Bases de données aujourd'hui et bases de données NoSQL

Antoine Augusti – antoine.augusti.fr

Thibaud Dauce – thibaud.dauce.fr

2 mars 2023

Architecture classique

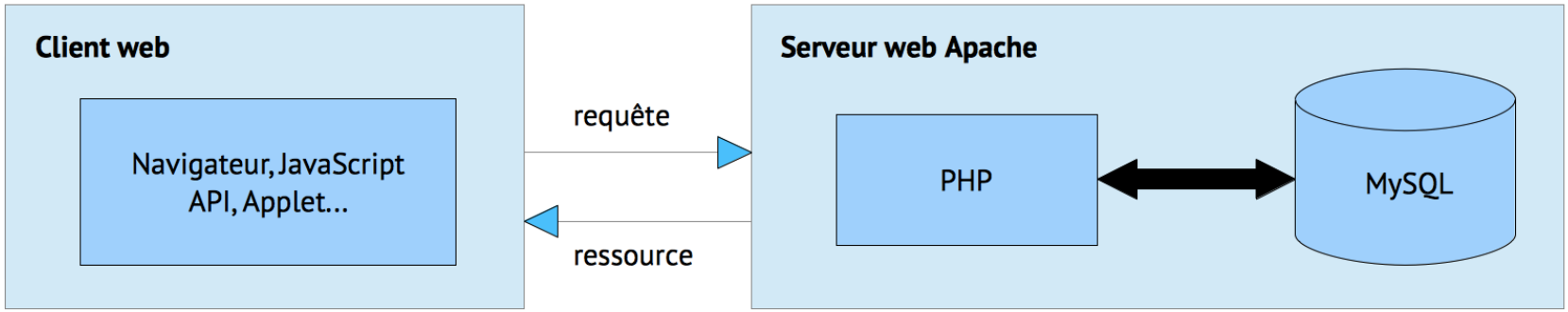

Figure: La stack LAMP.

Stacks habituelles :

- LAMP : Linux, Apache, MySQL, PHP — la plus commune ;
- LEMP : Linux, Nginx, MySQL, PHP-FPM — commence à remplacer LAMP ;
- MERN : MySQL, Express, ReactJS, Node.js — pour du JavaScript côté serveur.

Bilan architecture classique

Avantages :

- Rapide à mettre en place (installation en un clic généralement) ;
- Optimisation possible par **scalabilité verticale** (augmenter les capacités physiques du serveur)

Inconvénients :

- Peu de tolérence à la charge
- Augmentation complexe des peformances

Qu'est-ce qu'un ORM ?

Définition : ORM

L'**O**bject-**R**elationnal **M**apping est une technique qui simule une base de données orientée objet à partir d'une base de données relationnelle.

- Fait la liaison entre le monde relationnel dans la couche stockage et le monde objet dans l'application ;
- Facilité de développement : *pas besoin* d'une connaissance poussée du SQL ;
- Facilite les interactions avec la base de données pour les développeurs.

Les limites des ORM

Toujours **beaucoup** moins performant que des requêtes SQL optimisées dans les cas complexes.

ORM : exemple de requêtes

```java
// Création d'un utilisateur
Map<String, String> userData = new HashMap<String, String>();
userData.put("prenom", "Antoine");
userData.put("nom", "Augusti");
User user = User.create(userData);

// Sélection des utilisateurs majeurs et articles qu'ils ont écrits
ArrayList<User> users = User.with("articles")
    .where("age", ">=", 18)
    .get();

// Suppression des utilisateurs vivant à Paris
User.where("ville", "Paris").delete();

// Les derniers articles d'un utilisateur (3ème page)
final int NOMBRE_ARTICLES_PAR_PAGE = 10;
ArrayList<Article> articles = Article.whereUserId(user.getId())
    .latest()
    .skip(2*NOMBRE_ARTICLES_PAR_PAGE)
    .take(NOMBRE_ARTICLES_PAR_PAGE)
    .get();
```

Listing 1: Quelques requêtes basiques avec un ORM imaginaire.

D'autres ORM : Hibernate (Java), Eloquent (PHP), SQLAlchemy (Python), Mongoose (Node.js)...

