





**HephIA**  
Scalable Intelligence

# NoSQL

# 5- Jointures(lookup)

## 5- Jointures

### Comment faire des jointures en MongoDB?

- Aggregation pipeline
- Expression de traitement: **\$lookup**

Exemple: Collecter les commentaires pour un post

```
[
  {
    "title": "my first post",
    "author": "Jim",
    "likes": 5
  },
  {
    "title": "my second post",
    "author": "Jim",
    "likes": 2
  },
  {
    "title": "hello world",
    "author": "Joe",
    "likes": 3
  }
]
```

#### Posts

```
[
  {
    "postTitle": "my first post",
    "comment": "great read",
    "likes": 3
  },
  {
    "postTitle": "my second post",
    "comment": "good info",
    "likes": 0
  },
  {
    "postTitle": "my second post",
    "comment": "i liked this post",
    "likes": 12
  },
  {
    "postTitle": "hello world",
    "comment": "not my favorite",
    "likes": 8
  },
  {
    "postTitle": "my last post",
    "comment": null,
    "likes": 0
  }
]
```

#### Commentaires

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      localField: "title",
      foreignField: "postTitle",
      as: "comments"
    }
  }
])
```

#### Requête

- **from**: la collection cible de notre jointure
- **localField**: le champ que nous voulons joindre dans la collection locale (la collection sur laquelle on exécute la requête .ie **posts**)
- **foreignField**: le champ que nous voulons joindre dans la collection cible (dans notre cas **comments**)
- **as**: the nom du champ de résultat



## 5- Jointures

### Comment faire des jointures en MongoDB?

- Aggregation pipeline
- Expression de traitement: **\$lookup**

Exemple: Collecter les commentaires pour un post

```
[
  {
    "title": "my first post",
    "author": "Jim",
    "likes": 5
  },
  {
    "title": "my second post",
    "author": "Jim",
    "likes": 2
  },
  {
    "title": "hello world",
    "author": "Joe",
    "likes": 3
  }
]
```

**Posts**

```
[
  {
    "postTitle": "my first post",
    "comment": "great read",
    "likes": 3
  },
  {
    "postTitle": "my second post",
    "comment": "good info",
    "likes": 0
  },
  {
    "postTitle": "my second post",
    "comment": "i liked this post",
    "likes": 12
  },
  {
    "postTitle": "hello world",
    "comment": "not my favorite",
    "likes": 8
  },
  {
    "postTitle": "my last post",
    "comment": null,
    "likes": 0
  }
]
```

**Commentaires**

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      localField: "title",
      foreignField: "postTitle",
      as: "comments"
    }
  }
])
```

**Requête**

```
[
  {
    "title": "my first post",
    "author": "Jim",
    "likes": 5,
    "comments": [
      {
        "postTitle": "my first post",
        "comment": "great read",
        "likes": 3
      }
    ]
  },
  ...
]
```

**Résultat**



#### Jointure avec conditions

Exemple: Collecter les commentaires pour chaque post lorsque les commentaires sont liés que les posts

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      let: { post_likes: "$likes", post_title: "$title" },
      pipeline: [
        {
          $match:
          {
            $expr:
            {
              $and:
              [
                { $gt: ["$likes", "$$post_likes"] },
                { $eq: ["$$post_title", "$postTitle"] }
              ]
            }
          }
        },
        {
          $project:
          {
            _id: 1,
            post_likes: 1,
            post_title: 1
          }
        }
      ],
      as: "comments"
    }
  }
])
```

#### Requête

- let (optional)**: une expression déclarant les variables à utiliser dans l'étape de pipeline. Cela permet au pipeline, d'accéder aux champs de la collection locale (**posts** dans notre cas)
- pipeline**: un aggregation pipeline à exécuter sur la collection à joindre (**comments**)



#### Jointure avec conditions

Exemple: Collecter les commentaires pour chaque post lorsque les commentaires sont likés que les posts

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      let: { post_likes: "$likes", post_title: "$title" },
      pipeline: [
        {
          $match:
          {
            $expr:
            {
              $and:
              [
                { $gt: ["$likes", "$$post_likes"] },
                { $eq: ["$$post_title", "$postTitle"] }
              ]
            }
          }
        },
        {
          $project:
          {
            _id: 1,
            postTitle: 1,
            comment: 1,
            likes: 1
          }
        }
      ],
      as: "comments"
    }
  }
])
```

#### Requête

#### Résultat

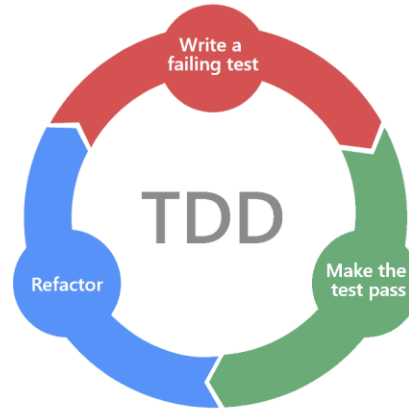
```
[
  {
    "title": "my second post",
    "author": "Jim",
    "likes": 2,
    "comments": [
      {
        "postTitle": "my second post",
        "comment": "i liked this post",
        "likes": 12
      }
    ]
  },
  {
    "title": "hello world",
    "author": "Joe",
    "likes": 3,
    "comments": [
      {
        "postTitle": "hello world",
        "comment": "not my favorite",
        "likes": 8
      }
    ]
  }
]
```



# Products Infos



# Test your code!



### Test Driven Development

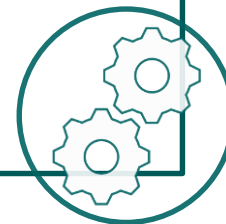
- Exactitude du code aujourd'hui et surtout demain
- Eviter de reproduire des bugs passés
- Refactoriser en toute confiance

Pourquoi?



- Tests Unitaires
- Tests d'Intégration
- Tests Fonctionnels
- Tests de Performance

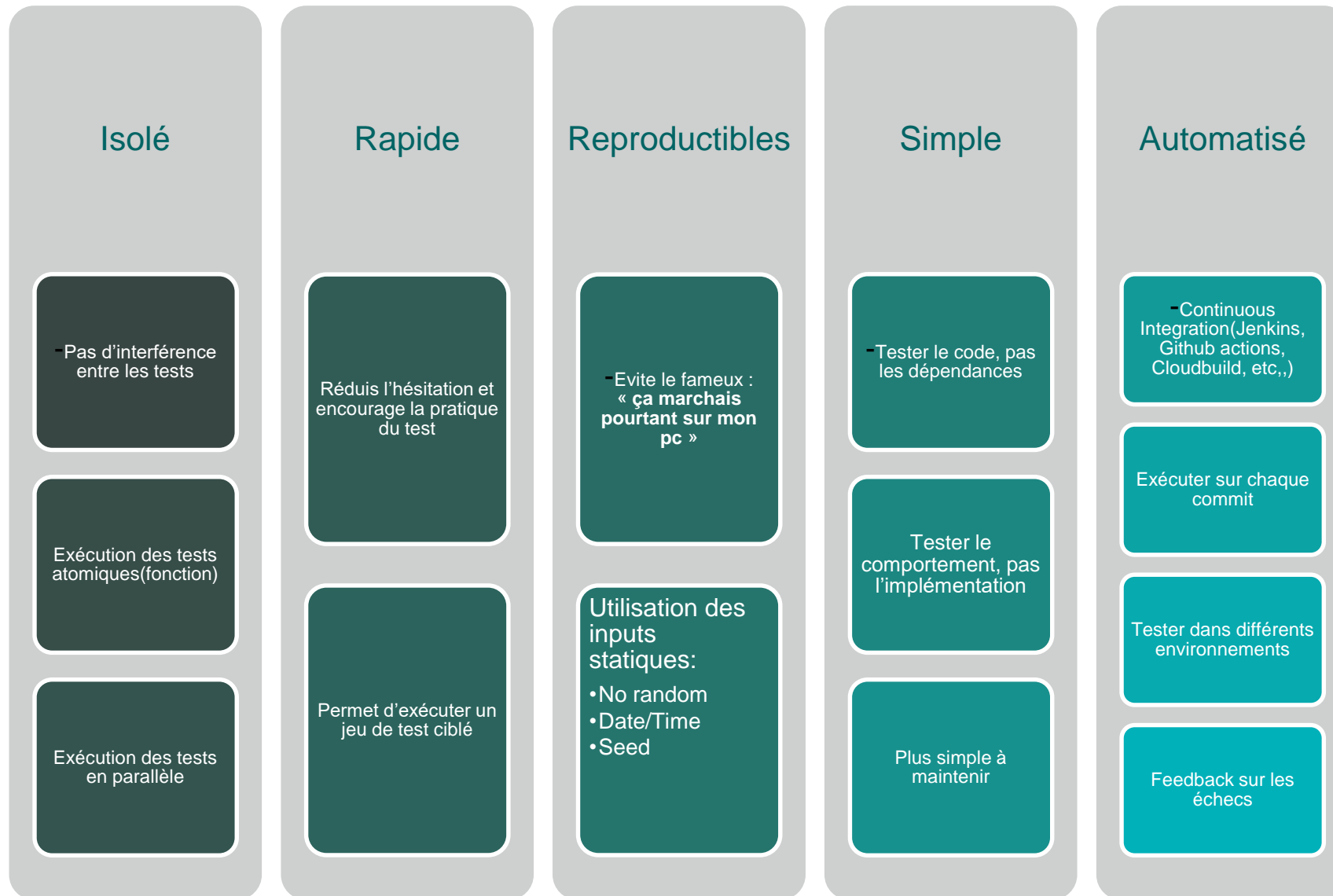
Comment?



