Neo4J

Système de Gestion de Bases de Données Orienté Graphe (SGBD-OG)

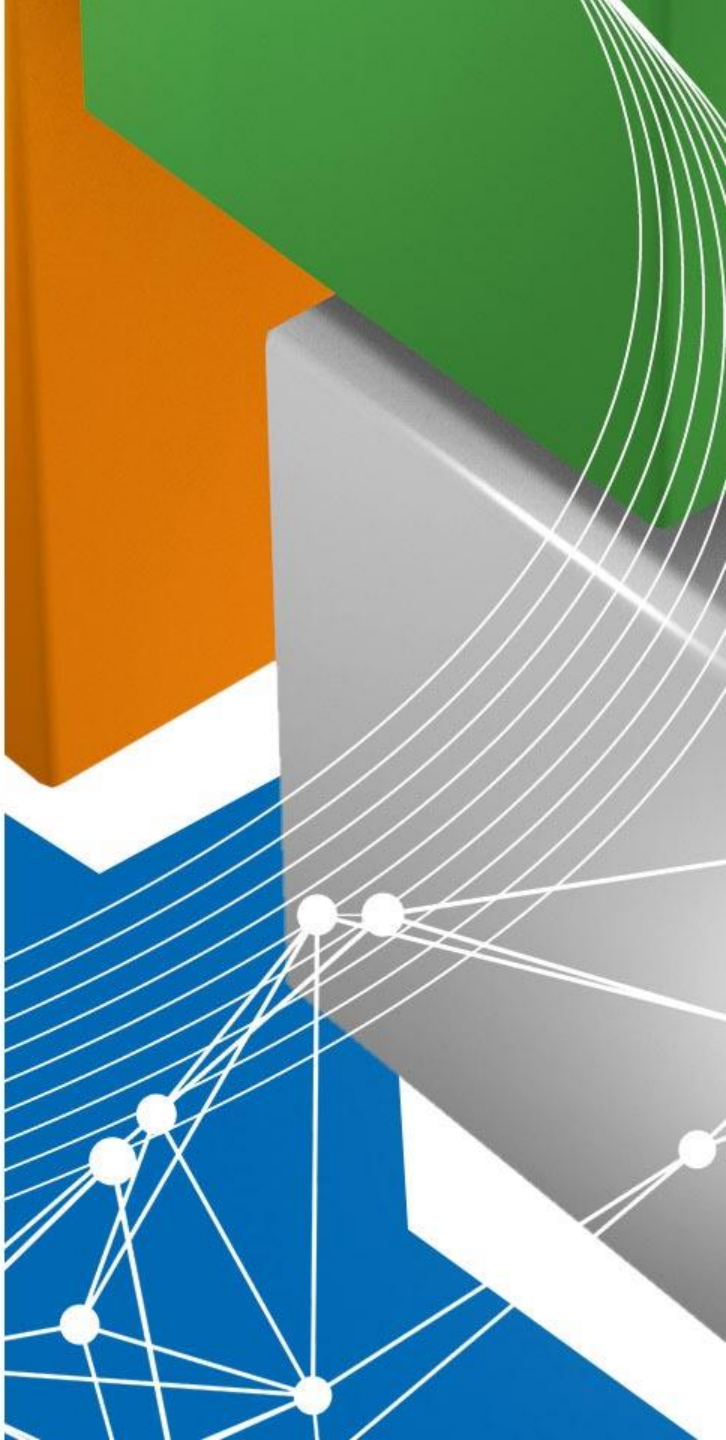
Samuel PARFOURU
Présentation de mars 2016

Objectif de la présentation

présentation pour des initiés au SGDB-R
Présentation des SGBD-OR illustrée via la technologie Neo4J

Plan

1. SGBD (Système de Gestion de Bases de)
2. SGBD-**OG** (**O**rienté **G**raphe)
3. Neo4J : solution industrielle leader du marché
4. Du "monde relationnel" au Graphe
5. Cypher : Langage de requête
6. Graphe et « problématiques PLM »
7. Exemples de mise en œuvre
8. Discussion



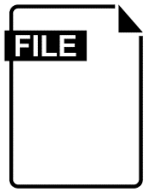
1

SGBD
(Système de Gestion de Bases de)

Historique

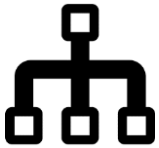
1960s

Traditional files



1970s

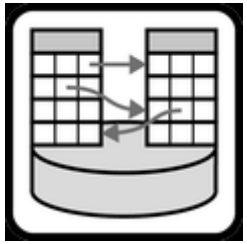
Hierarchical



Network

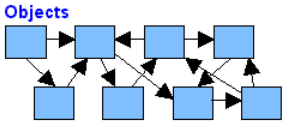
1980s

Relational

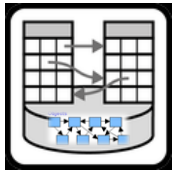


1990s

Object-oriented



Object-relationnal



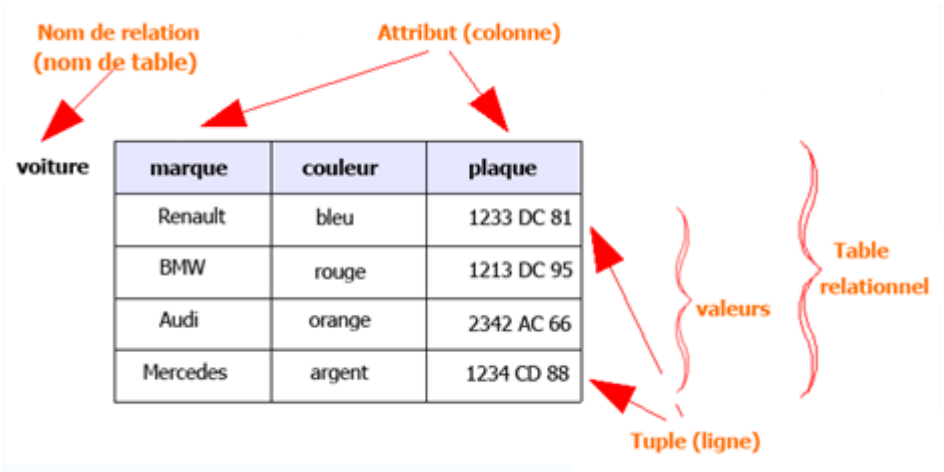
2000+

...

http://encyclopedia2.thefreedictionary.com/object-oriented+database+management+system

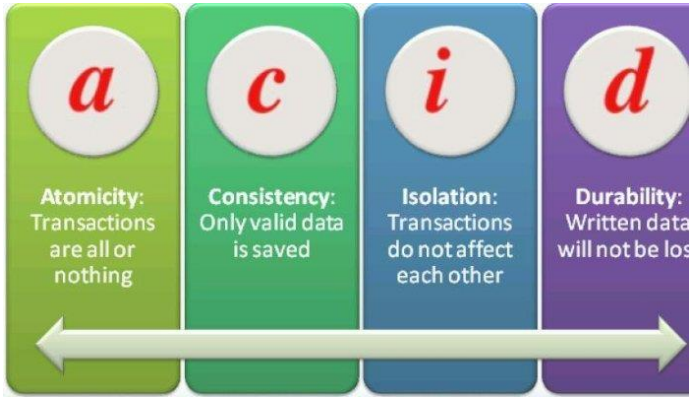
SGBD-R : Système de Gestion de Base de Données Relationnel

- Fondé sur le modèle de données relationnel



https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es

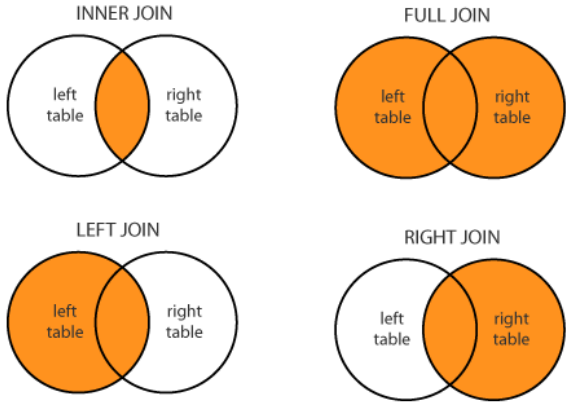
- Intégrité transactionnelle



- S'appuie sur le langage SQL* qui offre en particulier l'opération de ***Jointure***

« la jointure est l'opération permettant d'associer plusieurs tables ou vues de la base par le biais d'un lien logique de données »

« Le résultat de l'opération est une nouvelle table »

