



Modélisation des connaissances

Joseph Azar

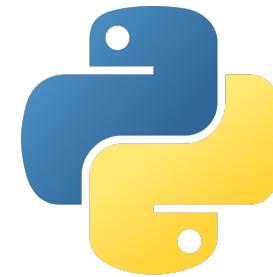
MCF Univ. Franche Comté / FEMTO-ST

https://github.com/josephazar/graph_of_things



Agenda

- Introduction à la modélisation des connaissances et à l'ontologie
- Introduction à la modélisation avec des graphes
- Graph of Things pour l'IoT avec NEO4J
- Comment construire un graphe de connaissances ?



La gestion des connaissances - Une force compétitive

- La gestion des connaissances est aujourd'hui un atout puissant pour rester compétitif sur presque tous les marchés.
- Elle consiste à transformer les connaissances individuelles et collectives en vue d'améliorer leur réutilisation.
- Cela implique trois éléments clés : **les personnes, les processus et la technologie**.
- L'objectif est de délivrer la bonne information, au bon moment, au bon endroit, au bon public, et dans le bon format.



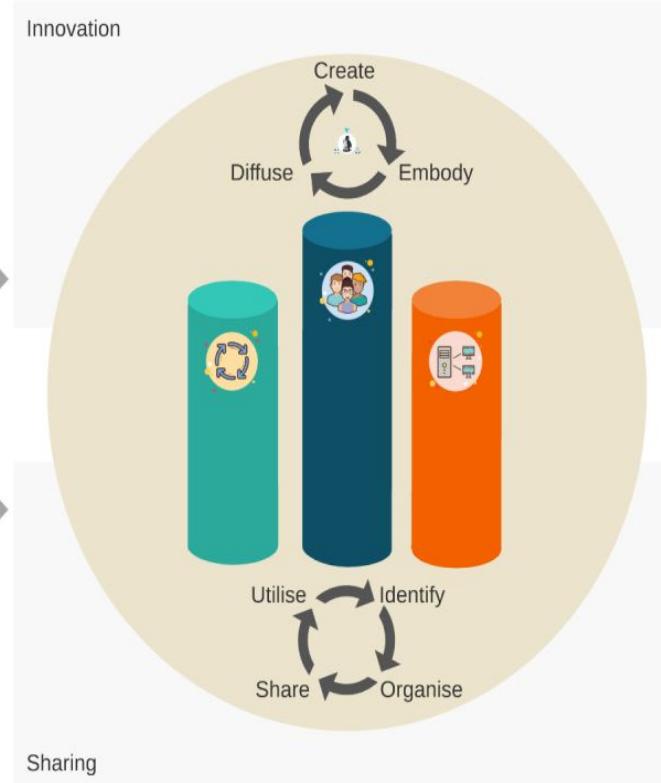
Les types de connaissances

- **Connaissances explicites** : formelles, systématiques, faciles à stocker et à partager (ex. : documents, vidéos).
- **Connaissances tacites** : acquises par l'expérience personnelle, difficiles à exprimer.
- **Connaissances implicites** : internes, mais exprimables à l'extérieur, par exemple via des interviews d'experts.



Le cycle de vie des connaissances

- Le cycle de vie des connaissances se divise en deux dimensions :
 - **La couche d'innovation :** création de nouvelles connaissances, qui sont ensuite intégrées dans des modèles et commercialisées.
 - **La couche de partage :** identification, organisation et diffusion de connaissances existantes pour la prise de décision.
- Les piliers de la gestion des connaissances : **les personnes, les processus et la technologie soutiennent ce cycle.**



Application

Conception d'un moteur automobile

- **Connaissance explicite** : Plans techniques, manuels de conception et normes industrielles (par exemple, un manuel expliquant comment assembler un moteur selon des normes spécifiques).
- **Connaissance tacite** : L'expérience d'un ingénieur expérimenté qui a travaillé sur la conception et l'assemblage de moteurs pendant des années, et qui sait instinctivement comment ajuster les pièces pour améliorer les performances.
- **Connaissance implicite** : Un étudiant peut être formé à analyser des moteurs à combustion et, au fil du temps, acquérir des connaissances sur les meilleures méthodes d'optimisation des performances, qu'il pourra ensuite partager sous forme de recommandations ou d'améliorations dans la conception.

Application du cycle de vie des connaissances :

- **Couche d'innovation** : Créer de nouvelles idées pour améliorer l'efficacité du moteur en utilisant les connaissances tacites et explicites.
- **Couche de partage** : Organiser et stocker ces connaissances dans des bases de données accessibles à l'équipe, puis les diffuser pour améliorer les conceptions futures.

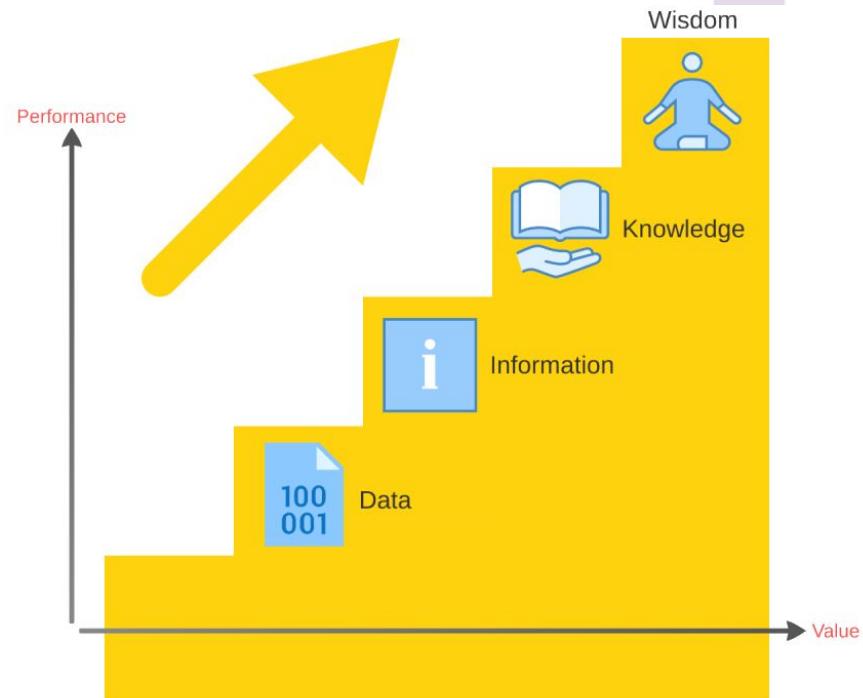
Importance de la capture des connaissances

Pourquoi capturer les connaissances ?

- Facilite la recherche, l'accès, le partage, la réutilisation et l'extension de connaissances précieuses.
- Réduit les coûts liés à la perte de connaissances au sein d'une entreprise, d'un département ou d'une équipe.
- Permet d'améliorer l'efficacité, de réduire le temps de formation et les ressources nécessaires.

Différence entre données, informations et connaissances :

- Données : Ensemble brut de caractères ou de symboles sans contexte explicite.
- Informations : Données organisées avec un contexte qui leur donne du sens et de la valeur.
- Connaissances : Informations enrichies par l'expérience et l'expertise, permettant d'obtenir des résultats spécifiques.



Application

Conception d'une pièce mécanique avec un logiciel de CAO (Conception Assistée par Ordinateur)

- **Phase des données** : Au début, les dimensions et spécifications techniques sont des ensembles de chiffres sans signification claire.
- **Phase des informations** : Après quelques semaines, en travaillant avec le logiciel, ces données prennent un sens concret, et on comprend comment elles influencent la conception de la pièce.
- **Phase des connaissances** : Avec l'expérience, on devient compétent dans l'utilisation du logiciel de CAO, on sait comment optimiser la conception et éviter les erreurs.
- **Sagesse** : La sagesse vient après plusieurs projets, où l'on développe une intuition sur les meilleures pratiques et les solutions innovantes dans la conception mécanique.

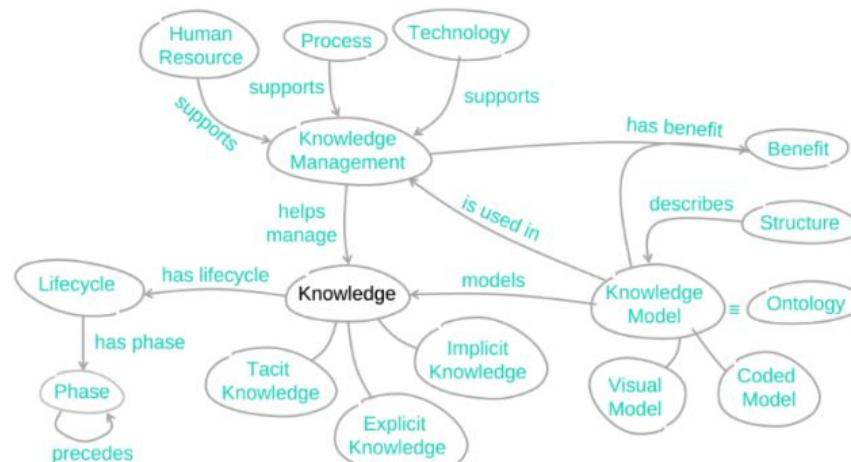
Modélisation des connaissances et cartes heuristiques

Cartes heuristiques (mind maps) : Un outil pour organiser des concepts autour d'un thème central.

- Exemple : Une carte heuristique des concepts liés à la gestion des connaissances permet de clarifier et de partager des idées.
- La carte permet de comprendre comment les concepts sont organisés et identifier les relations entre eux.

Modélisation des connaissances :

- Un modèle de connaissance va plus loin qu'une carte heuristique : il structure et clarifie les concepts pour une interprétation à la fois humaine et machine.
- Cette structure permet de mieux partager le sens et d'améliorer l'interaction entre humains et systèmes.



Ontologie - Une représentation formelle des connaissances

Qu'est-ce qu'une ontologie ?

- Une ontologie est une représentation formelle et explicite d'une conceptualisation partagée, compréhensible par les humains et les machines.
- Elle fonctionne comme un plan ou une blueprint pour représenter un domaine de connaissances, tout comme un plan d'architecture pour construire des bâtiments.

Applications des ontologies :

- Sémantique et représentation de domaines
- Conception de systèmes
- Modélisation d'entreprises, vocabulaire contrôlé, et plus encore

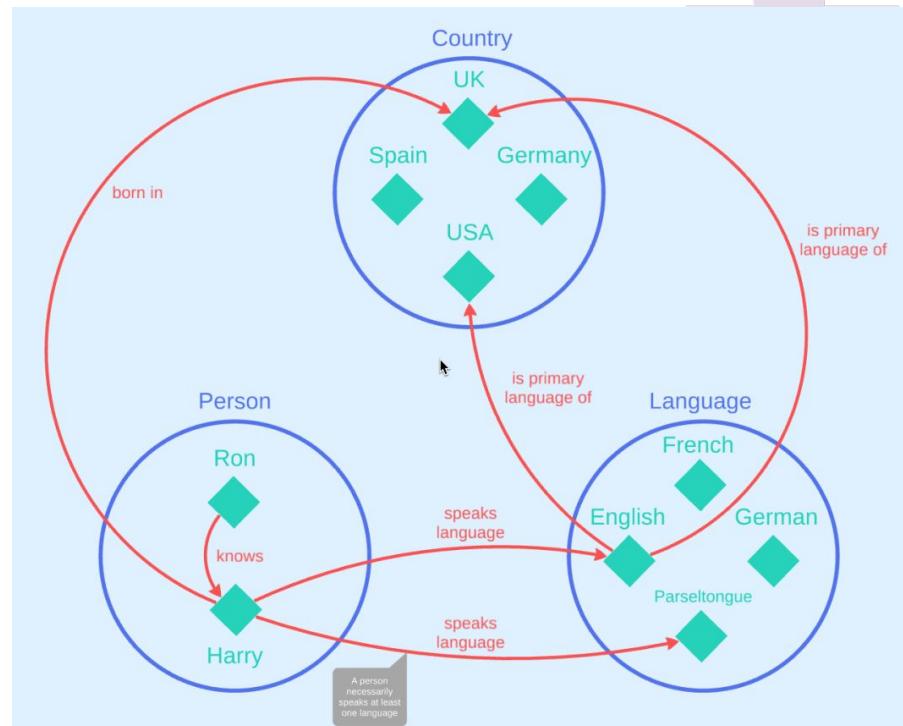
Perspective philosophique :

- L'ontologie en philosophie est l'étude de l'existence et des entités.
- Elle examine des questions fondamentales comme : « Qu'est-ce que l'existence ? » et « Qu'est-ce qu'une entité ? ».

Les instances et leurs relations

Instances : Représentent des occurrences spécifiques d'un concept (ex. : une tasse de café, un smartphone, un stylo).

- Exemples : Harry et Ron (personnages), l'anglais et le parseltongue (langues), le Royaume-Uni (pays).
- Relations entre instances : Harry connaît Ron, Harry parle anglais et parseltongue, l'anglais est la langue principale du Royaume-Uni.
- Ces relations aident à décrire et comprendre les objets dans un domaine. En modélisation des connaissances, on parle de triplet pour décrire une relation entre deux instances.



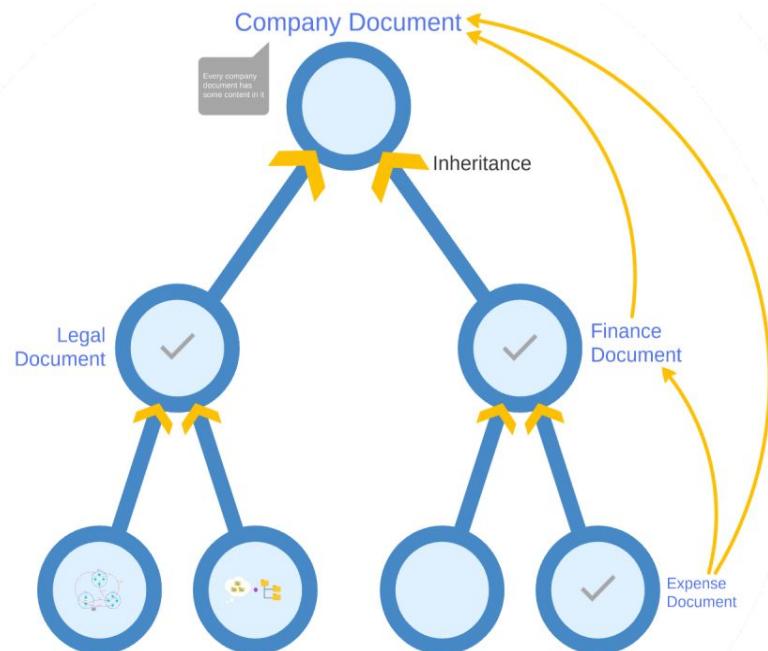
Classes et taxonomie

Classes : Groupes d'individus partageant des caractéristiques communes (ex. : Personne, Pays, Langue).

