Bases de données graphe

Stefania Dumbrava

ENSIIE

11 Décembre 2020

Evolution des SGBDs

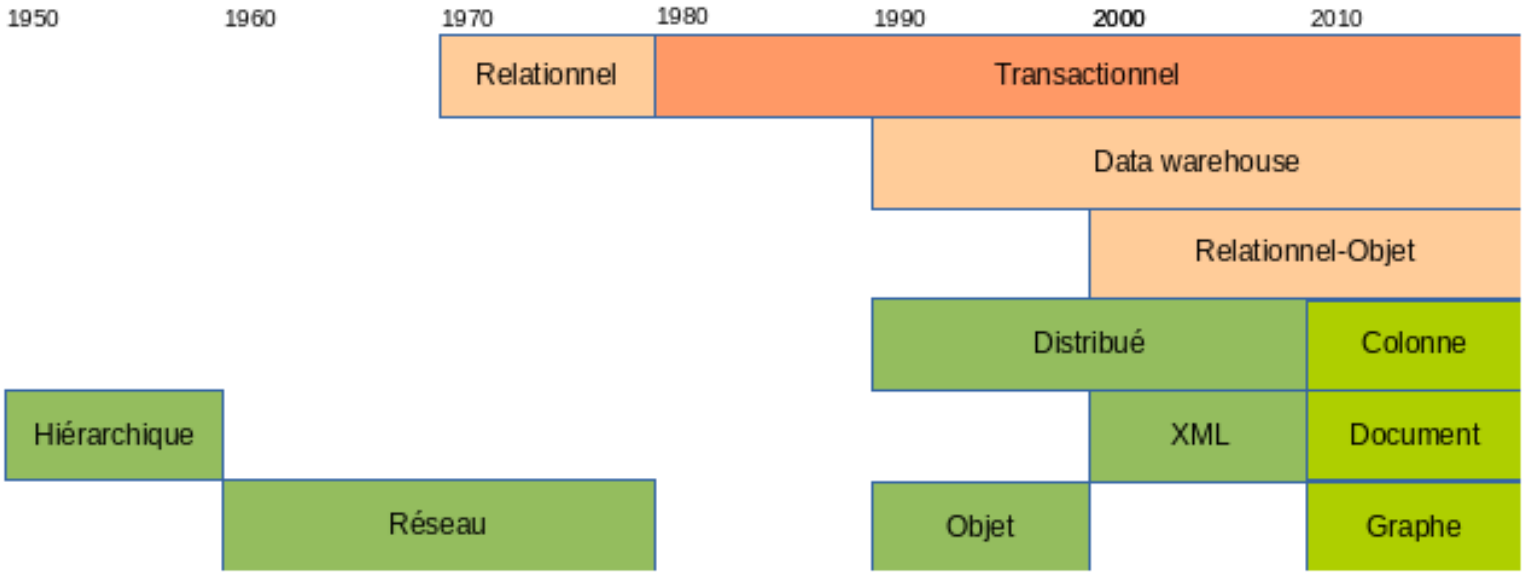


Figure: Évolution des SGBD

Bases de données NoSQL

- très grandes plateformes et applications Web (Google, Facebook, Twitter, LinkedIn, Amazon)
- volume considérable de données (*distribution*) → Data Centers
- données souvent associées à des objets *complexes* et *hétérogènes*

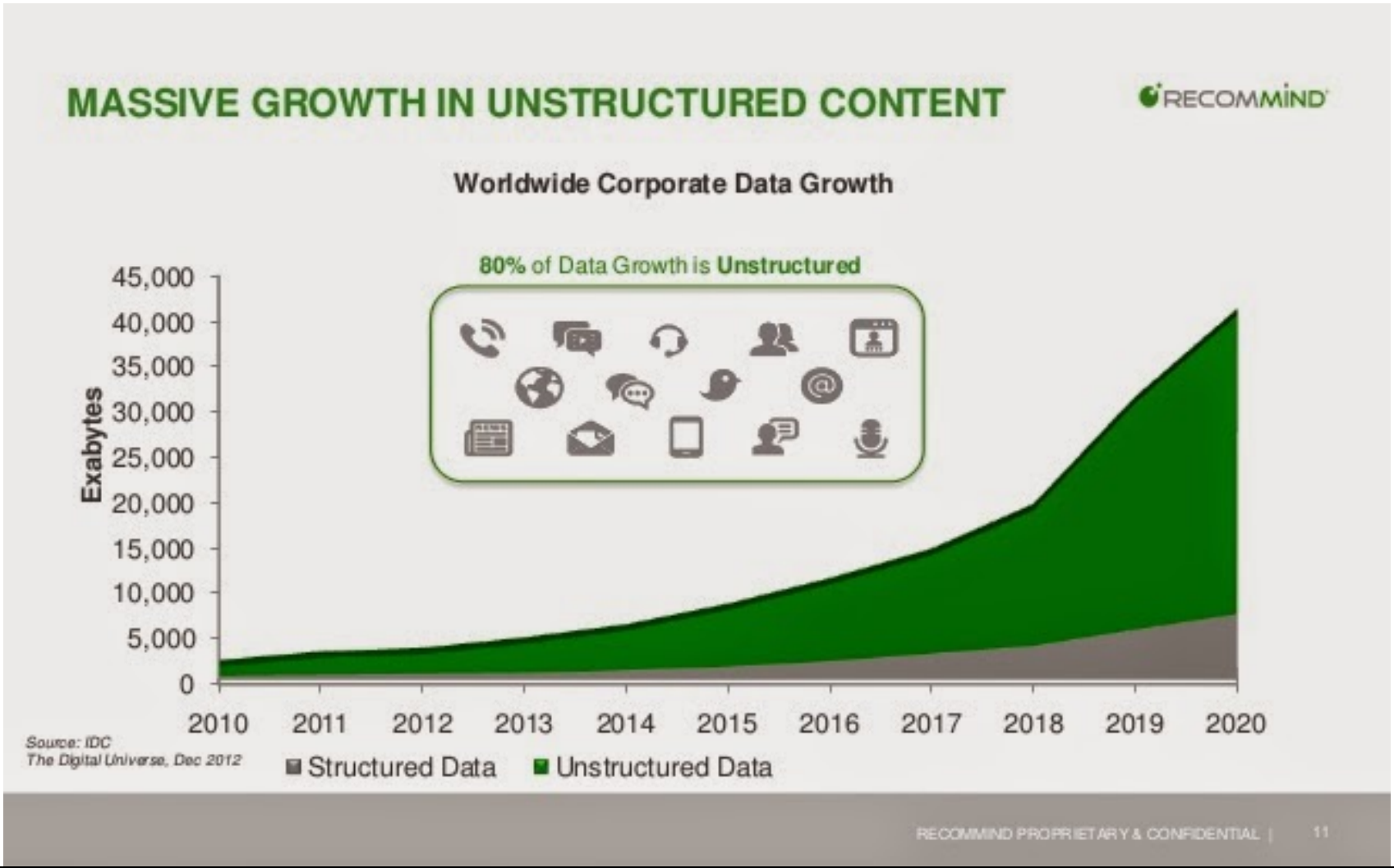
⇒ nouvelles approches de stockage et gestion de données pour:

- une meilleure scalabilité dans un contexte distribué
- une manipulation d'objets complexes et hétérogènes

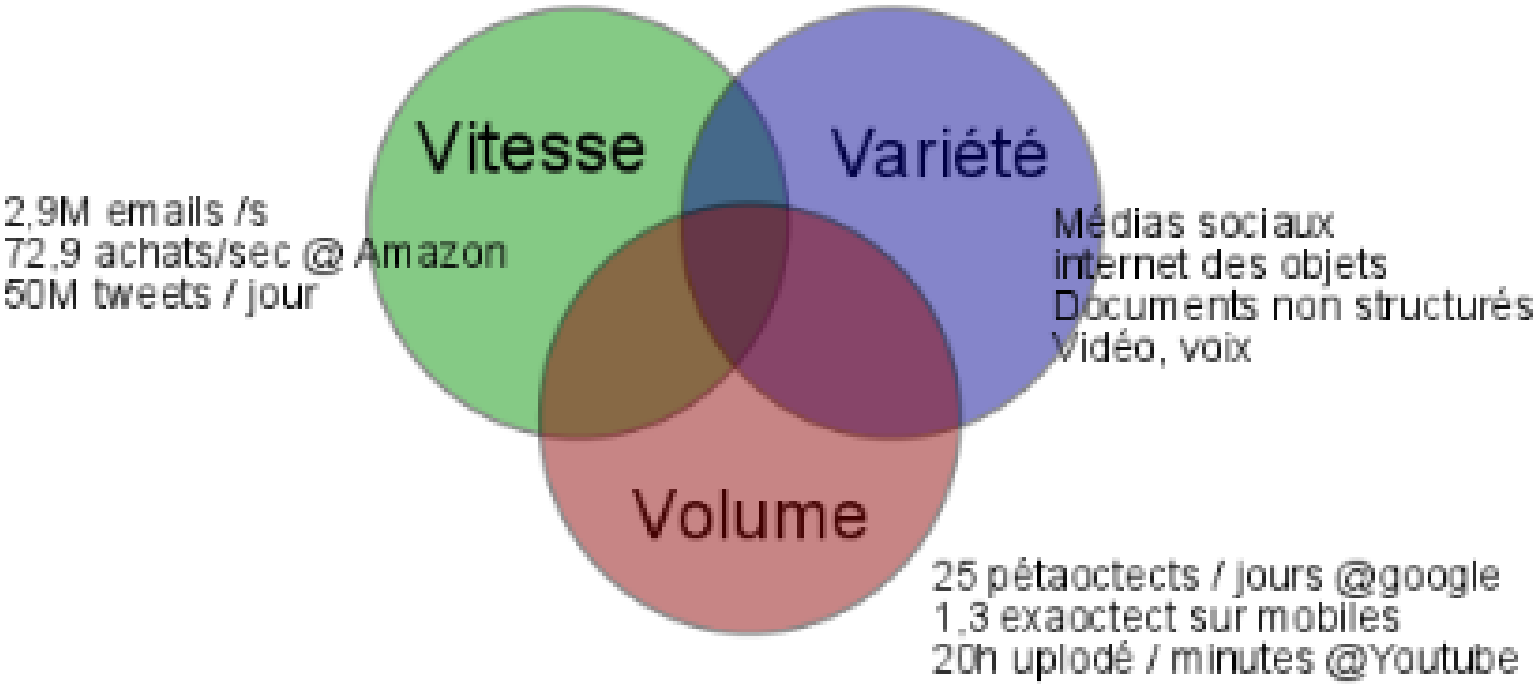
...pas en substituant les SGBD Relationnels, mais les complétant et en comblant leurs faiblesses (Not Only SQL)

Motivation

Big Data : ≈ 2,5 trillions d'octets de données/jour



Motivation


Vitesse
Variété
Volume
2,9M emails /s
72,9 achats/sec @ Amazon
50M tweets / jour
Médias sociaux
internet des objets
Documents non structurés
Vidéo, voix
25 pétaoctects / jours @google
1,3 exaoctect sur mobiles
20h uplodé / minutes @Youtube

Motivation

- **Volume** :
  - le maintien des propriétés ACID (Atomicity, Consistency, Isolation, Durability) est *coûteux* et *pas toujours nécessaire*

  → permettre le partitionnement des données sur plusieurs sites
- **Variété** :
  - modèle unique → *difficile d'intégrer de données hétérogènes*
  - sources externes → schéma *potentiellement inconnu*
  - modifier un schémas d'un SGBD relationnel → *coûteux*

  → permettre une approche schema-less
- **Vitesse** :
  - stocker tout sur un disque *peut être extrêmement coûteux*

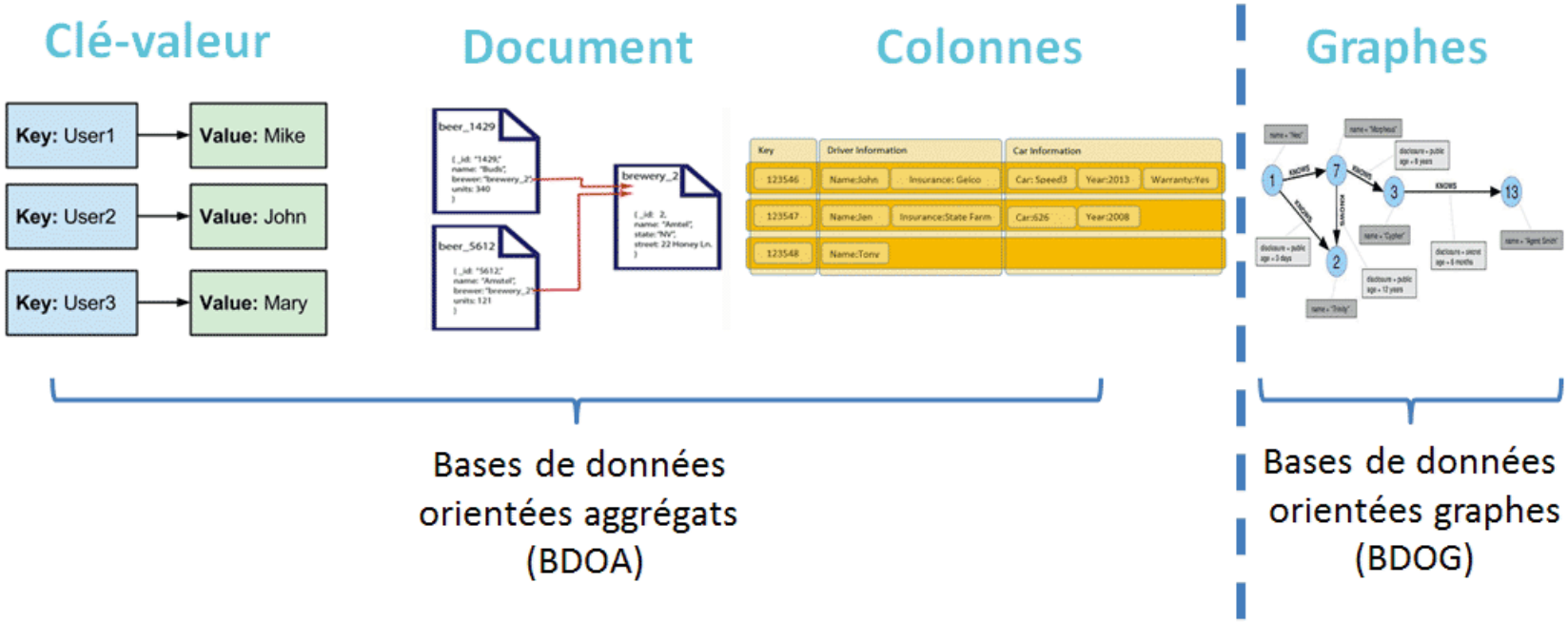
  → permettre le traitement des données en mémoire

SGBDs NoSQL

- pas de schéma fixe
- pas de langage de requêtes standard
- structure des données hétérogène et évolutive
- données complexes et pas toujours renseignées
- plus grande performance dans le contexte des applications Web (volumétries de données exponentielles)
- forte distribution des données et traitement sur de nombreux serveurs
- open source

SGBDs NoSQL

- retour aux modèles de données pré-relationnelles : e.g., représentations hiérarchiques/réseaux



Modèle Graphe

Briques de base

- nœuds : entités abstraites
- relations/arêtes : liens orientés entre les nœuds
- propriétés/attributs : étiquettes sur un nœud **ou** sur une relation

Modèle Relationnelle vs. Modèle Graphe

- tables → ensemble de nœuds et des arêts
- tuples → nœuds
- colonnes → paires clef-valeur
- jointures → chemins

Modèle Graphe : Applications

- résaux sociaux
- résaux de transport
- services financiers (détection des fraudes)
- moteurs de recommandation
- web sémantique
- internet des objets
- sciences de la vie (génomique),...

