e de datos no-relacionales?

Bibliografía

- NoSQL for Mere Mortals. Sullivan, Dan. 2015. 1st. edition. Addison-Wesley Professional (parte II y parte IV)
- Building Knowledge Graphs: A Practitioner's Guide. Jesus Barrasa, Jim Webber. 2023

5.1. Objetivos

1. Aprender a diseñar modelos de datos no relacionales. En ocasiones, exige enfoques de diseño y modelado de bases de datos fundamentalmente diferentes, alejándose de las técnicas relacionales convencionales.

2. Elección de tecnologías, el diseño de bases de datos de alto rendimiento y la planificación del mantenimiento a largo plazo.

2. Ejemplos destacados:
   - NoSQL: MongoDB, Cassandra, Redis.
   - Basadas en grafos: Neo4J, Cambridge Semantics, TypeDB
   - Vectoriales: milvus, Qdrant, Pinecone, Chroma, PgVector (paquete de PostgreSQL)

3. Pautas prácticas para elegir la base de datos más adecuada y orientación para diseñar bases de datos con eficiencia.

4. Evitar errores comunes que puedan resultar en un bajo rendimiento y requisitos insatisfechos.

5.1. ¿Por qué BBDD no-relacionales?

1. Coste

Description	License Cost per Single-Core CPU	Number of Licensed Servers	Number of Licensed CPUs	Number of Cores	Number of Equivalent CPUs	Cost
Enterprise Edition	$47,500	4	16	64	48	$2,280,000
Real Application Clusters	$23,000	2	8	32	24	$552,000
Partitioning Option	$11,500	4	16	64	48	$552,000
Active Data Guard	$5,800	3	12	48	36	$208,800
Total Recall	$5,800	3	12	48	36	$208,800
Diagnostics Pack	$3,500	3	12	48	36	$126,000
Tuning Pack	$3,500	3	12	48	36	$126,000
Configuration Management Pack	$3,500	4	16	64	48	$168,000
Provisioning Pack	$3,500	4	16	64	48	$168,000
Total Purchase Price						$4,389,600
Annual Support Fees						$965,712

2. Flexibilidad
   - Se espera conocer de antemano todas las tablas y columnas necesarias para respaldar una aplicación.
   - Se asume comúnmente que la mayoría de las columnas en una tabla serán necesarias para la mayoría de las filas.
   - Algunas aplicaciones pueden requerir una estructura de datos menos homogénea, como en el caso de atributos de productos en una aplicación de comercio electrónico.

3. No siempre un modelo relacional (plano) se adecúa al dominio de la aplicación

5.1. ¿Por qué BBDD no-relacionales?

Tema 5: Bases de datos no relacionales

1. Coste

Description	License Cost per Single-Core CPU	Number of Licensed Servers	Number of Licensed CPUs	Number of Cores	Number of Equivalent CPUs	Cost
Enterprise Edition	$47,500	4	16	64	48	$2,280,000
Real Application Clusters	$23,000	2	8	32	24	$552,000
Partitioning Option	$11,500	4	16	64	48	$552,000
Active Data Guard	$5,800	3	12	48	36	$208,800
Total Recall	$5,800	3	12	48	36	$208,800
Diagnostics Pack	$3,500	3	12	48	36	$126,000
Tuning Pack	$3,500	3	12	48	36	$126,000
Configuration Management Pack	$3,500	4	16	64	48	$168,000
Provisioning Pack	$3,500	4	16	64	48	$168,000
Total Purchase Price						$4,389,600
Annual Support Fees						$965,712

¿Qué elemento fundamental limita la flexibilidad de la BBDD?

2. Flexibilidad
   - Se espera conocer de antemano todas las tablas y columnas necesarias para respaldar una aplicación.
   - Se asume comúnmente que la mayoría de las columnas en una tabla serán necesarias para la mayoría de las filas.
   - Algunas aplicaciones pueden requerir una estructura de datos menos homogénea, como en el caso de atributos de productos en una aplicación de comercio electrónico.

3. No siempre un modelo relacional (plano) se adecúa al dominio de la aplicación


Grado en Informática Gestión y Administración de Bases de Datos

5.1. ¿Por qué BBDD no-relacionales?

1. Coste

Description	License Cost per Single-Core CPU	Number of Licensed Servers	Number of Licensed CPUs	Number of Cores	Number of Equivalent CPUs	Cost
Enterprise Edition	$47,500	4	16	64	48	$2,280,000
Real Application Clusters	$23,000	2	8	32	24	$552,000
Partitioning Option	$11,500	4	16	64	48	$552,000
Active Data Guard	$5,800	3	12	48	36	$208,800
Total Recall	$5,800	3	12	48	36	$208,800
Diagnostics Pack	$3,500	3	12	48	36	$126,000
Tuning Pack	$3,500	3	12	48	36	$126,000
Configuration Management Pack	$3,500	4	16	64	48	$168,000
Provisioning Pack	$3,500	4	16	64	48	$168,000
Total Purchase Price						$4,389,600
Annual Support Fees						$965,712

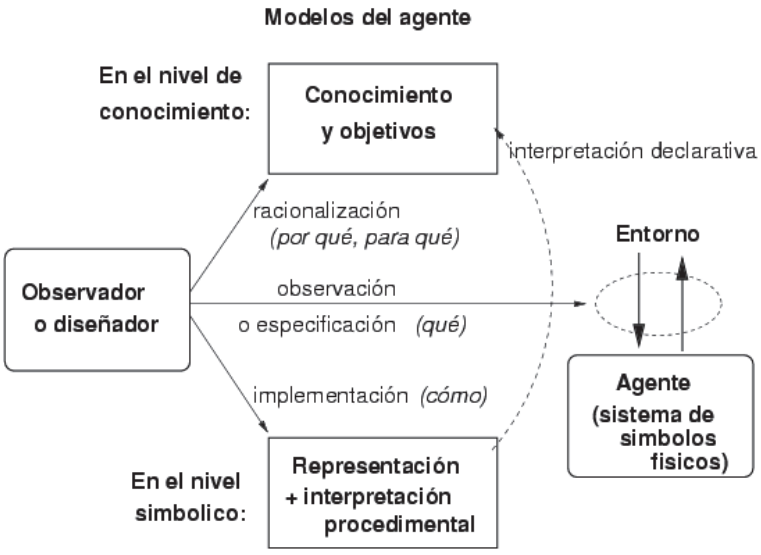
2. Flexibilidad
   - Se espera conocer de antemano todas las tablas y columnas necesarias para respaldar una aplicación.
   - Se asume comúnmente que la mayoría de las columnas en una tabla serán necesarias pa[illegible] filas.
   - Algu[illegible]a estructura de datos menos homoge[illegible]butos de productos en una aplicación de com[illegible]electrónico.

¿Por qué es necesario normalizar?

3. No siempre un modelo relacional (plano) se adecúa al dominio de la aplicación

5.1. ¿Por qué BBDD no-relacionales?

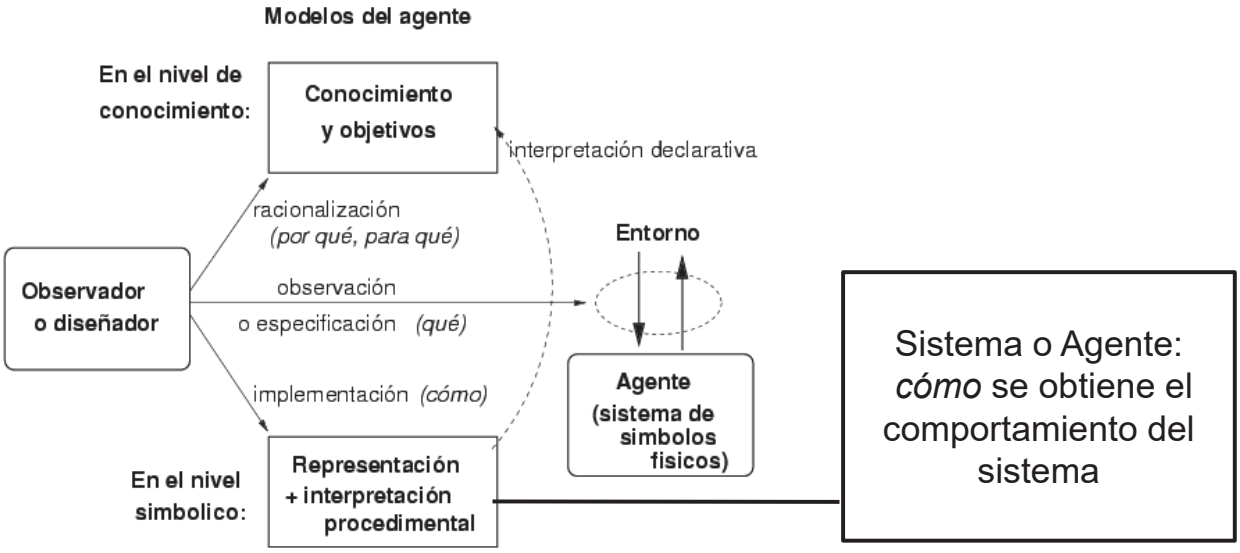
Tema 5: Bases de datos no relacionales



- Nivel simbólico o representacional: representación y manipulación de información mediante símbolos, como palabras, números o diagramas.

- Nivel de conocimiento: explica, o predice, el comportamiento atribuyendo al agente cierto conocimiento, ciertos objetivos y el principio de racionalidad.