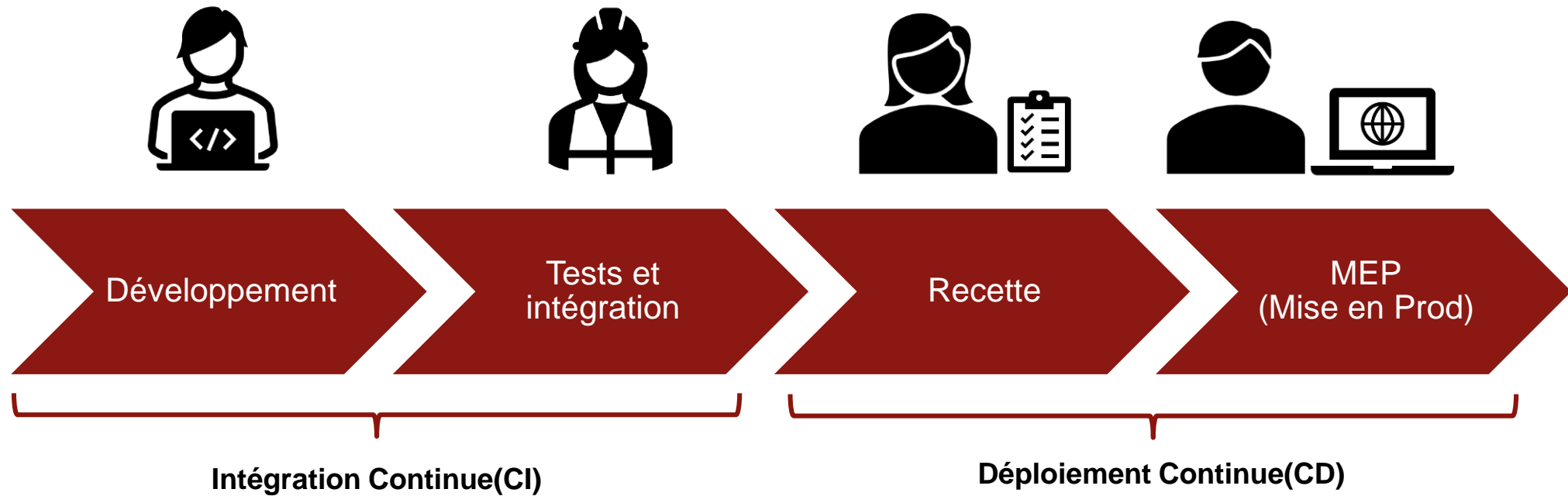
- Isolation
- Reproductibilité
- Rapidité
- Simplicité
- Automatisation(CI)

Bonnes pratiques





# CI/CD



- ☐ Découpage en micro-services
- ☐ Codes gérés et centralisé par SCM(GitHub, GitLab)
- ☐ Build, Tests(Unitaires, Intégration, QA, dépendances)

- ☐ Tests (recette, qualif, pré-prod)
- ☐ MEP rapide une fois tests validés



# Go and Test it!

# BDD Graphe?

Scan QR-code  
for questions!



[app.klaxoon.com/join/NHHGTDX](https://app.klaxoon.com/join/NHHGTDX)





Avantages	Inconvénients
ACID friendly contrairement aux autres BDD NoSQL	Ne scale pas horizontalement
Impressionnant quand les données ont une structure graphe	Cheminer à travers un graph avec des nœuds partagés à travers divers serveurs peut être très couteux
Performances constantes quand la taille des données croît	
Flexibilité et agilité	

**Use cases:**

- Données très connectées et reliées
- Réseaux sociaux
- Routing, spatial, map apps
- Systèmes de recommandation

**Use cases non adaptés:**

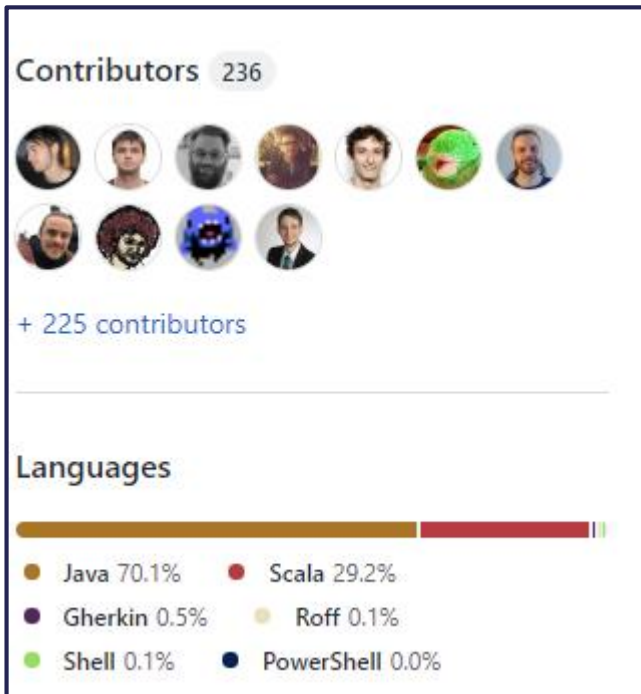
- Les cas couverts par les autres cas d'usage NoSQL
- Besoin de scaler horizontalement
- Mise à jour toutes les données ou un échantillon de nœuds avec paramètre donné

**Fiabilité > Disponibilité**





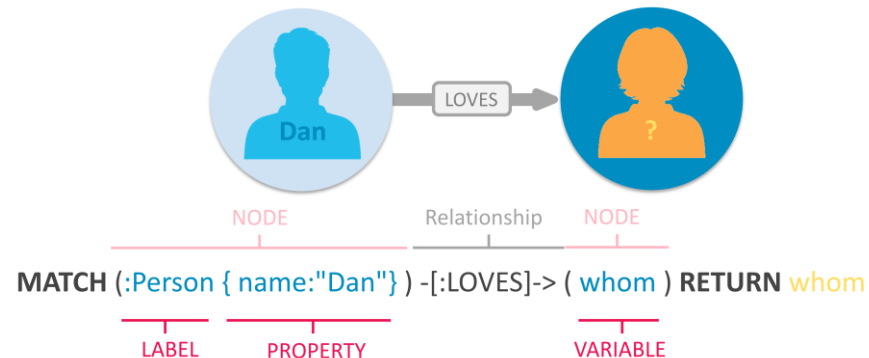
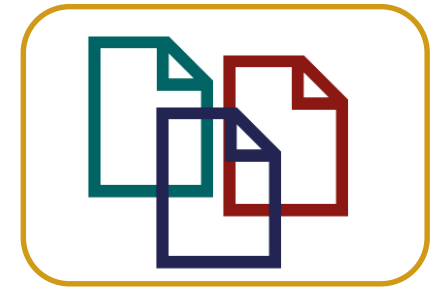
Communauté large et  
très active



### GraphGists

Use case and industry specific graph examples  
designed to inspire you towards your Graph  
epiphany.

Données sauvegardées  
dans des fichiers



DSL(Domain Specific Language):