Modèle Graphe

Limitations du modèle graphe

- gros volumes de données → répartition des données problématique

Modèle PGM Canonique

Multi-graphe étiquetté, orienté, avec des attributs

- graphe d'objets "riches"
- les noeuds **et** les arêtes peuvent avoir des éttiquetes (typage)
- les noeuds **et** les arêtes peuvent avoir des propriétés (paires clé-valeur)

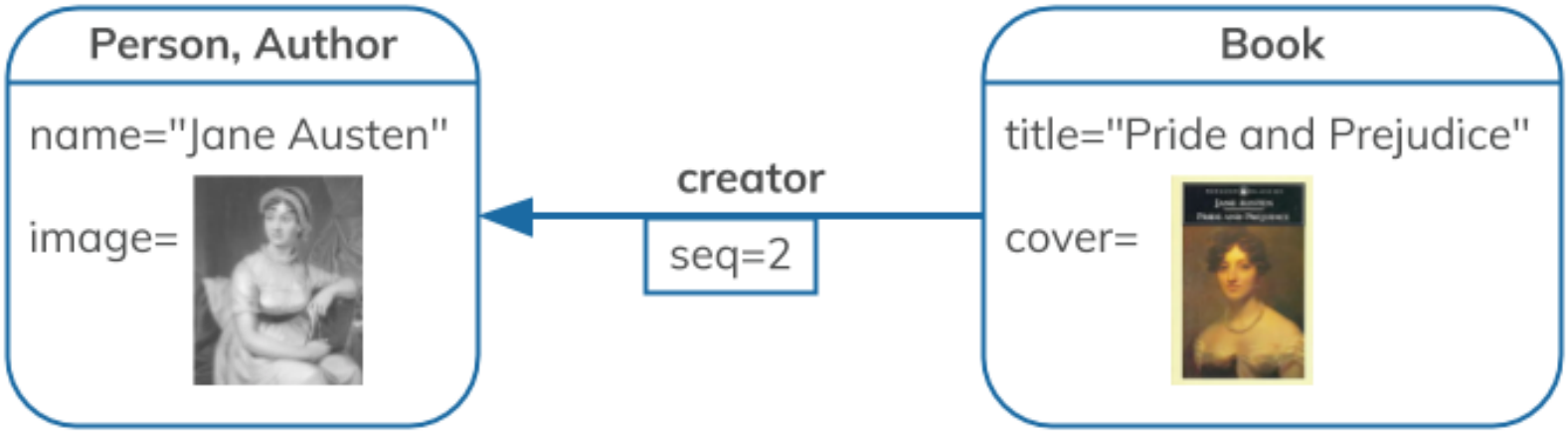


Figure: Exemple de Graphe de Propriétés

SGBDs Graphe

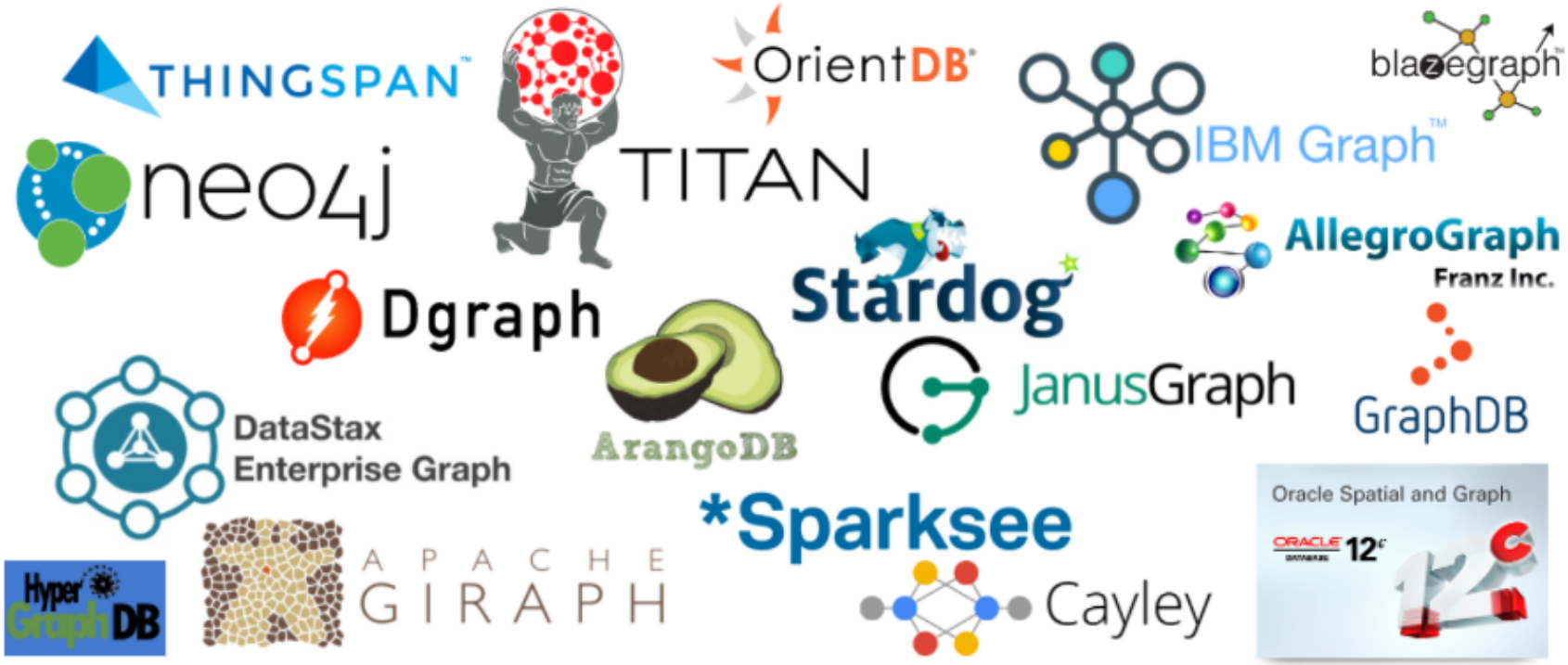


Figure: Ecosystème des SGBDs Graphe

SGBDs Graphe : Neo4j

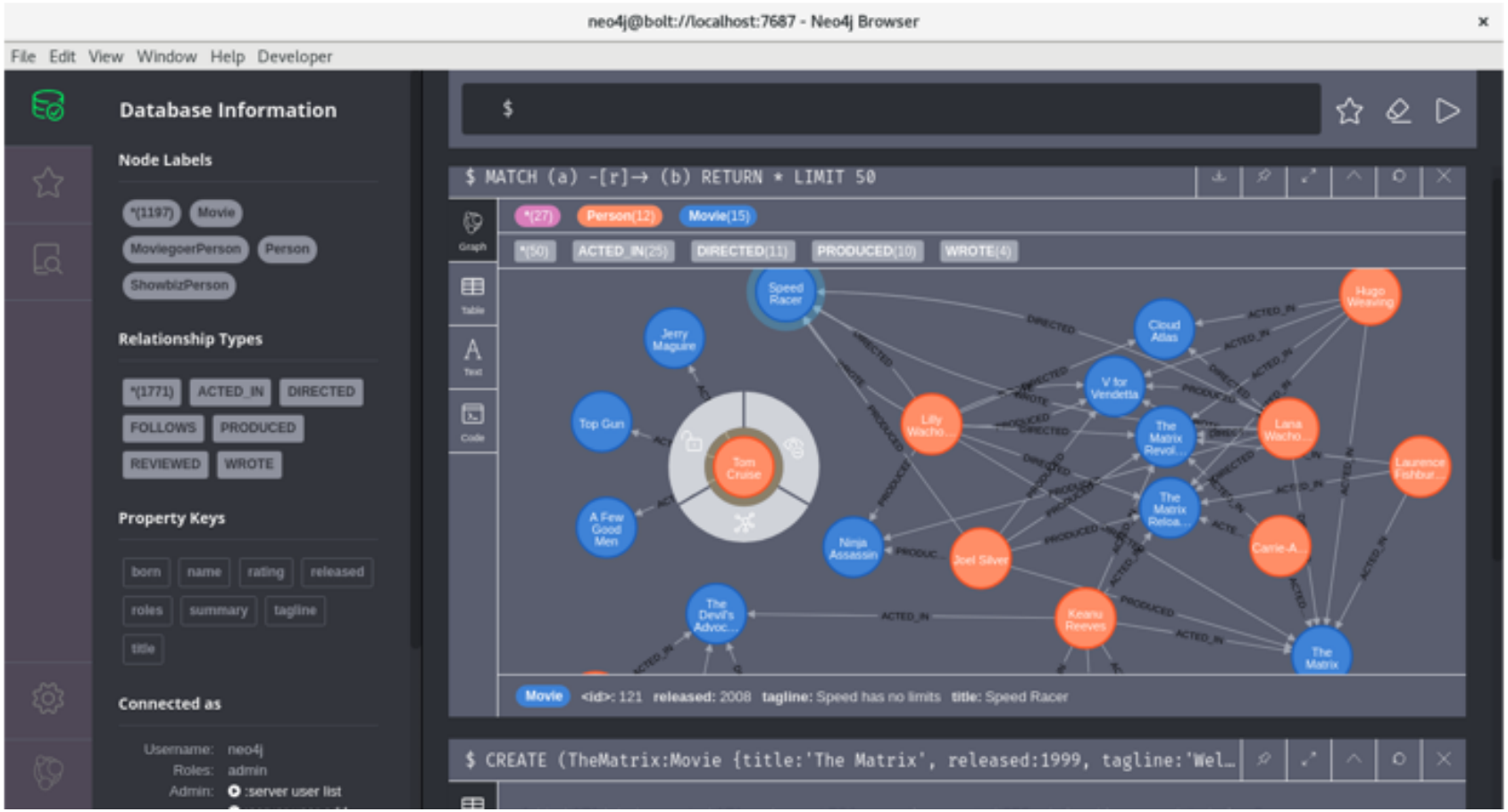
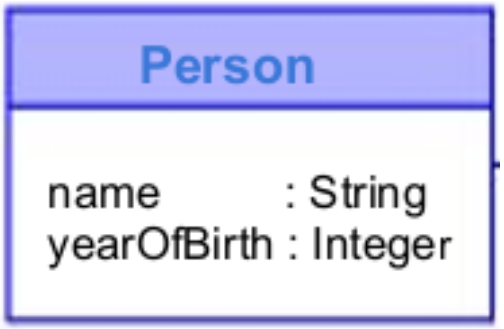


Figure: Screenshot Neo4j

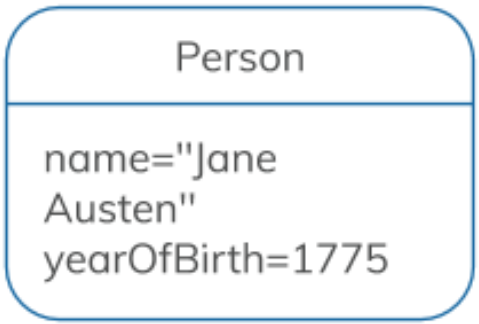
Passage modèle conceptuel → modéle graphe

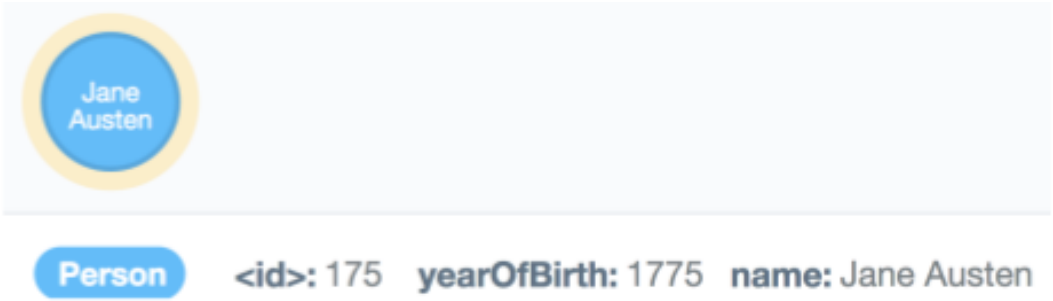
Transformation des classes



- dans le modèle graphe, le schéma est *implicite*
- *instances* → **noeuds** :

```
(ja:Person { name:  'Jane Austen', yearOfBirth:  1775 })
```



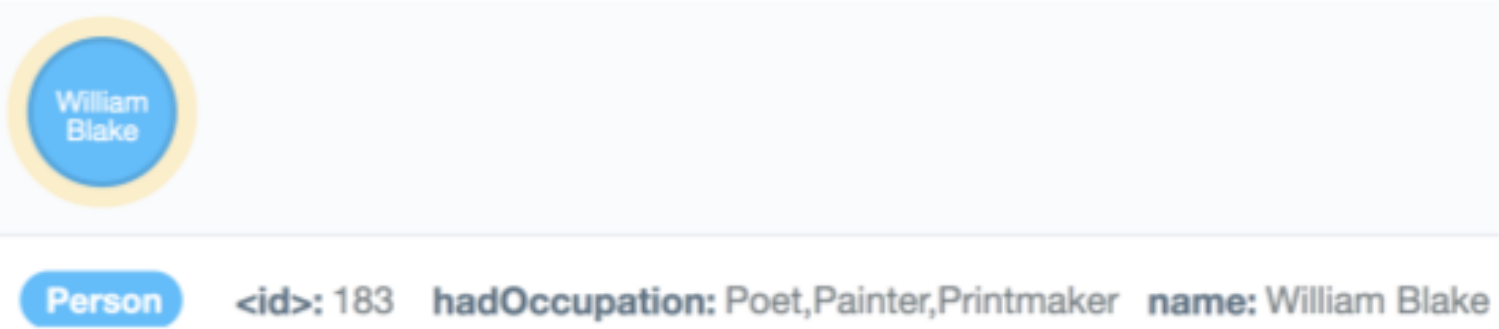


Transformation des classes

- les propriétés associées aux noeuds(/arêtes) peuvent être *multi-valuées* :

```
(wb:Person { name:  'William Blake', hadOccupation:  ['Poet','Painter','Printmaker'] })
```

