NoSQL - Neo4j

Rafael Garrote Hernández
Profesor en NoSQL

Grafo de propiedades etiquetado

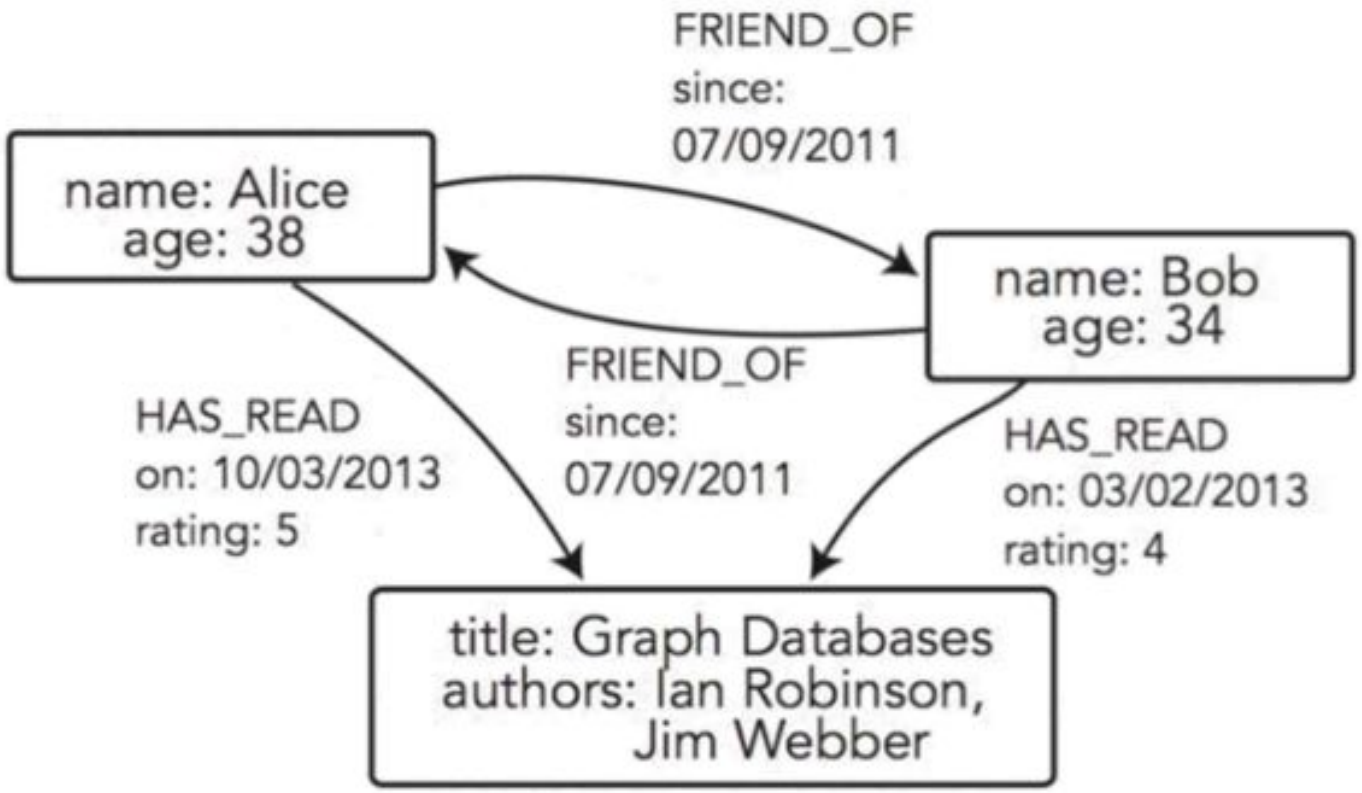
- Representan la información como si se tratase de un grafo.
- Son de propósito general como las bases de datos relacionales.
- La información viene representada por nodos y las relaciones entre los datos por aristas.
- Se utiliza la teoría de grafos para recorrer la base de datos y así procesar y gestionar la información.

Grafo de propiedades etiquetado

- El rendimiento no se deteriora con el crecimiento de los datos ni con la consulta o procesamiento intensivo de los datos.
- El rendimiento es proporcional al tamaño y sólo se centra en la parte del árbol que está implicada en la operación y no en todo el grafo.
- Permiten un mantenimiento progresivo y ágil de los sistemas a medida que la aplicación va creciendo y se puede ir adaptando a los cambios del negocio.

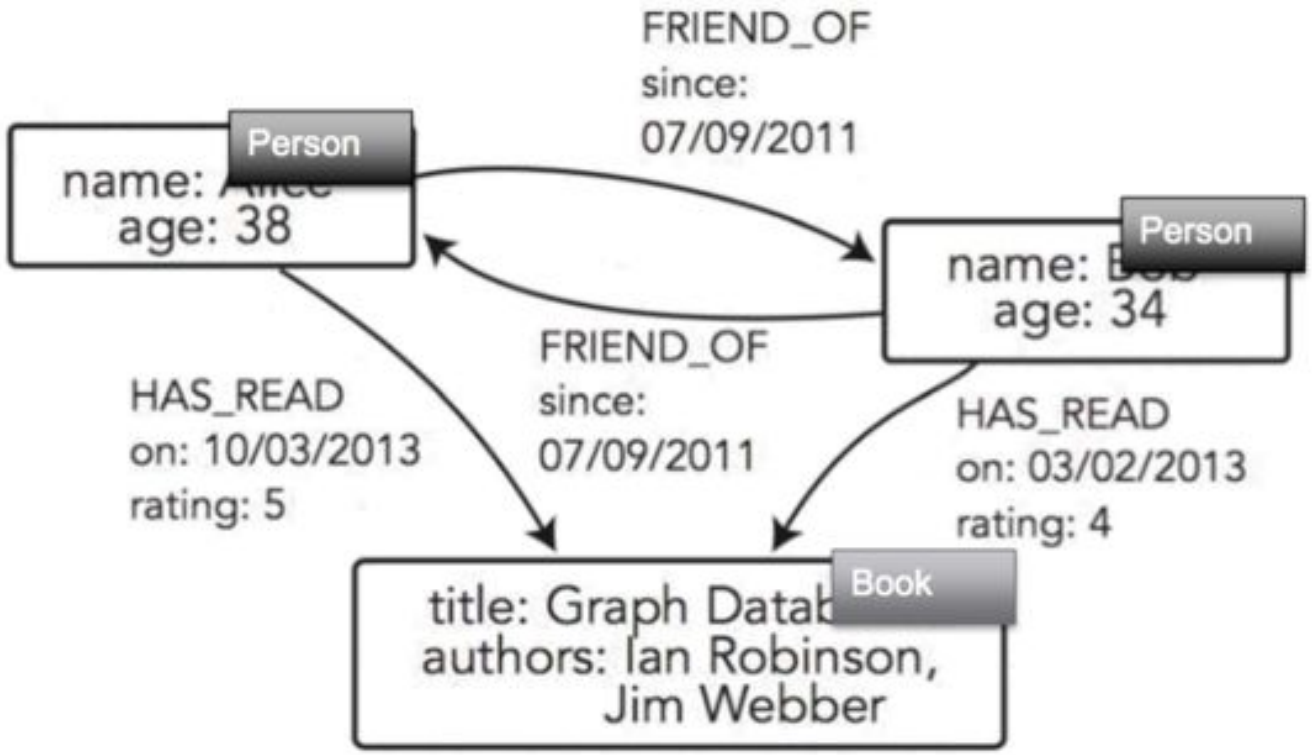
Building blocks

Nodos
Relaciones
Propiedades



Building blocks

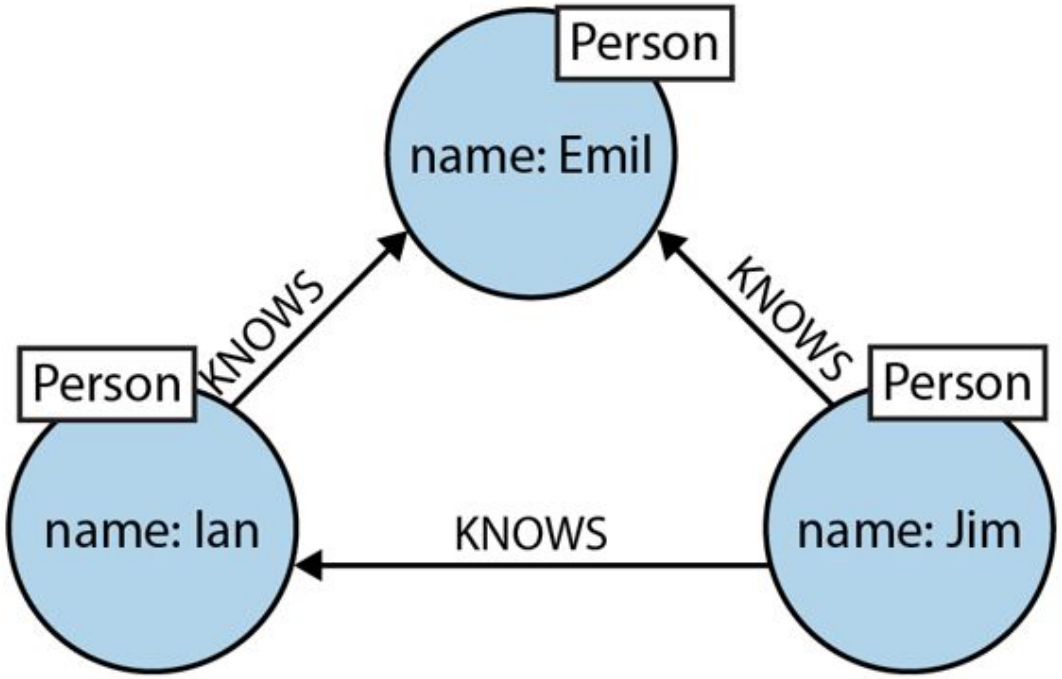
Etiquetas



Building blocks

- Contienen ***nodos*** y ***relaciones***.
- Los nodos tienen ***propiedades*** (pares clave/valor).
- Los nodos se pueden etiquetar con una o más ***etiquetas***. Las etiquetas representan el rol del nodo en el grafo.
- Los nodos se organizan en **relaciones**.
- Las relaciones tienen nombre y son ***dirigidas***, siempre tienen un nodo de inicio y un nodo final.
- Las relaciones también tienen **propiedades**.