Problèmes liés au relationnel

- Service réparti sur plusieurs continents
- Accès concurrent de plusieurs centaines de milliers de personnes
- Stockage de petites variations dans les schémas
- Temps de réponse
- Traitement de structures de données complexes en base de données

Pourquoi changer du relationnel ?

Besoin d'une alternative vers les années 2004 avec l'arrivée du *Big Data*.

- Des volumes de données important (plusieurs gigas, téraoctets, pétaoctets) ;
- Un nombre de transactions très important, une forte demande de disponibilité et de temps de réponse ;
- Des bases de données réparties sur plusieurs centres de données ou continents ;
- Préférence pour l'ajout de petites machines plutôt qu'une configuration poussée des BDs (concept de **scalabilité horizontale**).

Pourquoi ne pas changer du relationnel ?

- Technologie éprouvée ;
- Technologie **très** éprouvée ;

- Gestion jusqu'à 2To de RAM sur une seule machine ;
- Réplication de PostgreSQL sur plusieurs continents via master read-write et slaves read-only ;
- Ajout du type JSON pour relaxer les schémas ;
- Secteurs où la fiabilité des données est plus importante que le reste.

Relâchement des contraintes de transactions

Les BDs relationnelles et NoSQL font des arbitrages différents dans le théorème CAP :

- *Consistency* : chaque lecture reçoit l'écriture la plus récente ou une erreur ;
- *Availability* : chaque requête reçoit une réponse, sans garantie qu'elle contienne l'écriture la plus récente ;
- *Partition tolerance* : le système continue de fonctionner malgré la perte ou le retard de messages entre les nœuds.

Rappel relationnel

Propriétés (ACID) :

- **Atomicité :** Exécuter tout ou rien ! Revenir à l'état de départ en cas d'erreur ;
- **Cohérence :** La base passe d'un état cohérent à un autre état cohérent ;
- **Isolation :** Si les transactions s'exécutent simultanément, alors chacune doit demeurer indépendante de l'autre ;
- **Durabilité :** Une fois confirmée, la transaction demeure enregistrée même en cas de panne ou problème matériel.

Relationnel vs non-relationnel

Propriétés BASE :

- *Basic Availability* : chaque requête reçoit une réponse ;
- *Soft-state* : réponses approximatives acceptables ;
- *Eventual consistency* : les réponses sont de plus en plus cohérentes, au bout d'un temps le système sera totalement cohérent.

ACID vs BASE

- Relationnel (ACID) préfère la cohérence.
- Non-relationnel (BASE) préfère la disponibilité.

À quoi ressemble une BD NoSQL

- Un SGBD qui n'est pas structuré en tables et dont l'élément de base n'est pas un tuple mais dépend du type de BD NoSQL ;
- Un langage de requête non uniformisé, propre à chaque BD ;
- Une dénormalisation des données où certains enregistrements sont en partie ou entièrement dupliqués ;
- Type de base de données NoSQL à choisir en fonction de l'usage souhaité ;
- Types de base de données NoSQL existants : clé-valeur (Redis), colonnes (Cassandra), recherche (ElasticSearch), documents (MongoDB), graphe (Neo4j), évènements (Event Store)...

Modélisation BD clé-valeur

- **Modélisation :** la plus simple. À une clé, on associe une valeur. La valeur peut être de n'importe quel type (chaîne de caractères, entier, structure, objet sérialisé...).

Des exemples de paires clé-valeur avec des types différents.

Clé	**Valeur**
pays.id-42	{"id" :42,"name" :"Chad"}
statistiques.nombre-visiteurs	1337
configuration.periode-gratuite	false
articles.categories-sport.latest	[22, 45, 67, 200, 87]

BD clé-valeur en détail

- **Opérations** : création d'une paire clé-valeur, suppression, accès à une valeur à l'aide de la clé, incrémentation et décrémentation d'une valeur ;
- On peut définir la durée de vie d'une paire clé-valeur ou adopter une politique *least recently used* ;
- Stockage en RAM pour accélérer les temps de lecture. Mécanisme de reprise en cas de crash ;
- **Cas d'utilisation :** cache d'une autre BD, comptage d'éléments, gestion de files d'attente, opérations ensemblistes...
- **Principaux acteurs :** Redis, Memcached, Riak.