- Une classe définit les critères pour qu'un individu en fasse partie.
- Exemple : La classe "Personne" pourrait inclure tout individu ayant un numéro de sécurité sociale.

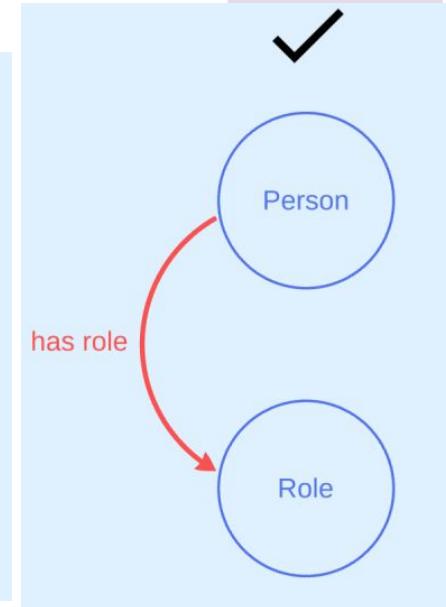
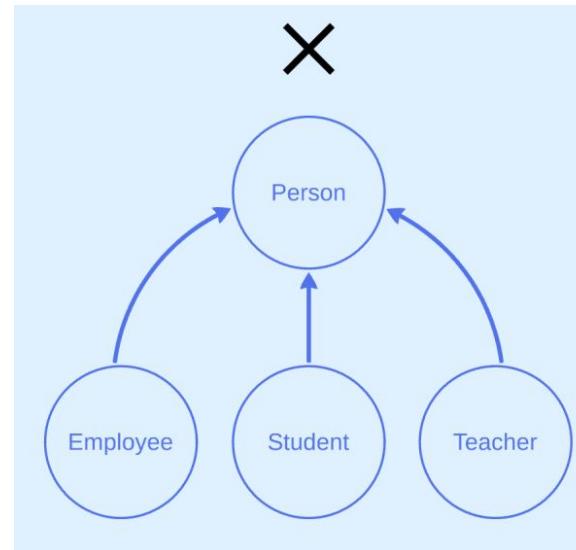
Taxonomie : Hiérarchie des classes organisée en structure d'arbre (ex. : Document d'entreprise > Document financier > Document de dépenses).

- Héritage : Les sous-classes héritent des attributs de leurs classes parentales.



Importance de la taxonomie dans la modélisation

- **Héritage transitive** : Une sous-classe hérite des caractéristiques de sa classe parentale, ainsi que des sous-classes au-dessus d'elle (ex. : Document de dépenses hérite des attributs de Document financier et Document d'entreprise).
- **Conception correcte de la taxonomie** : Important pour éviter des confusions (ex. : Ne pas classer une "Personne" uniquement en "Étudiant" ou "Employé", car ces rôles changent).
 - Solution : Créer une classe "Personne" avec une classe séparée "Rôle" pour définir les rôles individuels joués par chaque personne.
- Tout cela fait partie du domaine de discours, représentant l'ensemble des classes, relations et règles du modèle.



Application: domaine de discours

Classes : Les différentes catégories de composants ou sous-systèmes du moteur.

- Exemples : Cylindre, Piston, Vilebrequin, Soupape, Système de carburant.

Relations : Décrit comment ces composants interagissent entre eux.

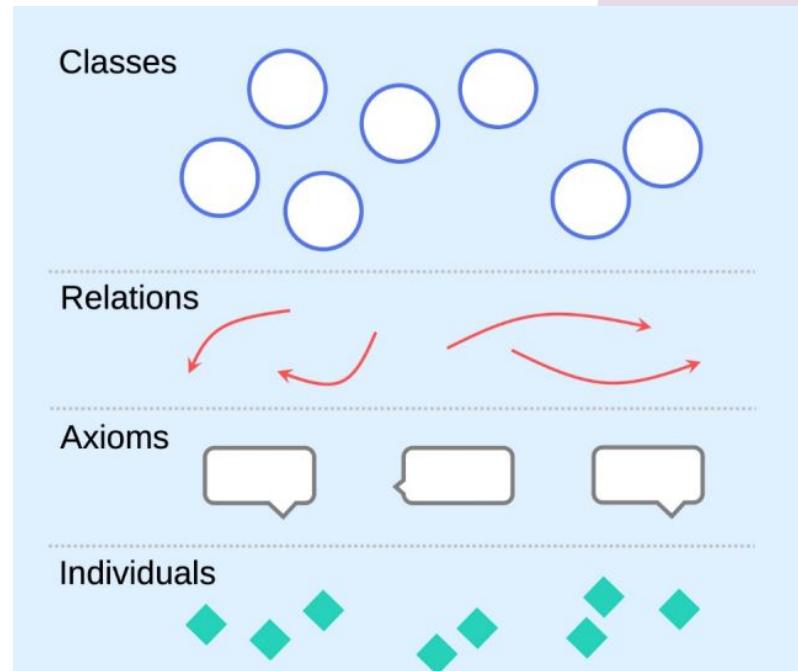
- Exemples : Le Piston se déplace à l'intérieur du Cylindre, le Carburant est injecté dans le Cylindre, le Vilebrequin est mis en rotation par le mouvement du Piston.

Axiomes : Règles ou contraintes définissant des propriétés spécifiques.

- Exemple : Un Piston doit correspondre à un Cylindre de taille donnée. Une Soupape s'ouvre uniquement à certains moments du cycle moteur.

Individus : Instances spécifiques de composants.

- Exemple : Cylindre #1, Piston #A.



Introduction aux graphes dans le monde réel

Exemples d'utilisation des graphes :

- Analyse des réseaux sociaux
- Détection de fraude
- Modélisation de la stabilité des systèmes (réseaux ferroviaires et énergétiques)
- Systèmes de recommandation

Pourquoi utiliser des graphes ?

- Les relations entre individus, comptes bancaires ou unités sont fondamentales pour décrire et modéliser les données.
- Les graphes représentent parfaitement les groupes d'éléments interagissant, contrairement aux modèles de données traditionnels.

Introduction aux graphes dans le monde réel

Exemples d'utilisation des graphes :

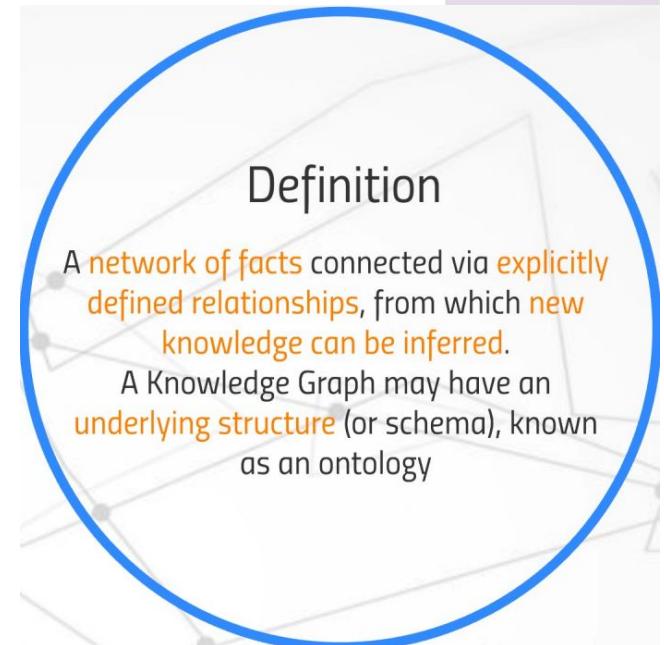
- Analyse des réseaux sociaux
- Détection de fraude
- Modélisation de la stabilité des systèmes (réseaux ferroviaires et énergétiques)
- Systèmes de recommandation

Pourquoi utiliser des graphes ?

- Les relations entre individus, comptes bancaires ou unités sont fondamentales pour décrire et modéliser les données.
- Les graphes représentent parfaitement les groupes d'éléments interagissant, contrairement aux modèles de données traditionnels.

Introduction au concept de graphe de connaissances

- Définition revisitée du graphe de connaissances.
- Les composants essentiels pour décrire un réseau basique : relation entre deux éléments.
- Exemple : Winston Churchill est lié à Elizabeth II.
- Le schéma « sujet-prédicat-objet » : la plus petite unité d'un réseau – aussi appelé « triple ».



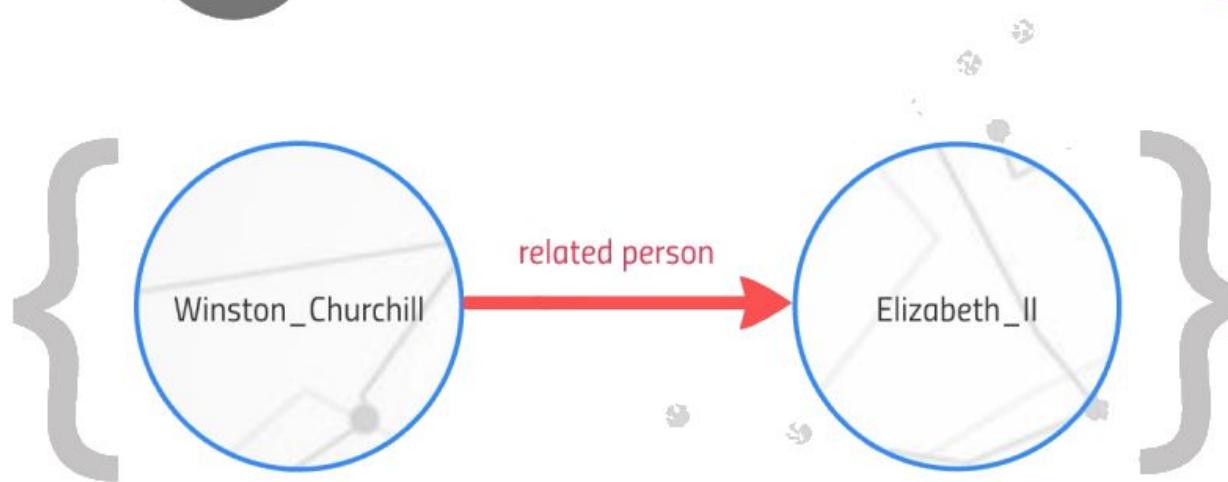
Introduction au concept de graphe de connaissances

1

Network of facts

A 'unit of knowledge'

Statement
(a.k.a. triple)



Fact ————— Relation ————— Fact

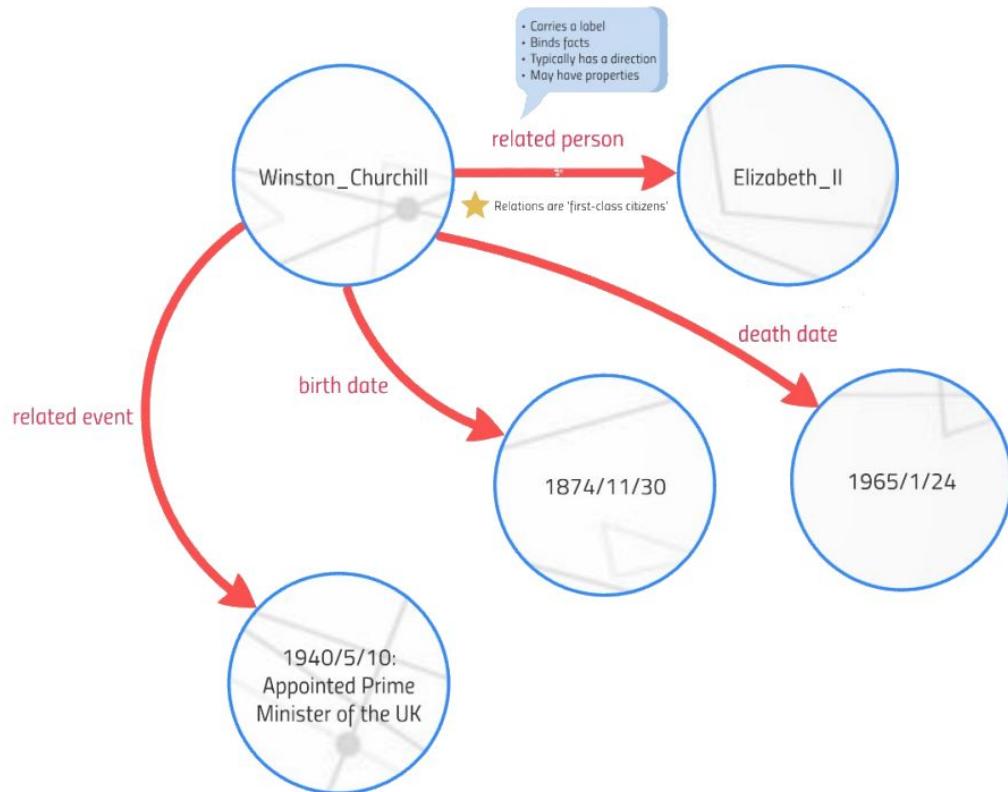
Node ————— Edge ————— Node

Vertex ————— Edge ————— Vertex

Subject ————— Predicate ————— Object

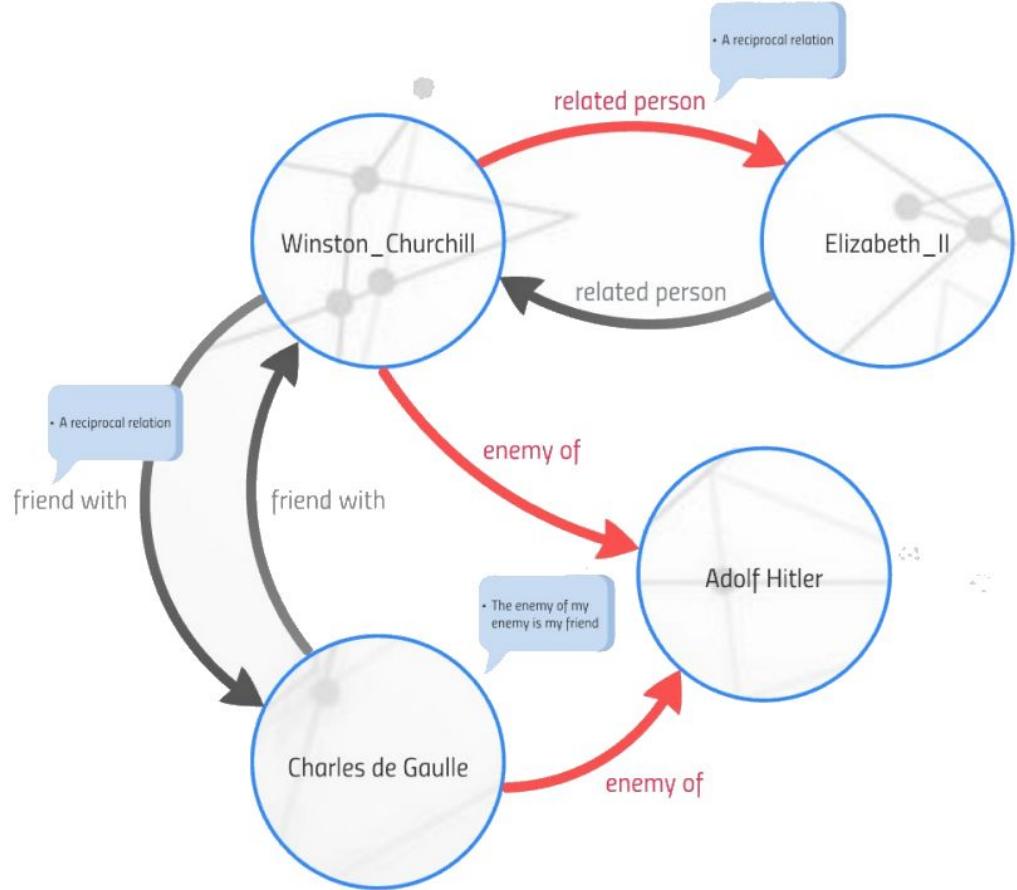
Les relations explicites dans les graphes de connaissances

- Les relations sont définies et étiquetées pour capturer du sens.
- Les relations peuvent avoir des propriétés (ex. : direction, réciprocité).
- Exemple : « Personne liée » est une relation réciproque entre Winston Churchill et Elizabeth II.



