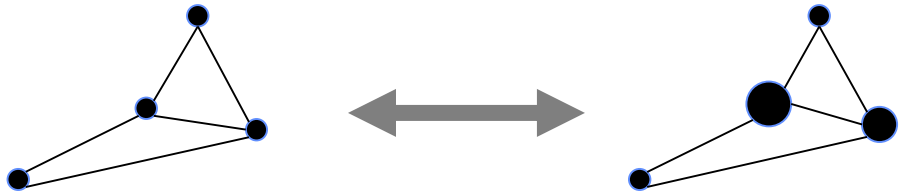
- **Bases de données non structurées**
  - **1.1. Pourquoi NoSQL ?**
    - » Limitations du modèle relationnel face aux contraintes actuelles :
      - ***Big User*** : grand nombre d'utilisateurs simultanés.
      - ***Big Data*** : masse de données considérable, généralement non structurées, collectées par le Web et les logiciels interactifs.
      - ***Cloud Computing*** : exploitation de la puissance de calcul ou de stockage de serveurs informatiques distants par l'intermédiaire d'un réseau, généralement Internet.
      - Lourdeur ou inadéquation du langage de requête **SQL** vis-à-vis des contraintes précédentes.
    - » Fin des années 2000, SGBD **propres** aux géants de l'Internet (*GAFAM*, *BATX et NATU)* s'écartant du paradigme relationnel :
      - *Exemples : BigTable (Google); Cassandra (Facebook), DynamoDB (Amazon), Voldemort (LinkedIn), etc.*
    - » Mouvement regroupant plusieurs dizaines (centaines) de solutions propriétaires ou libres.

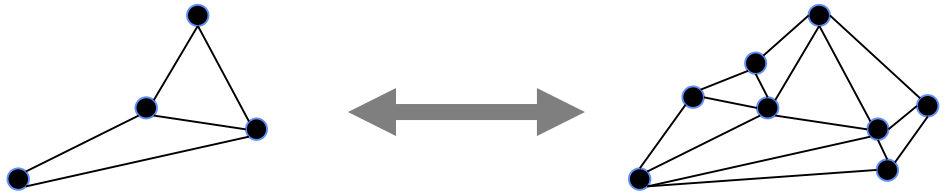
– 1.2. Évolutivité des systèmes (*scalability*)

» Capacité d'un système à (dé-)croître en fonction de ses besoins :

- **Évolution verticale** : augmenter/diminuer les capacités d'un ou plusieurs nœuds du système.



- **Évolution horizontale** : ajout/suppression de nœuds dans le système.



» Les systèmes relationnels gèrent mal le partitionnement et, d'une manière générale, les systèmes distribués. Ils évoluent essentiellement de façon **verticale**.

» Les systèmes NoSQL sont, à l'inverse, conçus pour évoluer **horizontalement**, face à un nombre d'utilisateurs de services pouvant croître rapidement et exponentiellement.

- *N.B. Particulièrement efficace dans les environnements « cloud » où les ressources sont réparties de manière dynamique entre les serveurs.*

– 1.3. Principes des systèmes NoSQL

» Rejet partiel des propriétés **ACID** des systèmes relationnels:

Ex. transactions bancaires

- *(A)tomicity* : atomicité des transactions (effectuée ou annulée).
- ***(C)onsistency*** : cohérence de la base avant et après transaction.
- *(I)solation* : non interférence de transactions simultanées.
- *(D)urability* : état de la base enregistré après la transaction.

» Choix d'une **tolérance au partitionnement** assurant une **disponibilité** permanente, au détriment éventuellement de la cohérence.

- En cas de problème (*sur un serveur par ex.*), il est jugé préférable de maintenir un service (*au départ d'un autre serveur / nœud du réseau*), puis de synchroniser les données entre les nœuds lorsque le problème sera levé, plutôt que d'interrompre le service pour conserver à tout prix l'intégrité des données.

» Propriétés des systèmes NoSQL : **BASE**

Ex. transactions de commerce en ligne

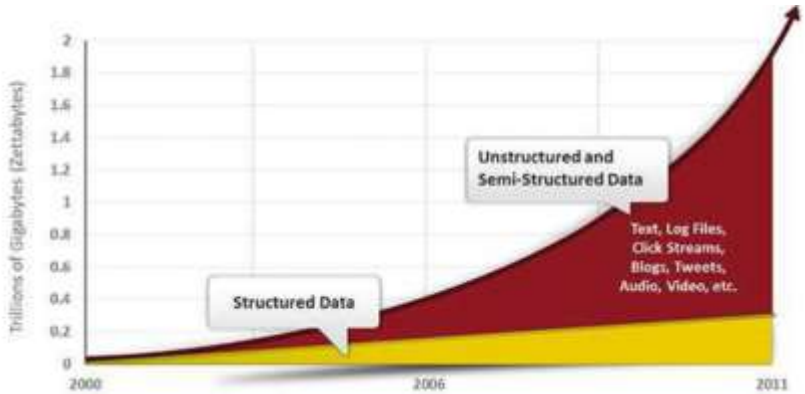
- *(BA)sic Availability* : BD toujours disponible.
- *(S)oft State* : non synchronisme systématique des réplicas à un temps donné.
- *(E)ventual Consistency* : synchronisation et cohérence finales des différents nœuds.

– 1.4. Schéma flexible

» Les bases relationnelles reposent sur un schéma interne (**MCD**), relativement rigide, en particulier dans les environnements distribués (modifications de schémas difficiles, longues et non sans risques).

» Les données du *Big Data* sont au contraire **non ou semi-structurées** (tweets, blogs, multimédia…). Elles sont aussi devenues largement majoritaires.

» Les systèmes NoSQL permettent de manipuler des données non structurées, très simplement structurées ou structurées selon des schémas flexibles.



