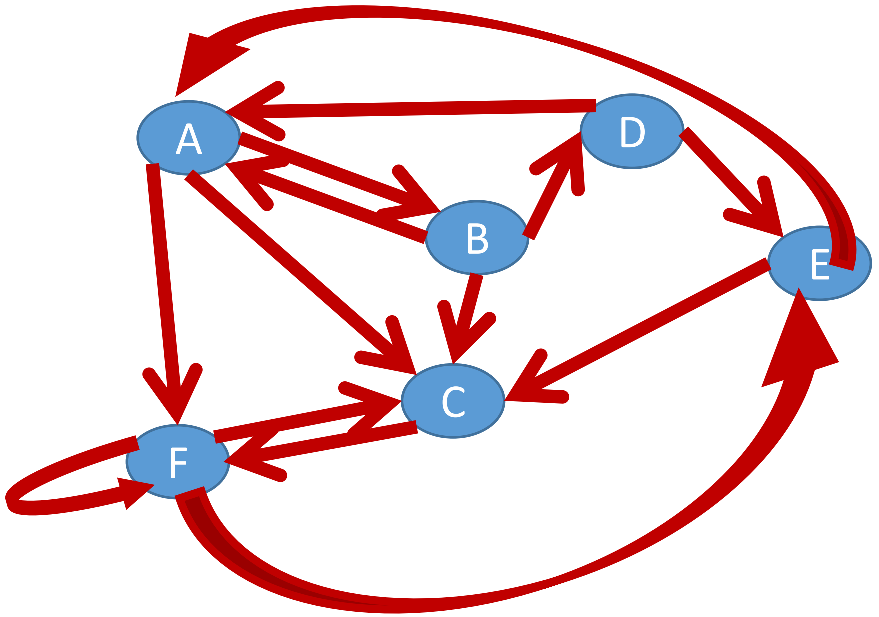
Cypher – Example

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

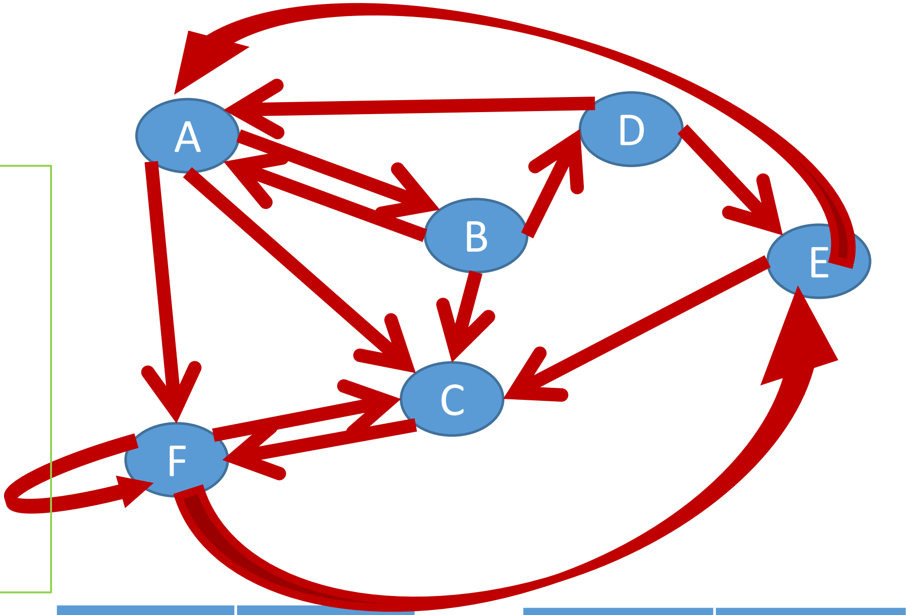


Now, for each of these 6 “p” nodes, look for the paths of length 1 where no reciprocity exists (e.g., delete A ->B and B -> A)