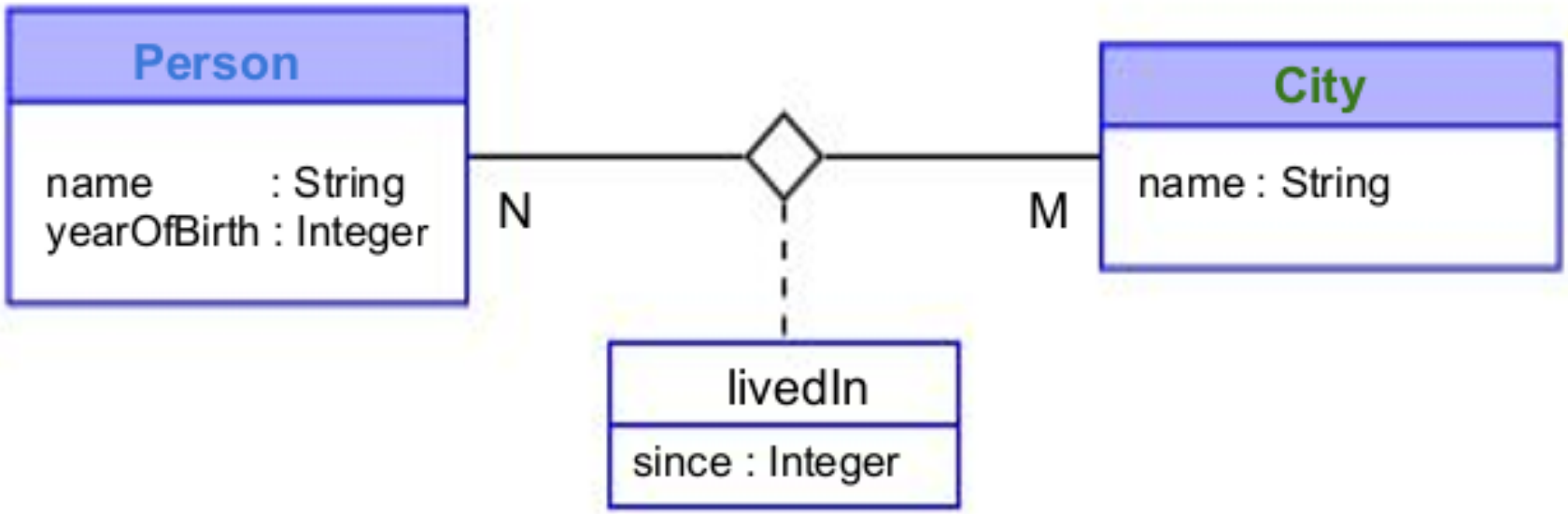
- les types composites en Neo4j sont `List` et `Map` (dictionnaires)



- pour eviter les valeur complexes → *réification* :

```
(wb:Person { name:  'William Blake'})-[:hadOccupation]-> (po:Occupation {type :  'Poet'})] })
(wb:Person { name:  'William Blake'})-[:hadOccupation]-> (pa:Occupation {type :  'Painter'})] })
(wb:Person { name:  'William Blake'})-[:hadOccupation]-> (pr:Occupation {type :  'Printmaker'})] })
```

Transformation des relations N:M - Exemple



- une personne peut avoir vécu dans plusieurs villes;
- une ville peut avoir été habité par plusieurs personnes.

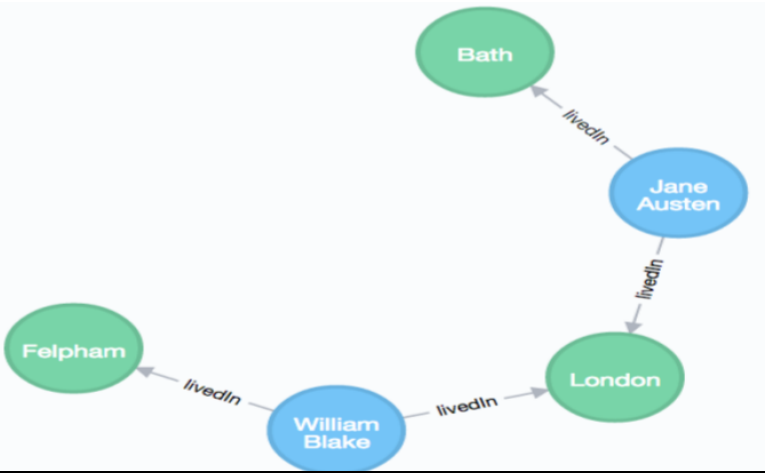
Transformation des relations N:M - Instance

- **Noeuds** :

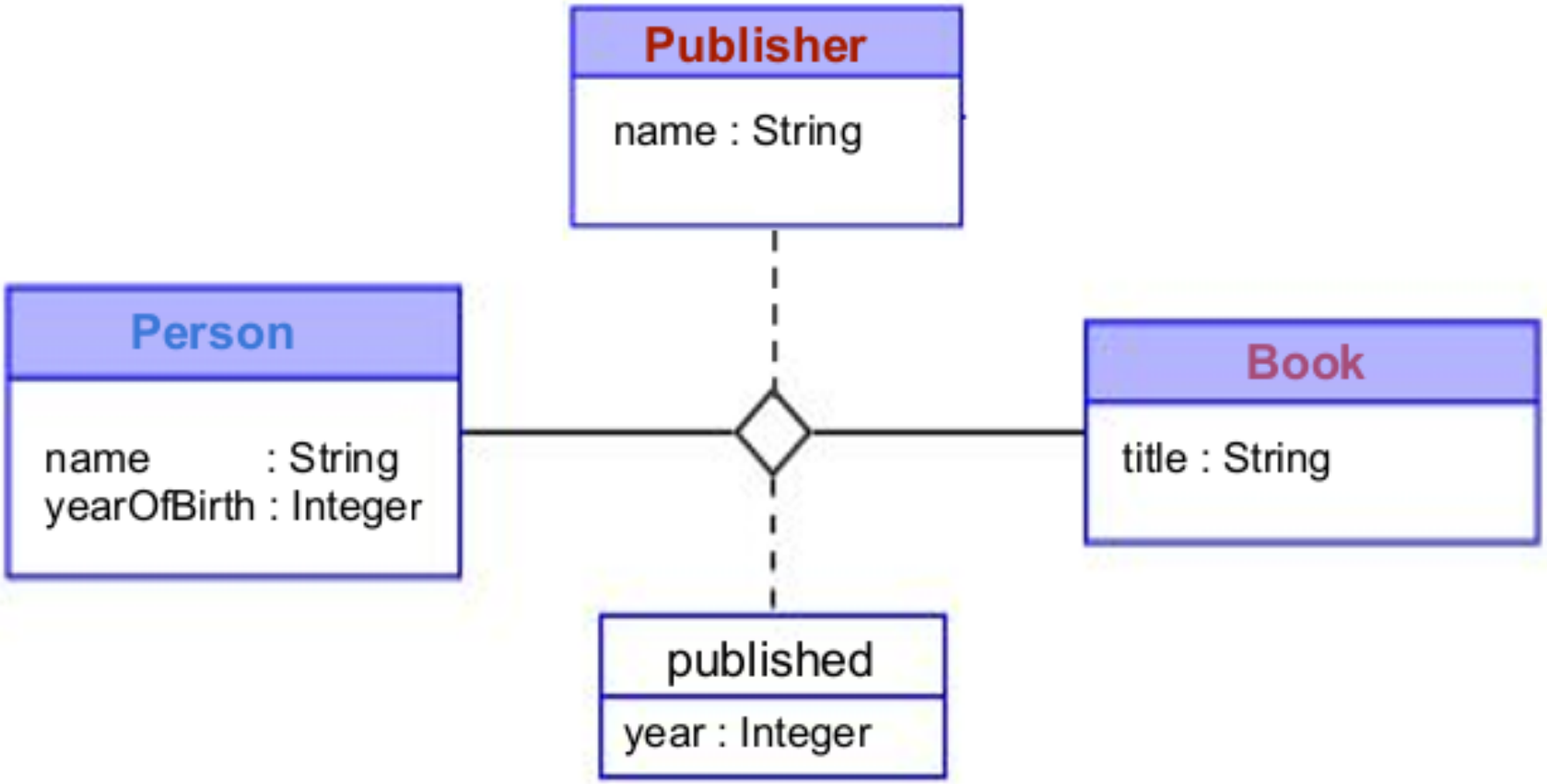
```
(ja:Person { name:  'Jane Austen'  , yearOfBirth:  1775 })
(wb:Person { name:  'William Blake', yearOfBirth:  1757 })
(lo:City { name:  'London'})
(ba:City { name:  'Bath'})
(fe:City { name:  'Felpham'})
```

- **Arêtes** :

```
(ja)-[:livedIn { since:  1775 }]->(lo)
(ja)-[:livedIn { since:  1800 }]->(ba)
(wb)-[:livedIn { since:  1757 }]->(lo)
(wb)-[:livedIn { since:  1800 }]->(fe)
```



Transformation des relations N-aires - Exemple



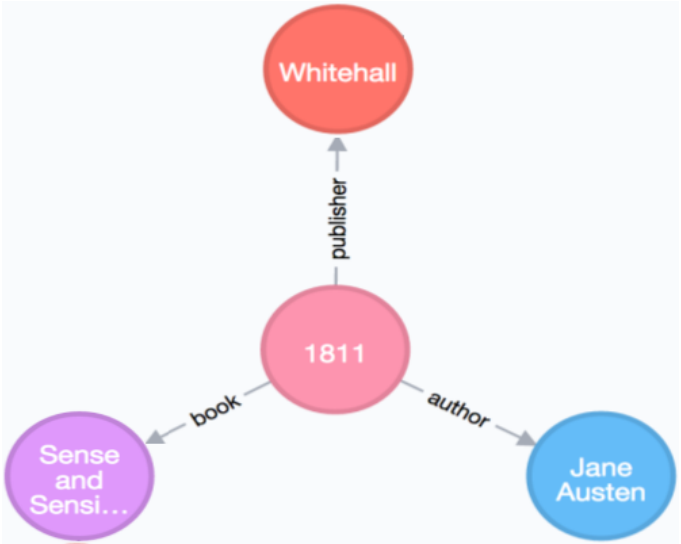
Transformation des relations N-aires : Instance

- **Noeuds** :

```
(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })
(wh:Publisher { name: 'Whitehall' })
(sas:Book {title: 'Sense and Sensibility' })
```

- **Arêtes** :

```
(pub:Publication {year: 1811 })
(pub)-[:author]->(ja)
(pub)-[:book]->(wh)
(pub)-[:publisher]->(sas)
```



Passage modèle relationnel → modéle graphe

Transformation Relationnel → Graphe

1. *clés étrangères → relations*
2. *tables de jointures simples → relations*
3. *tables de jointures attributaires → relations avec des propriétés*

Transformation Relationnel → Graphe (1)

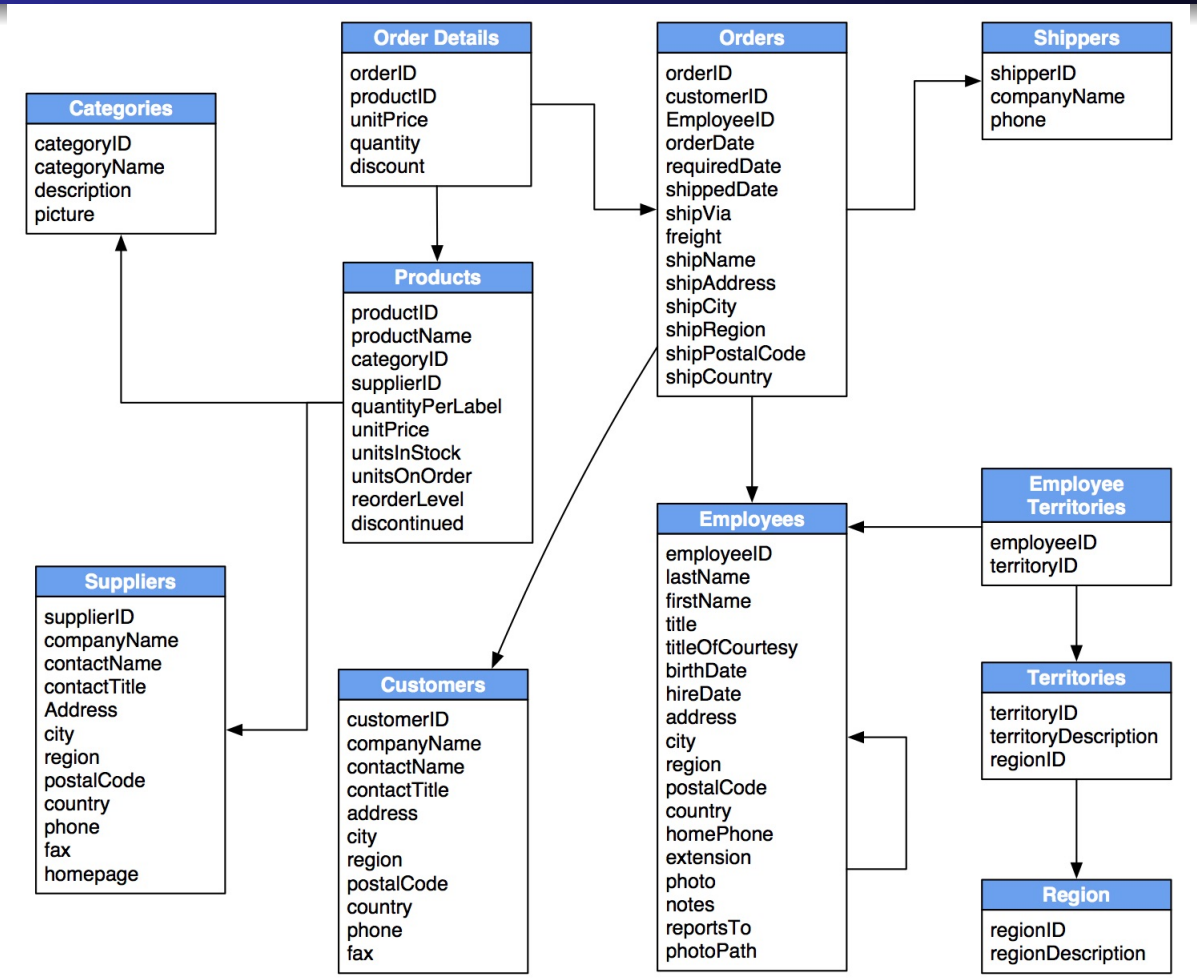


Figure: Base de Données Northwind - Modèle EA

Transformation Relationnel → Graphe (1)