– 1.5. Avantages de NoSQL

De façon générale :

- Absence de la notion de clé.

» **Dé-normalisation**

- La redondance est possible et parfois même encouragée.

» **Plus de jointures**

- Facilite les relations 1…n entre les enregistrements
  - *ex. (1 parcelle a plusieurs propriétaires) réclamerait une table supplémentaire en relationnel.*

– 1.6. Désavantages de NoSQL

» **Incohérence** momentanée des données (propriétés BASE), excluant certaines applications où la sécurité est critique.

» **Vulnérabilité** aux attaques diverses, par faiblesse des processus de sécurité en général (cryptage, authentification, etc.).

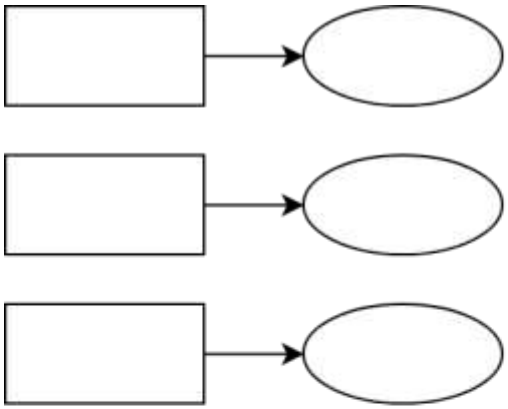
» Absence fréquente d'un langage complet tel que SQL, totalement indépendant des données :

- N'admet que des opérations élémentaires (*CRUD: Create,Read,Update,Delete*).
- Obligation de modéliser les données en fonction des applications.
- Très grande complexité de réaliser (programmer) certaines requêtes.

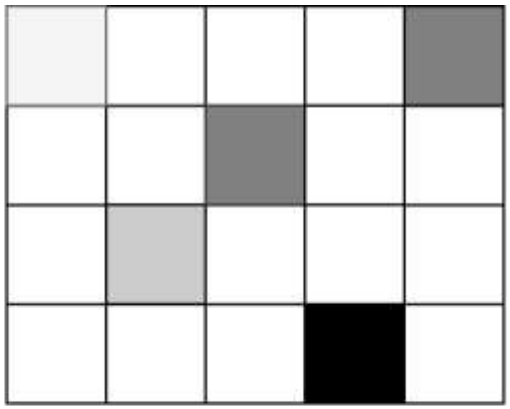
» Insuffisance des **outils** : environnement beaucoup moins développé que pour les SGBD relationnels (AGL, ETL, optimisation, backup, analyse, sécurité…) et faiblesse de la documentation (nombreuses solutions OS).

» Manque d'expertise : nouveauté, hétérogénéité, évolution rapide dans toutes les directions, manque de recul et d'évaluation, tout cela s'ajoutant à la faible documentation.

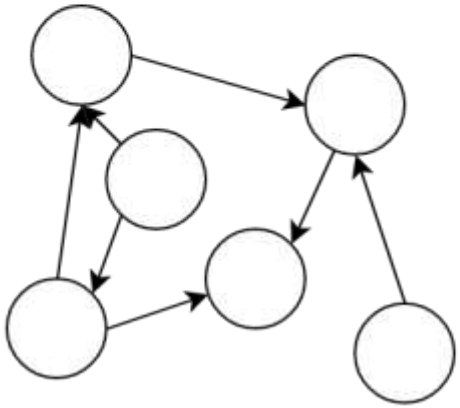
- **2. Typologie des systèmes NoSQL**



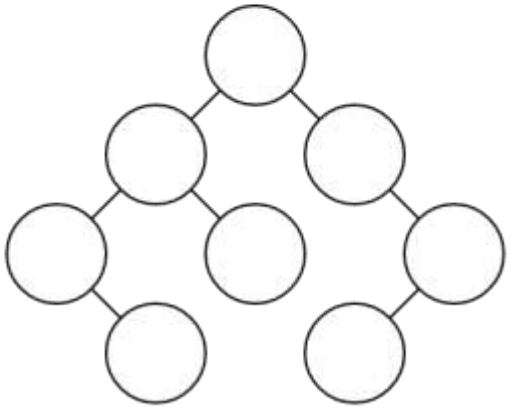
Key-Value



Wide-Columns

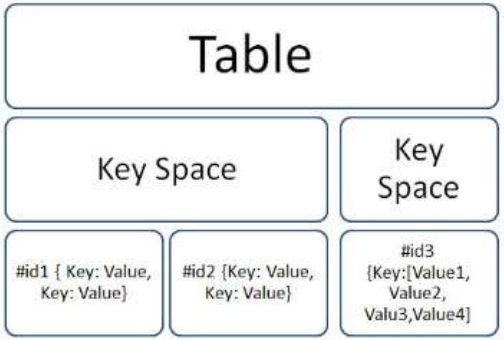


Graph-oriented



Document-oriented

- **2. Typologie des systèmes NoSQL**
  - **2.1. Clé-valeur (*Key-Value Store*)**
    - » Format de donnée constitué d'une clé (textuelle ou autre) et d'une valeur (textuelle, BLOB, JSON…), les requêtes sur les valeurs s'effectuant sur base de la clé (*Get[key], Put[key], MultiGet[key_1,…,key_n], Delete[key]*)
    - » Les couples clés-valeurs peuvent être regroupés dans les colonnes d'une table (*Key Space*).
    - » Solution simple (pas de schéma) et très rapide convenant au stockage de très longues listes (produits, profils d'utilisateurs…), mais très limitée vis-à-vis des types de requêtes possibles.
      - *Ex. Utilisée, entre autres systèmes, par DynamoDB d'Amazon, Voldemort (LinkedIn).*



*Rmq. **KVP** (keyword-value-pair) est une expression générique. Le principe est largement exploité pour l'encodage des paramètres dans l'appel de procédures, par exemple celui des méthodes HTTP.*