Cypher – Example

Possible solution

```
$ MATCH (p) --> (r)
  WITH  p, 1.0 / count(r) as vote
  MATCH (p) --> (x)
  WHERE NOT ( (x) --> (p) )
  RETURN  x, 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.

Example: Graph Aggregation - OLAP

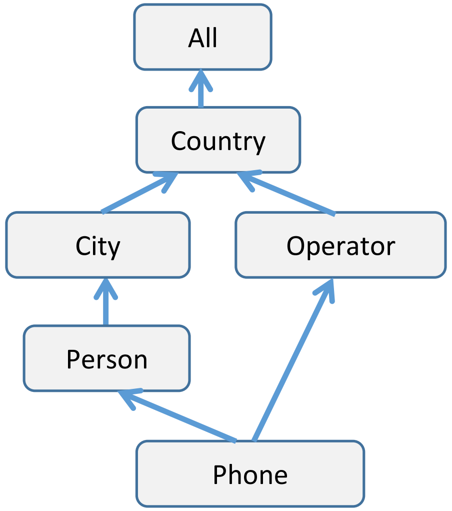
- Lot of work in graph summarization
- Not that much for OLAP
- Graphs can be good for some OLAP cases:
  - When the number of dimensions in a fact is not fixed
  - Eg.: group calls
- Let’s study a typical OLAP example, and implement it on Neo4j

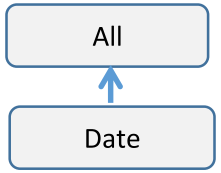
Example

- We have call data in a company
- Geography: Cities and countries, including languages and capital cities
- Operators by country
- Phone numbers by operator
- Persons that registered phones and city of residence. People may have several phones but only one place of residence