Les relations explicites dans les graphes de connaissances

- Les relations sont définies et étiquetées pour capturer du sens.
- Les relations peuvent avoir des propriétés (ex. : direction, **réciprocité**).
- Les graphes de connaissances permettent de déduire de nouvelles relations.
 - a. Exemple : Winston Churchill et Charles de Gaulle, ennemis d'Adolf Hitler -> « l'ennemi de mon ennemi est mon ami ».
 - b. Grâce aux règles, nous pouvons inférer que Churchill et de Gaulle sont amis.



Pourquoi utiliser les graphes ?

Importance des relations :

- Dans le monde moderne axé sur les données, les relations entre les éléments sont devenues aussi importantes, voire plus importantes que les éléments eux-mêmes.
- Les graphes aident à comprendre ces relations entre entités dans les entreprises et les industries modernes.

Exemples d'utilisation des graphes :

- Les graphes sont au cœur des systèmes que nous utilisons au quotidien (ex. : recommandations de produits en ligne).
- Ils sont essentiels dans les solutions modernes pour analyser et exploiter les relations entre les données.

Compétence clé en Data Science :

- Maîtriser les graphes et leur utilisation est une compétence de plus en plus recherchée en science des données.
- Les graphes permettent de représenter des systèmes mécaniques complexes, comme les réseaux de composants dans un moteur ou une chaîne de production.
- Ils facilitent la compréhension des interactions entre les différentes pièces et sous-systèmes.

Composants d'un graphe et domaines d'application

Les réseaux : un outil pour représenter des systèmes complexes

- Les graphes représentent les connexions complexes dans les systèmes modernes et les données actuelles.

Les graphes, ou réseaux, sont des structures de données particulièrement puissantes pour modéliser et décrire les relations entre les choses, que ces choses soient des personnes, des produits ou des machines. Dans un graphe, ces éléments sont représentés par des **nœuds (parfois appelés sommets)**. Les nœuds sont reliés par des **arêtes (parfois appelées relations)**. Dans un réseau, une arête représente une relation entre deux éléments, indiquant qu'ils sont liés d'une manière ou d'une autre.

Quatre domaines d'application des méthodes de graphes :

1. Mouvement :
 - Comment les objets se déplacent dans un réseau (ex. : GPS et solutions de routage).
 - Optimisation des trajets dans les réseaux routiers.
2. Influence :
 - Identifier les influenceurs sur les réseaux sociaux et comprendre la propagation de leur influence.
3. Groupes et interactions :
 - Détection de communautés dans un réseau et analyse des interactions entre acteurs (nœuds) et leurs connexions (arêtes).
4. Détection de motifs :
 - Identifier des similarités dans un réseau, par exemple entre des acteurs (personnes, profils d'auteurs, etc.).
 - Analyser les connexions entre les nœuds pour détecter des motifs ou des schémas.

Graphes non orientés

- Exemple simple : Réseau social :
 - Dans un graphe non orienté, les relations entre deux nœuds sont réciproques.
 - Exemple : Jeremy et Mark sont amis, cette relation d'amitié est représentée par une arête reliant les deux nœuds (Jeremy et Mark).
 - Ce type de relation est mutuelle, illustrée dans un graphe non orienté.
- Réseaux sociaux avec relations non réciproques :
 - Tous les réseaux sociaux ne fonctionnent pas de manière mutuelle.
 - Exemple : Sur Twitter, Jeremy peut suivre Mark sans que Mark suive Jeremy en retour.
 - Dans ce cas, on parlerait d'un graphe orienté où la relation a une direction.



Graphes orientés et relations directionnelles

Définition des graphes orientés :

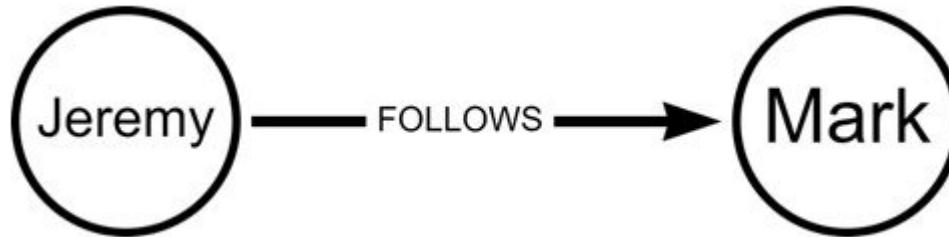
- Une arête directionnelle montre une relation asymétrique.
- Exemple : Jeremy suit Mark, une arête allant de Jeremy vers Mark, tandis qu'aucune arête n'existe dans l'autre sens (Mark ne suit pas Jeremy).

Relations bidirectionnelles :

- Dans certains cas, une relation mutuelle peut être représentée par deux arêtes orientées dans chaque sens, équivalentes à une arête non orientée (relation mutuelle).

Importance de nommer les relations :

- Lors de la modélisation de données avec des arêtes directionnelles, le nom de la relation doit être précis.
- Exemple : Si l'arête est nommée suit, elle va de Jeremy vers Mark. Si elle est nommée suivi par, elle doit aller dans l'autre sens (de Mark vers Jeremy).



Propriétés des nœuds et des arêtes

Propriétés des nœuds :

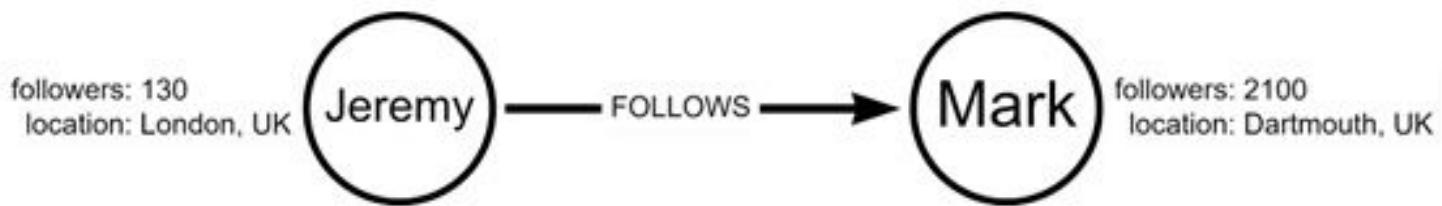
- Les nœuds peuvent avoir des données associées qui ne sont pas nécessairement relationnelles.
- Exemple : Le nombre d'abonnés ou la localisation de Jeremy et Mark sur Twitter.

Propriétés des arêtes :

- Les arêtes peuvent aussi contenir des informations supplémentaires en plus de représenter une relation.
- Exemple : Une ligne noire indique que Jeremy suit Mark (relation directionnelle).

Importance des propriétés pour les requêtes :

- Les propriétés des nœuds sont essentielles pour filtrer des données dans un graphe.
- Exemple : Identifier qui suit des utilisateurs ayant plus de 1 000 abonnés.



Propriétés des nœuds et des arêtes

Propriétés des nœuds :

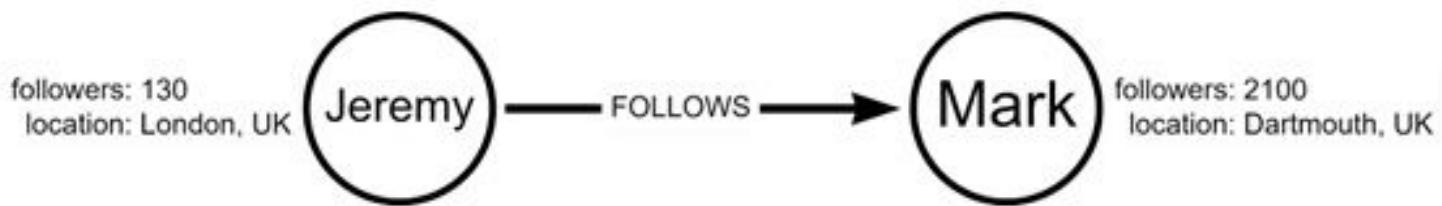
- Les nœuds peuvent avoir des données associées qui ne sont pas nécessairement relationnelles.
- Exemple : Le nombre d'abonnés ou la localisation de Jeremy et Mark sur Twitter.

Propriétés des arêtes :

- Les arêtes peuvent aussi contenir des informations supplémentaires en plus de représenter une relation.
- Exemple : Une ligne noire indique que Jeremy suit Mark (relation directionnelle).

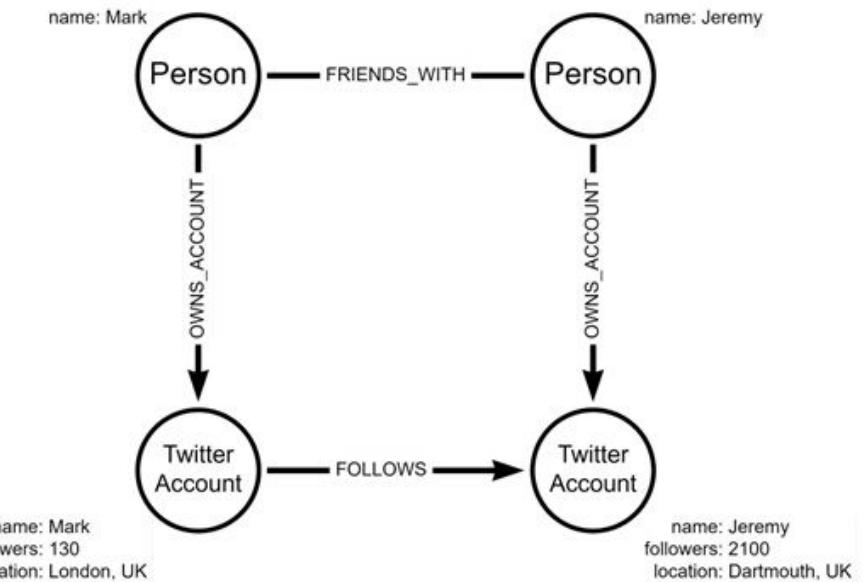
Importance des propriétés pour les requêtes :

- Les propriétés des nœuds sont essentielles pour filtrer des données dans un graphe.
- Exemple : Identifier qui suit des utilisateurs ayant plus de 1 000 abonnés.



Graphes hétérogènes

- Définition des graphes hétérogènes :
 - Un graphe est dit hétérogène lorsqu'il contient plusieurs types de nœuds et d'arêtes.
 - Exemple : Jeremy et Mark sont représentés en tant que personnes avec des relations différentes (relation en dehors de Twitter, propriété de compte).
- Propriétés des nœuds et des arêtes :
 - Chaque type de nœud peut avoir des propriétés distinctes (ex. : les personnes ont des propriétés comme le nom et l'âge, les comptes ont des propriétés comme le nombre d'abonnés).
 - Les arêtes doivent également refléter les différentes interactions entre ces types de nœuds.
- Graphes hétérogènes vs homogènes :
 - Graphes hétérogènes : Contiennent plusieurs types de nœuds et de relations.
 - Graphes homogènes : Contiennent un seul type de nœud et de relation.
- Importance des graphes hétérogènes :
 - Ils permettent de modéliser des réseaux complexes avec des interactions multiples entre différents types d'entités.



Conception de schéma

Définir les nœuds, arêtes et propriétés :

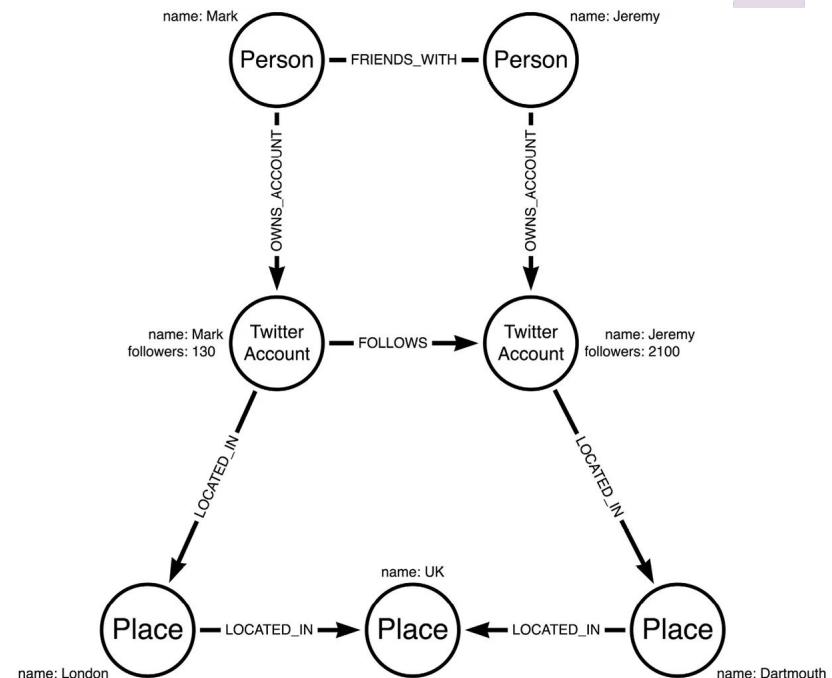
- Pour chaque ensemble de données, il existe différentes façons de représenter les informations en tant que graphe.
- Une bonne modélisation de graphe repose sur un schéma orienté par les cas d'utilisation ou les questions à traiter.

Exemple : Localisation des utilisateurs Twitter :

- Si la localisation est un élément clé, on peut déplacer la propriété "localisation" des nœuds utilisateurs vers un type de nœud distinct, puis créer une relation LOCATED_IN.
- On peut également ajouter des informations supplémentaires, comme le pays, en tant que nœud abstrait séparé.