# Penser graphe!

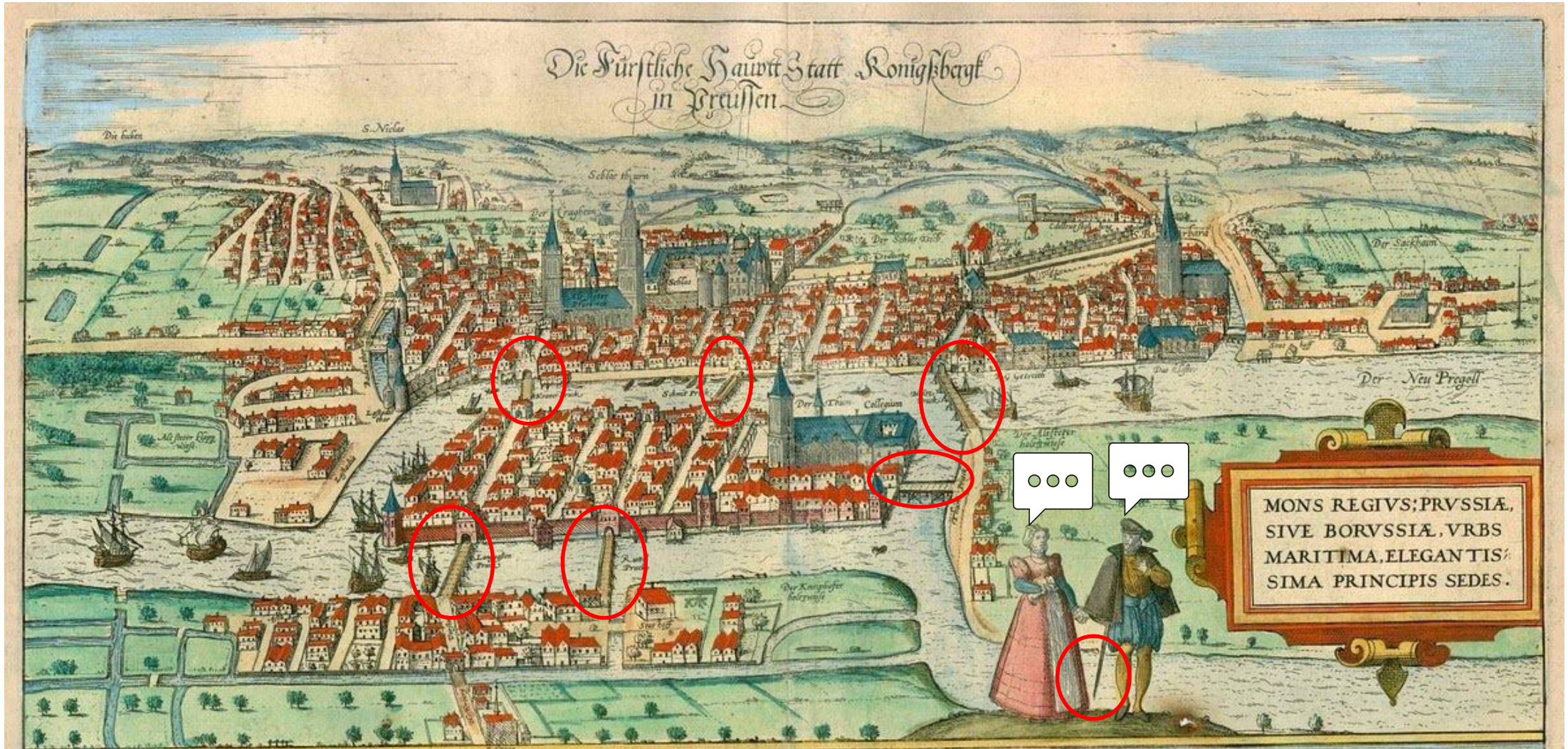
## Début XVIIIe, Königsberg, Prusse-Orientale



Actuellement Kaliningrad, Russie

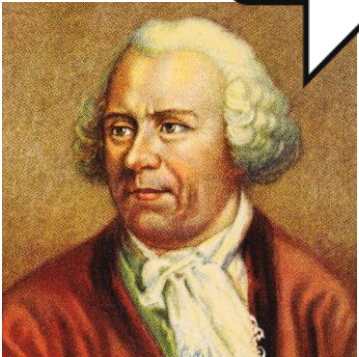






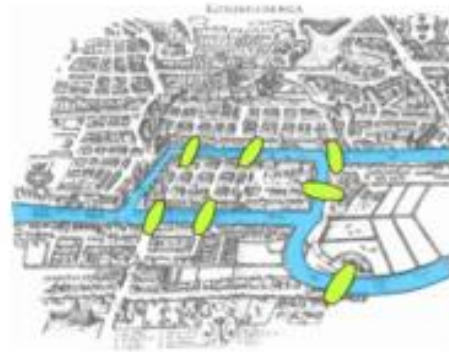


Peut on faire le tour de la ville en empruntant tous les ponts sans jamais en un prendre un **plus d'une fois**?

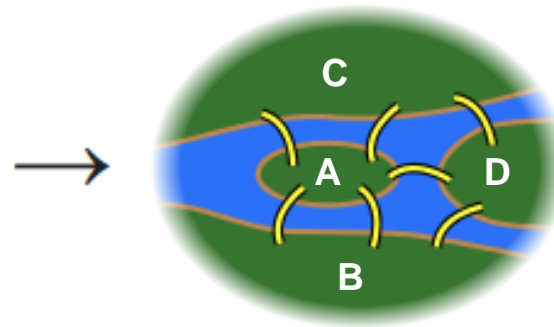
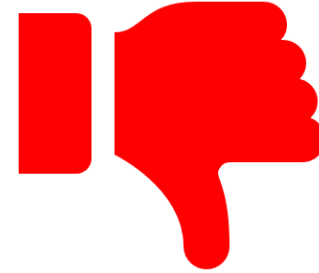


Leonhard Euler  
(1701- 1783)

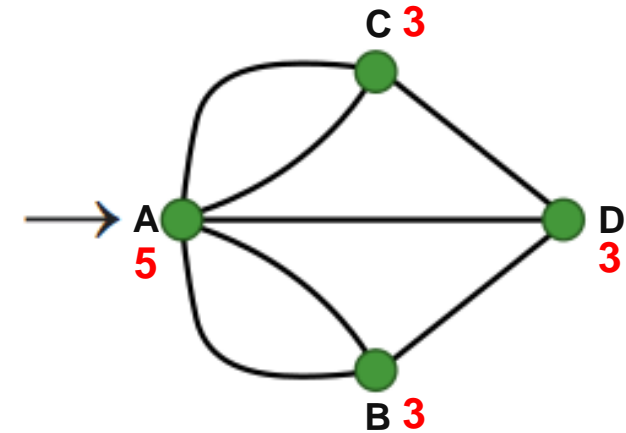
Simplifions le problème à l'extrême!



Réponse :



4 zones à visiter



Surfaces terrestres =>  
Sommets  
Ponts => arêtes

Démontré par  
Carl Hierrolzer  
(1840 - 1871)



- Un graphe connexe admet un **parcours eulérien** si et seulement si ses sommets sont tous de **degré pair** sauf au plus deux.
- Un graphe connexe admet un **circuit eulérien** si et seulement si tous ses sommets sont de **degré pair**.



### Prise de décision



### Optimisation

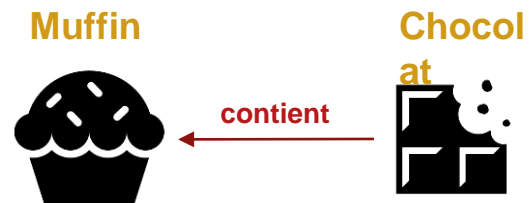
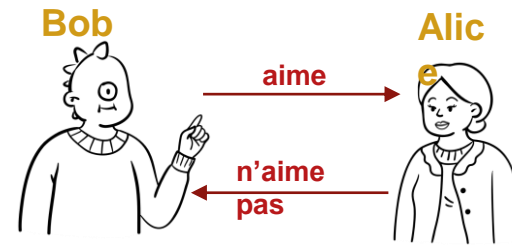
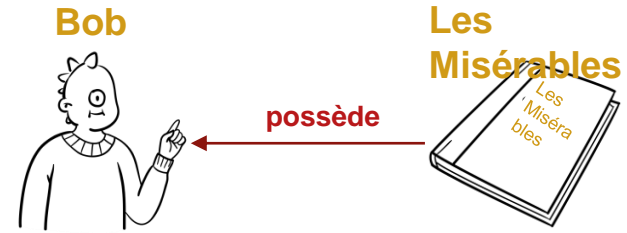
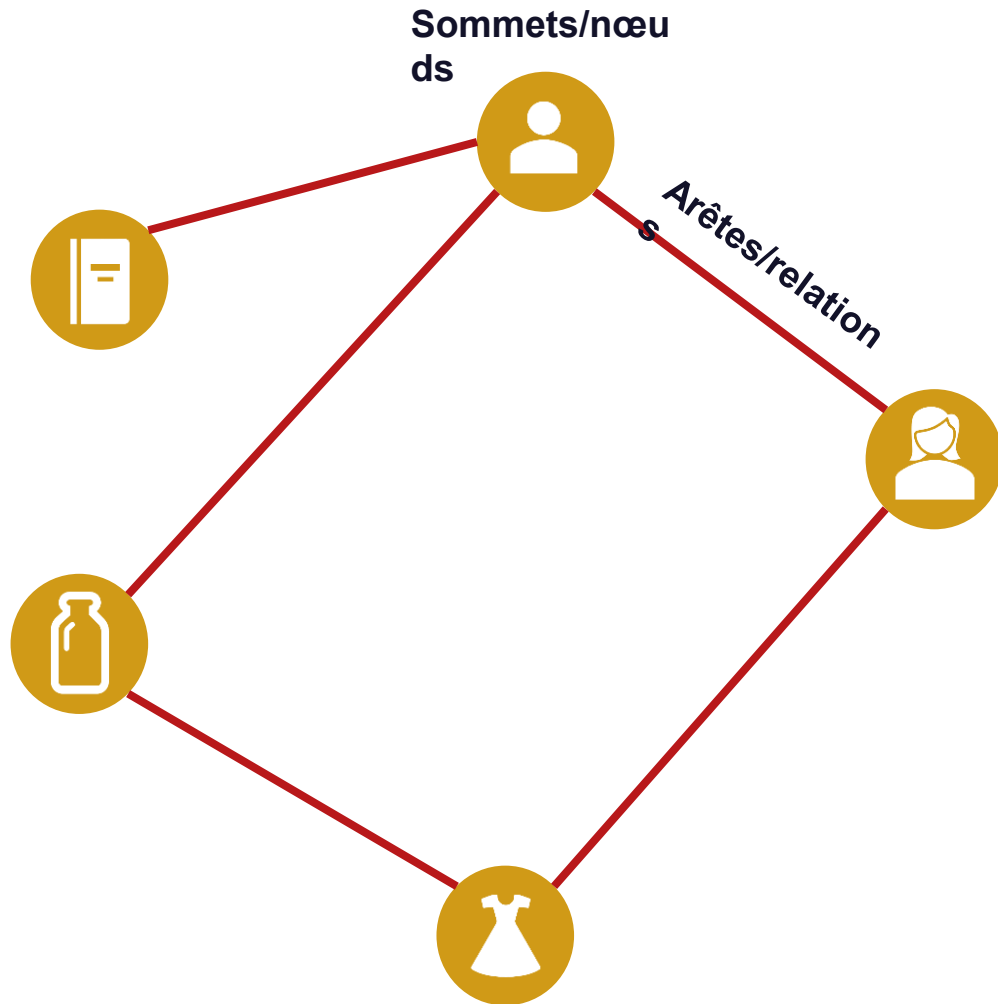


