Técnicas para Big Data

Clase 07 - Neo4J

Neo4J

Neo4J es la base de datos de grafos más utilizada en la industria

Utiliza el modelo de **property graphs**

El lenguaje de consultas asociado es **Cypher**

Modelo Property Graph

Modelo Property Graph

- Grafo

Modelo Property Graph

- Grafo
- Entidades (nodos)

Modelo Property Graph

- Grafo
- Entidades (nodos)
- Arcos (relaciones)

Modelo Property Graph

- Grafo
- Entidades (nodos)
- Arcos (relaciones)
- Nodos y arcos

Modelo Property Graph

- Grafo
- Entidades (nodos)
- Arcos (relaciones)
- Nodos y arcos
- Nodos y arcos tienen etiqueta

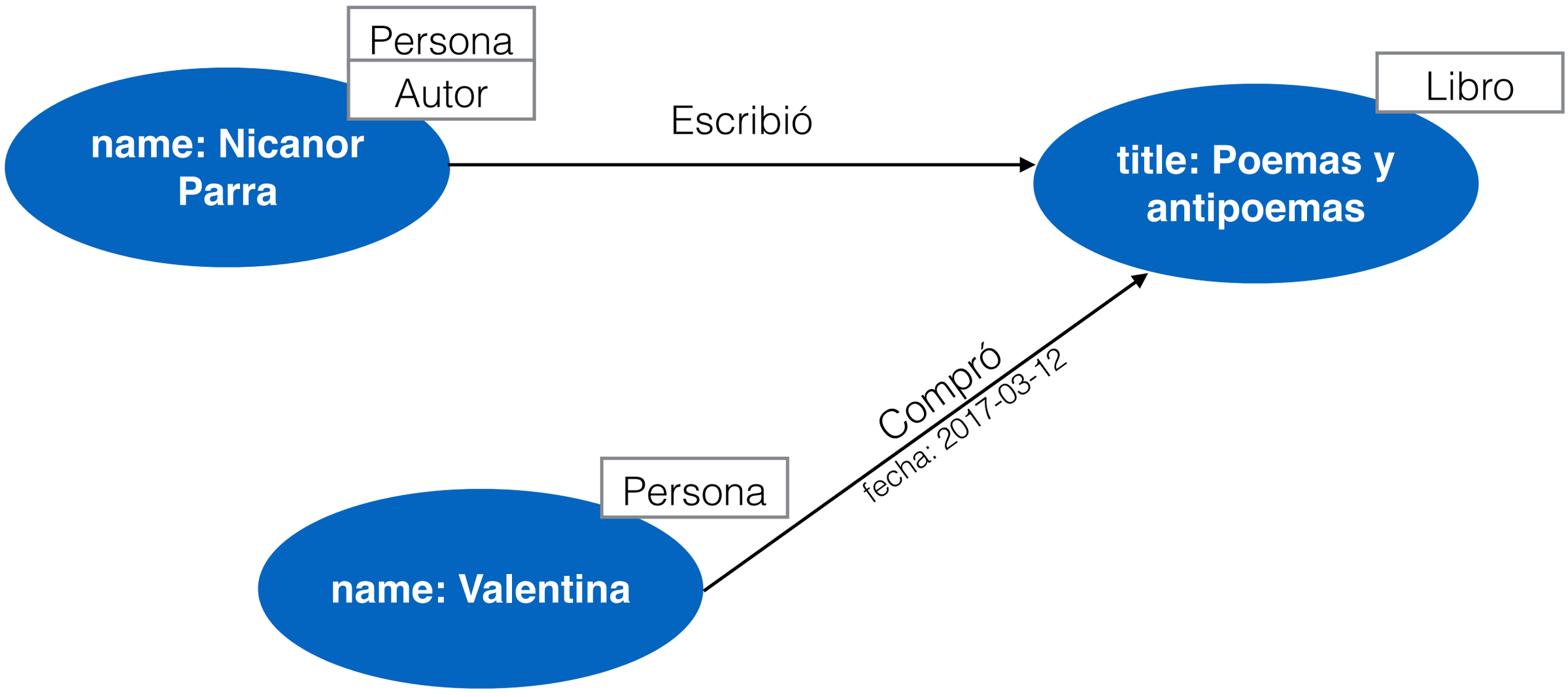
Modelo Property Graph

- Grafo
- Entidades (nodos)
- Arcos (relaciones)
- Nodos y arcos
- Nodos y arcos tienen etiqueta
- Nodos y arcos tienen atributos