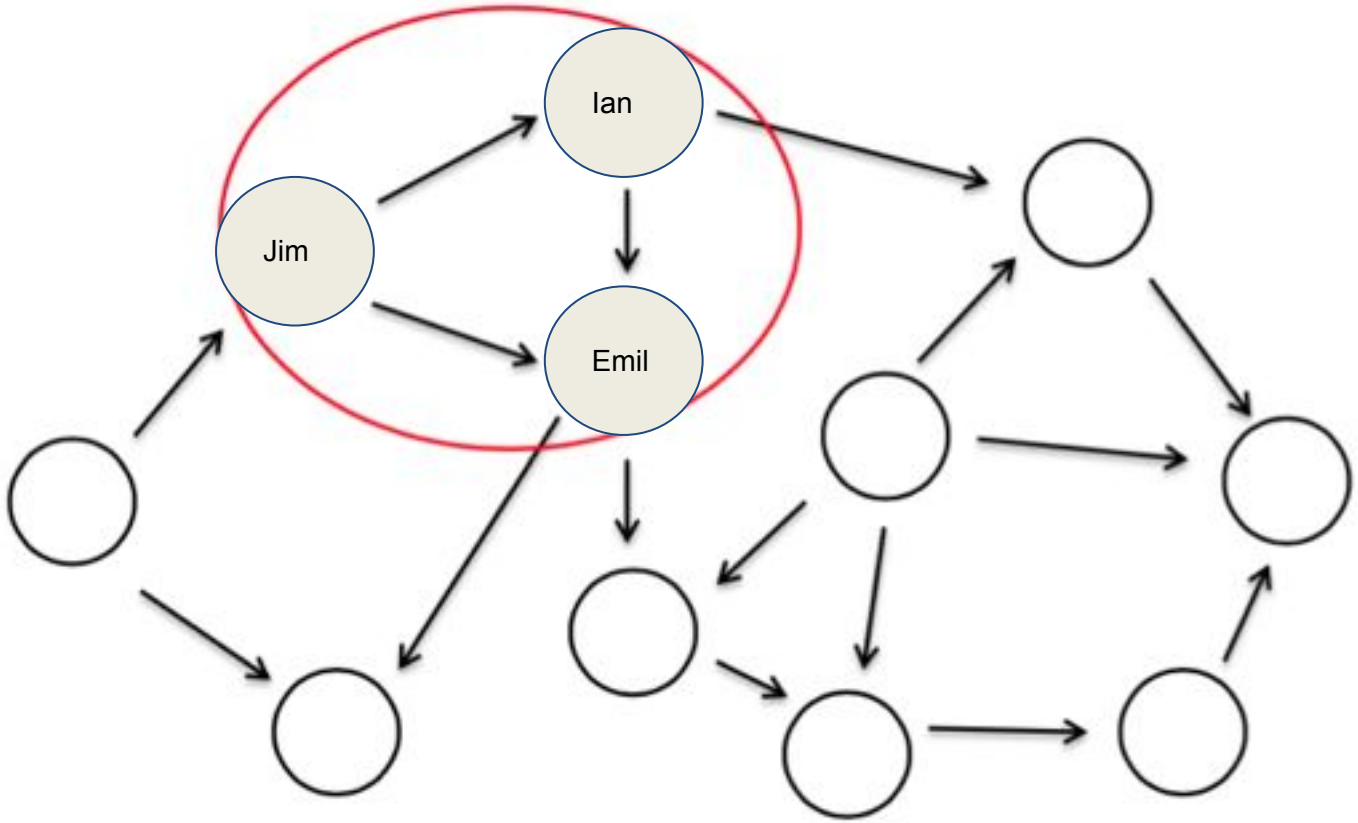
Consultas: Identificar Patrones



Amigos mutuos

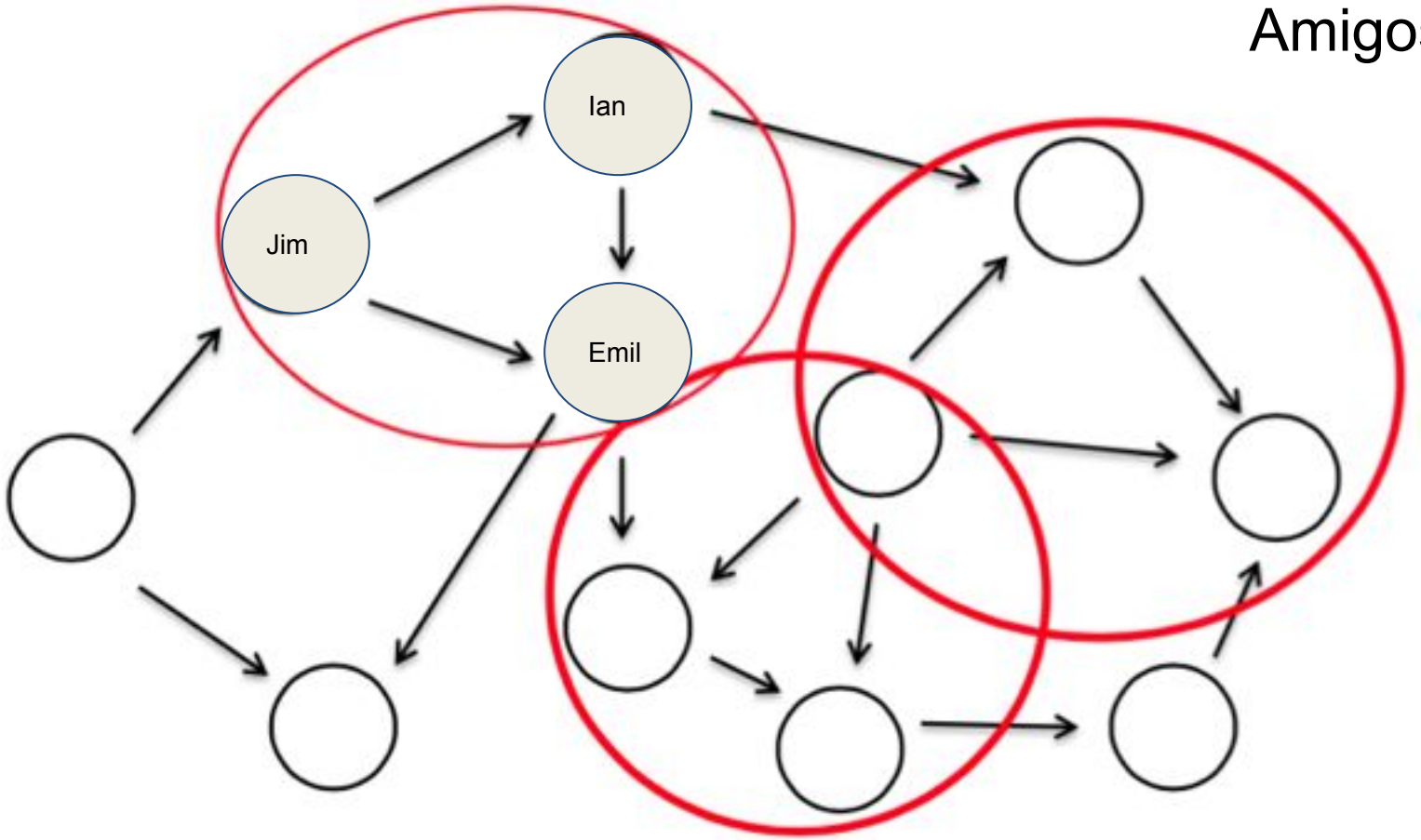
Amigos de Jim que se conocen entre sí.