### Gestion des réseaux(transports, énergie, informatique)

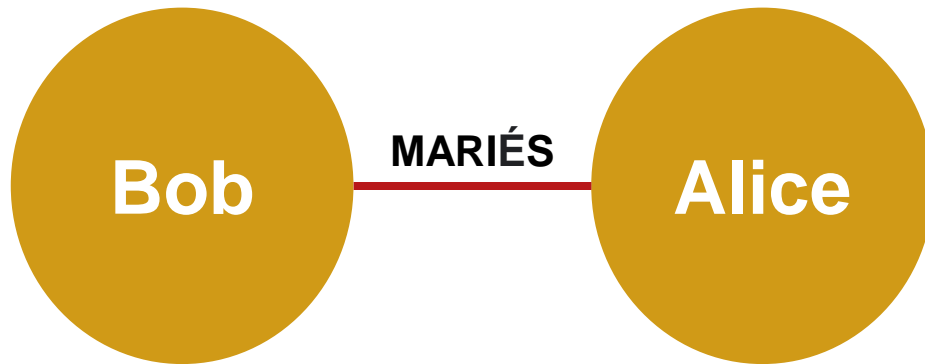


# Graphes: notions de base



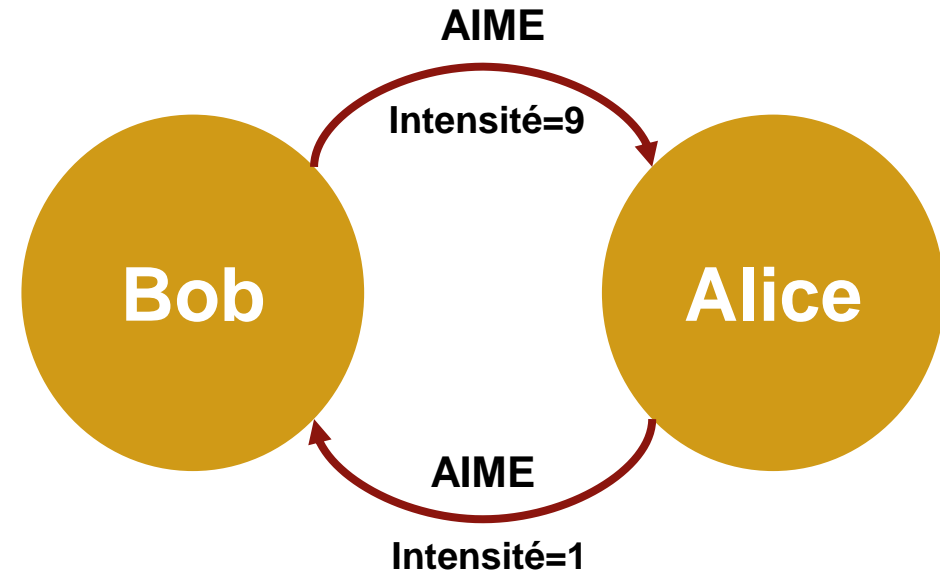


## Graphe Non-Orienté



Relation **bidirectionnelle/symétrique**

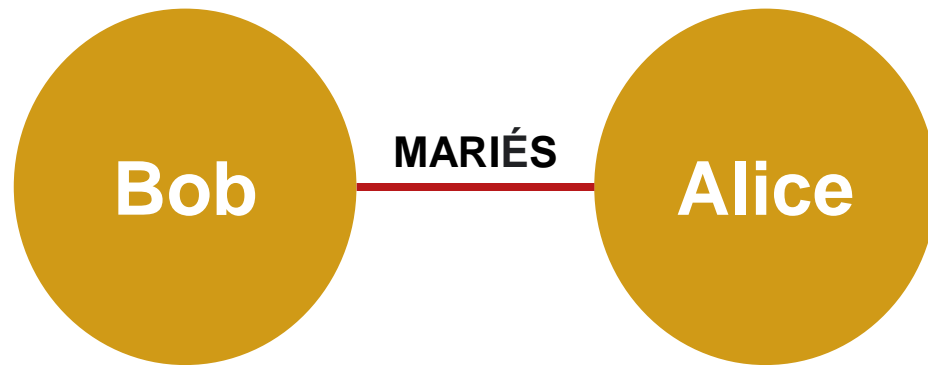
## Graphe Orienté



Relation **asymétrique**

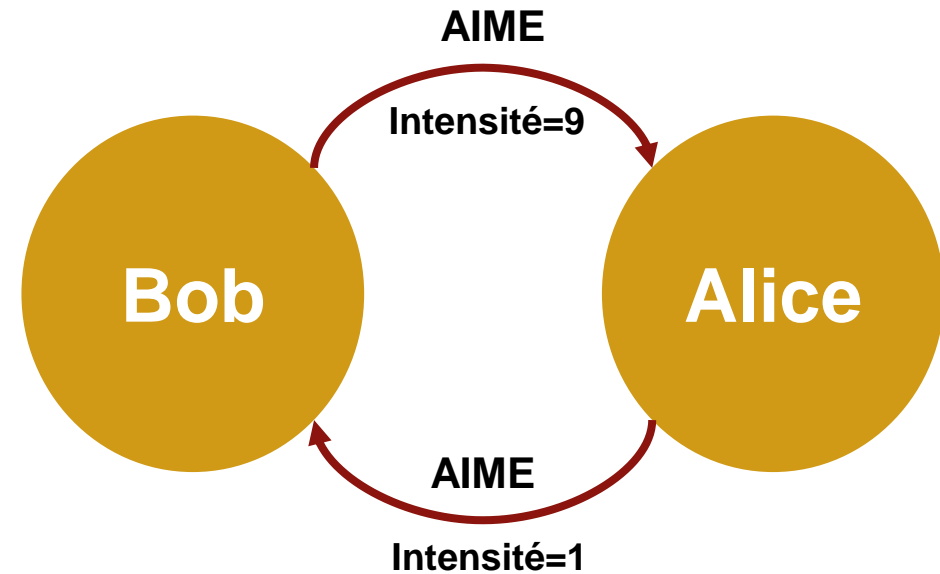


## Graphe Non Pondéré



Relation **bidirectionnelle/symétrique**

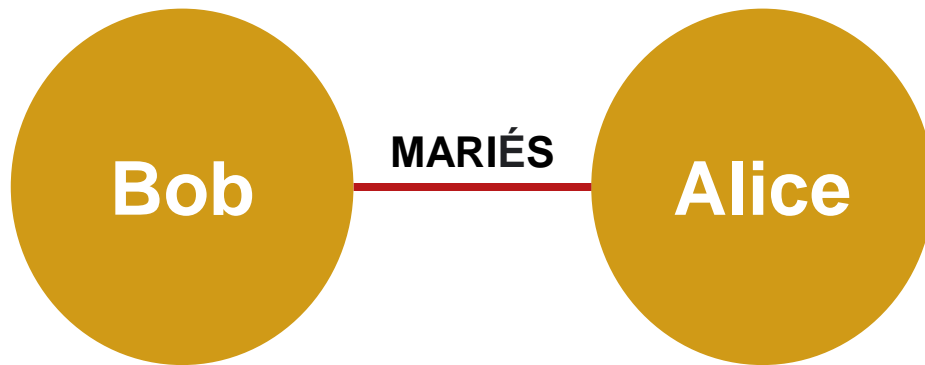