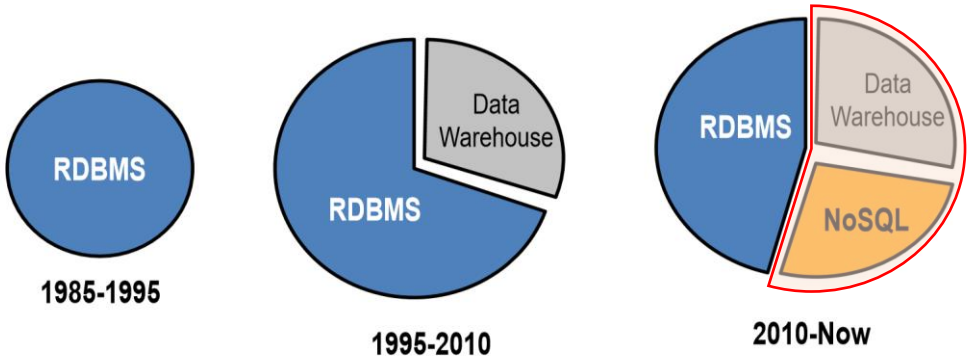


*Structured Query Language

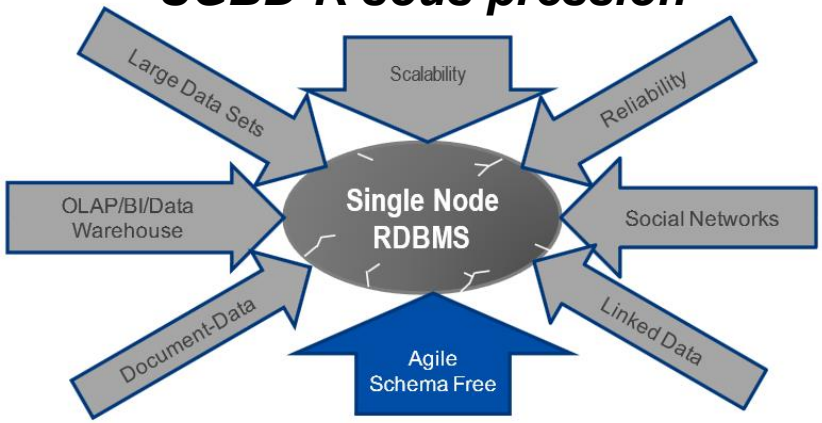
http://www.dofactory.com/sql/join

http://www.midwestacc.org/files/macc/files/Macc15/Slides/DanMcCreary.pdf

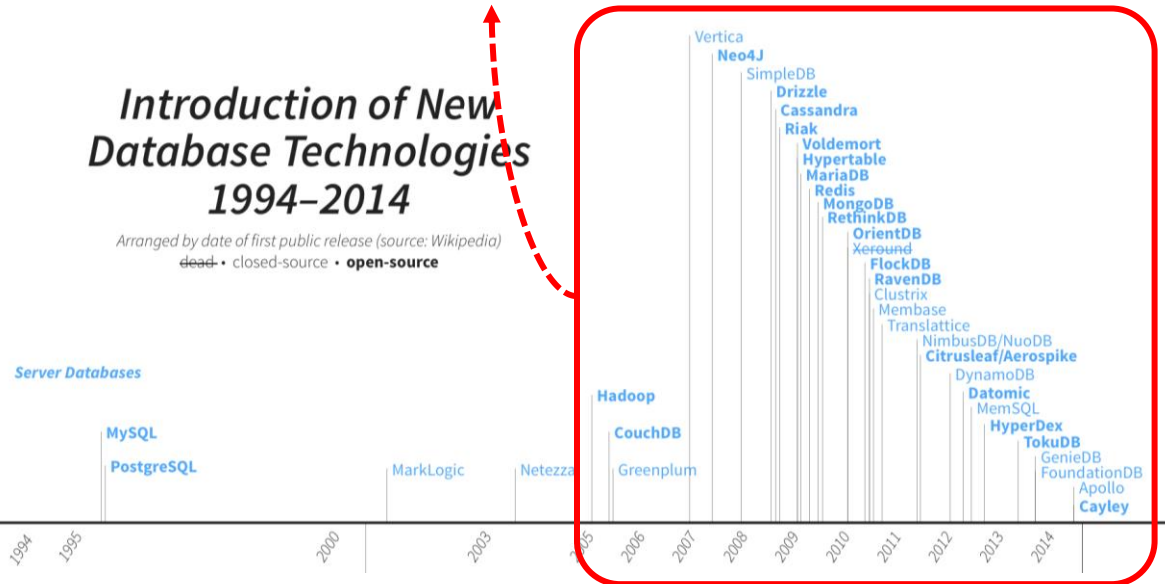
3 grandes ères



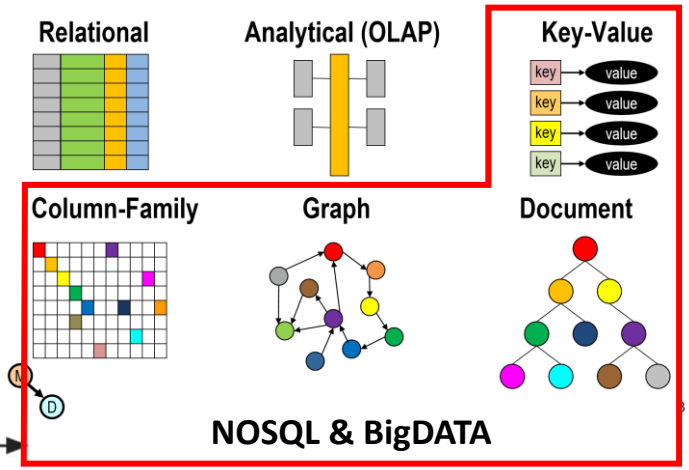
SGBD-R sous pression



RDMS for transations, Data Warehouse for analytics and **NoSQL for …** ?



Panaroma actuel



Popularité des SGBD

popularity -as in relevance in social networks or presence in professional profiles, frequency of technical discussions or even number of job offers…-

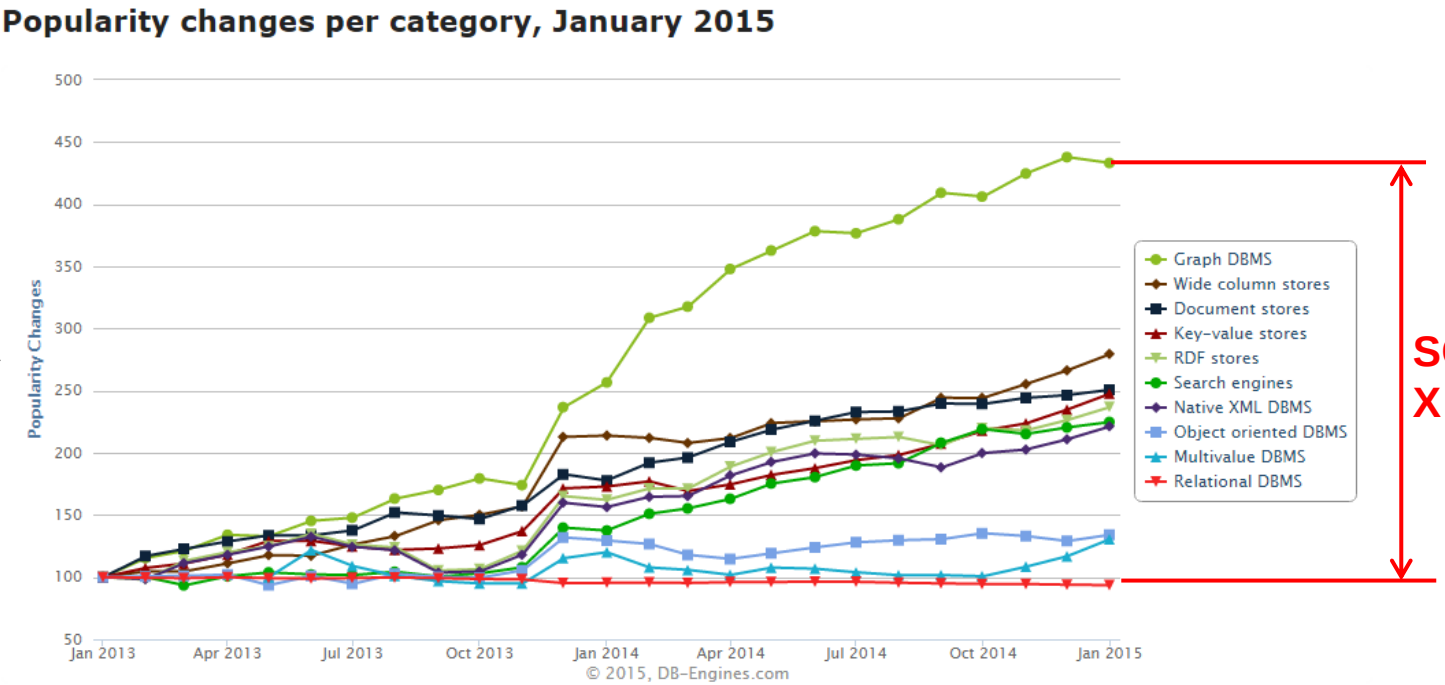


Fig. 1: Since the end of 2013, graph databases have dominated all other databases in DB-Engines.com's popularity ranking.

Gartner "Graph analysis is possibly the **single most effective competitive differentiator** for organizations pursuing data-driven operations and decisions after the design of data capture."

EMA "Neo4j is the current market **leader in graph databases**."

FORRESTER "Forrester estimates that ***over 25% of enterprises*** will be using graph databases by 2017"

http://db-engines.com/en/blog_post/43

http://www.datanami.com/2015/01/15/graph-databases-everywhere-2020-says-neo4j-chief/

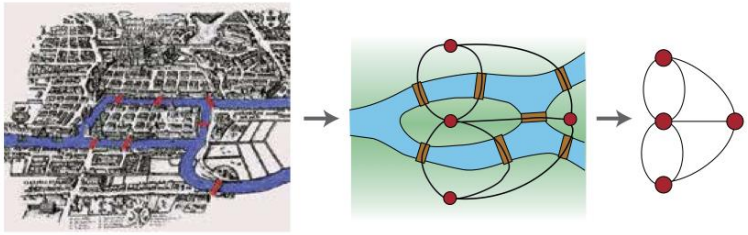

eDF
2
SGBD-OG

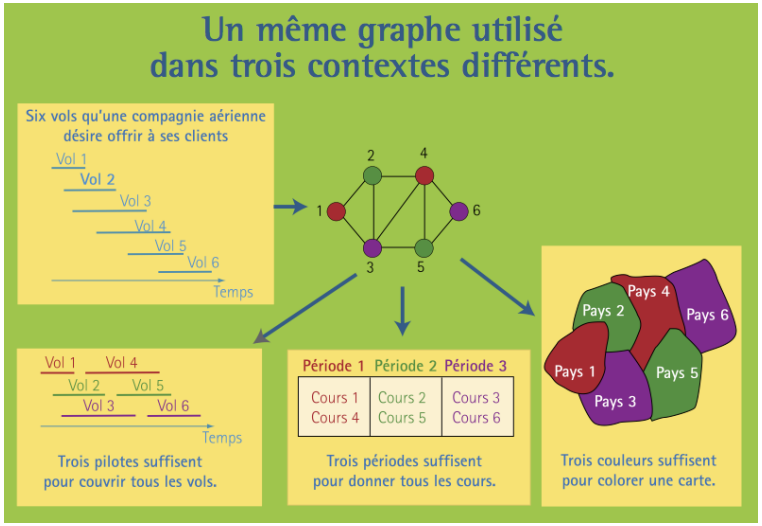
Théorie des graphes

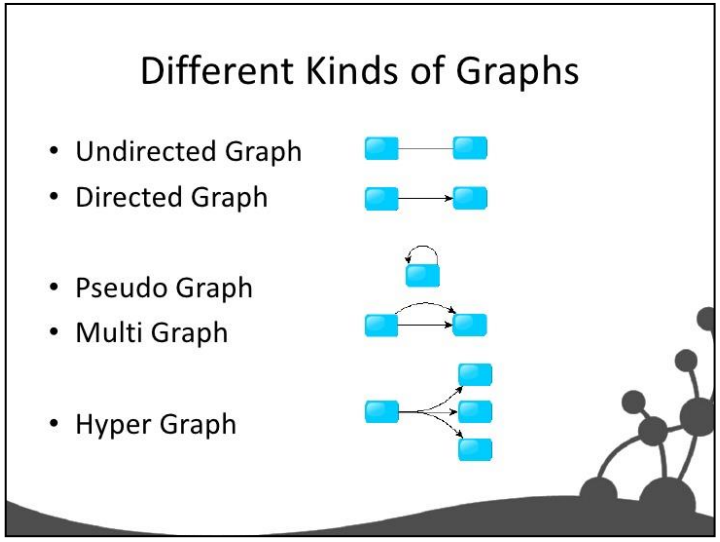
On accorde généralement à **Leonard Euler l'origine de la théorie des graphes**. Ce mathématicien et physicien suisse a été **le premier en 1736 à résoudre un problème (celui des ponts de Königsberg) en démontrant une propriété des graphes**.

Aujourd'hui, l'utilisation des graphes est devenue monnaie courante, car ceux-ci offrent une représentation imagée et colorée de situations très variées auxquelles nous faisons face au quotidien.

Les ponts de Königsberg





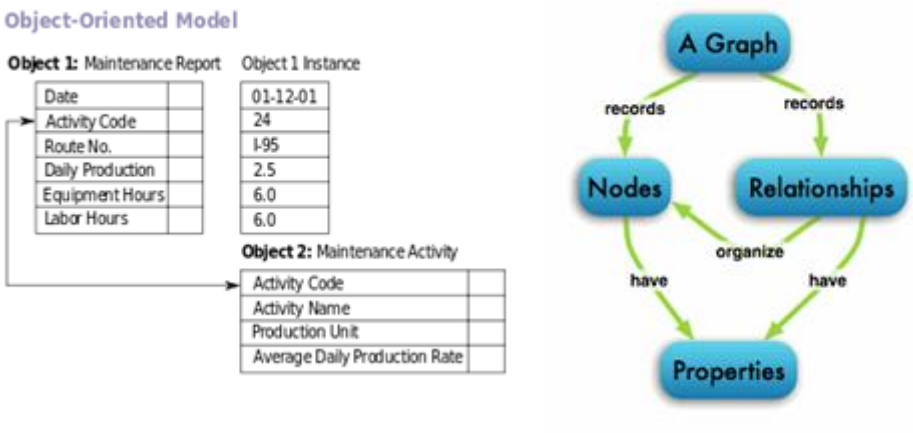


http://accromath.uqam.ca/2016/02/dessine-moi-un-graphe/

http://fr.slideshare.net/maxdemarzi/introduction-to-graph-databases-12735789

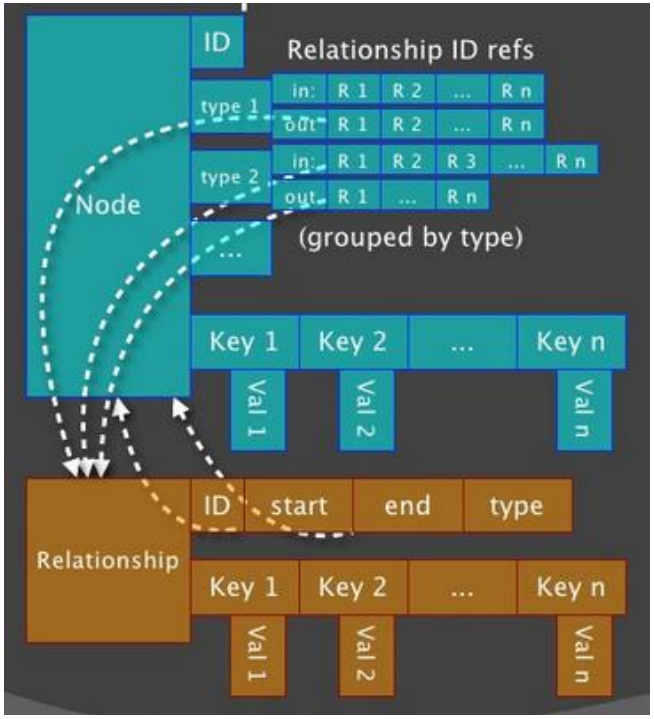
SGBD-OG

- *« Une base de données orientée graphe est une **base de données orientée objet utilisant la théorie des graphes**, donc avec des nœuds et des arcs, permettant de représenter et stocker les données. »*