Modelo Property Graph

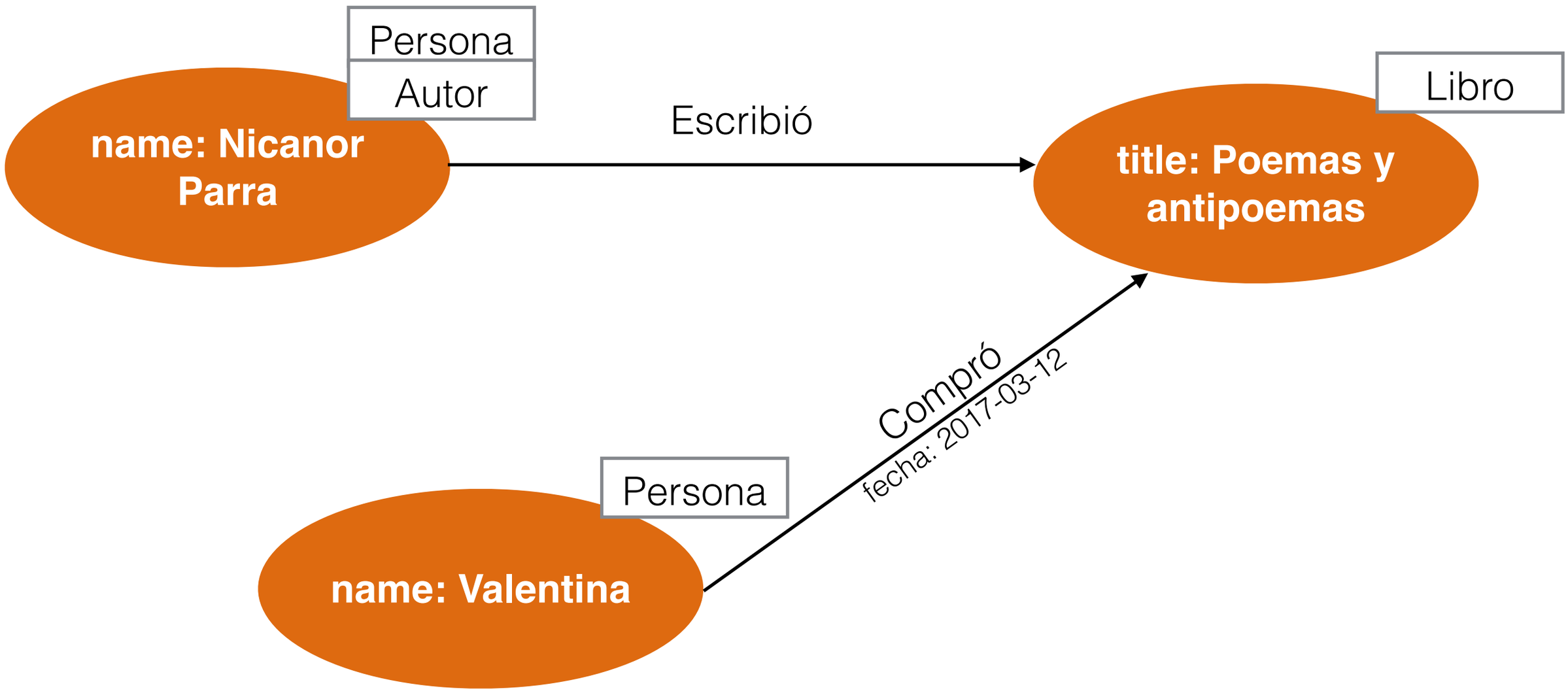
- Grafo
- Entidades (nodos)
- Arcos (relaciones)
- Nodos y arcos
- Nodos y arcos tienen etiqueta
- Nodos y arcos tienen atributos
- ¡No pueden haber links rotos!

