# þÿ I m p l é m e n t a t i o n d u n e b a s e d e d o n n é e s g é o s p a t i a l e N o S

**Auteur :** Holemans, Amandine
**Promoteur(s) :** Donnay, Jean-Paul
**Faculté :** Faculté des Sciences
**Diplôme :** Master en sciences géographiques, orientation géomatique et géométrologie, à finalité spécialisée
**Année académique :** 2016-2017
**URI/URL :** http://hdl.handle.net/2268.2/3136

**UNIVERSITÉ DE LIÈGE**
**Faculté des sciences**
**Sciences géographiques**


# Implémentation d'une base de données NoSQL de données géospatiales de l'AIDE

Création d'une base de données MongoDB orientée document à partir de données du réseau de canalisations d'Oreye

Mémoire présenté par :
**Amandine Holemans**
Pour l'obtention du titre de :
**Master en sciences géographiques, orientation géomatique et géométrologie**

Année académique :
**2016-2017**

**UNIVERSITÉ DE LIÈGE**
**Faculté des sciences**
**Sciences géographiques**


# Implémentation d'une base de données NoSQL de données géospatiales de l'AIDE

Création d'une base de données MongoDB orientée document à partir de données du réseau de canalisations d'Oreye

Mémoire présenté par :
**Amandine Holemans**
Pour l'obtention du titre de :
**Master en sciences géographiques, orientation géomatique et géométrologie**

Année académique :
**2016-2017**

# REMERCIEMENTS

Je voudrais tout d'abord remercier mon promoteur, M. Donnay, pour ses conseils, son accompagnement et sa patience.

Je remercie également M. Schmitz et M. Kasprzyk pour l'intérêt qu'ils portent à mon travail.

Je veux également remercier Mr Legros de m'avoir permis d'accéder au Cadastre et à ses données géospatiales.

Enfin, un tout grand merci à ma famille, mes parents qui m'ont soutenue et supportée tout le long de ce travail, ainsi que mes grands-parents pour leurs attentions particulières.

Table des matières

Liste des illustrations

# 1. INTRODUCTION

Le domaine des bases de données représente une problématique clé de notre époque. La capacité de gestion de l'information constitue la base de toute application. Que l'on parle d'applications locales, web ou mobiles, elles facilitent notre quotidien et sont de plus en plus nombreuses et complexes.

Pour répondre à cette évolution, les bases de données doivent s'adapter et, en particulier, pour répondre à une évidence : le développement récent d'internet, le web 2.0.

Le Web 2.0 est apparu dans les années 2000. Cette tendance est définie par une innovation des interfaces en vue d'en faciliter l'utilisation par les internautes, mais aussi par une complexification interne dû au progrès des technologies. L'aire du web 2.0 voit le nombre d'échanges sur le net décupler (forum, réseau sociaux, commerce, etc.). L'utilisateur passe ainsi de spectateur à acteur et cela a une conséquence : si plus d'acteurs sont en activité, plus de créations apparaissent, et donc plus de données. C'est l'apparition du Big Data et de ses enjeux en termes de systèmes de gestion de bases de données.

## 1.1 Qu'est-ce que le Big data ?

Le Big Data consiste à proposer une alternative à la gestion traditionnelle des données. Les données peuvent être chargées facilement et rapidement sur un grand nombre de plateformes ce qui crée un comportement différent quant à la production de données.

Chaque année 2,5 quintillions de bytes sont produits (IBM, 2012). La plupart des données sont produites par les réseaux sociaux, elles représentent plus ou moins 90% des données disponibles (Sharma, 2015). Pour Twitter, 400 millions de tweets ont été publiés par jour en 2014, 9100 tweets par seconde (Huang & Xu, 2014). Plus de données sont créées, mais on ne possède pourtant pas plus de moyens pour les stocker. Il faut donc trouver un moyen de gérer plus de données à un moindre coût.

De plus, les données de réseaux sociaux sont également composées de types de formats variés. 80 % des données sauvegardées mondialement sont non ou semi-structurées (Carolin, 2014). Sous forme de texte, de lien, d'images ou de vidéos, les données doivent toutes être stockées dans un même modèle de bases de données et cela avec des attentes de performances toujours plus importantes. Les données non structurées et le besoin de plus de performances sont deux problématiques supplémentaires pour le Big Data.

Les sources responsables de cet amas de données se retrouvent partout : des capteurs pour les analyses de climats, des photos et des vidéos, des enregistrements de transactions, des coordonnées GPS des smartphones, …

Pour résumer, on parle souvent de la théorie de 3 V (certains vont même jusque 5). Le premier est pour le Volume de données, le deuxième pour Vitesse, le troisième pour Variété (d'autre parle aussi de Valeur et Variabilité) (Bulk, 2011).

## 1.2 Apparition du NoSQL

Le NoSQL est une des solutions au Big Data. Le NoSQL est apparu au départ pour subvenir aux besoins de certaines applications particulières confrontées au Big Data (Oussous et al. , 2015). Des bases de données pensées autrement ont été créées par des sociétés et organisations pour répondre à leurs propres besoins et ont été par la suite qualifiées de « NoSQL » (Funck et al., 2011).

Les BDD NoSQL sont également apparues pour permettre la meilleure gestion des données non structurées. En effet, c'est l'apparition d'un besoin d'un modèle plus simple qui puisse convenir à des systèmes orientés objets qui a motivé l'apparition du NoSQL.

Le terme NoSQL est utilisé pour la première fois en 1998 pour une base de données relationnelle qui n'utilisait pas de SQL. Ce terme sera utilisé onze ans plus tard pour qualifier, cette fois-ci, une base de données non relationnelle (Oussous et al. , 2015). NoSQL signifierait donc ici plutôt « *Not only SQL* » que « pas de SQL ».

Le NoSQL est donc une adaptation aux nouvelles tendances, une révision de la manière de concevoir les bases de données.

## 1.3 Le Big Data et la géographie

Les données géolocalisées sont aussi concernées par le Big Data. Elles présentent les mêmes caractéristiques que les autres données : la grande quantité, le type de données non structurées ainsi que la demande de vitesse de réponse.

Pourtant, ces données sont particulières et présentent des caractéristiques impossibles à gérer de la même manière que le reste des informations. Les systèmes de gestion de bases de données NoSQL vont donc attirer notre attention. Le géospatiale représente très rarement une problématique de premier plan dans le Big Data et peu d'exemples de systèmes d'informations géographiques NoSQL fonctionnels sont connu.

Des exemples inexploités de ce type de données pouvant constituer un SIG (système d'informations géographiques), il en existe beaucoup. Une grande partie des sociétés créent un paquet de données de types documentaires et géolocalisables. Ces types de données offrent des possibilités intéressantes en termes de SIG, mais très peu d'entre elles sont exploitées.

Les compagnies de réseaux d'impétrants constituent un exemple intéressant. Elles sont nombreuses en Belgique, ces sociétés gèrent les réseaux électriques, la télédistribution, la téléphonie, la distribution d'eau, les eaux usées, etc.

L'AIDE fait partie de ces compagnies. Elle gère les zones de démergements et les réseaux d'assainissement des eaux usées dans la province de Liège. Leurs données peuvent atteindre des grandes quantités mais surtout présenter des types et relations fortement variées. Ce modèle de données peut poser des difficultés avec les types de bases de données relationnelles.

Ainsi, travailler avec les données de l'AIDE offre l'occasion de tester l'implémentation d'une base de données NoSQL géospatiale.

La question de ce travail serait donc : Comment introduire et gérer efficacement les données documentaires non structurées sélectionnées en parallèle avec l'infrastructure géomatique existante au sein d'une société comme l'AIDE ?

Ce travail permettra également de tester la capacité d'un système de gestion de bases de données NoSQL à intégrer des données issues d'une base de données relationnelle. Peuvent-elles supporter les données structurées issues d'un BD relationnelle ? Le schéma peut-il être modifié et comment ?

Enfin ce travail permettra également de tester les capacités de gestions des données géoréférencées d'une base de données NoSQL.

Ce travail sera structuré en 5 étapes différentes. Tout d'abord, une étude de la littérature est effectuée pour fournir les informations nécessaires au choix du type de système de base de données NoSQL présenté. Cette étape amène à la formulation de l'hypothèse de travail dans son contexte. Ensuite, les thèmes sont approfondis afin de pouvoir passer à l'application des théories, avant une présentation des conclusions de ce travail.

# 2. ETAT DE L'ART

## 2.1 Systèmes de gestion base de données

Les systèmes de gestion de bases de données constituent une grande problématique de nos jours. Ce sont des systèmes chargés d'enregistrer les données et de les classer pour offrir une recherche et un rangement facile et rapide des informations. Dans un programme, les BD (base de données) permettent de stocker à long terme ce que les variables enregistrent à court terme.

L'augmentation de la performance d'une application peut passer, entre autres, par l'adaptation de la base de données aux types de données et aux exigences de l'application. Il faut donc pouvoir déceler à quelle situation correspond quel SGBD (système de gestion de base de données) et quelles sont les bonnes pratiques pour en augmenter les performances.

Les bases de données les plus utilisées actuellement sont les bases de données relationnelles. Ce sont les premières base de données qui ont connu un succès général.

Pendant des années, ce sont ces systèmes de gestion relationnels qui ont constitué la base du stockage de bases de données géospatiales. Les SGBDR sont utilisés avec une extension pour pouvoir gérer les données géospatiales. Ces systèmes sont encore utilisés mais se confrontent aux mêmes problèmes que les systèmes non spatiaux : volume, rapidité demandée et la variabilité des composants de la base de données.

## 2.2 Modèle relationnel

### 2.2.1 Caractéristiques principales

L'apparition des bases de données relationnelles sur le marché date de 1980 (Losson, S.d.). En 10 ans, elles prennent la tête du marché. Voici les caractéristiques générales des systèmes de gestion de bases de données relationnelles afin de pouvoir comprendre leur fonctionnement et les comparer aux SGBD NoSQL.

Les SGBDR sont définis par un schéma rigide qui demande des données structurées. Les données sont classées en plusieurs tables, contenant chacune plusieurs lignes (aussi appelés tuples). Chaque table possède un certain nombre de colonnes fixées à l'avance. Les tables sont normalisées, c'est-à-dire qu'elles sont conçues afin d'éviter toute redondance dans les données. Ainsi, un type de données qui a tendance à se retrouver plusieurs fois dans la même table (comme le nom d'une ville, par exemple) se verra isolé dans une autre table et lié aux autres informations par une clé. Cette particularité implique un nombre plus important de tables que la modélisation logique des données en aurait créé (Parker et al. , 2013).

Les SGBDR assurent également l'intégrité des données et la cohérence des transactions. Cela signifie que les données visibles par les utilisateurs sont toutes communes et qu'une modification validée sera toujours en accord avec les conditions d'intégrité du système.

Ces deux caractéristiques sont acquises aux dépends d'une bonne performance et provoque un investissement en coût et en temps plus important (Oussous et al. , 2015). Il faut savoir que les bases de données relationnelles et NoSQL héritent toutes deux leurs caractéristiques du théorème CAP (cohérence, disponibilité, tolérance à la partition) énoncé par Éric Brewer. Ce théorème souligne que seules deux des trois caractéristiques peuvent être respectées simultanément pour une base de données (Cattell, 2010).

L'intégrité permet d'assurer une sécurité sur les données. Ce point est plus qu'utile pour certaines applications comme par exemple, les transactions bancaires. Cela signifie que tous les utilisateurs peuvent à tout moment avoir accès en même temps aux mêmes données (Amirian et al., 2014).

La demande de structure des SGDR apporte un grand avantage car il permet d'offrir des fonctions prédéfinies qui rendent la gestion de la base de données plus facile. En effet, pas besoin d'avoir implanté toute la base de données pour pouvoir déjà concevoir des manipulations car le schéma est fixé à l'avance.

Les SGBD relationnels respectent les termes « ACID » qui signifient : Atomicité, cohérence, isolation et durabilité.

Le langage de requête du relationnel est le bien connu Structured Query Language (SQL). Voici quelques systèmes existants présents sur le marché :

- MySQL
- Oracle
- Access
- PostgreSQL

### 2.2.2 Désavantages du système relationnel

Un premier désavantage des SGBDR est que l'évolution de la quantité de données pose problème à un certain moment. Les bases de données relationnelles ont le désavantage de ne pouvoir évoluer que de manière verticale. L'évolutivité verticale (« *vertical scalability* » en anglais) désigne l'augmentation des ressources déjà existantes. C'est-à-dire, par exemple, augmenter la capacité de stockage du serveur ou améliorer la puissance de son processeur. Par contre, une évolution horizontale (l'augmentation de la puissance grâce à la connexion de plusieurs serveurs travaillant comme un seul) est complexe à mettre en place dans un système relationnel car les joints qui relient les tables des différents serveurs ralentissent les performances dans un système distribué (Funck et al., 2011). Une évolution verticale peut s'avérer être assez couteuse et difficile à mettre en place.

La gestion des changements pose donc problème dans le cas de SGBD relationnels et pas uniquement dans le cas de l'augmentation de la quantité de données à stocker. Le schéma doit être établi à l'avance pour la création de la BD, ce qui implique des complications si les données sont non structurées ou si elles évoluent au cours du temps. Des processus existent pour modifier une table (Alter table), mais ils sont très couteux en termes de performance. De plus, beaucoup d'applications géospatiales ont des géométries fortement variables, pouvant prendre des formes différentes, ce qui se heurte au schéma structuré des bases de données relationnelles.

La normalisation des données implique un grand nombre d'opérations de jointures pour interroger la base de données et plus elle sera large, plus la requête prendra du temps. Cette disposition a des désavantages car le joint est un des processus les plus couteux en termes de calcul informatique (Amirian et al., 2014). Dans certains cas, on peut éviter les joints par la dénormalisation, ce qui peut amener à une croissance de la taille de la base de données ainsi qu'à une incohérence lors de la mise à jour de données qui se retrouvent dans plusieurs tables à la fois. A nouveau, l'augmentation de la quantité de données peut rapidement ralentir un SGBD relationnel comme mentionné plus haut.

Les SGDBR ne sont pas créés à l'origine pour gérer le « *sharding* ». Cette notion rassemble différentes techniques permettant de stocker les données sur des machines différentes. Pourtant, cette capacité a été ajoutée par la suite aux techniques des SGBDR (Oussous et al. , 2015). Ainsi, le « *sharding* » fonctionne mal vu que la structure des bases de données n'est pas imaginée dans ce but. Les tables sont divisées en plusieurs « *shard* » et un schéma de cette division est stocké pour permettre au système de réassembler les données quand des requêtes ou transactions le demandent.

Ces affirmations amènent également à comprendre que les SGBDR ne correspondent pas non plus aux environnement Cloud (Oussous et al. , 2015). En grande partie à cause de leur faible capacité d'évolutivité horizontale.

## 2.3 NoSQL

### 2.3.1 Avantages d'une BD NoSQL

L'utilisation en forte progression des systèmes de gestion de bases de données NoSQL ne signifie pas que les bases de données relationnelles sont dépassées, mais elles répondent plutôt à un nouveau type de demande. Le NoSQL et les bases de données relationnelles ne sont pas apparues à la même époque. Le relationnel apparait à une époque où c'est le stockage qui constituait la partie la plus chère du système tandis que les BD NoSQL sont nées à une époque où c'est le temps de travail des développeurs qui a le plus de valeur (MongoDB, 2015).