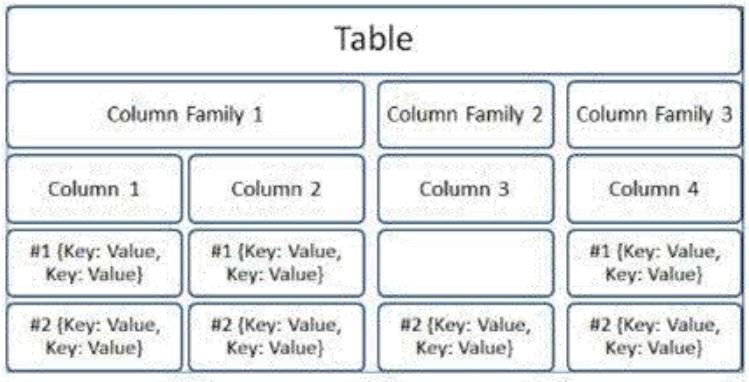
– 2.2. Clé-colonne (*Column Store*)

» *Remarque préliminaire sur les **bases de données** « **orientées colonnes** »*

- *Une table peut être organisée en fonction des colonnes et non des lignes, tout en restant conçue et implémentée, en première analyse, comme une table du modèle relationnel.*
- *Le « moteur » de recherche fonctionne cependant de façon très différente :*
  - *Les colonnes sont traitées individuellement : la table sérialise les valeurs d'une colonne ensemble, puis les valeurs de la colonne suivante, etc.*
- *Cette technologie réduit fortement les temps d'agrégations des données et favorise le compactage (espace disque moins important et temps d'accès plus court).*
- *Elle est largement utilisée dans le domaine décisionnel (Business Intelligence), sur de grandes quantités de données (sur des **entrepôts** de données par exemple).*
- *Elle ne doit pas être confondue avec la technique « clé-colonne » du NoSQL, même si certains principes sont similaires.*

» La technique « **clé-colonne** » du **NoSQL** peut être considérée comme une évolution multidimensionnelle du modèle précédent, *key-value*, où un identifiant est associé à une colonne de clés-valeurs.

- Les données (clés-valeurs) sont disposées en colonnes.
- Les données peuvent être réparties sur plusieurs serveurs.
- La structure ***ColumnFamily*** permet en option de regrouper plusieurs colonnes (*Super/Sub-Columns*).
- De nouvelles colonnes peuvent être ajoutées à tout moment.
- Le type de données n'est pas forcé (identique pour une même clé pour faciliter les opérations statistiques sur les valeurs).
- Les « rangées » de la table ne sont pas homogènes (nombre variable de colonnes, pas de valeurs, types différents…).



» *Ex. BigTable (Google), HBase et Cassandra (Facebook) sont des bases NoSQL relevant de ce type (avec des différences entre elles).*

– 2.3. Orienté document (*Document Store*)

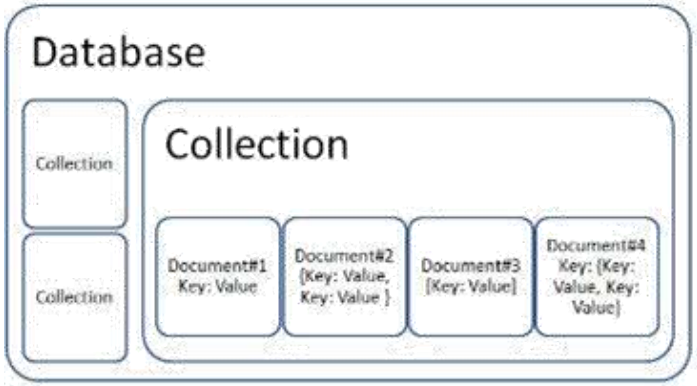
» Dédié tout particulièrement à l'enregistrement des données **non structurées**.

» Collection de clés-valeurs où, à la différence des modèles précédents, les valeurs constituent des « **documents** » qui :

- Sont identifiés par une clé unique (string, URI, path…).
- Sont encodés dans des langages de description de données (XML, JSON, BSON, YAML) ou des formats binaires courants (PDF, MS Office…).

Yet Another Markup Language

- Sont accompagnés de **métadonnées** qui permettent d'interroger le **contenu**.
- Peuvent eux-mêmes contenir des documents selon une structure arborescente (***embedded*** *documents*).



» Les données ne sont pas encodées dans des tables, mais dans une collection de documents (**fichiers**) **indépendants** et **sans schémas**.

» Les langages Javascript, Python, etc. ou des API proposées par le système sont utilisés pour interroger les données.

» *Ex. MongoDB (Open Source) et Apache CouchDB sont des exemples de systèmes orientés documents.*

- **3. NoSQL et données géographiques**
  - **3.1. Conditions d'un SGBD spatial**
    - » Doit pouvoir stocker et gérer des données géo-spatiales :
      - **Hybrides** : géométrie [selon un modèle vectoriel ou maillé (primitive graphique = point / pixel)] + attributs.
      - Géométries **géo-référencées**.
      - Géométries **standardisées** selon un standard (*ex. Simple Features vectorielles de l'OGC, modèle ISO 19107*).
      - Accompagnées de **métadonnées** géographiques relatives au type de géométrie, au SRID, etc.
    - » Doit disposer d'un mécanisme d'**indexation spatiale**.
      - Par découpage de l'espace-utilisateur (*geo-hashing*).
      - Ou par regroupement des entités spatiales voisines.
    - » Doit pouvoir effectuer des **requêtes spatiales**
      - Exploitation des relations topologiques et proximales (sur base des positions), sauvegardées dans la base explicitement ou implicitement (raster par ex.) ou recalculées à la volée.