Grado en Informática Gestión y Administración de Bases de Datos

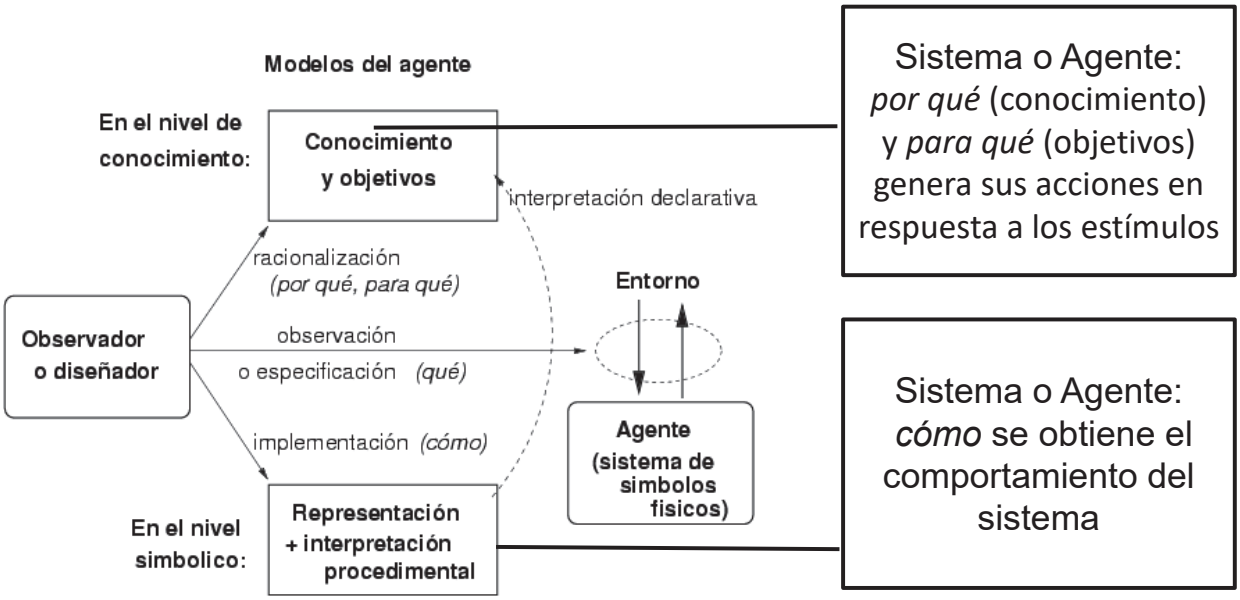


- Nivel simbólico o representacional: representación y manipulación de información mediante símbolos, como palabras, números o diagramas.

- Nivel de conocimiento: explica, o predice, el comportamiento atribuyendo al agente cierto conocimiento, ciertos objetivos y el principio de racionalidad.

5.1. ¿Por qué BBDD no-relacionales?

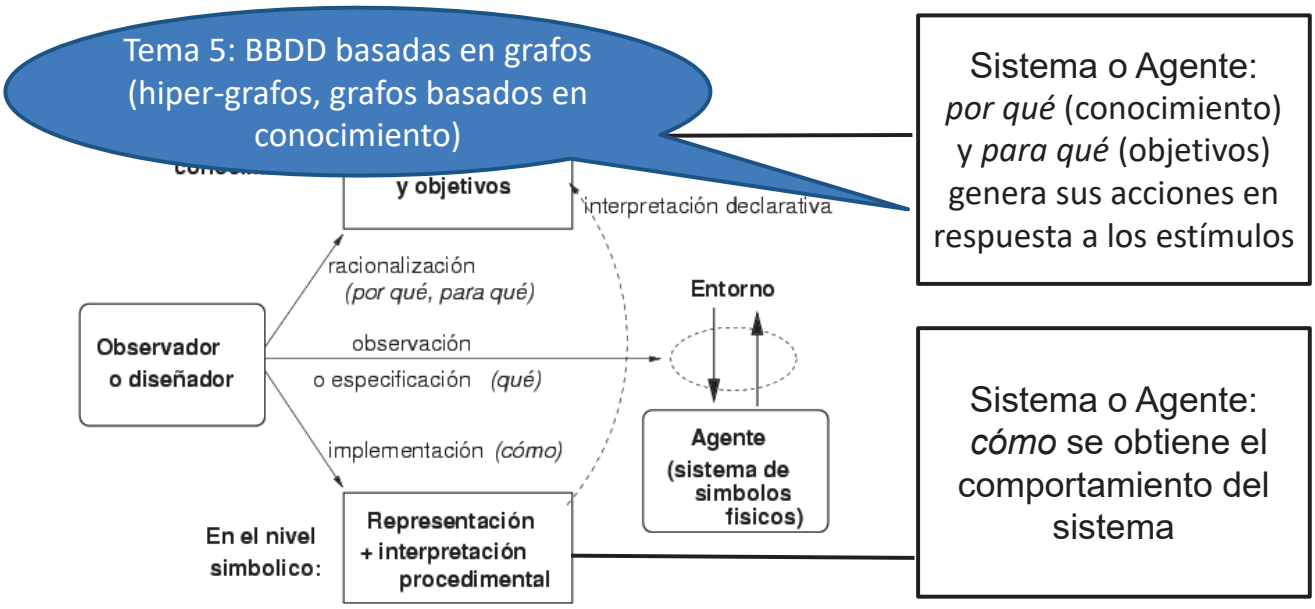
Tema 5: Bases de datos no relacionales



- Nivel simbólico o representacional: representación y manipulación de información mediante símbolos, como palabras, números o diagramas.

- Nivel de conocimiento: explica, o predice, el comportamiento atribuyendo al agente cierto conocimiento, ciertos objetivos y el principio de racionalidad.

5.1. ¿Por qué BBDD no-relacionales?



- Nivel simbólico o representacional: representación y manipulación de información mediante símbolos, como palabras, números o diagramas.

- Nivel de conocimiento: explica, o predice, el comportamiento atribuyendo al agente cierto conocimiento, ciertos objetivos y el principio de racionalidad.

5.2. BBDD no-SQL

¿Por qué?

1. Se desarrolló para superar las limitaciones de las bases de datos relacionales en aplicaciones web a gran escala en empresas como Google, Yahoo y Facebook.

2. Simplifica la escalabilidad y requiere menos código y sobrecarga de gestión en comparación con las bases de datos relacionales.
3. Principales ejemplos:
   - MongoDB, una base de datos de documentos
   - Cassandra, una base de datos de familia de columnas
   - Redis, una base de datos clave-valor;

¿Por qué?

¿No-SQL==No relacional?

1. Se desarrolló para superar las limitaciones de las bases de datos relacionales en aplicaciones web a gran escala en empresas como Google, Yahoo y Facebook.

2. Simplifica la escalabilidad y requiere menos código y sobrecarga de gestión en comparación con las bases de datos relacionales.
3. Principales ejemplos:
   - MongoDB, una base de datos de documentos
   - Cassandra, una base de datos de familia de columnas
   - Redis, una base de datos clave-valor;

¿Qué son?

- Las bases de datos clave-valor son las más simples dentro de las bases de datos NoSQL.
- Almacenan datos con identificadores llamados claves.

Database

Bucket 1	
'Foo1'	'Bar'
'Foo2'	'Bar2'
'Foo3'	'Bar7'

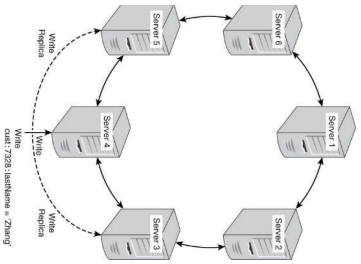
Bucket 2	
'Foo1'	'Baz'
'Foo4'	'Baz3'
'Foo6'	'Baz2'

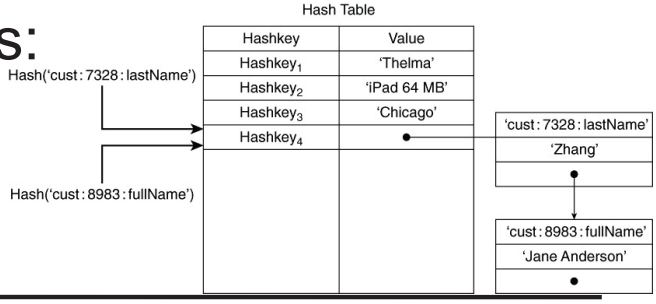
Bucket 3	
'Foo1'	'Bar7'
'Foo4'	'Baz3'
'Foo7'	'Baz9'

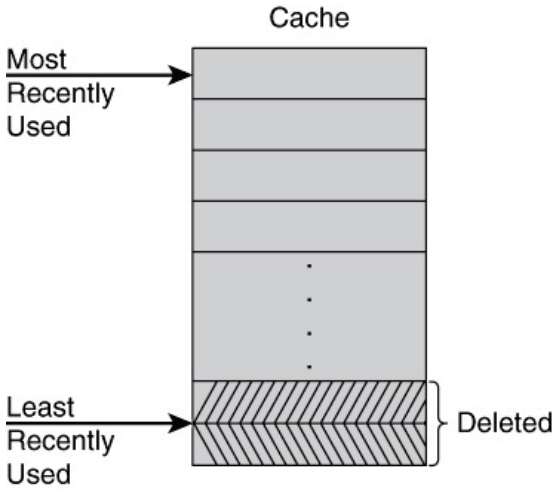
- **Amazon DynamoDB**: pertenece a Amazon Web Services (AWS) y puede emplearse también como base de datos de documentos.
- **Berkeley DB**: desarrollado por Oracle, ofrece interfaces para diferentes lenguajes de programación.
- **Redis:** proyecto de código abierto. Constituye uno de los SGBD más utilizados y en una fase temprana fue empleado por Instagram y GitHub.
- **Riak**: Riak existe en una variante de código libre, como solución empresarial y en forma de almacenamiento en la nube.
- **Voldemort**: extendido SGBD, utilizado e impulsado por LinkedIn, entre otros.