La plupart des systèmes NoSQL sont open source. Ce terme fait même partie des points définissant le NoSQL (NoSQL, 2011).

Les SGDB NoSQL sont, cette fois-ci, évolutifs horizontalement. Ainsi, pour gérer une grande quantité de données, il faut ajouter un serveur et non en augmenter sa capacité ou ses performances (plus facile et moins cher) (Oussous et al. , 2015). Le « *sharding* » dans le cas du NoSQL n'affecte pas les performances, la plupart des systèmes sont conçus à une époque où le « *sharding* » est une solution évidente aux problèmes de limitation des bases de données. Ainsi les BD NoSQL peuvent gérer un plus grand nombre de données et de transactions. Le NoSQL peut donc offrir plus de performances pour moins de coûts.

La flexibilité et l'évolutivité horizontale offrent aux bases de données NoSQL une grande capacité d'adaptation au changement sans affecter le système ou les performances. C'est cette caractéristique qui lui permet de tourner sur plusieurs serveurs (McKnight Consulting Group, 2014).

Elles offrent plus de flexibilité ce qui implique moins d'intégrité pour les données (Oussous et al. , 2015). Les transactions qui modifient les données prennent parfois du temps à être prises en compte par le système de gestion. Ainsi, les versions de la base de données visibles par les différents clients ne sont pas toujours identiques. Cette fois-ci, ce type de caractéristiques correspond plus à des applications moins exigeantes en termes d'intégrité de données, comme par exemple les sites de réseaux sociaux.

Les systèmes de gestion bases de données NoSQL peuvent s'adapter à tous les types de données car ils supportent les données structurées comme les non structurées. Ainsi, certains schémas correspondant au relationnel peuvent migrer vers le NoSQL, s'il offre plus d'avantages.

La réplication des bases de données NoSQL est pensée autrement que pour les bases de données classiques. Les réplications sont effectuées automatiquement dans les clusters (ensemble de plusieurs serveurs) et dans les centres de données. C'est ce qu'on appelle le processus « *Map/Reduce* ». C'est un système de traitement qui gère le partitionnement tout en assurant le fonctionnement du système en cas de pannes. La fonction « *Map* » collecte les données demandées par la fonction et « *Reduce* » fournit le résultat agrégé (Carolin, 2014). Contrairement au relationnel la réplication dans le NoSQL ne demande pas d'applications supplémentaires. La réplication en temps réel permet une grande disponibilité des données. Les données sont divisées et stockées dans des serveurs différents, ce qui permet à l'application de continuer à fonctionner si un problème avec un serveur apparait. Mais cette solution n'est pas encore mature (Oussous et al. , 2015).

Le NoSQL utilise une programmation orientée objet beaucoup plus facile à manipuler et à modifier pour toute personne ayant l'habitude de programmer.

Enfin, le Nosql est basé sur le principe de BASE (« *basically avaliable, soft state, enventual consistency* »). Ce qui signifie que le système doit toujours être accessible mais que la cohérence n'est pas toujours respectée et n'est pas primordiale.

### 2.3.2 Désavantages

Bien évidemment le NoSQL connait également beaucoup d'inconvénients.

Premièrement, le langage de requêtes ne connait pas de standard, chacun des systèmes de gestion de base de données possède son propre langage et son propre système de gestion. Ceux-ci ne sont pas toujours performants et ne connaissent pas encore toutes les capacités fonctionnelles des systèmes relationnels (Carolin, 2014). Ces problèmes sont souvent dus à la jeunesse du système. Il faut donc s'adapter à chacun des systèmes utilisés. Dans ce cas-ci, la gestion de requêtes plus complexes devient plus difficile qu'avec le SQL (Oussous et al. , 2015). De plus de nombreuses opérations de base des bases de données relationnelles ne sont pas encore disponibles (exemples : opérations atomiques disponibles uniquement sur un document) (Parker et al. , 2013).

Il n'y a également pas d'intégrité de données, comme déjà mentionné, et cela peut représenter un inconvénient pour beaucoup de types d'applications.

Un autre point posant problème est la question de la sécurité. Les chercheurs sont toujours en train d'explorer des solutions à ces problèmes : il n'existe pas de sécurité du serveur client lors des

communications et parfois pas d'authentifications. Certains systèmes ont un login et un mot de passe qui sont stockées dans des fichiers internes donc offrent peu de sécurité (Oussous et al. , 2015).

Les systèmes NoSQL ne sont pas matures, et sont en constante évolution, ce qui offrent une quantité de désavantages, du moins, actuellement. Il n'existe pas encore de standards concernant les formats ou les langages utilisés pour les SGBD NoSQL.

### 2.3.3 Le NoSQL et le géospatial

Il reste encore un point très important qui n'a pas encore été abordé. Les données composant les Big Data ont souvent une caractéristique commune : un composant géospatial (qui peut apparaître sous des formes variées). Les données géospatiales sont une grande opportunité pour un grand nombre d'acteurs. Pour le particulier, les commerçants pour des analyse du marché, les scientifiques … Les données géospatiales existent sous plusieurs formes et viennent de plusieurs sources. Les données raster géolocalisées peuvent provenir de caméras emportées, caméra de sécurité et satellites qui provoquent chaque jours la création de grandes quantités de données (Lee et al., 2015). Certaines données ont également la particularité d'être également référencées dans le temps, ce qui augmente la complexité des données. Les données spatiales augmentent également avec les services Web géospatiaux de plus en plus nombreux ainsi qu'avec l'utilisation accrue du mobile. En lien avec cette tendance, les besoins de rapidité aux requêtes augmentent également (Sarthak et al., 2015).

Les données géospatiales doivent être traitées d'une autre manière que les données classiques à cause de leurs particularités. Ce type de données est plus complexe et souligne souvent les limites des bases de données classiques (Amirian et al., 2014). Les gros volumes, les différentes relations géospatiales entre les entités géographiques, les différentes sources de données ainsi que les différents formats de données peuvent parfois rendre les transactions très lourdes. Les limites de performances des systèmes existant empêchaient d'exploiter pleinement le grand volume et la grande rapidité des données géospatiales (Lee et al., 2015). Les systèmes d'informations géographiques (SIG) actuels ont du mal à gérer ces nouveaux types de données en constante évolution, en grande quantité et au contenu volatile.

Certains scientifiques ont même affirmé que les données géospatiales seraient le plus gros challenge des Big Data (Minelli et al., 2013). Des tests ont été réalisés pour comparer les performances des BDD NoSQL par rapport au BDD relationnel, et il a été conclu que les BDD NoSQL offraient une meilleure évolutivité et meilleures performances pour des données Big Data géospatiales (Amirian et al., 2014). La plupart des systèmes mis en place dans la communauté scientifique sont uniquement présentés dans un but expérimental.

Le NoSQL n'a pas été conçu à l'origine pour gérer les BDD géospatiale, certains systèmes incluent des supports pour gérer ce type de données mais d'autres auront besoin d'extensions.

## 2.4 Type modèle de SGBD NoSQL

Même si plusieurs caractéristiques générales ont été soulignées précédemment, il faut savoir qu'il existe plusieurs familles de systèmes de bases de données NoSQL étant donné qu'elles sont apparues progressivement, pour des buts différents et conçues par des auteurs différents. Chacune de ces familles peuvent parfois s'éloigner des définitions générales.

Il existe 4 principaux types de modèles de BD NoSQL (trié selon leur type de modélisation de données) : les BD clé-valeur, les BD document, les BD orientée colonnes et les bases de données graphes. Voici une description rapide des 4 types.

### 2.4.1 Base de données clé-valeurs

Les bases de données clé-valeurs sont les bases de données NoSQL les plus simples, elles sont constituées d'une clé unique et d'un pointeur renvoyant à un certain type de données, créant des paires clé-valeur regroupées en une table de hachage. Une illustration de cette configuration est disponible à la Figure 1. Ce type de BD est utile pour des données analytiques. Elle ne possède aucune capacité de base des BD, donc pour cela, l'utilisateur devra uniquement se baser sur l'application (Oussous et al. , 2015). Les requêtes sont limitées à la valeur de la clé, ce qui permet d'offrir une grande vitesse de réponse, mais ce qui complique les mises à jour.

Ce type de base de données peut stocker des données géospatiales, mais son incapacité à gérer les relations limite fortement les possibilités d'utilisation. Pour que les requêtes fonctionnent, il faut que les données soient indexées spatialement, ce qui au final donne des moins bonnes performances que pour les bases de données relationnelles.

Voici quelques exemples de base de données clé-valeurs : Redis, Riak et Voldemort. Cette dernière est utilisée pour une grande quantités de données, souvent géologique et les métadonnées des cartes. Un exemple de grand utilisateur est LinkedIn.

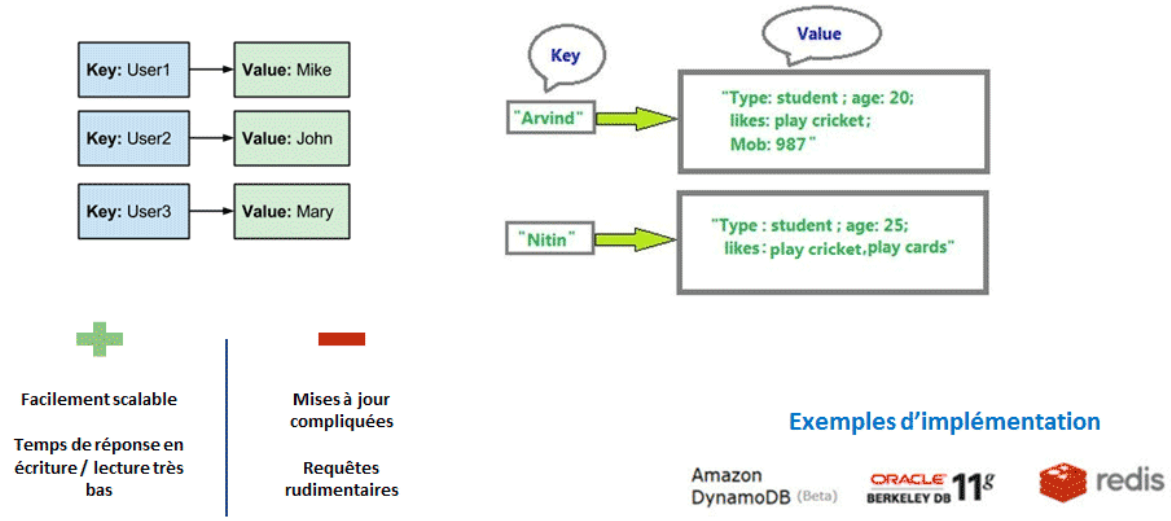



*Figure 1 Illustration d'une BD clé-valeur (Digora, s.d.)*

### 2.4.2 Base de données orientées colonne

Ce type de modèle est une extension des bases de données clé-valeurs, avec des colonnes et des caractéristiques des bases de données relationnelles (Oussous et al. , 2015). Elles sont peu adaptées aux mises à jour fréquentes et préfèrent les ajouts. Les relations ne sont pas supportées, ce qui implique la création d'une structure en réseau pour les données connectées, ce qui n'est pas la solution la plus

efficace. Utile quand une grande quantité de données doit être stockée et qu'une grande disponibilité est demandée (Amirian et al., 2014).

La plupart des systèmes de type orienté colonne peuvent supporter les données géospatiales, mais surtout avec des données intégrées et des requêtes simples avec une demande de réponse rapide (comme par exemple Facebook).

HBase, Cassandra,Big Table, DynamoDB et Accumulo sont des exemples de bases de données orientées colonnes. Elles sont utilisées par Amazon, Google et Facebook.

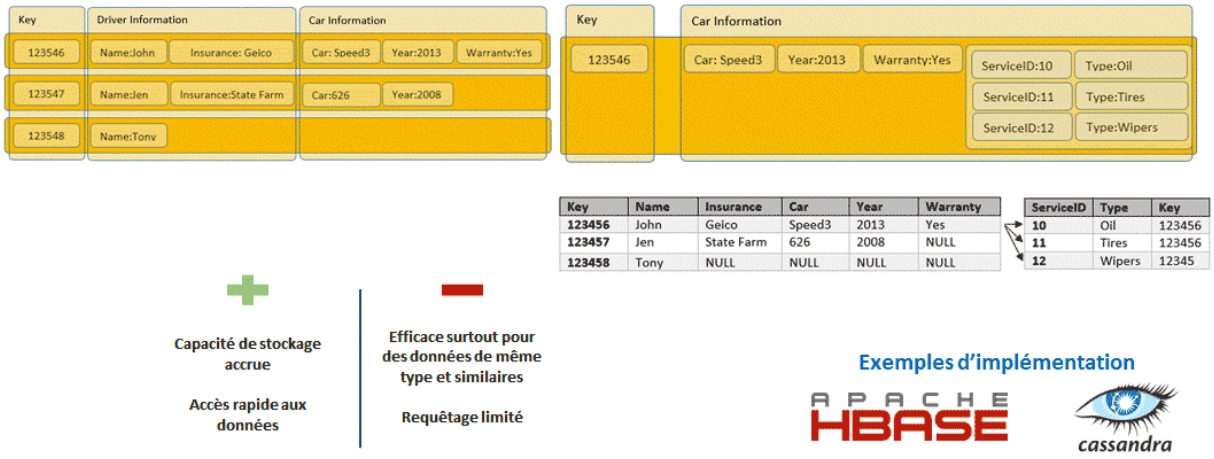



*Figure 2 Illustration d'une BD orienté colonne (Digora, s.d.)*

### 2.4.3 Base de données orientée graphes

Les sets de données fortement connectées sont les types de données qui correspondent avec ce type de BD. Les données peuvent être représentées par un graphe sans schéma fixe rassemblant arrêtes et nœuds pouvant chacun posséder des caractéristiques particulières. Ce modèle est capable de gérer des données relationnelles. Les BD graphes sont utiles pour les données constituées en réseau comme les routes, les réseaux sociaux, … Contrairement aux autres types de bases de données NoSQL, les BD orientées graphes ne sont pas optimales pour les grandes quantités de données si celles si ne sont pas assez connectées (Oussous et al., 2015). Sans surprises, les données géospatiales peuvent être concernées par ces BD. Elles supportent les relations topologiques, ce qui facilite la gestion de données spatiales (Amirian et al., 2014). C'est une des bases de données NoSQL la plus complexe à mettre en place (Carolin, 2014).