Consultas: Identificar Patrones

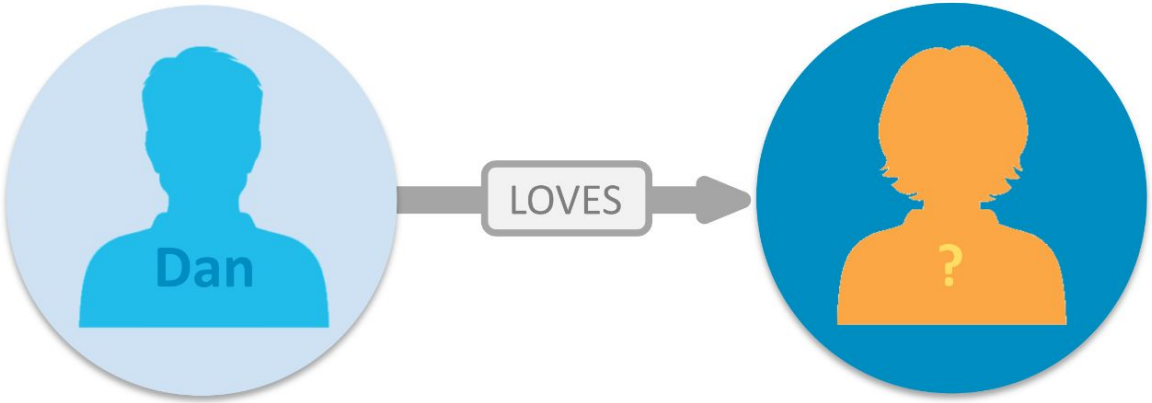


Consultas: Identificar Patrones

Amigos mutuos



Consultas: Cypher

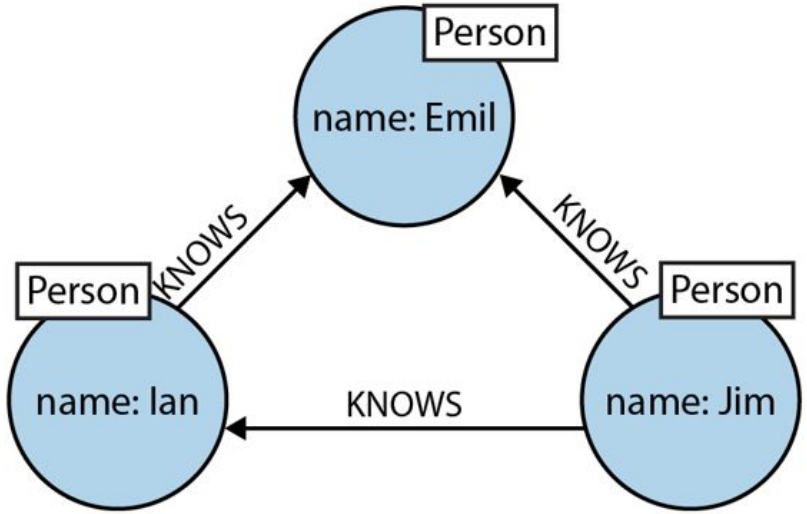


NODE Relationship NODE

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

LABEL PROPERTY VARIABLE

Consultas: Identificar Patrones




```
()<-[:KNOWS]-()-[:KNOWS]->()-[:KNOWS]->()
```

```
(a)<-[:KNOWS]-(b)-[:KNOWS]->(c)-[:KNOWS]->(a)
```

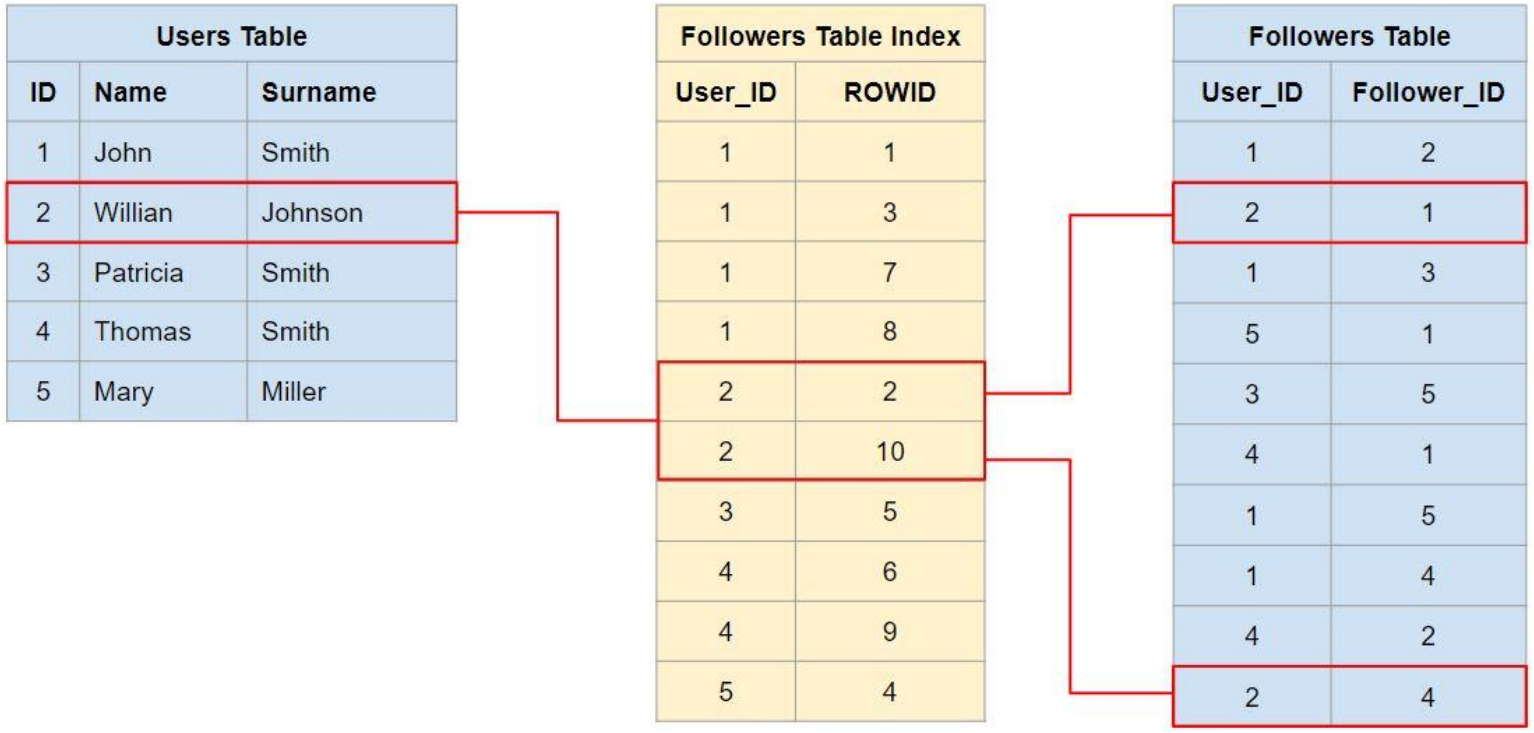
```
(a:Person)<-[:KNOWS]-(:Person)-[:KNOWS]->(:Person)-[:KNOWS]->(a)
```

```
(a:Person {name:'Emil'})
  <-[:KNOWS]-(b:Person {name:'Jim'})
  -[:KNOWS]->(c:Person {name:'Ian'})
  -[:KNOWS]->(a)
```

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c),
      (a)-[:KNOWS]->(c)
RETURN b, c
```