Impact sur la conception et les performances :

- Ce changement dans la structure du graphe permet de modéliser les mêmes données de différentes manières, selon les besoins.
- La conception du schéma influence les types de questions qu'on peut poser et les performances des requêtes.

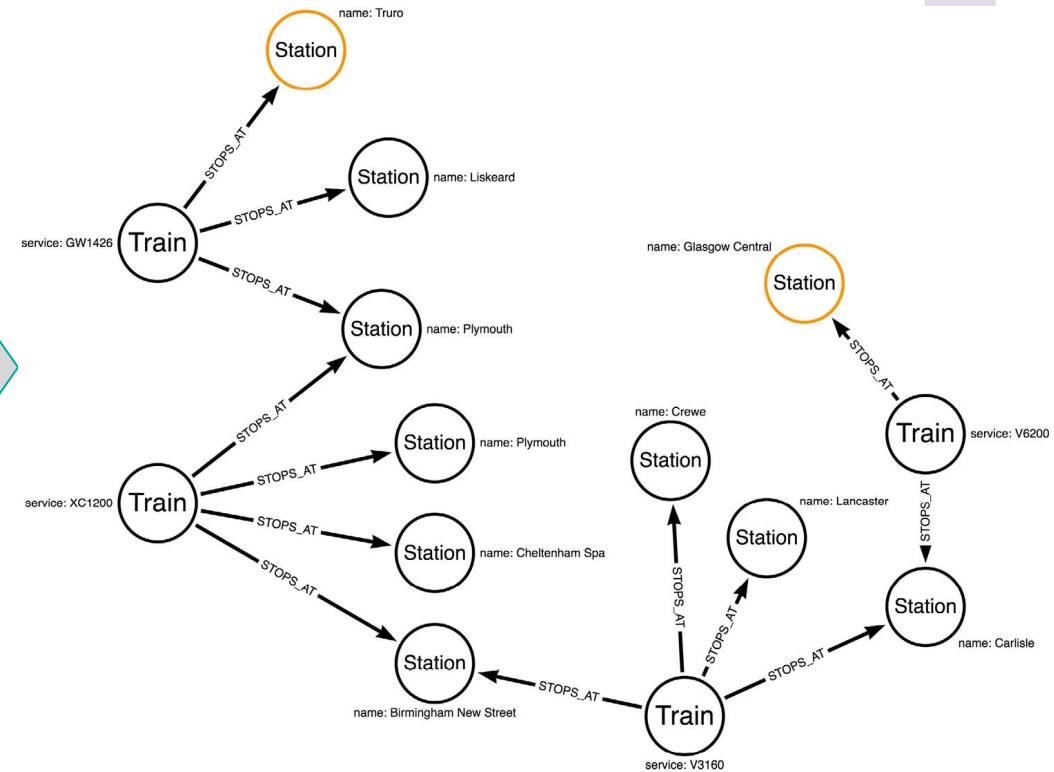
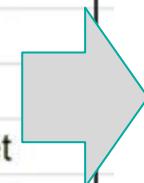


Comparaison entre RDB et GDB

- Bases de données relationnelles (RDBs) :
 - Les RDBs sont utilisées depuis longtemps pour stocker et analyser des données dans différents secteurs.
 - Elles permettent de lier des données à travers plusieurs tables grâce à des champs communs.
 - Avantage : Capables de renvoyer un grand nombre de lignes répondant à des critères spécifiques.
- Limites des RDBs :
 - Moins adaptées pour des questions impliquant des relations enchaînées multiples.
 - Exemple : Trouver un trajet entre deux gares nécessite plusieurs opérations de jointures coûteuses entre les données.
- Bases de données orientées graphes (GDBs) :
 - Les GDBs sont mieux adaptées pour modéliser et interroger des relations complexes entre des entités.
 - Exemple : Dans un graphe, le trajet entre deux gares peut être trouvé plus efficacement en naviguant directement les relations entre les nœuds (gares et services de train).
- Avantage des GDBs :
 - Les graphes permettent de traverser facilement les relations, sans nécessiter de jointures complexes, rendant les requêtes plus rapides et plus adaptées à des données fortement liées.

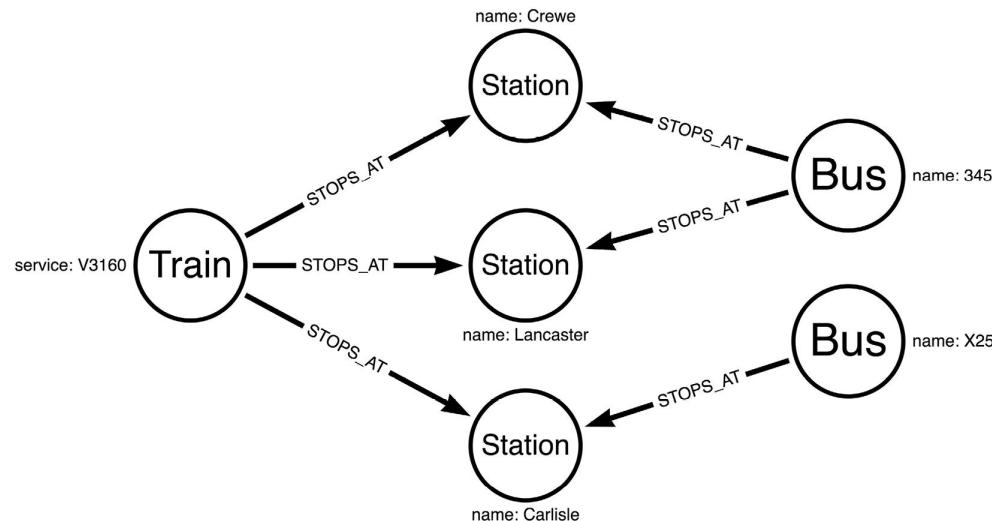
Comparaison entre RDB et GDB

Train	Stops at
GW1426	Truro
GW1426	Liskeard
GW1426	Plymouth
XC1200	Plymouth
XC1200	Bristol Parkway
XC1200	Cheltenham Spa
XC1200	Birmingham New Street
VT3160	Birmingham New Street
VT3160	Crewe
VT3160	Lancaster
VT3160	Carlisle
VT6200	Carlisle
VT6200	Glasgow Central



Comparaison entre RDB et GDB

- Importance des relations dans les graphes :
 - En utilisant une structure de graphe, l'accent est mis sur les relations entre les points de données (stations et trains).
 - Exemple : À partir de Truro, la recherche d'un trajet vers Glasgow Central est plus rapide car chaque station ou nœud de train a moins d'options à considérer.
- Comparaison avec les RDB :
 - Contrairement aux RDB où plusieurs jointures sont nécessaires, le graphe permet de traverser les relations directement, réduisant ainsi la complexité des opérations.
 - Résultat : Une méthode plus rapide et plus efficace, particulièrement adaptée aux requêtes de recherche de chemin.
- Modèle de données flexible et évolutif :
 - Les graphes sont utiles pour des modèles de données flexibles qui évoluent facilement.
 - Exemple : Ajouter des lignes de bus dans un réseau de train serait plus complexe avec une RDB, nécessitant une nouvelle table et la duplication des données des stations.
 - Avec un graphe, cette extension serait plus simple et sans augmenter la complexité du schéma.



Utilisation des graphes dans divers secteurs

- Finance :
 - Détection de fraude et analyse des risques de portefeuille.
- Gouvernement :
 - Profilage d'intelligence et analyse de la chaîne d'approvisionnement.
- Sciences de la vie :
 - Parcours des patients à l'hôpital, taux de réponse aux médicaments, propagation des infections dans une population.
- Réseaux & IT :
 - Sécurité des réseaux et gestion des accès utilisateurs (chaque utilisateur étant représenté par un nœud).
- Télécommunications :
 - Optimisation des réseaux et prédition du churn (attrition des clients).
- Marketing :
 - Segmentation des clients et du marché.
- Analyse des réseaux sociaux :
 - Modération des plateformes, protection contre les atteintes en ligne et défense des marques.
- Mécanique :
 - Optimisation des systèmes complexes, analyse des flux dans des chaînes de production, et maintenance prédictive.

Modélisation d'une page Web ou de documents

Étape 1 : Comprendre le contexte et l'objectif du document

Avant de plonger dans les détails techniques, les étudiants doivent se poser les questions suivantes :

- Quel est l'objectif du document ? Est-ce un projet technologique, un rapport, un article de blog, etc. ?
- Quelles sont les informations principales ? Par exemple, dans le cas de la page du Living Lab H2-Bois, les informations clés sont la transition énergétique, l'utilisation de l'hydrogène, l'innovation, etc.

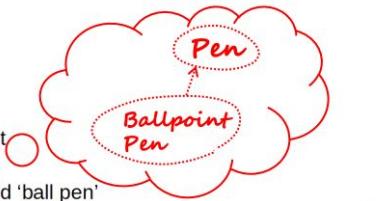
Cette étape aide à mieux cerner les éléments à représenter sous forme de graphe et à identifier les concepts clés.

Listing and analysing statements

Ballpoint Pen Example

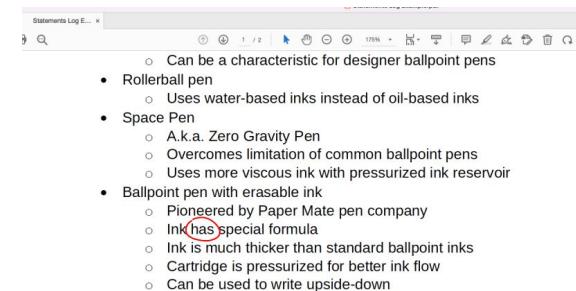
→ General

- A pen is a writing implement/instrument
- Ballpoint pen is a pen
- Ballpoint pen is also known as 'biro' and 'ball pen'
- Ballpoint pen dispenses ink over a metal ball at its point, i.e. over a 'ball point'
- Ballpoint pen is manufactured
- Most ballpoint pens rely on gravity to coat the ball with ink
- Most ballpoint pens cannot be used to write upside-down



→ Types of ballpoint pens

- Designer ballpoint pen
 - Intended for high-end and collectors' markets
 - Is produced by manufacturers
- Bic Cristal is a type of ballpoint pen
 - Is disposable



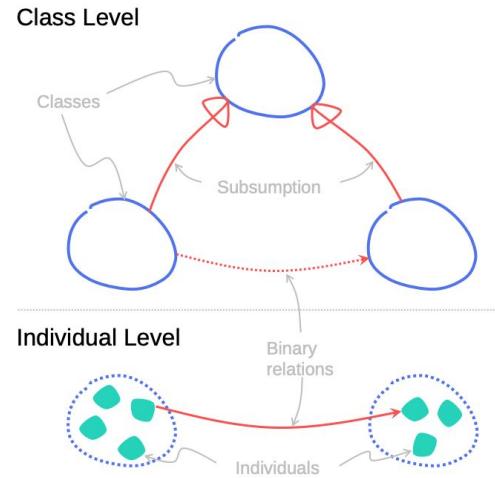
Parts and manufacturing

- Disposable type
 - Can have a cap to cover tip when pen is not in use
 - Can have a mechanism for retracting the tip
- Designs and constructions vary between brands
- Universal basic components
 - Free-floating ball point – distributes ink
 - Socket – holds the ball in place
 - Self-contained ink reservoir – supplies ink to the ball

Modélisation d'une page Web ou de documents

Étape 2: Identifier les entités et les relations

- Repérer les entités, identifier les relations et les individus
 - Les nœuds : Représentent les entités clés.
 - Les arêtes : Représentent les relations entre les nœuds.
 - Les attributs : Chaque nœud ou arête peut avoir des attributs supplémentaires (par exemple, la puissance d'une source d'énergie ou la capacité de stockage d'un système).



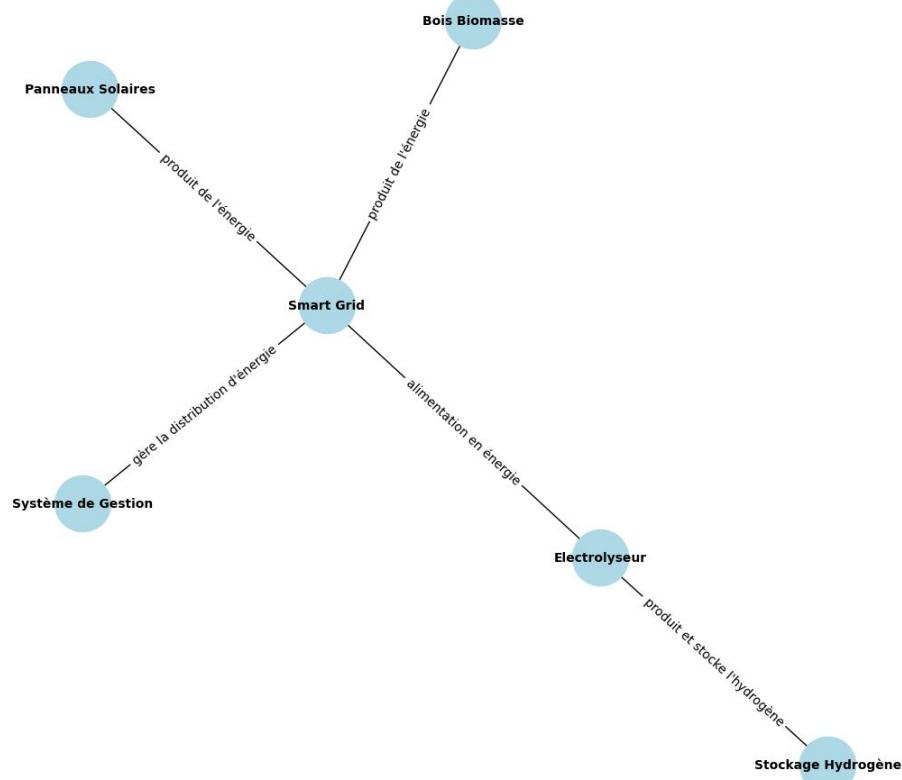
Entity	Camel Case Convention (Examples)	Underscore Convention (Examples)
Class	Pen, BallpointPen	Pen, Ballpoint_Pen
Relation	dispenses, hasComponent	dispenses, has_component
Individual	Pen001, MyFavouritePen	Pen_001, My_Favourite_Pen

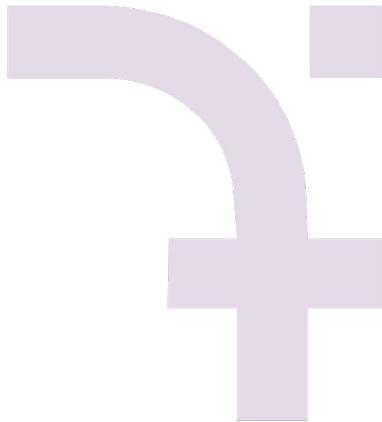
Modélisation d'une page Web ou de documents

Étape 3 : Dessiner un schéma

Avant d'écrire du code, il peut être utile de dessiner le schéma du graphe sur papier ou sur un tableau. Cela permet de visualiser les entités et leurs relations. Par exemple :

- Un nœud pour les panneaux solaires.
- Un nœud pour le Smart Grid.
- Une arête entre ces deux nœuds indiquant que le Smart Grid est alimenté par les panneaux solaires.