Modelo Property Graph



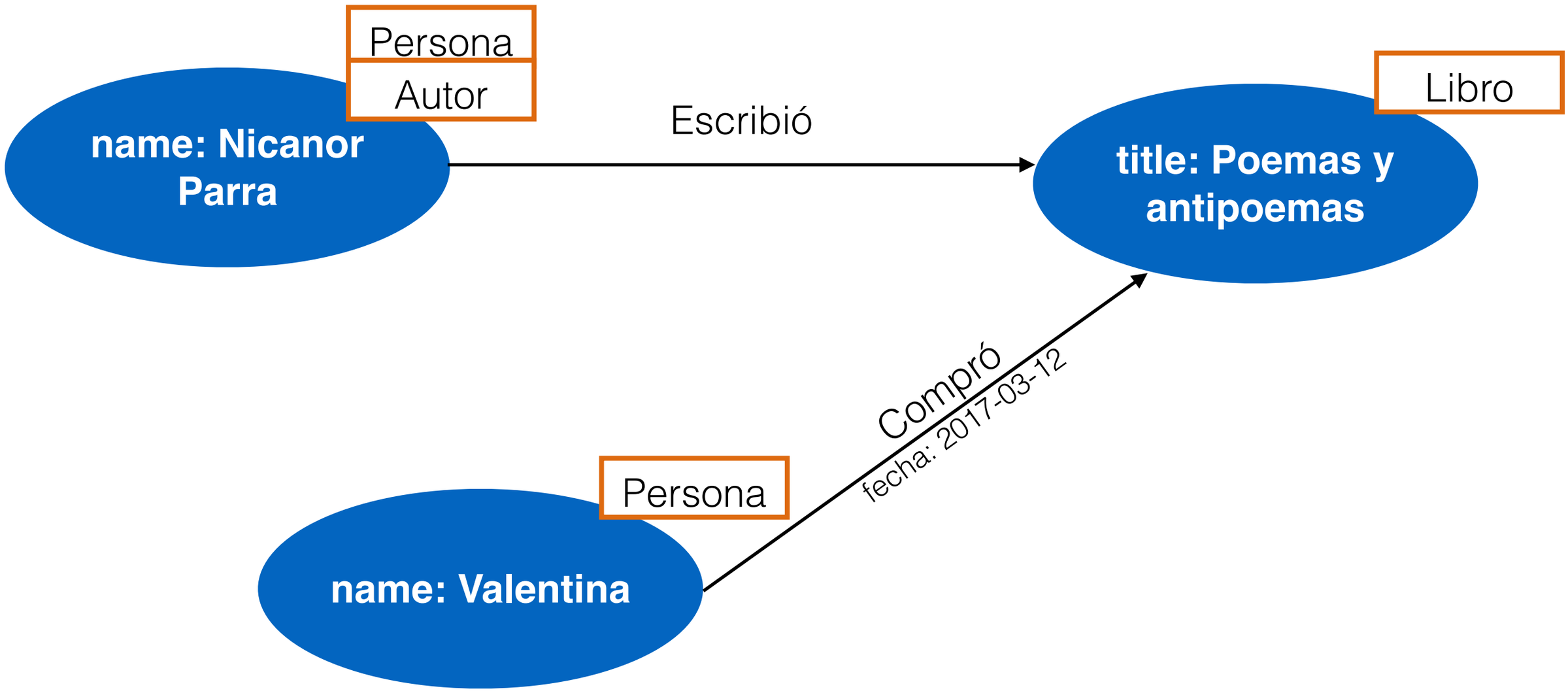
Modelo Property Graph

Nodos



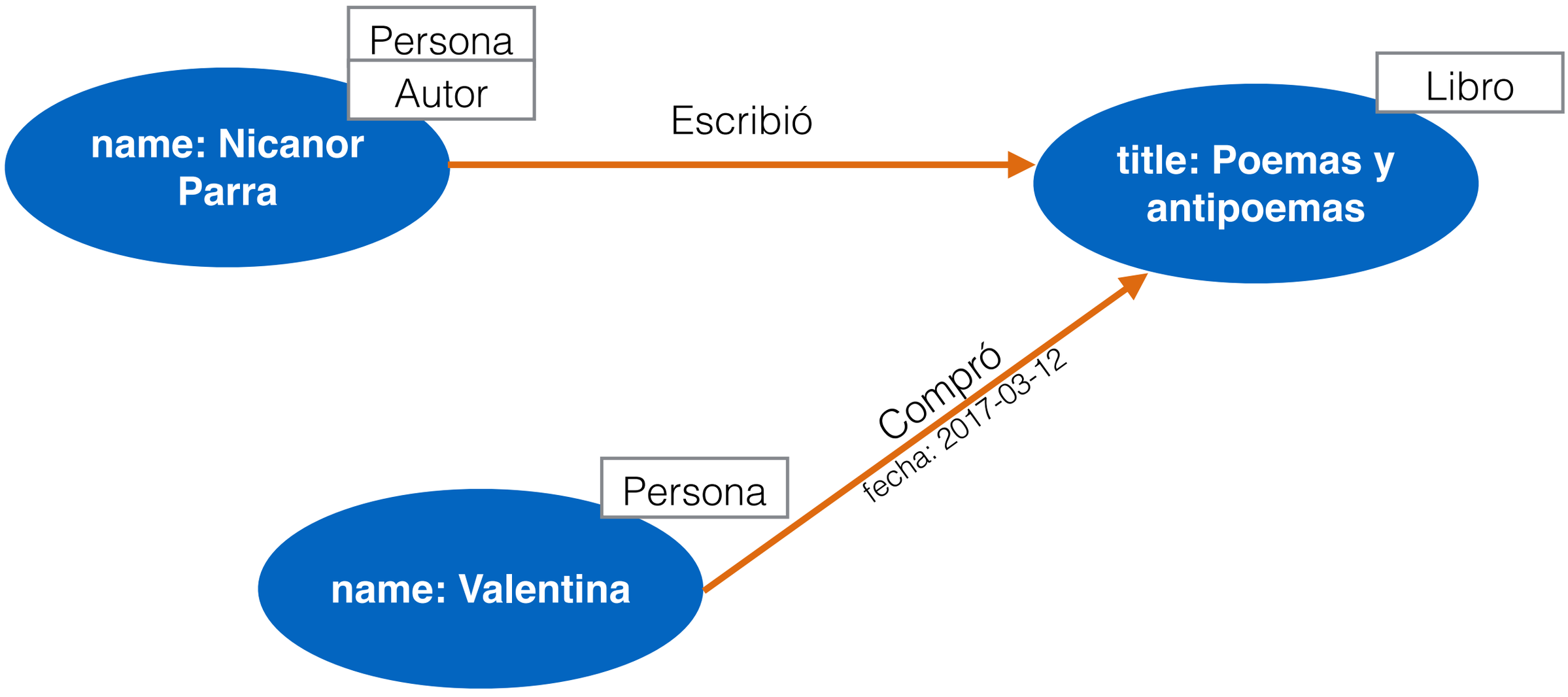
Modelo Property Graph

Tipos de nodos



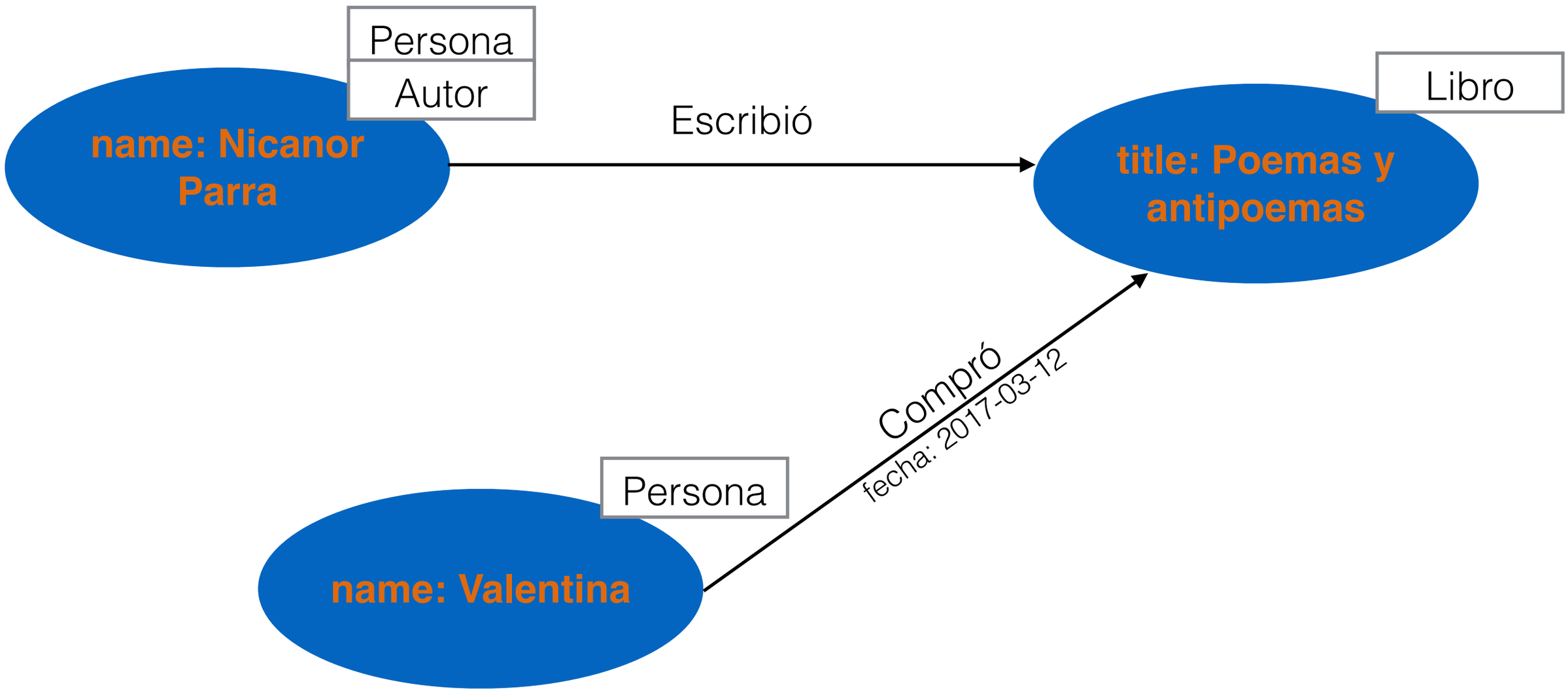
Modelo Property Graph

Relaciones

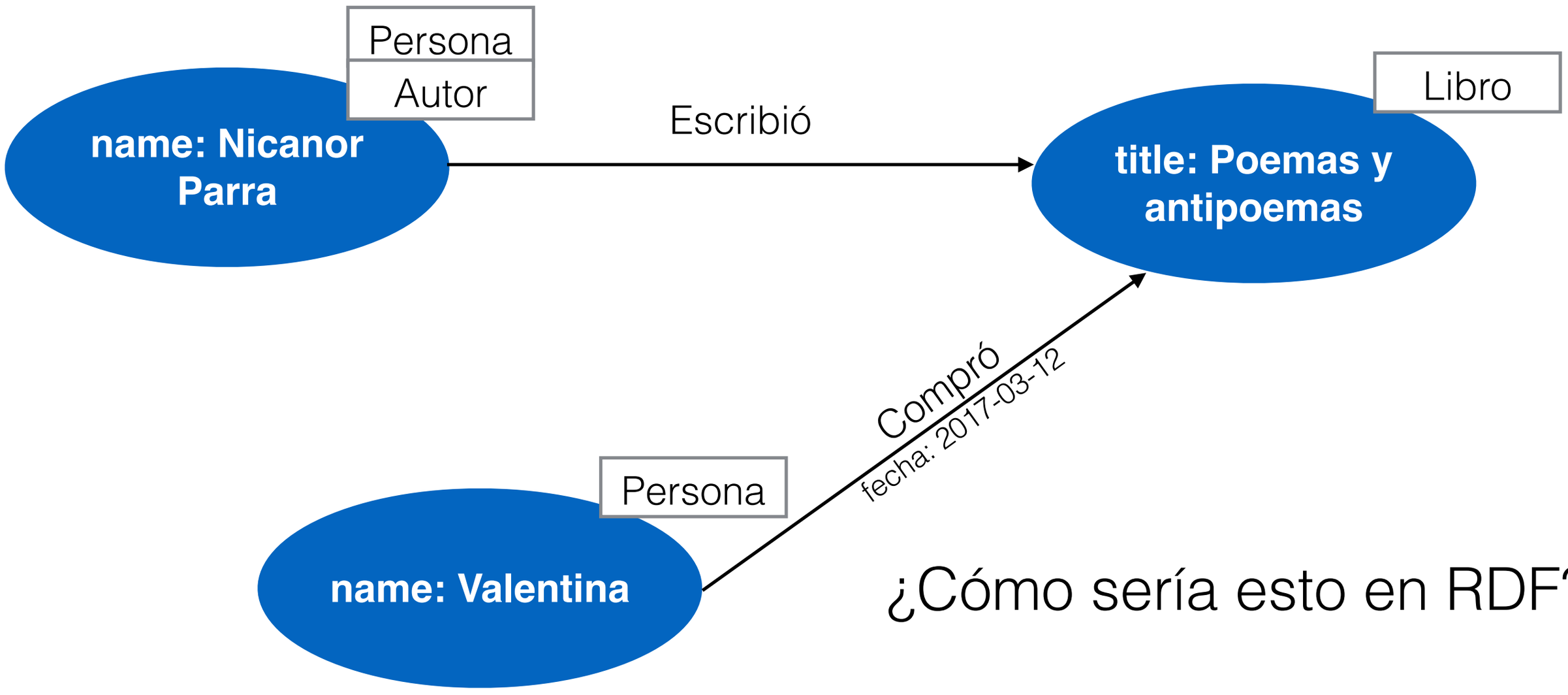


Modelo Property Graph

Atributos



Modelo Property Graph



¿Cómo sería esto en RDF?

RDF vs Property Graph

RDF

- Reglas (OWL, RDFs)
  - "Todo perro es un animal", "El papá de mi papá es mi ancestro"
- Hechos (S-P-O)
- Estándar W3C
- Encontrar relaciones eficientemente

Property Graph

- Nodos y arcos
- Nodos y arcos pueden tener propiedades
- Recorrer el grafo eficientemente

Cypher

Cypher

Cypher es el lenguaje de consulta de Neo4J

Es un lenguaje basado en patrones (como SPARQL), pero añade algunas funcionalidades