```
MATCH (a:Person)-[:KNOWS]->(b)-[:KNOWS]->(c),
      (a)-[:KNOWS]->(c)
WHRE a.name = 'Jim'
RETURN b, c
```


Index-free adjacency



Users Table

ID	Name	Surname
1	John	Smith
2	Willian	Johnson
3	Patricia	Smith
4	Thomas	Smith
5	Mary	Miller

Followers Table Index

User_ID	ROWID
1	1
1	3
1	7
1	8
2	2
2	10
3	5
4	6
4	9
5	4

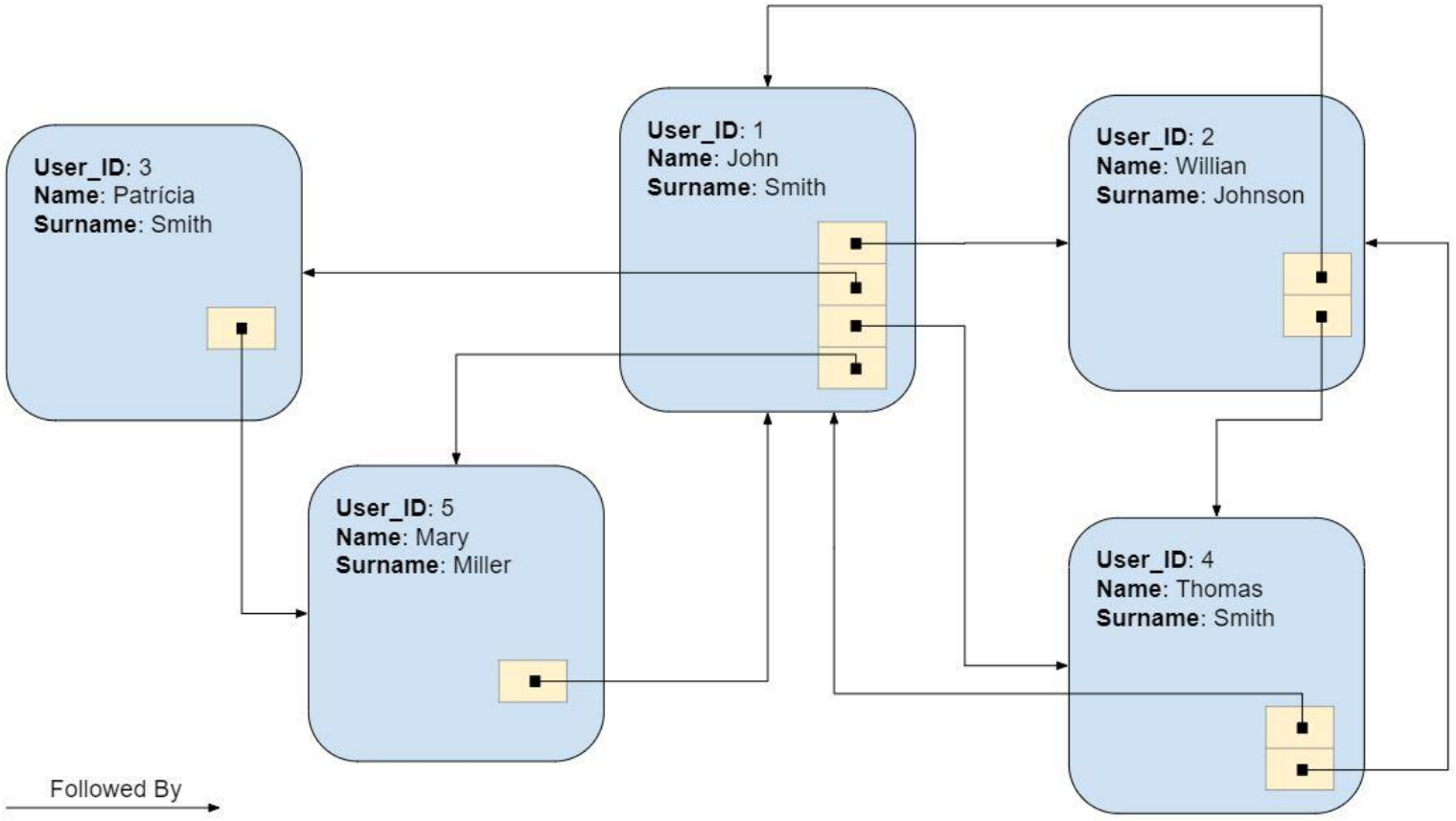
Followers Table

User_ID	Follower_ID
1	2
2	1
1	3
5	1
3	5
4	1
1	5
1	4
4	2
2	4

Bases de datos relacionales:

- Orientadas a registros.
- Utilizan índices globales para encontrar los registros.
- Mantienen PK y FK para gestionar las relaciones.
- Operan con Joins, operaciones muy costosas en cómputo.
- Modeladas para minimizar las relaciones entre tablas.

Index-free adjacency



Grafos nativos:

- Realizan un recorrido transversal del grafo.
- No usan índices globales, cada nodo tiene un índice de sus nodos vecinos.
- Guardan en memoria RAM la representación del grafo para agilizar las búsquedas.
- Las relaciones son ciudadanos de primer nivel.
- Cuantas más relaciones tengamos, más eficiente será la consulta del grafo.

Index-free adjacency

- Las relaciones se guardan en memoria RAM.
- Los atributos se guardan en disco.
- El acceso a memoria RAM es mucho más rápido que el acceso a disco.
- Cuanto más **expresivo** sea el grafo (más relaciones contenga), más **eficientes** serán sus consultas.

- Hay búsquedas que parten de un nodo origen.
- Hay búsquedas que trabajan sobre nodos con etiquetas concretas.
- Hay búsquedas en las que es necesario acceder a los atributos del nodo.
- Neo4j dispone de índices para agilizar la búsqueda de estos nodos sobre sus etiquetas o sobre sus atributos.

Modelado

Modelado

- **Nodos**: Representan **entidades**. Las "*cosas*" del dominio que pueden ser **etiquetadas** y **agrupadas**.
- **Relaciones**: Expresan **conexiones** entre las entidades, establecen la **semántica** del contexto y **estructuran** el dominio.