Commandes sous Redis

```
1   # Assignation de chaîne "bonjour" à la clé "cleTexte"
2   redis> SET cleTexte bonjour
3   OK
4   # Récupération de la valeur de la clé "cleTexte"
5   redis> GET cleTexte
6   "bonjour"
7
8   # Incrémentation d'un compteur
9   redis> INCR compteur
10  (integer) 42
11  # Suppression du compteur
12  redis> DEL compteur
13  (integer) 1
14
15  # Création d'une liste avec insertion en fin de liste
16  redis> RPUSH liste "Hello"
17  (integer) 1
18  # Insertion en fin de liste
19  redis> RPUSH liste "World"
20  (integer) 2
```

Listing 2: Quelques commandes Redis en console.

Bases de données orientées documents

- **Modélisation :** aucun schéma fixe, un document peut contenir n'importe quel type d'information ;
- Optimisation horizontale ;
- **Opérations :** utilise le JSON pour les requêtes et pour le stockage ;
- Facile à prendre en main en venant des bases de données relationnelles ;
- **Cas d'utilisation :** base de données principalement utilisées pour du stockage ;
- **Principaux acteurs :** MongoDB, CouchDB, CouchBase.

Exemple d'un document

```json
{
   "id":1500,
   "nom":"Thibaud",
   "prénom":"Dauce",
   "professeur_préférée":{
      "numero":42,
      "prenom":"Géraldine",
      "nom":"Del Mondo",
      "matières": ["BD", "Réseaux"]
   },
   "commentaires":[
      {
        "id":646,
        "contenu":"J'adore la BD."
      },
      {
        "id":647,
        "contenu":"Boule et Bill aussi."
      },
      {
        "id":648,
        "contenu":"Et pleins d'autres trucs."
      }
   ]
}
```

Listing 3: Exemple d'un document JSON.

Exemple d'une requête MongoDB

```
db.inventory.find(
    {
      type: 'food',
      $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } }
    }
)
```

Listing 4: Exemple de requête find sur MongoDB.

Bases de données orientées recherche

- Dérivées des bases de données orientées documents ;
- **Modélisation :** identique aux bases de données documents, choix des index en plus ;
- **Opérations :** peu d'options, uniquement de la recherche ;
- **Cas d'utilisation :** recherche full-text, recherche tolérante... ;
- Souvent utilisées en parallèle d'une autre base de données de stockage ;
- **Principaux acteurs :** ElasticSearch.

Concurrence forte du SQL sur les documents / recherche

Ces dernières années, les SGBD relationnels ont intégré et amélioré le type JSON pour relaxer les schémas.

- **Modélisation SQL :** utilisation des colonnes JSON ou BJSON, création d'index de recherche full-text ;
- **Opérations :** SQL très connu et maîtrisé ;
- **Cas d'utilisation :** stockage de données normalisées **ET** de données *souples*, recherche full-text, recherche tolérante... ;
- **Principaux acteurs :** PostgreSQL, MySQL...