– 3.2. Utilisation de la paire « clé-valeur » en NoSQL

» Quel que soit le modèle, les données sont toujours véhiculées en NoSQL via la paire « clé-valeur », où :

- La clé peut être codée sous forme alphanumérique (*ex. identifiant numérique ou toponyme alphabétique*).
- La valeur peut être codée sous de multiples formes (variables selon le modèle) : textuelle, dans un langage de description de données, listes, BLOB, etc.

» Le stockage des données géo-spatiales via les paires « clés-valeurs » est donc tout-à-fait possible.

Clé	***Valeur***
‘’Liège’’	{Latitude:‘’50.633730’’,Longitude:’’5.467590’’,Population:’’195645’’}
‘’62063’’	

Image 8 bits/pixel dans un BLOB

– 3.3. Absence de structure et de métadonnées

» L'absence de schéma et de jointure facilite la construction de collections de données géo-spatiales et leur accès individuel.

- *Ex. : liste des coordonnées d'un nombre variable de points délimitant un polygone ou un multi-polygone.*

» Par contre, l'extraction de données précises (filtrées) pour les soumettre à un traitement devient très compliquée en l'absence de schéma, de métadonnées et d'un langage tel que SQL.

- *Ex. : calculer la superficie de l'intersection de 2 polygones si elle existe.*
- De telles opérations nécessitent le développement d'applications spécifiques dans un langage de programmation compatible avec la base NoSQL (association données-traitements, # SGBD relationnel !).

» Pour remédier à ces difficultés, plusieurs approches sont suivies :

Rôle de l'utilisateur

- Maintenir un MCD utilisant les standards (OGC…) et ne charger que des données (semi-)structurées dans la base NoSQL.
- Utiliser un système hybride SGBD-R + NoSQL.

Rôle du développeur du système NoSQL

- Fournir des API dédiées aux traitements sur des données géo-spatiales.
- Ajouter des métadonnées (paramétrables ou fixes) à l'environnement géographique (SRID, etc.).

– **3.4. Familles de bases NoSQL orientées Documents et orientées Graphes**

» Ces 2 familles offrent des avantages dans des cas spécifiques d'applications géographiques.

» Les bases **orientées documents** utilisent :

- Des métadonnées facilitant l'extraction des données depuis les documents.
- Des langages de description de données standardisés, extensibles aux données géographiques (XML → GML, JSON →GeoJSON).
- Paraissent bien adaptées à la gestion de données par projets : documents imbriqués comportant des données de ≠ natures et de taille variable.

» Les bases **orientées graphes :**

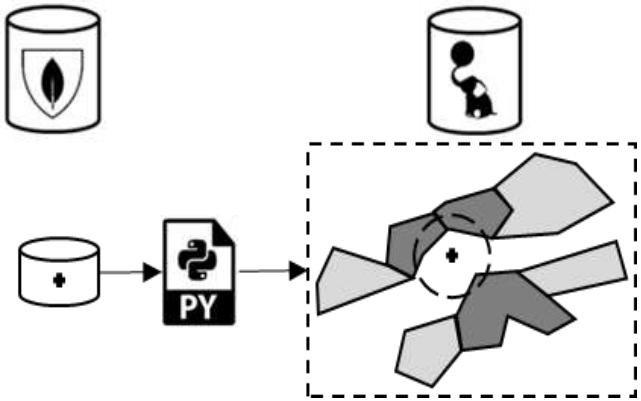
- Présentent une forte analogie avec le modèle « géo-relationnel » de données géographiques.
  - Les relations topologiques planaires devraient pouvoir être conservées et exploitées facilement.
- Paraissent par nature bien adaptées aux données géographiques de géométrie linéaire organisées en réseaux (hiérarchiques ou maillés).
  - *Ex. Hydrologie, distribution d'énergie et impétrants divers.*

- **4. MongoDB**
  - **4.1. Caractéristiques générales**
    - » Principal système NoSQL **orienté Documents** (O.S.); vaste communauté d'utilisateurs; en évolution constante.
    - » Dispose de nombreux « *drivers* » permettant son utilisation à travers toute sorte de langages de programmation (Python, PHP, JavaScript…)
    - » Utilise préférentiellement les données au format **JSON** (*JavaScript Object Notation*) et ses dérivés.
    - » Gère **nativement** les données **géo-spatiales** :
      - Index spatiaux (*geo-hashing*), exploitation de quelques prédicats spatiaux, SRID prédéfini, langage **GeoJSON**, etc.
      - *Rmq. Concerne essentiellement des données **ponctuelles**, attachées à des évènements (appels, SMS, photographies…).*

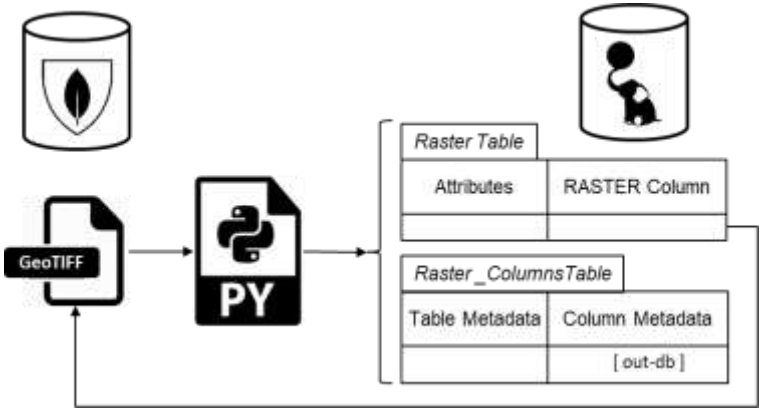
mongoDB

- **En résumé**
  - **Base de données en clé-valeur/colonne**
    - » Importante quantité de données mais peu de différences
  - **Base de données document**
    - » Structure importante mais données hétérogènes.
  - **Base de données en graphe**
    - » Fortes interractions et relations entre les informations.