Des exemples sont : Oracle Graph, Neo4J, Arango et OrientDB. Elles sont utilisées par Walmart et Ebay

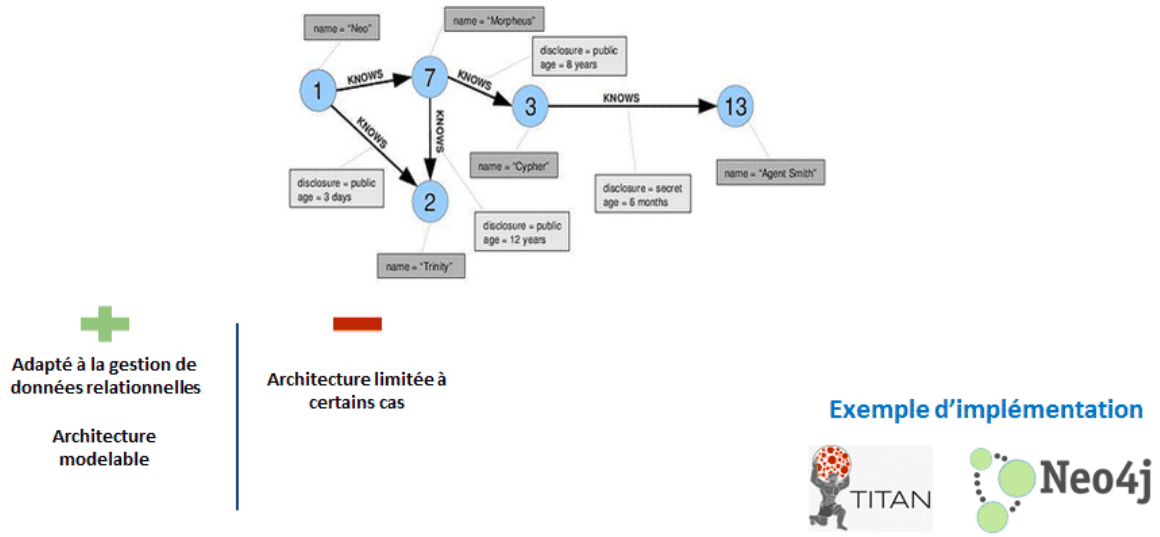



*Figure 3 Illustration d'une BD orienté graphe (Digora, s.d.)*

### 2.4.4 Base de données orientées document

Le type de modèle de base de données NoSQL qui va nous intéresser le plus sont les BD orientées documents. Nous le verrons par la suite en détail, mais les données qui vont devoir être introduites dans la base de données sont constituées d'un grand nombre de formats et de types différents.

Les BD orientées documents sont destinées à gérer une grande quantité de données. Une clé est assignée à un document qui peut lui-même posséder plusieurs couples clé-document intégrés (voir Figure 4). Un document peut être importé sous différents formats standards comme XML, JSON (Javascript Option Notation) ou BSON (Binary JSON). Ici, contrairement aux BD clé-valeurs, les recherches peuvent être effectuées sur le contenu des documents et pas uniquement leur clé. La présence des documents intégrés permet d'effectuer des recherches avancées et ne requiert pas de schémas prédéfinis.

Les BDD orientées document permettent à un document d'être décrit par un grand nombre de valeurs et peuvent supporter un schéma flexible. Elles peuvent donc accueillir une grande quantité de données sous des formats fortement différents. La relation entre les documents est représentée par l'intégration de ceux-ci, il n'existe pas de relations extérieures, un document est indépendant d'un autre qui ne lui est pas intégré. Ainsi, pour des requêtes qui interrogent les documents d'une même collection (ensemble de document), la réponse sera très rapide. L'utilisation de document XML peut améliorer les BDD orientées documents en offrant des fonctionnalités complémentaires (XQuery, Xpath, XPointer,…) et en offrant le principe de relation qui n'existait pas sans le XML (Amirian et al., 2014).

Ce type de base de données gère mieux les données géospatiales que les BDD orientées graphes grâces aux requêtes plus flexibles. Selon les types de base de données orientées documents, les données géospatiales sont gérées nativement ou grâce à des extensions.

CouchDB et MongoDB sont deux exemples de bases de données orientées documents. Twitter est un exemple d'utilisateur.

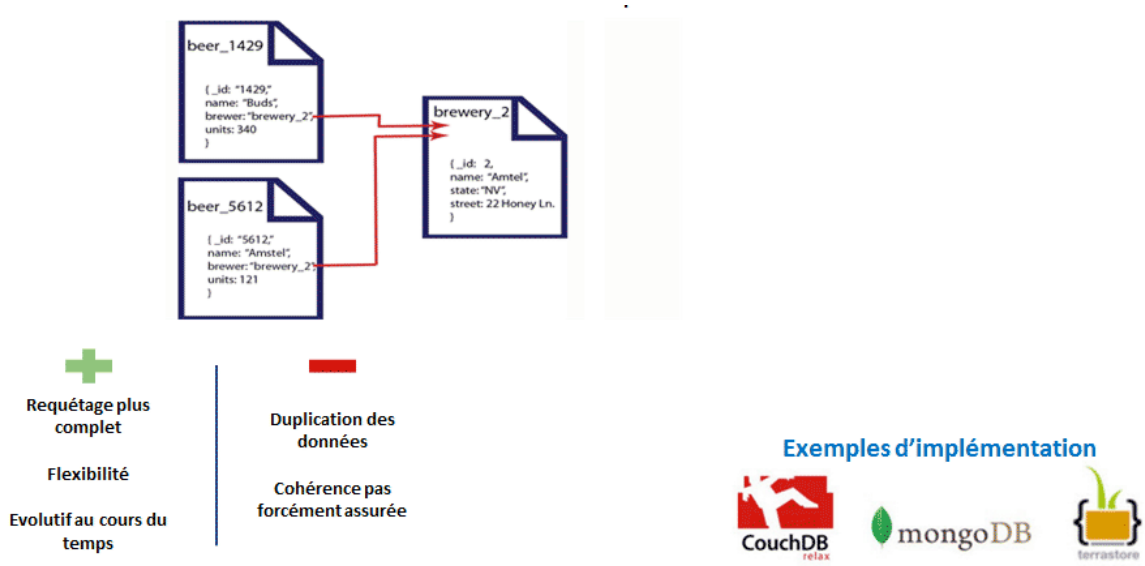



*Figure 4 Illustration d'une BD orienté document (Digora, s.d.)*

### 2.4.5 Vision d'ensemble

Voici un schéma (Figure 5) utilisant le théorème « CAP » pour comparer chacun des modèles de BD relationnelles et NoSQL. Pour rappel, un système ne peut réunir que deux des trois caractéristiques suivantes : tolérance à la partition, disponibilité et cohérence. Les bases de données relationnelles se retrouvent toutes sur le côté « CA » (cohérence et disponibilité) mais en ce qui concerne les systèmes NoSQL, un même modèle de BD peut avoir différentes caractéristiques selon son concepteur.

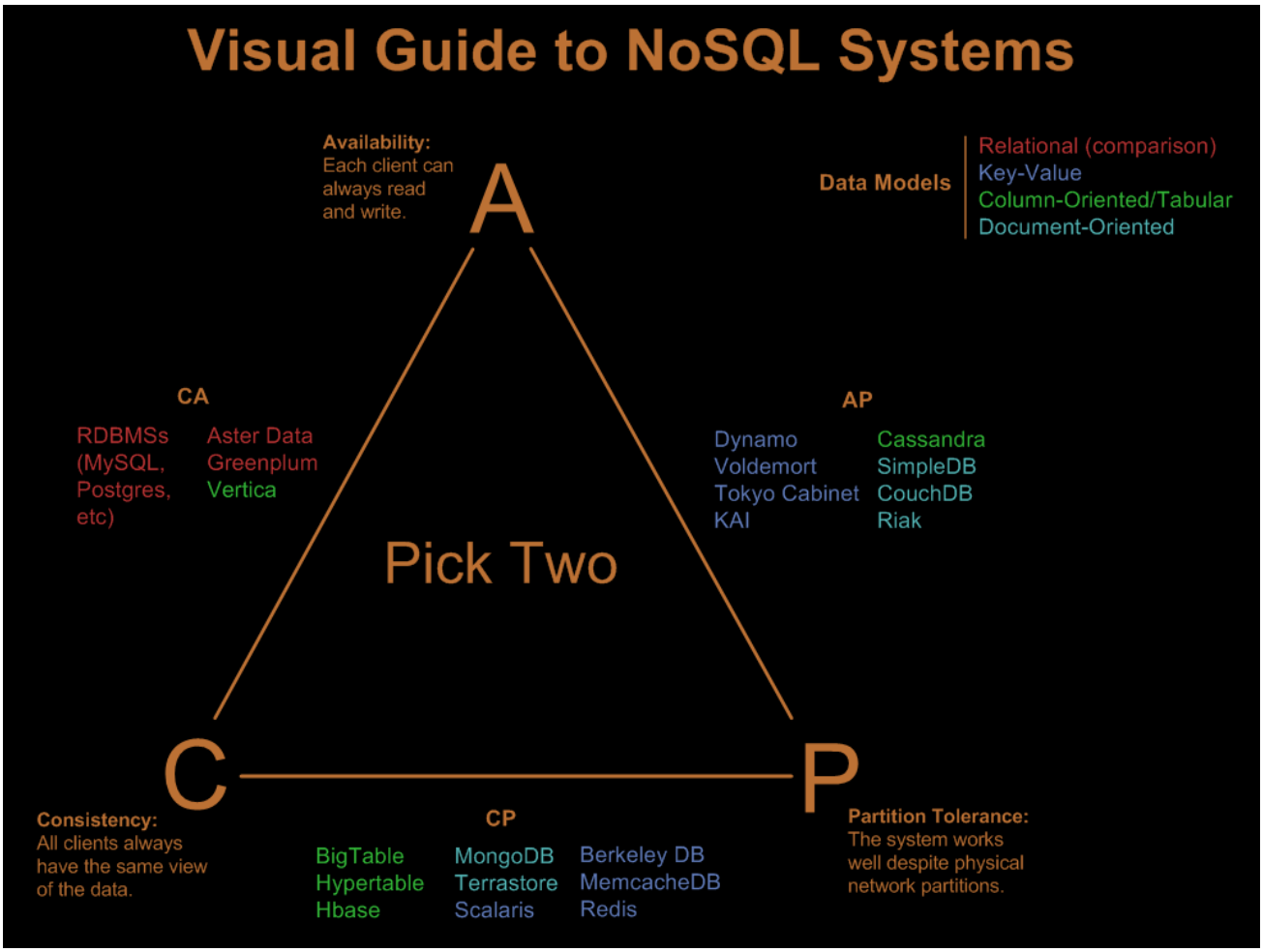



*Figure 5 Visualisation des différents SGBD (Hurst, 2010)*

## 2.5 Choix de la base de données NoSQL

### 2.5.1 Comparaison des outils de gestion des BD NoSQL

Le Tableau 1 résume les caractéristiques des principales bases de données NoSQL supportant les données géospatiales. CouchDB a beaucoup de points en commun avec MongoDB excepté que CouchDB est moins adapté pour les données fortement variables (Oussous et al. , 2015).

CouchDB assurera la disponibilité et la tolérance à la partition là où MongoDB assurera la disponibilité et la cohérence. Le choix de l'importance de ces caractéristiques est à effectuer selon les besoins de l'application. De prime à bord, dans les conditions de création du prototype dans ce travail, ni la disponibilité ni la cohérence semble avoir plus d'importances l'une que l'autre. Par contre MongoDB semble supporter des requêtes plus complexes et a l'avantage de supporter nativement le partitionnement (Lourenço et al., 2015).

*Tableau 1 Comparaison de système de BDD NoSQL géospatiales (de Souza Baptista et al., 2014)*

| | CouchDB | MongoDB | Neo4j | BigTable |
|---|---|---|---|---|
| **Basic Concepts** | Document-oriented | Document-oriented | Network-oriented | Column-oriented |
| **Indexing** | R-Tree<br>Only2D | 2D<br>2Dsphere | R-Tree<br>2D and partially 3D | B-Tree<br>Only2D |
| **Vector Data Types** | Fully | Fully | Fully Basic Types and Limited MultiGeometry Types | Fully Basic Types |
| **Topological functions** | Only<br>Within()<br>Contains() | Only<br>Within(Point)<br>Contains(Point) | Almost fully | Not supported |
| **Analysis and metric functions** | Only Distance() | Only Distance (Point) | Fully | Only Distance() |
| **Set functions** | Not supported | Only Intersection (Point) | Fully | Not supported |
| **Input/output Format** | Input:<br>.SHP<br>Output:<br>.KML<br>.CSV<br>.GeoJSON | Input:<br>GeoJSON<br>Output:<br>GeoJSON | Input:<br>.SHP<br>.OSM<br>Output:<br>SLD styled PNG | Input<br>.JSON<br>.KML<br>Output<br>.JSON<br>.KML |

MongoDB semble donc être plus approprié pour la gestion de nos données. L'utilisation de CouchDB n'est pas pour autant inintéressante, au contraire, elle offre bon nombre d'avantages simplement par ses objectifs différents (disponibilité au lieu de cohérence).

### 2.5.2 Présentation rapide de MongoDB

MongoDB est un système de gestion de base de données qui tire son nom de « *humongous* » qui signifie « énorme » en anglais (Sharma, 2015). Il est développé par la compagnie 10gen.company et est disponible en open source.

MongoDB possède une documentation très riche. Parmi celles-ci, un tableau compare toutes les caractéristiques et fonction du SQL avec MongoDB et offrant tous les équivalents. Ce fichier est disponible sur le lien suivant : https://docs.mongodb.com/manual/reference/sql-comparison/. Une base de données correspond pour MongoDB approximativement à un schéma, une table à une collection et un tuple à un document.

Voici une vue d'ensemble sur les caractéristiques principales de MongoDB (voir Figure 6). Ce système rassemble des caractéristiques du relationnel et du NoSQL. L'utilisation de MongoDB pourrait donc s'avérer plus facile pour des utilisateurs habitués au SQL.

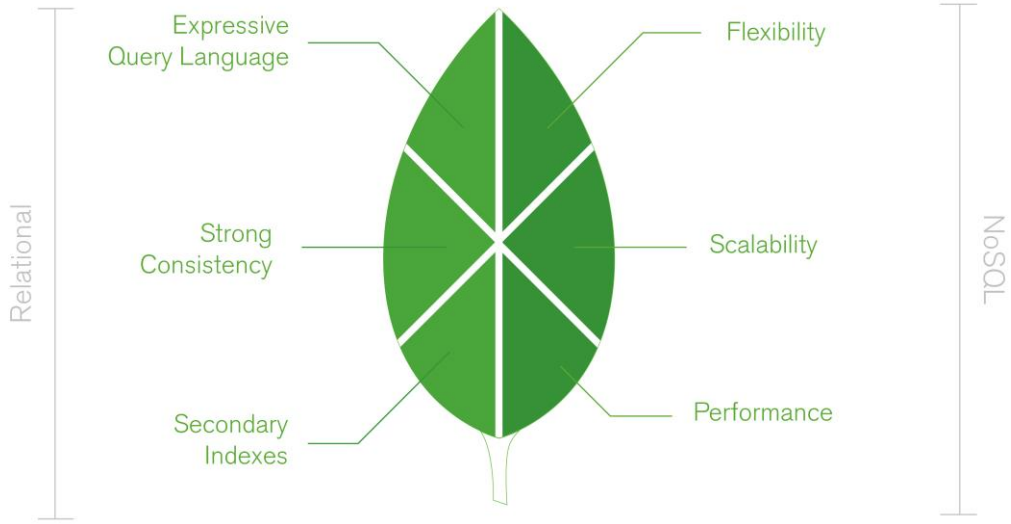



*Figure 6 architecture de connexion de MongoDB (MongoDB, s.d.)*

Il est considéré aujourd'hui comme un des systèmes maître esclave les plus performants qui offrent une sécurité de sauvegarde des données en cas de panne (structure des données dans les serveurs).

MongoDB utilise le format JSON mais en binaire (BSON) et offre la possibilité de schémas dynamiques (GeoJSON pour les données géospatiales). Cette utilisation offre un grand avantage car un bon nombre de langages de programmation utilise également ce format, ce qui offre un gain de temps conséquent.