Además, se pueden realizar consultas por las propiedades de los nodos y aristas

Consultas

```
MATCH (node1:Label1)--> (node2:Label2)
WHERE node1.propertyA = {value}
RETURN node2.propertyA, node2.propertyB
```

Nodos

- ( )
- (a)
- (:Persona)
- (a: Persona)

Patrones


```
(a) -[:COMPRA]-> (b)
```

Relaciones

- Las relaciones se representan con una flecha
- –––>

Relaciones

- Las relaciones se representan con una flecha
- `–[:ETIQUETA]–>`

Relaciones

- Las relaciones se representan con una flecha
- `–[:ETIQUETA]–>`
- `–[:ETIQUETA1 | :ETIQUETA2]–>`

Relaciones

- Las relaciones se representan con una flecha
- `–[:ETIQUETA]->`
- `–[:ETIQUETA1 | :ETIQUETA2]->`
- `–[{since:2010}]->`

Relaciones

- Las relaciones se representan con una flecha
- `–[:ETIQUETA]->`
- `–[:ETIQUETA1 | :ETIQUETA2]->`
- `–[{since:2010}]->`
- `–[:KNOWS*..4]->`

Relaciones

Relaciones

- Para acceder a información sobre la relación se puede guardar su valor en una variable

Relaciones

- Para acceder a información sobre la relación se puede guardar su valor en una variable

- `-[rel:ETIQUETA]->`

Relaciones

- Para acceder a información sobre la relación se puede guardar su valor en una variable

- `-[rel:ETIQUETA]->`

- `-[rel]->`

Relaciones

Relaciones

- `() --> ()`

Relaciones

- `() --> ()`
- `(a) --> (b)`

Relaciones

- `() --> ()`
- `(a) --> (b)`
- `(a) -[:COMPRA]-> (b)`

Relaciones

- `() --> ()`
- `(a) --> (b)`
- `(a) -[:COMPRA]-> (b)`
- `(a:PERSONA) -[:COMPRA]-> (b:LIBRO)`

Relaciones

- `() --> ()`
- `(a) --> (b)`
- `(a) -[:COMPRA]-> (b)`
- `(a:PERSONA) -[:COMPRA]-> (b:LIBRO)`
- `(a:PERSONA) -[r:COMPRA]-> (b:LIBRO)`