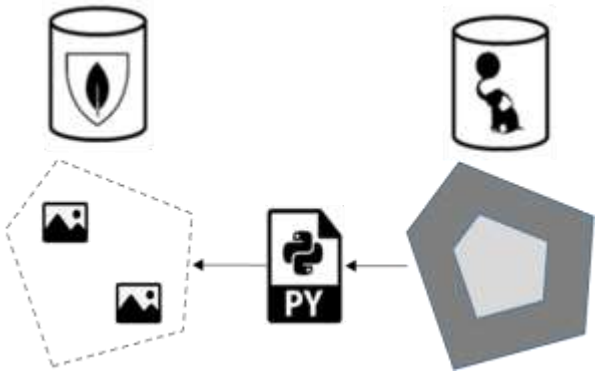
Les modèles multi-solutions sont possibles.



Utilisation d'une donnée géographique ponctuelle de MongoDB pour lancer une recherche proximale sur des entités polygonales stockées par PostGIS



Partage d'une couche raster géolocalisée (GeoTIFF) gérée hors des bases de données (tant MongoDB que PostGIS)



Définition d'un buffer autour d'un polygone dans PostGIS, soumis au prédicat « Within » dans MongoDB pour retrouver des photographies géo-localisées par un point

Exemples d'opérations sur des données géographiques dans un système hybride : MongoDB + PostGIS (opérations du driver en Python)

Source : Holemans, Kasprzyk, Donnay, Proceedings GeoProcessing 2018
cf. Article complet sur eCampus

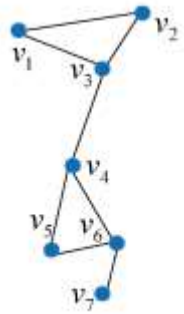
- **5. GraphStore en théorie**
  - **Définition du concept de graphe**
    - » **Non orienté**

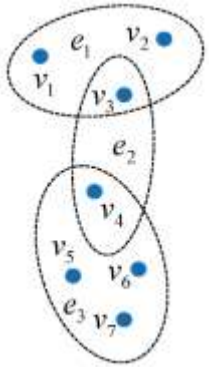
      Couple formé d'un ensemble de sommets (*vertices*) et d'un ensemble d'arêtes (*edges*). Chaque arête correspond à une paire de sommets.
    - » **Hypergraphe**

      Généralisation de la notion de graphe non orienté où les arêtes ne relient plus un ou deux sommets mais un nombre quelconque de sommets.
    - » **Taxonomie (taxinomie – typologie)**

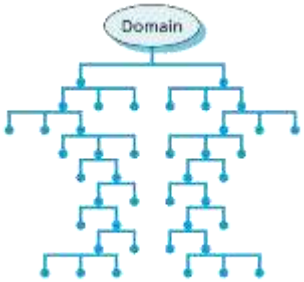
      Liste ordonnée (classée) des éléments d'un domaine. Se traduit souvent par un arbre (graphe connexe sans circuit) de catégorisation thématique.

Graphe simple non orienté





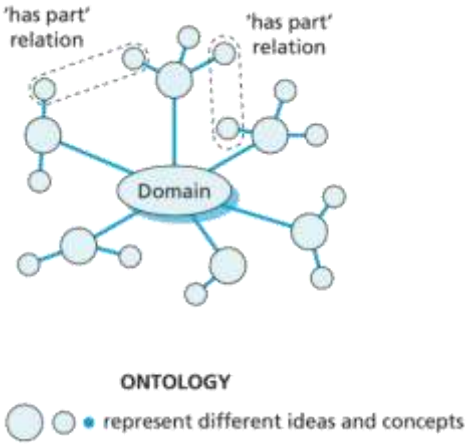
Hypergraphe



- **5. GraphStore en théorie**
  - **Définition du concept de graphe**
    - » **Thesaurus (thésaurus)**

      Liste organisée (alphabétique, analogique, normalisée, exhaustive) des termes couvrant les concepts d'un domaine et utilisée pour sa classification (indexation). Fréquemment représenté sous formes de graphes pour faciliter son exploration.
    - » **Ontologie (sens informatique !)**

      Ensemble structuré de concepts, eux-mêmes organisés dans un graphe dont les relations peuvent être sémantiques, de composition et d'héritage (au sens objet).



- **Avantages de la representation en graphe**
  - » **Avantages du NoSQL (voir Slide 5)**
  - » **Théorie des graphes**

    Les algorithmes classiques de théorie des graphes sont applicables :
    - Plus-court-chemin (algorithme de Dijkstra, A*, etc.).
    - Calcul de centralité.
    - Groupement (*clustering*).
    - Etc.
  - » **Développement intuitif**
    - Une fois maitrisée, l'expressivité des arcs et nœuds est d'une efficacité rare pour le RDF.

– Inconvénients de la représentation en graphe

» Désavantages du NoSQL (voir Slide 6)

» Indexation difficile

Une requête de type « *Quels sont les éléments de type X* ? » correspond à extraire toutes les ressources dans tous les triplets de la BD qui correspondent à la forme

:Element rdf:type :typeX

Une base de donnée relationnelle se limiterait à lire un des attribut de tous les tuples d'une table.

» Complexité quasi-quadratique des modèles

L'ajout d'un élément, concept, relation… peut engendrer un nombre important d'arcs et de nœuds. Dans un SGBR, cela correspond à ajouter un tuple.

Cette affirmation sous-entend aussi le fait que la redondance est possible et n'est plus à éviter. Ceci au détriment du temps de calcul et du volume de données.