Introduction aux bases de données graphiques





Database Systems: SQL vs NoSQL

- Les bases de données SQL utilisent un langage de requête structuré pour accéder aux données et les manipuler, tandis que les bases de données NoSQL utilisent des modèles de données non structurés ou semi-structurés.
- Les bases de données NoSQL peuvent être classées en quatre catégories : basées sur des documents, clés-valeurs, colonnes et graphiques.
- Les bases de données basées sur des documents stockent les données dans des documents, qui peuvent être imbriqués avec des champs et des tableaux.
- Les bases de données clés-valeurs stockent les données dans une paire clé-valeur.
- Les bases de données en colonnes organisent les données par colonnes au lieu de lignes.
- Les bases de données graphiques stockent les données dans des nœuds et des arêtes, ce qui les rend idéales pour les cas d'utilisation impliquant des relations.

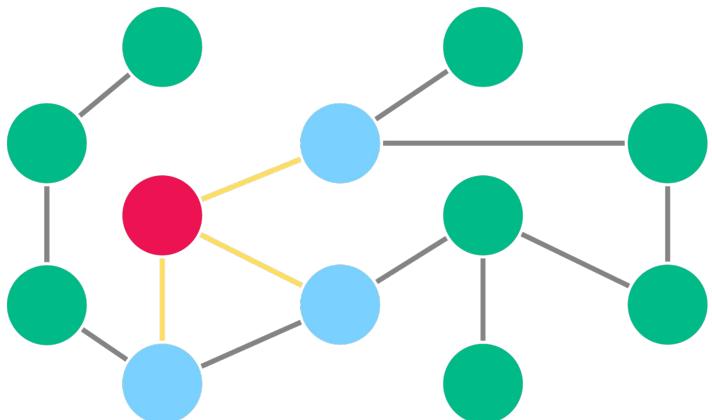
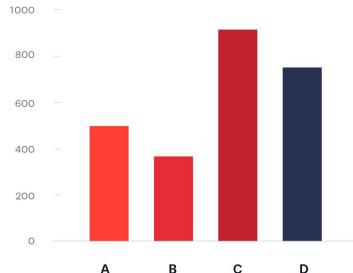
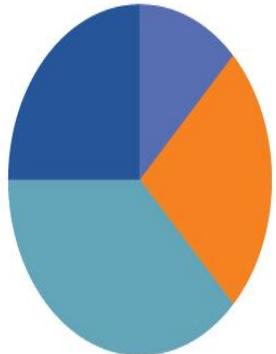


Chart vs Graph

- Les diagrammes et les graphiques sont des représentations visuelles de données.
- Les diagrammes sont utilisés pour afficher des données qui ne sont pas connectées, tandis que les graphiques affichent des données qui sont connectées.
- Travailler avec des graphiques ne signifie pas nécessairement visualiser.

What is a graph?

Graph Definition

Les graphes sont partout :

- Les graphes sont utilisés dans nos vies quotidiennes, des paquets de données sur Internet aux cartes pour expliquer des itinéraires.
- Lorsqu'on modélise des objets et leurs relations, nous utilisons des bulles pour représenter les objets et des flèches pour indiquer leurs interactions.
- Les graphes simplifient la découverte d'informations et facilitent la modélisation des relations, ce qui permet de mieux comprendre les systèmes complexes.

Définition : Un graphe est une structure mathématique composée d'objets (nœuds/vertices) reliés par des liens (relations/edges).

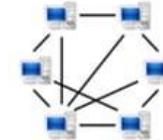
Exemple :

- Les entités (nœuds) comme Alice, Bob, et Londres.
- Les relations entre les nœuds comme "Alice est mariée à Bob" (relation non orientée) et "Alice vit à Londres" (relation orientée).

Représentation :

- Les graphes peuvent être orientés (directionnel) ou non-orientés selon la sémantique des relations.

a) Computer network



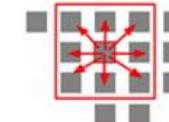
b) Social network



c) Road network

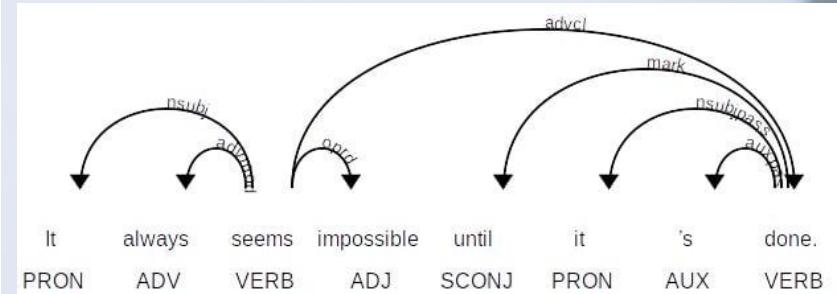
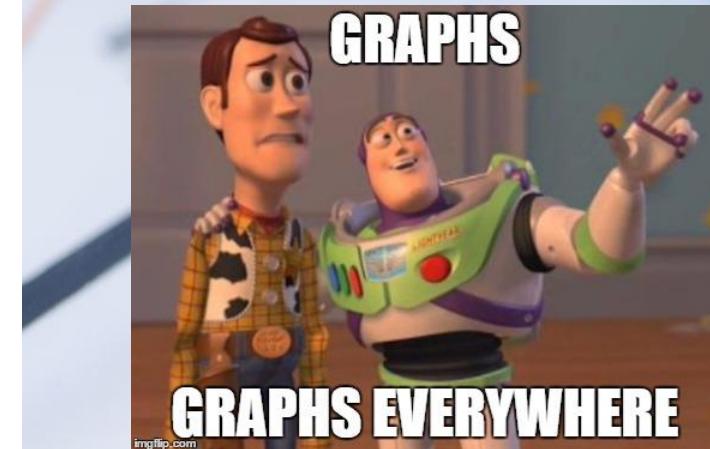


d) Image as a graph



Objects as Graphs

- Séries temporelles : Chaque observation est reliée à la suivante
- Images : Chaque pixel est relié à ses huit voisins
- Textes : Ici, chaque mot est relié à ses mots environnants ou à une cartographie plus complexe, selon sa signification

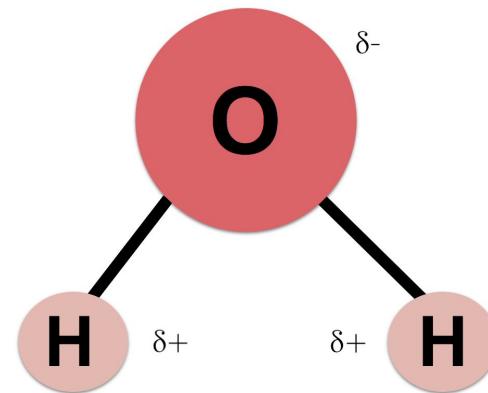


Anything can be a graph - do you have examples too?

The Internet



Water molecule



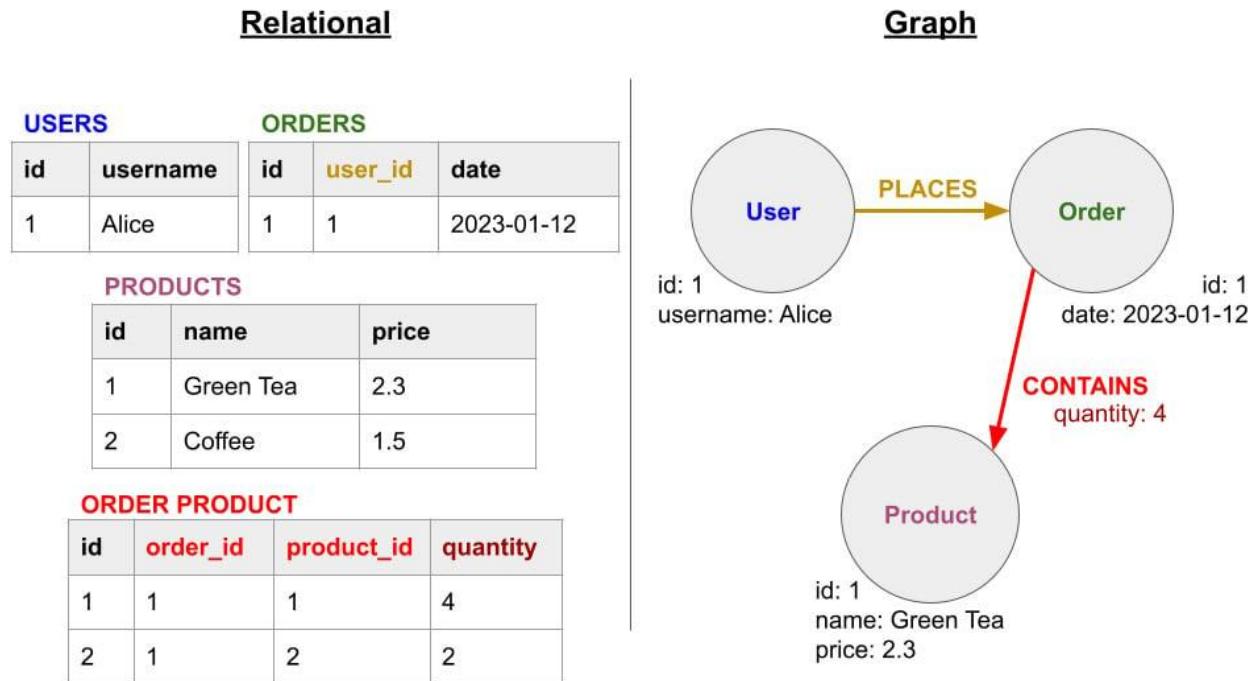
What is a Graph Database?

- Une base de données graphique stocke les données dans des nœuds et des arêtes pour modéliser les relations entre eux
- Les données sont enregistrées dans des nœuds et peuvent être connectées via des arêtes
- Une base de données graphique est utilisée pour modéliser des relations complexes entre les données
- Contrairement à une base de données relationnelle, une base de données graphique est entièrement structurée autour des relations de données. Les bases de données graphiques traitent les relations non pas comme une structure de schéma mais comme des données, comme d'autres valeurs

Pourquoi utiliser une base de données en graphe ?

- Performance : Les requêtes sont proportionnelles au nombre de nœuds visités plutôt qu'à la quantité totale de données, ce qui maintient des performances constantes.
- Flexibilité : Les modèles peuvent être modifiés de manière incrémentale sans affecter les requêtes existantes.
- Conception simplifiée : La modélisation sur tableau blanc est directement transposable en base de données, sans conversion en tables relationnelles.
-

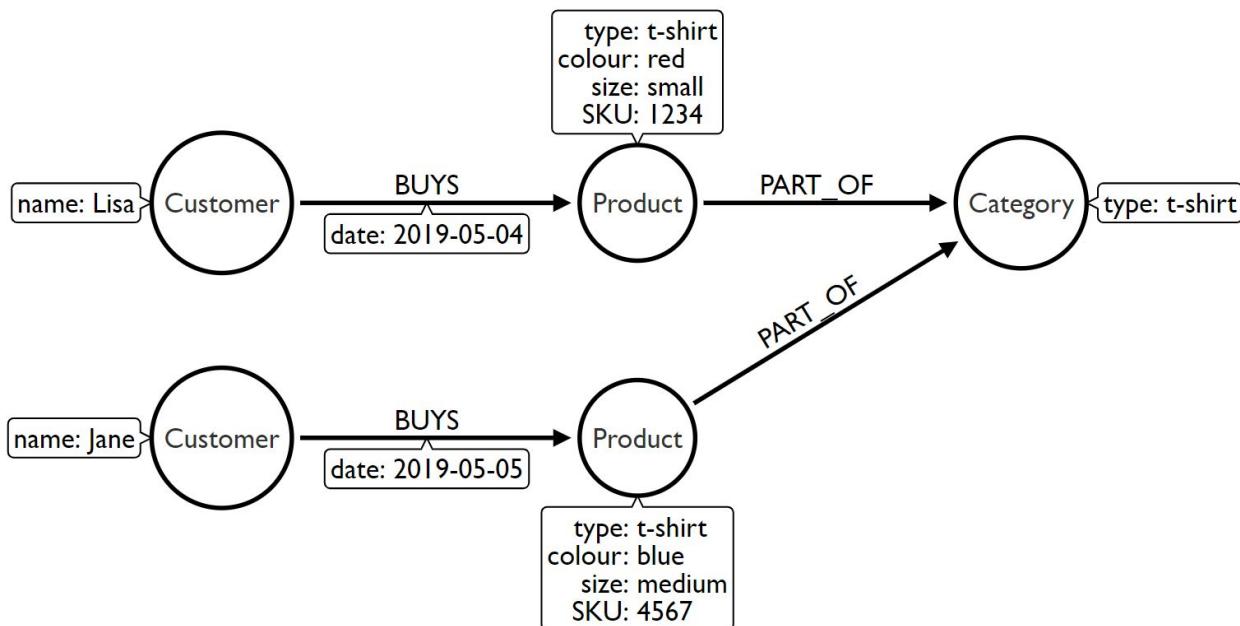
Modéliser vos données sous forme de graphique



What are good graph scenarios?

Identifying good graph scenarios

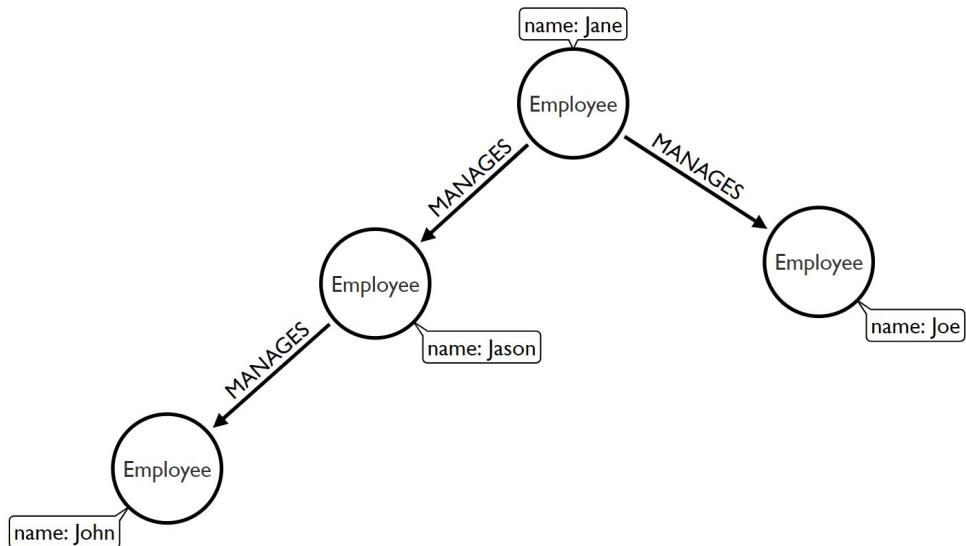
Scenario 1: Does our problem involve understanding relationships between entities?