Bases de données orientées graphes

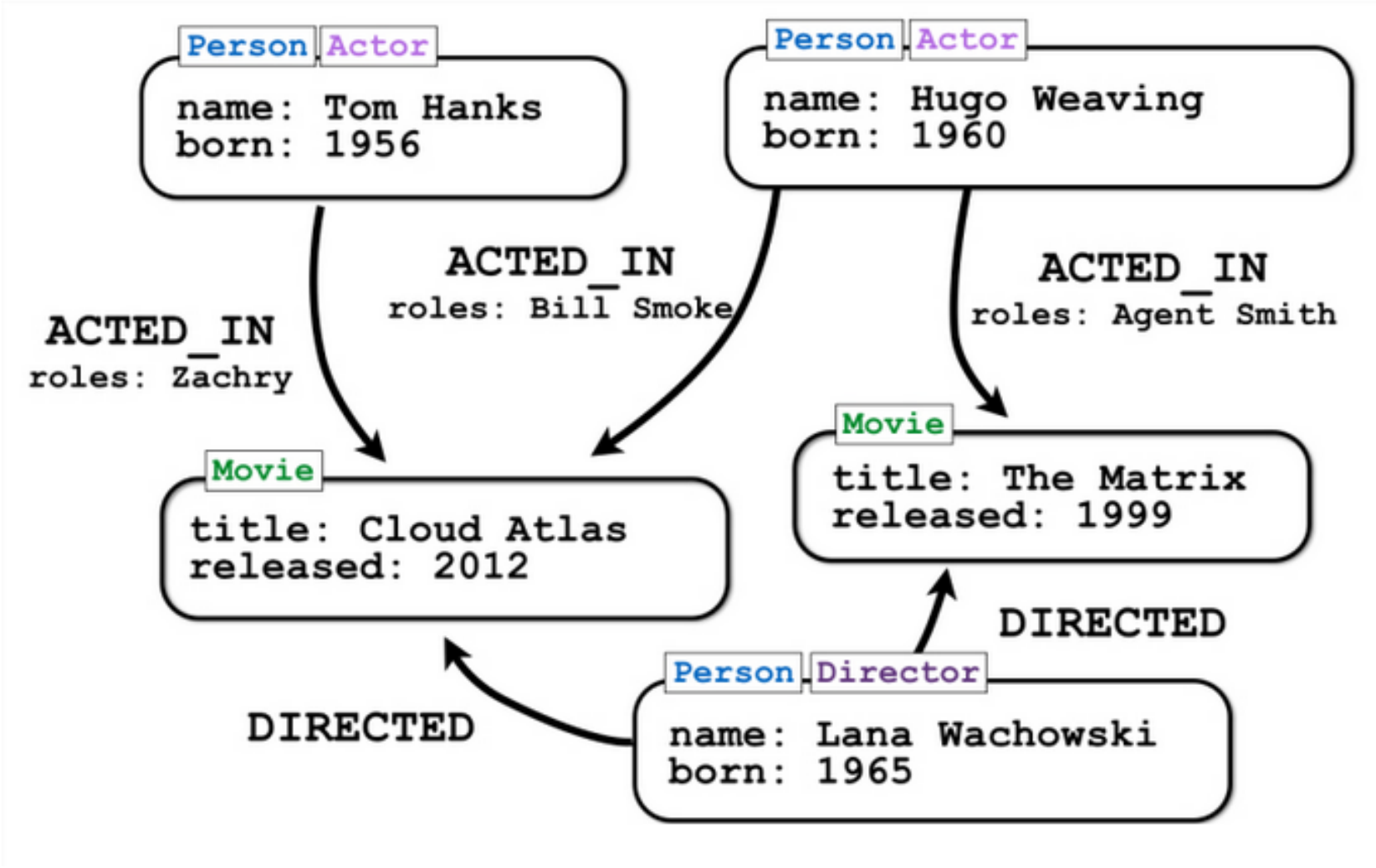

Figure: Exemple de graphe.

Bases de données orientées graphes

- **Modélisation :** représentation de l'information de manière très particulière (nœuds et arcs de même importance) ;
- **Opérations :** traitement très particulier de l'information (voir slide précédente) ;
- **Cas d'utilisation :** utilisé principalement pour les réseaux (liens entre des équipements ou des personnes) ;
- Base de données rarement utilisée pour du stockage ;
- **Principaux acteurs :** Neo4j, Titan, OrientDB.