Le format BSON permet une flexibilité des modèles, des relations mère-fille dans une modèle relationnel peuvent se retrouver dans un même document dans MongoDB grâce aux documents intégrés (MongoDB, 2015). Cette disposition est beaucoup plus intuitive que les schémas normalisés des modèles relationnels. Chaque document d'une collection peut varier les uns par rapport aux autres.

Le format GeoJSON permet de rassembler des données sous forme de points, de lignes ou de polygones. Par contre le format GeoJSON ne répond pas aux normes OGC (Open Geospatial Consortium) qui élaborent des normes internationales pour la gestion de données géographiques.

L'avantage de MongoDB par rapport à CouchDB (qui est également une BD orientée document) est que l'input peut s'effectuer en GeoJson et pas uniquement en SHP (voir Tableau 1). Ainsi, on peut également travailler sur des documents intégrés pour les tables d'attributs des couches géospatiales.

11 pilotes existent pour les langages les plus fréquents : Python, Java, .NET, PHP, … Ce qui réduit fortement le temps d'adaptation des programmateurs lors de la migration vers MongoDB. Ces caractéristiques permettent d'introduire des fonctions JavaScripts complexes (Sharma, 2015). CouchDB possède également plusieurs pilotes mais en moins grande quantité.

MongoDB n'a pas besoin d'extensions pour gérer les données géolocalisées. Il inclut des index spatiaux et possède un grand nombre de types d'index secondaires. L'index principal est un B-Tree. Il existe plusieurs types d'indices géospatiaux : 2D, 2D sphérique. Ils sont de type Quad-tree et permettent d'optimiser les requêtes sur les documents composés de points, polygones ou lignes dans

un espace 2D. Il n'existe à ce jour toujours pas d'index 3D pour MongoDB. MongoDB travaille dans le système de référence WGS84.

Un des désavantages de MongoDB est que pour les SIG, les interfaces sont trop lourdes ou non fonctionnelles, ce qui impose souvent la création d'applications pour la visualisation de données.

### 2.5.3 Quelques utilisateurs connus de MongoDB :

"*Comparative Analysis of MongoDB Deployments in Diverse Application Areas*" (Abraham, 2016) est un article qui énonce plusieurs grandes applications utilisant MongoDB et qui souligne les avantages de son utilisation. Les applications mentionnées sont entre autres Expédia, Metlife, Shutterfly, Google, Ebay, MTV, ect.

Les avantages soulignés sont la rapidité de prise en main pour la mise en place de MongoDB, sa capacité à rassembler des données issues de nombreuses sources différentes, l'augmentation des performances et la diminution des coûts, la facilité de manipulation de données hiérarchisées, la facilité de segmentation des données qui facilite leur migration.

L'article mentionne tout de même que pour les opérations de transaction et pour les données sensibles le SQL est maintenu pour conserver les caractéristiques ACID.

La plupart du temps, les organisations préfèrent utiliser MongoDB pour leurs nouvelles applications plutôt que de migrer des applications existantes (Abraham, 2016). Cela peut donc amener à supposer que l'importation d'une BDD classique vers une BDD MongoDB est laborieuse.

Le Tableau 2 reprend tous les exemples mentionnés et les caractéristiques de leurs applications MongoDB

*Tableau 2 Comparaison d'applications MongoDB existantes (Abraham, 2016)*

| Organization | Application Description | Data Storage/ Search Queries | Transactional Queries | Data Analytics | New App/ Migrated from | Reasons for deploying MongoDB |
|---|---|---|---|---|---|---|
| Expedia | Planning Vacation | MongoDB | SQL | SAS, Hadoop, Teradata | New App | Storing unstructured data, Managing evolving schema, Data analytics |
| Metlife | 360 degree customer view | MongoDB | SQL | Hadoop | New App | Scaling, Storing structured &unstructured data, Data analytics |
| Shutterfly | Web and Mobile photo services | MongoDB | SQL | Teradata | Oracle | Scaling, Storing structured &unstructured data, Data analytics |
| Google | Back end for Cloud launched applications | MongoDB | SQL | - | NA | High performance, reliability and cost-effective platform for developing and deploying applications |
| ebay | Search suggestion list, meta data storage | MongoDB | SQL | Hadoop | New App | Speedy Queries |
| MTV | Centralized Content Management | MongoDB | NA | - | Multiple RDBMS | Scaling, Storing structured & unstructured data, Managing evolving schema, Speedy Queries |
| Craiglist | Archival of Classifieds | MySQL, MongoDB | SQL | Hadoop | My SQL | Managing Evolving schema, Scaling, Storing structured &unstructured data, Speedy Queries |
| Adhaar | Storing and searching, demographic biometric data and images | SQL,MongoDB, Hbase | NA | Hadoop | New App | Scaling, Managing evolving schema, Data analytics, Storing unstructured data set which includes biometric data and images |

## 2.6 Comparaison des performances par rapport au relationnel

Beaucoup de comparaisons entre les SGBDR relationnels et NoSQL existent sous forme de blog mais peu existent dans la littérature scientifique. Les études ont surtout exploré les performances respectives des bases de données SQL et NoSQL. Voici quelques cas de comparaisons dans la littérature scientifique.

« *A comparative study of performance analysis of MongoDB and PostGIS/PostGreSQL database systems* » (Sarthak et al., 2015) est une étude qui compare les performances de deux bases de données géospatiales, PostGIS et MongoDB. Ces deux BDD sont open source. PostGIS est une extension spatiale à PostgreSQL.

L'intersection de ligne et la recherche de point dans une aire sont les deux requêtes de base utilisées pour comparer les deux modèles.

Dans l'espace plane, PostGIS possède un grand nombre d'opérations sur la géométrie, mais MongoDB, lui ne supporte que les requêtes de présence contenue dans des cercles, boîtes et polygones. Dans l'espace sphérique, MongoDB peut effectuer des requêtes sur des objets GeoJSON, tandis que Postgis

ne permet d'effectuer nativement que les intersections et la recherche d'éléments contenus dans une zone.

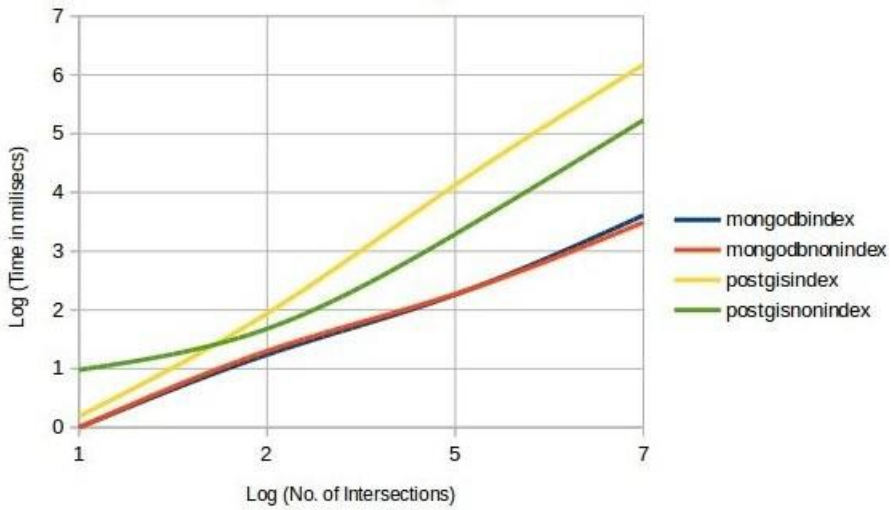



*Figure 7 Graphe des intersections par rapport au temps (Sarthak et al., 2015)*

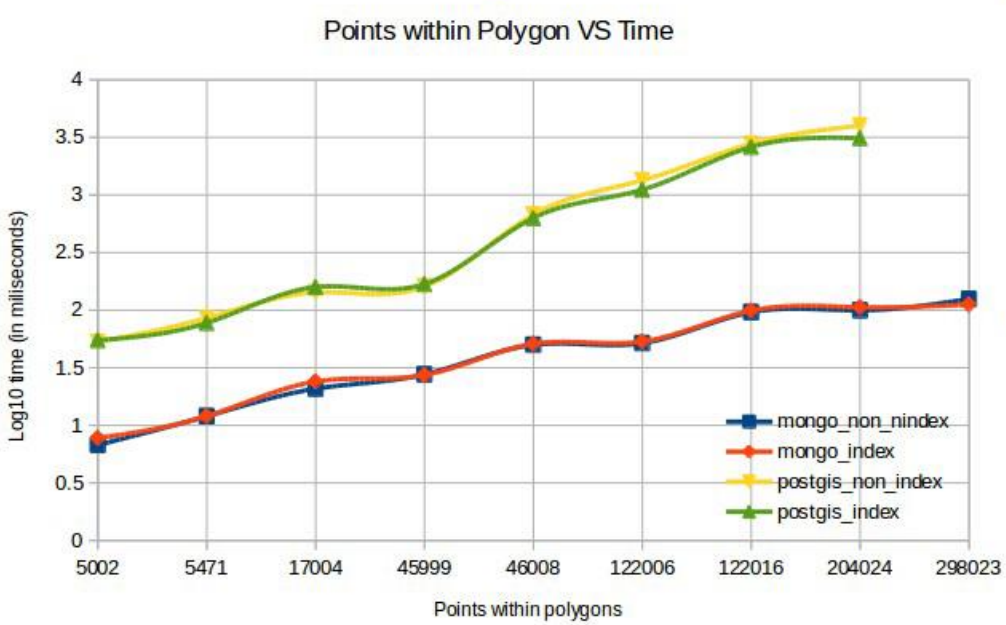



*Figure 8 Graphe du temps de recherche de point dans une zone par rapport au temps(Sarthak et al., 2015)*

En observant la Figure 7 et la Figure 8, on peut conclure que les performances de MongoDB sont bien plus importantes (25 fois et de manière exponentielle) que PostGIS lorsque la taille de la base de données augmente. L'apport de l'index est plus avantageux pour PostGIS. Il faut être vigilant car même si l'utilisation d'un index permet souvent l'accélération du temps de réponse à une requête, son application connait également des points négatifs. En effet, elle augmente les opérations d'écriture et la quantité de ressources utilisées (MongoDB, 2015).

Cet article démontre donc les capacités de MongoDB pour les applications SIG mobiles et web avec un usage multiutilisateurs.

Une autre étude compare une base de données SQL serveur avec une BD MongoDB, mais en utilisant des données structurées et en quantité modeste pour pouvoir définir si le NoSQL serait également efficace dans ces cas-là. Cet article s'intitule "*Comparing NoSQL MongoDB to an SQL DB"* (Parker et al. , 2013). Cette étude compare les BD sur base de trois manipulations : l'insertion, la mise à jour et une opération de sélection qui requiert un joint.

L'agrégation est la fonction qui diffère le plus entre les deux systèmes car l'agrégation dans MongoDB requiert la fonction « *Map/Reduce* » qui complique fortement le code. MongoDB surpasse le SQL dans tous les cas qui utilisent une clé primaire. Cela serait dû à l'index fournit par MongoDB qui est plus efficace que celui de SQL server.

En ce qui concerne les sélections, MongoDB est plus rapide pour les requêtes simples et encore bien plus performant pour les requêtes plus complexes. Ce serait à nouveau causé par son index et également la manière qu'utilise MongoDB pour gérer sa mémoire. Une exception apparait pourtant lors de l'usage de « *Map/Reduce* » qui augmentent significativement le temps de réponse.

SQL est donc préférable uniquement dans le cas de données ne possédant pas de clé ou demandant une agrégation. De plus, cet article n'aborde pas les bases de données distribuées, qui sont également le point fort de MongoDB.

On peut donc en conclure que si les données ne connaissent pas de structures fortes, MongoDB reste le premier choix entre les deux options.

Voici comme dernier exemple un article qui compare la création d'une application mobile avec MongoDB et PostgreSQL. Cet article, *"NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL"* (Fotache et al., 2013), permet de prouver que MongoDB correspond bien également à ce type d'applications (pour le client et pour le serveur) et souligne que quelques unes de ses caractéristiques qui sont communes avec les SGBD relationnels.

Le NoSQL possède donc bien des avantages qui méritent d'être étudiés concernant la gestion de bases de données géospatiales. En vue des données utilisées, il semble que les modèles orientés documents sont les plus intéressantes à appliquer, et en particulier le SGBD MongoDB, grâce à ses performances, sa gestion de grandes quantités de données, sa gestion de données variées et sa rapidité de prise en main.

# 3. CONTEXTE ET HYPOTHÈSE

## 3.1 AIDE

Ce travail est effectué à partir de données procurées par l'AIDE. L'AIDE est l'Association Intercommunale pour le Démergement et l'Epuration des communes de la province de Liège. Elle rassemble 84 communes (AIDE, 2016). Elle agit pour la protection de la région contre les inondations provoquées par les affaissements miniers et gère les réseaux d'assainissement des eaux usées.

Ce travail n'est pas un résultat d'une demande de l'AIDE, mais une proposition du département de géomatique pour une utilisation des données dans le cadre d'un mémoire. Après plusieurs réunions, l'AIDE a fait parvenir un set de données susceptibles d'être utilisé dans une base de données ainsi qu'un échantillon d'une base de données relationnelle existante.

La base de données actuelle est hébergée par Access, en ce qui concerne la gestion du réseau d'égouttages. Un très grand nombre de type de données différentes est disponible pour des régions variées. Les données utilisées pour ce travail seront sélectionnées selon leur contenu et les richesses qu'elles peuvent apporter à notre analyse. Seul un petit échantillon de données sera utilisé, car le but n'est pas de mettre en place une base de données fonctionnelle pour une utilisation immédiate par l'AIDE. Bien au contraire, le but de ce travail est l'élaboration d'un prototype jetable pour explorer les possibilités et les fonctions offertes par une base de données NoSQL et non une maquette fonctionnelle. L'intention ici est d'explorer la piste du NoSQL pour pouvoir éventuellement considérer son implémentation par la suite.

Ce travail ne traite donc pas une demande de l'AIDE, mais une certaine collaboration est tout de même mise en place pour comprendre le fonctionnement et les besoins de la société afin de pouvoir créer un prototype vraisemblable.

## 3.2 Type de données

L'ensemble des données reçues est constitué par des données concernant les installations d'assainissement des eaux, l'analyse de l'état d'égouts, les analyses de permis demandant un système d'épuration individuelle ou une dérogation, ainsi que le levé des ouvrages d'égouttage existants.

Le dossier de données sélectionné est celui du cadastre et de l'égouttage. Ces données sont disponibles pour 4 zones différentes : Oreye, Juprelle, Liège et Butgenbach. Une seule zone est sélectionnée pour la mise en place du prototype et c'est la zone d'Oreye qui sera retenue.

Ce dossier a été retenu car il contient une variété intéressante de types de données. Il rassemble des informations sur le réseau d'égouttages contenu dans une base de données Access, des coordonnées dans un fichier Excel, des fiches PDF reprenant ces caractéristiques, des photos des regards du réseau d'égouttage en Jpeg, ainsi que des plans PDF.