    - **Dirección**: clarifican la semántica. En relaciones bidireccionales es mejor ignorar la dirección en las sentencias que mantener las dos direcciones.
- **Propiedades** de los **nodos**: Representan **atributos** de las entidades y otros metadatos como timestamps o número de versión.
- **Propiedades** de las **relaciones**: Se utilizan para expresar fuerza, peso o calidad de la relación y otros metadatos como timestamps o número de versión.

Modelado

Relaciones:

- Específicas: DELIVERY_ADDRESS, HOME_ADDRESS
  - En el caso de que hay un conjunto con pocos tipos diferentes.
  - Más eficiente, sólo utiliza las relaciones implicadas en la consulta.
  - Complica las consultas cuando queremos consultar más de una relación: MATCH (user)-[:HOME_ADDRESS|WORK_ADDRESS|DELIVERY_ADDRESS]->(address)
- Genéricas: ADDRESS {type: 'delivery'}, ADDRESS{type: 'home'}
  - En el caso de que el conjunto de tipos de relaciones sea muy grande.
  - Las consultas tienen que acceder a todas las relaciones para filtrarlas.

Modelado

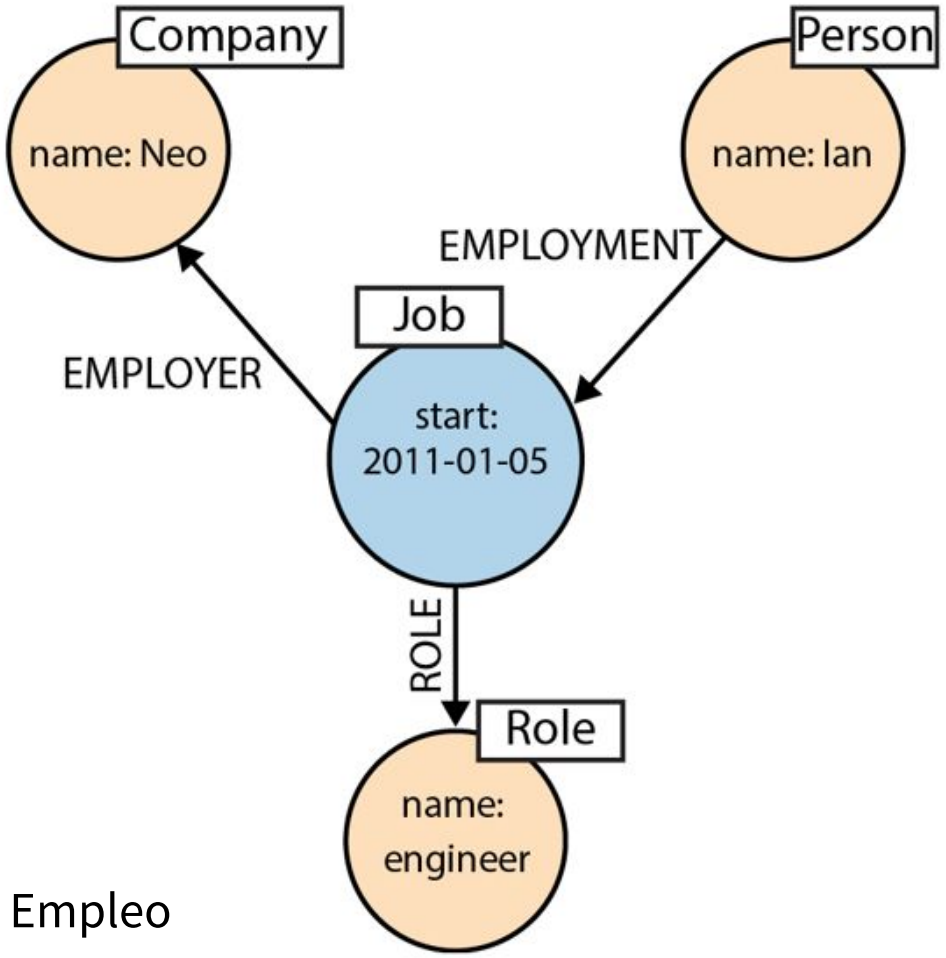
Relaciones:

- Separar en relaciones específicas las más utilizadas y mantener en relación genérica el resto.
    - DELIVERY_ADDRESS, ADDRESS {type: ‘home’}

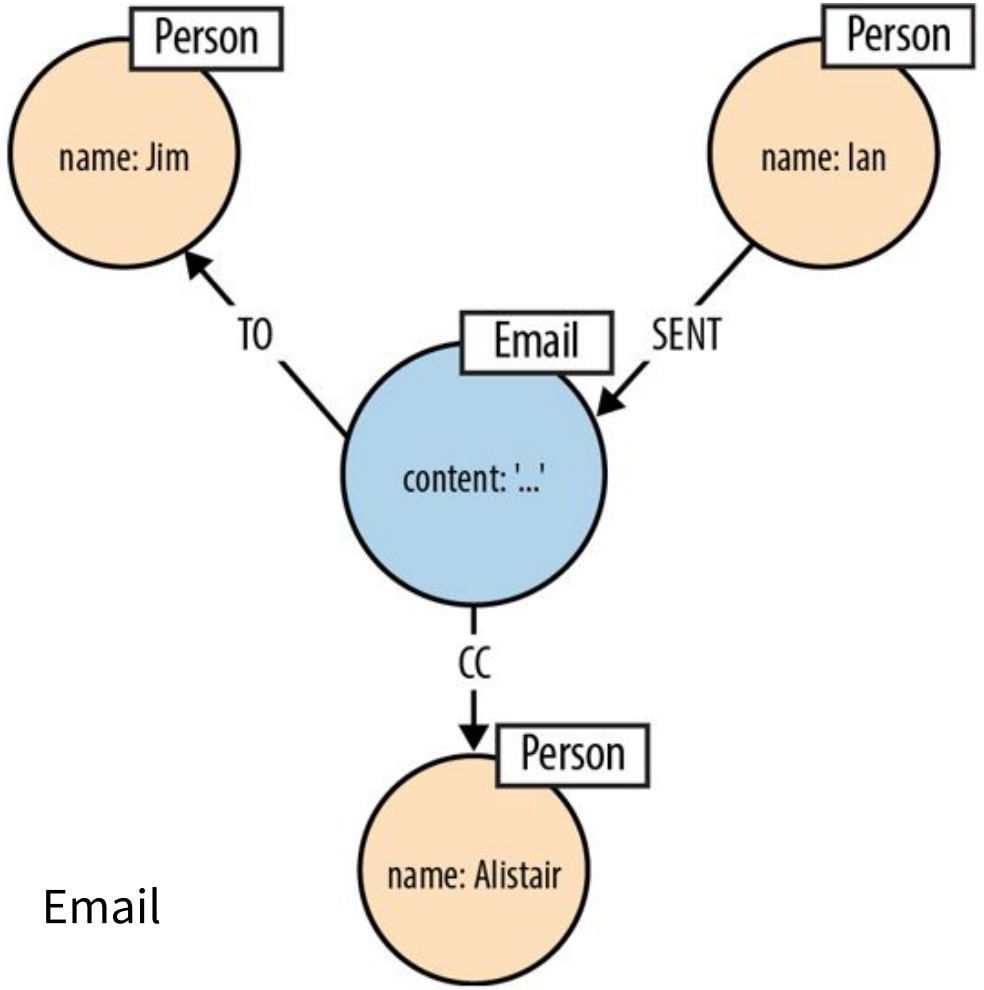
Hechos:

- Cuando dos entidades interaccionan en un periodo de tiempo.
- Se representa como un nodo nuevo relacionado con las entidades que interaccionan.
- Se pueden utilizar timestamp como propiedades para indicar en qué momento de tiempo estas dos entidades interaccionaron.

Modelado



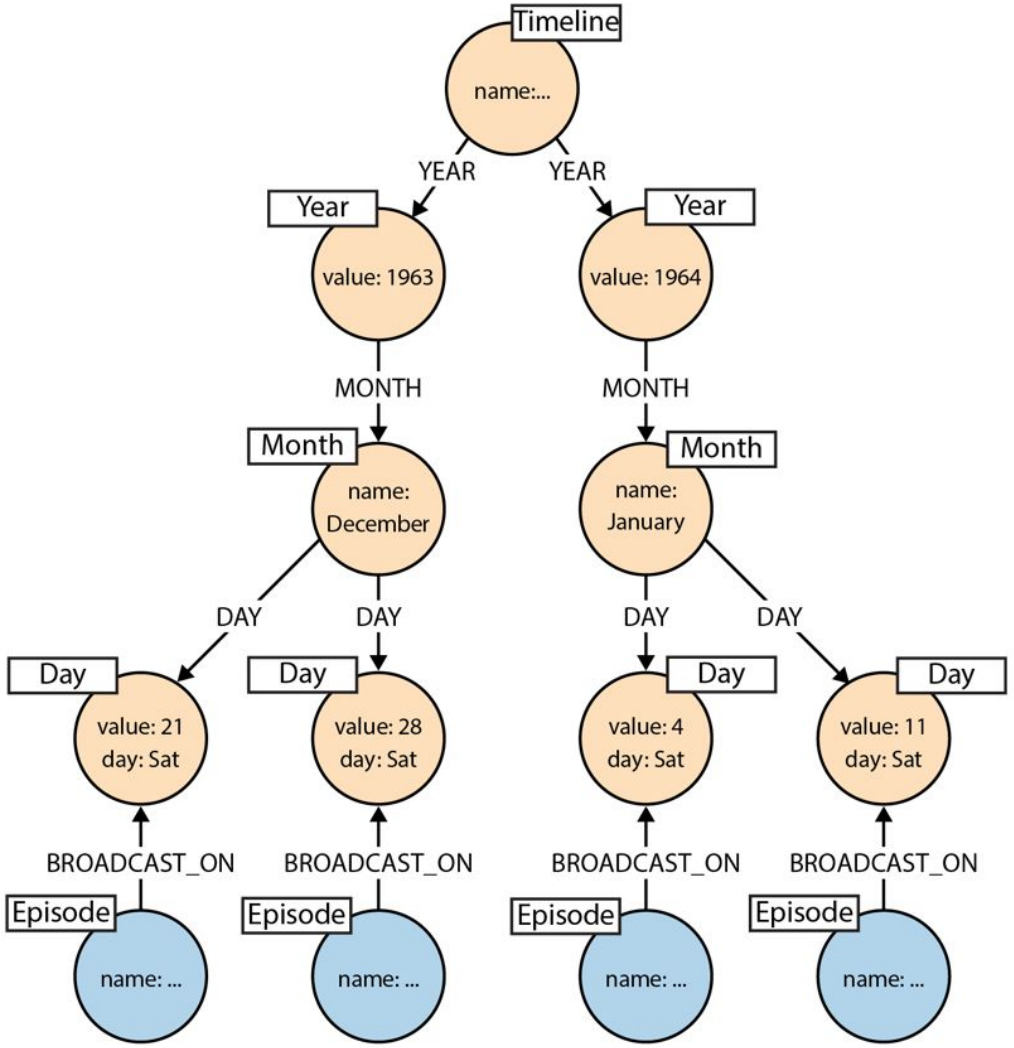
Empleo



Email

Modelado

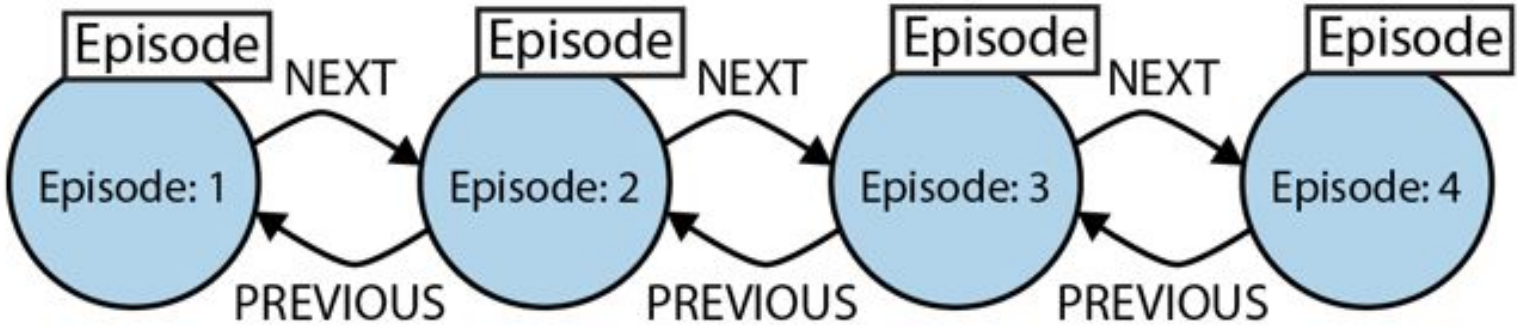
Tiempo



Rafael Garrote Hernández

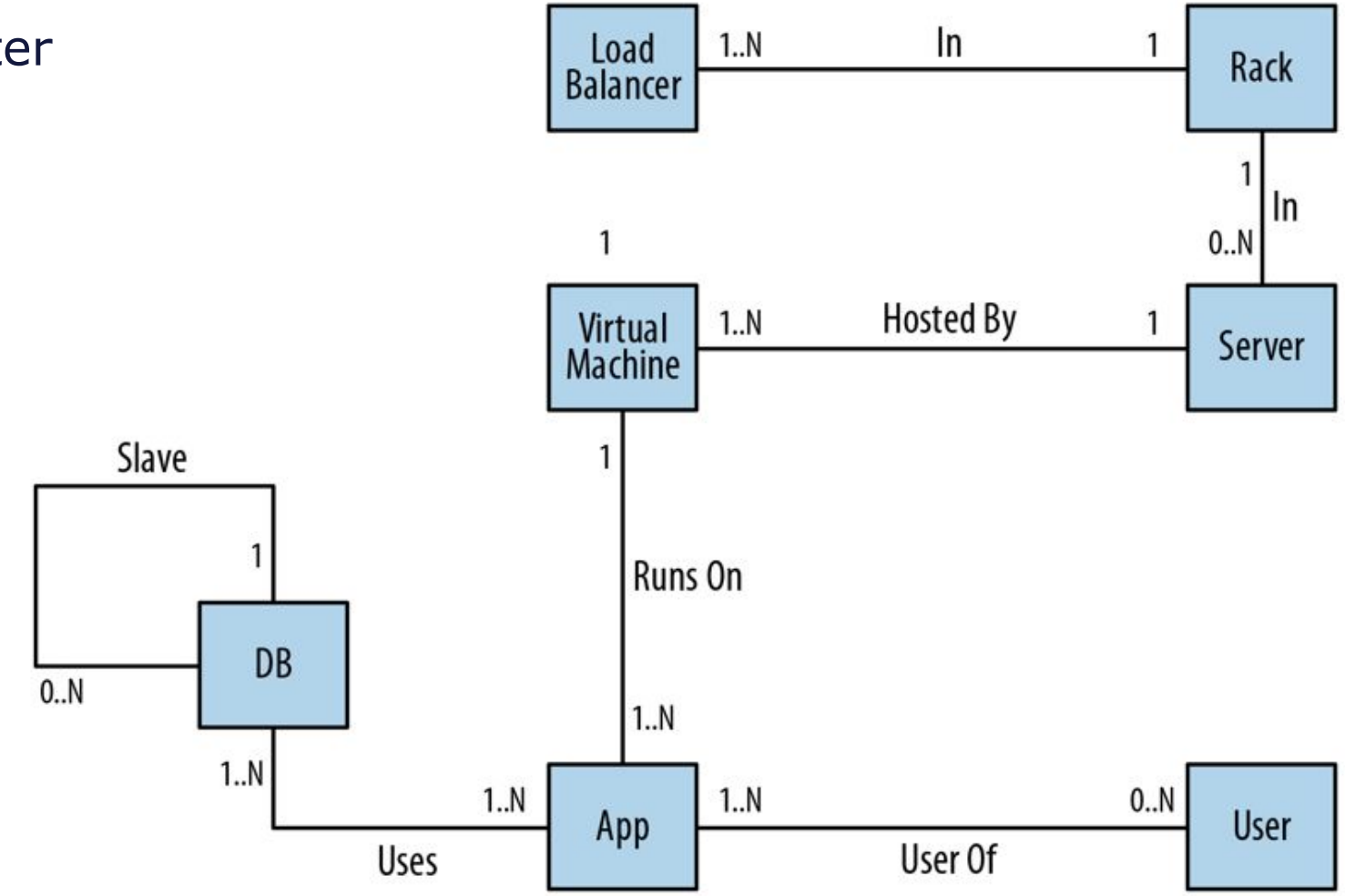
Modelado

Listas enlazadas

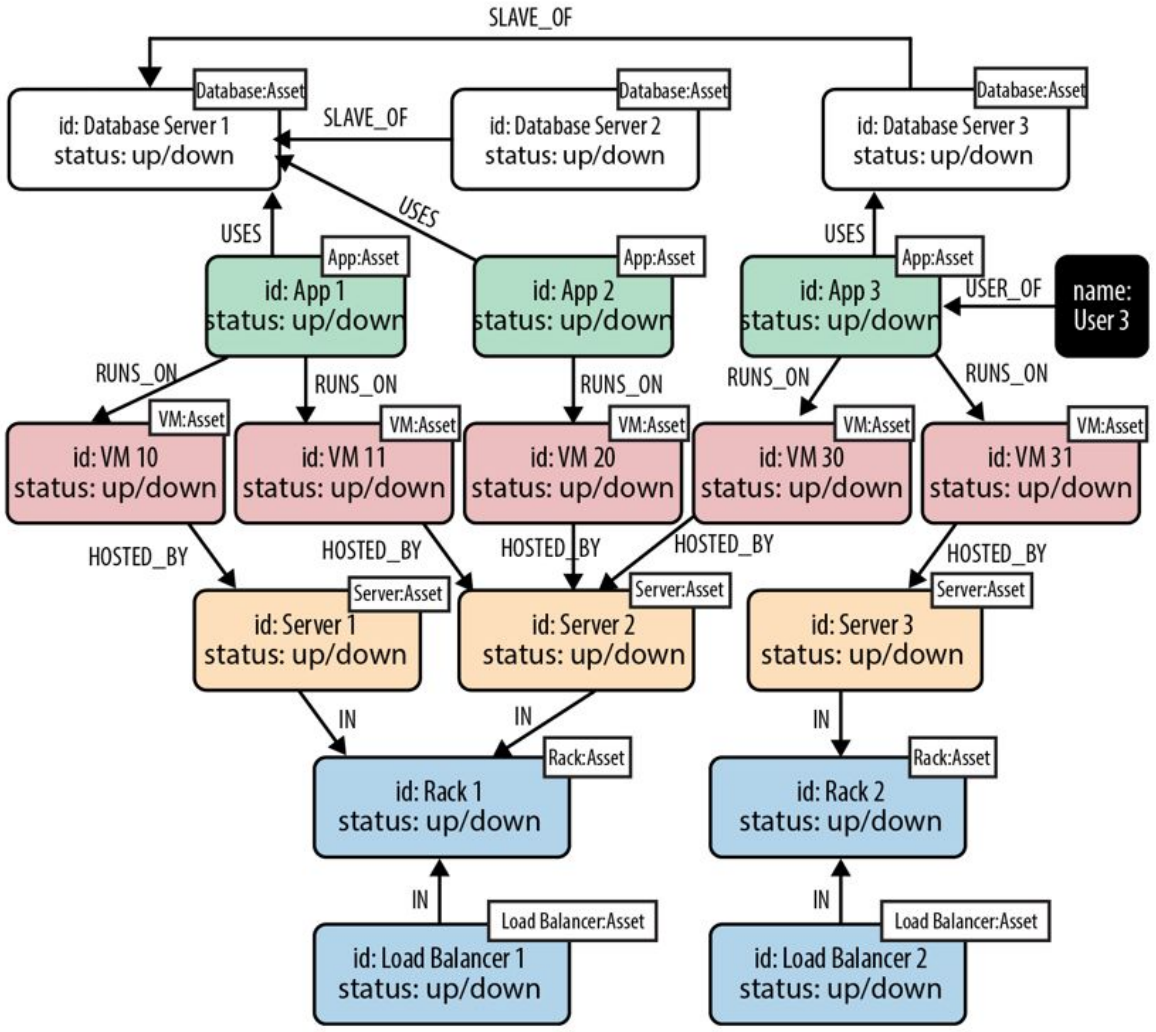


Modelado

Datacenter



Modelado