- Recommendations
- Fraud detection
- Finding duplicates
- Data lineage
- Social Networks

Identifying good graph scenarios

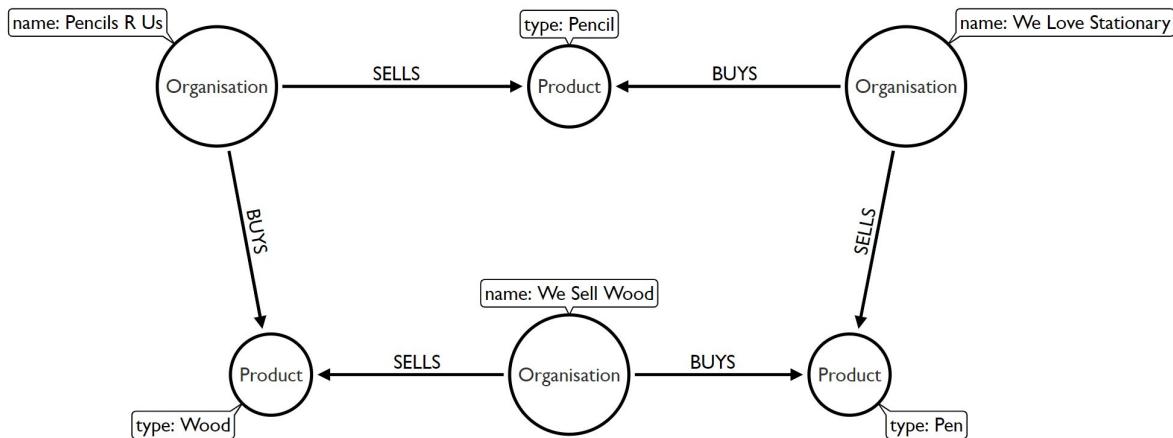
Scenario 2: Does the problem involve a lot of self-referencing to the same type of entity?



- Organisational hierarchies
- Access management
- Social influencers
- Friends of friends

Identifying good graph scenarios

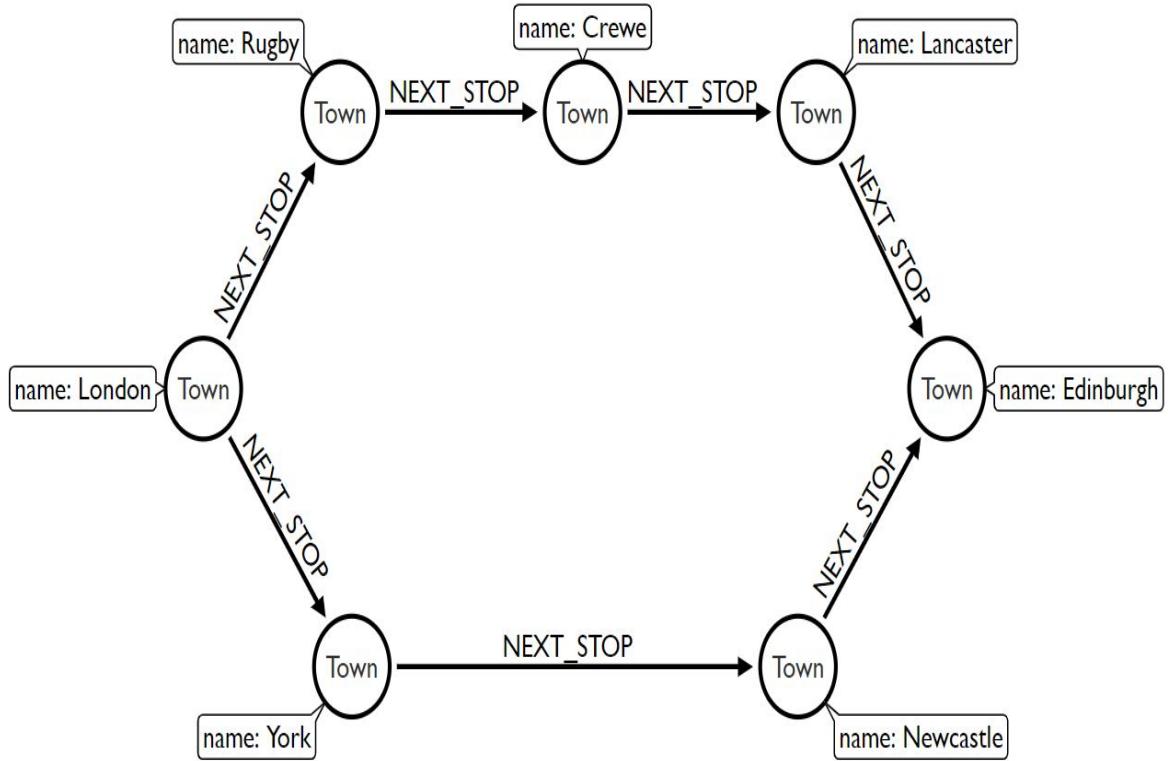
Scenario 3: Does the problem explore relationships of varying or unknown depth?



- Supply chain visibility
- Bill of Materials
- Network management
- Routing

Identifying good graph scenarios

Scenario 4: Does our problem involve discovering lots of different routes or paths?



- Logistics and routing
- Infrastructure management
- Dependency tracing

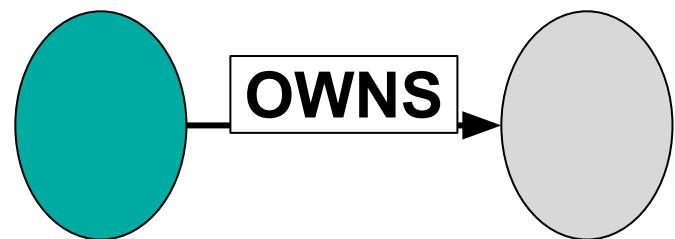
Concepts clés

- Nœuds : Entités du modèle (ex : Alice, Londres).
- Labels : Définissent le rôle d'un nœud dans un domaine.
- Relations : Connexions orientées et sémantiquement pertinentes entre les nœuds.
- Propriétés : Paires clé-valeur associées aux nœuds ou relations (ex : nom, depuis).

Property graph database

Node (Vertex)

Relationship (Edge)



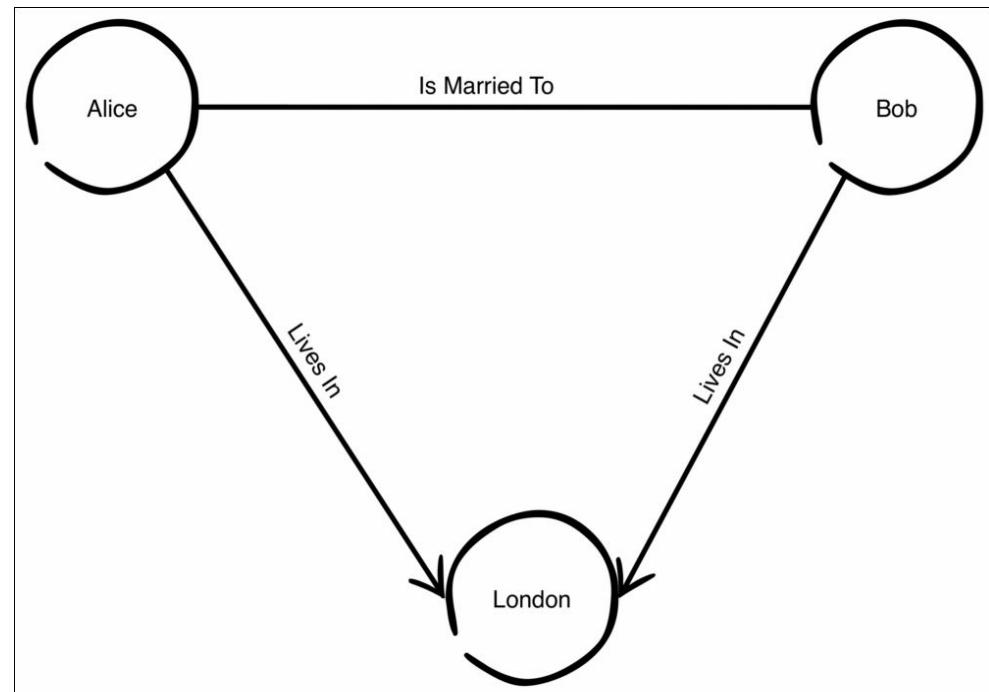
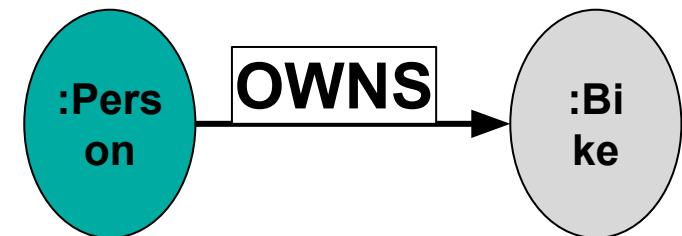
Property graph database

Node (Vertex)

Relationship (Edge)

Label

- Define node role
(optional)



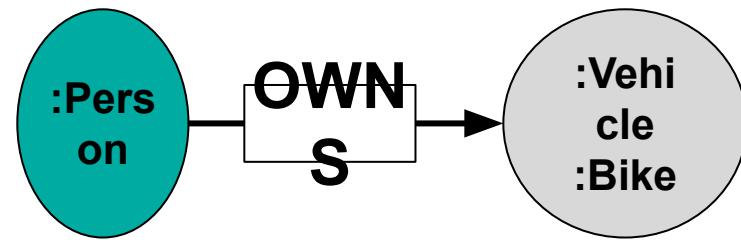
Property graph database

Node (Vertex)

Relationship (Edge)

Label

- Define node role
(optional)
- Can have more than
one



Property graph database

Node (Vertex)

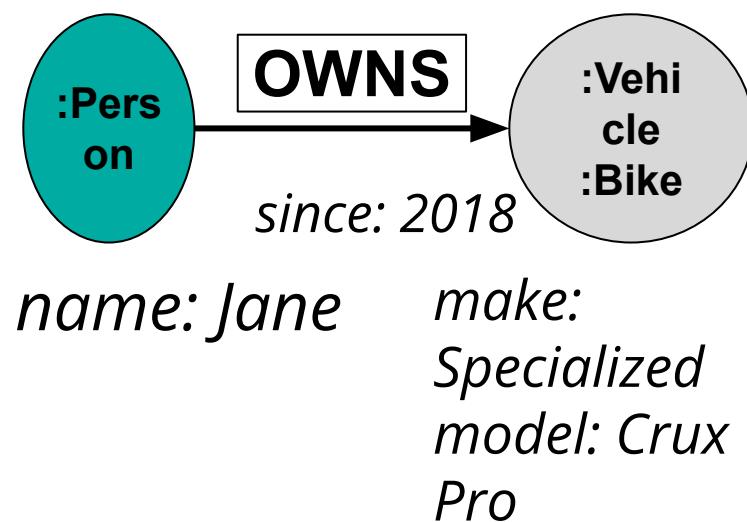
Relationship (Edge)

Label

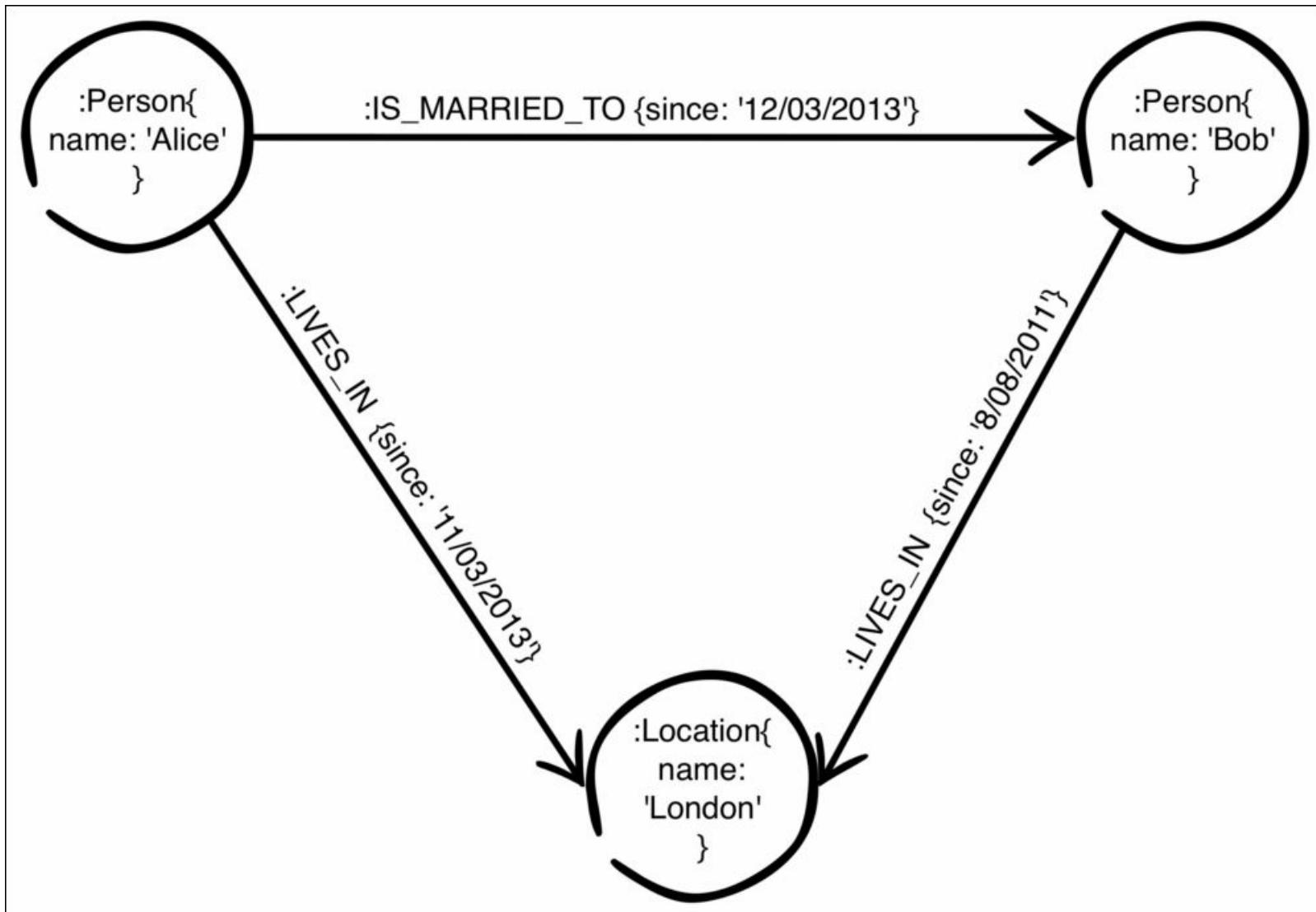
- Define node role (optional)
- Can have more than one

Properties

- Enrich a node or relationship
- No need for nulls!