- Communication between phones, either sms's (with date and length) or calls (with date and duration) = > **facts**

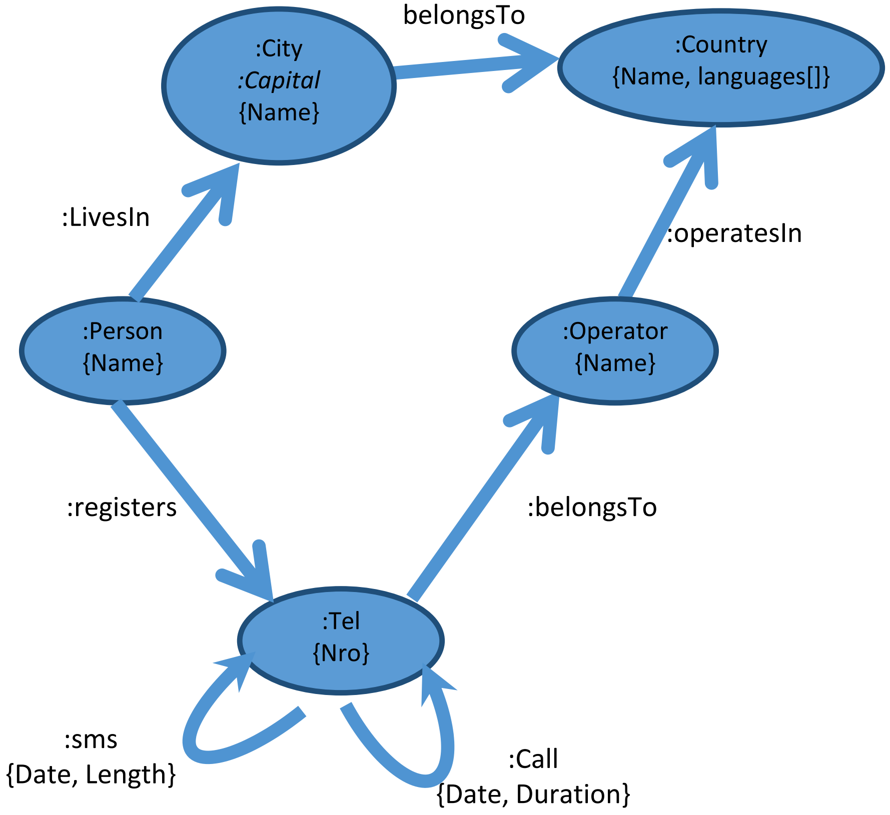
Conceptual model

- 3 dimensions: Caller, Callee, Time
- 2 measures: length(SMS), duration (call)





Logical model in Neo4j (“schema”)



Example 1.

- The following query computes *the average length of the calls corresponding to each Caller-Callee pair.*
- In OLAP this is called a Slice on the Time dimension and on the measure Length, summarizing the remaining measures with the function avg.

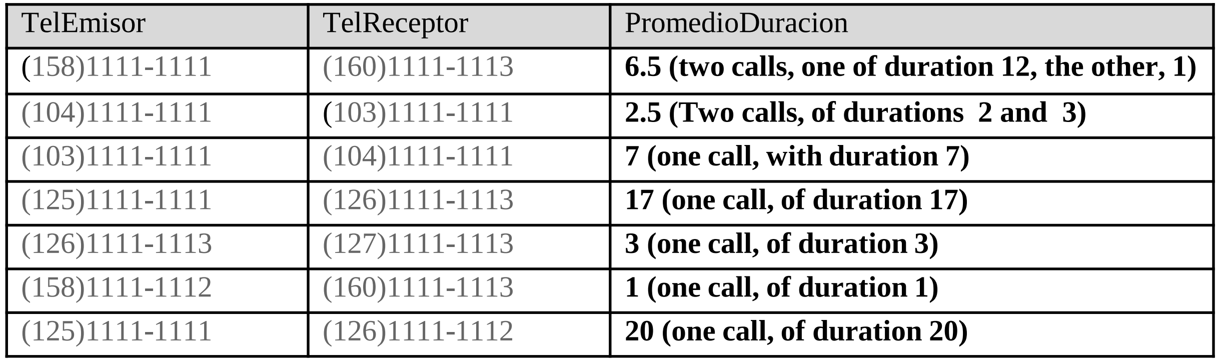
Example 1.

- The following query computes *the average length of the calls corresponding to each Caller-Callee pair.*
- In OLAP this is called a Slice on the Time dimension and on the measure Length, summarizing the remaining measures with the function avg.