Ces données sont choisies d'abord pour un critère indispensable à la réalisation de ce travail : des données géolocalisées. De plus, la variété de format de données permettra l'analyse de plusieurs scénarios d'importation. Certaines caractéristiques du réseau de tuyaux attirent également l'attention,

chacun des tuyaux se rapportent à deux regards (un en amont et un en aval), mais un regard peu correspondre à plusieurs tuyaux, ce qui offre une irrégularité dans la structure de données. Une fiche résumant les caractéristiques d'un regard est disponible en Annexe 9 pour avoir une vue d'ensemble sur les données.

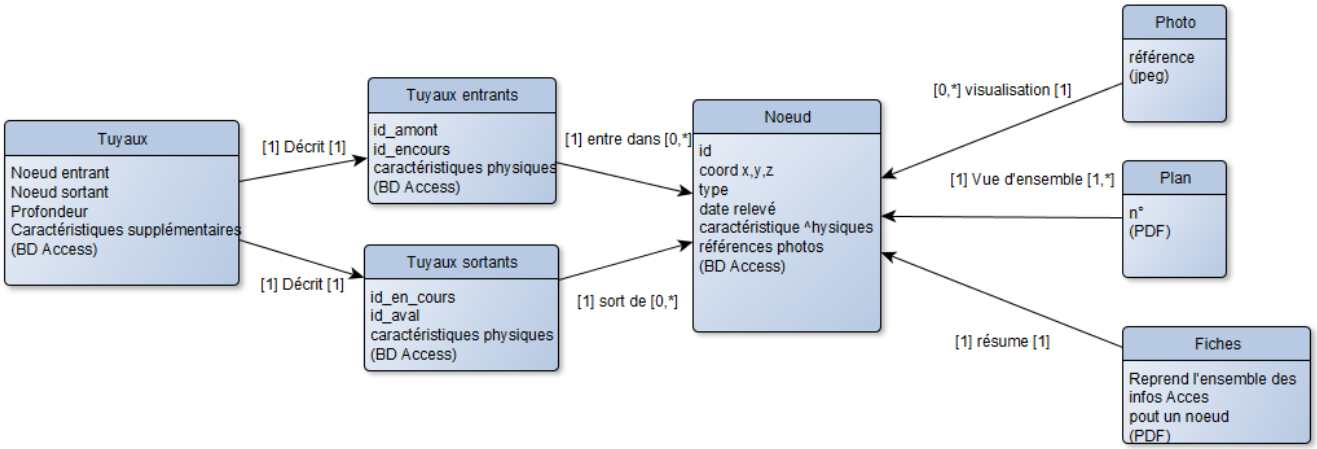



*Figure 9: Schéma des relations entre les données*

Les données d'Oreye sont complètes, chaque dossier contient un nombre représentatif de données, contrairement à d'autres dossiers. Voici une visualisation de la localisation des regards de ce dossier.

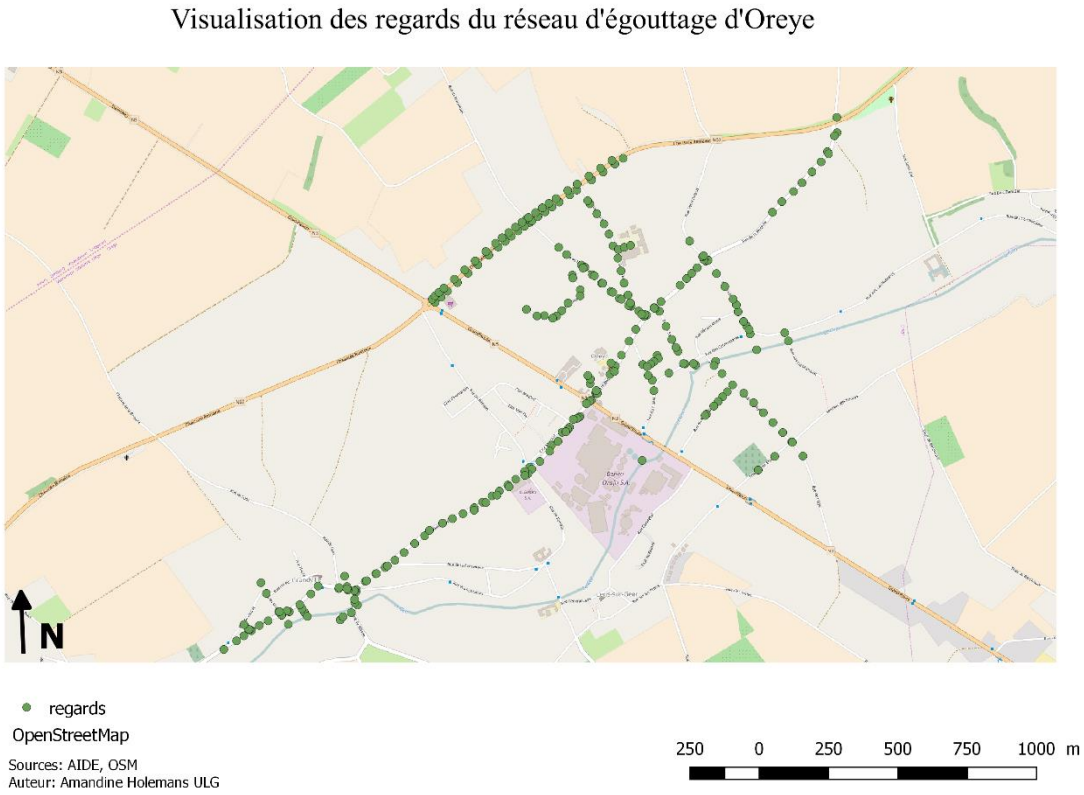



*Figure 10: Visualisation des regards*

Les coordonnées sont issues d'un levé topographique. Elles sont projetées en Lambert 72 et possèdent un attribut Z.

Ces données procurées par l'AIDE ont été complétées par des données provenant du cadastre. Cet ajout permet d'effectuer des requêtes sur les numéros de parcelles cadastrales.

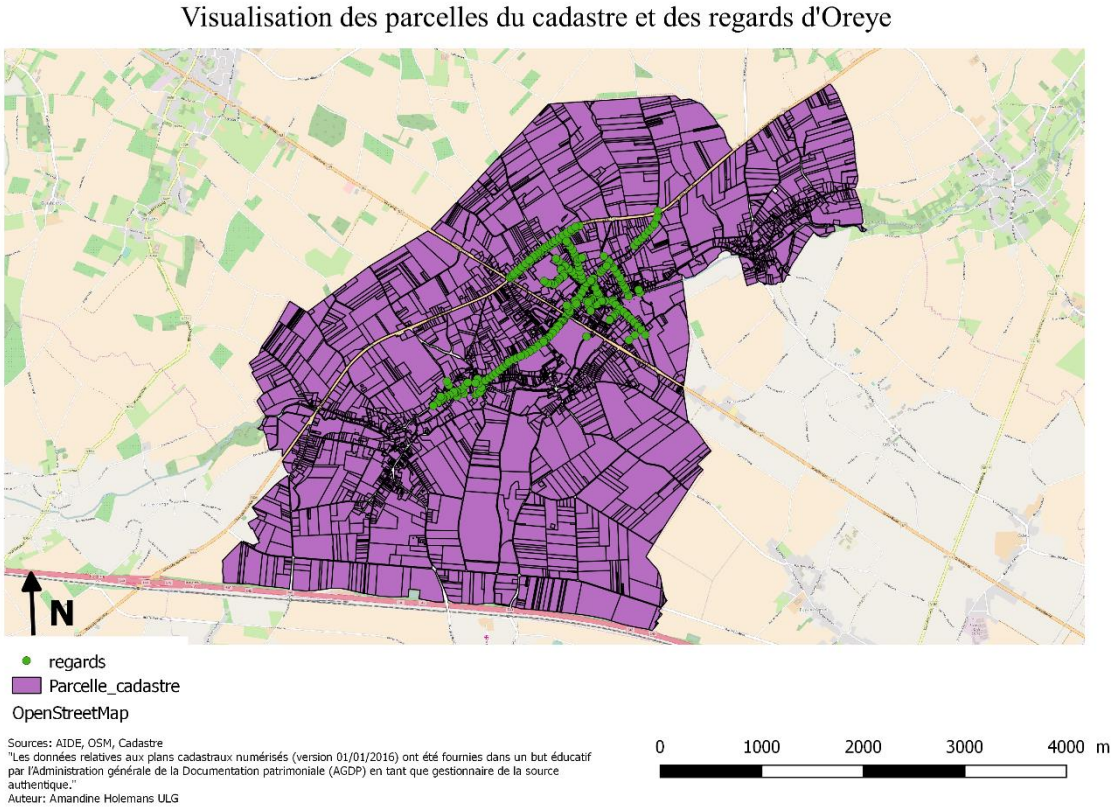



*Figure 11: Visualisation des parcelles du cadastre*

## 3.3 Correspondance avec le modèle orienté document

Après l'observation des données, le modèle de données NoSQL orienté document semble bien le plus adéquat. Premièrement, il offre la possibilité d'effectuer des requêtes sur les champs des documents. Ensuite, son type de schéma convient également très bien à une phase d'exploration en vue de la création d'un prototype. En effet, le schéma est évolutif et peut progresser fortement au cours du temps grâce aux capacités des documents d'en intégrer d'autres.

MongoDB sera utilisée comme SGBD NoSQL. C'est un système répandu et populaire, il est donc intéressant à tester et facile à prendre en main grâce aux aides de la grande communauté d'utilisateurs. Ces données proviennent de plusieurs sources, c'est une caractéristique qui convient grandement à MongoDB.

Une base de données NoSQL clé valeur aurait également pu être envisagée si les recherches n'impliquaient que l'utilisation des identifiants. Cette hypothèse est envisageable grâce à l'existence des fiches de rapport. Une requête renverrait vers un identifiant qui serait relié à une fiche PDF. Cette configuration perd de l'avantage à cause de la mauvaise capacité de ces systèmes à gérer les relations ainsi que les requêtes spatiales.

Les SGBD NoSQL orienté colonnes sont également à mettre de côté à cause de leurs mauvaises capacités à gérer les relations. Ces systèmes sont fortement limités pour les éléments géospatiaux et leur interrogation.

Par contre, un type de base de données NoSQL qui a également attiré l'attention après l'étude des données de l'AIDE est le modèle orienté graphe. Les dispositions des tuyaux constituent en effet un réseau, ce qui est la spécialité des SGBD orientés graphe. Il offre également de bonnes solutions pour les requêtes spatiales. Cette option n'a pas été retenue car elle est considérée comme un système relativement complexe à mettre en place, ce qui peut compliquer les étapes d'un travail d'exploration.

L'implémentation de la base de données sera donc effectuée via MongoDB, un système de gestion de base de données NoSQL orienté document.

## 3.4 Type de demande

L'objectif final dans la conception de la base de données est la création d'un prototype jetable. Le système doit être appliqué afin de dresser un modèle de base de données NoSQL MongoDB et un exemple démonstratif des avantages et des capacités de ces nouvelles bases de données.

Les types de données envoyées par l'AIDE et sélectionnées ici sont considérées comme importantes et représentatives de la BD. Leur utilisation est donc capable d'illustrer les capacités d'une base de données réelle.

Les types de requêtes ont été établies en fonctions des champs que l'AIDE a désignés comme éléments clés.

## 3.5 Hypothèse

Cela nous ramène donc à la question de recherche. Comment introduire et gérer efficacement les données documentaires sélectionnées en parallèle avec l'infrastructure géomatique existante au sein de l'AIDE ?

Le NoSQL semble une option envisageable, et MongoDB un outil adéquat pour la mise en place d'un prototype de base de données gérant les données sélectionnées.

Cette option semble pouvoir être implémentée en restant dans le domaine de l'open source et ses requêtes semblent pouvoir interroger efficacement des données géospatiales.

L'hypothèse à vérifier dans ce travail sera donc :

« L'implémentation d'un SGBD NoSQL orienté documents, en particulier MongoDB, permet de prendre en charge les données non structurées de l'AIDE et les fonctionnalités de MongoDB sont suffisantes et efficaces pour effectuer les traitements pressentis sur ces données. »

# 4. DÉVELOPPEMENT

## 4.1 Notions de base de MongoDB

### 4.1.1 Introduction

Utiliser MongoDB demande un changement de la vision d'une base de données. Structurer les données d'une base de données NoSQL comme une base de données relationnelles reviendrait à passer à côté de toutes les innovations et avantages de celles-ci. Ainsi, avant de la mettre en place, il est important d'étudier toutes les particularités d'une base de données MongoDB.

### 4.1.2 Schéma

La conception du schéma doit prendre en compte les particularités de l'application elle-même. Il faut regarder le nombre de lectures et l'importance de chacune. Les requêtes et la fréquence des mises à jour doivent également être étudiées bien avant la conception du modèle.

#### *4.1.2.1 Document*

Le modèle de données de MongoDB est entièrement basé sur le document. Le document est représenté par un fichier BSON. Les caractéristiques de ces fichiers permettent d'introduire des documents intégrés, ce qui permet d'éviter les jointures ou transactions lourdes. Les documents peuvent donc être comparés à plusieurs lignes d'une table d'une BD relationnelle.

Des règles générales aussi bien structurées que pour le passage du modèle conceptuel au modèle physique des SGBDR ne connaissent pas leur pareil dans ce cas-ci. Le schéma se crée au cas par cas en fonction de l'application mise en place. La flexibilité des documents est un grand avantage car il permet de rassembler tous les types de modèles de bases de données. Elle permet également de revenir sur le schéma d'un document après sa création sans coût trop important.

MongoDB assure la conformité ACID (atomicité, cohérence, isolation et durabilité) au niveau du document. En effet, toutes les données comprises dans un document peuvent être rassemblées en une seule opération. Ainsi, on peut stocker toutes les données dans un seul document pour appliquer les caractéristiques ACID à l'ensemble de la base de données.

Cette structure rend la maintenance beaucoup plus facile et moins pénible. Elle remplace la fonction assez lourde des SGBDR « *Alter table* », car un document peut être rapidement modifié sans impacter sur les autres documents de la collection (Abraham, 2016).

Cette solution semble miraculeuse en rassemblant les caractéristiques du relationnel et du NoSQL en un seul système, mais regrouper toutes les données dans un seul document peut forcer à réaliser des compromis pour d'autres applications. Un certain modèle de collection peut correspondre à une application et pas à une autre. Cette configuration peut également amener le document à dépasser la limite de taille (16Mo). Le format document peut aussi offrir des désavantages dus à son absence de structure, les requêtes peuvent s'avérer compliquées car il faut savoir exactement quelle partie du document il faut interroger car celles-ci varient dans toute la collection (Parker et al., 2013).

#### *4.1.2.2 Collection*

Une collection est l'ensemble des documents de la même famille. Il n'y a pas de structure imposée pour les documents d'une même collection, ils peuvent tous varier entre eux.

Il existe des collections typiques appelées « collections limitées », créées à l'avance avec une limite de taille. Quand l'ajout de documents provoque le dépassement de la taille limite, ce sont les plus vieux documents qui vont être supprimés (Chodorow, 2013). Cette configuration offre une gestion particulière de la mémoire.

#### *4.1.2.3 Documents intégrés*

La Figure 12 illustre la correspondance d'un modèle MongoDB avec un modèle relationnelle et comment les données reliées sont transformées en documents intégrés.