Property graph database



Introduction à Neo4j

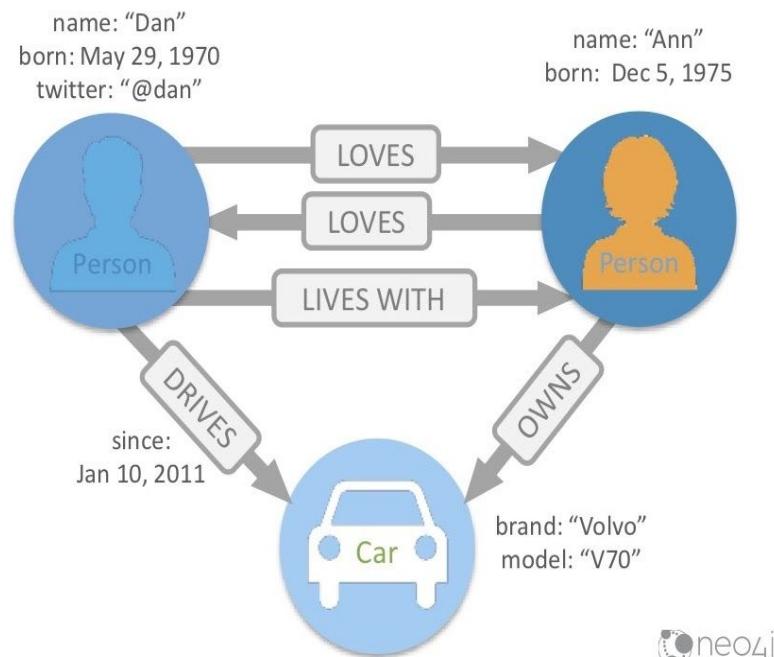


- Neo4j est un **système de gestion de bases de données graphiques** doté de capacités de stockage et de traitement de graphiques natifs.
- Il dispose d'un modèle de données flexible et prend en charge les requêtes complexes. **Cypher est un langage de requête graphique déclaratif pris en charge par Neo4j**. Cypher prend en charge les mots-clés de requête de type SQL qui peuvent fonctionner sur une base de données graphique complexe comportant des millions de nœuds.
- Stockage natif : Neo4j utilise un stockage de graphiques natif, ce qui lui permet de traiter les requêtes plus rapidement avec l'adjacence sans index (Index-free adjacency).
- Langage Cypher : Un **langage de requêtes déclaratif** et performant, conçu spécifiquement pour les bases de données en graphique.
- Communauté et support : Neo4j est une base de données mature avec une grande communauté et est utilisée par de nombreuses entreprises comme Walmart, Cisco, et eBay.

Introduction à Neo4j



- **Nodes:**
 - Represent things, nouns, or objects (circles).
 - Can have labels for classification.
 - Example: Two Person nodes and one Car node.
- **Properties:**
 - Name-value pairs on nodes and relationships.
 - Example:
 - Dan: name (Dan), born (May 29th, 1970), twitter (@dan).
 - Ann: name (Ann), born (December 5th, 1975).
- **Graph Database Flexibility:**
 - No need for every node to have the same properties.
 - Unlike relational databases, no null values required for missing properties.
- **Relationships:**
 - Combine nodes and properties.
 - Show how nodes are related.
 - Example:
 - Dan LOVES Ann.
 - Ann LOVES Dan.
 - Dan LIVES WITH Ann.
 - Relationships have directions, which can be important or less so.



Use Case: Automotive Data

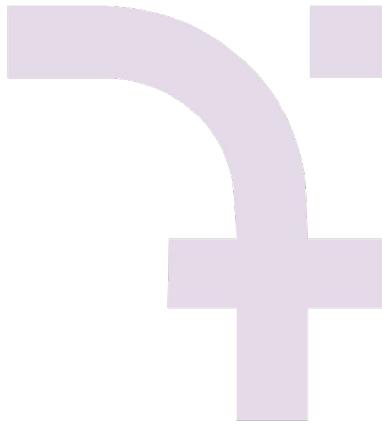
- **Organizational Data:** Internal company data including documentation, processes, facilities, organizational hierarchy, KPIs, reports, systems, databases, and IT infrastructure.
- **Product Data:** Information on products, including documentation, customer contact processes, product details, and product hierarchy.
- **Bills of Materials (BOM):** Detailed data on components and their assembly, forming a graph-like structure, used to track materials in products.
- **Customer Data:** Data about customers, often indirect through dealers, including personal or corporate information and customer relationships.
- **Third-Party and Event Data:** Data from partners, social media, market data, sensor/telematics data, warranty claims, customer contacts, and supply chain information including logistics and inventory data.



Some Examples of Typical Automotive Data



9



Introduction to Cypher Query Language



Objectives

- Creating your first node and relationships using Cypher
- Querying nodes and relationships using Cypher
- Deleting data from Neo4j using the Cypher query
- The Boolean operators with Cypher
- Changing order of result with Cypher
- Limiting and skipping results with Cypher

Creating your first node and relationship using Cypher



```
CREATE (JFK:Airport { name:'JF Kennedy Airport', city:'New York' }),  
(AUS:Airport { name:'Austin-Bergstrom International', city:'Austin' })  
  
CREATE (flight1:Flight { flight_number:'BG45', month:'August' })
```

The **CREATE** keyword is used
for creating nodes and
relationships.

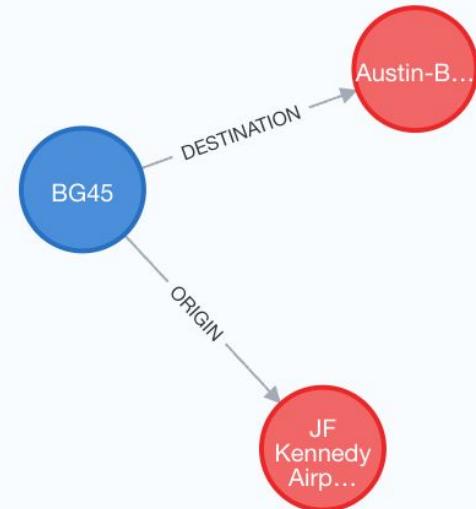


Creating your first node and relationship using Cypher

```
● ● ●  
MATCH (JFK:Airport { name:'JF Kennedy Airport' })  
MATCH (AUS:Airport { name:'Austin-Bergstrom International' })  
MATCH (flight1:Flight { flight_number:'BG45' })  
CREATE (flight1)-[:ORIGIN]->(JFK),(flight1)-[:DESTINATION]->(AUS)
```

The relationships are created between the two nodes, as specified in the following code:

```
CREATE (flight1)-[:ORIGIN]->(JFK),(flight1)-[:DESTINATION]->(AUS)
```



Querying nodes and relationships using Cypher

- Counting all nodes that exist in the graph
- Counting all relationships that exist in the graph
- Finding all distinct labels that exist in the graph
- Finding all distinct relationship types that exist in the graph
- Finding all nodes that are disjoint, which means that they do not have any relationship with the other nodes
- Finding all nodes that have some specific property
- Finding all nodes that have some specific relationship, regardless of the direction

- **CREATE:** This clause is used to create nodes and relationships.
- **MATCH:** This clause matches a certain set of nodes and relationships following the patterns specified.

- **RETURN:** This decides which part of the created data should be returned. It can be used to return nodes, relationships, or even individual properties.

- **DISTINCT:** The distinct clause is used to remove duplicates from the result.
- **COUNT:** The count clause is used to aggregate the result.

Counting all nodes that exist in the graph



```
MATCH (n) RETURN COUNT(n)
```

Counting all relationships that exist in the graph



```
MATCH (n)-[r]->(m)  
RETURN COUNT(r);
```

Finding all distinct labels that exist in the graph:



```
MATCH (n) RETURN DISTINCT LABELS(n)
```

Finding all distinct relationship types that exist in the graph



```
MATCH (n)-[r]-( ) RETURN DISTINCT TYPE(r);
```

Finding all nodes that are disjoint, which means that they do not have any relationship with the other nodes



```
MATCH (n) WHERE NOT (n)-[]-( ) RETURN n
```

Finding all nodes that have some specific property:

```
MATCH (n)
WHERE (n.name) IS NOT NULL
RETURN n;
```

Finding all nodes that have some specific relationship, regardless of the direction



```
MATCH (n)-[:ORIGIN]-() RETURN DISTINCT n
```

Deleting data from Neo4j using the Cypher query

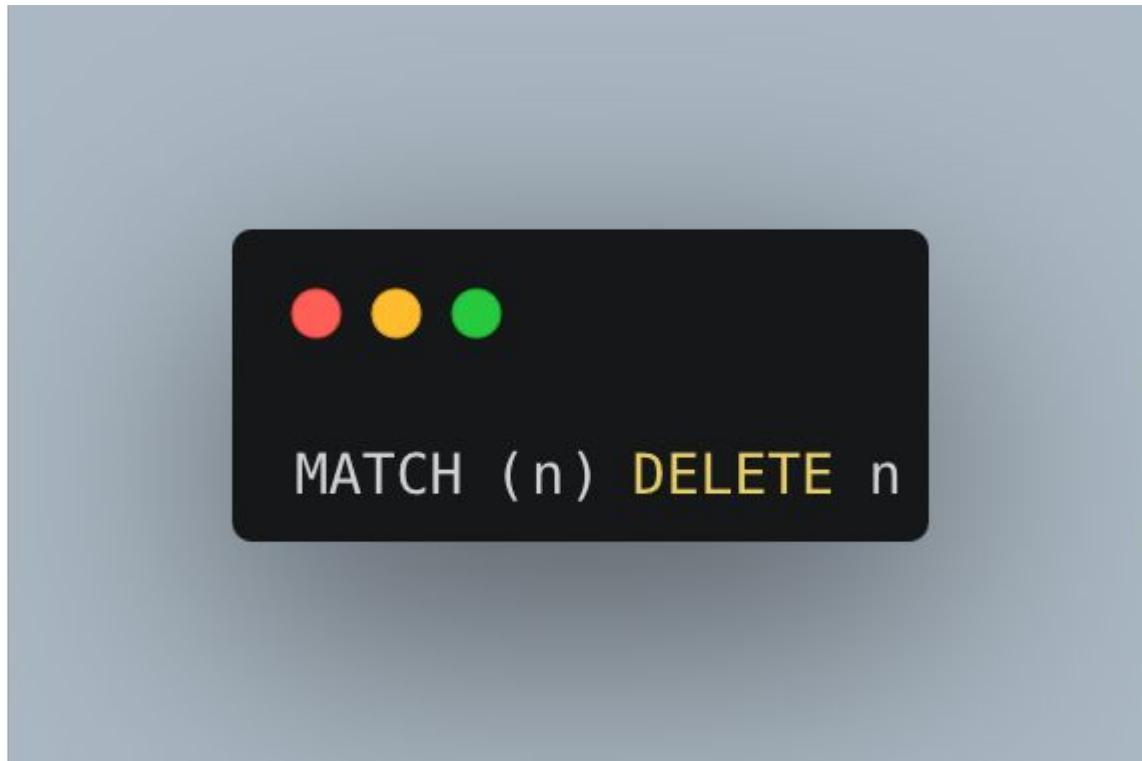
- Deleting all relationships from the Neo4j graph
- Deleting all nodes from the Neo4j graph
- Deleting all nodes from the Neo4j graph matching a condition
- Deleting all relationships of a particular type
- Deleting a property/properties from a particular node/nodes
- Removing a label/labels from a particular node/nodes

- **DELETE:** This is used to delete nodes or relationships based on a matched pattern
- **REMOVE:** This is used to delete a particular property or label from the nodes or relationships

Deleting all relationships from the Neo4j graph

```
MATCH (n)-[r]-( ) DELETE r
```

Deleting all nodes from the Neo4j graph



Deleting all nodes from the Neo4j graph matching a condition



```
MATCH (n) WHERE n.city = "Atlanta" DELETE n  
# You have to delete all relationships from that node before deleting that node
```

Deleting all relationships of a particular type



```
MATCH (n)-[r:ORIGIN]-() DELETE r
```

Deleting a property/properties from a particular node/nodes



```
MATCH (n) REMOVE n.city
```

Removing a label/labels from a particular node/nodes

```
MATCH (n) REMOVE n:Airport
```

Boolean operators with Cypher

Cypher supports the Boolean operators AND, OR, NOT, and XOR, which are very useful to describe more than one condition to be checked simultaneously.

```
MATCH (o)<-[ :`ORIGIN` ]-(f)-[ :`DESTINATION` ]->(d) where o.city = "Atlanta" AND d.city = "Dallas/Fort  
Worth" return o,f,d

MATCH (o)<-[:ORIGIN]-(f) where o.city = "Atlanta" OR o.city = "Dallas/Fort Worth" return o,f

MATCH (o)<-[ :`ORIGIN` ]-(f) WHERE NOT o.city = "Atlanta" RETURN o

MATCH (o)<-[ :`ORIGIN` ]-(f)-[ :`DELAYED_BY` ]-(r) where (o.city = "Atlanta" OR o.city = "Dallas/Fort  
Worth")AND r.name = "Late Aircraft" return o,f
```

Changing the order of results with Cypher

ORDER BY on numerical value

```
MATCH (f)-[r:`DELAYED_BY`]->(x) RETURN f.flight_number,r.time,x.name ORDER BY r.time DESC
```

ORDER BY on unicode strings

```
MATCH (f)-[r:ORIGIN]->(x) RETURN DISTINCT x.name ORDER BY x.name
```

ORDER BY on multiple properties

```
MATCH (f)-[r:ORIGIN]->(x) RETURN DISTINCT x.name,x.city ORDER BY x.city,x.name
```

Limiting and skipping results with Cypher

Limiting the results: Let's get the delay times of the topmost three delayed flights, along with the flight number and reason of delay:

```
MATCH (f)-[r:DELAYED_BY]->(x) RETURN f.flight_number,r.time,x.name ORDER BY r.time DESC LIMIT 3
```

Skipping the results: Let's get the delay times of flights. We will skip the first three results and get the next ones

```
MATCH (f)-[r:DELAYED_BY]->(x) RETURN f.flight_number,r.time,x.name ORDER BY r.time DESC SKIP 3
```

Skipping with limits: Let's get the delay time of a flight that is on the third rank with respect to the flight delay timing:

```
MATCH (f)-[r:DELAYED_BY]->(x) RETURN f.flight_number,r.time,x.name ORDER BY r.time DESC SKIP 2 LIMIT 1
```

Activity 1: Robotics Pathfinding in a Factory

In this example, let's consider a robotic system navigating through a factory floor. The robot needs to find the **shortest path** between two points: **Point A** and **Point B**, which represent different locations in the factory. Each point is connected to other points (like corridors or workstations) through different paths with varying distances (representing time, obstacles, or ease of traversal).