Relaciones

- `() --> ()`
- `(a) --> (b)`
- `(a) -[:COMPRA]-> (b)`
- `(a:PERSONA) -[:COMPRA]-> (b:LIBRO)`
- `(a:PERSONA) -[r:COMPRA]-> (b:LIBRO)`
- `MATCH (a)-[:HERMANO]-(b)`

Relaciones

```
MATCH (node:Label) RETURN node.property
```

```
MATCH (node1:Label1)--> (node2:Label2)
WHERE node1.propertyA = {value}
RETURN node2.propertyA, node2.propertyB
```

Patrones

Patrones

- Friend-of-a-friend:
  `(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)`

Patrones

- Friend-of-a-friend:
  `(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)`

- Shortest path:

```
path = shortestPath( (user)-
[:KNOWS*..5]-(other) )
```

Patrones

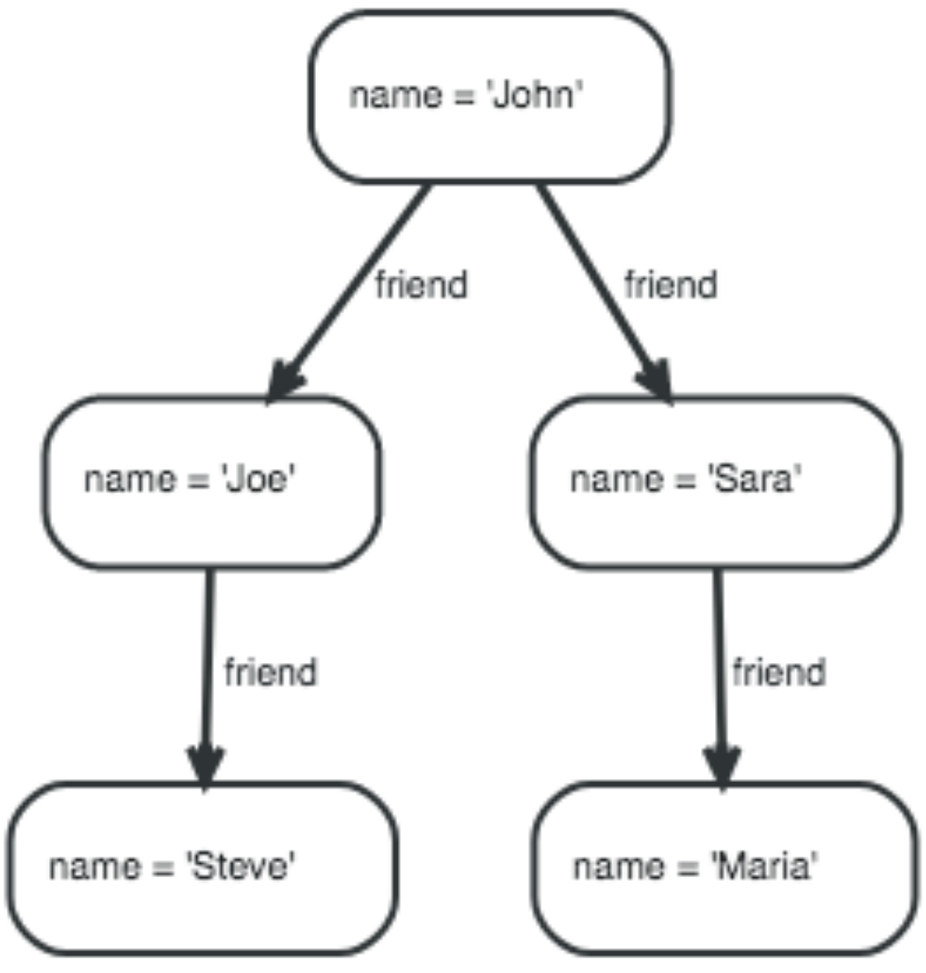
- Friend-of-a-friend:
  `(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)`

- Shortest path:
  ```
  path = shortestPath( (user)-
  [:KNOWS*..5]-(other) )
  ```

- Collaborative filtering
  ```
  (user)-[:PURCHASED]->(product)<-
  [:PURCHASED]-()-[:PURCHASED]-
  >(otherProduct)
  ```