## Graphe Pondéré



Relation **asymétrique**

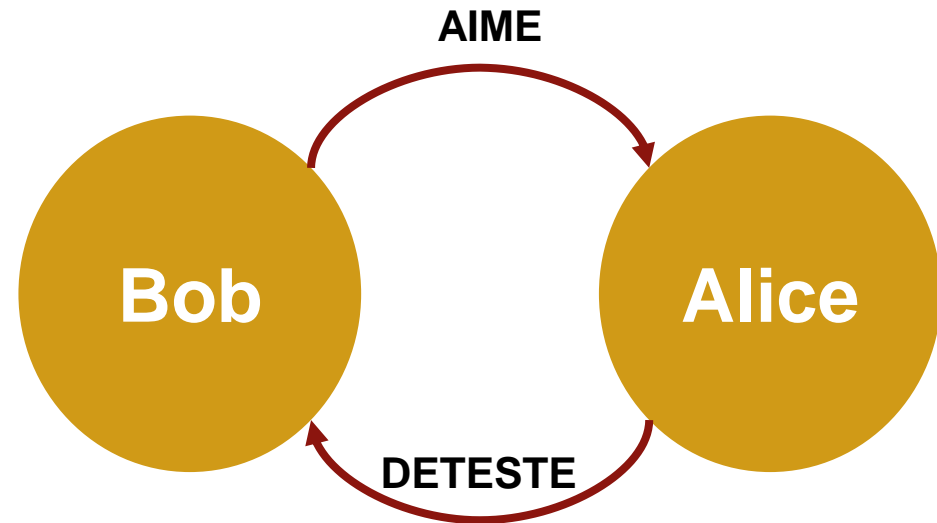


## Graphe Non Pondéré



Relation **bidirectionnelle/symétrique**

## Graphe Non Pondéré



Relation **asymétrique**

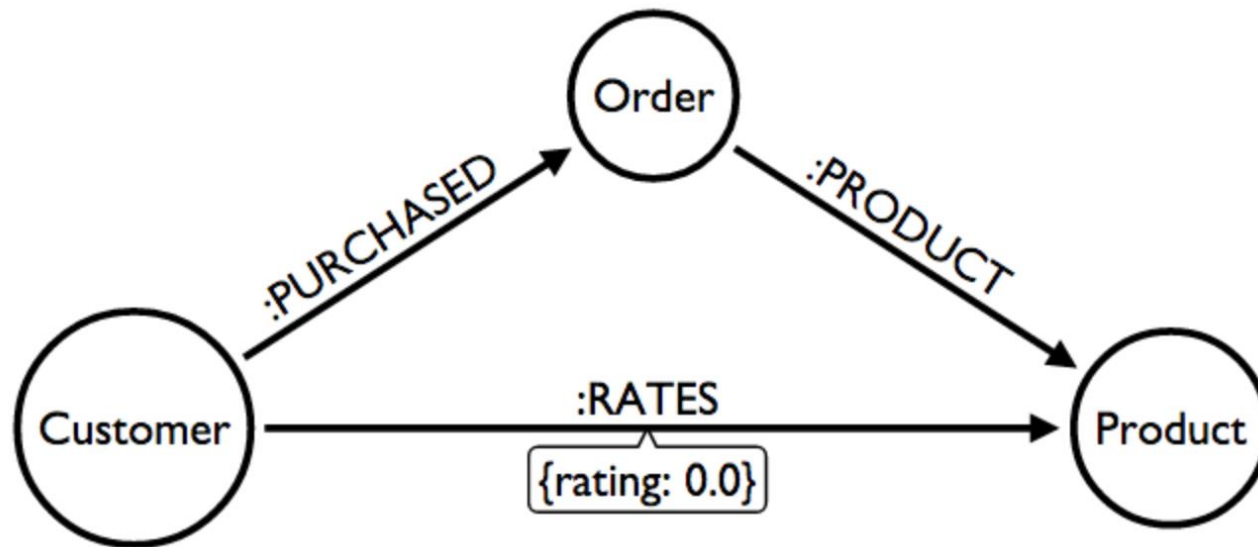


# Graphs are everywhere!

# Graphs are everywhere!

## Widely used Graph Database





Walmart 

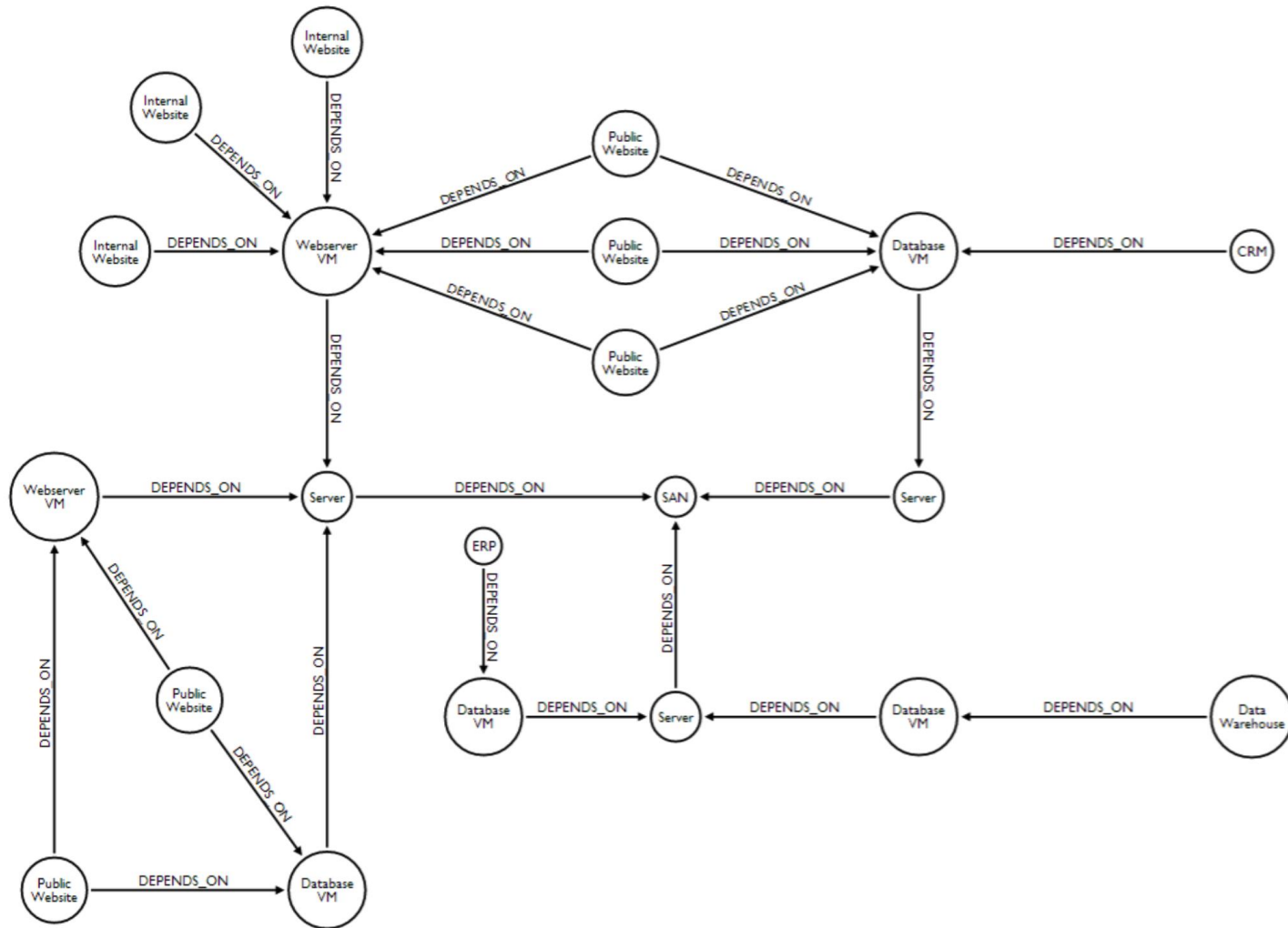
 Medium

mestic



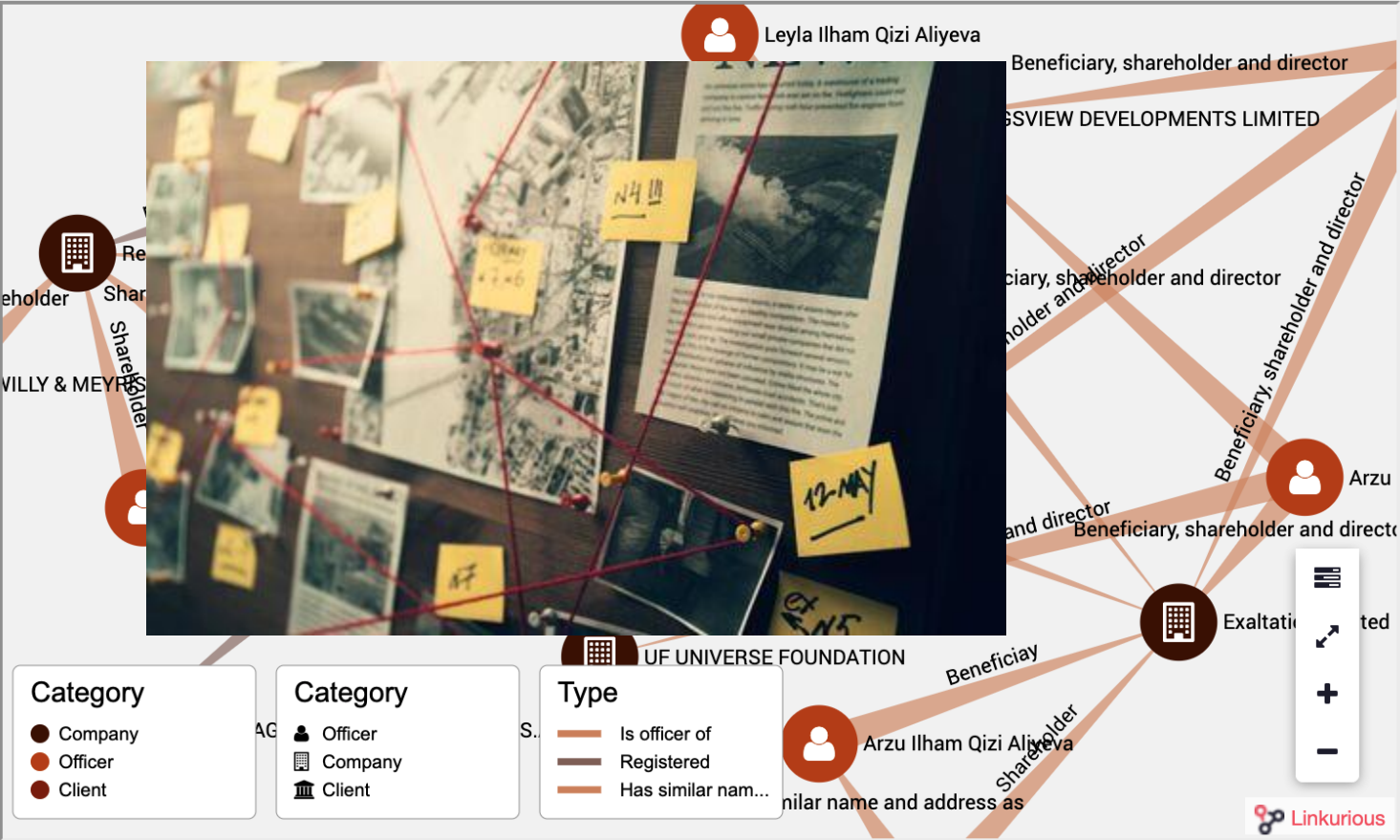
# Graphs are everywhere!