Bases de données orientées graphes

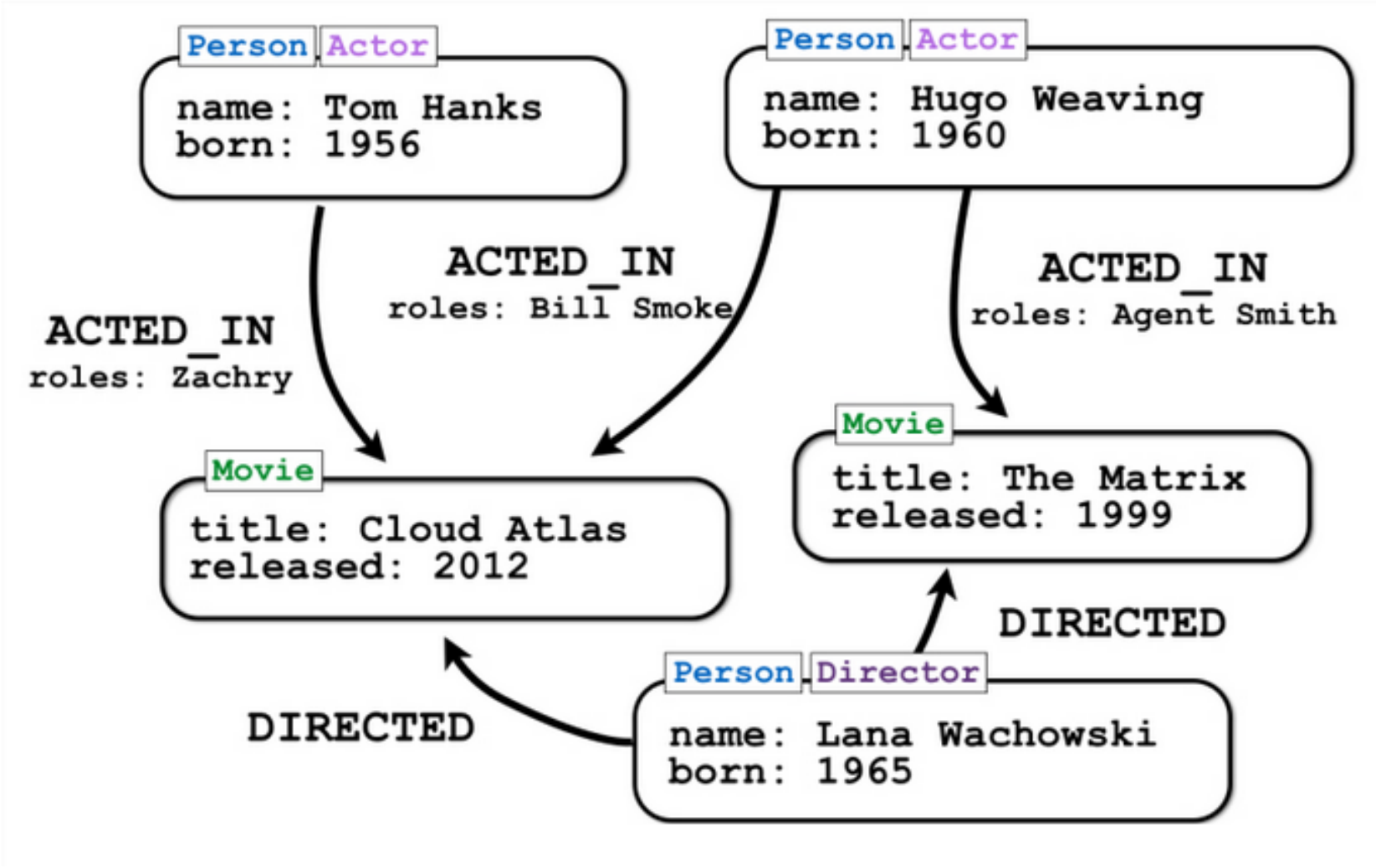

Figure: Exemple de graphe.