```
MATCH (user:User)-[*1..5]-(asset:Asset)
WHERE user.name = 'User 3' AND asset.status = 'down'
RETURN DISTINCT asset
```

```
(user)-[:USER_OF]->(app)
(user)-[:USER_OF]->(app)-[:USES]->(database)
(user)-[:USER_OF]->(app)-[:USES]->(database)-[:SLAVE_OF]->(another-database)
(user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)
(user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)-[:HOSTED_BY]->(server)
(user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)-[:HOSTED_BY]->(server)-[:IN]->(rack)
(user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)-[:HOSTED_BY]->(server)-[:IN]->(rack)
  <-[:IN]-(load-balancer)
```

Rafael Garrote Hernández

Cypher

Cypher

Match: Se utiliza para indicar el path que tiene que cumplir el patrón de búsqueda

```
()
(matrix)
(:Movie)
(matrix:Movie)
(matrix:Movie {title: "The Matrix"})
(matrix:Movie {title: "The Matrix", released: 1997})
```

```
-->
-[role]->
-[:ACTED_IN]->
-[role:ACTED_IN]->
-[role:ACTED_IN {roles: ["Neo"]}]->
```


Cypher

Where: Filtros a realizar sobre el patrón de búsqueda.

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
WHERE p.name =~ "K.+" OR m.released > 2000 OR "Neo" IN r.roles
RETURN p,r,m
```

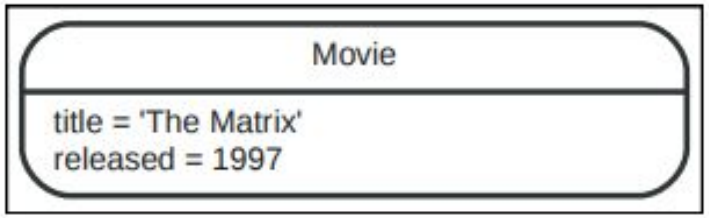
```
+--------------------------------------------------------------------------------------------------------
---------------------+
| p                                          | r                                | m
|
+--------------------------------------------------------------------------------------------------------
---------------------+
| (:Person {name: "Tom Hanks", born: 1956}) | [:ACTED_IN {roles: ["Zachry"]}] | (:Movie {title: "Cloud
Atlas", released: 2012}) |
+--------------------------------------------------------------------------------------------------------
---------------------+

1 row
```