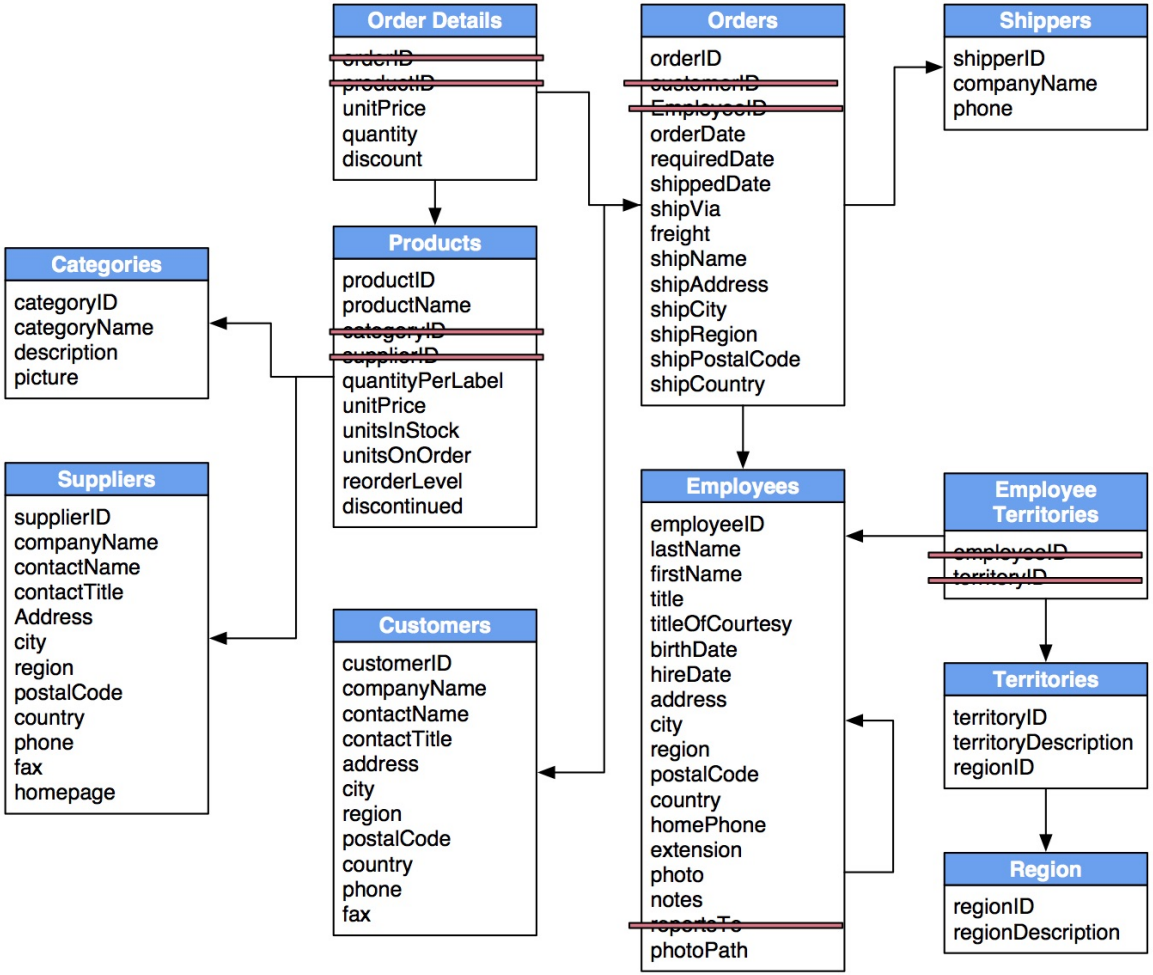


Figure: Base de Données Northwind - Modèle EA

Transformation Relationnel → Graphe (1)

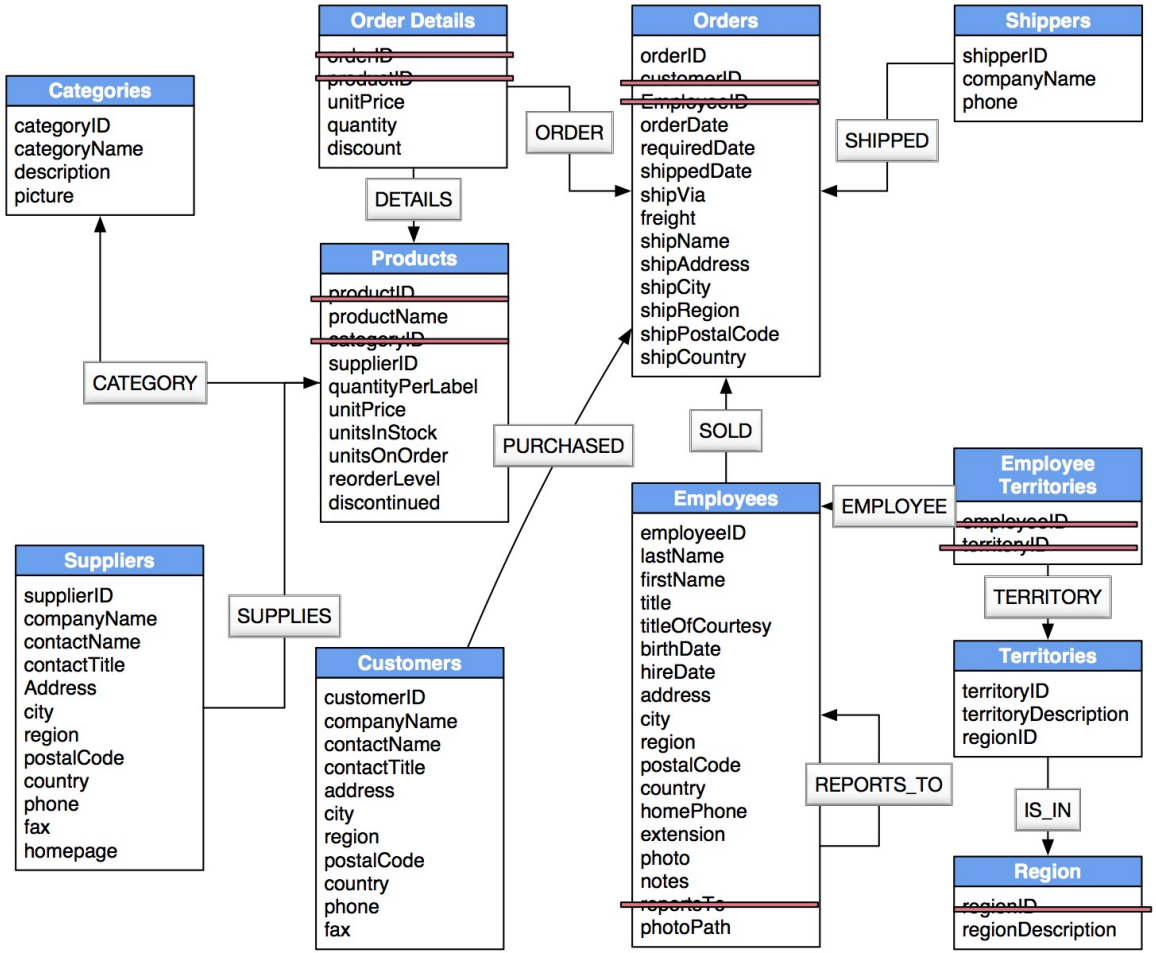


Figure: Base de Données Northwind - Modèle EA

Transformation Relationnel → Graphe (2)

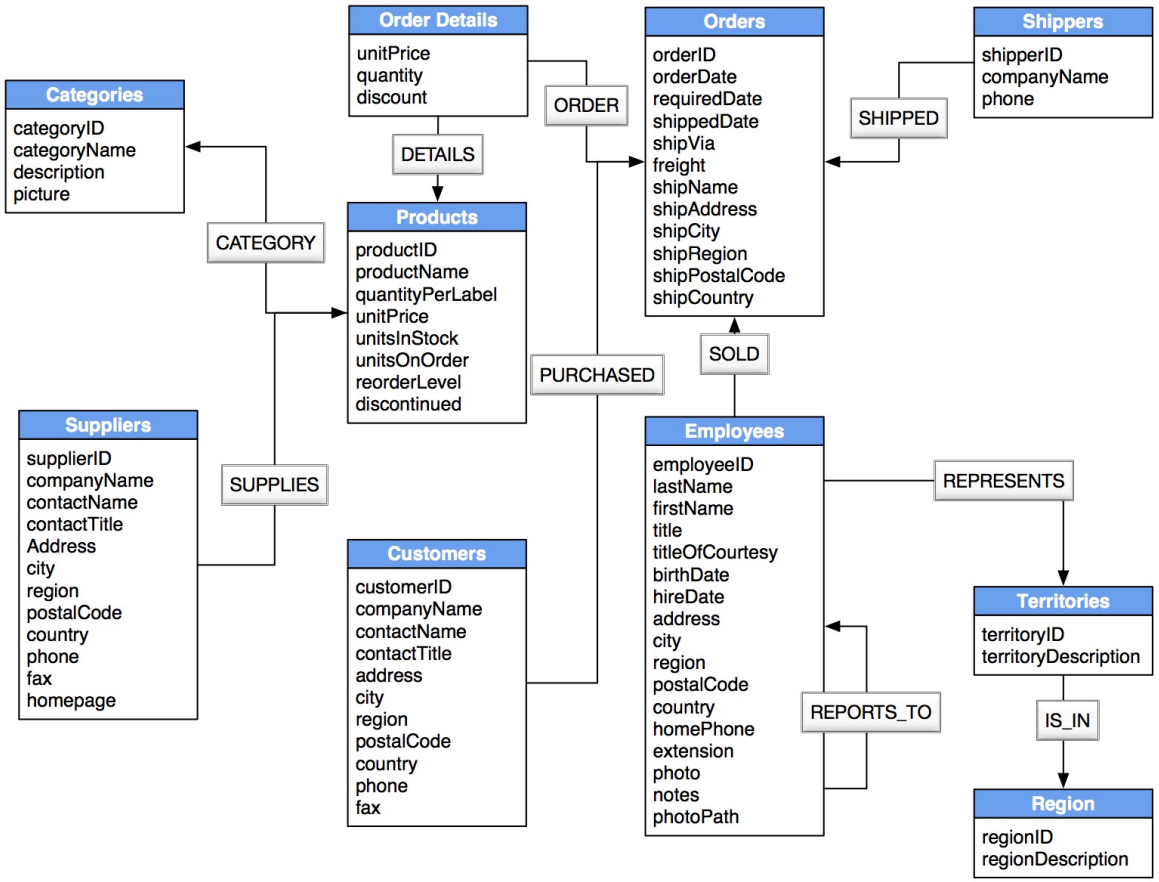


Figure: Base de Données Northwind - Modèle EA

Transformation Relationnel → Graphe (3)

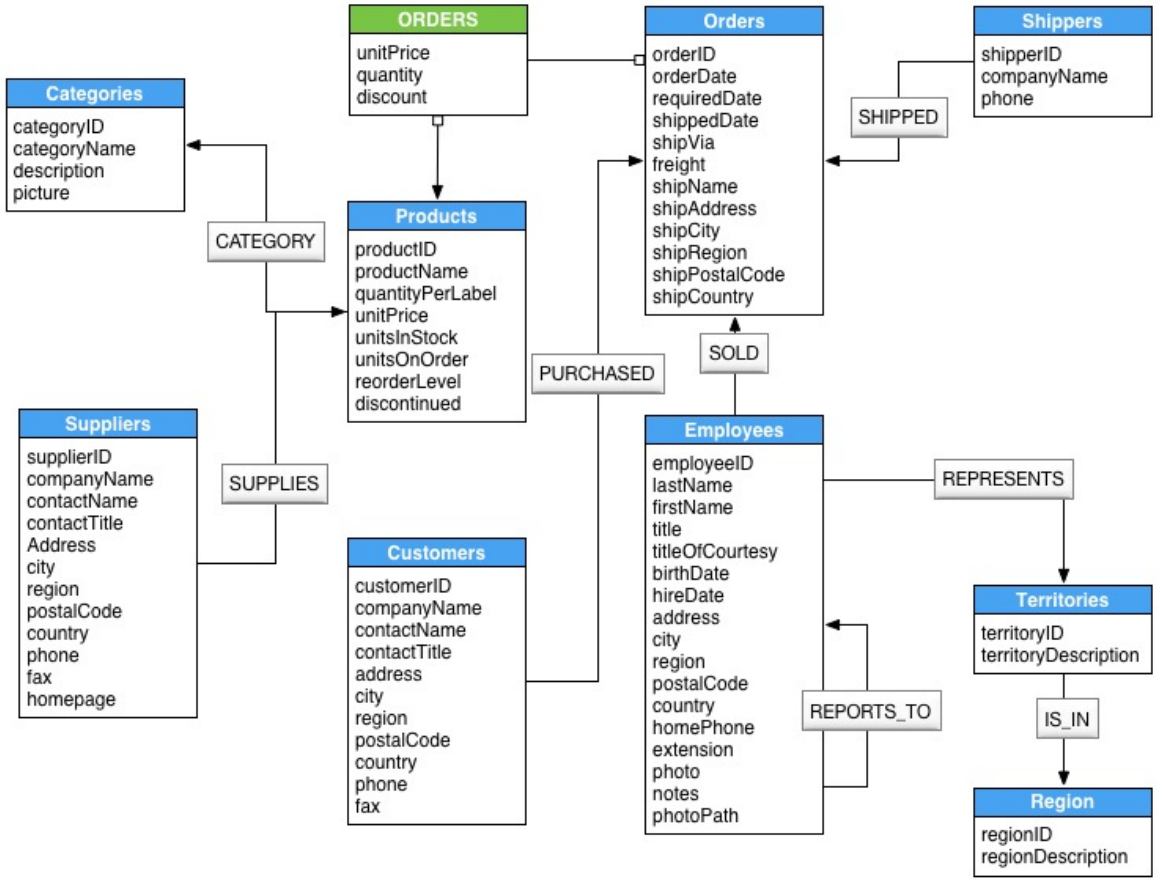


Figure: Base de Données Northwind - Modèle EA

Transformation Relationnel → Graphe

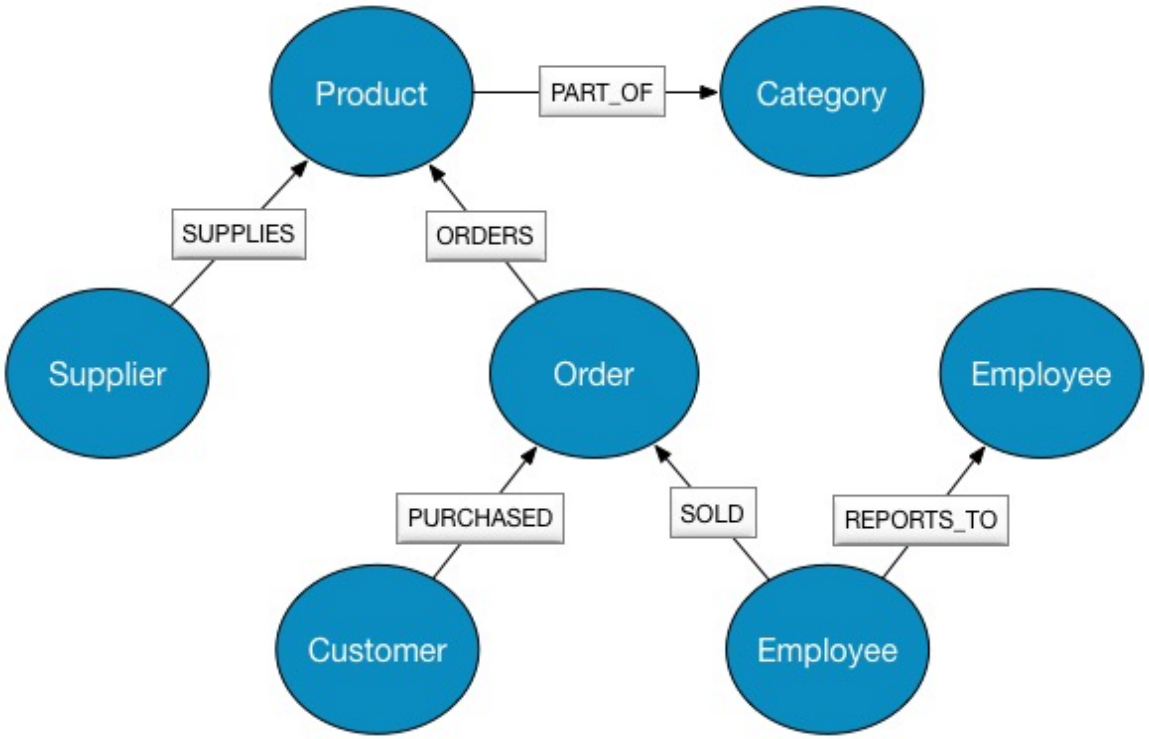


Figure: Base de Données Northwind - Modèle EA

Exercice

Transformer la base de données relationnelle dans une base de données graphe :