- Elementos estructurales:
  - clave
  - valor
  - espacio de nombres
  - particiones
- Aspectos arquitectónicos:
  - clusters
  - anillos (rings)
  - réplicas
- Aspectos metodológicos:
  - funciones hash
  - colisiones
  - no tipadas (schemaless)

Server Name	Partition Range
Server 1	0–11
Server 2	12–23
Server 3	24–35
Server 4	36–47
Server 5	48–59
Server 6	60–71
Server 7	72–83
Server 8	84–95

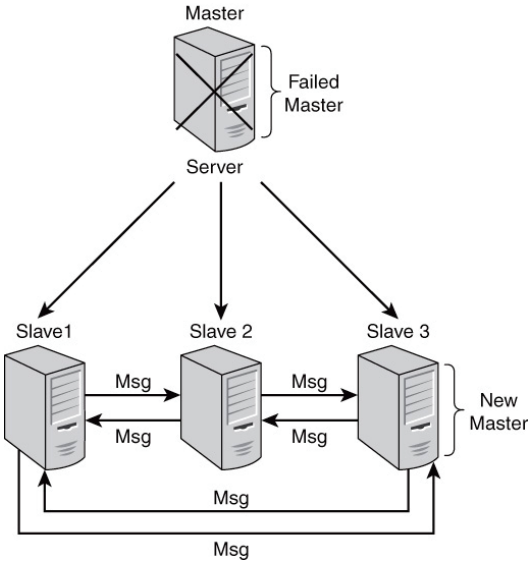






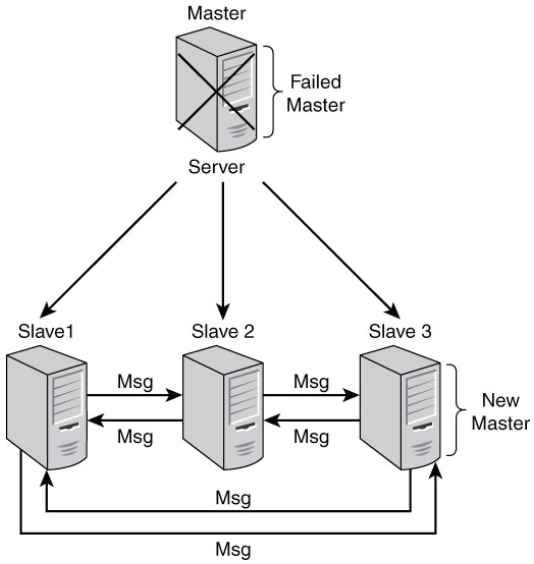
- Memorias asociativas
- Caches
- Bases de datos clave-valor persistentes.

- Simplicidad
- Velocidad
- Escalabilidad



- Simplicidad
- Velocidad
- Escalabilidad

¿Y las propiedades ACID?



- La única forma de buscar valores es por clave.

Range of Values	Assigned Node
cust:00001-cust:00999	Server 1
cust:01000-cust:01999	Server 2
cust:02000-cust:02999	Server 3
cust:04000-cust:04999	Server 4

- Algunas bases de datos clave-valor no admiten consultas por rango.

```
• cust061514:1:custId
• cust061514:2:custId
• cust061514:3:custId
• cust061514:4:custId
• ...
• cust061514:10:custId
```

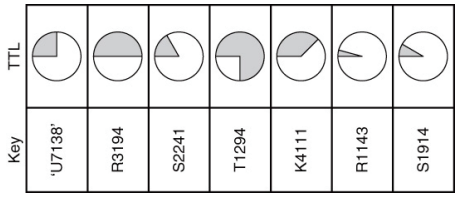
- No existe un lenguaje de consulta estándar comparable a SQL para las bases de datos relacionales.

```
field: { 'motherboard' AND 'computer case') AND NOT
'CPU'
```

- Propiedades ACID

```
ConcertApp[ticketLog:9888] = {'conDate':15-Mar-2015,
  'locDescr':
'Springfield Civic Center', 'assgnSeat': 'J38'}
```

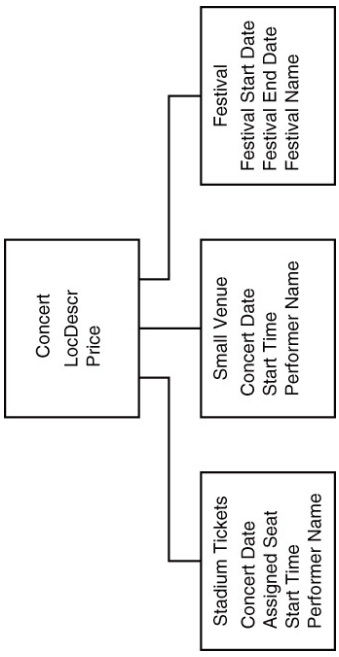
- Tiempo de Vida de claves



- Emulación de tablas

```
define addCustRecord (p_id, p_fname,
p_lname, p_addr,
 p_city, p_state, p_zip)
  begin
     setCustAttr(p_id,'fname', p_fname);
     setCustAttr(p_id,'lname',p_lname);
     setCustAttr(p_id,'addr',p_addr);
     setCustAttr(p_id,'city',p_city);
     setCustAttr(p_id,'state',p_state);
     setCustAttr(p_id,'zip', p_zip);
  end;
```


- Agregados (relaciones is-a)




```
ConcertApp[ticket:18382] = {'type':'festival',
  'festStartDate': 01-Feb-2015, 'festEndDate':
  01-Feb-2015, 'locDescr': 'Portland, OR',
  'price':'$100.00', 'festName':'PDX Jazz
Festival'}

ConcertApp[ticket:18381] = {'type':'small venue',
  'conDate': 12-Jun-2015, 'locDescr': 'Plymoth
Concert  Hall', 'startTime':'17:30',
'price':'$75.00',
  'perfName':'Joshua Redman' }

ConcertApp[ticket:18380] = {'type':'stadium',
  'conDate':15-Mar-2015, 'locDescr': 'Springfield
  Civic Center', 'assgnSeat': 'J38',
'startTime':'17:30',  'price':'$50.00', 'perfName':
'The National' }
```

- Índices invertidos

```
ConcertApp[ticketLog:9888] =
{'conDate':15-Mar-2015,
  'locDescr':
'Springfield Civic Center',
'assgnSeat': 'J38'}
```

```
define addLocAssgnSeat(p_locDescr,
p_seat)
    begin
        v_seatList =
ConcertApp[p_locDescr]
        v_seatList =
append(v_seatList, p_seat)
        ConcertApp[p_locDescr] =
v_seatList
    end;
```

5.2. BBDD No-SQL: BBDD Clave-Valor
Diseño
Tema 5: Bases de datos no relacionales

Estudio de caso: Bases de datos clave-valor para la configuración de aplicaciones móviles

TransGlobal Transport and Shipping (TGTS) coordina el movimiento de mercancías en todo el mundo para empresas de todos los tamaños. Para ayudar a sus clientes a rastrear sus envíos, TGTS está desarrollando una aplicación móvil llamada TGTS Tracker. Esta aplicación debe ejecutarse en las plataformas de dispositivos móviles más populares y mantener información de configuración sobre cada cliente en una base de datos centralizada. La información de configuración incluye:

- nombre del cliente
- número de cuenta
- moneda predeterminada
- atributos de envío
- preferencias de alertas y notificaciones
- opciones de interfaz de usuario.

- Se eligió "TrackerNS" como el nombre del espacio de nombres de la aplicación.
- Cada cliente se identifica mediante un número de cuenta único.
- Los valores se estructuran de manera que el nombre, el número de cuenta y la moneda predeterminada se mantengan juntos en una lista de valores.

```
TrackerNS['cust:4719364'] = {'name':'Prime Machine, Inc.',
  'currency':'USD'}
```

- La base de datos utiliza una convención de nomenclatura de claves que incluye el tipo de entidad seguido del número de cuenta.
- Los tipos de entidad incluyen información del cliente, configuración del panel de control, especificaciones de alertas y notificaciones, y configuraciones de interfaz de usuario.
- Para las especificaciones de configuración del panel de control, se incluyen hasta seis atributos de envío que aparecerán en una pantalla de resumen.

```
TrackerNS['dash:4719364'] =
  {'shpComp','shpState','shpDate','shpDelivDate'}
```