## Network and IT operations

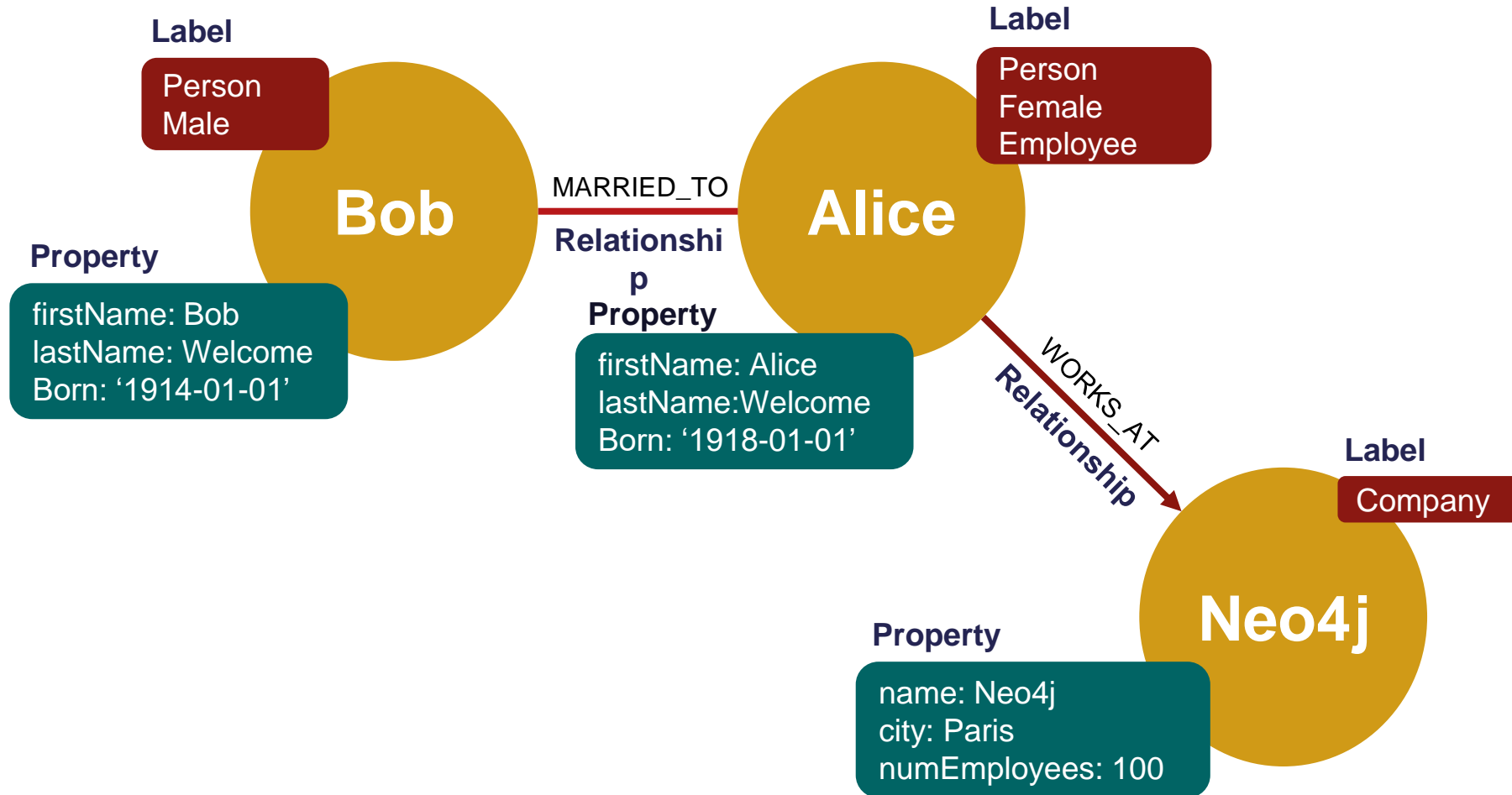




PANAMA PAPERS



# Modélisation





**Répondre aux cas  
d'usages principaux  
de votre application**



**-Fournir les meilleurs  
performances de  
requêtes Cypher  
pour les cas d'usage**



-1- Comprendre le domaine et définir les cas d'usages spécifiques

-3- Confronter les cas d'usages aux modèle initial

-5- Tester les cas d'usages, et les performances

-7- Refactoriser le graphe et retester en utilisant Cypher

-2- Développer le modèle de données graphe initial

- Modéliser les sommets
- Modéliser les relations entre



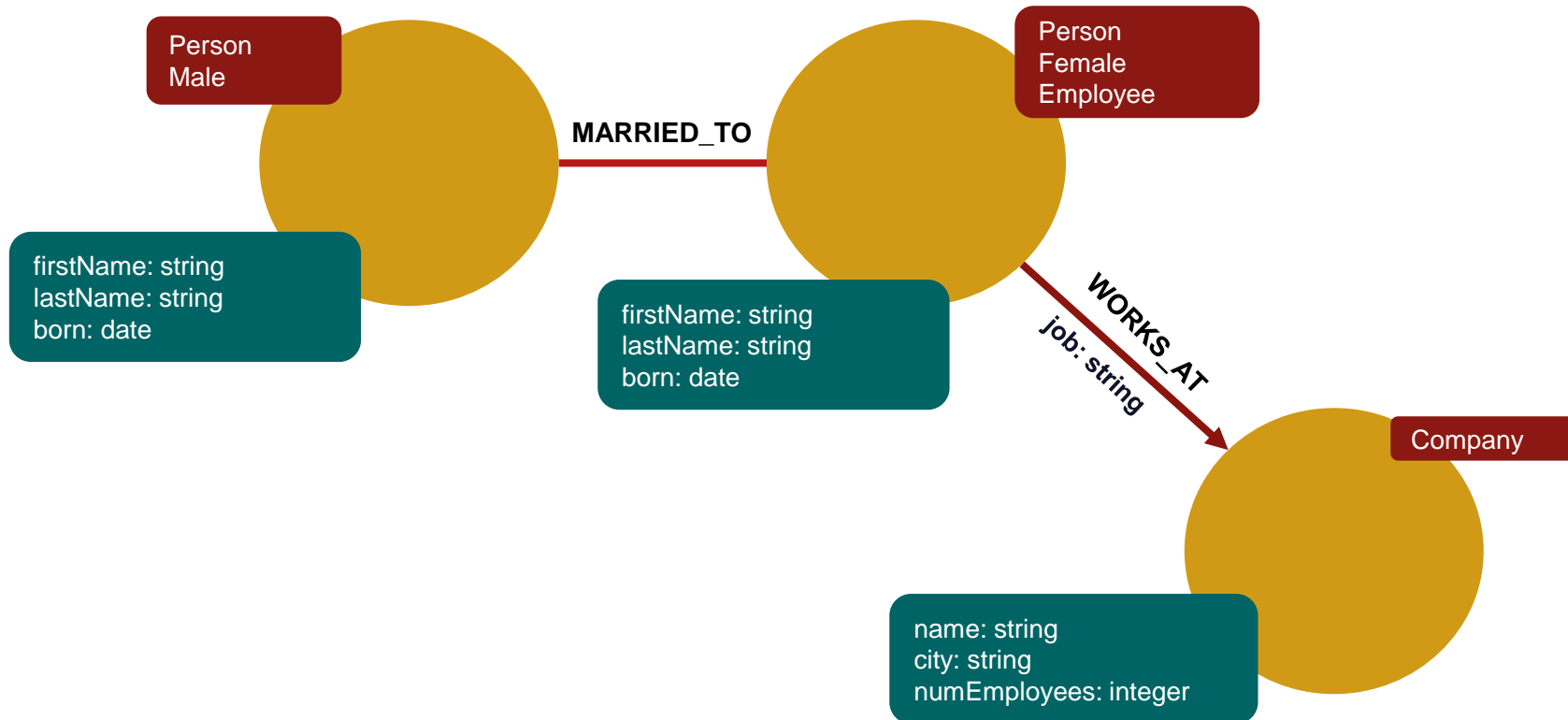
-4- Créer le graphe (modèle d'instance) avec des données de test en utilisant Cypher