Person

tid	name	year
1	zoe	1990
2	joe	null
3	sue	1998

City

tid	cname	size
4	smallville	big

Partners

tid	partnerA	partnerB	location	since
5	zoe	joe	smallville	2000

Exercice (Solution)

Transformer la base de données relationnelle dans une base de données graphe :

Person

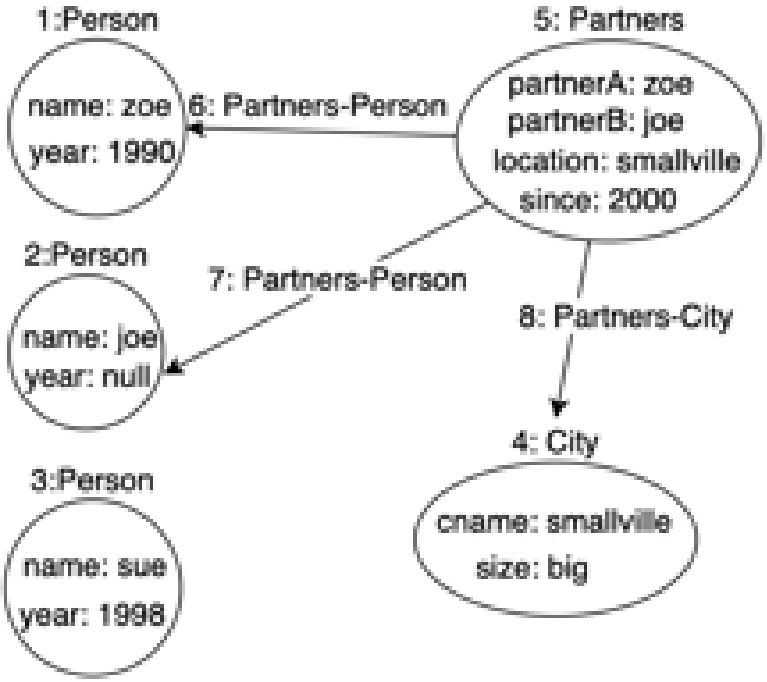
tid	name	year
1	zoe	1990
2	joe	null
3	sue	1998

City

tid	cname	size
4	smallville	big

Partners

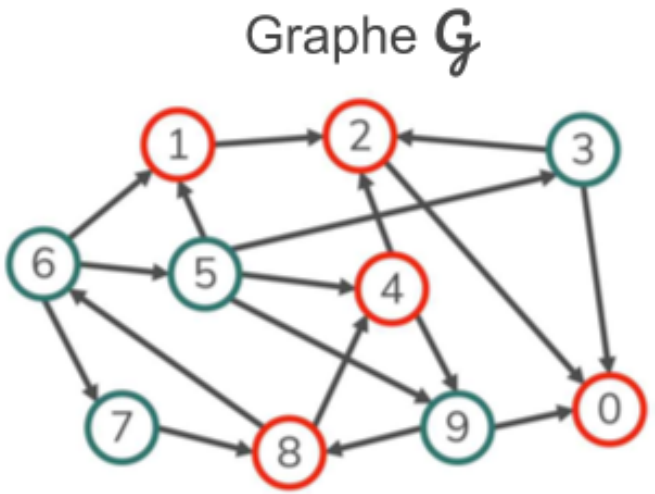
tid	partnerA	partnerB	location	since
5	zoe	joe	smallville	2000



Cypher : Langage des requêtes graphe

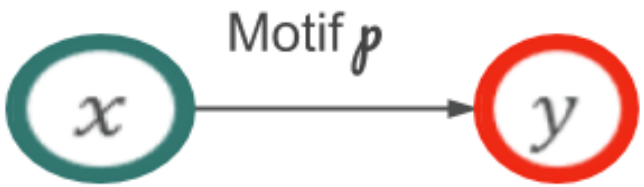
Motifs graphe

Graphe avec des variables



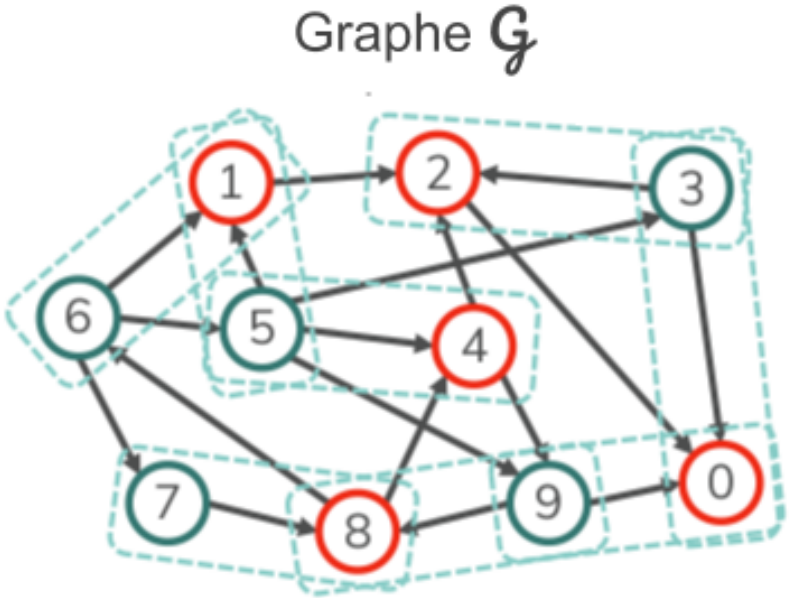
Trouver les sous-graphes de G
correspondant au motif p

Motifs graphe



Valuations ν

$\nu : \{ x \mapsto 6, y \mapsto 1\}$
$\nu : \{ x \mapsto 3, y \mapsto 2\}$
$\nu : \{ x \mapsto 5, y \mapsto 4\}$
$\nu : \{ x \mapsto 3, y \mapsto 0\}$
$\nu : \{ x \mapsto 9, y \mapsto 0\}$
$\nu : \{ x \mapsto 9, y \mapsto 8\}$
$\nu : \{ x \mapsto 7, y \mapsto 8\}$
$\nu : \{ x \mapsto 5, y \mapsto 1\}$



Cypher

Motifs pour les noeuds

- `()` noeud anonyme
- `(matrix)` noeud identifié par la variable *matrix*
- `(:Movie)` noeud anonyme avec l'étiquette *Movie*
- `(matrix:Movie:Action)` noeud avec l'étiquettes *Movie* et *Action* identifié par la variable *matrix*
- `(matrix:Movie {title: "The Matrix"})` + étiquette *title* avec valeur "The Matrix"
- `(matrix:Movie {title: "The Matrix", released: 1997})` + étiquette *released* avec valeur *1997*

Motifs pour les arêtes

- `--` arête anonyme sans direction
- `-->` arête anonyme avec direction
- `-[role]->` arête dirigé identifié par la variable *role*
- `-[:ACTED_IN]->` arête anonyme dirigé avec l'étiquette *ACTED_IN*
- `-[role:ACTED_IN]->` arête dirigé avec l'étiquette *ACTED_IN* identifié par la variable *role*
- `-[role:ACTED_IN {roles: ["Neo"]}]->` + étiquette *roles* avec valeur "Neo"

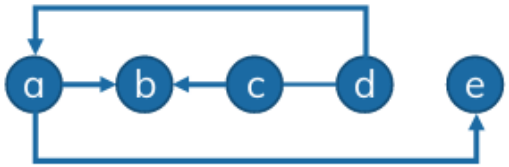
Figure: Syntaxe des Motifs Cypher

`https://neo4j.com/docs/cypher-manual/current/syntax/patterns/`

Cypher

Motifs pour les chemins

- Chaîne des motifs noeud/arête alternants
- ...qui commence et se termine avec un motif noeud
- `(a)-->(b)<--(c)--(d)-->(a)-->(e)`
- `(keanu:Person:Actor {name: "Keanu Reeves"})-[role:ACTED_IN {roles: ["Neo"]}]-> (matrix:Movie {title: "The Matrix"})`



Motifs pour les graphes

- Un ou plusieurs motifs chemin
- Les motifs chemin doivent avoir au moins une variable commune
- Sinon, le motif est déconnecté
  - Complexité quadratique
- `(a)-->(b)<--(c)--(d)-->(a)-->(e)`, `(e)-->(b)-->(d)`, `(a)-->(a)`

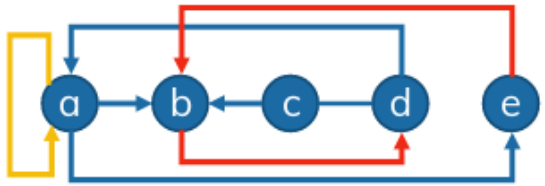


Figure: Syntaxe des Motifs Cypher

`https://neo4j.com/docs/cypher-manual/current/syntax/patterns/`

Cypher

Q1 : Quels sont les motifs valides ?

A `(a,b:Movie)-[:SHOWN_IN]-> (e),(f)`

B `(a:Movie)-[:SHOWN_IN]->()`

C `(:Movie)-[:SHOWN_IN]->`

D `()<--(a:Movie)`

Cypher

Q1 : Quels sont les motifs valides ?

A `(a,b:Movie)-[:SHOWN_IN]-> (e),(f)`

B `(a:Movie)-[:SHOWN_IN]->()`

C `(:Movie)-[:SHOWN_IN]->`

D `()<--(a:Movie)`

Cypher

Q2 : Quels sont les motifs qui spécifient un cycle ?

A `(a:Movie {name: "Matrix"})-->(a)`

B `(a:Movie {name: "Matrix"})-->(b:Movie {name: "Matrix"})`

C `(a:Movie {name: "Matrix"})-->(a:Movie {name: "Matrix"})`

D `(a:Movie {name: "Matrix"})-->({name: "Matrix"})`

Cypher

Q2 : Quels sont les motifs qui spécifient un cycle ?

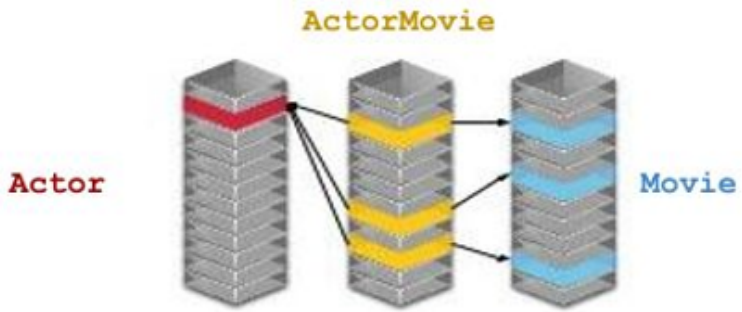
A `(a:Movie {name: "Matrix"})-->(a)`

B `(a:Movie {name: "Matrix"})-->(b:Movie {name: "Matrix"})`

C `(a:Movie {name: "Matrix"})-->(a:Movie {name: "Matrix"})`

D `(a:Movie {name: "Matrix"})-->({name: "Matrix"})`

Cypher



Quels sont les acteurs qui ont joué dans le film "V for Vendetta" ?

```
SELECT name FROM Actor
LEFT JOIN ActorMovie
  ON Actor.Id = ActorMovie.ActorId
LEFT JOIN Movie
  ON Movie.Id = ActorMovie.MovieId
WHERE Movie.name = "V for Vendetta"
```

SQL

```
MATCH (a:Actor)-[:ACTED_IN]-(m:Movie)
WHERE m.name = "V for Vendetta"
RETURN a.name
```

Cypher

Cypher

Briques de base

- MATCH : spécification de la *forme* des structures récherchés
  - nœuds simples : (a)
  - nœuds reliés : (a) → (b) ou (a) → (b) ← (c) ou (a) → () ← (c)
  - étiquettes : (a:Person) → (b)
    - (a :Person :Actor) → (b)
    - (a :Actor | :Director) → (b)
  - propriétés : (a {name: 'Michelle Williams', born: '1980'})
  - rélations : (a) - (b) ou (a) -[r]→ (b) ou
    - (a) -[*k]→ (b)
    - (a) -[*i..j]→ (b)
    - (a) -[*i..]→ (b)
    - (a) -[*..j]→ (b)
    - (a) -[*]→ (b)

Cypher

Briques de base

- MATCH : spécification de la *forme* des structures récherchés

  (a : T1 {pa_1 : va_1,...}) -[:e {pe_1 : ve_1,...} *i..j]→ (b : T2 {pb_1 : vb_1,...})

- WHERE : filtrage des résultats
- RETURN ... AS : formatage des résultats dans une table
- LIMIT : spécification de la taille des résultats affiché

Clause MATCH

Q3: Combien des résultats vont être renvoyés par la requête suivante ?