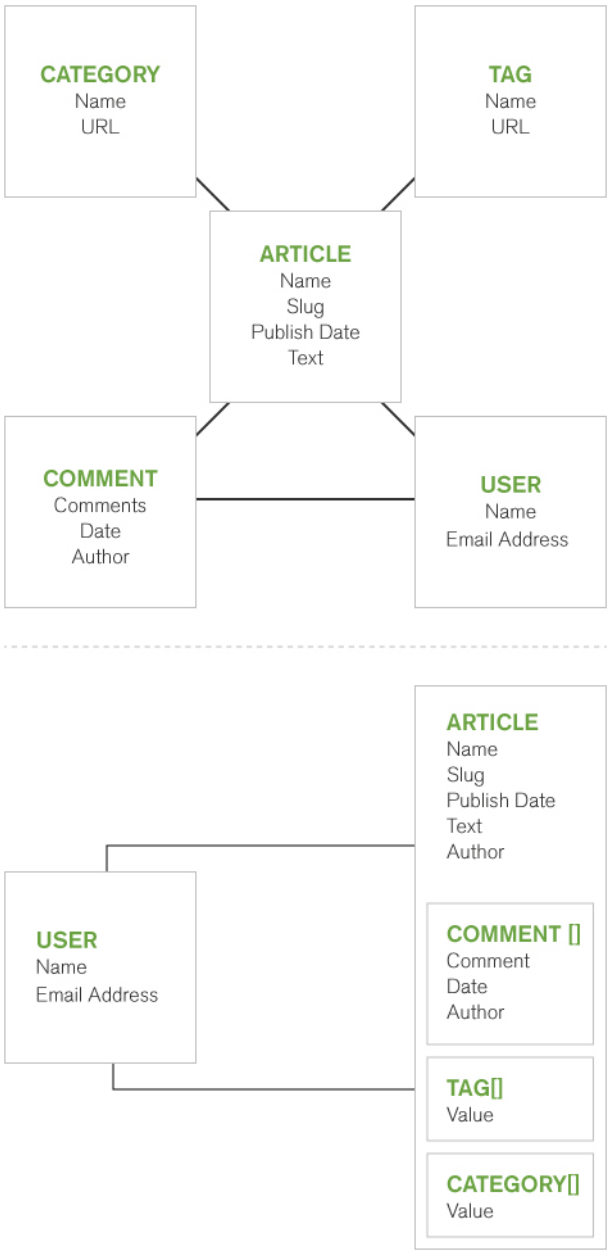



*Figure 12 Jointure préliminaire des données d'un SGBDR vers un document MongoDB (MongoDB, 2015)*

Les relations un à un ou un à plusieurs correspondent parfaitement au type de document intégré (MongoDB, 2015).

Les documents ne doivent pas être intégrés dans trois cas (MongoDB, 2015) :

- Si le document à intégrer demande beaucoup moins de lecture que le document principal. Cela ne ferait qu'augmenter la mémoire nécessaire pour les opérations fréquentes.
- Si une partie du document est fréquemment mise à jour contrairement au reste du document.
- Si la taille totale du document dépasse la limite des 16 Mo imposée par MongoDB

Le format en document de MongoDB permet d'accéder à l'ensemble des données sans joint, ce qui augmente la rapidité de réponse. De plus, chacun des documents est donc indépendant et peut être distribués horizontalement (MongoDB, 2015). S'il faut normaliser le modèle, le référencement existe pour faire le lien entre deux documents. Pour des situations pouvant trouver un avantage à une normalisation, MongoDB offre tout de même une fonction d'agrégation (moins puissante que le joint des bases de données relationnelle).

Il apparait clairement que son utilisation diminuerait les performances, mais MongoDB offre la possibilité d'un joint contrairement à la plupart des bases de données NoSQL. Le référencement est le plus souvent appliqué sur le champ _id (par la commande `$lookup`) dans le document de référence. L'usage du référencement est conseillé dans plusieurs cas (MongoDB, 2015):

- Si l'intégration n'a vraiment pas l'avantage sur le référencement
- Si l'objet est référencé de plusieurs sources
- Pour les relations plusieurs à plusieurs
- Pour les jeux de données volumineux et hiérarchiques

Il existe plusieurs outils appelés ETL *(« extract, transform and load »)* qui sont capables d'assister la migration d'une base de données relationnelle vers MongoDB (MongoDB, 2015). Ces assistants sont surtout utiles lors du déplacement de grandes quantités de données et sont compris dans le package MongoDB Entreprise, qui lui, n'est pas open source.

### 4.1.3 JSON et GeoJSON

Le format JSON est un format de fichiers JavaScript léger.

JSON est un format lisible par l'homme, semblable au XML mais plus compact, vu l'absence de balises. Voici un exemple de fichier JSON.

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Les différents éléments pouvant être présents dans un document JSON sont soit un objet, soit un tableau qui eux-mêmes peuvent rassembler plusieurs types de valeurs. Voici la schématisation de ces éléments.

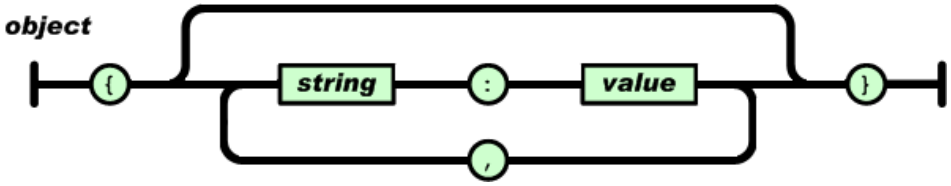



*Figure 13 Schématisation d'un objet JSON (JSON, s.d.)*

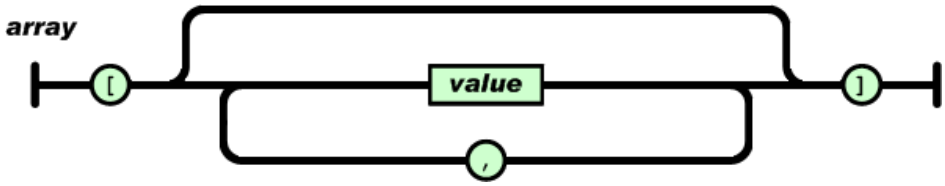



*Figure 14 Schématisation d'un tableau JSON*

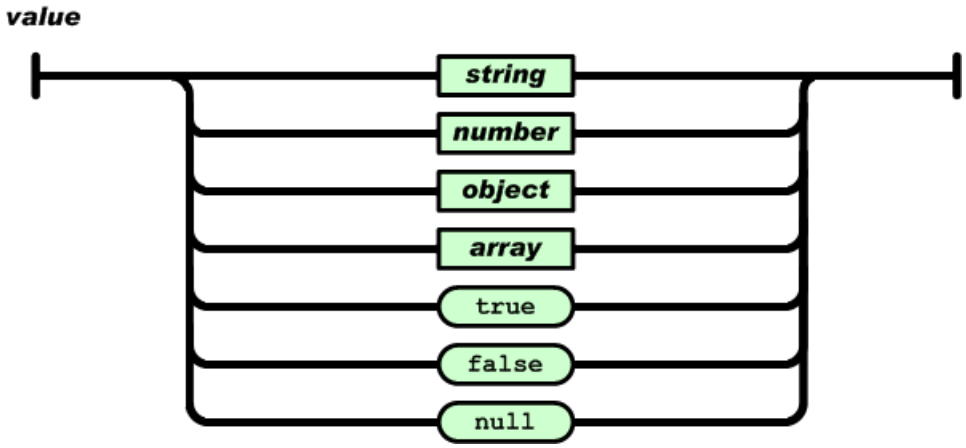



*Figure 15 Schématisation des différents types de valeur contenu dans un document JSON*

L'importation peut être effectuée à partir de fichier JSON ou GeoJSON qui seront stockés dans la BD sous format BSON. Le BSON est une extension au JSON, il permet l'utilisation d'autre type de données comme les entiers, les types longs, les dates et les virgules flottantes.

Il faut souligner que l'utilisation de noms de champs trop long diminue les performances de la BDD étant donné que celui-ci est répété pour chaque document de la collection.

### 4.1.4 Modélisation

La modélisation n'est pas indispensable car le modèle ne connait pas de schémas fixes, mais son utilisation offre une meilleure compréhension ainsi qu'une vision des cardinalités.

### 4.1.5 Sélection de l'index

La sélection de l'indice se réalise une nouvelle fois en fonction du type et de la fréquence des requêtes. La création des index consomme beaucoup de ressources, l'utilisation de chaque index doit donc être justifiée.

Même si l'utilisation d'un index n'est pas utile pour le nombre de données manipulées dans le cas de ce travail, il est toujours intéressant de prévoir l'utilisation de ceux-ci dans le cas d'une croissance de la base de données.

L'index principal de MongoDB est un index B-Tree. Ce type d'index est déjà utilisé dans les BD relationnelle. Voici une représentation de l'indexation des informations par l'index B-Tree pour une meilleure compréhension.

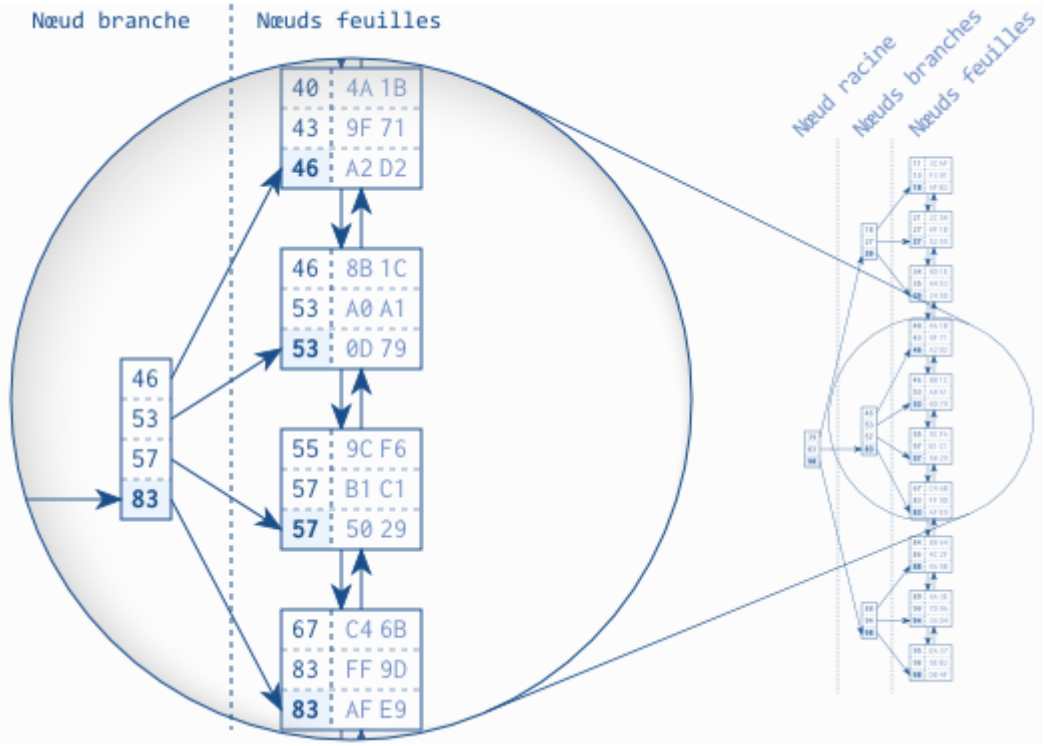



*Figure 16 Visualisation de la structure d'un index B-Tree (Winand, s.d.)*

D'autre index secondaires pouvant être introduits par l'utilisateur sont supportés nativement par MongoDB. Les indexes de MongoDB correspondent pour la plupart à ceux des BD relationnelles (MongoDB, 2015).

Les index secondaires spatiaux sont les seuls qui vont être utilisés dans le cadre de ce travail. Ce sont des index Quadtree qui sont utilisés dans ce cas (Figure 17).

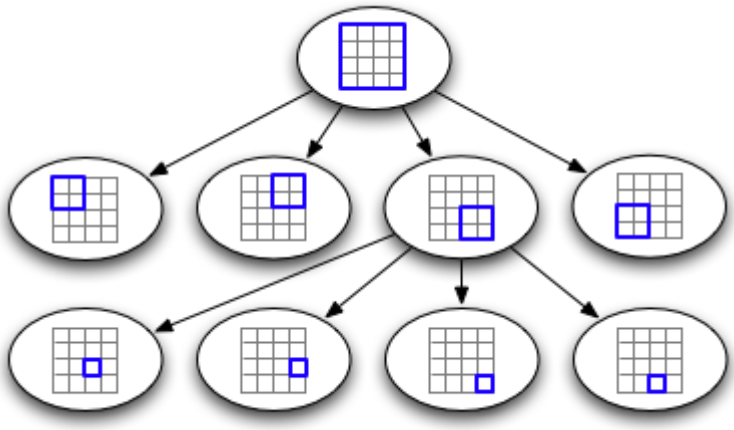


*Figure 17 Visualisation indexation Quadtree (Johnson, 2009)*

Voici d'autre index secondaires pour information : index composé, index de durée de vie, index syntaxiques, index haché, etc.

#### *4.1.5.1 2d*

Cet indice est utilisé pour des entités disposées sur une surface plane (cartes de jeux vidéo, données chronologiques, etc.). Ce type d'index ne supporte pas le format GeoJSON et ne peut indexer que des points.

#### *4.1.5.2 2dsphere*

C'est index le plus utilisé, il peut gérer les systèmes géodésiques, et gère donc les latitudes et longitudes. Ces index permettent l'utilisation de format GeoJSON. Voici la procédure pour créer un index 2d sur le champ géospatial.

```
db.collection.createIndex({« geometry » : « 2dsphere»})
```

Le « 2Dsphere » est l'index qui correspond le plus à notre utilisation de données spatiales, dû à l'espace sphérique et à l'utilisation du GeoJSON et de tous ses avantages.

### 4.1.6 Validation de documents

Les schémas dynamiques de MongoDB ont beaucoup d'avantages, mais pour une bonne qualité de données, il faut continuer à les contrôler. La validation de documents peut être utile lorsqu'il se trouvent des applications en aval qui ont des exigences particulières. MongoDB offre la possibilité d'effectuer ces vérifications au sein même de la BDD. Par exemple : vérifier la clé, si le type de valeur est correct, etc.

### 4.1.7 « *Sharding* » *ou* distribution horizontale

Contrairement aux bases de données relationnelles, la distribution horizontale est installée automatiquement dans le système de gestion de base de données. Ce qui constitue déjà une complexité en moins pour le développeur. Elle est totalement invisible pour l'application, car elle n'a aucun impact sur celle-ci. C'est un routeur de requêtes qui va rediriger les requêtes concernant les clés de partitionnement vers les bons « *shards* ».

Le « *sharding* » est également utilisé pour la réplication. MongoDB est capable de ce qui est appelé « auto-guérision » en sauvegardant plusieurs copies des données. Les opérations de lecture et d'écriture ont uniquement accès au premier réplica. Ainsi, si une erreur a lieu, un des autres réplicas secondaires peut devenir le réplica principal. Cette caractéristique lui permet également d'être capable de maintenir une forte cohérence en dirigeant toutes les opérations de lectures vers les serveurs primaires (MongoDB, 2015).

Le « *sharding* » ne sera pas abordé dans ce travail, étant donné que l'objectif est la création d'un prototype jetable mis en place sur une seule machine. Le partitionnement horizontal concerne surtout les grandes quantités de données et ne fonctionne que quand le système de gestion de la base de données a accès à plusieurs machines pour pouvoir distribuer l'information.