- *« C'est une base de données orientée objet adaptée à **l'exploitation des structures de données de type graphe ou dérivée, comme des arbres**. »*

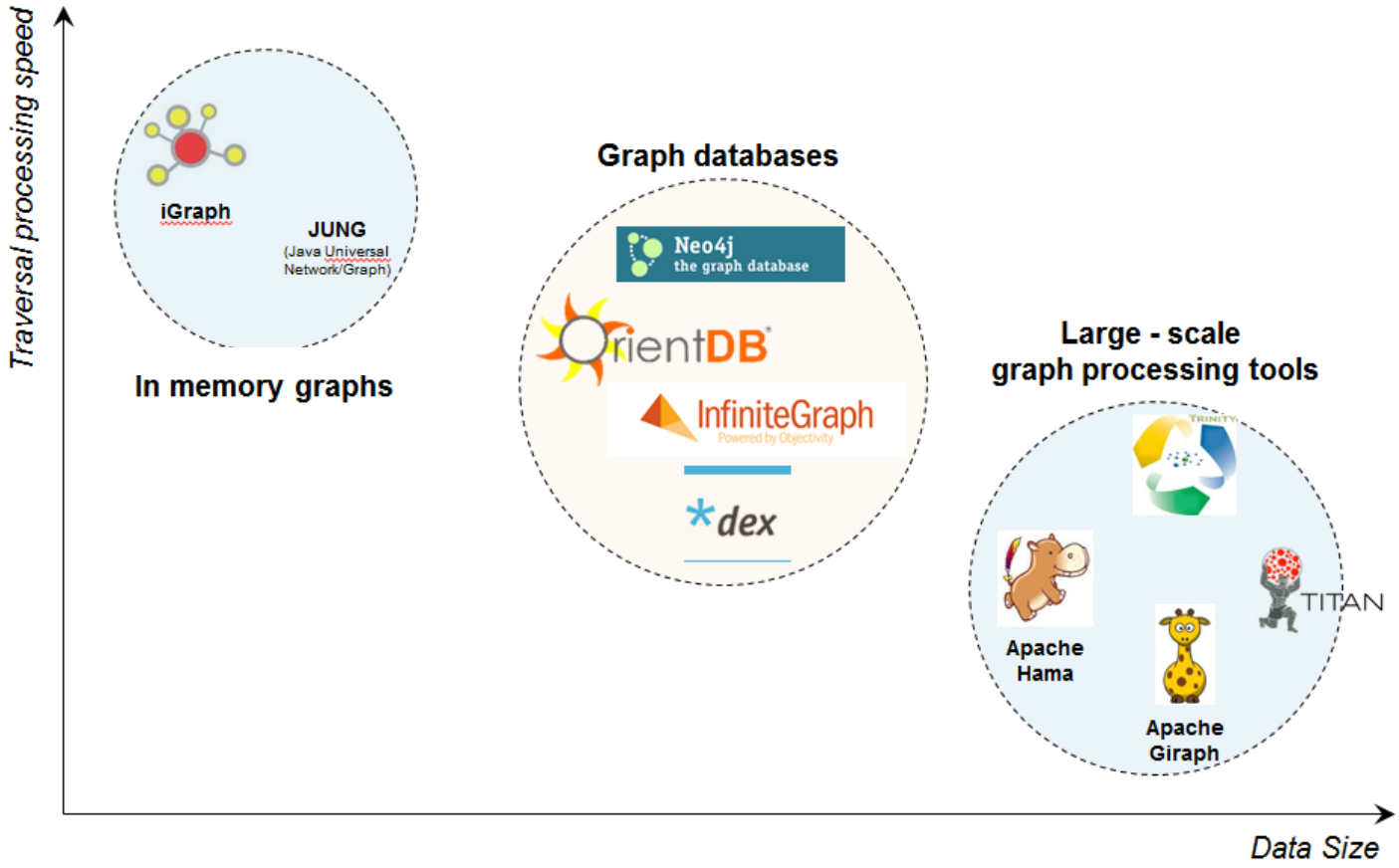
- *« Par définition, une base de données orientée graphe correspond à un **système de stockage capable de fournir une adjacence entre éléments voisins** : **chaque voisin d'une entité est accessible grâce à un pointeur physique**. »*



http://fr.slideshare.net/thobe/an-overview-of-neo4j-internals

Panorama des SGBD-OG

Classification of graph processing tools (source: Marko Rodriguez)



http://blog.octo.com/en/graph-databases-an-overview/

Besoin d'un SGBD-OG ?

- Voici une liste de questions à se poser avant de partir sur une base de données orientée graphe :
  - Vos données sont-elles **dynamiques** ?
  - Vos données sont-elles **connectées** ?
  - Avez-vous besoin d'un **schéma flexible** ?
  - Devez-vous faire du **temps réel** ?

Si vous répondez **OUI à au moins 2 de ces questions** le SGBD-OR peut être considéré

http://logisima.developpez.com/tutoriel/nosql/neo4j/introduction-neo4j/

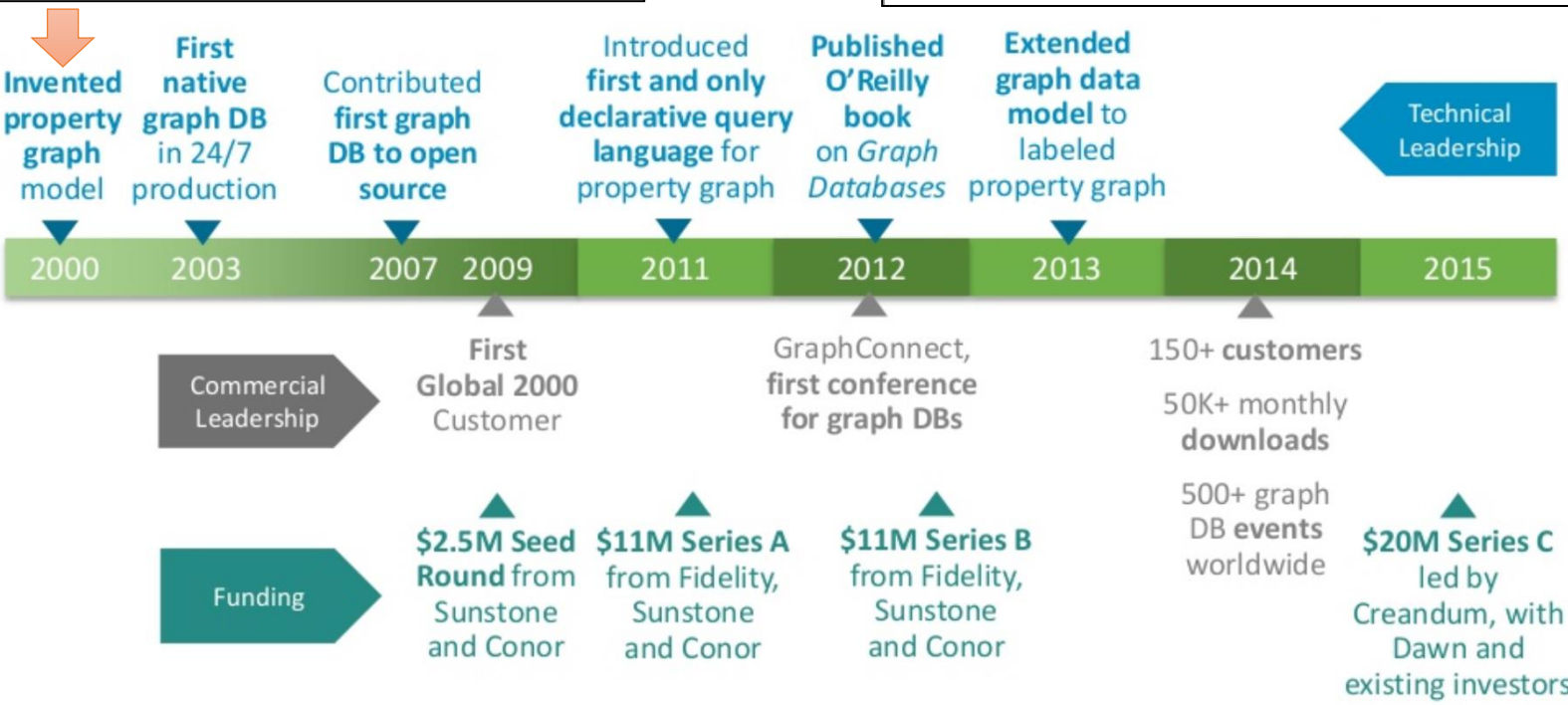

eDF
3
Neo4J
solution industrielle
leader du marché

History of Neo4j - Problem

- Digital Asset Management System in 2000
- SaaS many users in many countries
- Two hard use-cases
  - Multi language keyword search
    - Including synonyms / word hierarchies
  - Access Management to Assets for SaaS Scale

History of Neo4j – Relational Attempt

- Tried with many relational DBs
  - JOIN Performance Problems
    - Hierarchies, Networks, Graphs
  - Modeling Problems
  - Data Model evolution
- No Success, even …
  - With **expensive database consultants**!

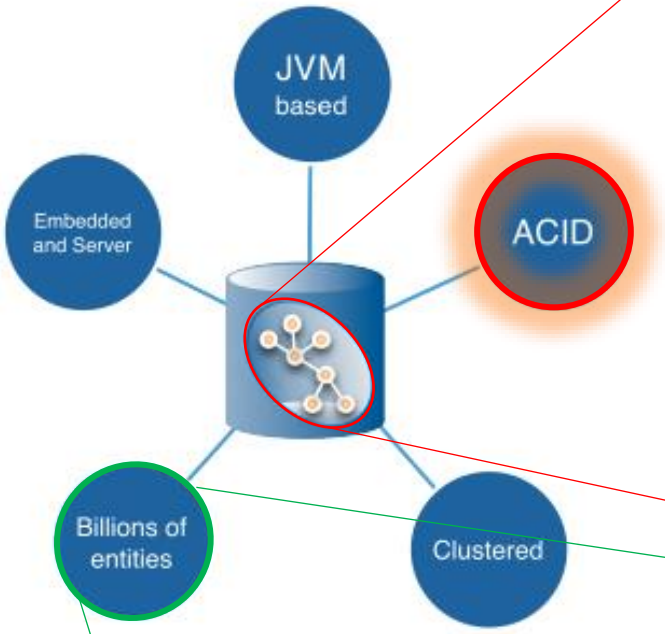


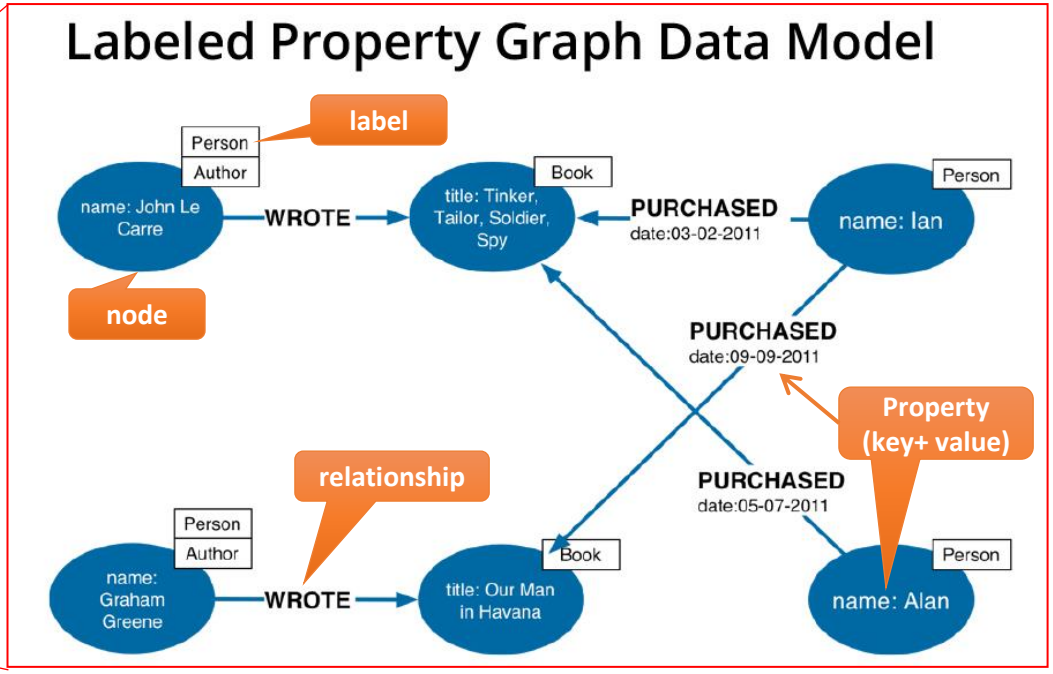
http://fr.slideshare.net/neo4j/relational-to-big-graph

http://fr.slideshare.net/neo4j/graphtalk-berlin-einfhrung-in-graphdatenbanken

Technologie

https://dzone.com/refcardz/querying-graphs-neo4j





Data size

In Neo4j, data size is mainly limited by the address space of the primary keys for Nodes, Relationships, Properties and RelationshipTypes. Currently, the address space is as follows:

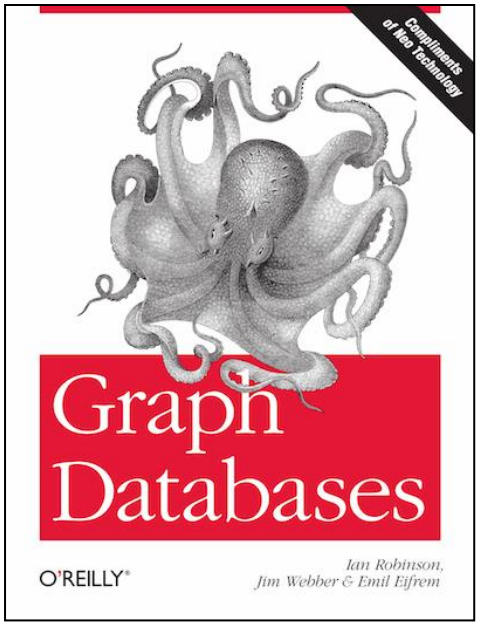
nodes	2^{35} (~ 34 billion)
relationships	2^{35} (~ 34 billion)
properties	2^{36} to 2^{38} depending on property types (maximum ~ 274 billion, always at least ~ 68 billion)
relationship types	2^{16} (~ 65 000)

http://neo4j.com/docs/stable/capabilities-capacity.html

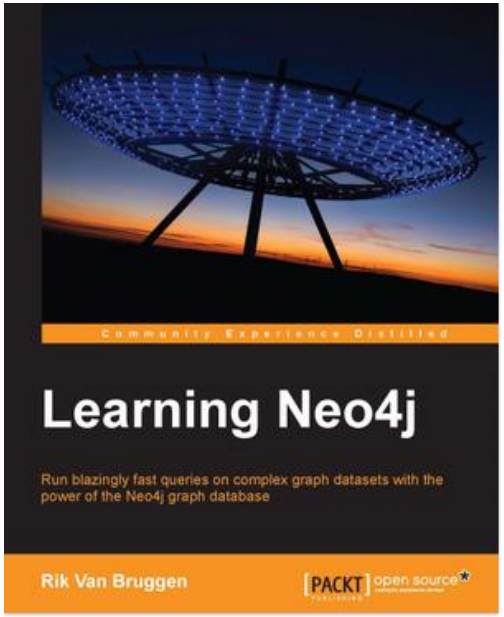
limitations qui devraient disparaître dans la prochaine version (à confirmer conférence GraphConnect 2016)

Bibliographie

Livres gratuits en téléchargement (pdf)

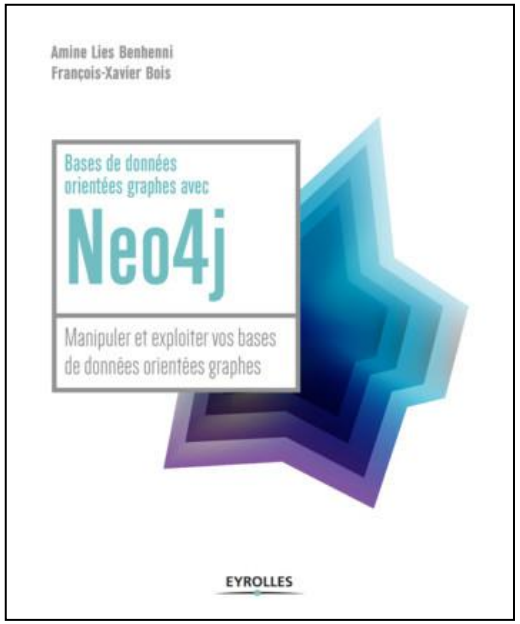


http://www.neo4j.org/learn



http://neo4j.com/book-learning-neo4j/

Bases de données orientées graphes avec Neo4j
Manipuler et exploiter vos bases de données orientées graphes



François-Xavier Bois , Amine Lies Benhenni
Collection Blanche
18 février 2016

La société NeoTechnology

- **Solidité**
  - **45 M$ de fonds levés** par Fidelity, Sunstone, Conor, Creandum, Dawn Capital
  - **Résultat doublés, quasi triplés tous les ans**, équilibre en ligne de mire, modèle économique "bankable" basé sur la souscription.
  - 120 Employés dont une équipe en France
  - **Leader des bases de données de graphes**
  - **Plus de 1 millions de téléchargements** (50000 / mois)
  - Produit date de 2000, société crée pour le promouvoir en 2008
- Clients
  - 150 Clients parmi les plus grand et pour des projets coeur de métier (MDM chez **Cisco**, Routage de colis pour la poste d'un grand pays d'Europe, Analyse d'impact réseau chez **SFR**, projet **PLM chez Airbus**...)

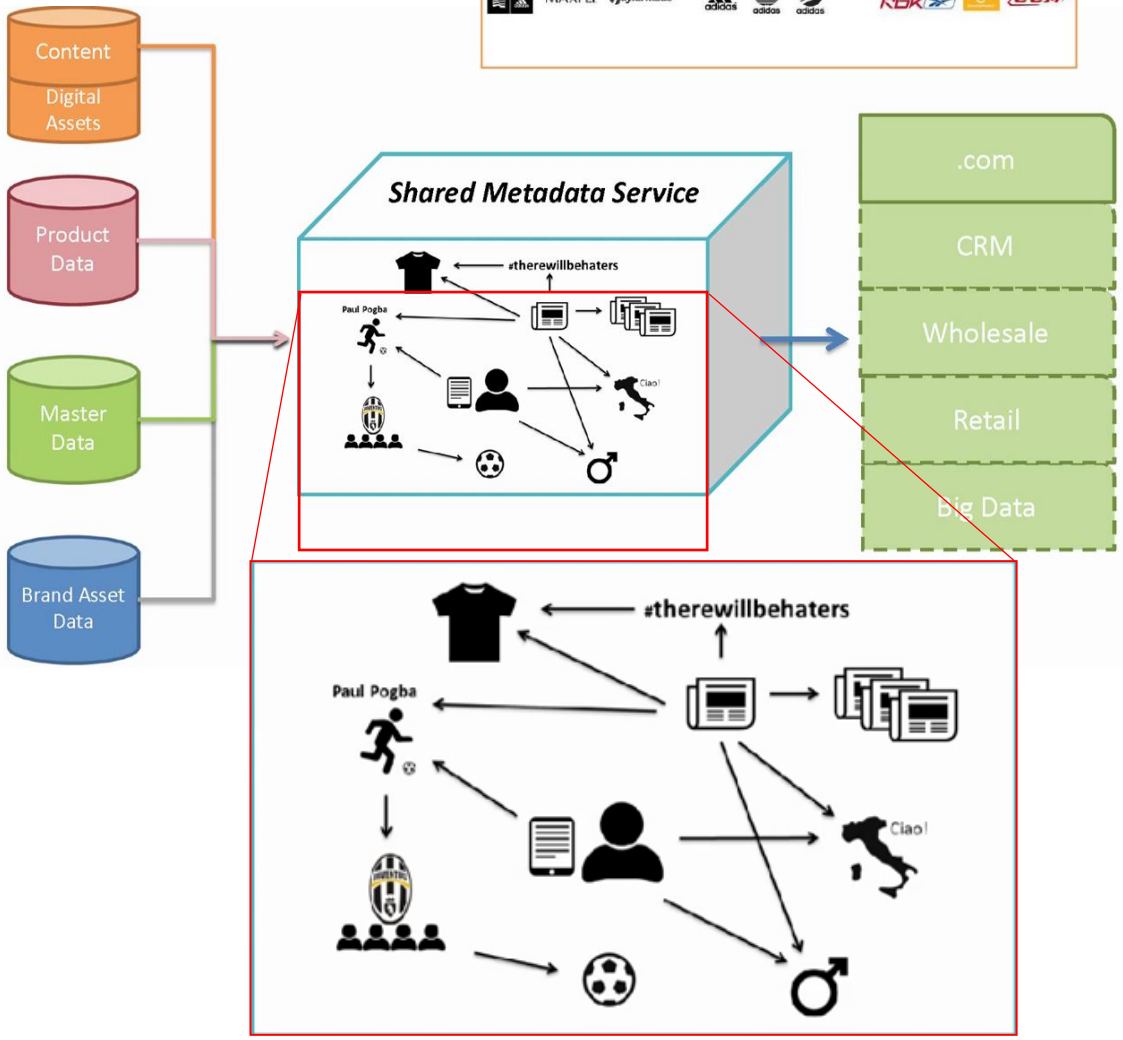
Exemples de mise en oeuvre

Master Data Management (MDM)

*"We have **many different silos, many different data domains**, and in order to make sense out of our data, **we needed to bring those together and make them useful for us,**" said Senior Project Manager Sokratis Kartelias.*

*"On the technical level, **data models didn't align between the information silos, and there wasn't a standard, consistent way to communicate between the different data domains.***
***Without a way to effectively consolidate such data, the Group felt it was missing out on opportunities to provide the most compelling and relevant content to its consumers,** as well as offering enhanced product recommendations "*

http://neo4j.com/case-studies/adidas/



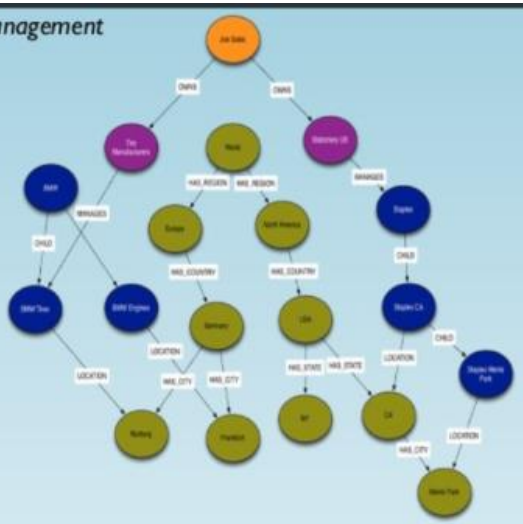
Master Data Management (MDM)

Industry: Communications Use case: Master Data Management

San Jose, CA

CISCO Cisco HMP

- One of the world's largest communications equipment manufacturers#91 Global 2000. $44B in annual sales.
- Needed a system that could accommodate its master data hierarchies in a performant way
- HMP is a Master Data Management system at whose heart is Neo4j. Data access services available 24x7 to applications companywide



- Sales compensation system had become unable to meet Cisco's needs
- Existing Oracle RAC system had reached its limits:
  - Insufficient flexibility for handling complex organizational hierarchies and mappings
  - "Real-time" queries were taking > 1 minute!
- Business-critical "P1" system needs to be continually available, with zero downtime

- Cisco created a new system: the Hierarchy Management Platform (HMP)
- Allows Cisco to manage master data centrally, and centralize data access and business rules
- Neo4j provided "Minutes to Milliseconds" performance over Oracle RAC, serving master data in real time
- The graph database model provided exactly the flexibility needed to support Cisco's business rules
- HMP so successful that it has expanded to include product hierarchy

http://fr.slideshare.net/barcelonajug/introduction-to-graph-databases-and-neo4j-by-stefan-armbruster