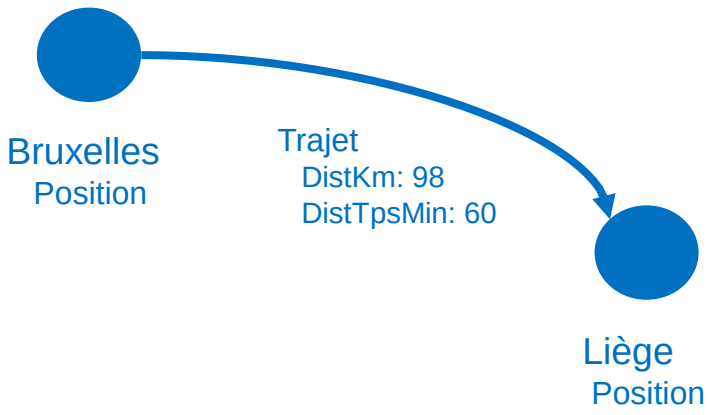
» Coûts en temps et personnel importants

La création et la maintenance d'une base de données en graphe demande un personnel expert et une temps certain.

– Modèle de données

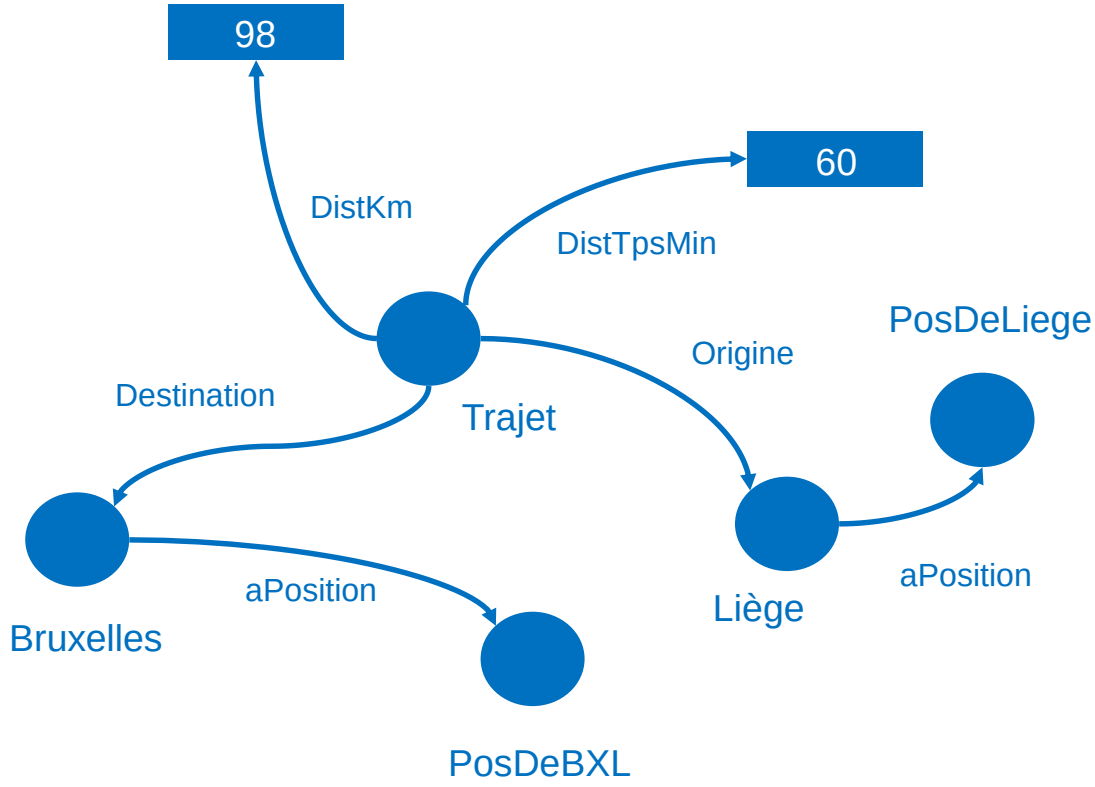
» **Labelled Property Graph (*Storage*)**

- ID + paire clé/valeur



» **RDF (*Data exchange*)**

- URI (*Uniform Resource Identifier*)



– **Utilisation de l'URI** *(Uniform Resource Identifier)*

» **Unicité des éléments**

RDF ne permet pas de reproduire un même triplet.

» **Conception des propriétés « Multi-valeurs »**

Différence de philosophie :

- RDF utilise n pointeurs : cheminement aisé du graphe.
- LPG utilise des *Arrays*.

» **Conception des arcs**

- RDF(OWL) définit des **caractéristiques** des arcs.
- LPG définit des **attributs** des arcs.

» **Fusion des graphes**

- Identification unique de toutes les ressources.

– **Langage d'exploitation**

CRUD *(Create, Read, Update, Delete)*

» LPG

- Cypher : propriétaire Neo4J
- GraphQL, Gremlin…

» RDF

- SPARQL 1.1

  CONSTRUCT : extraction de sous-graphes

  DESCRIBE : extraction de la description d'une ressource (connexions directes).

  ASK : boolean if exist

- DQL, XUL, Adenine…

– **OWL** *(Web Ontology Language)*
  - » **Language de définition des ontologies**
  - » Utilisation des URIs.
  - » Caractérisations des arcs.

  - » Raisonnement de premier ordre (déduction, abduction et induction).
  - » Raisonnement spatio-temporel.
  - » Règles de cohérence / incohérence.

- **6. GraphStore en pratique**
  - **De façon générale**
    - » Mutualisation de l'information.
    - » Bases de données colossales accessibles et normées.

    - » **Semantic Web**

    1239 Bases de données liées par 16147 liaisons (Mars 2019):
      - DBPedia : Wikipédia
      - GeoNames : NUTS, toponymies, …
      - MusicBrainz : Open Music Encyclopedia
      - BBC: Public Service Broadcaster
      - Etc.