```
MATCH (n :Tel) -[r :call]-> (m: Tel)
RETURN n as TelCaller, m as TelCallee, AVG(toFloat(r.Duration)) AS AvgDuration
```

Example 1.

```
MATCH (n :Tel) -[r :call]-> (m: Tel)

RETURN n as TelCaller, m as TelCallee, AVG(toFloat(r.Duration)) AS AvgDuration
```

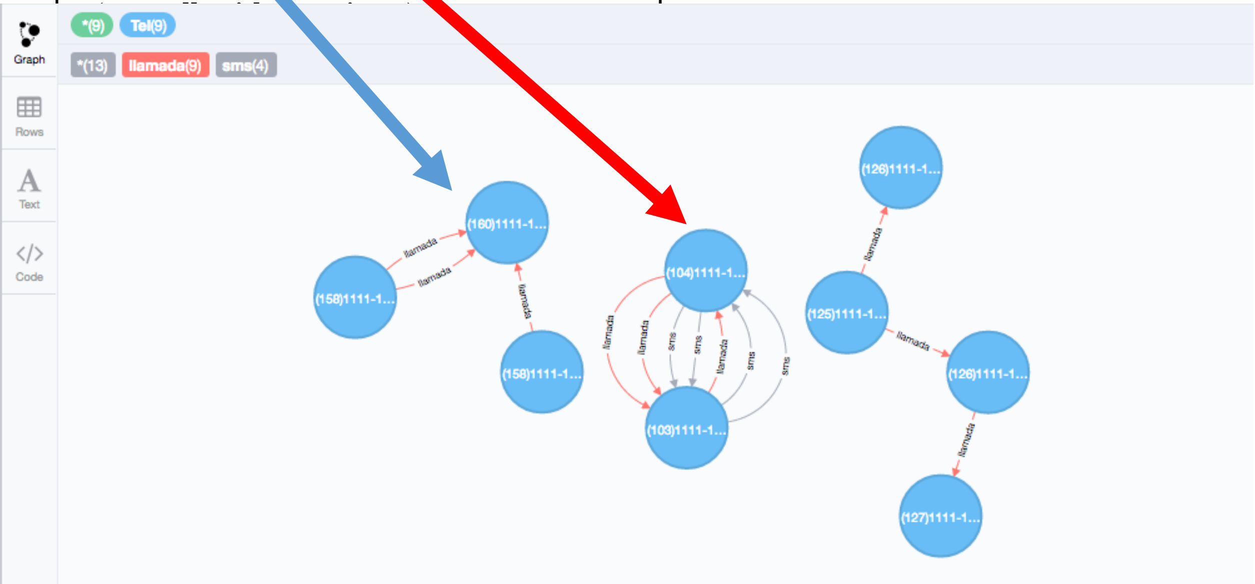


TelEmisor	TelReceptor	PromedioDuracion
(158)1111-1111	(160)1111-1113	**6.5 (two calls, one of duration 12, the other, 1)**
(104)1111-1111	(103)1111-1111	**2.5 (Two calls, of durations 2 and 3)**
(103)1111-1111	(104)1111-1111	**7 (one call, with duration 7)**
(125)1111-1111	(126)1111-1113	**17 (one call, of duration 17)**
(126)1111-1113	(127)1111-1113	**3 (one call, of duration 3)**
(158)1111-1112	(160)1111-1113	**1 (one call, of duration 1)**
(125)1111-1111	(126)1111-1112	**20 (one call, of duration 20)**

Example 1.

```
MATCH (n :Tel) -[r :call]-> (m: Tel)
RETURN n as TelCaller, m as TelCallee, AVG(toFloat(r.Duration)) AS AvgDuration
```

TelEmisor	TelReceptor	PromedioDuracion
(158)1111-1111	(160)1111-1113	**6.5 (two calls, one of duration 12, the other, 1)**
(104)1111-1111	(103)1111-1111	**2.5 (Two calls, of durations 2 and 3)**
(103)1111-1111	(104)1111-1111	
(125)1111-1111	(126)1111-1113	
(126)1111-1113	(127)1111-1113	
(158)1111-1112	(160)1111-1113	
(125)1111-1111	(126)1111-1112	



Example 2.

- Same as before, *but summarizing calls regardless who started them.*

Example 2.

- Same as before, *but summarizing calls regardless who started them.*