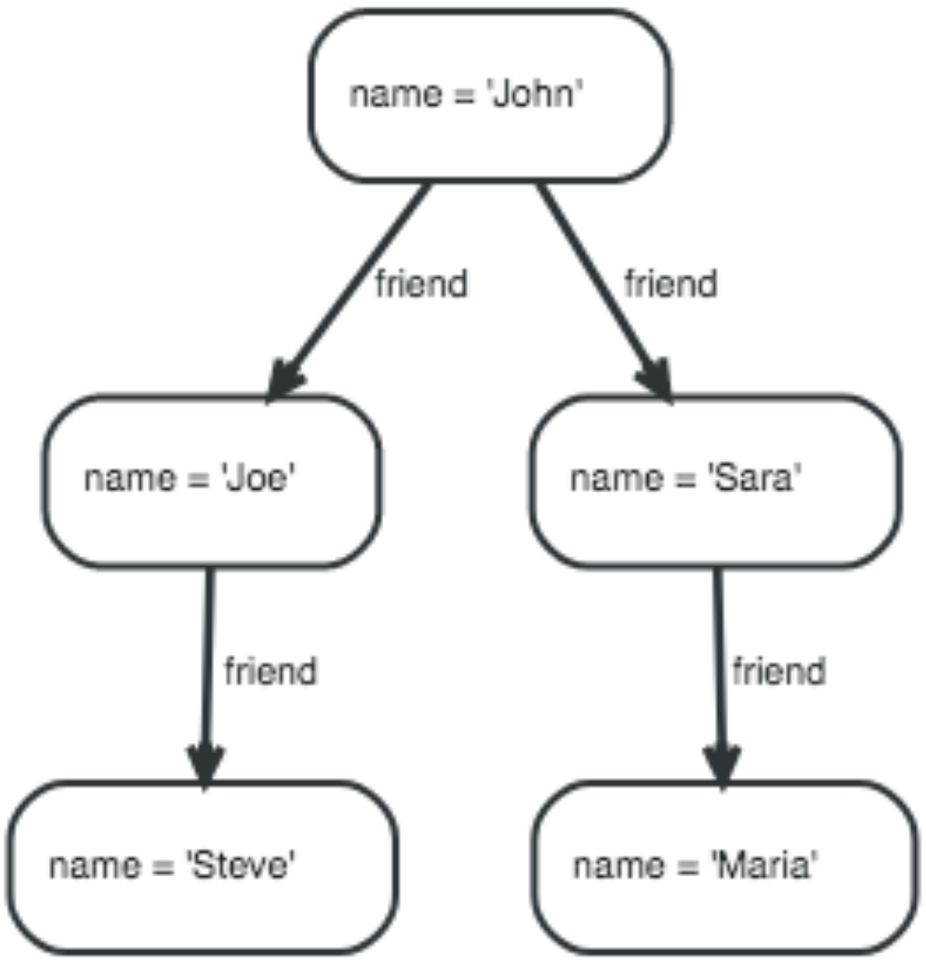
Evaluando Consultas

```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john.name, fof.name
```



Evaluando Consultas

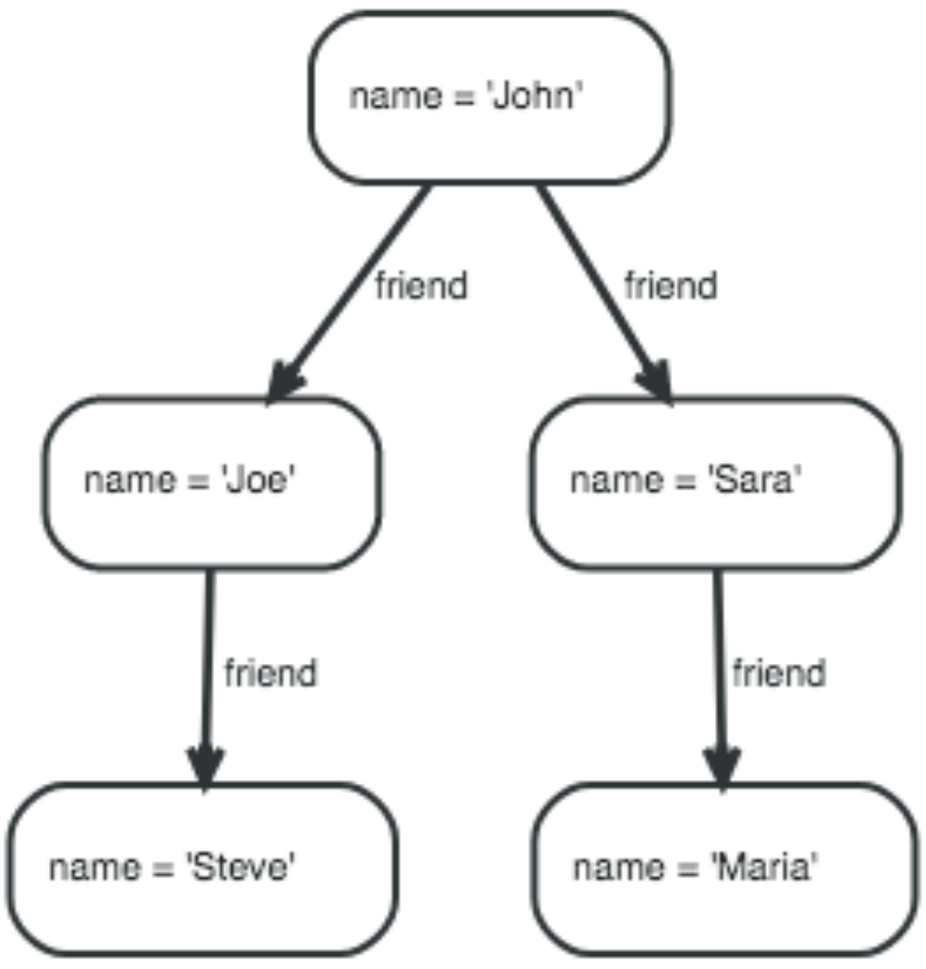
```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john.name, fof.name
```



john.name	fof.name
"John"	"Maria"
"John"	"Steve"