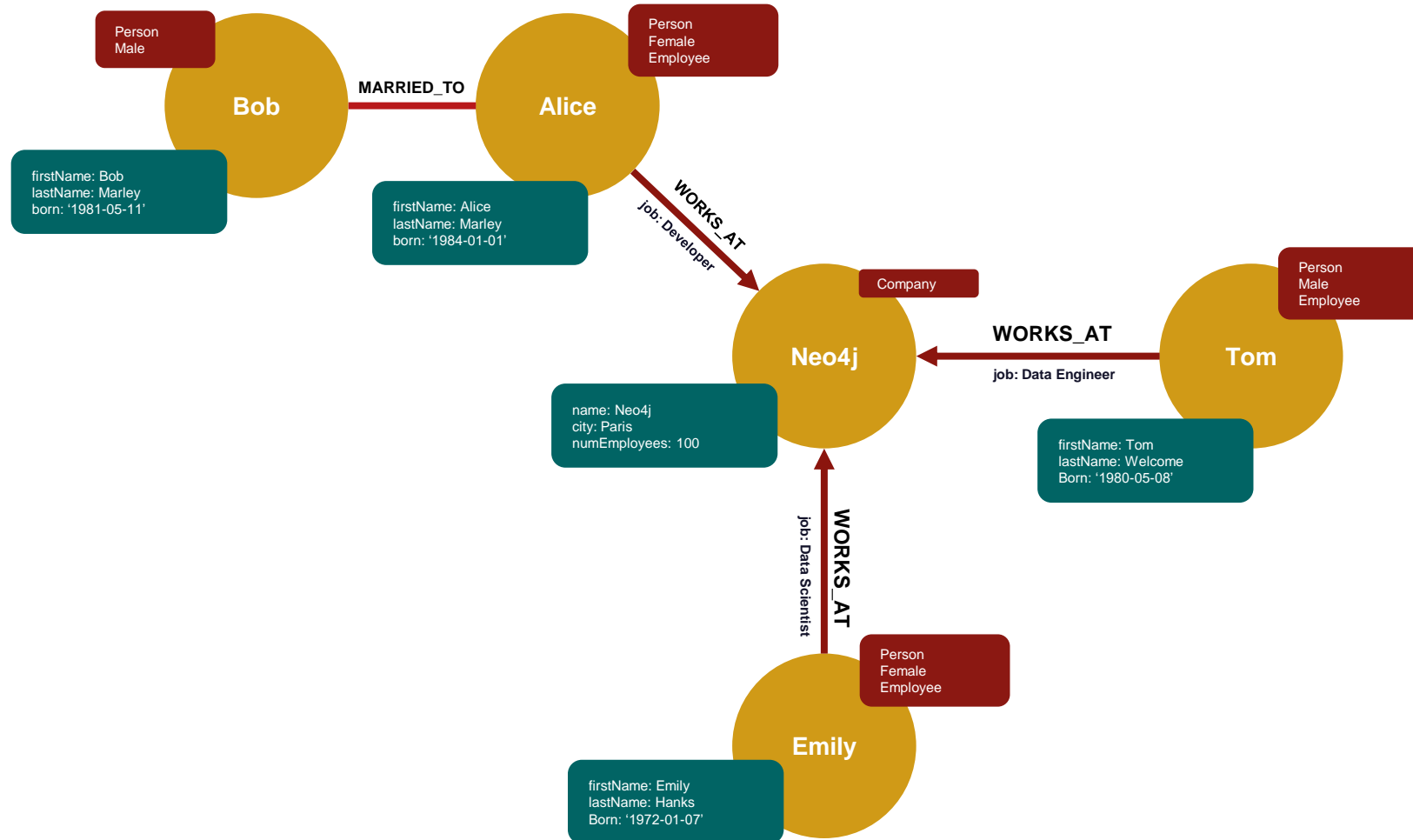
-6- Refactoriser (améliorer) le modèle du à des changements dans la vie de l'application/perforances