```
MATCH (n :Tel) -[r :call]- (m: Tel)
WHERE n.Nro < m.Nro
RETURN n as Tel1, m as Tel2, AVG(toFloat(r.Duration)) As AvgDuration;
```

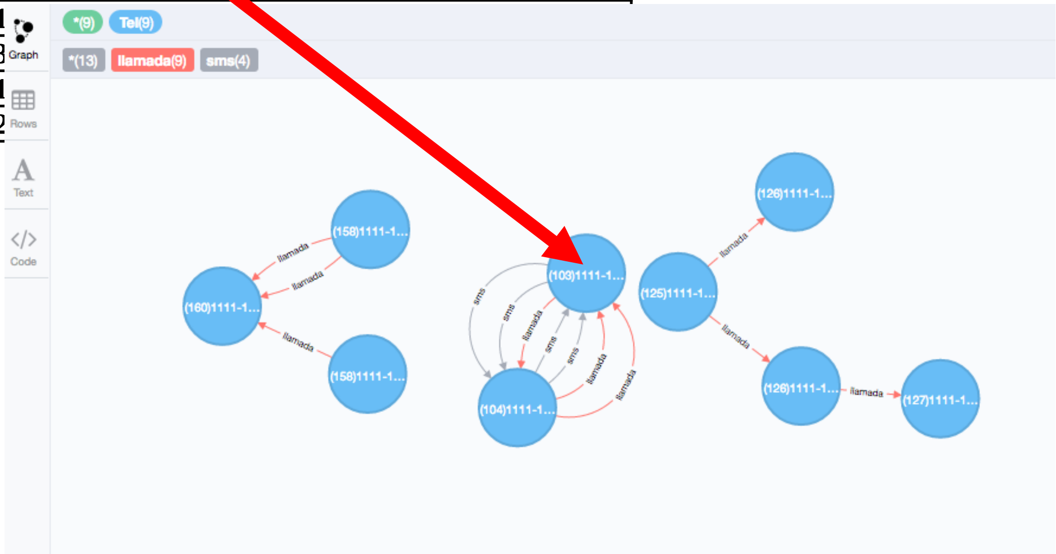
OLAP Operations: SLICE

MATCH (n :Tel) -[r :Call]- (m: Tel)

WHERE n.Nbr < m.Nbr

RETURN n as Tel1, m as Tel2, AVG(toFloat(r.Duration)) As AvgDuration;

Tel1	Tel2	PromedioDuracion
(158)1111-1111	(160)1111-1113	**6.5 (two calls, one of duration 12, the other one 1)**
(103)1111-1111	(104)1111-1111	**4 (three calls summarized, regardless who started the call: 2, 3, 7)**
(125)1111-1111	(126)1111-1113	1
(126)1111-1113	(127)1111-1113	3
(158)1111-1112	(160)1111-1113	1
(125)1111-1111	(126)1111-1112	2



Example 3.

- Same as before, but **rolling up to** Person, either for the caller and the callee. That means, phones belonging to the same person must be summarized. Then, we want the average duration of calls between each pair of persons, regardless who started them.

Example 3.

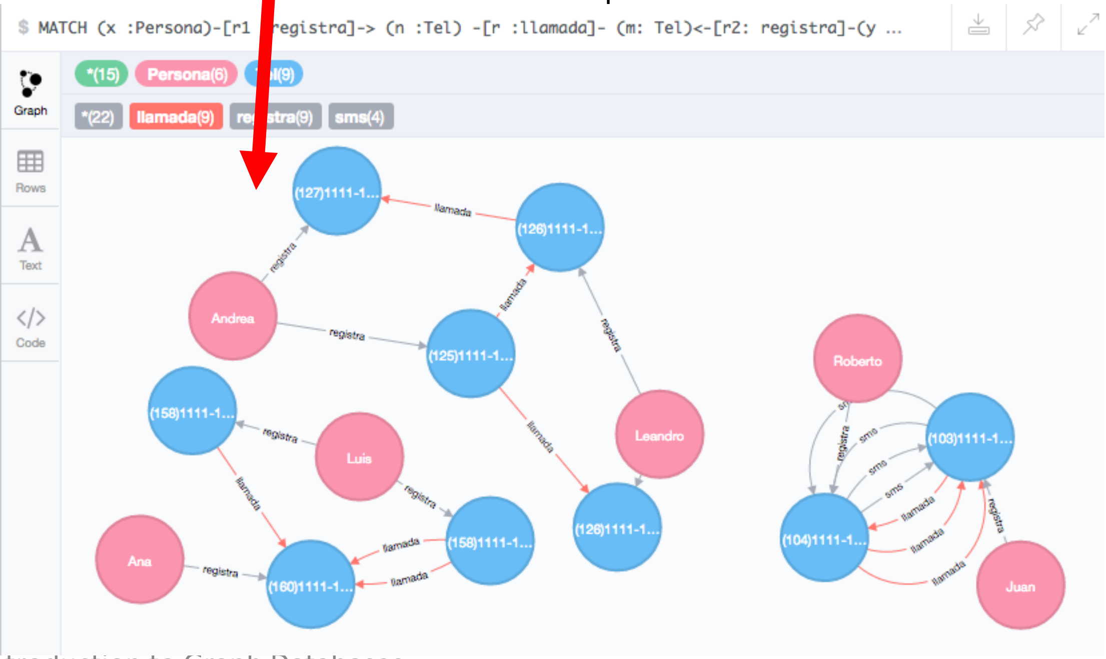
- Same as before, but **rolling up to** Person, either for the caller and the callee. That means, phones belonging to the same person must be summarized. Then, *we want the average duration of calls between each pair of persons, regardless who started them*.