Exemple de requête

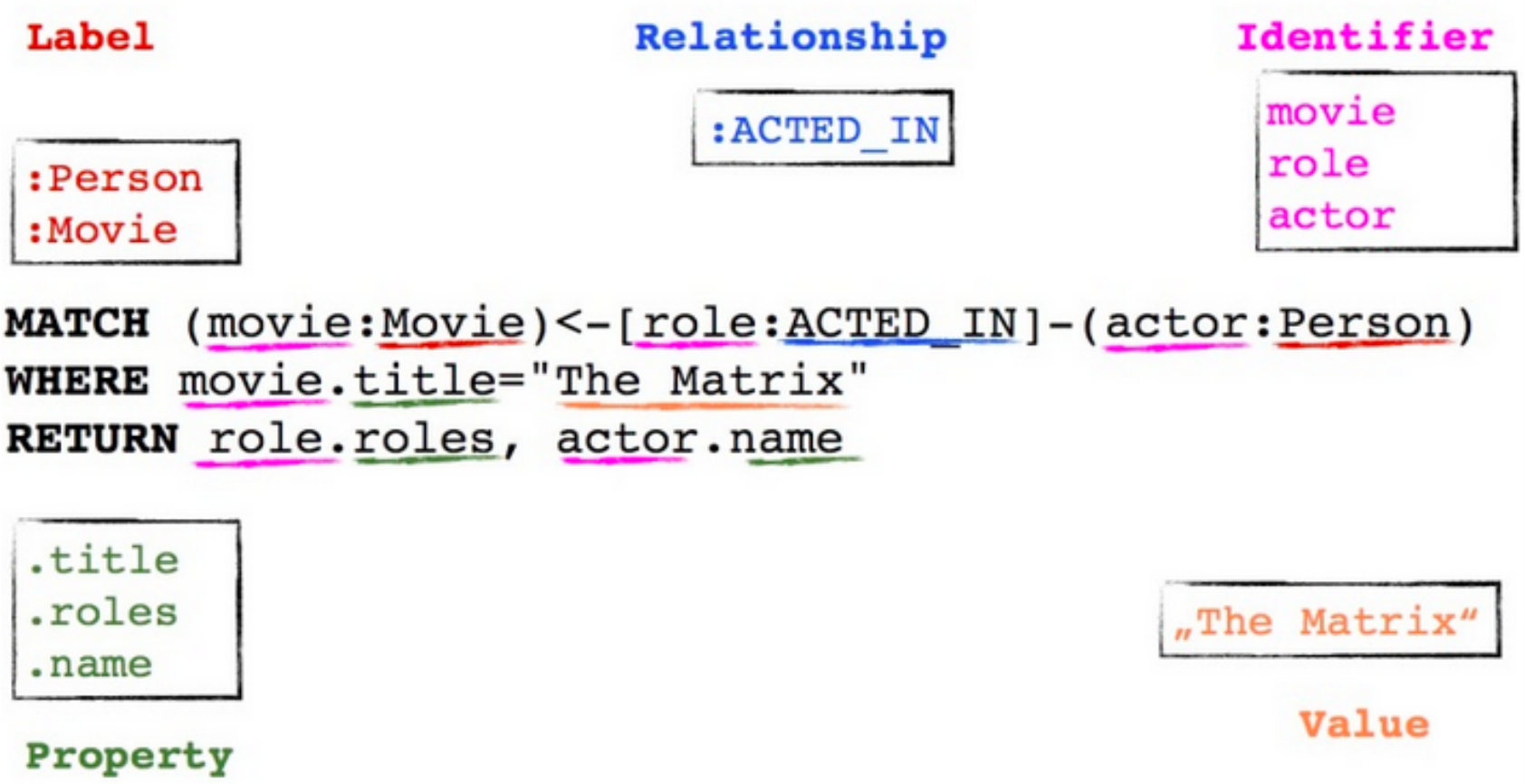

Figure: Exemple de requête Cypher pour Neo4j.

Les bonnes pratiques du NoSQL

- Ne pas commencer par du NoSQL ;
- Bien connaître ses données ;
- Ne pas avoir peur de la redondance ;
- Ne pas trop redonder ;
- Et surtout, bien quantifier ses besoins.

RTFM

La majorité des bases de données NoSQL connues sur le marché possèdent une très bonne documentation, souvent faite à destination de personnes venant du monde du relationnel.

D'autres cas d'usage du NoSQL

Première ligne en prod', mais pas que

Le NoSQL ne permet pas uniquement de contenir la charge, de répondre aux problématiques du distribué.

D'autres attraits du NoSQL :

- **Dénormalisation :** business intelligence, machine learning, data mining (BDD colonnes, documents)...
- **Stockage :** entrepôt d'agrégation de données (BDD colonnes, documents)
- **Modélisation :** systèmes de recommandation (BDD graphe)
- ...

NoSQL ou relationnel ?

Les deux mon capitaine !

Les bases de données NoSQL ont été inventées afin de résoudre des problèmes insolubles par les bases de données relationnelle et non **pas pour les remplacer**.

NoSQL	**Relationnel**
stockage de masse	stockage fiable
données diverses	données formatées
scalabilité horizontale	scalabilité verticale

Choix du type de BD NoSQL

	Modélisation	Cas d'utilisation
Clé / valeur	Modélisation simple, permettant d'indexer des informations diverses via une clé	Mise en cache
Documents	Modélisation souple permettant de stocker des documents au format JSON dans des collections	Stockage de masse
Graphes	Modélisation optimisée pour les problèmes de graphes	Stockage provisoire pour traiter les données

Non exhaustif : il existe d'autres types de BDD NoSQL !