5.2. BBDD No-SQL: BBDD Clave-Valor
Diseño
Tema 5: Bases de datos no relacionales

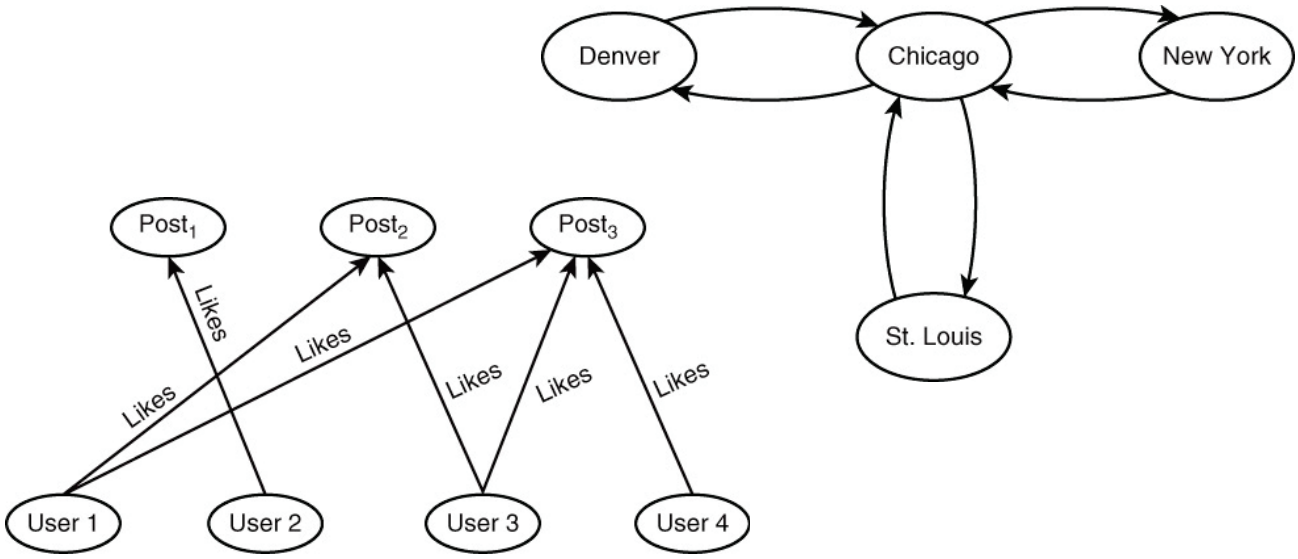
- Para las alertas y notificaciones, se modela un sistema que puede enviar mensajes por correo electrónico o mensajes de texto a múltiples destinatarios bajo diferentes condiciones.

```
TrackerNS[alrt:4719364] =
    { altList :
    {'jane.washingon@primemachineinc.com','pickup'},
    {'(202)555-9812','delay'}
}
```

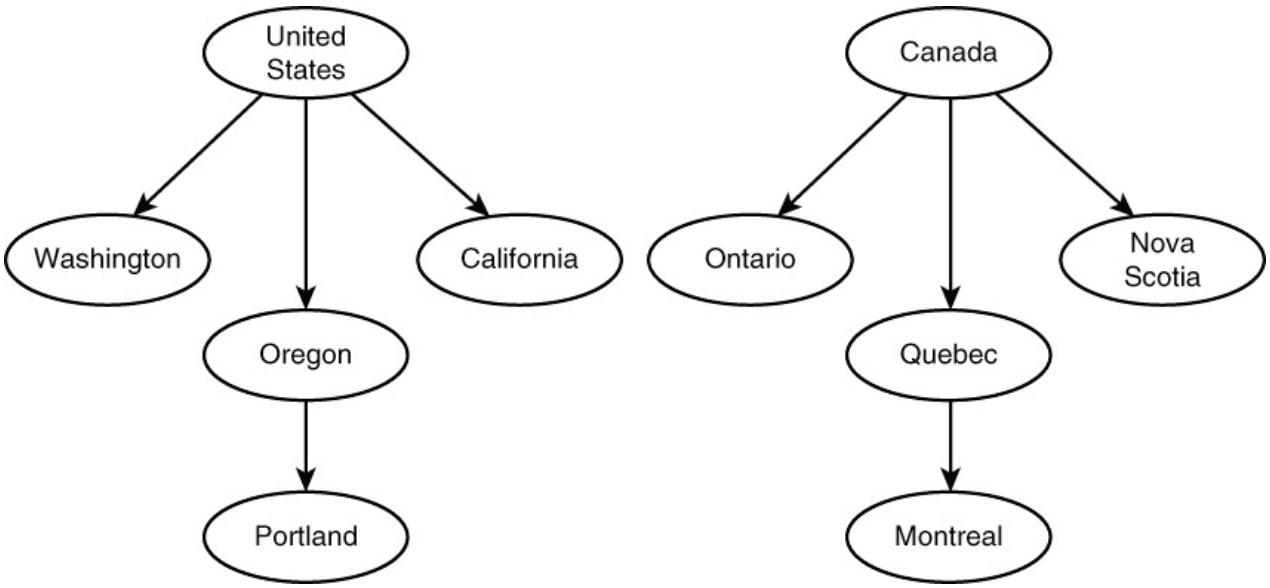
- La configuración de la interfaz de usuario se almacena como pares de atributos y valores.

```
TrackerNS[alrt:4719364] = { 'fontName': 'Cambria',
        'fontSize': 9,
        'colorScheme' : 'default'
   }
```

- Objeto matemático que consta de:
  - vértices o nodos
  - aristas o arcos

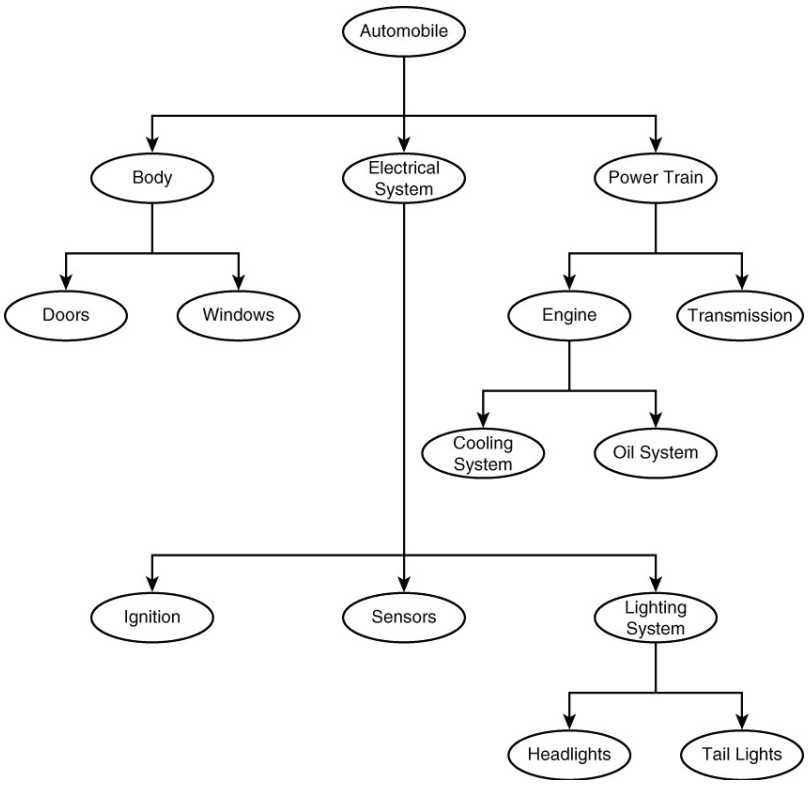


- Jerarquías

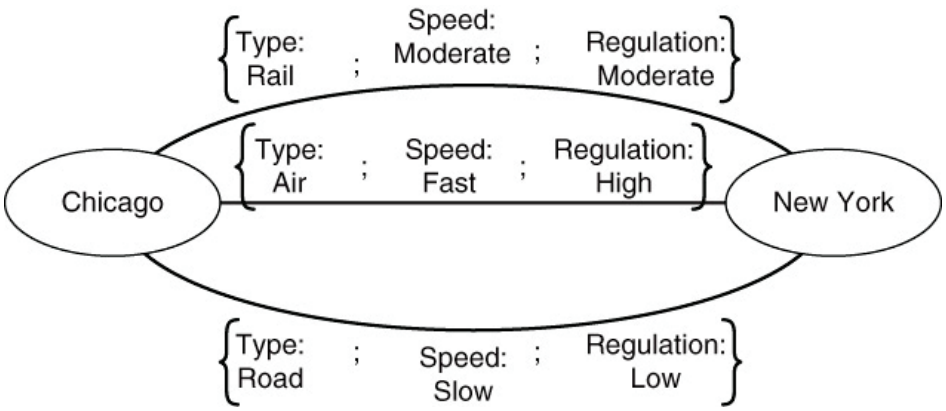


Algunas relaciones útiles

- Parte-de

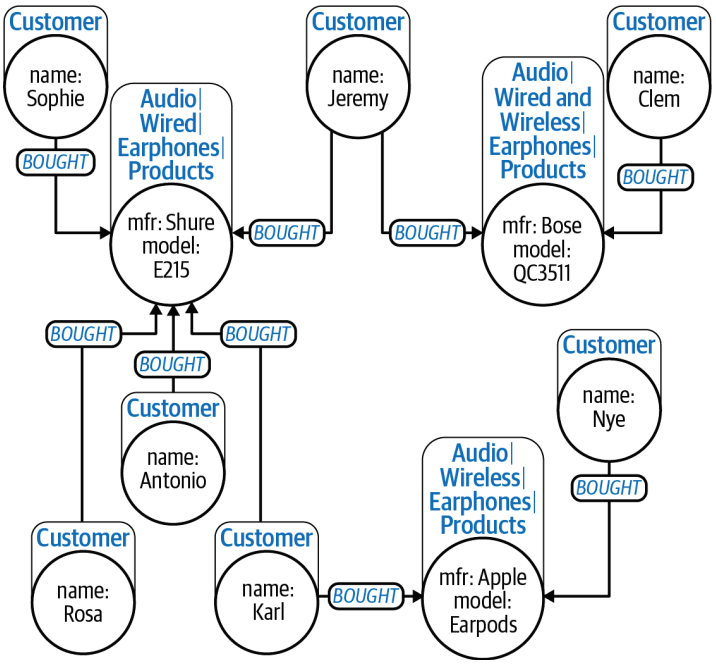


Hiper-grafos o multi-grafos



Grafos de propiedades *(property graphs)*

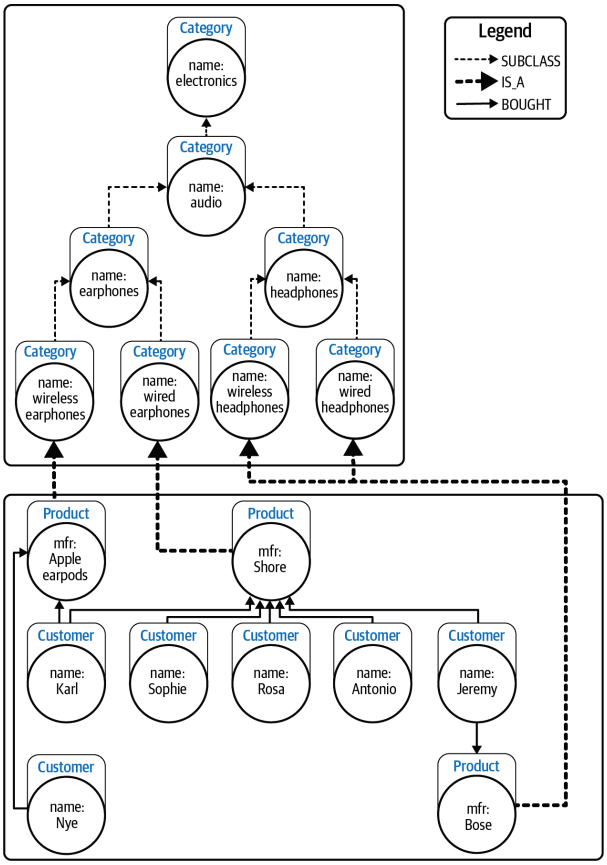
- nodos etiquetados
- tipo y dirección de relaciones
- propiedades (pares clave-valor) tanto en nodos como en relaciones.
- no existe *necesariamente* un esquema