```
MATCH (x :Person)-[r1 :registers]->
(n :Tel) -[r:call]- (m: Tel)<-[r2: registers]-(y :Person)
WHERE x.Name < y.Name
RETURN x as Person1, y as Person2 , AVG(toFloat(r.Duration)) As AvgDuration;
```

Example 3.

Persona1	Persona2	Promedio Duración
Name: Ana (Liverpool) **ID: 315** **Sexo: F**	**Name: Luis (Londres)** **ID: 313** **Sexo: M**	4.666667 (3 calls, with duración 12, 1 & 1, respectively)
Name: Juan (Amberes) **ID: 300** **Sexo: M**	**Name: Roberto (Amberes)** **ID: 301** **Sexo: M**	4 (3 calls, with duración 2, 3 & 7, respectively)
Name: Andrea (Roma) **ID: 307** **Sexo: M**	**Name: Leandro (Roma)** **ID: 308** **Sexo: M**	13.333333 (3 calls, with duración 17, 3 & 20 respectively)

Note: the figure shows the calls, not the average duration



```
$ MATCH (x :Persona)-[r1 registra]-> (n :Tel) -[r :llamada]- (m: Tel)<-[r2: registra]-(y ...
```

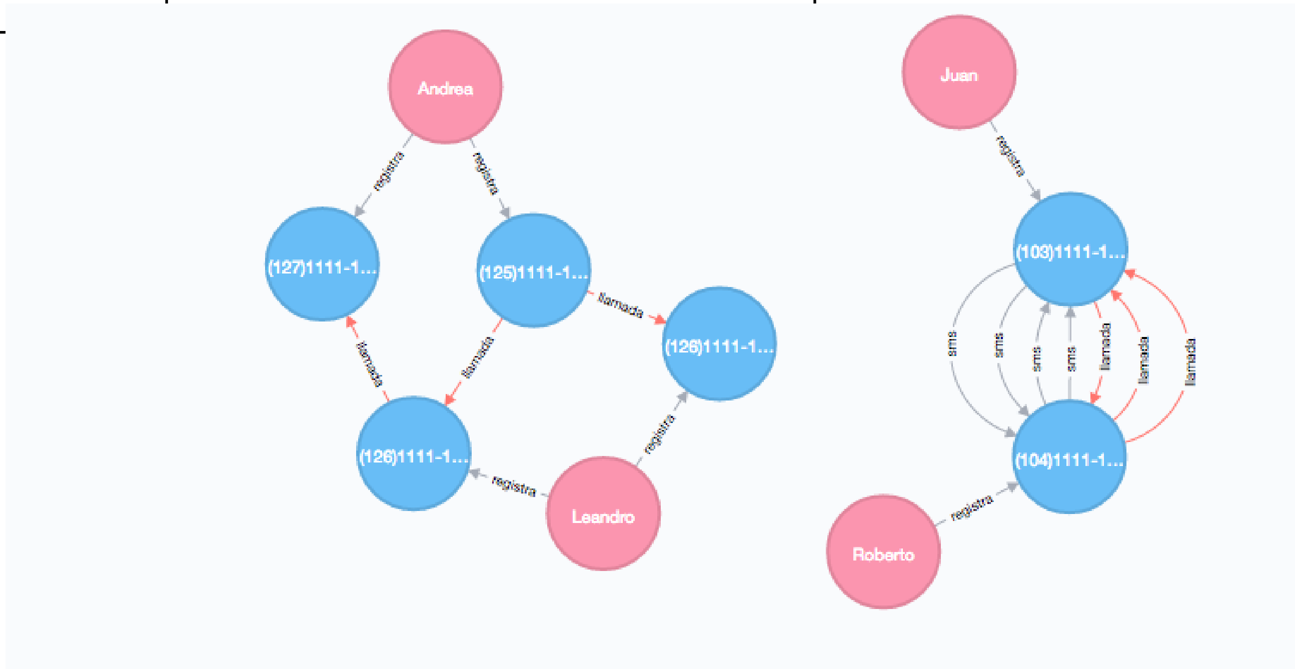
Example 4.

- Same as before, but keeping only the pairs of users of the same gender, F-F o M-M.
- In OLAP jargon, this is called a **Dice**