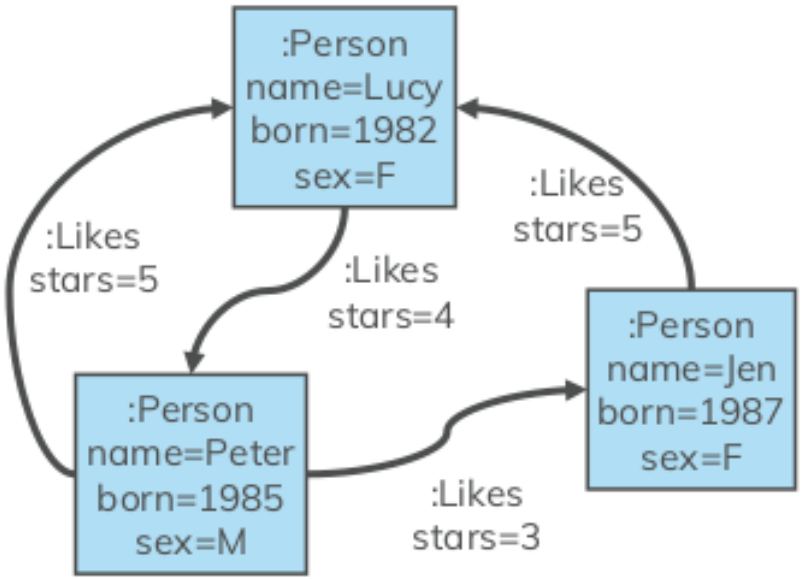


Figure: Example Graphe PGM

```
MATCH (p:Person) -[:Likes]-> (f:Person)
RETURN p.name, f.sex
```

Clause MATCH

Q3: Combien des résultats vont être renvoyés par la requête suivante ?

p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

Figure: Example Graphe PGM

```
MATCH (p:Person) -[:Likes]-> (f:Person)
RETURN p.name, f.sex
```

Clause MATCH

Q4: Combien des résultats vont être renvoyés par la requête suivante ?

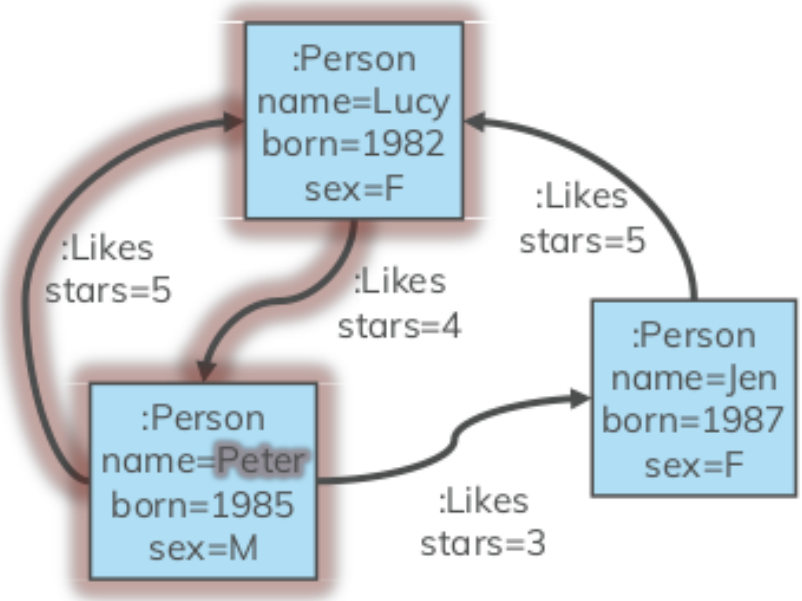


Figure: Example Graphe PGM

```
MATCH (p:Person) -[:Likes]-> (f:Person) -[:Likes]-> (fof:Person)
RETURN p.name, fof.name
```

Clause MATCH

Q4: Combien des résultats vont être renvoyés par la requête suivante ?

p.name	fof.name
Lucy	Jen
Peter	Lucy
Peter	Peter
Jen	Peter
Lucy	Lucy

Figure: Example Graphe PGM

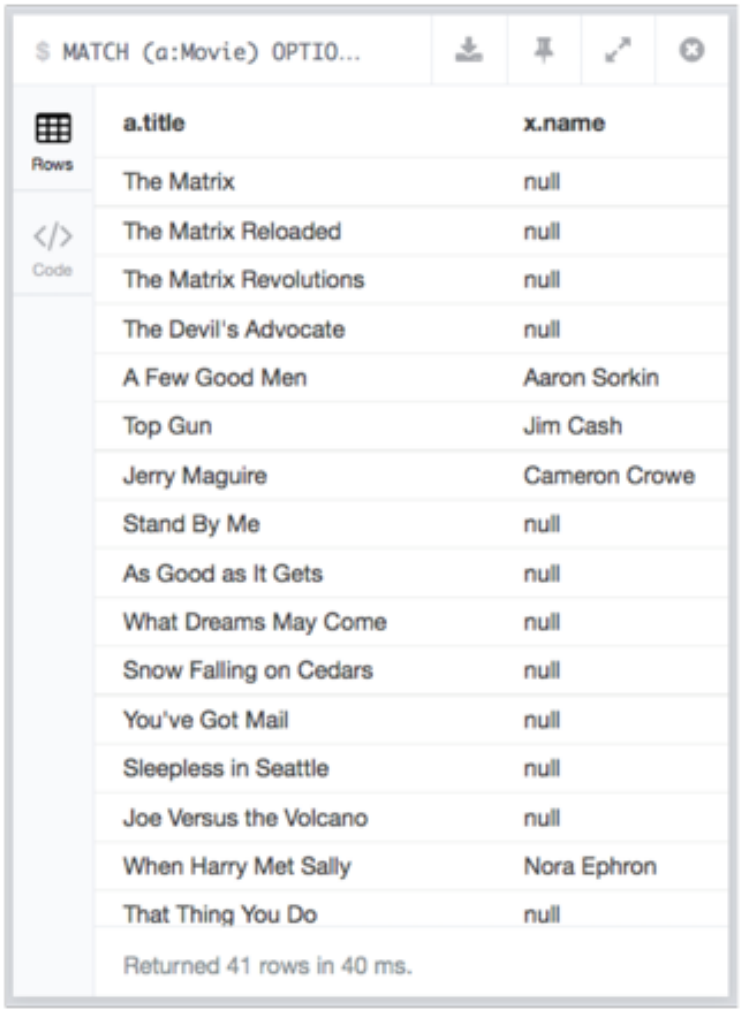
```
MATCH (p:Person) -[:Likes]-> (f:Person) -[:Likes]-> (fof:Person)
RETURN p.name, fof.name
```

Clause OPTIONAL MATCH

- Les motifs spécifies dans `OPTIONAL MATCH` sont les mêmes que dans `MATCH`.
- Les résultats peuvent correspondre à des sous-motifs.
- Usage des `NULL`s comme dans le `LEFT JOIN` relationnel.

```
MATCH (a:Movie)
OPTIONAL MATCH (a)<-[:WROTE]-(x)
RETURN a.title, x.name
```

Clause OPTIONAL MATCH

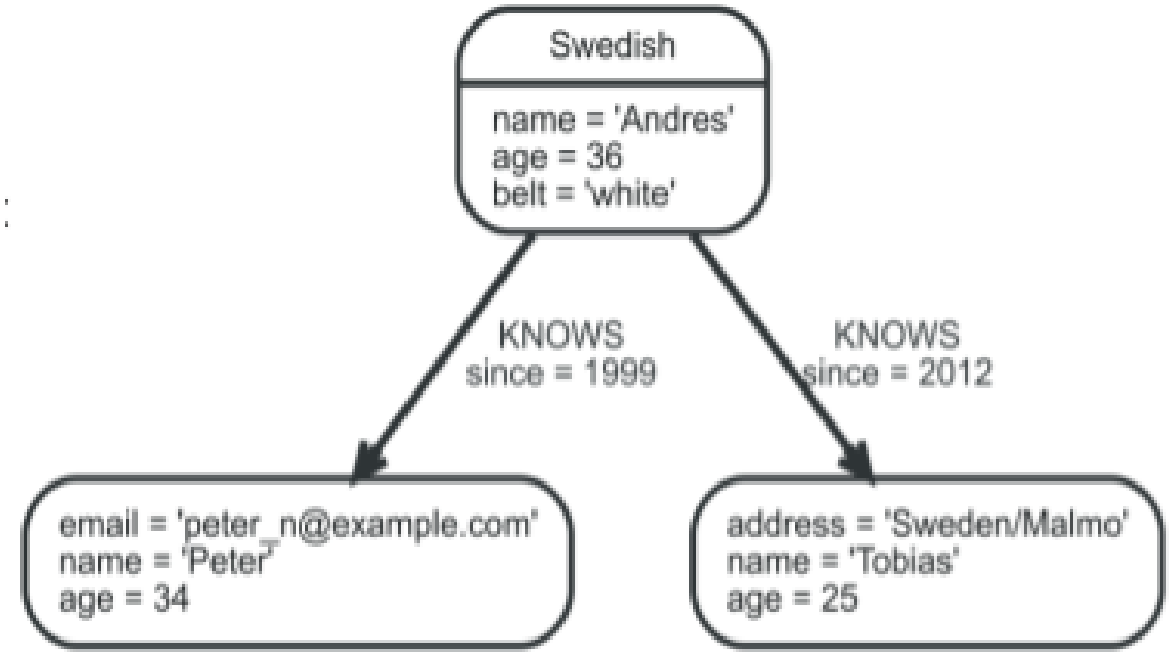


Clause WHERE

- ajout des contraintes aux motifs d'un MATCH/OPTIONAL MATCH
- ... ou filtrage des résultats d'une clause WITH

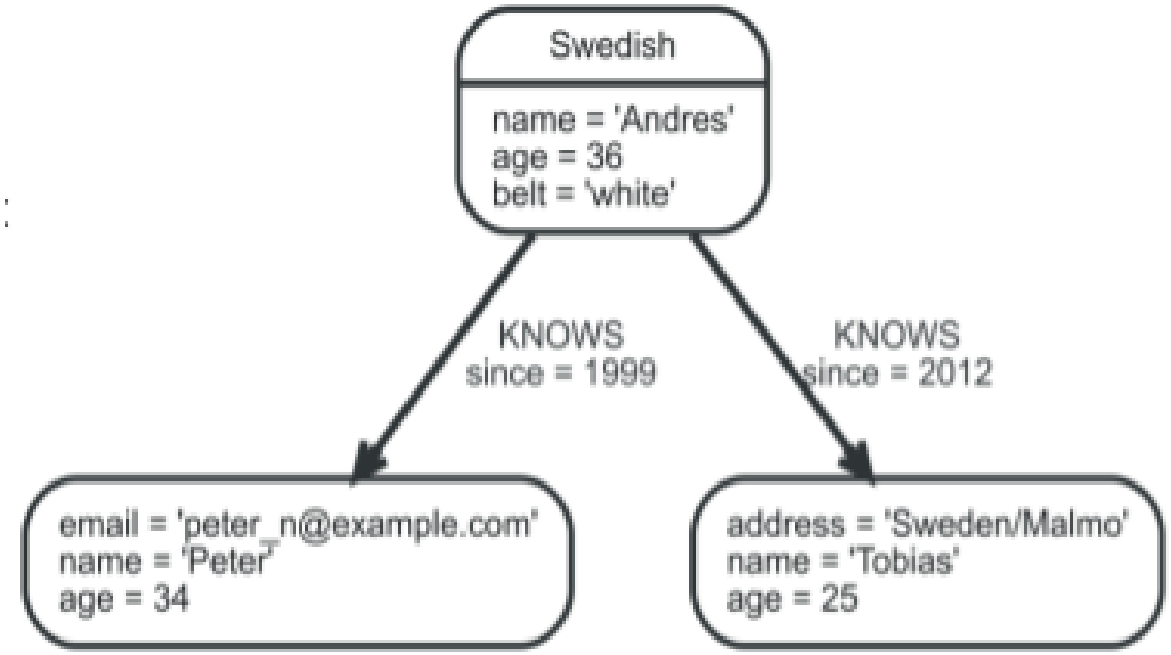
`https://neo4j.com/docs/cypher-manual/current/clauses/where/`

Clause WHERE




```
MATCH (n)
WHERE n.name = "Peter" XOR (n.age < 30 AND n.name="Tobias")
OR NOT (n.name="Tobias" OR n.name="Peter")
RETURN n
```

Clause WHERE



n
Node[0]{name:"Andres",age:36,belt:"white"}
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

Clause WHERE

- Filtrer l'étiquette d'un nœud

```
MATCH (n) WHERE n:Swedish RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}

- ... ou d'une propriété d'un nœud

```
MATCH (n) WHERE n.age < 30 RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

- ... ou d'un type relation

```
MATCH (n)-[k]->(f) WHERE type(k) = 'KNOWS'
AND k.since < 2000 RETURN f
```

n
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

Clause WHERE

- ... ou une liste

```
MATCH (n) WHERE a.name IN ["Peter", "Tobias"] RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

- ... ou des chaînes
  - préfixes
  - suffixes
  - infixes
  - expressions régulières

```
MATCH (n) WHERE n.name = 'Peter'          RETURN n
MATCH (n) WHERE n.name STARTS WITH 'Pet' RETURN n
MATCH (n) WHERE n.name ENDS WITH 'ter'   RETURN n
MATCH (n) WHERE n.name CONTAINS 'ete'    RETURN n
MATCH (n) WHERE n.name =~ 'P[et]+r?'     RETURN n
```

n
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

Clause WHERE

- ... ou l'existence des propriétés
- `false` par défaut, si valeurs manquantes

```
MATCH (n) WHERE exists(n.belt) RETURN n
MATCH (n) WHERE n.belt IS NOT NULL RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}

- ... ou la non-existence des propriétés

```
MATCH (n) WHERE NOT exists(n.belt) RETURN n
MATCH (n) WHERE n.belt IS NULL RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

Clause WHERE

- Usage des motifs pour le filtrage

```
MATCH (tobias { name: 'Tobias' }),(others)
WHERE others.age > 30 AND (tobias)<--(others)
RETURN others
```

n
Node[0]{name:"Andres",age:36,belt:"white"}

- ... avec négation

```
MATCH (persons),(peter { name: 'Peter' })
WHERE NOT (persons)-->(peter)
RETURN persons
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

- ... avec propriétés d'existence

```
MATCH (person)
WHERE EXIST((person)-->())
RETURN person
```

other
Node[0]{name:"Andres",age:36,belt:"white"}

Clause WHERE

Clause RETURN

- Renvoie des nœuds

```
MATCH (n { name: "B" }) RETURN n
```

n
Node[1]{name:"B"}

- Renvoie des relations

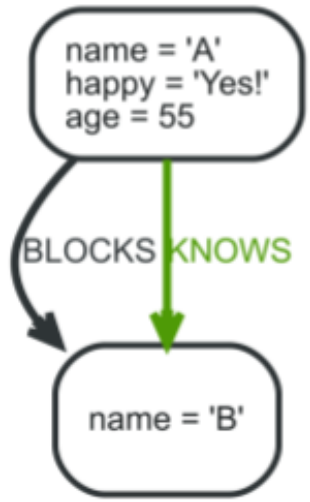
```
MATCH (n { name: "A" })-[r:KNOWS]->(c) RETURN r
```

r
:KNOWS[0]{}

- Renvoie des propriétés

```
MATCH (n { name: "A" }) RETURN n.name
```

n.name
"A"