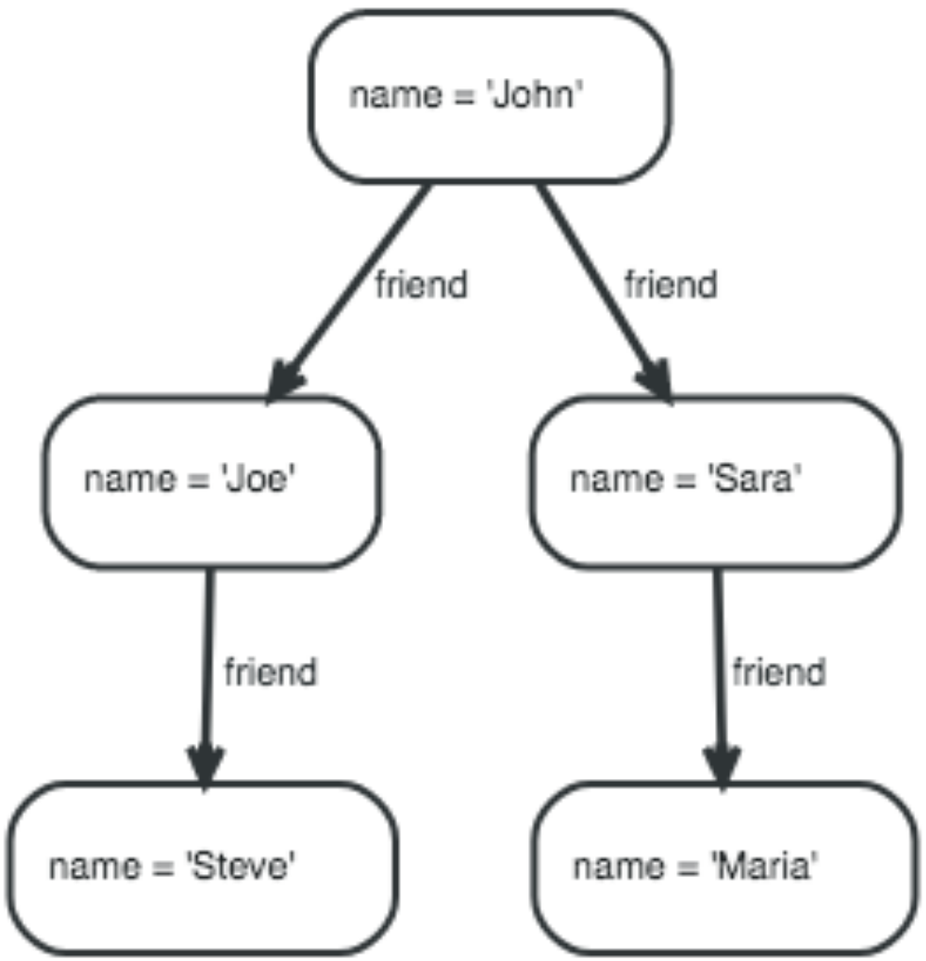
Evaluando Consultas

```
MATCH (user)-[:friend]->(follower)
WHERE user.name IN ['Joe', 'John',
   'Sara', 'Maria', 'Steve'] AND follower.name =~ 'S.*'
RETURN user.name, follower.name
```



Evaluando Consultas

```
MATCH (user)-[:friend]->(follower)
WHERE user.name IN ['Joe', 'John',
    'Sara', 'Maria', 'Steve'] AND follower.name =~ 'S.*'
RETURN user.name, follower.name
```



user.name	follower.name
"Joe"	"Steve"
"John"	"Sarah"

Múltiples patrones

```
MATCH (user:Person) WITH user LIMIT 20
MATCH (user)-[:friend]->(follower)
WHERE friend.name IN ['Joe', 'John', 'Sara', 'Maria', 'Steve']
RETURN user.name, follower.name
```

Patrones opcionales

```
MATCH (user:Person)
OPTIONAL MATCH (user)-[:friend]->(follower)
RETURN user.name, follower.name
```

Creando nodos

```
CREATE (nodo:Person {name:"Miguel Romero"})
RETURN nodo
```

Creando nodos

```
CREATE (nodo:Person {name:"Miguel Romero"})
RETURN nodo
```

```
MATCH  (nodo:Person {name:"Miguel Romero"})
CREATE (nodo)-[trabajo:TRABAJA_EN]->(univ:Universidad {name:"UAI" })
RETURN nodo
```

Creando muchos nodos

```
MATCH (nodo:Person {name:"Miguel Romero"})
FOREACH (nombre in ["Juan","Cristian","Adrian","Andres"] |
  CREATE (nodo)-[:FRIEND]->(:Person {name:nombre}))
```

Eliminando nodos

```
MATCH (nodo:Person {name:"Miguel Romero"})
DELETE nodo
```

Eliminando nodos

```
MATCH (nodo:Person {name:"Miguel Romero"})
DETACH DELETE nodo
```

Python

Existe un driver para trabajar con Neo4J y Python

Puedes encontrar más detalles en https://neo4j.com/developer/python/

Técnicas para Big Data

Clase 07 - Neo4J