```
MATCH (x :Person)-[r1 :registers]-> (n :Tel) -[r:Call]- (m: Tel)<-[r2: registers]-(y :Person)

WHERE  x.Name < y.Name  AND x.Gender = y.Gender

RETURN  x as Person1, y as Person2, AVG(toFloat(r.Duration))  as AvgDuration
```

Example 4.

Persona1	Persona2	Promedio Duración
Name: Juan (Amberes) **ID: 300** **Sexo: M**	**Name: Roberto (Amberes)** **ID: 301** **Sexo: M**	4 (3 calls, with durations 2, 3 & 7, respectively)
Name: Andrea (Roma) **ID: 307** **Sexo: M**	**Name: Leandro (Roma)** **ID: 308** **Sexo: M**	13.333333 (3 calls, with durations 17, 3 & 20, respectively)



More examples (coalesce)

- Same temporal SLICE with a Rollup to Person, regardless who initiated the call or sent the SMS but:
  - For each pairs of persons who only exchanged calls or only exchanged SMSs, the value for the missing measure should be set to “0”. If there is a pair of persons who did not communicate at all, the pair is not displayed. Consider that a person can send a self-message.

```
MATCH (x:Person)-[:registers]->(n:Tel)-[r1]-(m:Tel)<-[:registers] -(y:Person)
WHERE x.Name < y.Name
RETURN x as Person1,y as Person2, COALESCE(Avg(toFloat(r1.Duration)),0) as AvgDuration,
COALESCE(Avg(toFloat(r1.Length)),0) as AvgLength
ORDER BY x.Name
```

More examples (coalesce)

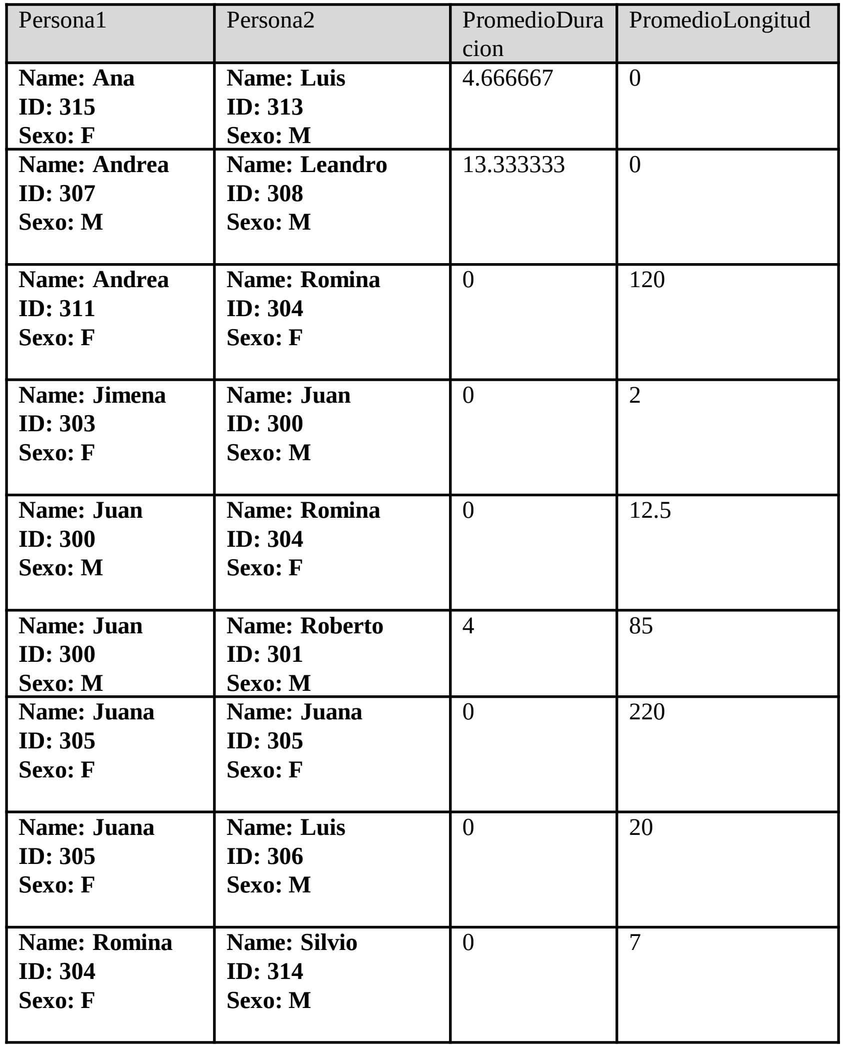
```
MATCH (x:Person)-[:registers]->(n:Tel)-[r1]-(m:Tel)<-[:registers] -(y:Person)
WHERE x.Name <=y.Name
RETURN x as Person1,y as Person2, COALESCE(Avg(toFloat(r1.Duration)),0) as AvgDuration,
COALESCE(Avg(toFloat(r1.Length)),0) as AvgLength
ORDER BY x.Name
```

Alternative Solution