Gestion actifs et Analyse d'impact

SFR

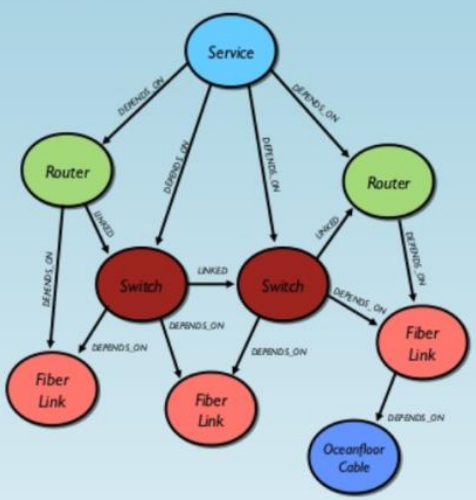
Industry: Communications

Use case: Network Management

Paris, France

SFR

- Second largest communications company in France
- Part of Vivendi Group, partnering with Vodafone

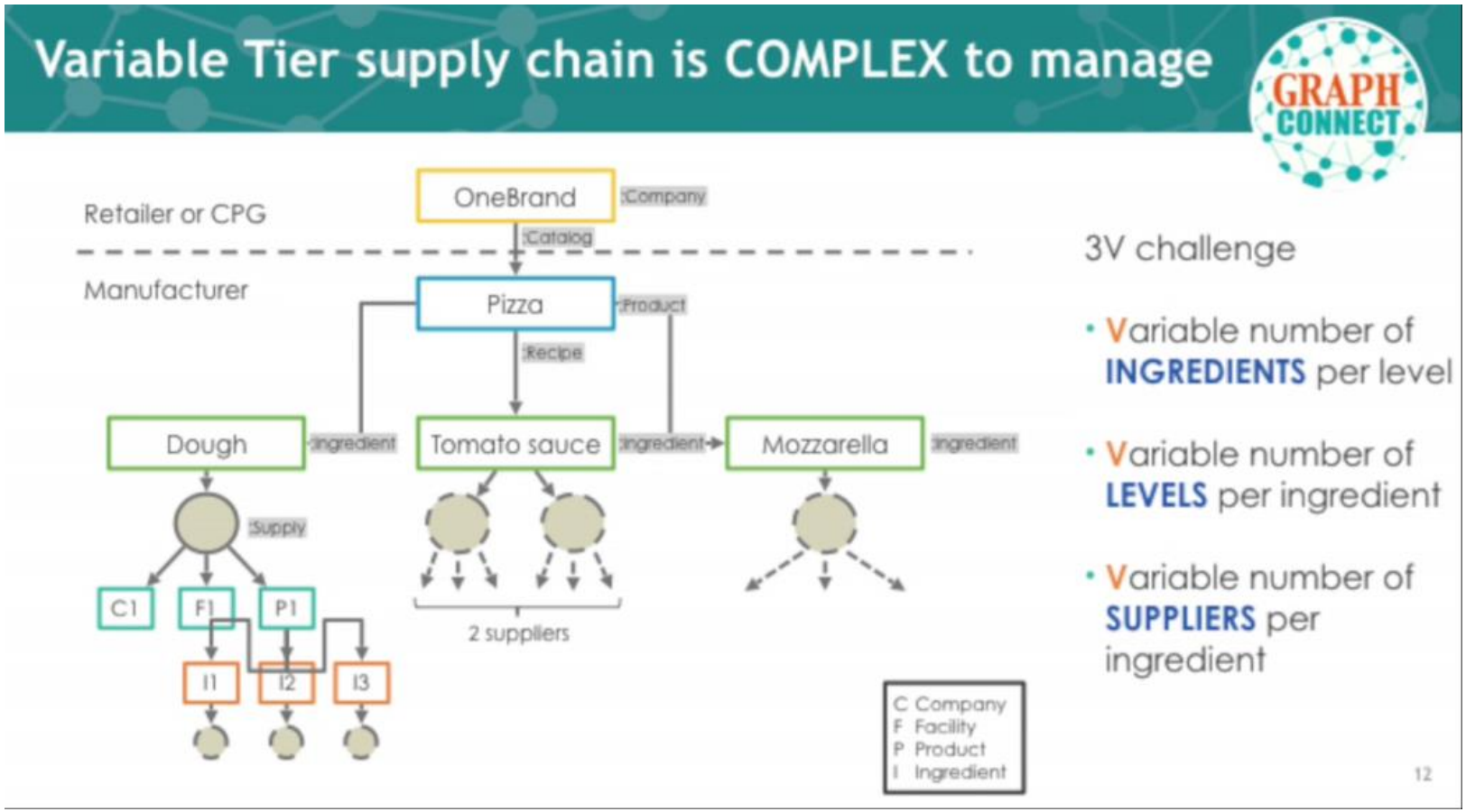


- Infrastructure maintenance took one full week to plan, because of the need to model network impacts
- Needed rapid, automated "what if" analysis to ensure resilience during unplanned network outagesIdentify weaknesses in the network to uncover the need for additional redundancy
- Network information spread across > 30 systems, with daily changes to network infrastructureBusiness needs sometimes changed very rapidly

- Flexible network inventory management system, to support modeling, aggregation & troubleshooting
- Single source of truth (Neo4j) representing the entire network
- Dynamic system loads data from 30+ systems, and allows new applications to access network data
- Modeling efforts greatly reduced because of the near 1:1 mapping between the real world and the graph
- Flexible schema highly adaptable to changing business requirements

http://fr.slideshare.net/barcelonajug/introduction-to-graph-databases-and-neo4j-by-stefan-armbruster

Modélisation de chaîne logistique ("supply chain") et Analyse d'impact



http://neo4j.com/blog/tracing-world-food-supply-farm-to-fork-neo4j/

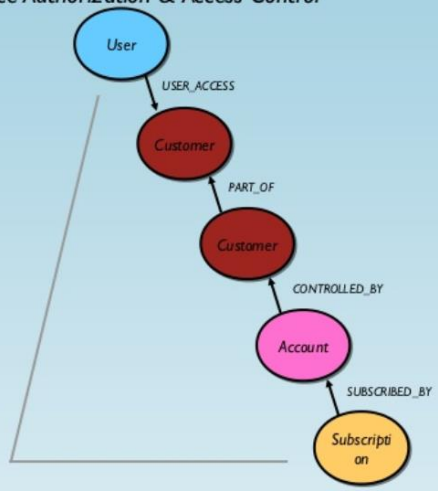
ACL : Access Control List

Industry: Communications

Use case: Resource Authorization & Access Control

telenor Oslo, Norway

Telenor



- 10th largest Telco provider in the world, leading in the Nordics
- Online self-serve system where large business admins manage employee subscriptions and plans
- Mission-critical system whose availability and responsiveness is critical to customer satisfaction

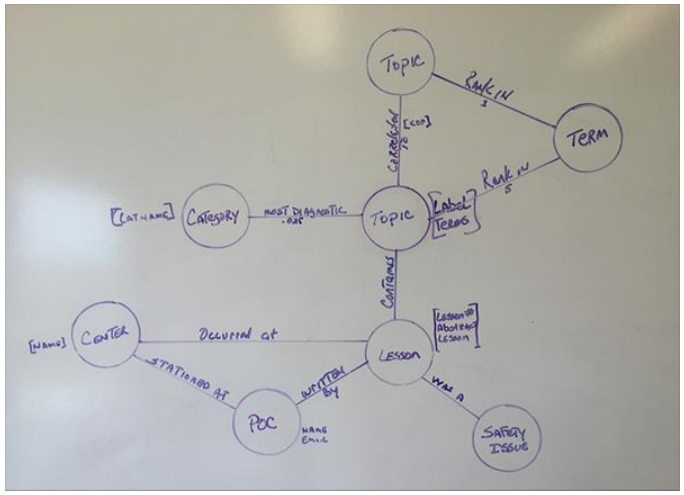
- Degrading relational performance. User login taking minutes while system retrieved access rights
- Millions of plans, customers, admins, groups. Highly interconnected data set w/massive joins
- Nightly batch workaround solved the performance problem, but meant data was no longer current
- Primary system was Sybase. Batch pre-compute workaround projected to reach 9 hours by 2014: longer than the nightly batch window

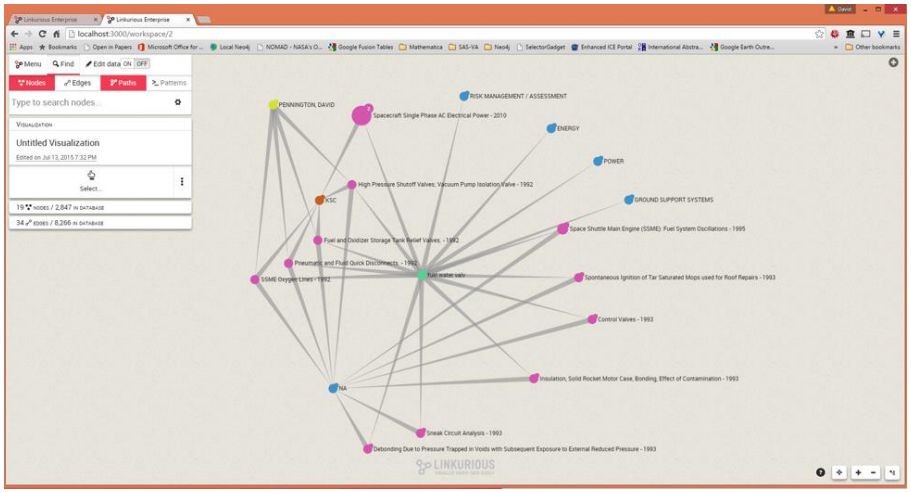
- Moved authorization functionality from Sybase to Neo4j
- Modeling the resource graph in Neo4j was straightforward, as the domain is inherently a graph
- Able to retire the batch process, and move to real-time responses: measured in milliseconds
- Users able to see fresh data, not yesterday's snapshotCustomer retention risks fully mitigated

http://fr.slideshare.net/barcelonajug/introduction-to-graph-databases-and-neo4j-by-stefan-armbruster

En vrac

- **Airbus**, problématique du manuel Avion
  - **Mars 2016** : choix de Neo4J sur la problématique du manuel avion (**contrat de 1M€ pour NeoTechnology** dans le cadre d'un projet de ~ 4M€ pour Sopra Steria)
- Collaboration en cours avec Intergraph sur l'intégration de Neo4J au sein du produit SmarPlant Foundation
- **Graphing a Lesson Learned Database for NASA Using Neo4j, R/RStudio & Linkurious**





http://neo4j.com/blog/nasa-lesson-learned-database-using-neo4j-linkurious/

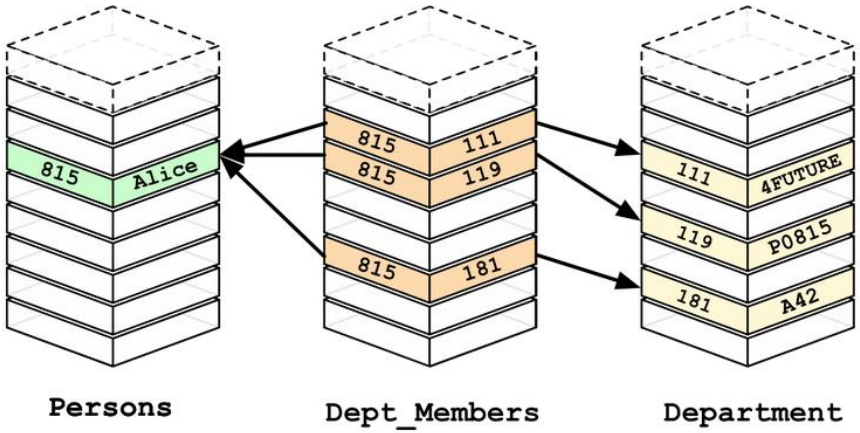
4

Du "monde relationnel" au Graphe

Cas simple

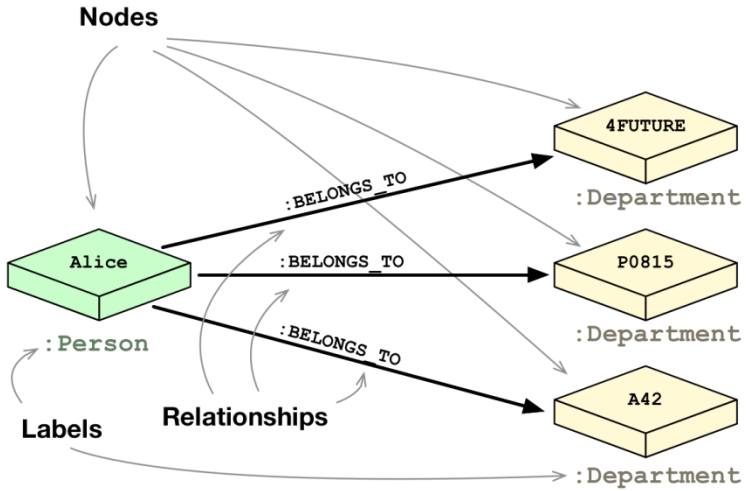
- Modéliser l'appartenance de personnes à des départements ?
  - *Hypothèse : une personne peut appartenir à plusieurs départements*

SGBD-R



• L'implémentation nécessite 3 tables relationnelles,
• Mise en œuvre d'une table de jonction (« JOIN TABLE »)
• personne ↔ département peut être résolu via une double jointure .