Nodes:

- Points in the factory are represented by nodes (e.g., **A**, **B**, **C**, **D**, etc.).
- The relationships between these points represent **paths** that the robot can take, each with an associated **distance**.

Relationships:

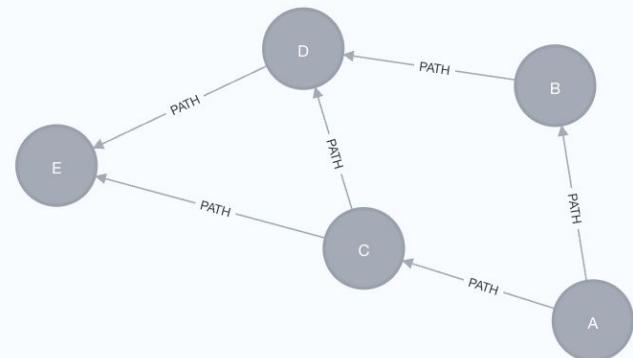
- The **relationships** between points have a **distance** property that indicates how far the robot has to travel between two connected points.

Q: The robot needs to move from Point A to Point E. Find the most efficient route and the distance.



```
// Create Points (Nodes)
CREATE (a:Point {name: "A"}),
       (b:Point {name: "B"}),
       (c:Point {name: "C"}),
       (d:Point {name: "D"}),
       (e:Point {name: "E"})

// Create Relationships (Paths with Distances)
CREATE (a)-[:PATH {distance: 5}]->(b),
       (a)-[:PATH {distance: 10}]->(c),
       (b)-[:PATH {distance: 15}]->(d),
       (c)-[:PATH {distance: 5}]->(d),
       (d)-[:PATH {distance: 10}]->(e),
       (c)-[:PATH {distance: 20}]->(e);
```



Activity 2: Diagramme d'utilisation

The diagram shows a system ("Truc") interacting with different entities and fulfilling various functionalities (FP1, FC1, etc.). Create a graph modeling for this diagram.

Diagramme d'utilisation :

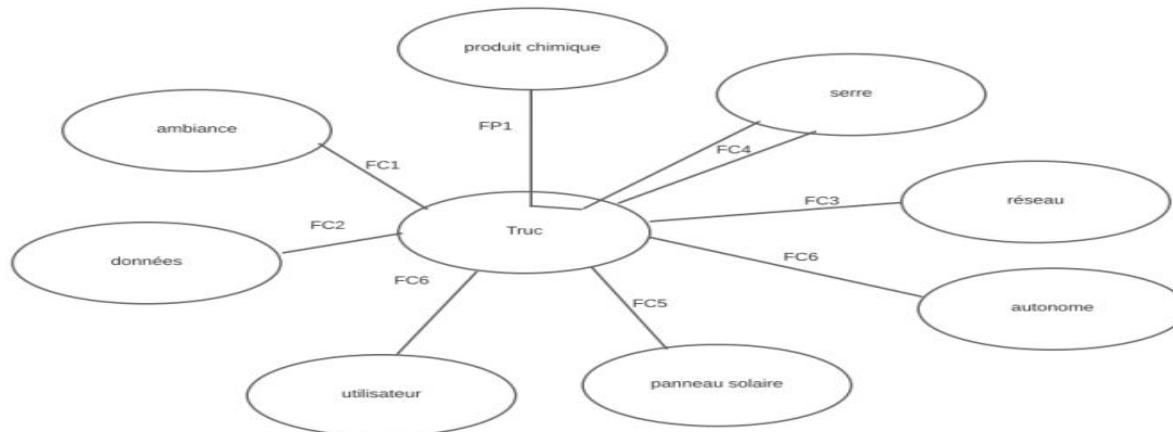
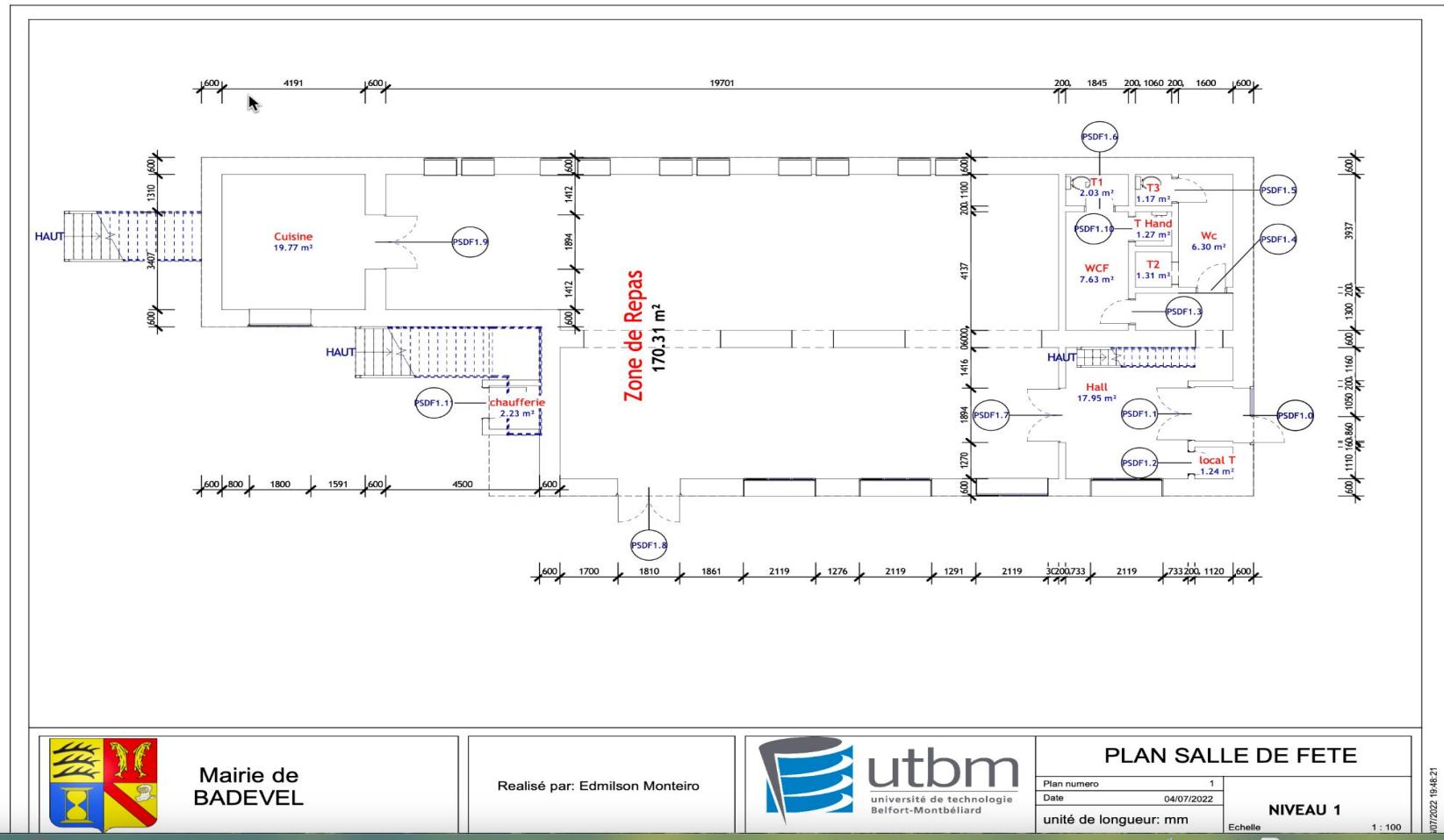


Figure 2, diagramme pieuvre Utilisation

FP1	Permettre à la serre d'être approvisionné en produit chimique nécessaire à son fonctionnement optimal
FC1	Résister au milieu ambiant
FC2	Recueillir les données
FC3	Communiquer avec le réseau
FC4	S'adapter à la serre
FC5	Être alimenté par un panneau solaire
FC6	Être contrôlable par l'utilisateur
FC7	Être partiellement autonome

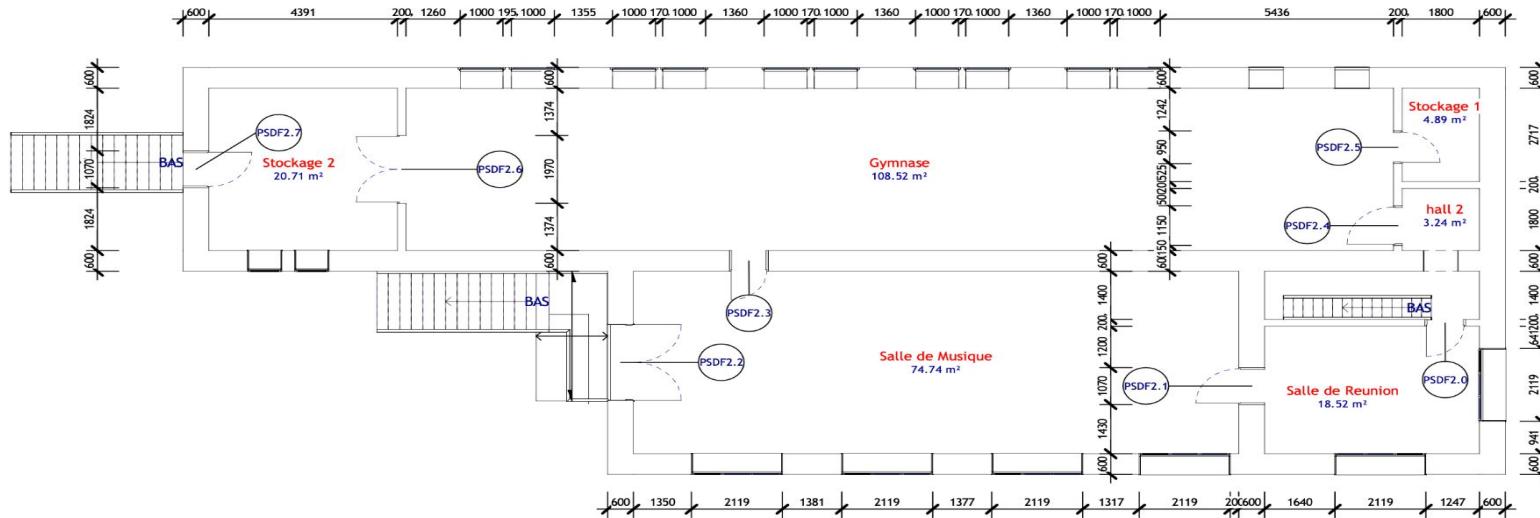
Activity 3: Modeling SALLE DE FETE - 1 -

Create a graph model for the “salle de fete” a Badevel



Activity 3: Modeling SALLE DE FETE - 2 -

Create a graph model for the “salle de fete” a Badevel



Mairie de
BADEVEL

Realisé par: Edmilson Monteiro



PLAN SALLE DE FETE

Plan numero 1
Date 04/07/2022

unité de longueur: mm

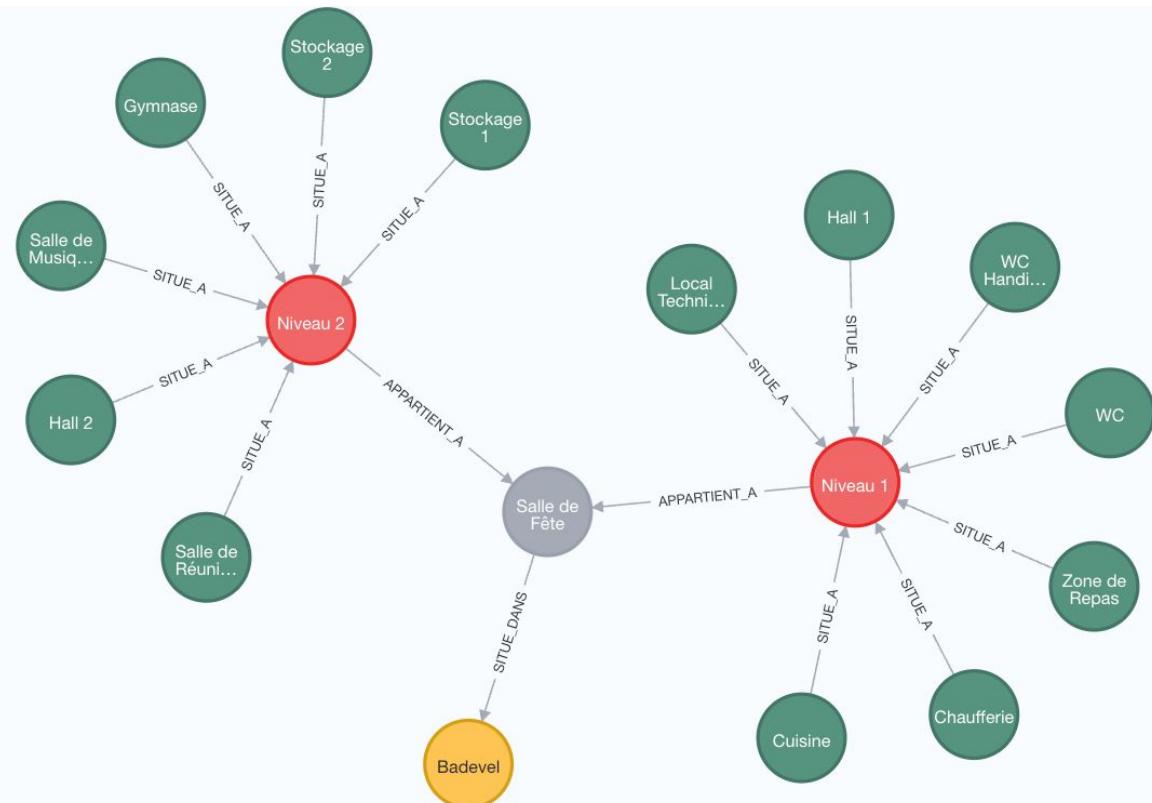
Niveau 2

Echelle 1 : 100

16/07/2022 19:48:22

Activity 3: Modeling SALLE DE FETE - 3 -

Expected result:



Overview
Node labels
* (17) Etage (2) Piece (13) Batiment (1)
Region (1)
Relationship types
* (16) SITUE_A (13) SITUE_DANS (1)
APPARTIENT_A (2)

Displaying 17 nodes, 0 relationships.