```
MATCH (x:Person)-[:registers]->(n:Tel)-[r1]-(m:Tel)<-[:registers] -(y:Person)
WHERE x.Name <=y.Name
RETURN x as Person1,y as Person2, CASE WHEN Avg(toFloat(r1.Duration)) IS NULL THEN 0 ELSE Avg(to
Float(r1.Duration))  END AS AvgDuration,
CASE WHEN Avg(toFloat(r1.Length)) IS NULL THEN 0 ELSE Avg(toFloat(r1.Length)) END  AS AvgLength
ORDER BY x.Name
```

More examples (coalesce)

```
MATCH (x:Person)-[:registers]->(n:Tel)-[r1]-(m:Tel)<-
[:registers] -(y:Person)
WHERE x.Name <=y.Name
RETURN x as Person1,y as Person2,
COALESCE(Avg(toFloat(r1.Duration)),0) as AvgDuration,
COALESCE(Avg(toFloat(r1.Length)),0) as AvgLength
ORDER BY x.Name
```



Persona1	Persona2	PromedioDuracion	PromedioLongitud
Name: Ana **ID: 315** **Sexo: F**	**Name: Luis** **ID: 313** **Sexo: M**	4.666667	0
Name: Andrea **ID: 307** **Sexo: M**	**Name: Leandro** **ID: 308** **Sexo: M**	13.333333	0
Name: Andrea **ID: 311** **Sexo: F**	**Name: Romina** **ID: 304** **Sexo: F**	0	120
Name: Jimena **ID: 303** **Sexo: F**	**Name: Juan** **ID: 300** **Sexo: M**	0	2
Name: Juan **ID: 300** **Sexo: M**	**Name: Romina** **ID: 304** **Sexo: F**	0	12.5
Name: Juan **ID: 300** **Sexo: M**	**Name: Roberto** **ID: 301** **Sexo: M**	4	85
Name: Juana **ID: 305** **Sexo: F**	**Name: Juana** **ID: 305** **Sexo: F**	0	220
Name: Juana **ID: 305** **Sexo: F**	**Name: Luis** **ID: 306** **Sexo: M**	0	20
Name: Romina **ID: 304** **Sexo: F**	**Name: Silvio** **ID: 314** **Sexo: M**	0	7