SGBD-OG



• Le graphe correspond ici à ce que l'on dessinerait sur un tableau blanc

"property graphs are "whiteboard friendly" as they are aligned with the semantics of modern object oriented languages and the diagramming techniques"

• *Le graphe correspond au modèle physique de données*

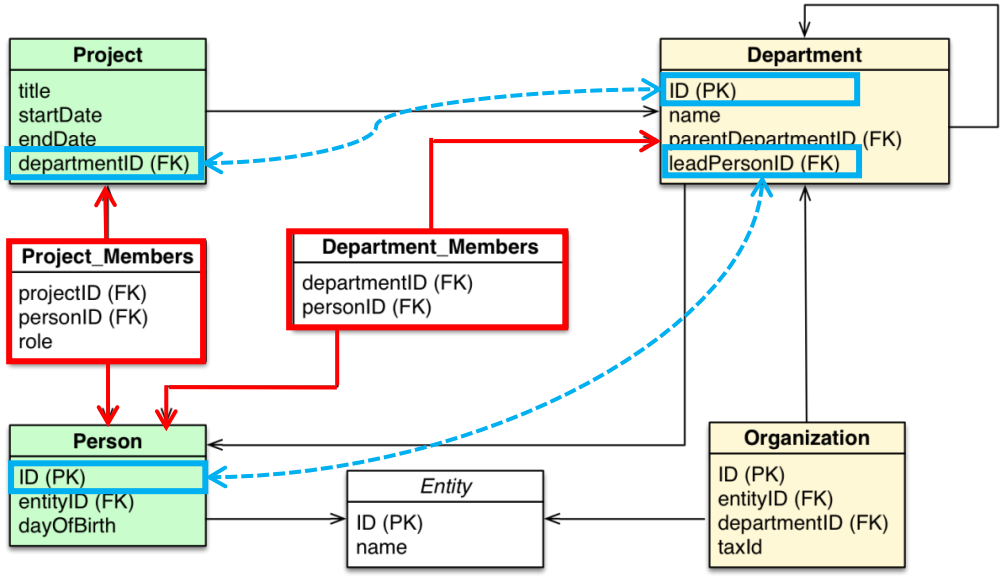
http://neo4j.com/developer/graph-db-vs-rdbms/

Modèle plus évolué

SGBD-R

La modélisation de relations entre entités se fait via :

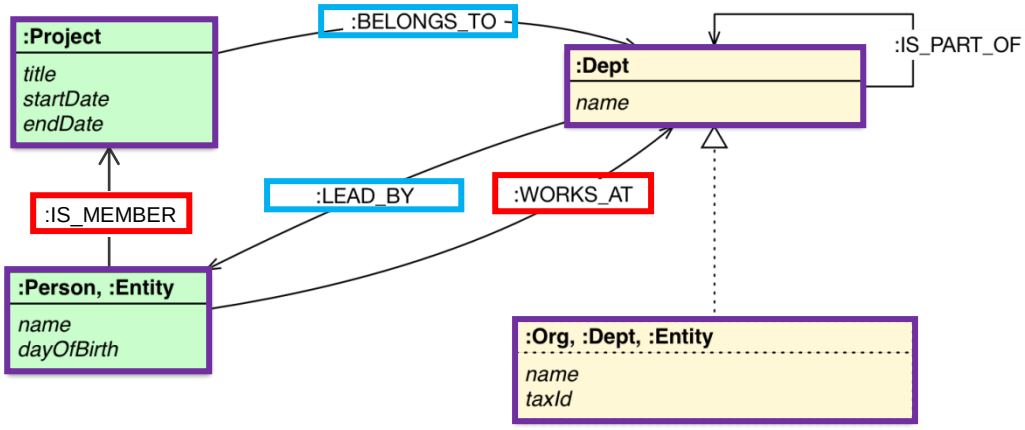
- la création de tables de jointure intégrant des couples de clés étrangères(✳→✳)
- un mécanisme logique de mise en œuvre de clé étrangère (1→✳)



SGBD-OR

La modélisation de relations entre entités se fait via :

- les tuples des tables deviennent des nœuds « taggés » avec un LABEL
- les tables de jointure sont remplacés au profit d'une relation explicite dans le graphe (✳→✳)
- les clés étrangères sont supprimées au profit d'une relation explitie (1→✳)



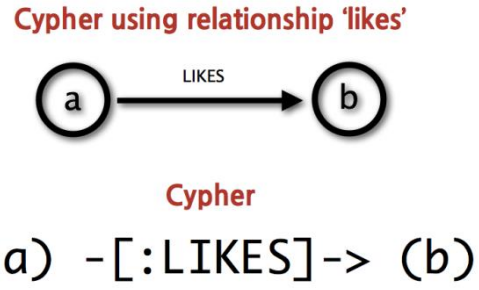
http://neo4j.com/developer/graph-db-vs-rdbms/

5

Cypher
langage de requête

Cypher : un langage de requête de type "pattern-matching"

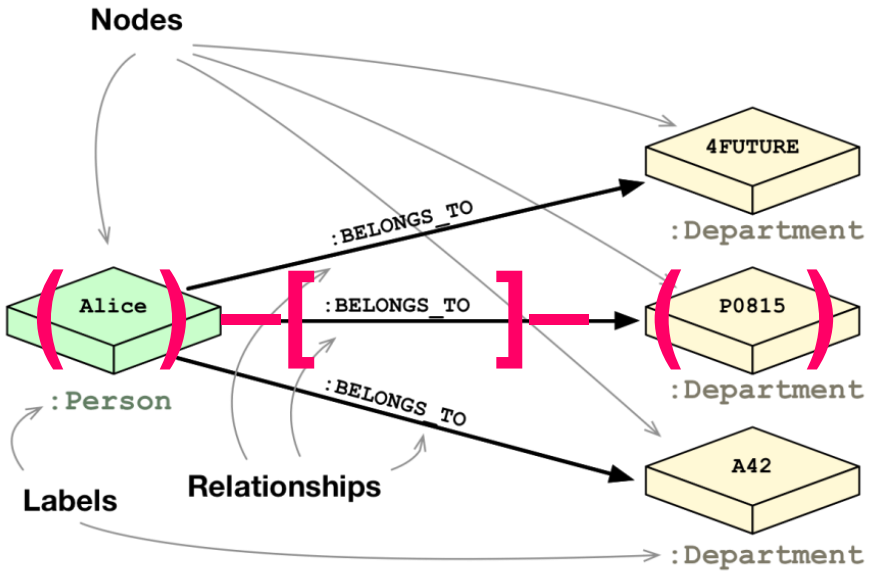
- ***Cypher* is a declarative graph query language** that allows for expressive and efficient querying and updating of the graph store.
- Cypher is a relatively simple but still very powerful language.
- **This allows you to focus on your domain instead of getting lost in database access.**
- **Cypher is designed to be a humane query language**, suitable for both developers and (importantly, we think) operations professionals.
- Our guiding goal is **to make the simple things easy, and the complex things possible**.
- **Based on English prose and neat iconography** which helps to make queries more self-explanatory.
- **Optimize the language for reading** and not for writing.



Illustration

"Being a declarative language, Cypher focuses on the clarity of expressing ***what*** **to retrieve from a graph, not on *how* to retrieve it."**

Lister les département auxquels appartient Alice ?



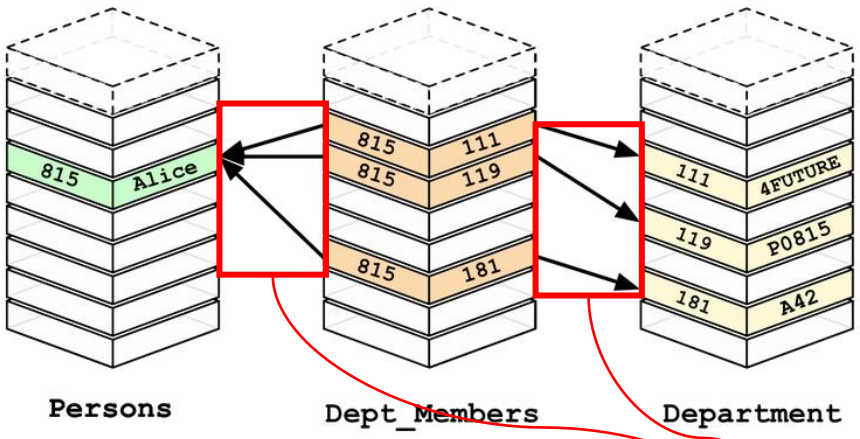
Pattern matching

```
MATCH (p:Person)-[:BELONG_TO]->(d:Department) WHERE p.name = 'Alice' RETURN d.name
MATCH (p:Person {name: 'Alice'})-[:BELONG_TO]->(d:Department) RETURN d.name
```

http://neo4j.com/developer/graph-db-vs-rdbms/

Comparaison SQL et Cypher

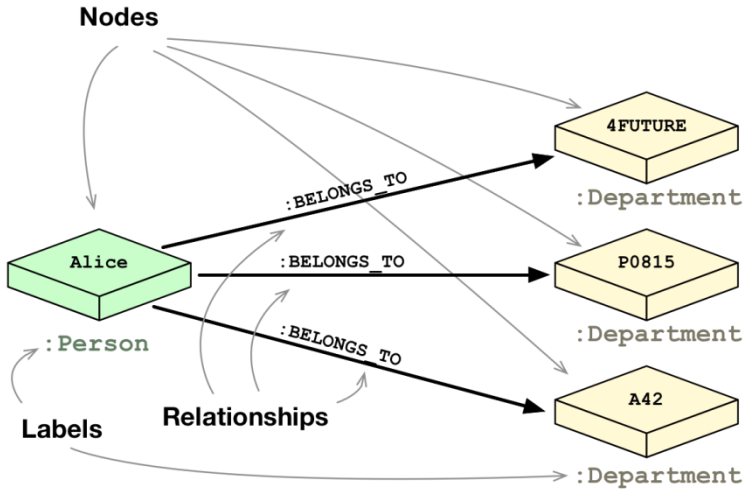
SGBD-R




```
SELECT * FROM Department
LEFT JOIN Dept_Members
ON Department.Id = Dept_Members.DepartmentId
LEFT JOIN Persons
ON Persons.Id = Dept_Members.PersonId
WHERE Person.name = 'Alice'
```

Double JOIN

SGBD-OG




```
MATCH (p:Person)-[:BELONG_TO]->(d:Department)
WHERE p.name = 'Alice'
RETURN d.name
```

http://neo4j.com/docs/stable/cypher-introduction.html

neo4j

Neo4j Cypher Refcard 2.3.3

Cypher is the declarative query language for Neo4j, the world's leading graph database.

Key principles and capabilities of Cypher are as follows:

- Cypher matches patterns of nodes and relationship in the graph, to extract information or modify the data.
- Cypher has the concept of identifiers which denote named, bound elements and parameters.
- Cypher can create, update, and remove nodes, relationships, labels, and properties.
- Cypher manages indexes and constraints.

You can try Cypher snippets live in the Neo4j Console at console.neo4j.org or read the full Cypher documentation in the Neo4j Manual. For live graph models using Cypher check out GraphGist.