### 4.1.8 Sécurité

La sécurité est un point important à aborder pour les systèmes de gestion de bases de données NoSQL, et cela concerne également MongoDB. En effet, un rapport du 13 janvier 2017 énonce les bonnes pratiques pour protéger ses données et cela en réaction à des attaques sur des BD MongoDB (Nilsson, 2017).

Une solution est offerte par MongoDB Enterprise Advanced qui est une version payante de MongoDB. Ce package offre entre-autres de défendre, détecter et contrôler l'accès aux données.

La documentation de MongoDB contient beaucoup d'informations sur les précautions à prendre pour sécuriser ses données qui semblent devoir être suivies à la lettre pour être entièrement efficaces, vu le rapport publié.

En ce qui concerne la simple authentification, voici la marche à suivre pour créer un identifiant et un mot de passe pour l'utilisation d'un BD.

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Et voici comment se connecter une fois l'identifiant créé :

```
use admin
db.auth("myUserAdmin", "abc123" )
```

Une fois encore, la question de sécurité ne sera pas abordée dans ce travail. Vu la complexité, des travaux entiers pourraient être dédiés à cette problématique et ne concerne pas notre cas de création d'un prototype.

### 4.1.9 GridFs

#### *4.1.9.1 Définition*

C'est un mécanisme qui permet de stocker des fichiers binaires importants jusqu'à 2GB par fichier (Chodorow, 2013). GridFS divise la quantité de données dans plusieurs documents et collections particuliers.

Il faut tout d'abord savoir que son utilisation présente des désavantages. Premièrement, elle diminue les performances : aller chercher un fichier via MongoDB ne sera pas aussi rapide que d'aller directement le chercher dans le système de fichiers. Ensuite, les documents peuvent seulement être modifiés par la suppression ou la sauvegarde car ils sont stockés à plusieurs endroits et MongoDB ne sait pas les bloquer tous ensembles pour permettre une modification plus importante.

GridFS est surtout utile quand on utilise des fichiers larges qui demandent un accès non fréquent. Son utilisation permet d'éviter l'utilisation d'un autre outil pour le stockage de fichiers.

GridFS a un avantage quand le système de fichiers limite le nombre de fichier dans le répertoire, GridFS lui n'a pas de nombres limites. De plus, si le but est d'accéder à uniquement une petite partie du fichier sans en ouvrir l'entièreté, GridFS est un bon outil. C'est le principe du streaming, c'est lire un fichier sans en télécharger l'entièreté. Enfin, si on veut que les fichiers et méta-datas profitent automatiquement de l'auto-guérison et de la distribution horizontale implémentés dans le système, GridFS doit être utilisé (MongoDB, s.d.).

#### *4.1.9.2 Mongofiles*

Mongofiles est l'outil de commande le plus pratique pour utiliser GridFS. Il permet de manipuler et effectuer des requêtes sur les fichiers contenus dans GridFS.

Voici les commandes principales :

*Tableau 3 Commande principales de mongofiles*

| | |
|---|---|
| List | Fait l'inventaire des fichiers stockés dans GridFS |
| Search | Recherche un fichier avec une chaîne de caractères |
| Put | Copie un fichier du système de fichier au stockage de GridFS |
| Get | Copier un fichier de GridFS vers le système de fichier par le nom de l'objet |
| Get_id | Copier un fichier de GridFS vers le système de fichier par l'id de l'objet |
| Delete | Supprime le fichier par son nom |
| Delete_id | Supprime le fichier par son id |

Voici un exemple pour le type de commande classique de mongofile :

```
mongofiles -d records list
```

#### *4.1.9.3 Discussion sur l'utilisation de GridFS*

L'utilisation de GridFS provoque un grand nombre de discussions. Car utiliser GridFS signifie stocker les fichiers dans la base de données, ce qui coûte plus cher que le Cloud. De plus, cela provoque des problèmes de performances comme mentionné plus haut. Pourtant, GridFS est souvent conseillé comme solution aux questions posées par la communauté.

Certains conseillent de plutôt stocker sous une URL et de ne jamais stocker d'images dans une base de données. D'autres ont des conseils pour les images légères, qui consistent en, soit, stocker le chemin relatif du fichier, soit utiliser GridFs et agréger les deux collections que créées la fonction, pour éviter d'effectuer trop de requêtes vers des collections différentes (Menge, 2015).

L'option de GridFS est donc malgré tout intéressante et il est important de s'y arrêter pour l'étudier.

## 4.2 Installation MongoDB et choix des outils

### 4.2.1 Version

La version installée de MongoDB est la version 3.4.2, mais des mises à jour mineures sont disponibles chaque mois. La dernière version en date est la version 3.4.7 mise en ligne depuis le 8 août 2017.

Plusieurs produits sont offerts par MongoDB et c'est le pack « Community Server » qui sera téléchargé et installé. Il existe également une version d'entreprise et une version professionnelle qui sont toutes les deux payantes et qui offrent des avantages par rapport à la version de la communauté qui correspond uniquement à un but de développement.

### 4.2.2 Quel pilote ?

MongoDB fonctionne sur plusieurs pilotes, créés par des équipes expertes dans le langage utilisé et connaissant les demandes et préférences des utilisateurs. Ainsi, les capacités déjà acquises dans un certain langage de programmation peuvent être utilisées pour la gestion des données du système MongoDB. Voici les différents langages pris en compte : Java, .NET, Ruby, Node.js, Perl, Python, PHP, C, C++, C#, Javascript et Scala. Il faut également savoir qu'en dehors de l'entreprise MongoDB, la communauté d'utilisateurs offre également des pilotes dans plus de 30 autres langages (MongoDB, s.d.).

C'est par les pilotes que les requêtes sont implémentées, elles sont donc commandées par un langage de programmation lui correspondant et non un langage séparé comme le SQL.

#### *4.2.2.1 Mongo Shell*

Mongo Shell est compris dans les fichiers de téléchargement de MongoDB, c'est un pilote utilisant le langage JavaScript. Il est accessible par l'exécutable « mongo.exe ».

Mongo Shell est assez rudimentaire et son interface n'offre pas certaines applications basiques comme le copier-coller, ce qui peut rendre son utilisation pénible à la longue.

#### *4.2.2.2 Pymongo*

Pymongo est un outil distribué par Python, c'est le driver conseillé pour l'utilisation de ce langage. Son installation est requise pour l'utilisation des plugins QGIS pour la lecture d'une base de données MongoDB. Ainsi, ce sera également le deuxième pilote testé pour la gestion de la base de données.

Python est un langage de programmation considéré comme facile à prendre en main et à apprendre (Python, 2017). Le choix de Pymongo est donc intéressant, car il permet l'appréhension rapide du pilote tout en permettant d'ajouter un nouveau langage de programmation à la liste des langages maitrisés.

Pour l'utilisation de la version 2.7 de Pymongo et utilisée avec la version 3.6 de python.

### 4.2.3 Quelle interface ?

Les interfaces peuvent être utiles pour l'étude du schéma et des données.

#### *4.2.3.1 MongoDB compass*

MongoDB est le GUI proposé par MongoDB. Cette interface est assez intuitive et ne demande pas de connaissances préliminaires.

Elle possède plusieurs onglets permettant d'analyser une collection :

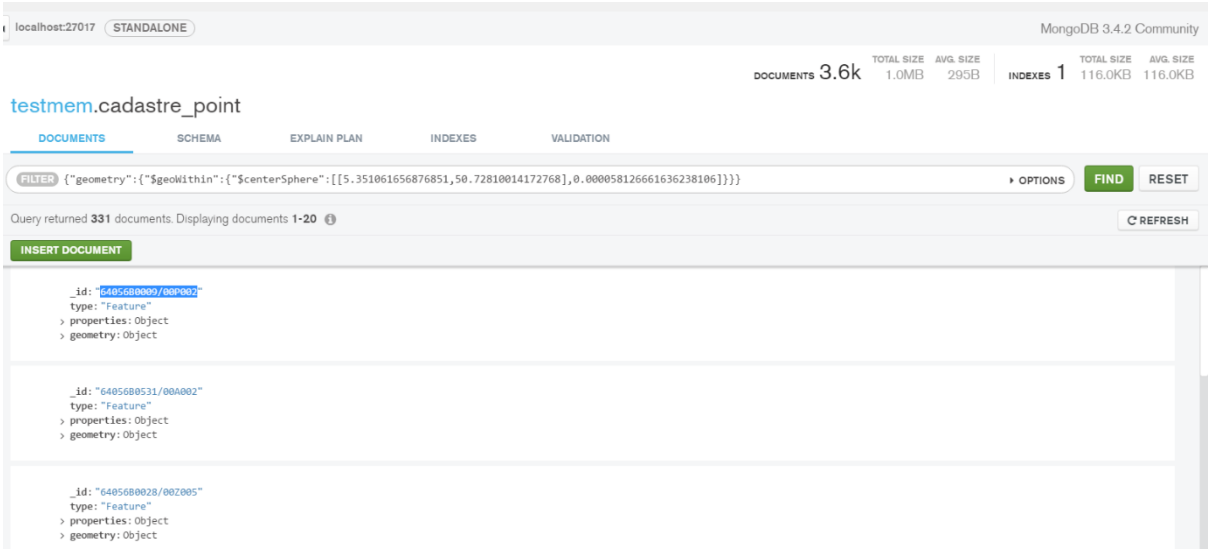



*Figure 18 Vision des options disponibles de Compass*

- « *Document* » : cet onglet permet la visualisation des données sous le format BSON. Il offre également comme chacun des onglets, un champ permettant d'insérer des requêtes pour limiter l'affichage aux documents désirés. Sur cette page, apparait également un onglet « *insert document* » qui consiste à l'ajout d'un seul document BSON à compléter.
- « *Schema* » : Cet onglet permet l'étude en profondeur du schéma. Chacun des champs présents sont étudiés un par un par la définition de leur type et par des exemples. C'est dans cet onglet que les points sont visualisables (voir Figure 19). Les différents champs intégrés sont également toujours présentés nichés les uns aux autres dans cette présentation.

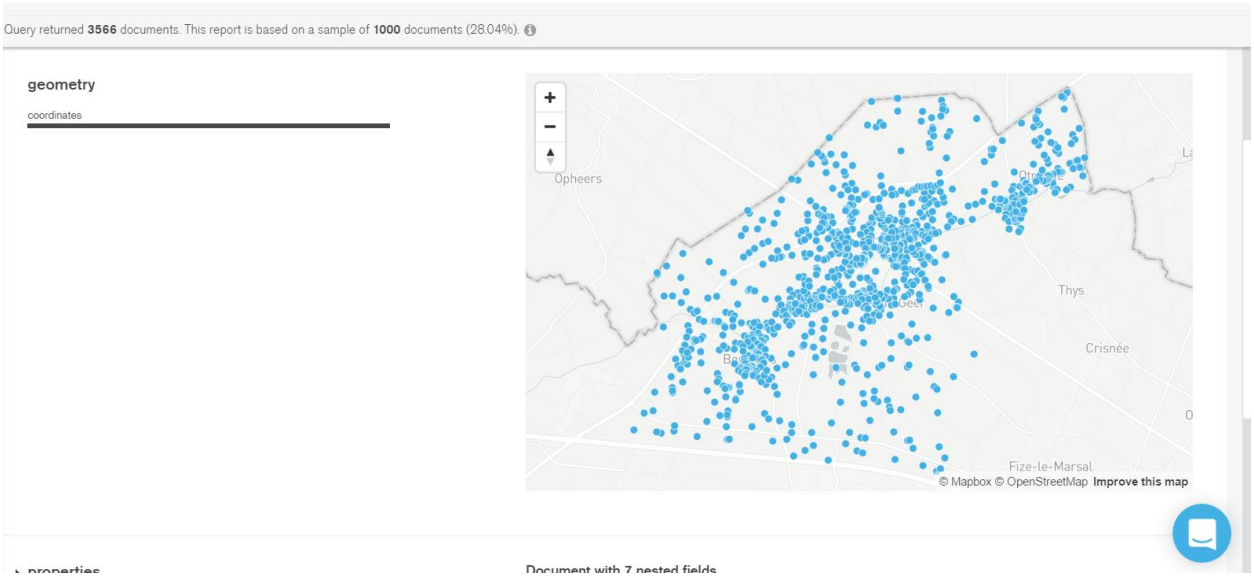



*Figure 19 Visualisation de l'affichage des points géospatiaux dans MongoDB Compass (exemple du cadastre)*

- *« Explain Plan » :* Cette option offre une analyse des performances et l'explication de celles-ci. Cet onglet représente l'équivalent de la fonction `explain()`.
- *« Indexes »* : Rassemble les propriétés de chacun des index créés et de leur utilisation
- *« Validation »* : Permet d'imposer des règles et des avertissements sur des valeurs de champs de documents pour certifier que ceux-ci répondent aux demandes de l'application.

Compass possède un avantage car il permet de visualiser les données géolocalisées sur un fond de carte OpenStreetMap. Malheureusement après plusieurs manipulations, il semble que seuls les points peuvent être affichés dans cette interface.

#### *4.2.3.2 Robomongo*

Téléchargé comme « Robomongo », maintenant appelé « Robo 3T » depuis juin 2017, ce logiciel prouve encore une fois l'évolution constante du milieu des bases de données.

Son utilisation permet la comparaison de l'interface de MongoDB avec celle d'un autre distributeur. Robo 3T offre moins de fonctionnalités (pas de visualisation des index, ni des points géoréférencés, ni de validation), mais possède quelques compléments à Compass. L'affichages des documents peut être effectué également sous forme de tableau, pour les utilisateurs habitués aux bases de données relationnelles, ou sous forme de JSON simplifié (permettant de faire ressortir la structure).

Robo 3T permet également une meilleure gestion des fonctions opérant sur les données en permettant de les enregistrer ce que n'offre pas Compass.

## 4.3 Fonctions de base

### 4.3.1 Importation

#### *4.3.1.1 Mongoimport*

Vu que le prototype à réaliser ici est issu d'une base de données relationnelle à l'origine, l'outil `mongoimport()` sera plus qu'utile pour importer rapidement une grande quantité de données. L'utilisation de `mongoimport()` n'est évidemment pas si simple qu'elle ne le semble, car il s'agit ici de créer une base de données possédant des documents intégrés. Ainsi, des manipulations complémentaires devront être mises en œuvre pour permettre cette réalisation.

`Mongoimport()` permet l'import de fichiers sous forme JSON ou CSV ou TSV (« *tab separate values* »). Voici les options de l'importation.

*Tableau 4 Options de mongoimport()*

| | |
|---|---|
| - - db <database> | Permet de préciser la base de données concernée |
| - - collection <collection> | Idem pour la collection |
| - - fields <field1,field2> | CSV : spécifie les champs à importer |
| - - type <json\| csv\| tsv> | Type de fichier à importer |
| - - headerline | CSV : précise si une ligne contient le nom des champs |
| - - mode insert \| upsert \| merge | Insérer, remplacer, ou rassembler |
| - - columnsHaveTypes<br><br><nomCol>.<type>(arg) | Permet de spécifier le type d'un champ |

### 4.3.2 Requêtes de base

- "`db.collection.find` (query, projection)" :
  « *Query* » et « *projection »* sont tous les deux optionnels, si aucun paramètre n'est entré, la requête renverra tous les documents. « *Query* » spécifie à quels arguments doivent correspondre les documents, et *« projection »* définit les paramètres qui seront retournés par la fonction.
  Plusieurs options peuvent être introduites à la suite de la fonction « *find* » de la manière suivante : « `db.collection.find` (query, projection).option() ». Voici les options principales :

*Tableau 5 Options de la fonction find()*

| | |
|---|---|
| « .sort ({champ :1})» | Le résultat sera trié de manière ascendante par le champ introduit. |
| « .limit () » | Le nombre de documents retournés est limité au chiffre introduit dans la commande |
| « .skip() » | Les x premier documents ne seront pas affichés |

- Query : {« champ » : « valeurs »}.