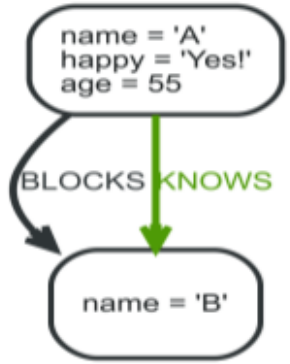
Clause WHERE

Clause RETURN

- Alias de Colonne

```
MATCH (a { name: "A" })
RETURN a.age AS SomethingTotallyDifferent
```

SomethingTotallyDifferent
55



- Renvoie tous les éléments `MATCH p = (a { name: "A" })-[r]->(b) RETURN *`

a	b	p	r
Node[0]{name:"A",happy:"Yes!",age:55}	Node[1]{name:"B"}	[Node[0]{name:"A",happy:"Yes!",age:55},:BLOCKS[1]{},Node[1]{name:"B"}]	:BLOCKS[1]{}
Node[0]{name:"A",happy:"Yes!",age:55}	Node[1]{name:"B"}	[Node[0]{name:"A",happy:"Yes!",age:55},:KNOWS[0]{},Node[1]{name:"B"}]	:KNOWS[0]{}

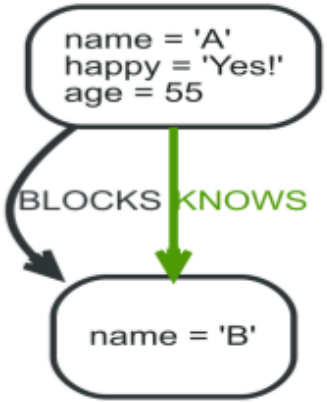
Clause WHERE

Clause RETURN

- Propriétés optionnelles

```
MATCH (n) RETURN n.age
```

n.age
55
<null>



- Autres expressions

```
MATCH (a { name: "A" })
RETURN a.age > 30, "I'm a literal", (a)-->()
```

a.age > 30	"I'm a literal"	(a)-->()
true	"I'm a literal"	[[Node[0]{name:"A",happy:"Yes!",age:55},:BLOCKS[1]{},Node[1]{name:"B"}], [Node[0]{name:"A",happy:"Yes!",age:55},:KNOWS[0]{},Node[1]{name:"B"}]]

- Résultats uniques

```
MATCH (a { name: "A" })-->(b) RETURN DISTINCT b
```

b
Node[1]{name:"B"}

Modification du Résultat

Clause ORDER BY

- Sous-clause après `RETURN` ou `WITH`
- Spécifie que le résultat doit être ordonné et comment

- Ordonner par propriétés
  `MATCH (n) RETURN n ORDER BY n.name`

- Ordonner par propriétés multiples
  `MATCH (n) RETURN n ORDER BY n.age, n.name`

- Ordonner de manière descendante
  `MATCH (n) RETURN n ORDER BY n.name DESC`

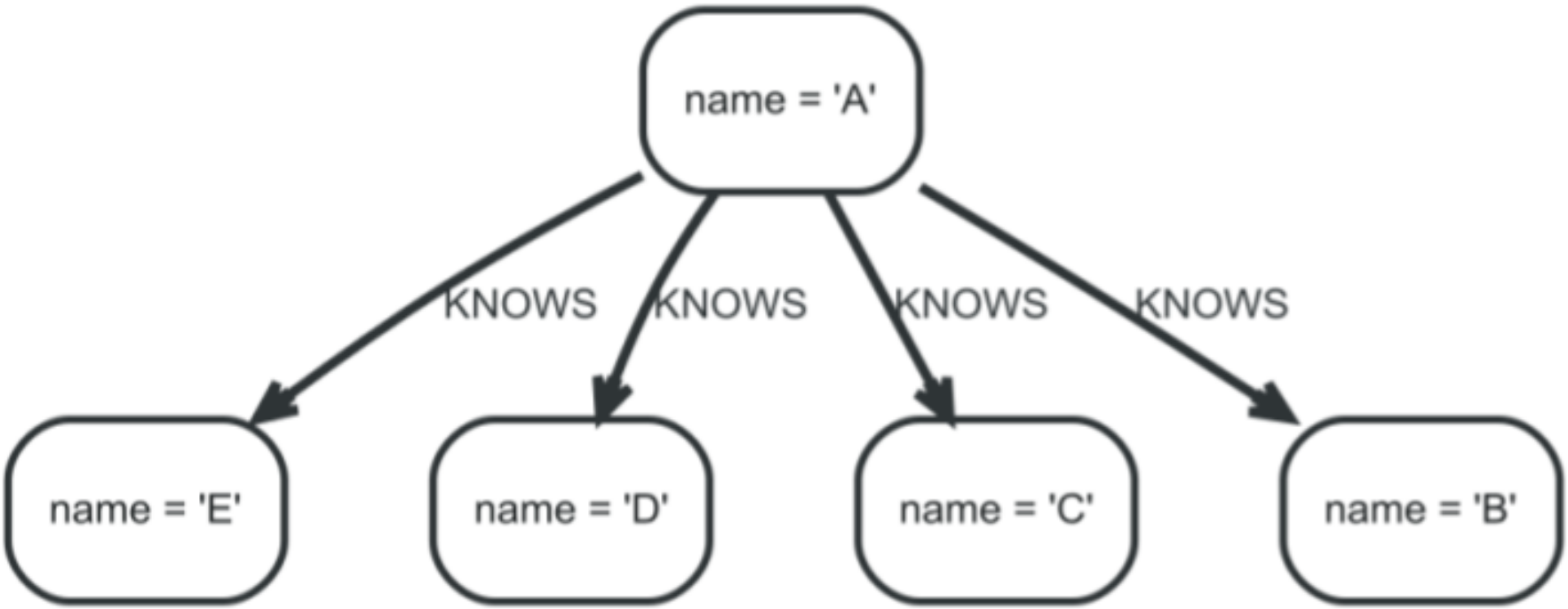
n
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}
Node[2]{name:"C",age:32,length:185}

n
Node[2]{name:"C",age:32,length:185}
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}

n
Node[2]{name:"C",age:32,length:185}
Node[1]{name:"B",age:34}
Node[0]{name:"A",age:34,length:170}

- On ne peut pas ordonner sur les nœuds/relations, seulement sur les propriétés
- `NULL` apparaît le dernier en ordre ascendante (`ASC`) et premier en ordre descendante (`DESC`)

Modification du Résultat



Modification du Résultat

Clause LIMIT

- Contraint le nombre des tuples dans le résultat
- Accepte toute expression qui peut être évalué comme un entier positif
- Expression ne peut pas faire référence aux nœuds/relations
- Renvoie premier tuple en ordre ascendente
  `MATCH (n) RETURN n ORDER BY n.name LIMIT 3`
- Return premier tuple qui satisfait l'expression
  `MATCH (n) RETURN n ORDER BY n.name LIMIT toInt(3 * rand()) + 1`

n
Node[0]{name:"A"}
Node[1]{name:"B"}
Node[2]{name:"C"}

Clause SKIP

- Définit à partir de quel tuple commencer y compris les tuples du résultat
- L'ensemble des tuples du résultat va être réduit à partir du dessus
- Même règle que pour LIMIT
- Soter les trois premiers tuples
  `MATCH (n) RETURN n ORDER BY n.name SKIP 3`

n
Node[3]{name:"D"}
Node[4]{name:"E"}

Manipulation du Graphe - Insertion

Clause CREATE

- Création d'une structure de graphe fournie par un motif

Création des nœuds

- Un nœud : `CREATE (n)`
- Plusieurs nœuds : `CREATE (n),(m)`
- Un nœud et plusieurs étiquettes : `CREATE (n:Person:Swedish)`
- Un nœud avec des étiquettes et propriétés : `CREATE (n:Person { name: 'Andres' })`

Renvoie des nœuds créés

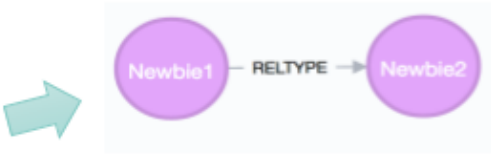
- Combiner avec la clause RETURN : `CREATE (a { name: 'Andres' }) RETURN a`

Manipulation du Graphe - Insertion

Creation des Relations

- Entre des nouveaux nœuds

```
CREATE (a:New {name: 'Newbie1'})
     -[r:RELTYPE]->(b:New {name: 'Newbie2'})
```



- Entre des nœuds existants

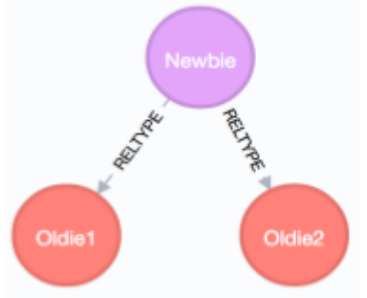
```
MATCH (a:Old {name: 'Oldie1'}),(b:Old {name: 'Oldie2'})
CREATE (a)-[r:RELTYPE]->(b)
```

- Entre des nœuds nouveaux et existants

```
MATCH (a:Old {name: 'Oldie1'})
CREATE (a)-[r:RELTYPE]->(b:New {name: 'Newbie'})
```

- Relations multiples

```
MATCH (a:Old {name: 'Oldie1'}),(b:Old {name: 'Oldie2'})
CREATE (a)<-[r1:RELTYPE]-(c:New {name: 'Newbie'})
-[r2:RELTYPE]->(b)
```



Manipulation du Graphe - Insertion

Clause SET

- Ajoute et met à jour les étiquettes des noeuds et les propriétés des noeuds et des relations
- ...sur les propriétés

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

n
Node[0]{surname:"Taylor", position:"Developer",name:"Andres",age:36,hungry:true}

- ...sur les étiquettes

```
MATCH (n { name: 'Emil' })
SET n:Swedish:Bossman
RETURN labels(n)
```

labels(n)
["Swedish","Bossman"]

Manipulation du Graphe - Suppression

Clause REMOVE

- Enlève des propriétés/étiquettes des éléments du graphe
- Enlever des propriétés

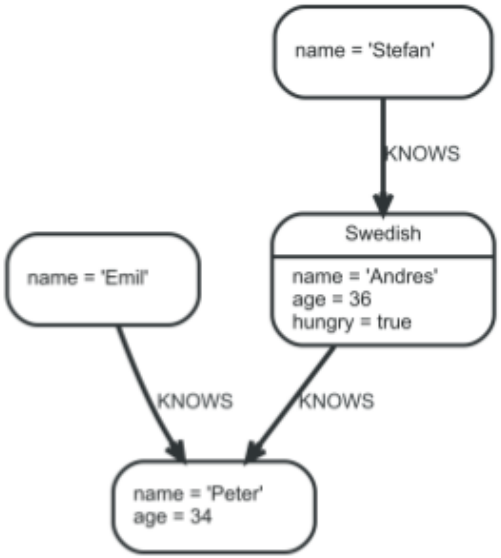
```
MATCH (n { name: 'Andres' })
REMOVE n.age, n.hungry
RETURN n
```

n
Node[0]{name:"Andres"}

- Enlever des étiquettes

```
MATCH (n { name: 'Emil' })
REMOVE n:Swedish:Bossman
RETURN labels(n)
```

labels(n)
[]



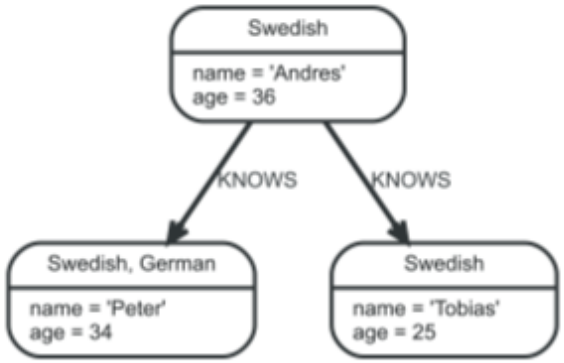
Manipulation du Graphe - Delete

Clause DELETE

- Supprime des éléments d'un graphe (noeuds/relations/chemins)

- Supprime un noeud – pas besoin d'être attaché à une relation
```
MATCH (n:Useless) DELETE n
// rien ne va être supprimé dans l'exemple ->
```
- Supprime un noeud avec toutes ses relations
```
MATCH (n { name: 'Andres' }) DETACH DELETE n
// Nodes deleted: 1 Relationships deleted: 2
```
- Supprime seulement des relations
```
MATCH (n { name: 'Andres' })-[r:KNOWS]->() DELETE r
// Nodes deleted: 0 Relationships deleted: 2
```

- Supprime tout
```
MATCH (n) DETACH DELETE n
```



Listes

Listes

- Éléments séparées par des virgules : `[0,1,2,3,4,5]  //=> [0,1,2,3,4,5]`
- Intervalles : `range(0, 5)  //=> [0,1,2,3,4,5]`

Accéder aux éléments d'une liste

- à un élément : `range(0, 5)[3]  //=> 3`
- à un élément de la fin : `range(0, 5)[-2]  // => 4`
- specification des sous-listes :
- `range(0, 5)[1..3]// => [1,2]`
- `range(0, 5)[..3] // => [0,1,2]`
- `range(0, 5)[3..] // => [3,4,5]`

Longueur d'une liste

- `size(range(0, 5)[0..3])  // => 3`

Expressions sur les listes

Comprehensions des listes

- Création d'une liste à partir des listes existantes
- Syntaxe: '[' <variable> 'IN' <listExpression> (('WHERE' <condition>) | ('|' <expression>)) ']'
- <condition> filtrage selon <listExpression>
- <expression> permet de construir un nouveau élément pour chaque élément de la liste filtrée

```
[x IN range(0,4) WHERE x % 2 = 0 | x^3]    // => [0.0,8.0,64.0]
[x IN range(0,4) WHERE x % 2 = 0]      // => [0,2,4]
[x IN range(0,4) | x^3]            // => [0.0,1.0,8.0,27.0,64.0]
```

Avec des motifs

```
MATCH (a:Person { name: 'Charlie Sheen' })
RETURN [(a)-->(b:Movie) WHERE b.year > 1980 | b.name] AS years
                          // => ['Wall Street', 'Red Dawn']
```

Expressions sur les listes

Fonctions sur les listes

Function	Description
`extract()`	Returns a single property, or the value of a function from a list of nodes or relationships.
`filter()`	Returns all the elements in a list complying with a predicate.
`keys()`	Returns a list of string representations for the property names of a node, relationship, or map.
`labels()`	Returns a list of string representations for the labels attached to a node.
`nodes()`	Returns all nodes in a path.
`range()`	Returns numerical values in a range.
`reduce()`	Runs an expression against individual elements of a list, storing the result of the expression in an accumulator.
`relationships()/rels()`	Returns all relationships in a path.
`tail()`	Returns all but the first element in a list.