The Cypher Refcard is also available in PDF format.

Note: `{value}` denotes either literals, for ad hoc Cypher queries; or parameters, which is the best practice for applications. Neo4j properties can be strings, numbers, booleans or arrays thereof. Cypher also supports maps and collections.

Syntax

Read Query Structure

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

MATCH

```
MATCH (n:Person)-[:KNOWS]->(m:Person)
WHERE n.name = 'Alice'
```
Node patterns can contain labels and properties.

```
MATCH (n)-->(m)
```
Any pattern can be used in `MATCH`.

```
MATCH (n {name: "Alice"})-->(m)
```
Patterns with node properties.

```
MATCH p = (n)-->(m)
```
Assign a path to `p`.

```
OPTIONAL MATCH (n)-[r]->(m)
```
Optional pattern, `NULL`s will be used for missing parts.

```
WHERE n.name = 'Alice'
```
Force the planner to use a label scan to solve the query (for manual performance tuning).

WHERE

```
WHERE n.property <> {value}
```
Use a predicate to filter. Note that `WHERE` is always part of a `MATCH`, `OPTIONAL MATCH`, `WITH` or `START` clause. Putting it after a different clause in a query will alter what it does.

Write-Only Query Structure

```
(CREATE [UNIQUE] | MERGE)*
[SET|DELETE|REMOVE|FOREACH]*
[RETURN [ORDER BY] [SKIP] [LIMIT]]
```

Read-Write Query Structure

```
[MATCH WHERE]
```

RETURN

```
RETURN *
```
Return the value of all identifiers.

```
RETURN n AS columnName
```
Use alias for result column name.

```
RETURN DISTINCT n
```
Return unique rows.

```
ORDER BY n.property
```
Sort the result.

```
ORDER BY n.property DESC
```
Sort the result in descending order.

```
SKIP {skipNumber}
```
Skip a number of results.

```
LIMIT {limitNumber}
```
Limit the number of results.

```
SKIP {skipNumber} LIMIT {limitNumber}
```
Skip results at the top and limit the number of results.

```
RETURN count(*)
```
The number of matching rows. See Aggregation for more.

WITH

```
MATCH (user)-[:FRIEND]-(friend)
WHERE user.name = {name}
WITH user, count(friend) AS friends
WHERE friends > 10
RETURN user
```
The `WITH` syntax is similar to `RETURN`. It separates query parts explicitly, allowing you to declare which identifiers to carry over to the next part.

```
MATCH (user)-[:FRIEND]-(friend)
WITH user, count(friend) AS friends
ORDER BY friends DESC
SKIP 1 LIMIT 3
RETURN user
```
You can also use `ORDER BY`, `SKIP`, `LIMIT` with `WITH`.

UNION

```
MATCH (a)-[:KNOWS]->(b)
RETURN b.name
UNION
MATCH (a)-[:LOVES]->(b)
RETURN b.name
```
Returns the distinct union of all query results. Result column types and names have to match.

```
MATCH (a)-[:KNOWS]->(b)
RETURN b.name
UNION ALL
MATCH (a)-[:LOVES]->(b)
RETURN b.name
```
Returns the union of all query results, including duplicated rows.

SET

```
SET n.property1 = {value1},
    n.property2 = {value2}
```
Update or create a property.

```
SET n = {map}
```
Set all properties. This will remove any existing properties.

```
SET n += {map}
```
Add and update properties, while keeping existing ones.

```
SET n:Person
```
Adds a label `Person` to a node.

INDEX

```
CREATE INDEX ON :Person(name)
```
Create an index on the label `Person` and property `name`.

```
MATCH (n:Person) WHERE n.name = {value}
```
An index can be automatically used for the equality comparison. Note that for example `lower(n.name) = {value}` will not use an index.

```
MATCH (n:Person) WHERE n.name IN [{value}]
```
An index can be automatically used for the IN collection checks.

```
MATCH (n:Person)
USING INDEX n:Person(name)
WHERE n.name = {value}
```
Index usage can be enforced, when Cypher uses a suboptimal index or more than one index should be used.

```
DROP INDEX ON :Person(name)
```
Drop the index on the label `Person` and property `name`.

CONSTRAINT

```
CREATE CONSTRAINT ON (p:Person)
       ASSERT p.name IS UNIQUE
```
Create a unique property constraint on the label `Person` and property `name`. If any other node with that label is updated or created with a `name` that already exists, the write operation will fail. This constraint will create an accompanying index.

```
DROP CONSTRAINT ON (p:Person)
       ASSERT p.name IS UNIQUE
```
Drop the unique constraint and index on the label `Person` and property `name`.

```
CREATE CONSTRAINT ON (p:Person)
       ASSERT exists(p.name)
```
Create a node property existence constraint on the label `Person` and property `name`. If a node with that label is created without a `name`, or if the `name` property is removed from an existing node with the `Person` label, the write operation will fail.

```
DROP CONSTRAINT ON (p:Person)
       ASSERT exists(p.name)
```
Drop the node property existence constraint on the label `Person` and property `name`.

```
CREATE CONSTRAINT ON ()-[l:LIKED]-()
       ASSERT exists(l.when)
```
Create a relationship property existence constraint on the type `LIKED` and property `when`. If a relationship with that type is created without a `when`, or if the `when` property is removed from an existing relationship with the `LIKED` type, the write operation will fail.

```
DROP CONSTRAINT ON ()-[l:LIKED]-()
       ASSERT exists(l.when)
```
Drop the relationship property existence constraint on the type `LIKED` and property `when`.

Import

```
LOAD CSV FROM
'http://neo4j.com/docs/2.3.3/cypher-refcard
/csv/artists.csv' AS line
CREATE (:Artist {name: line[1], year: toInt(line[2])})
```
Load data from a CSV file and create nodes.

```
LOAD CSV WITH HEADERS FROM
'http://neo4j.com/docs/2.3.3/cypher-refcard/csv/artists-
with-headers.csv' AS line
CREATE (:Artist {name: line.Name, year: toInt(line.Year)})
```
Load CSV data which has headers.

```
LOAD CSV FROM
```

Patterns

```
(n:Person)
```
Node with `Person` label.

```
(n:Person:Swedish)
```
Node with both `Person` and `Swedish` labels.

```
(n:Person {name: {value}})
```
Node with the declared properties.

```
(n)-->(m)
```
Relationship from `n` to `m`.

```
(n)--(m)
```
Relationship in any direction between `n` and `m`.

```
(n:Person)-->(m)
```
Node `n` labeled `Person` with relationship to `m`.

```
(n)<-[:KNOWS]-(m)
```
Relationship of type `KNOWS` from `n` to `m`.

```
(n)-[:KNOWS|:LOVES]->(m)
```
Relationship of type `KNOWS` or of type `LOVES` from `n` to `m`.

```
(n)-[r]->(m)
```
Bind the relationship to identifier `r`.

```
(n)-[*1..5]->(m)
```
Variable length path of between 1 and 5 relationships from `n` to `m`.

```
(n)-[*]->(m)
```
Variable length path of any number of relationships from `n` to `m`. (Please see the performance tips.)

```
(n)-[:KNOWS]->(m {property: {value}})
```
A relationship of type `KNOWS` from a node `n` to a node `m` with the declared property.

```
shortestPath((n1:Person)-[*..6]-(n2:Person))
```
Find a single shortest path.

```
allShortestPaths((n1:Person)-[*..6]->(n2:Person))
```
Find all shortest paths.

```
size((n)-->()-->())
```
Count the paths matching the pattern.

Collections

```
["a", "b", "c"] AS coll
```
Literal collections are declared in square brackets.

```
size({coll}) AS len, {coll}[0] AS value
```
Collections can be passed in as parameters.

```
range({firstNum}, {lastNum}, {step}) AS coll
```
Range creates a collection of numbers (`step` is optional), other functions returning collections are: `labels`, `nodes`, `relationships`, `rels`, `filter`, `extract`.

```
MATCH (a)-[r:KNOWS*]->()
RETURN r AS rels
```
Relationship identifiers of a variable length path contain a collection of relationships.

```
RETURN matchedNode.coll[0] AS value,
       size(matchedNode.coll) AS len
```
Properties can be arrays/collections of strings, numbers or booleans.

```
coll[{idx}] AS value,
coll[{startIdx}..{endIdx}] AS slice
```
Collection elements can be accessed with `idx` subscripts in square brackets. Invalid indexes return `NULL`. Slices can be retrieved with intervals from `start_idx` to `end_idx` each of which can be omitted or negative. Out of range elements are ignored.

```
UNWIND {names} AS name
MATCH (n {name: name})
```

Relationship Functions

```
type(a_relationship)
```
String representation of the relationship type.

```
startNode(a_relationship)
```
Start node of the relationship.

```
endNode(a_relationship)
```
End node of the relationship.

```
id(a_relationship)
```
The internal id of the relationship.

Collection Predicates

```
all(x IN coll WHERE exists(x.property))
```
Returns `true` if the predicate is `TRUE` for all elements of the collection.

```
any(x IN coll WHERE exists(x.property))
```
Returns `true` if the predicate is `TRUE` for at least one element of the collection.

```
none(x IN coll WHERE exists(x.property))
```
Returns `TRUE` if the predicate is `FALSE` for all elements of the collection.

```
single(x IN coll WHERE exists(x.property))
```
Returns `TRUE` if the predicate is `TRUE` for exactly one element in the collection.

Functions

```
coalesce(n.property, {defaultValue})
```
The first non-`NULL` expression.

```
timestamp()
```
Milliseconds since midnight, January 1, 1970 UTC.

```
id(nodeOrRelationship)
```
The internal id of the relationship or node.

```
toInt({expr})
```
Converts the given input into an integer if possible; otherwise it returns `NULL`.

```
toFloat({expr})
```
Converts the given input into a floating point number if possible; otherwise it returns `NULL`.

```
keys({expr})
```
Returns a collection of string representations for the property names of a node, relationship, or map.

Path Functions

```
length(path)
```
The number of relationships in the path.

```
nodes(path)
```
The nodes in the path as a collection.

```
relationships(path)
```
The relationships in the path as a collection.

```
extract(x IN nodes(path) | x.prop)
```
Extract properties from the nodes in a path.

Mathematical Functions

```
abs({expr})
```
The absolute value.

```
rand()
```
A random number between 0 and 1. Returns a new value for each call. Also useful for selecting subset or random ordering.

```
round({expr})
```
Round to the nearest integer, `ceil` and `floor` find the

Predicates

```
n.property <> {value}
```
Use comparison operators.

```
exists(n.property)
```
Use functions.

```
n.number >= 1 AND n.number <= 10
```
Use boolean operators to combine predicates.

```
1 <= n.number <= 10
```
Use chained operators to combine predicates.

```
n:Person
```
Check for node labels.

```
identifier IS NULL
```
Check if something is `NULL`.

```
NOT exists(n.property) OR n.property = {value}
```
Either property does not exist or predicate is `TRUE`.

```
n.property = {value}
```
Non-existing property returns `NULL`, which is not equal to anything.

```
n["property"] = {value}
```
Properties may also be accessed using a dynamically computed property name.

```
n.property STARTS WITH "Tob" OR
n.property ENDS WITH "n" OR
n.property CONTAINS "goodie"
```
String matching.

```
n.property =~ "Tob.*"
```
String regular expression matching.

```
(n)-[:KNOWS]->(m)
```
Make sure the pattern has at least one match.

```
NOT (n)-[:KNOWS]->(m)
```
Exclude matches to `(n)-[:KNOWS]->(m)` from the result.

```
n.property IN [{value1}, {value2}]
```
Check if an element exists in a collection.

Collection Expressions

```
size({coll})
```
Number of elements in the collection.

```
head({coll}), last({coll}), tail({coll})
```
`head` returns the first, `last` the last element of the collection. `tail` returns all but the first element. All return `NULL` for an empty collection.

```
[x IN coll WHERE x.prop <> {value} | x.prop]
```
Combination of filter and extract in a concise notation.

```
extract(x IN coll | x.prop)
```
A collection of the value of the expression for each element in the original collection.

```
filter(x IN coll WHERE x.prop <> {value})
```
A filtered collection of the elements where the predicate is `TRUE`.

```
reduce(s = "", x IN coll | s + x.prop)
```
Evaluate expression for each element in the collection, accumulate the results.