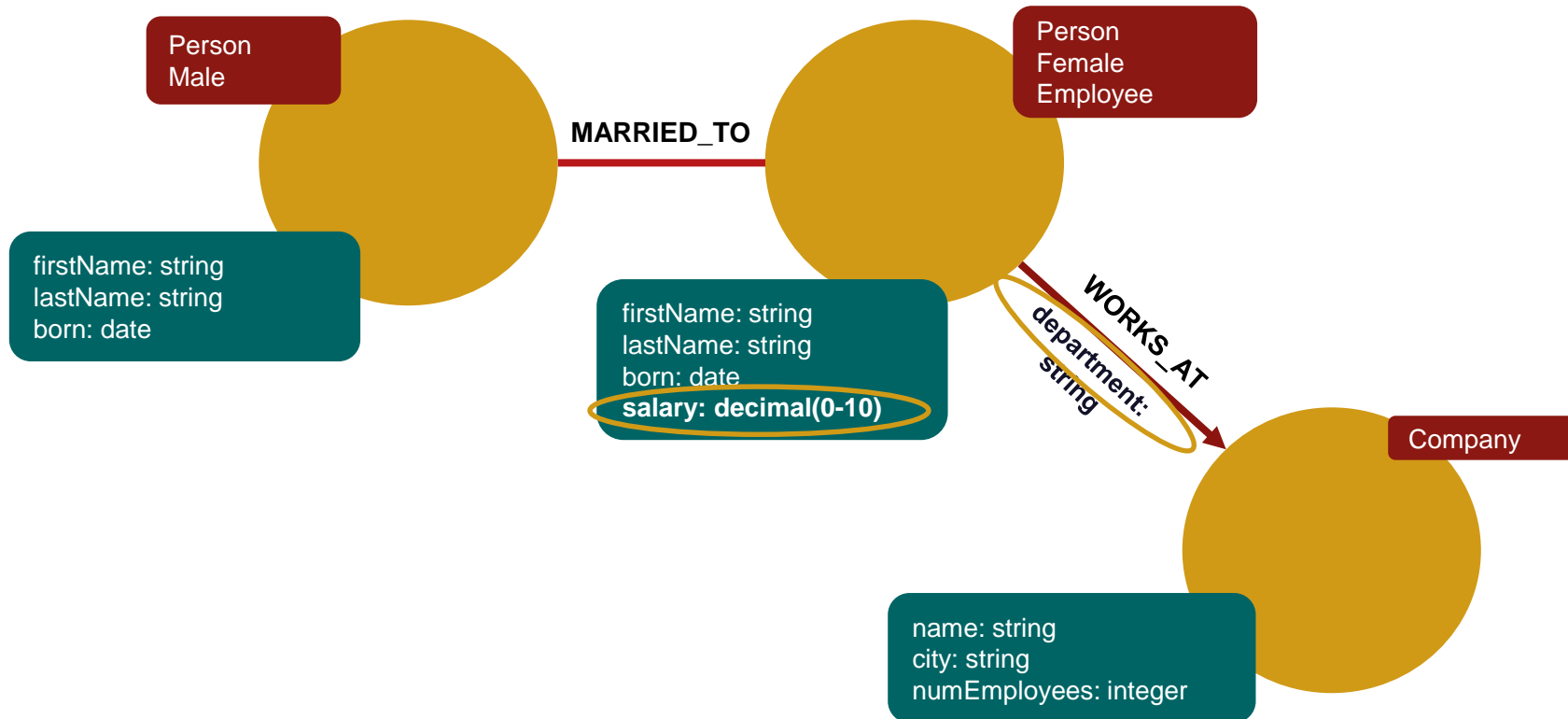
## Modèle de données



### Modèle d'instance

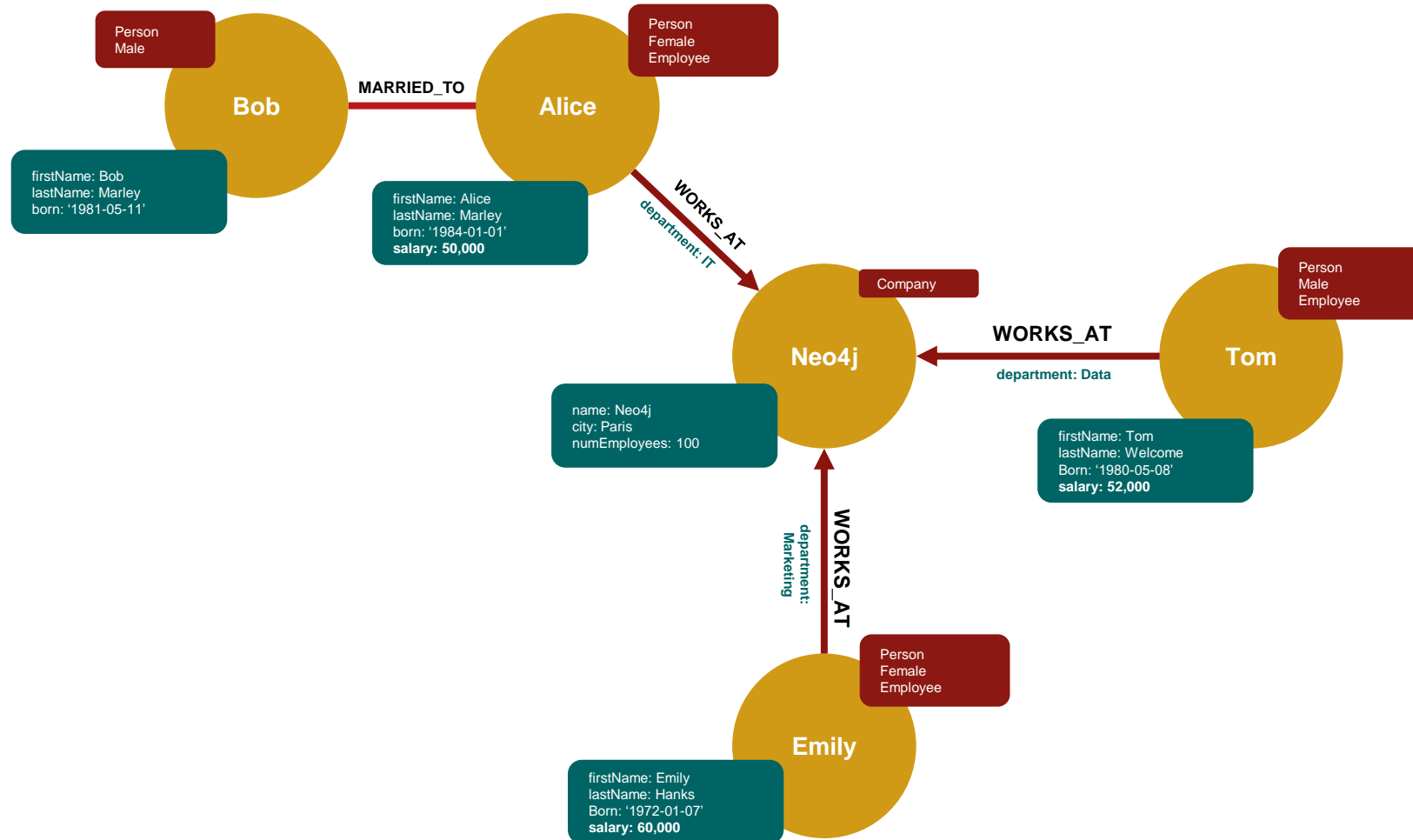


### Modèle de données



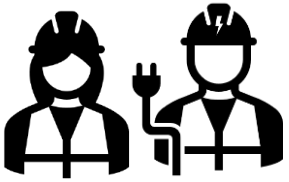


## Modèle d'instance





## Compréhension du domaine



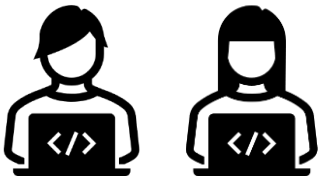
Parties prenantes



Décrire l'application en détail



Identifier et prioriser les cas d'usages



Développeurs de l'application



Identifier les utilisateurs(personnes, systèmes, etc...)



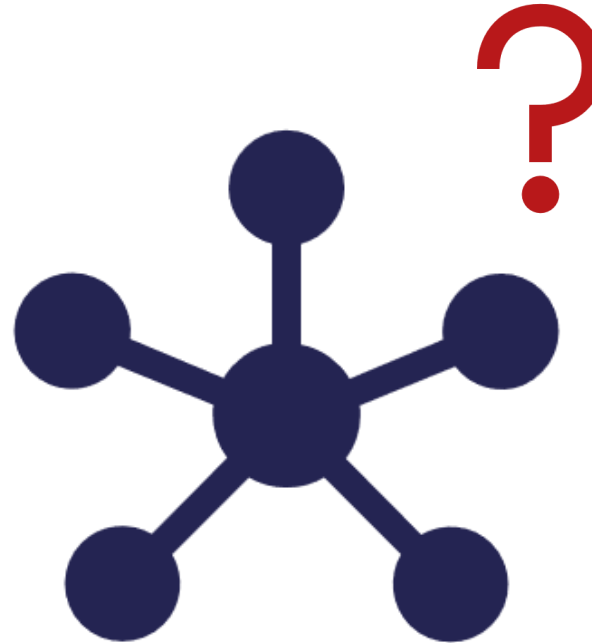
# The Movie Graph



## Movie Graph

### Cas d'usage

1. Quels sont les acteurs d'un film?
2. Dans quel film a joué un personne?
3. Qui est la personnes la plus jeune dans un film?
4. Quel rôle a joué une personne dans un film?
5. Quels sont les réalisateurs d'un film?

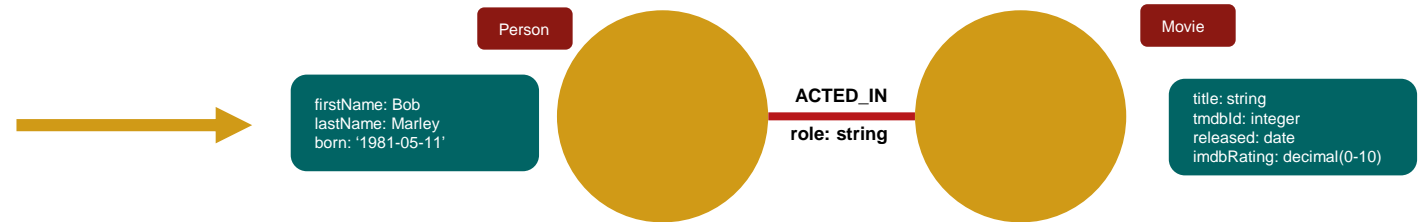




## Movie Graph

### Cas d'usage

1. Quels sont les **acteurs** d'un **film**?
2. Dans quel **film** a joué un **personne**?
3. Qui est la **personnes** la plus jeune dans un film?
4. Quel rôle a joué une **personne** dans un **film**?
5. Quels sont les **réalisateurs** d'un **film**?



**Properties**  
name: string  
born: integer

**Label: Person**

**Label: Movie**

**Properties**  
title: string  
released: integer  
tagline: string

**Nœuds**

**Relations**

```
1 CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
2 CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
3 CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
4 CREATE (LillyW:Person {name:'Lilly Wachowski'})
5
6 CREATE
7 (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
8 (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
9 (LillyW)-[:DIRECTED]->(TheMatrix)
10
11
12
```

**Relationship: ACTED\_IN**

**Properties**  
Roles : string

**Relationship: DIRECTED**



Table

Added 4 labels, created 4 nodes, set 11 properties, created 3 relationships, completed after 4 ms.



Clause	Rôle
MATCH	spécifie un pattern à chercher dans la base de données
RETURN	spécifie le contenu souhaité dans le résultat
WHERE	n'est pas une clause en elle-même, mais un complément à la clause MATCH
LIMIT	restreint le nombre de lignes retournées

1 MATCH (keanu {name: "Keanu Reeves"})

2 RETURN keanu

Graph

Table

Text

Code

Keanu Reeves

1 MATCH (people:Person)

2 RETURN people.name

3 LIMIT 10

Table

Text

Code

	people.name
1	"Keanu Reeves"
2	"Carrie-Anne Moss"
3	"Laurence Fishburne"
4	"Hugo Weaving"

1 MATCH (nineties:Movie)

2 WHERE nineties.released ≥ 1990 AND nineties.released < 2000

3 RETURN nineties.title

Table

Text

Code

	nineties.title
1	"The Matrix"
2	"The Devil's Advocate"
3	"A Few Good Men"
4	"As Good as It Gets"



Symbole	Relation
--	Relation quelconque
-->,<--	Relation dirigée
-[r]->, -[r]-, <-[r]-	Relation nommée
-[:TYPE]->, -[:TYPE]-, < -[:TYPE]-	Relation typée

1 MATCH (mat

2 RETURN dir

Table

Text

Code

directo

"Andy

"Lana

1 MATCH (people:Person)-[relatedTo]-(:Movie {title: "The Matrix"})

2 RETURN people.name, Type(relatedTo), relatedTo

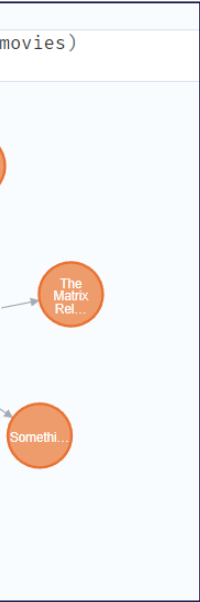
Table

Text

Code

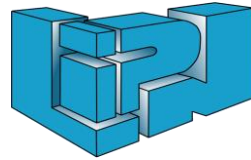
	people.name	Type(relatedTo)	relatedTo
1	"Emil Eifrem"	"ACTED_IN"	{ "identity": 7, "start": 8, "end": 0, "type": "ACTED_IN", "properties": { "roles": [ "Emil" ] }, "elementId": "7", "startNodeElementId": "8", "endNodeElementId": "0" }
2	"Joel Silver"	"PRODUCED"	{

Started streaming 19 records after 1 ms and completed after 2 ms.





## Nos partenaires



**HephIA**  
Scalable Intelligence

Brice Fotzo

[brice.fotzo@hephia.com](mailto:brice.fotzo@hephia.com)

+33 6 60 88 46 39

[hephia.com](http://hephia.com)