Expressions sur les listes

Clause UNWIND

- Transforme une liste dans un tuple
- `UNWIND [1, 2, 3] AS x RETURN x`

x
1
2
3

- `MATCH (n) WHERE id(n)>171 RETURN n.name, labels(n)`

name	labels
Tim	[Student, SoccerPlayer]
Ann	[Professor, Boss]
Bob	[Employee, PhdStudent, TeachingAssistant]

- `MATCH (n) WHERE id(n)>171`
  `UNWIND labels(n) AS label RETURN n.name, label`

name	label
Tim	Student
Tim	SoccerPlayer
Ann	Professor
Ann	Boss
Bob	Employee
Bob	PhdStudent
Bob	TeachingAssistant

Expressions sur les maps

Litteraux map

- Syntaxe JSON : `{ name:'Bob', like:[{ name:'TV' },{ name:'NBA' }]}`

Acceder aux elements d'un map

- Notation : `bob.name // => 'Bob'`
- Noeud graphe est comme un map : `MATCH (bob:Person) RETURN bob.name // => 'Bob'`

Projections map

- Construire des projections map a partir des noeuds, relations et des autres valeurs
- Syntaxe : <mapVariable> '{' <map_element> {, <map_element>} '}'
- <mapVariable> dépend de l'entité graphe à partir de laquelle elle va être projeté

Expressions sur les maps

- Éléments d'un map produisent des paires clé-valeur
- Types d'éléments :
  - **Sélecteurs de propriétés**
    ```
    MATCH (actor:Person { name: 'Kevin Bacon' })
    RETURN actor { .name }                    //=> { name: 'Kevin Bacon' }
    ```
  - **Sélecteurs de toutes les propriétés**
    ```
    MATCH (actor:Person { name: 'Kevin Bacon' })
    RETURN actor { .* }                       //=> { name: 'Kevin Bacon', born: 1958 }
    ```
  - **Sélecteurs des valeurs**
    ```
    MATCH (actor:Person { name: 'Kevin Bacon' })-[:ACTED_IN]->(movie:Movie)
    RETURN actor {movies: count(movie) }       //=> { movies: 3 }
    ```
  - **Sélecteurs de variables**
    ```
    MATCH (actor:Person { name: 'Kevin Bacon' })-[:ACTED_IN]->(movie:Movie)
    WITH count(movie) AS numOfMovies
    RETURN actor { numOfMovies }              //=> { numOfMovies: 3 }
    ```
- Les expressions map peuvent être emboîtées

Chemins

Motifs de chemin de longueur variable

- Les types d'arrêts peuvent etre exprimes en spécifiant une longueur (borne inférieure/supérieure)
- Exemple: `(a)-[:x*2]->(b)` est la même chose que `(a)-[:x]->()-[:x]->(b)`
- Autres exemples:
```
(a)-[*3..5]->(b)
(a)-[*3..]->(b)
(a)-[*..5]->(b)
(a)-[*]->(b)
```
- Exemple complet:
```
MATCH (me)-[:KNOWS*1..2]-(remote_friend)
WHERE me.name = "Filipa" RETURN remote_friend.name
```

Variables de chemin

- Exemple: `p = ((a)-[*3..5]->(b))`

Chemins

Chemin le plus court

- Chemin entre deux noeuds avec le nombre minimal d'arrêts
- Appliquer la fonction **shortestPath/allShortestPath** à un motif de requête pour extraire toutes les chemins les plus courts
- Plus des prédicats de filtrage peuvent être ajoutées dans la clause WHERE
  - les prédicats universels (NONE/ALL) peuvent être évalués pendant la recherche du plus court chemin
  - autres prédicats peuvent être évalués une fois que le chemin le plus court a été trouvé
- Algorithme d'évaluation rapide
  - BFS bidirectionnel
  - Chemins sans prédicats supplémentaires et chemins avec des prédicats universels
- Algorithme d'évaluation lent
  - DFS
  - Fallback for paths with non-universal predicates

Chemins

Exemple (évaluation rapide)

```
MATCH (m { name:"Martin Sheen" }), (o { name:"Oliver Stone" }),
      p = shortestPath((m)-[*..15]-(o))
WHERE NONE(r IN rels(p) WHERE type(r)= "FATHER") RETURN p
```

Exemple (évaluation lente)

```
MATCH (m { name:"Martin Sheen" }), (o { name:"Oliver Stone" }),
      p = shortestPath((m)-[*..15]-(o))
WHERE length(p) > 1 RETURN p
```

longueur de chemin de max. 15 arrets

Composition des Requêtes

Clause WITH

- Enchaîne des parties d'une requête ensemble en passant les résultats d'une partie comme le point de départ d'une autre
- **Filtre sur les aggregates**
  Exemple: Equipe de futbol avec une age moyenne inférieure à 25
  ```
  MATCH (p)-[:PLAYS]->(t) WITH t, AVG(p.age) AS a WHERE a < 25 RETURN t
  ```
- **Aggregation des agregats**
  Exemple: L'âge moyenne du joueur le plus jeune de chaque équipe
  ```
  MATCH (p)-[:PLAYS]->(t) WITH t, MIN(p.age) AS a RETURN AVG(a)
  ```
- **Limite l'espace de recherche selon l'ordre des propriétés/agrégats**
  Exemple: Amis des cinq meilleurs amis
  ```
  MATCH (p)-[f:FRIENDS]->(p2)
  WITH f,p2 ORDER BY f.rating DESC LIMIT 5
  MATCH (p2)-[f:FRIENDS]->(p3) RETURN DISTINCT p3
  ```

Composition des Requêtes

Clause UNION

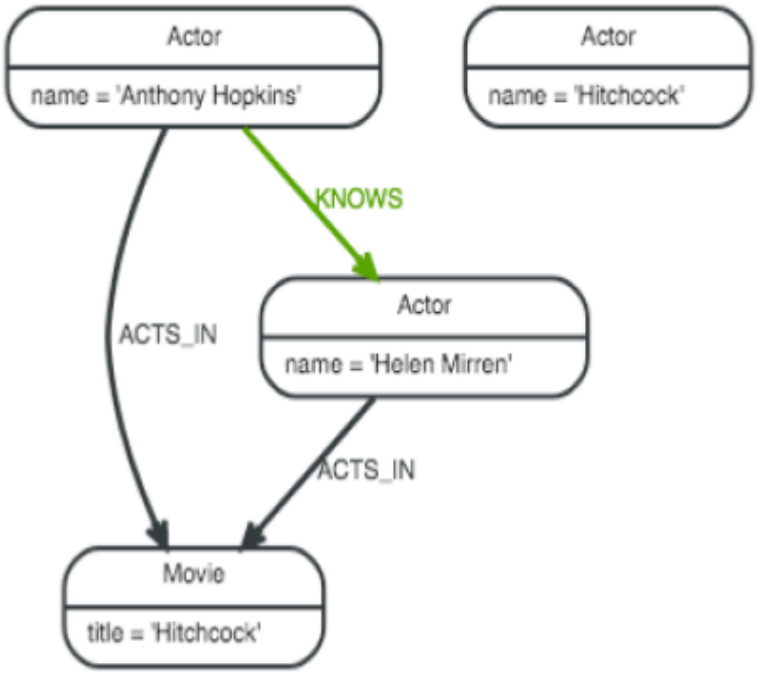
- Combine deux résultats et enlève les doublons
-
```
MATCH (n:Actor)
RETURN n.name AS name
UNION
MATCH (n:Movie)
RETURN n.title AS name
```

name
"Anthony Hopkins"
"Helen Mirren"
"Hitchcock"

Clause UNION ALL

- Combine deux résultats et retient les doublons
-
```
MATCH (n:Actor)
RETURN n.name AS name
UNION ALL
MATCH (n:Movie)
RETURN n.title AS name
```

name
"Anthony Hopkins"
"Helen Mirren"
"Hitchcock"
"Hitchcock"



Teaser

Pour apprendre plus :

- modèles graphe RDF (pour le web sématique)
- stockage des données graphe
- analyse des données graphe

... cours de Géstion de données avancé en S4

Exercices Neo4j

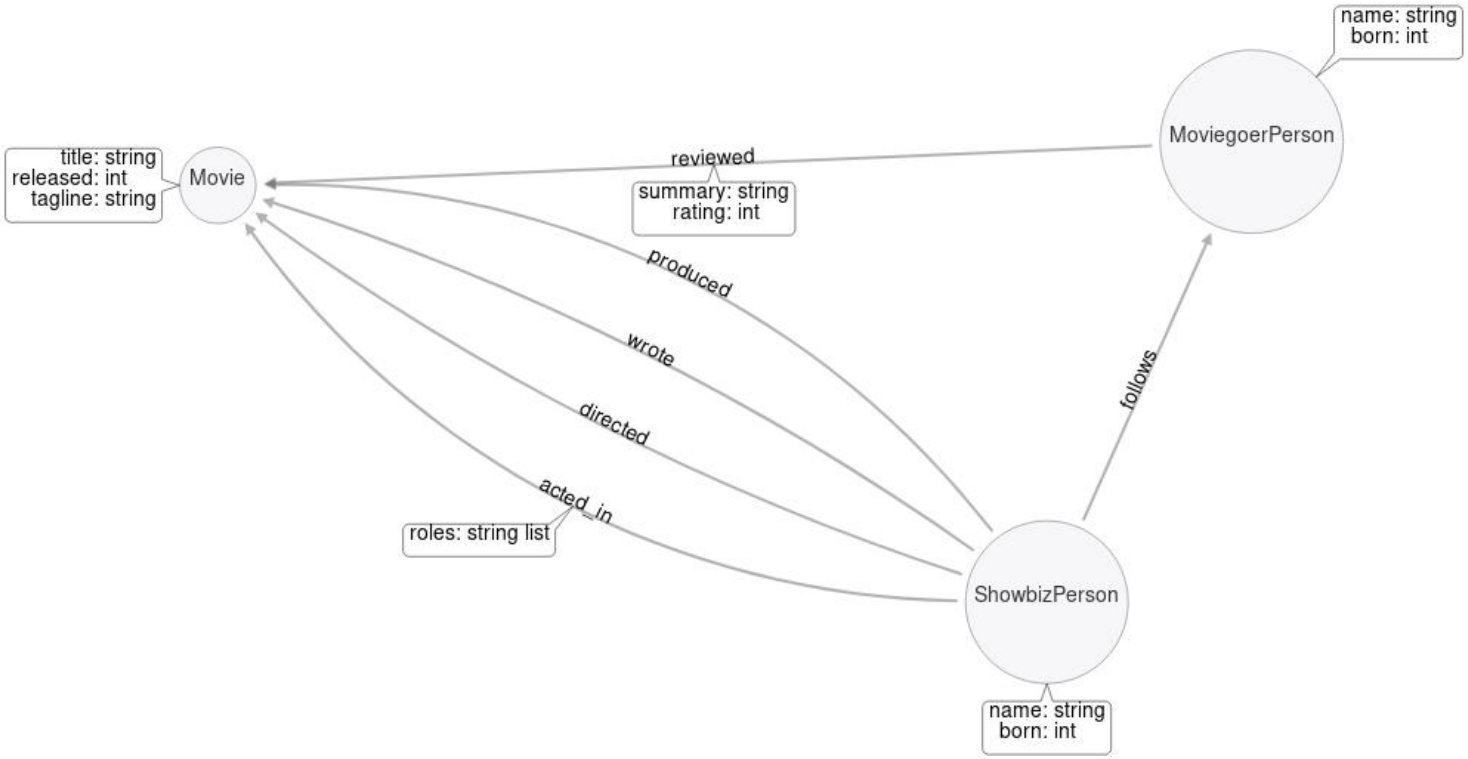


Figure: Base de Données FILM

Exercices Neo4j

Q1 Quels sont tous les noms des films dans lesquels à joué Meg Ryan ?

Exercices Neo4j

```
MATCH (a:ShowbizPerson {name : "Meg Ryan"}) -[:ACTED_IN]→ (b)
RETURN a,b
```

Exercices

Q2: Quels sont le(s) noms des réalisateur.ice(s) du film “The Matrix” ?

Exercices

```
MATCH (a:Movie {title: 'The Matrix'}) <-[:DIRECTED]- (b)
RETURN b
```

Exercices

Q3: Quels sont tous les noms des films qui ont été mis en vente entre 1990 et 2000 ?

Exercices

MATCH (a:Movie)
WHERE a.released $>$ 1990 AND a.released $<$ 2000
RETURN a.title

Exercices

Q4: Qui sont les acteurs qui ont joué dans les mêmes films que “Tom Cruise” ?

Exercices

```
MATCH (tom:ShowbizPerson {name:'Tom
Cruise'})−[:ACTED_IN]→(m)←[:ACTED_IN]-(c)
RETURN c.name
```

Exercices

Q5: Quels sont les personnes qui ont écrit *et* produit le même film ?

Exercices

```
MATCH (p)-[:WROTE]→(m:Movie)
WHERE (p)-[:PRODUCED]→(m)
RETURN p.name, m.title
```

Exercices

Q6: Combien y a t-il de personnes qui ont écrit *ou* produit un film ?

Exercices

```
MATCH (p)-[r:WROTE | :DIRECTED]→(m:Movie)
RETURN count(p)
```

Exercices

Q7: Combien d'acteurs sont reliés jusqu à trois dégrés à "Kevin Bacon" ?

Exercices

```
MATCH (b name:"Kevin Bacon")-[*1..3]-(c)
RETURN DISTINCT count(c)
```

Exercices

Q8: Qui sont tous les acteurs *directement* reliés à “Kevin Bacon” ?

Exercices

```
MATCH p=shortestPath(
(b name:"Kevin Bacon")-[*1..2]-(c)
)
WHERE c.name <> "Kevin Bacon"
WITH collect(c.name) AS names
RETURN names
```

Exercices

Q9: Qui sont tous les acteurs *indirectement* reliés à “Kevin Bacon” ?

Exercices

```
MATCH p=shortestPath(
(b name:"Kevin Bacon")-[*1..2]-(c)
)
WHERE c.name <> "Kevin Bacon"
WITH collect(c.name) AS names
MATCH (a) WHERE NOT a.name in names
WITH collect(a.name) AS cnames
RETURN cnames
```

Exercices

Q10: Y a t-il des acteurs à $>$ 6 dégrés de séparation de “Kevin Bacon” ?

Exercices

(132 personnes)

```
MATCH (a)
WHERE a.name <> "Kevin Bacon"
RETURN count(a)
```

(132 personnes)

```
MATCH p=shortestPath(
(b name:"Kevin Bacon")-[*1..6]-(c)
)
WHERE c.name <> "Kevin Bacon"
RETURN count(c)
```