Aggregation

```
count(*)
```
The number of matching rows.

```
count(identifier)
```
The number of non-`NULL` values.

```
count(DISTINCT identifier)
```
All aggregation functions also take the `DISTINCT` modifier,

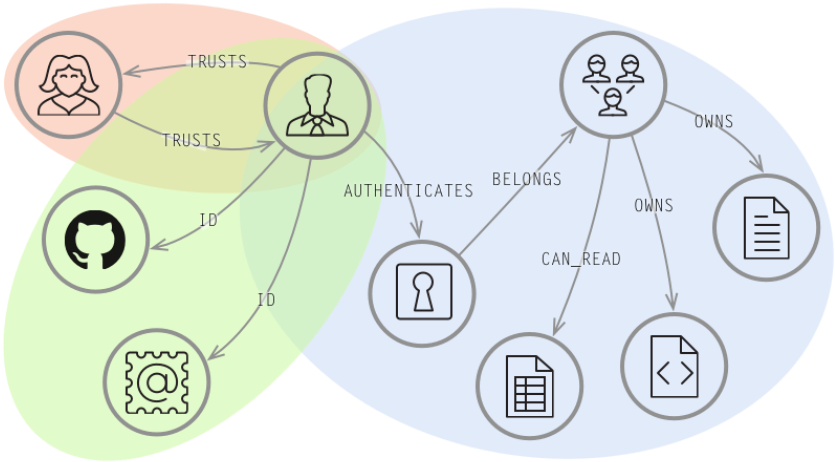
http://neo4j.com/docs/stable/cypher-refcard/

6

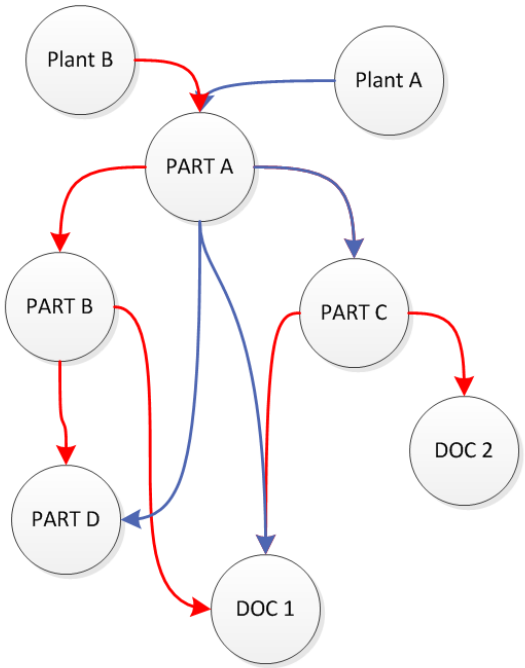
Graphe et "problématiques PLM"

Quelques exemples

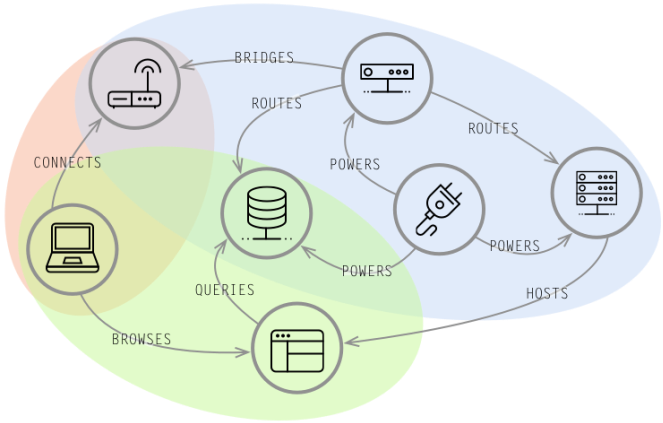
Permissions & Access rights



BOM, Configurable BOM, Multi BOM



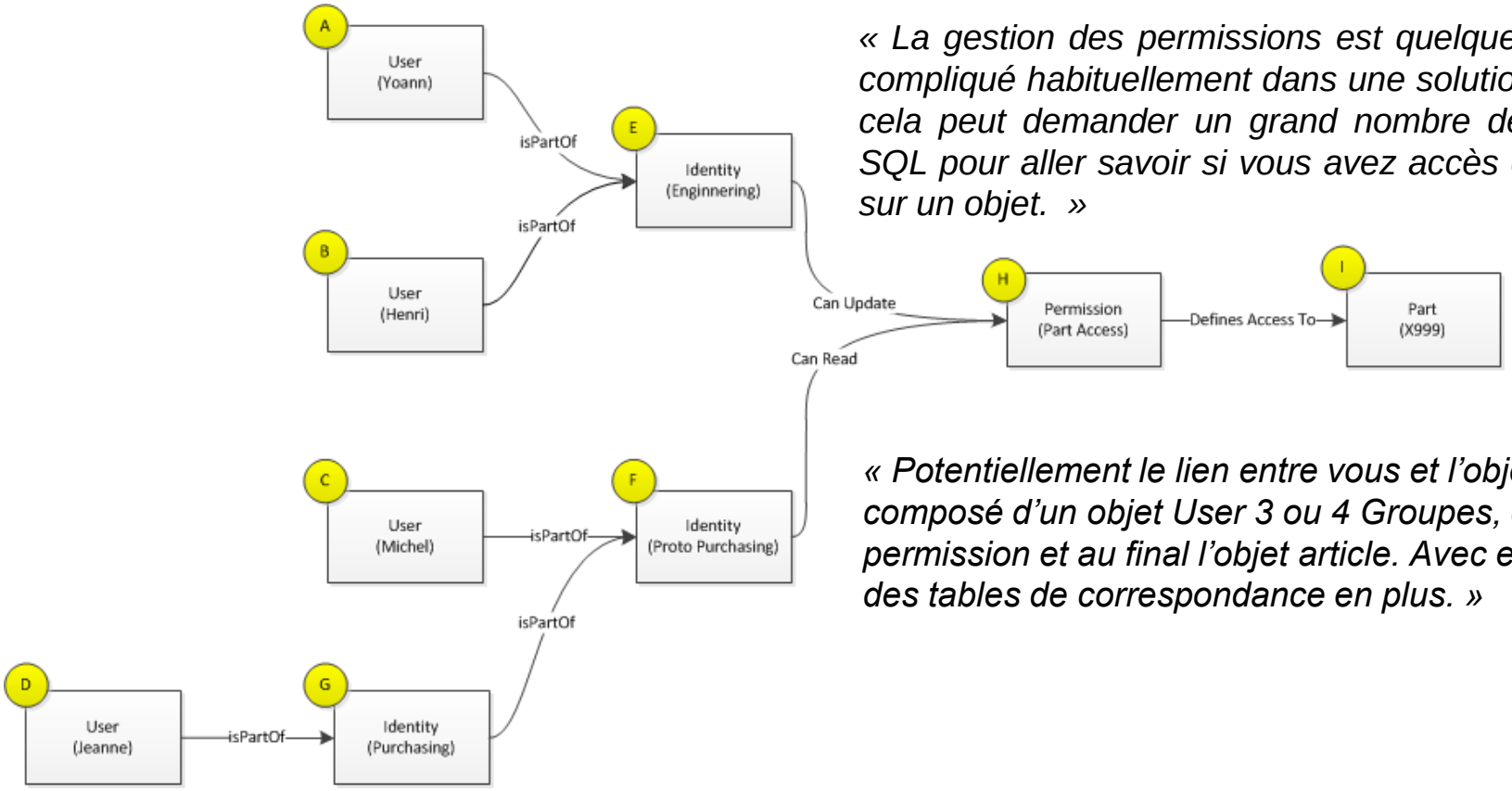
Impact Analysis



http://slides.com/yoannmaingon/graphdb-for-plm-plmlab

Focus : Gestion des permissions

http://plm-ouvert.fr/2014/07/neo4j-pourquoi-une-base-de-donnees-graphique-facilite-la-gestion-des-acces-pour-le-plm/



« La gestion des permissions est quelque chose de compliqué habituellement dans une solution PLM car cela peut demander un grand nombre de JOIN en SQL pour aller savoir si vous avez accès en écriture sur un objet. »

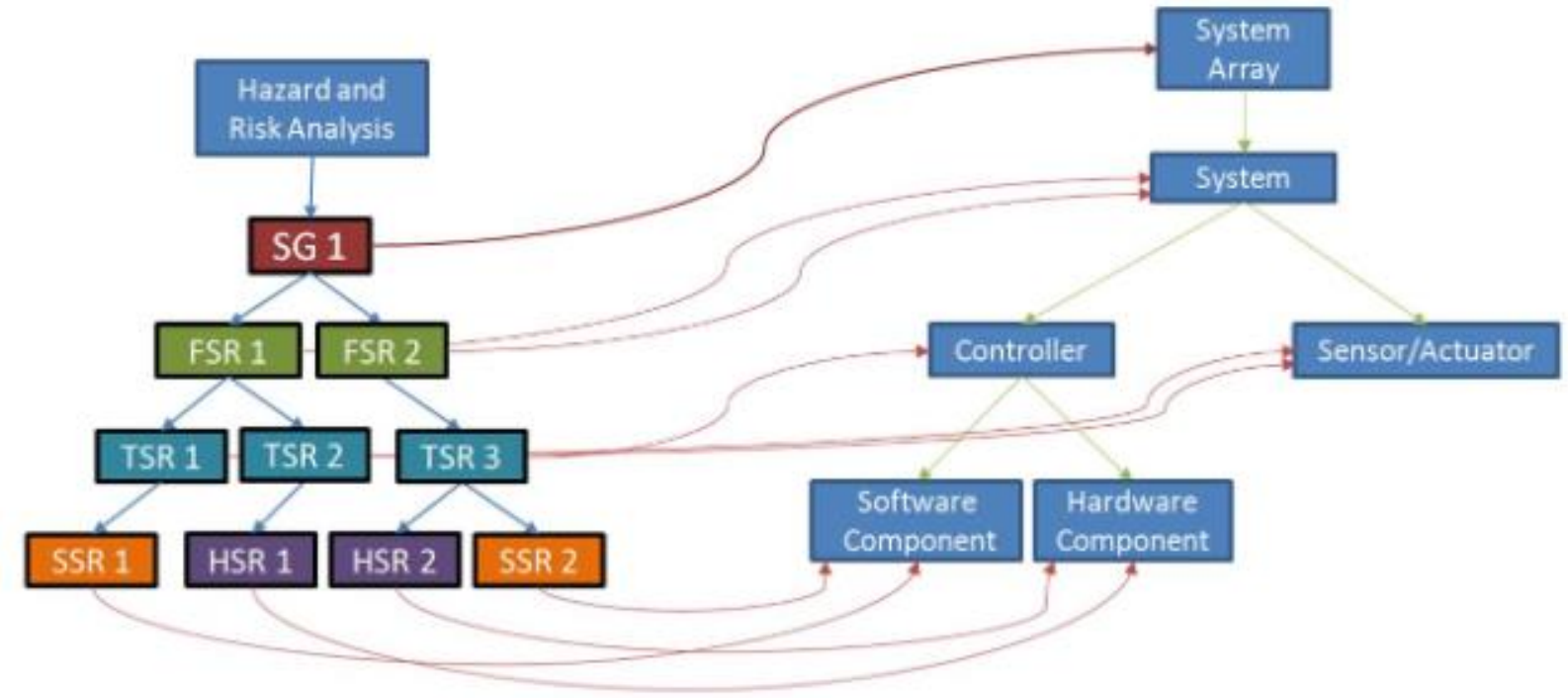
« Potentiellement le lien entre vous et l'objet sera composé d'un objet User 3 ou 4 Groupes, d'un objet de permission et au final l'objet article. Avec entre chacun, des tables de correspondance en plus. »

```
MATCH (a:User)-[:isPartOf*]->(b:Identity)-[:canUpdate]->(c:Permission)-[:definesAccessTo]->(d:Part)
WHERE d.partNumber= »X999" RETURN a

MATCH (a:User)-[:isPartOf*]->(b:Identity)-[:canRead]->(c:Permission)-[:definesAccessTo]->(d:Part)
WHERE d.partNumber= »X999" RETURN a
```

Application Neo4J aux Exigences

- Visualisation of requirements and their relations in embedded systems



http://www.diva-portal.org/smash/get/diva2:757476/FULLTEXT01.pdf