Cypher

Create: Permite crear nodos y relaciones.

```
CREATE (:Movie { title:"The Matrix",released:1997 })
```

```
+-------------------+
| No data returned. |
+-------------------+
Nodes created: 1
Properties set: 2
Labels added: 1
```

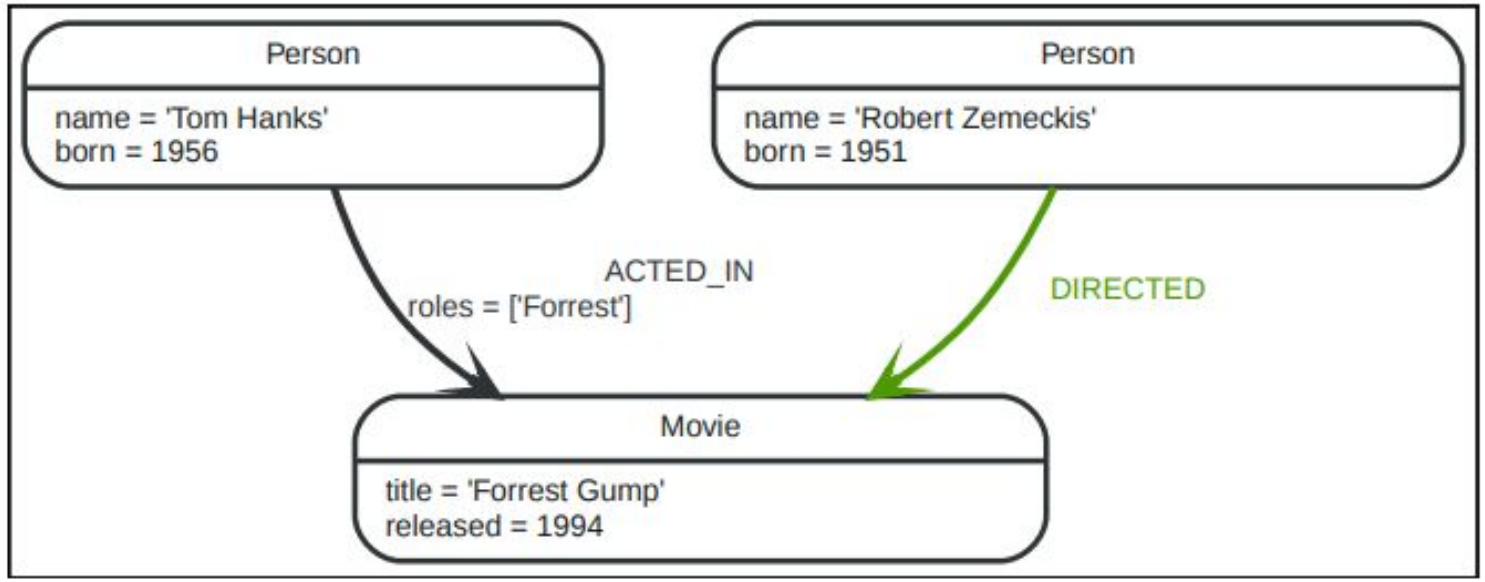



```
CREATE (p:Person { name:"Keanu Reeves", born:1964 })
RETURN p
```

```
+--------------------------------------+
| p                                    |
+--------------------------------------+
| Node[1]{name:"Keanu Reeves",born:1964} |
+--------------------------------------+
1 row
Nodes created: 1
Properties set: 2
Labels added: 1
```

Cypher

```
CREATE (a:Person { name:"Tom Hanks",
  born:1956 })-[r:ACTED_IN { roles: ["Forrest"]}]->(m:Movie { title:"Forrest Gump",released:1994 })
CREATE (d:Person { name:"Robert Zemeckis", born:1951 })-[:DIRECTED]->(m)
RETURN a,d,r,m
```



Cypher

Merge: Se asegura de que el patrón indicado se cumple, bien porque los nodos y relaciones existen o creando nuevos nodos y relaciones.

```
MERGE (m:Movie { title:"Cloud Atlas" })
ON CREATE SET m.released = 2012
RETURN m
```

```
+-------------------------------------------+
| m                                         |
+-------------------------------------------+
| Node[5]{title:"Cloud Atlas",released:2012} |
+-------------------------------------------+
1 row
```


Cypher

Union: Junta el resultado de una o más sentencias.

```
MATCH (actor:Person)-[r:ACTED_IN]->(movie:Movie)
RETURN actor.name AS name, type(r) AS type, movie.title AS title
UNION
MATCH (director:Person)-[r:DIRECTED]->(movie:Movie)
RETURN director.name AS name, type(r) AS type, movie.title AS title
```

```
+-----------------------------------------------+
| name              | type       | title        |
+-----------------------------------------------+
| "Tom Hanks"       | "ACTED_IN" | "Cloud Atlas"  |
| "Tom Hanks"       | "ACTED_IN" | "Forrest Gump" |
| "Robert Zemeckis" | "DIRECTED" | "Forrest Gump" |
+-----------------------------------------------+

3 rows
```


Cypher

With: Encadena los resultados de una búsqueda con una nueva búsqueda.

```
MATCH (person:Person)-[:ACTED_IN]->(m:Movie)
WITH person, count(*) AS appearances, collect(m.title) AS movies
WHERE appearances > 1
RETURN person.name, appearances, movies
```

```
+-------------------------------------------------------------+
| person.name | appearances | movies                          |
+-------------------------------------------------------------+
| "Tom Hanks" | 2           | ["Cloud Atlas", "Forrest Gump"] |
+-------------------------------------------------------------+

1 row
```


Cypher

Set: Permite informar el valor de una propiedad.

```
MATCH (n { name: 'Andy' })
SET n.surname = 'Taylor'
RETURN n.name, n.surname
```

n.name	n.surname
"Andy"	"Taylor"