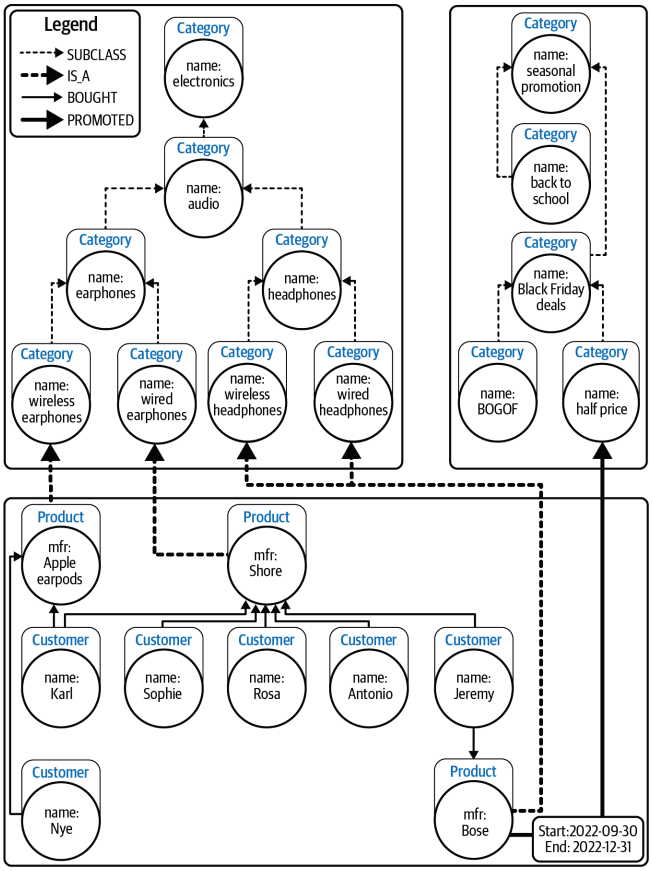
5.3. BBDD basadas en grafos (*graph databases*)
Tema 5: Bases de datos no relacionales

Grafos de propiedades *(property graphs)*

- No existe *necesariamente* un esquema:
  - definición de una o más jerarquías
  - uso de ontologías bien conocidas



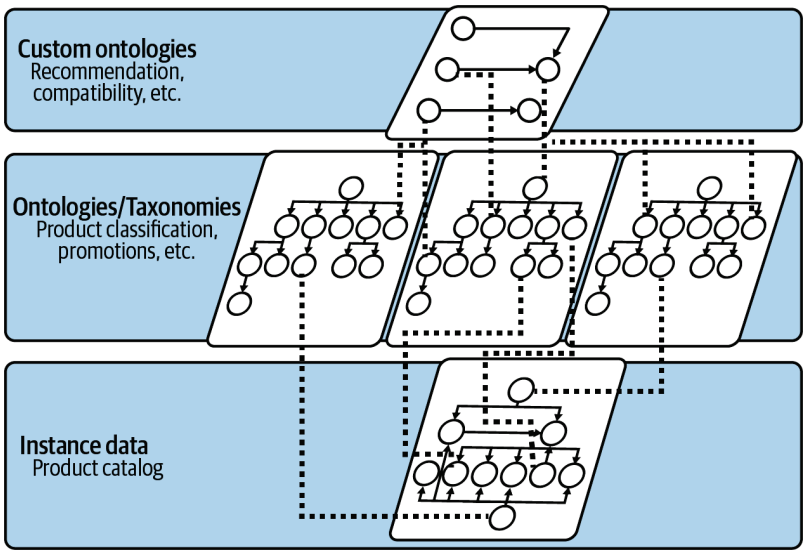
Grafos de propiedades *(property graphs)*



5.3. BBDD basadas en grafos (*graph databases*)

Ontologías

Tema 5: Bases de datos no relacionales



- RDF
- RDF/S
- OWL

- SNOMED CT para documentación e informes clínicos
- Schema.org , ontología compartida para la web
- Dublin Core Metadata Initiative (DCMI) , describir recursos web y más.
- Basic Formal Ontology (BFO), “upper ontology”, pequeña y generalista.

DBMS

- Neo4j
  - Es la más popular
  - Flexible en cuanto al uso de esquemas
  - Recientemente, soporta vectores nativamente (BBDD vectorial)
  - Recursos interesantes:
    - Neo4j community edition. Versión gratuita con varias limitaciones (una única BBDD de tamaño limitado, soporte a solo algunas reglas de integridad)
    - Neo4j Sandbox: herramienta de aprendizaje
    - Neo4j AuraDB: versión SaaS con funcionalidades gratuitas
- TypeDB
  - Obligatorio el uso de un esquema
  - Permite inferir nuevo conocimiento mediante la implementación de reglas.
  - Polimorfismo a nivel de entidad, relación y atributos (polymorphic entity-relation-attribute, PERA)
- Cambridge Semantics
  - Basada en RDF

Neo4j – principales propiedades

- DBMS transaccional
  - Implícitas
    - cada operación CRUD es una transacción
    - CALL { … } IN TRANSACTIONS
  - Explícitas
    - Disponibles desde la consola de comandos (Cypher shell)
    - :begin :commit/:rollback
    - Los cambios se ven por otras sesiones/transacciones una vez son confirmados

Neo4j – principales propiedades

- DBMS transaccional
  - Implícitas
    - cada operación CRUD es una transacción
    - CALL { … } IN TRANSACTIONS
  - Explícitas
    - Disponible[illegible] (Cypher shell)
    - :begin :commit/:rollback
    - Los cambios se ven por otras sesiones/transacciones una vez son confirmados

¿qué nivel de aislamiento se consigue? ¿es recuperable? ¿evita borrados en cascada?

Neo4j – principales propiedades

- DBMS sin esquema (schemaless)
- Se pueden definir restrcciones (*constraints)* allí donde hace falta
- No todos los tipos de restricciones están disponibles para la versión *community*

- Restricciones de propiedad de nodo único
- Restricciones de propiedad de relación única

- Restricciones de existencia de propiedad de nodo
- Restricciones de existencia de propiedad de relación

- Restricciones de tipo de propiedad de nodo
- Restricciones de tipo de propiedad de relación:

- https://neo4j.com/docs/cypher-manual/current/constraints/

Neo4j – principales propiedades

- DBMS sin esquem
- Se pueden defini e hace falta
- No todos los tipos de restricciones están onibles para la versión *community*

Restricciones de identidad, de valores no nulos y de dominio ¿schemaless?

- **Restricciones de propiedad de nodo único**
- **Restricciones de propiedad de relación única**

- **Restricciones de existencia de propiedad de nodo**
- **Restricciones de existencia de propiedad de relación**

- **Restricciones de tipo de propiedad de nodo**
- **Restricciones de tipo de propiedad de relación**:

- https://neo4j.com/docs/cypher-manual/current/constraints/

Neo4j – principales propiedades

- DBMS transaccional
  - Implícitas
    - cada operación CRUD es una transacción
    - CALL { … } IN TRANSACTIONS
  - Explícitas
    - Disponibles desde l ndos (Cypher shell)
    - :begin :commit/:rollback
    - Los cambios se ven por otras sesiones/transacciones una vez son confirmados

¿Nivel de aislamiento?