– Illustrations par les outils

Core

neo4j

Plus utilisé, incontournable.
LPG avec moteur d'analyse, interface user-friendly…
Cypher

GraphDB

Bonne entrée en RDF Store mais avec limitations
SPARQL, Sesame, GeoSPARQL…

Jena

OpenSource (License Apache 2.0) → très modulable
Supporte OWL2 et Reasoners
Fuseki, TDB, RDF API, ARQ, OWL API…

Java

Java RDF4J

Java

– **Outils de gestion des données géospatiales**

» LPG :

- Neo4J - Cypher : distance(), point() [Coord2WGS84], point() [Coord2Cart]
- Possibilités de trouver des extensions non officielles sur GitHub

» RDF :

- **GeoSPARQL**

Standard supportés :

– GML (*Geography Markup Language)*
– WKT (*Well-know text*)
– Définition des relations : RCC8 et DE-9IM

– Illustrations par projets

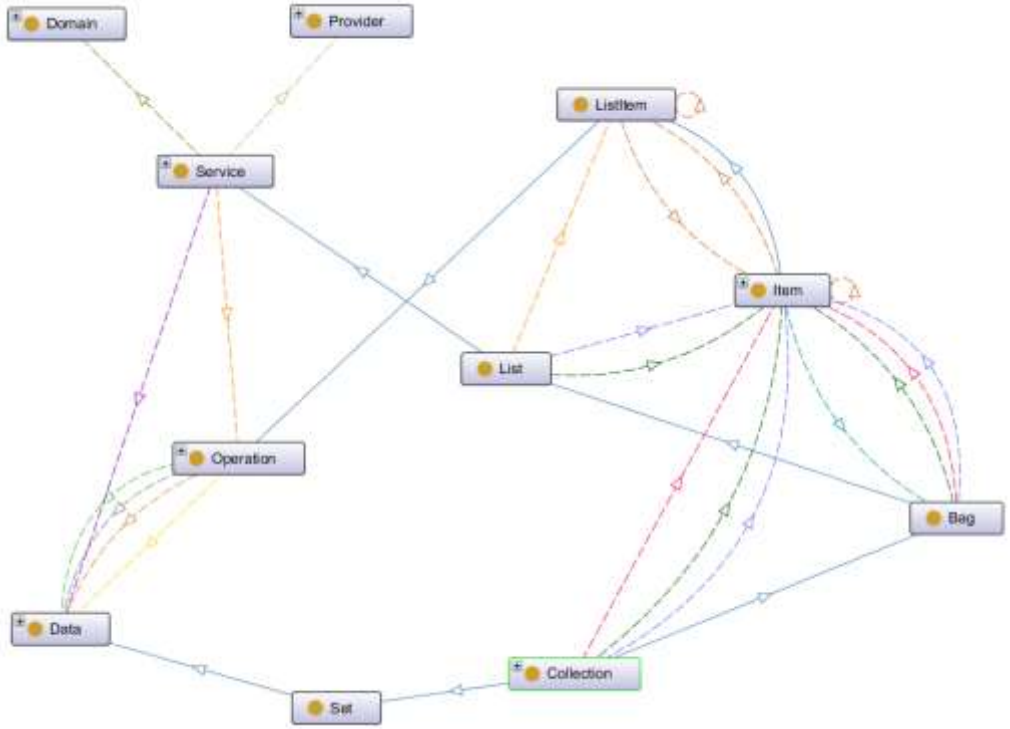
EORegions!

Towards AN Ontology for the Structuring of Remote Sensing Operations Shared by Different Processing Chains GA Nys, JP Kasprzyk, P Hallot, R Billen - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2018

EORégions! est le projet qui réunit et développe les compétences des acteurs wallons actifs dans le domaine géospatial, afin de déployer une plateforme numérique accessible aux utilisateurs et fournisseurs de données d'observation de la Terre.

Structure de la notion des services.

Analyse des « *patterns* » et « *paths* ».



Architecture :

- RDF Triple Store
  - » Fuseki 3.4.0 + Jena API 3.6.0
- Reasoner Openllet v2.6.3
- SPARQL Endpoint : SPARQL 1.1

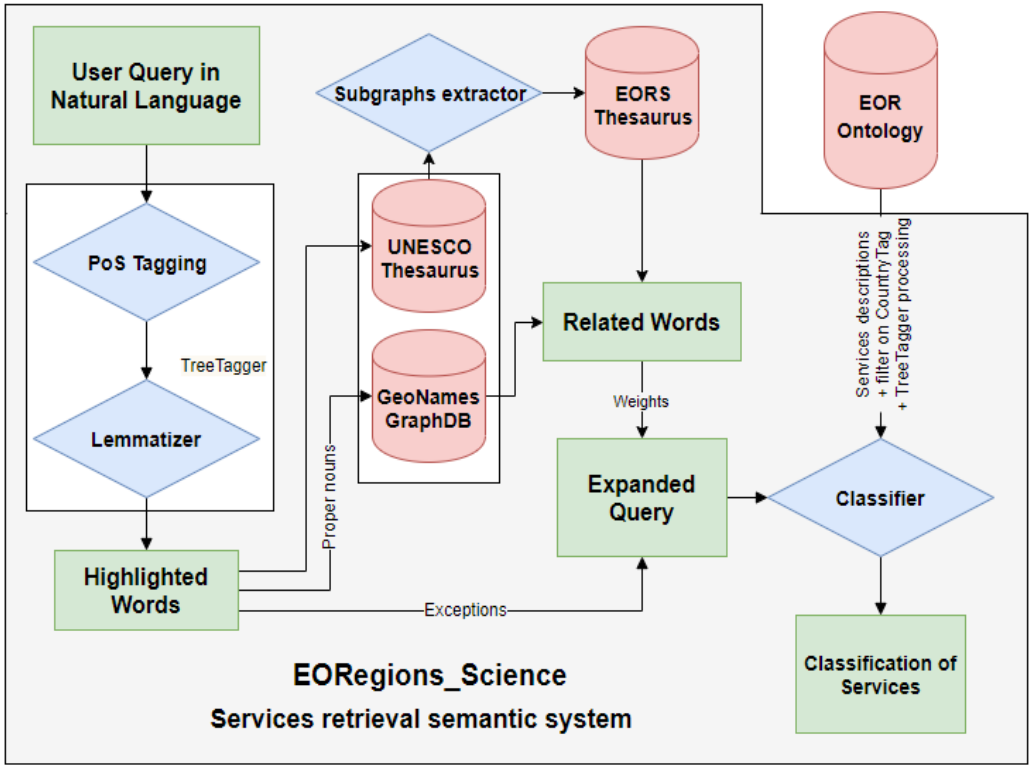
Dockerisée et accessible (Google Cloud Platform)