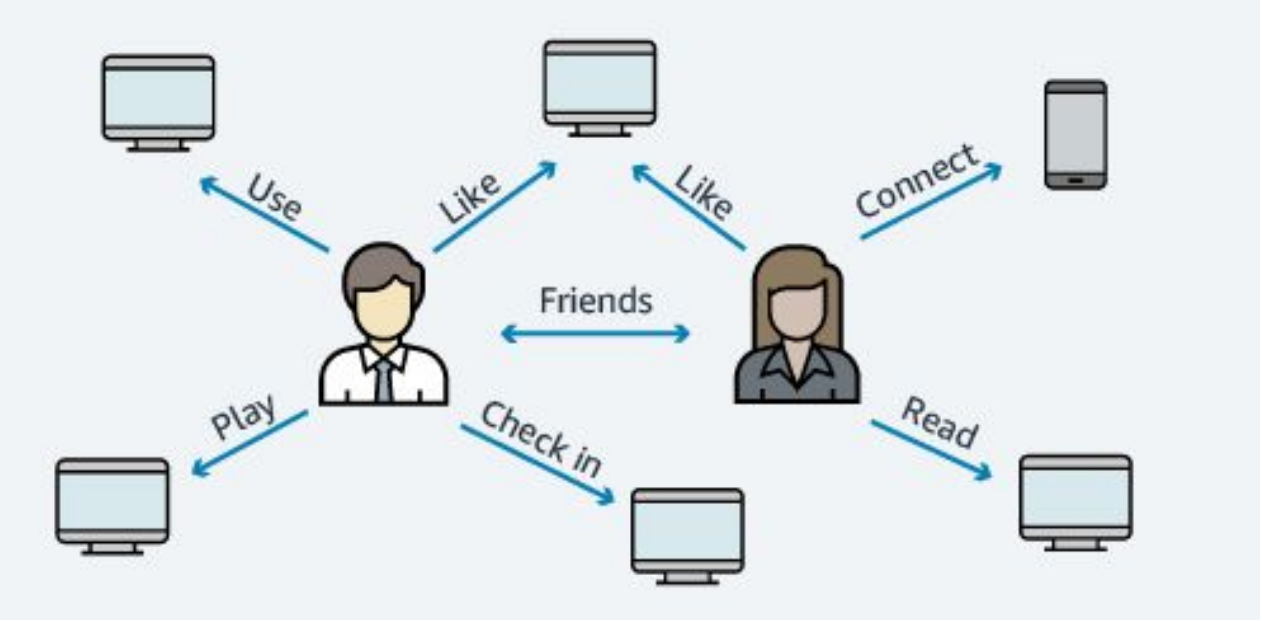
1 row, Properties set: 1

Casos de uso

Casos de uso

Redes sociales

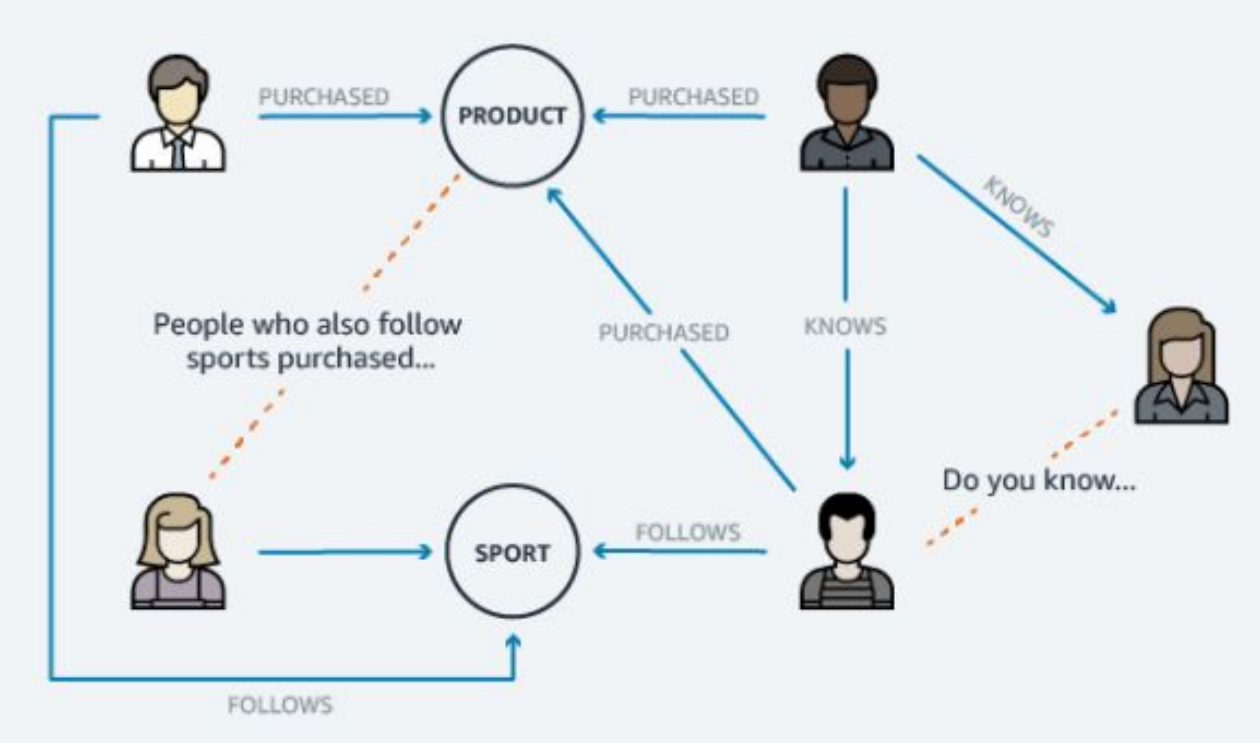
- Gestionar grandes volúmenes de perfiles de usuario y sus interacciones.
- Se puede construir un feed que provea resultados que priorice las modificaciones de perfiles de su familia, de los amigos que hagan un like y los amigos que vivan cerca.
- Detectar qué usuarios tienen más influencia en una red social.



Casos de uso

Motores de recomendación

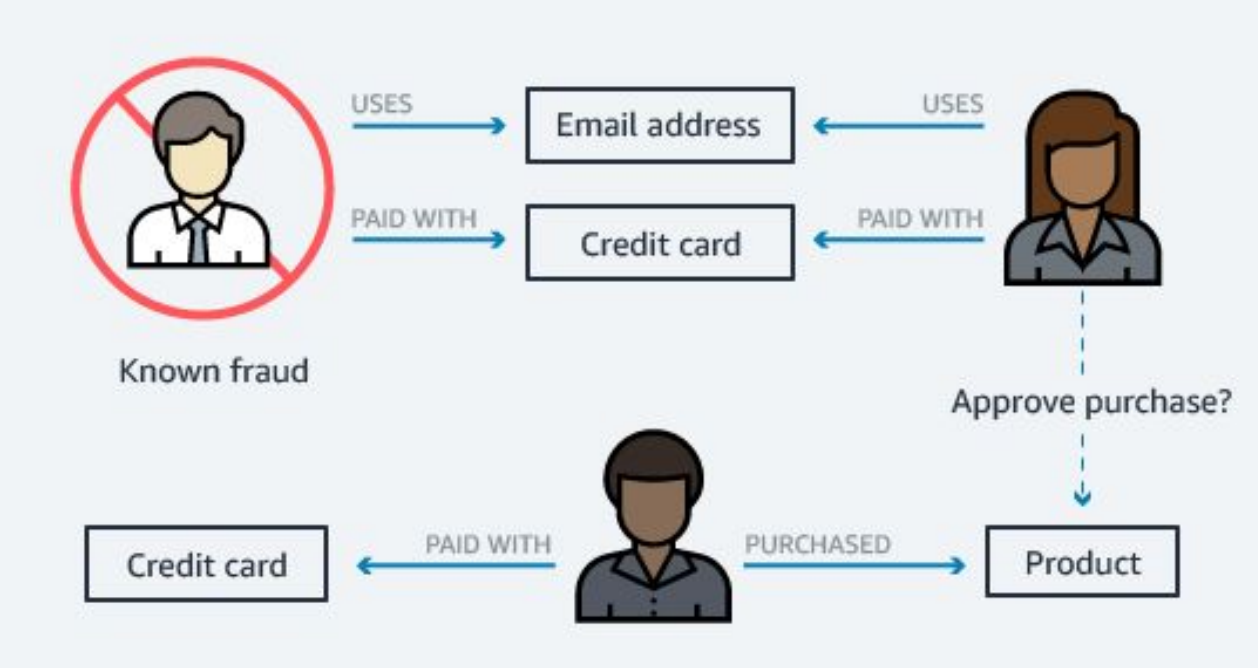
- Almacenar las relaciones de información como intereses de los compradores, amigos, historial de compra… para crear recomendaciones personalizadas y relevantes.
- Por ejemplo realizar recomendaciones en función de los historiales de compra de usuarios que siguen otros usuarios o de usuarios con gustos similares en función de sus likes.



Casos de uso

Detección de fraude

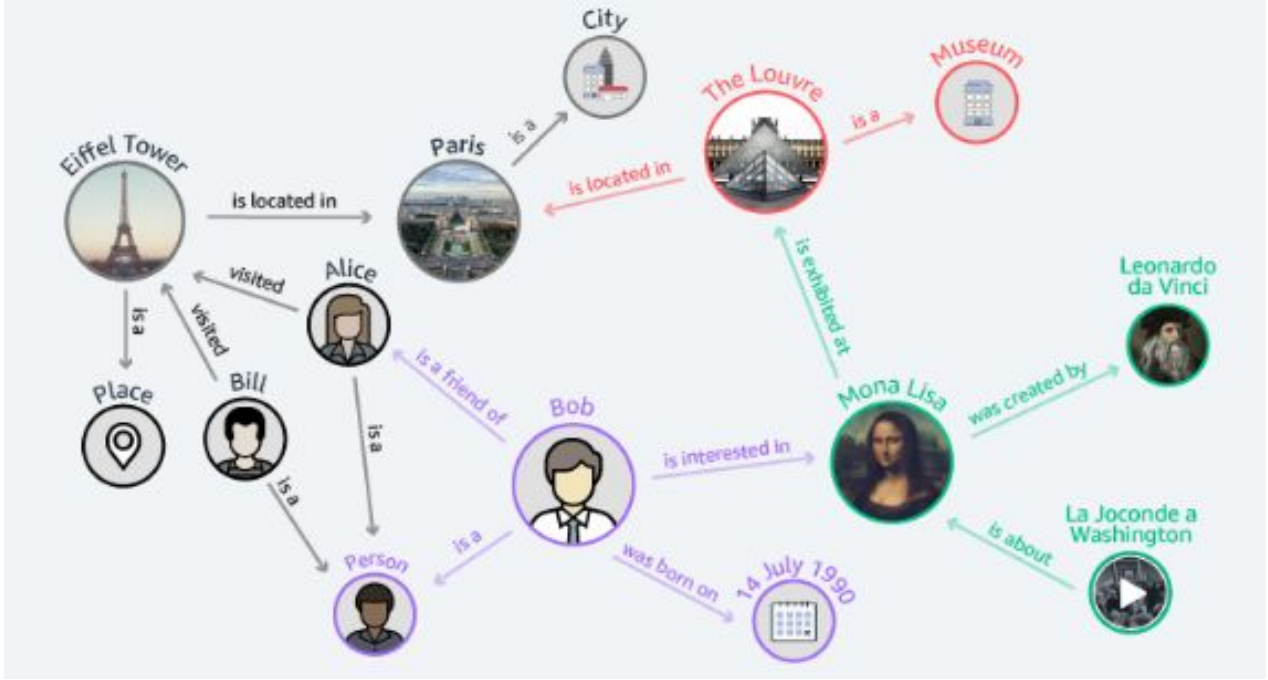
- Almacenar la relación de información de transacciones financieras.
- Buscar patrones de fraude como que varios usuarios utilicen el mismo correo o número de tarjeta de crédito.
- O transacciones realizadas con la misma IP pero que residen en países diferentes.



Casos de uso

Bases de conocimiento

- Relaccionar datasets altamente conectados.
- Por ejemplo, si un usuario está interesado por conocer la Mona Lisa, también se puede dar información sobre más obras de Leonardo o datos sobre París, la ciudad donde está expuesto el cuadro. También otras obras que a otros usuarios que le gusten la Mona Lisa también le interesen.



Muchas gracias

datahack
INSPIRING BIG DATA