Neo4j – Transacciones implícitas

Query

```
UNWIND [1, 0, 2, 4] AS i
CALL {
  WITH i
  CREATE (n:Person {num: 100/i}) // Note, fail
  RETURN n
} IN TRANSACTIONS
  OF 2 ROWS
  ON ERROR CONTINUE
RETURN n.num;
```

commit cada 2 filas

Table 8. Result

n.num
null
null
50
25

Rows: 4

5.3. BBDD basadas en grafos (*graph databases*)

Neo4j – Transacciones implícitas

Tema 5: Bases de datos no relacionales

Query

```
UNWIND [1, 0, 2, 4] AS i
CALL {
  WITH i
  CREATE (n:Person {num: 100/i}) // Note, fails when i
  RETURN n
} IN TRANSACTIONS
  OF 2 ROWS
  ON ERROR CONTINUE
RETURN n.num;
```

ON ERROR CONTINUE
ON ERROR BREAK
ON ERROR FAIL

Table 8. Result

n.num
null
null
50
25

Rows: 4

Neo4j – Transacciones implícitas


```
LOAD CSV FROM 'file:///friends.csv' AS line
CALL {
  WITH line
  CREATE (:Person {name: line[1], age: toInteger(line[2])})
} IN TRANSACTIONS
  OF 2 ROWS
  ON ERROR CONTINUE
```

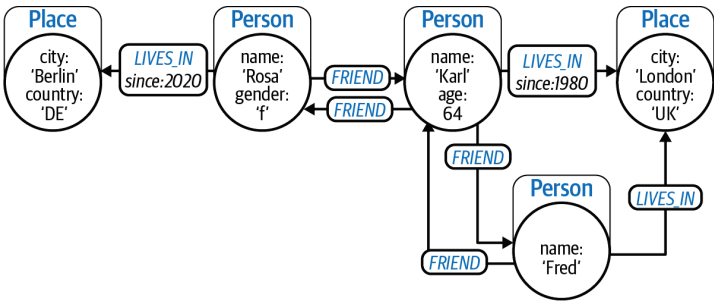
5.3. BBDD basadas en grafos (*graph databases*)

Neo4j -- Transacciones implícitas

Tema 5: Bases de datos no relacionales

```
LOAD CSV FROM 'file:///friends.csv' AS line
CALL {
  WITH line
  CREATE (:Person {name: line[1], age: toInteger(line[2])})
} IN TRANSACTIONS
  OF 2 ROWS
  ON ERROR CONTINUE
```

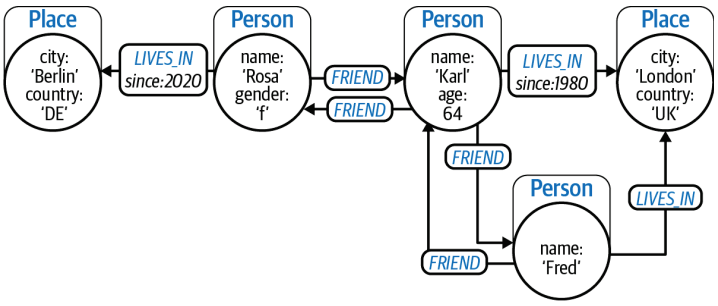
- Cada transacción está compuesta por un máximo de 2 filas
- Si una transacción falla, solo se deshará esa transacción, y continua con las siguientes



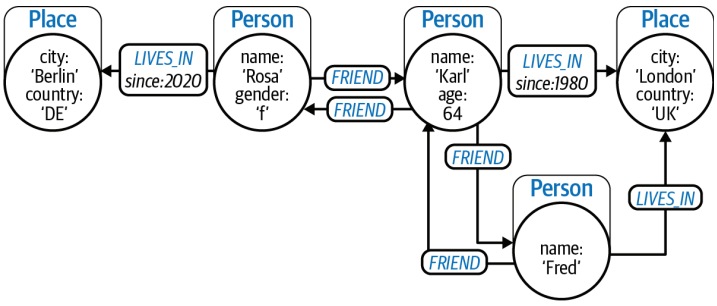

```
{name:'Rosa'})-[:LIVES_IN]->(:Place {city:'Berlin', country:'DE'})
```

- Los elementos entre `()` paréntesis representan nodos.
- El texto `:Person` y `:Place` representa etiquetas en esos nodos.
- `-[:LIVES_IN]->` es una relación nombrada `LIVES_IN` entre personas y lugares.
- `{name:'Rosa'}` y `{city:'Berlin', country:'DE'}` son propiedades clave-valor que se pueden almacenar en nodos (y relaciones).

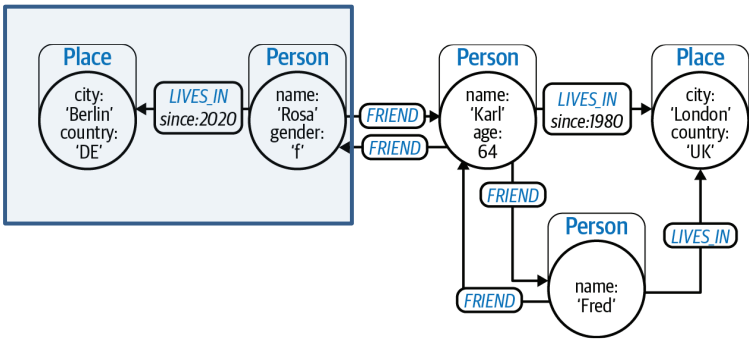
5.3. BBDD basadas en grafos (*graph databases*)
Neo4j -- Cypher
Tema 5: Bases de datos no relacionales



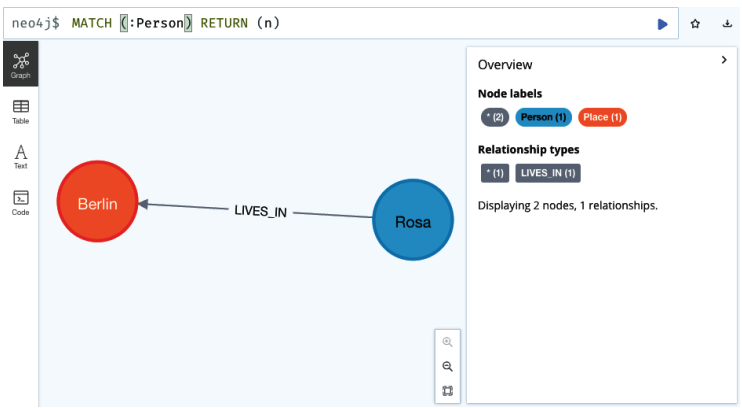

```
CREATE (:Person {name:'Rosa'})-[:LIVES_IN {since:2020}]-> (:Place {city:'Berlin', country:'DE'})
```

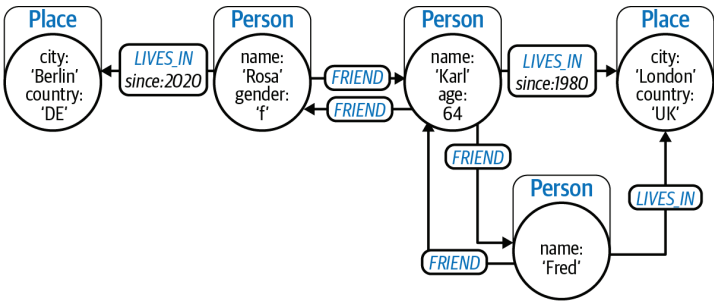



```
CREATE (:Person {name:'Rosa'})-[:LIVES_IN {since:2020}]-> (:Place {city:'Berlin', country:'DE'})

MATCH (p:Person) WHERE p.name = 'Rosa' SET p.dob = 19841203
MATCH (p:Person) WHERE p.name = 'Rosa' REMOVE p.dob
MATCH (p:Person) WHERE p.name = 'Rosa' REMOVE p:Person
```

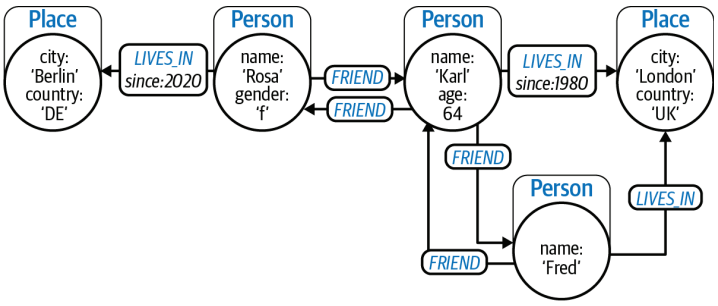

```
MATCH (n:Person) RETURN n
```





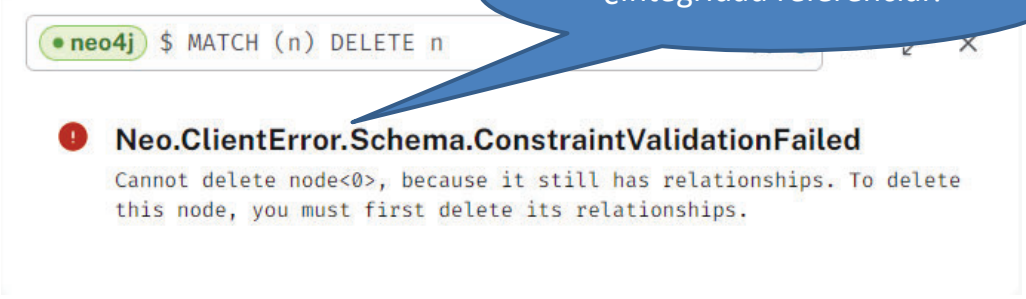

```
MATCH (n) DELETE n // Elimina nodos
                   // Aborta si deja las relaciones pendientes.
```

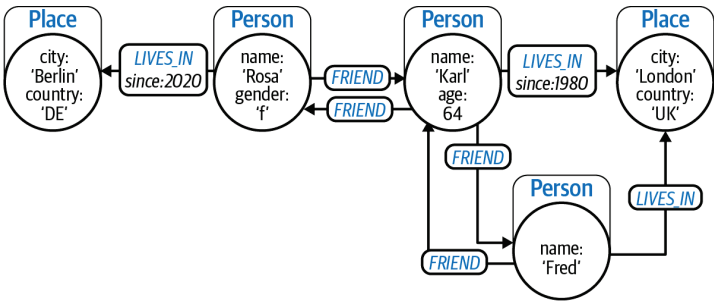
5.3. BBDD basadas en grafos (*graph databases*)
Neo4j -- Cypher
Tema 5: Bases de datos no relacionales




```
MATCH (n) DELETE n // Elimina nodos
                   // Aborta si deja las relacione
```

¿Integridad referencial?