– Illustrations par projets

EORegion_Science

Soumis : A semantic retrieval system for remote sensing web platforms, GA Nys, JP Kasprzyk, P Hallot, R Billen - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2019

Linking queries with natural language and consistency analysis.



» RDF Store

- UNESCO Thesaurus

  Listes de termes contrôlées et structurées en graphe.

- GeoNames

  Base de données en graphe (NUTS, Toponymies…).

- EORS Thesaurus

  Listes de termes structurées en graphe et contrôlées, dédiées à EORegion_Science.

- EOR Thesaurus

EARTH OBSERVATION REGIONS !

– Illustrations par projets

Thèse Muriel van Ruymbeke

Spatio-temporal reasoning in CIDOC CRM: an hybrid ontology with GeoSPARQL and OWL-Time, GA Nys, M Van Ruymbeke, R Billen - CEUR Workshop Proceedings, 2018

Raisonnement spatio-temporal sur une base de connaissance archéologique.

» RDF Store – GraphDB

- CRM-CIDOC

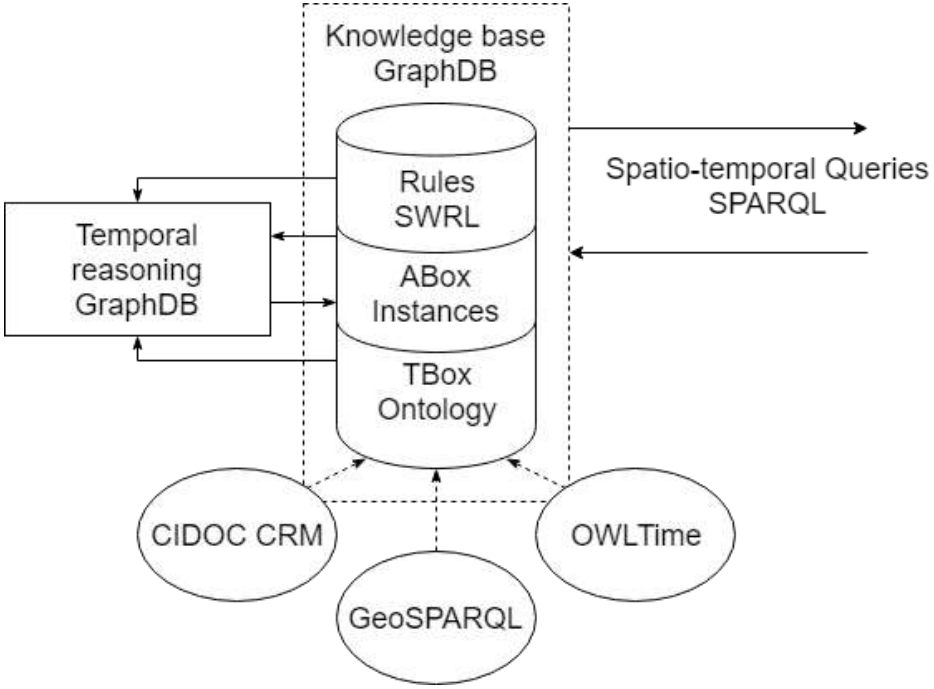
  Liste de termes contrôlées et structurées en graphe pour la gestion de la donnée archéologique

- GeoSPARQL

  Liste de termes contrôlées et structurées en graphe pour la gestion de la donnée spatiale

- OWL-Time

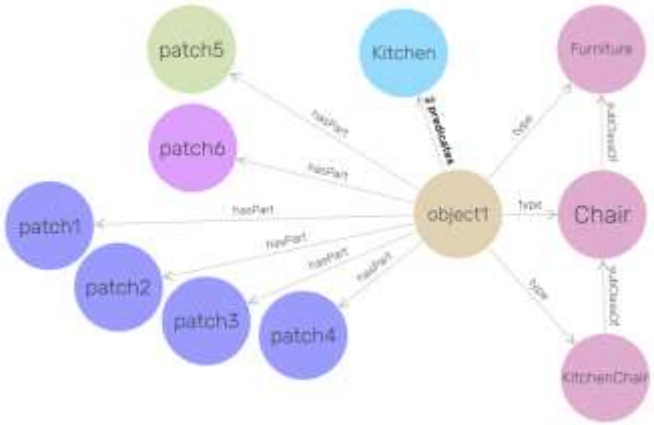
  Liste de termes contrôlées et structurées en graphe pour la gestion de la donnée temporelle



– Illustrations par projets

Thèse Florent Poux

3D Point Cloud Semantic Modelling: Integrated Framework for Indoor Spaces and Furniture, F. Poux, R. Neuville, GA Nys, R Billen – Remote Sensing, 2018

Structuration de la connaissance en soutient de la segmentation automatisée. Reconnaissance des élements constituants des objets.

» Ontologie de description des éléments intérieurs de construction