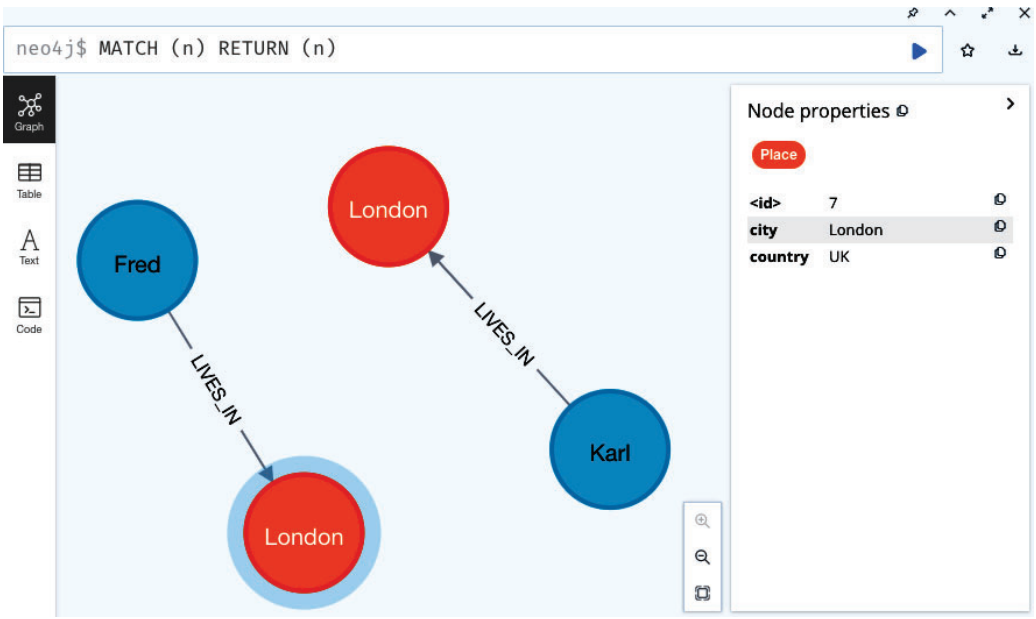





```
MATCH ()-[r:LIVES_IN]->() // Elimina las relaciones LIVES_IN entre cualquier nodo
DELETE r
MATCH (n:Place) DELETE n // Elimina nodos
                         // Aborta limpiamente si deja las relaciones pendientes.


MATCH (n) DETACH DELETE n // Elimina todos los nodos y cualquier relación adjunta,
                          // eliminando efectivamente el gráfico completo.
```



```
CREATE (:Person {name:'Karl', age:64})-[:LIVES_IN {since:1980}]
→(:Place {city:'London', country:'UK'})

CREATE (:Person {name:'Fred'})-[:LIVES_IN]
→(:Place {city:'London', country:'UK'})
```



Neo4j – Restricción de identidad

```
CREATE CONSTRAINT no_duplicate_cities FOR (p:Place)
REQUIRE (p.country, p.city) IS NODE KEY
```

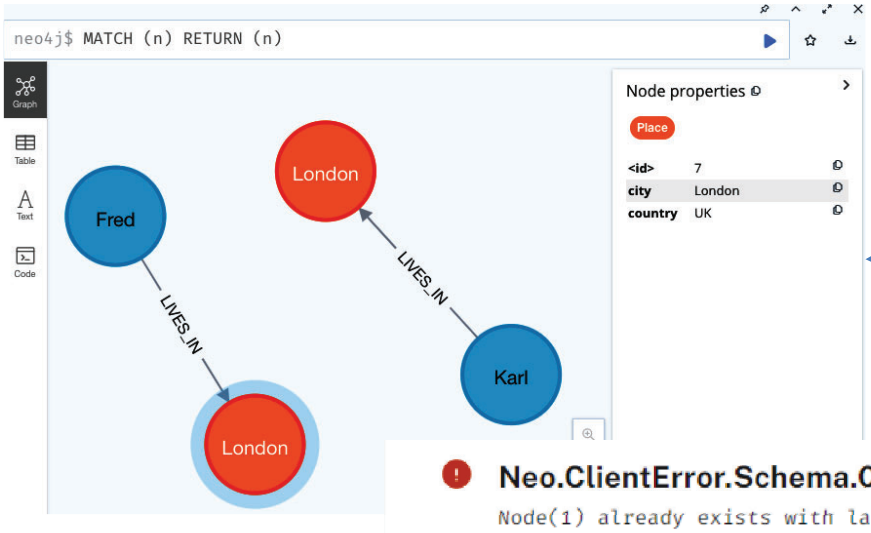
```
CREATE (:Person {name:'Karl', age:64})-[:LIVES_IN {since:1980}]
→(:Place {city:'London', country:'UK’})

CREATE (:Person {name:'Fred'})-[:LIVES_IN]
→(:Place {city:'London', country:'UK'})
```

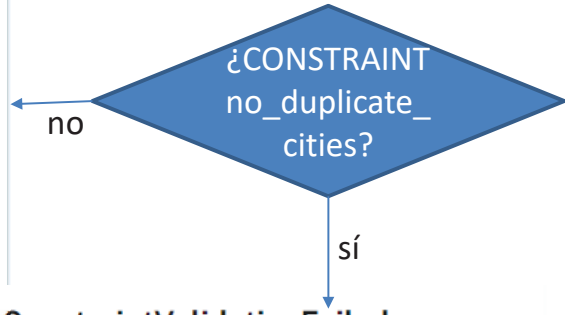
5.3. BBDD basadas en grafos (*graph databases*)
Neo4j – Restricción de identidad
Tema 5: Bases de datos no relacionales

```
MERGE (:Person {name:'Karl', age:64})-[:LIVES_IN {since:1980}]
→(:Place {city:'London', country:'UK’})

MERGE (:Person {name:'Fred'})-[:LIVES_IN]
→(:Place {city:'London', country:'UK'})
```

Neo4j – Restricción de identidad


```
MERGE (:Person {name:'Karl', age:64})-[:LIVES_IN {since:1980}]
→(:Place {city:'London', country:'UK’})

MERGE (:Person {name:'Fred'})-[:LIVES_IN]
→(:Place {city:'London', country:'UK'})
```





Neo.ClientError.Schema.ConstraintValidationFailed

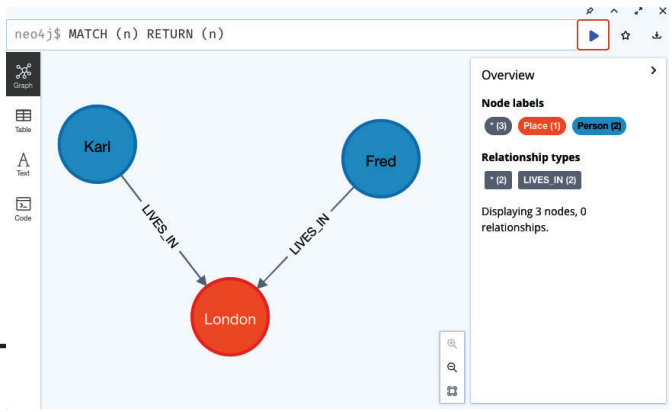
```
Node(1) already exists with label `Place` and properties `country` =
'UK', `city` = 'London'
```


5.3. BBDD basadas en grafos (*graph databases*)

Neo4j – Restricción de identidad

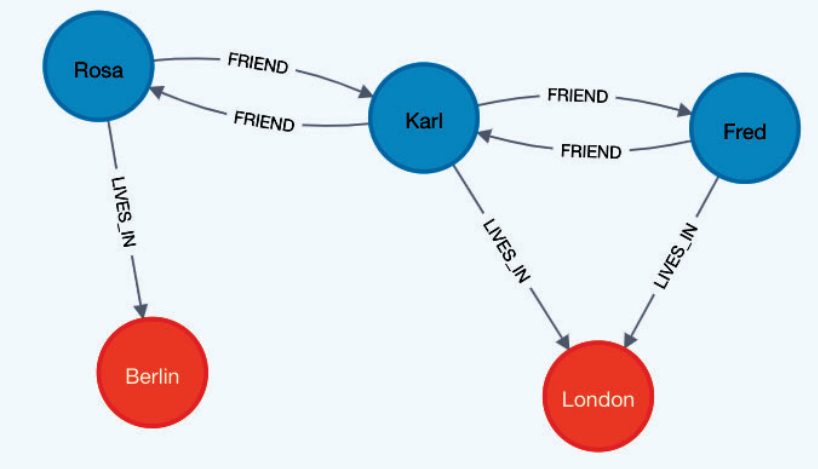
Tema 5: Bases de datos no relacionales

```
MERGE (london:Place {city:'London’, country:’UK’}) // Crea o relaciona un nodo con
                                                   // representar a Londres, Reino Unido
                                                  // Lo vincula a la variable "Londres"

MERGE (fred:Person {name:'Fred'}) // Crea o combina un nodo para representar a Fred
                                  // Lo vincula a la variable "fred"

MERGE (fred)-[:LIVES_IN]->(london) // Crea o combina una relación LIVES_IN
                                   // entre los nodos fred y london

MERGE (karl:Person {name:'Karl'}) // Crea o une un nodo para representar a Karl
                                  // Lo vincula a la variable "karl"

MERGE (karl)-[:LIVES_IN]->(london) // Crea o combina una relación LIVES_IN
                                   // entre los nodos karl y london
```



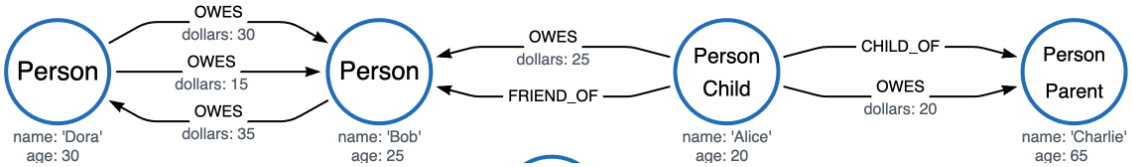
Neo4j -- Consultas

```
MATCH (p:Person {name:'Rosa'})
MATCH (p2:Person {name:'Fred'})
MERGE (p)-[:FRIEND]->(p2)

MATCH (p2:Person {name:'Fred'})
MATCH (p:Person {name:'Karl'})
MERGE (p)-[:FRIEND]->(p2)

MATCH (rosa:Person {name:'Rosa'})-[:FRIEND*2..2]->(fof:Person)
WHERE rosa <> fof
RETURN (fof)

MATCH . . .
```



5.3. BBDD basadas en grafos (*graph databases*)
Neo4j -- Consultas
Tema 5: Bases de datos no relacionales

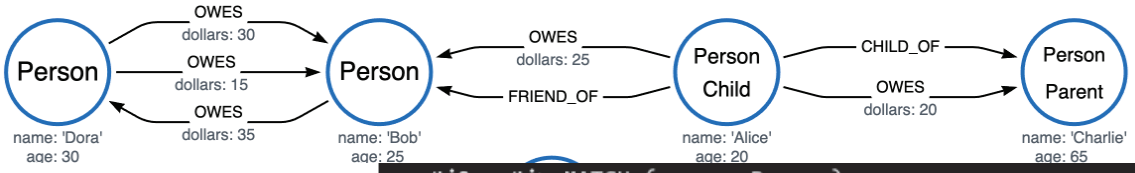
```
MATCH (p:Person)-[:LIVES_IN]->(:Place {city:'Berlin', country:'DE'})
RETURN (p)

MATCH (p:Person)-[:LIVES_IN]->(pl:Place {country:'DE'})
RETURN p.name AS NAME, pl.city as city


MATCH (:Place {city:'Berlin'})<-[:LIVES_IN]-(p:Person)<-[:FRIEND*1..2]-(f:Person)
WHERE f <> p
RETURN f

MATCH (p:Person)-[l:LIVES_IN]->(pl:Place)
            RETURN  pl.city as city, count (l) as rels order by rels desc
```




```
CREATE
 (a:Person:Child {name: 'Alice', age: 20}),
 (b:Person {name: 'Bob', age: 27}),
 (c:Person:Parent {name: 'Charlie', age: 65}),
 (d:Person {name: 'Dora', age: 30})
 CREATE (a)-[:FRIEND_OF]->(b)
 CREATE (a)-[:CHILD_OF]->(c)
 CREATE (a)-[:OWES {dollars: 20}]->(c)
 CREATE (a)-[:OWES {dollars: 25}]->(b)
 CREATE (b)-[:OWES {dollars: 35}]->(d)
 CREATE (d)-[:OWES {dollars: 15}]->(b)
 CREATE (d)-[:OWES {dollars: 30}]->(b)
```

```
neo4j@neo4j> MATCH
               (p1:Person {name: 'Dora'}),
               (p2:Person {name: 'Charlie'}),
               p = shortestPath((p1)-[*]-(p2))
             WHERE length(p) > 1
             RETURN length(p)
             ;
+-----------+
| length(p) |
+-----------+
| 3         |
+-----------+
```


5.3. BBDD basadas en grafos (*graph databases*)
Neo4j – Procedimientos almacenados
Tema 5: Bases de datos no relacionales




```
neo4j@neo4j> MATCH (person:Person)
             WITH person ORDER BY person.age ASC LIMIT 1
               SET person:ListHead
             WITH *
             MATCH (nextPerson: Person&!ListHead)
             WITH nextPerson ORDER BY nextPerson.age
             CALL {
               WITH nextPerson
               MATCH (current:ListHead)
                 REMOVE current:ListHead
                 SET nextPerson:ListHead
                 CREATE(current)-[:IS_YOUNGER_THAN]->(nextPerson)
               RETURN current AS from, nextPerson AS to
             }
             RETURN
               from.name AS name,
               from.age AS age,
               to.name AS closestOlderName,
               to.age AS closestOlderAge;
+-----------------------------------------------------------+
| name      | age | closestOlderName | closestOlderAge |
+-----------------------------------------------------------+
| "Alice"   | 20  | "Bob"            | 27              |
| "Bob"     | 27  | "Dora"           | 30              |
| "Dora"    | 30  | "Charlie"        | 65              |
| "Charlie" | 65  | NULL             | NULL            |
+-----------------------------------------------------------+
```

Typeless vs Typed


```
class User {
    String email;
    String name;

    User(String email, String name) {
        this.name = name;
        this.email = email;
    }
}

class Employee extends User {
    int employeeId;

    Employee(String email, String name, int employeeId) {
        super(email, name);
        this.employeeId = employeeId;
    }
}

class PartTimeEmployee extends Employee {
    int weeklyHour;

    PartTimeEmployee(String email, String name,
                     int employeeId, int weeklyHour) {
        super(email, name, employeeId);
        this.wheeklyHour = weeklyHour;
    }
}

// Class instantiation in Java

PartTimeEmployee john =
    new PartTimeEmployee("john.doe@vaticle.com",
                         "John Doe", 346523, 35);
```

5.3. BBDD basadas en grafos (*graph databases*)

Typeless vs Typed: SQL


```
class User {
    String email;
    String name;

    User(String email, String name) {
        this.name = name;
        this.email = email;
    }
}

class Employee extends User {
    int employeeId;

    Employee(String email, String name, int employeeId) {
        super(email, name);
        this.employeeId = employeeId;
    }
}

class PartTimeEmployee extends Employee {
    int weeklyHour;

    PartTimeEmployee(String email, String name,
                     int employeeId, int weeklyHour) {
        super(email, name, employeeId);
        this.wheeklyHour = weeklyHour;
    }
}

// Class instantiation in Java

PartTimeEmployee john =
    new PartTimeEmployee("john.doe@vaticle.com",
                         "John Doe", 346523, 35);
```

```
SELECT
    Users.id AS id,
    Users.email AS email,
    Users.name AS name,
    null AS employeeId,
    null AS weeklyHours,
    "user" AS userType
FROM Users
WHERE NOT EXISTS (
    SELECT id FROM Employees WHERE Users.id = Employees.id
)
UNION ALL
SELECT
    Users.id AS id,
    Users.email AS email,
    Users.name AS name,
    employeeId,
    null AS weeklyHours,
    "employee" AS userType
FROM Users
INNER JOIN Employees ON Users.id = Employees.id
WHERE NOT EXISTS (
    SELECT Employees.id
    FROM PartTimeEmployee
    WHERE Employees.id = PartTimeEmployee.id
)
UNION ALL
SELECT
    Users.id AS id,
    Users.email AS email,
    Users.name AS name,
    employeeId,
    weeklyHours,
    "partTimeEmployee" AS userType
FROM Users
INNER JOIN Employees ON Users.id = Employees.id
INNER JOIN PartTimeEmployee ON Employees.id = PartTimeEmployee.id
```

Typeless vs Typed: Neo4j


```
class User {
    String email;
    String name;

    User(String email, String name) {
        this.name = name;
        this.email = email;
    }
}

class Employee extends User {
    int employeeId;

    Employee(String email, String name, int employeeId) {
        super(email, name);
        this.employeeId = employeeId;
    }
}

class PartTimeEmployee extends Employee {
    int weeklyHour;

    PartTimeEmployee(String email, String name,
                     int employeeId, int weeklyHour) {
        super(email, name, employeeId);
        this.wheeklyHour = weeklyHour;
    }
}

// Class instantiation in Java

PartTimeEmployee john =
    new PartTimeEmployee("john.doe@vaticle.com",
                         "John Doe", 346523, 35);
```

```
MATCH
    (user:User)-[:OWNS]->(rsrc:Resource)
WITH
    rsrc,
    [user.primary_email] + user.alias_emails AS emails,
    labels(rsrc) AS resource_types,
    keys(rsrc) AS properties
UNWIND emails AS email
UNWIND resource_types AS resource_type
WITH
    rsrc, email, resource_type, properties,
    {
        File: "path",
        Directory: "path",
        Commit: "hash",
        Repository: "name",
        Table: "name",
        Database: "name"
    } AS id_type_map
WHERE resource_type IN keys(id_type_map)
AND id_type_map[resource_type] IN properties
RETURN email,
       resource_type,
       rsrc[id_type_map[resource_type]] AS id
```

5.3. BBDD basadas en grafos (*graph databases*)

Typeless vs Typed: TypeDB


```
class User {
    String email;
    String name;

    User(String email, String name) {
        this.name = name;
        this.email = email;
    }
}

class Employee extends User {
    int employeeId;

    Employee(String email, String name, int employeeId) {
        super(email, name);
        this.employeeId = employeeId;
    }
}

class PartTimeEmployee extends Employee {
    int weeklyHour;

    PartTimeEmployee(String email, String name,
                     int employeeId, int weeklyHour) {
        super(email, name, employeeId);
        this.wheeklyHour = weeklyHour;
    }
}

// Class instantiation in Java

PartTimeEmployee john =
    new PartTimeEmployee("john.doe@vaticle.com",
                         "John Doe", 346523, 35);
```

```
define

id sub attribute, abstract, value string;
email sub id;
name sub id;
path sub id;

user sub entity;
admin sub user;
user-group sub entity;
resource sub entity, abstract;
file sub resource;
```

```
ownership sub relation, abstract,
    relates owned,
    relates owner;
group-ownership sub ownership,
    relates group as owned;
resource-ownership sub ownership,
    relates resource as owned;
```