*Tableau 6 Opérateurs principaux pour une requête*

| | |
|---|---|
| $gt : valeur | Plus grand que |
| $lt | Plus petit que |
| $in :[valeur, valeur] | ou |
| $elemMatch :{} | Atteindre un document intégré |
| « document_int.champ » | Atteindre le champ d'un document intégré |

- Projection : {champ : binaire}

La projection permet de préciser quels champs prendre (1) ou ne pas prendre (0).

*Tableau 7 Option de la projection*

| | |
|---|---|
| $slice : nombre | Affiche les x premiers champs d'un tableau |

### 4.3.3 Requêtes Spatial

- `$geoIntersects` : Sélectionne une géométrie qui intersecte une autre géométrie GeoJSON (uniquement supporté par 2dSphere)

```
{
  <location field>: {
     $geoIntersects: {
        $geometry: {
           type: "<GeoJSON object type>" ,
           coordinates: [ <coordinates> ]
        }
     }
  }
}
```

- `$geoWithin` : Sélectionne les géométries dans l'entourage d'une géométrie GeoJSON (supporté par l'index 2d et 2dsphere)

```
{
   <location field>: {
      $geoWithin: {
         $geometry: {
            type: <"Polygon" or "MultiPolygon"> ,
            coordinates: [ <coordinates> ]
         }
      }
   }
}
```

- `$near` : retourne les objets géospatiaux dans la proximité d'un point (supporté par les deux index). Mais pour pouvoir utiliser un point GeoJSON, il faut évidemment utiliser l'index 2D sphère. Cette opération peut être utilisée avec « `sort`() » pour retourner les documents triés.

```
{
   <location field>: {
     $near: {
       $geometry: {
          type: "Point" ,
          coordinates: [ <longitude> , <latitude> ]
       },
       $maxDistance: <distance in meters>,
       $minDistance: <distance in meters>
     }
   }
}
```

- `$nearSphere` : retourne les objets géospatiaux contenus dans la sphère autour d'un point (supporté par les deux index). Idem que « *Near* » mais avec un calcul utilisant la géométrie sphérique.

- `Geonear` : est une commande qui permet d'ajouter à l'opération « *near* » une requête sur le document. « *Spherical* » est une option qui influence l'utilisation des unités de mesure, « *false* » impliquera une utilisation de mètres et « *true* » l'utilisation des radians.

```
db.runCommand( {
   geoNear: <collection> ,
   near: { type: "Point" , coordinates: [ <coordinates> ] } ,
   spherical: true,
   query : {field : « valeur » }
} )
```

Toutes les syntaxes présentées ici, sont écrites en JavaScript, le langage de base de MongoDB. Certaines petites modifications devront être effectuées pour convenir à d'autres pilotes.

## 4.4 Solutions complémentaires

### 4.4.1 QGIS et Plugins

QGIS et GDAL sont utilisés pour les transformations et prévisualisations de données, mais QGIS est également intéressant pour ses extensions. Il existe actuellement 4 extensions offrant des interactions avec MongoDB :

- Load MongoDB Layers
- Mongo_memorizer
- MongoConnector
- Save Layer in MongoDB

### 4.4.2 Notepad++ et l'invite de commande

Notepad++ est également fortement utile pour la manipulation des fichiers JSON, GeoJSON et CSV, ainsi que pour créer des programmes Python.

# 5. APPLICATION

## 5.1 Détails de l'installation

Le package open source de MongoDB contient un serveur et un client.

Pour lancer le serveur, il suffit d'exécuter bin/mongod.exe. Le serveur tourne par défaut sur le port 27017. Lors de son lancement, il est également possible d'établir le répertoire ou stocker les données du serveur grâce à la commande `dbpath`.

Pour connecter le client par défaut de MongoDB (mongo shell), il suffit de lancer l'exécutable mongo.

## 5.2 Traitement des données

### 5.2.1 Schémas

#### *5.2.1.1 Schéma conceptuel*

Le schéma conceptuel utilisé pour la mise en place des données est presque identique à celui utilisé pour l'illustration du choix des données, à l'exception de deux points. Les deux types de données faisant office de résumé (fiche de regard PDF et plan d'ensemble PDF) n'ont pas été intégrés dans la base de données. Premièrement, leur information est redondante avec le reste des données, ensuite il est conseillé de stocker les PDF dans un chemin relatif vers un cloud. Le cloud ne faisant pas partie du travail, les seuls objets volumineux conservés sont les images des regards. Voici le schéma décrivant les données implémentées dans la BD (Figure 20).

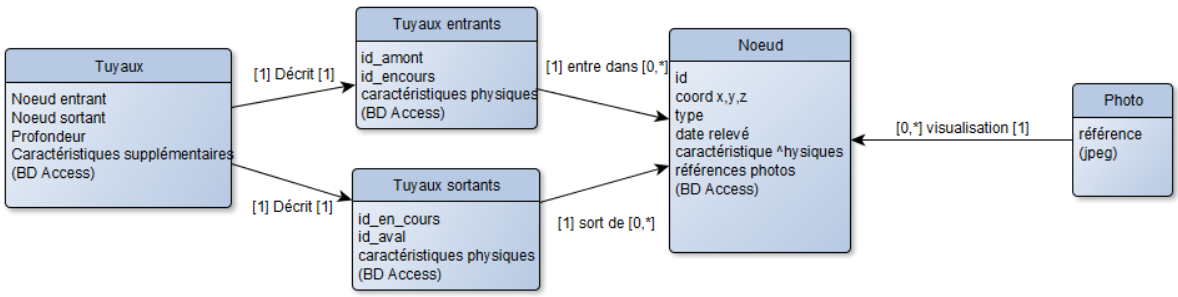



*Figure 20 Modèle conceptuel des données sélectionnées*

#### *5.2.1.2 Adaptation à MongoDB*

On peut donc constater ici que les tuyaux entrants et sortants ont été introduits au document concernant le nœud. Ces deux sous-documents sont au fait des tableaux, c'est-à-dire qu'ils peuvent contenir eux-mêmes plusieurs objets. Les collections (tables pour le relationnel) ne sont pas réellement connectées par une clé étrangère. Si le besoin en est de les relier, l'outil « *lookup* » s'en chargera. Voici la modification à effectuer au schéma pour représenter la modélisation importée dans MonogDB.

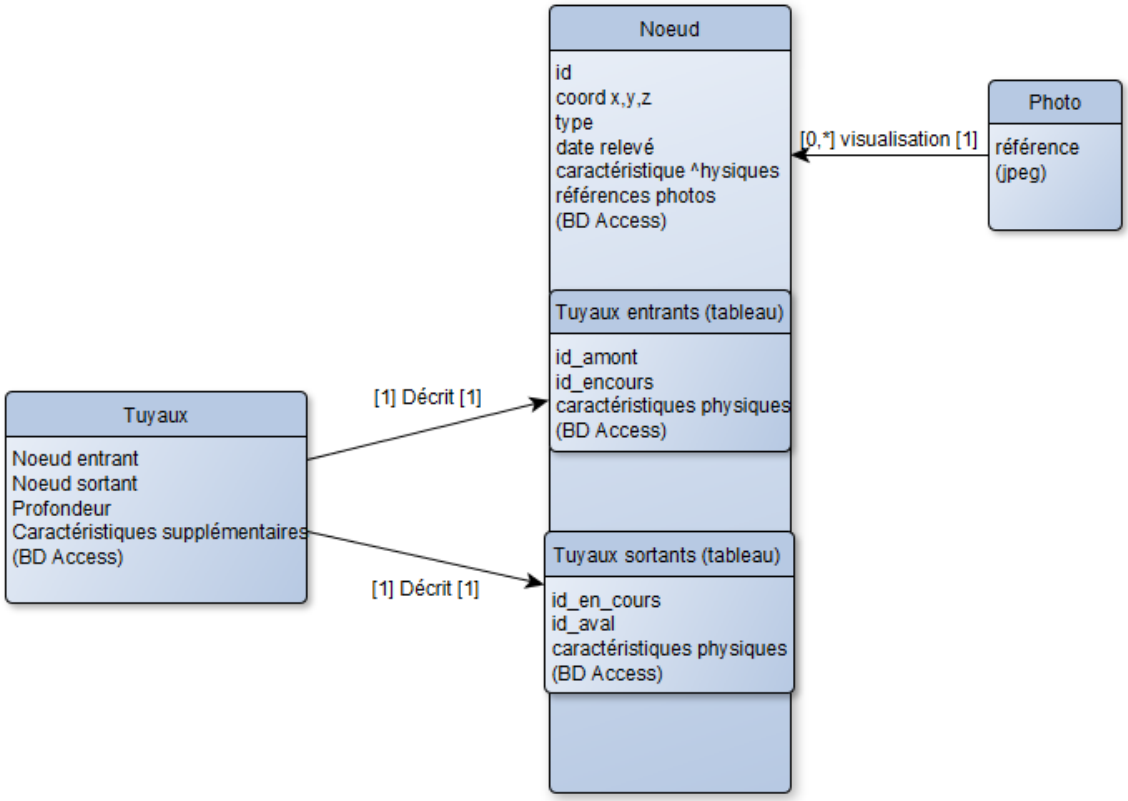



*Figure 21 Modèle adapté à MongoDB*

### 5.2.2 Base de données Access

La base de données actuelle de l'AIDE est hébergée par Access. Cette base de données contient une quarantaine de table, vu la normalisation élevée.

Seules 4 tables sont exportées en CSV :

- Tuyaux_entrants
- Tuyaux_eortants
- Tuyaux
- Releve_regard

### 5.2.3 Données géolocalisées

#### *5.2.3.1 Préparation des données*

Certaines données, sont déjà disponibles en fichier de formes (comme le cadastre), d'autres sont uniquement sous forme de tableur issus d'une base de données Access avec des champs comprenant des coordonnées Lambert 72. Il faut donc l'importer dans QGIS pour le transformer en couche spatiale.

#### *5.2.3.2 Transformation en GeoJSON*

Une fois importées dans QGIS, les couches sont exportées en GeoJSON sous la projection WGS84.

### 5.2.4 Données non géolocalisées

Deux types de manipulations différentes ont été utilisées pour les données non géolocalisées. Ces deux manipulations dépendent de la manière donc les données doivent être intégrées dans la base de données. Certaines données représentent la base d'une collection à elles seules et d'autres viennent s'intégrer dans les documents d'une collection existante.

Tout d'abord chacun des types de données sont exportés en CSV depuis la base de données Access.

#### *5.2.4.1 Données à la base d'une collection*

Ces données à importer sont de simples documents et l'importation de celles-ci simplement via le CSV est suffisant, car leur schéma ne doit pas être modifié et peut rester identique au schéma du modèle relationnel.

#### *5.2.4.2 Données à intégrer dans les documents d'une collection*

Pour pouvoir créer des documents intégrés par importation, il faut que ceux-ci présentent déjà les caractéristiques d'un document hiérarchisé. Ainsi c'est le format JSON qui sera utilisé pour l'importation de ces données.

Pour la transformation d'un CSV en JSON, c'est un script JavaScript qui a été utilisé (voir annexe 2).

## 5.3 Importation des données

### 5.3.1 Format GeoJSON compatible avec MongoDB

Le format GeoJSON produit par le module GDAL qui effectue la transformation des fichiers de formes dans QGIS ne correspond pas parfaitement au format JSON que MongoDB accepte.

Il faut donc effectuer quelques modifications pour pouvoir importer ces fichiers. Voici les trois étapes à suivre (Flo, 2016).

1) Retirer l'entête
2) Retirer la virgule en fin de ligne
3) Retirer les deux crochets finaux

Voici un exemple d'un fichier GeoJSON produit par GDAL. Pour l'utiliser pour une importation dans MongoDB, il faut supprimer chaque partie écrite en rouge.

```
{
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
} },
"features": [
{ "type": "Feature", "properties": { "FID": 0 }, "geometry": { "type": "Point",
"coordinates": [ 5.36534670637232, 50.736312609676354 ] } },
{ "type": "Feature", "properties": { "FID": 271 }, "geometry": { "type": "Point",
"coordinates": [ 5.33499456749666, 50.720464147548704 ] } },
{ "type": "Feature", "properties": { "FID": 272 }, "geometry": { "type": "Point",
"coordinates": [ 5.334953176509237, 50.720476177467077 ] } },
```

```
{ "type": "Feature", "properties": { "FID": 273 }, "geometry": { "type": "Point",
"coordinates": [ 5.334710923216598, 50.720631379138865 ] } }
]
}
```

### 5.3.2 Remarques globales

#### *5.3.2.1 Id*

Il a été décidé d'introduire l'identifiant mis en place par l'AIDE pour un soucis de cohérence étant donné que ce sont eux qui réalisent le lien entre tous les éléments de l'AIDE. Pour profiter des grandes performances offertes par MongoDB, il est conseillé de laisser l'index implémenté par MongoDB.

Pour pouvoir exporter ce champ en tant que clé primaire, il suffit de renommer le nom du champ des documents JSON en « _id ». De plus pour que le champ soit détecté en temps qu'identifiant, il doit se retrouver dans les documents JSON aux hiérarchies les plus hautes.

#### *5.3.2.2 Invite de commande*

L'opération `mongoimport()` doit être implémentée dans l'invite de commandes à la racine du répertoire « *bin* » de MongoDB et non dans les clients MongoDB.

### 5.3.3 Données géoréférencées

Comme mentionné plus haut, les couches géoréférencées sont uniquement exportées en GeoJSON car MongoDB est incapable de détecter des champs contenant des coordonnées lorsque on les importe en CSV. En effet, la fonction `columnsHaveTypes` de `mongoimport()` offre beaucoup de types mais pas ceux des coordonnées.

Voici un exemple type de commande d'importation de fichier GeoJSON :

```
mongoimport --db testmem --collection regards --file
C:\Users\amand\Documents\Am\Master2\memoire\donnees\point_taques\regards.geojson
```

Une fois les données importées dans la collection correspondante, l'index géospatial peut être créé sur le champ possédant la géométrie. L'indiex doit être créé pour pouvoir profiter des requêtes spatiales correspondant à celui-ci.

```
db.regards.createIndex({« coordinates » : « 2dsphere»})
```

Par ces manipulations, les polygones et les points du cadastre peuvent être importés ainsi que les regards de l'AIDE. Il faut maintenant ajouter les tuyaux, qui après analyse de la structure des données et des requêtes à effectuer, ont été intégré aux documents tuyaux.

Pour cela, quelques manipulations supplémentaires sont nécessaires. En effet, une particularité rend la manœuvre plus difficile : un regard peut connaître plusieurs tuyaux entrants et plusieurs tuyaux sortants. Ainsi, si on se limite à importer les tuyaux en tant que tel, lorsque plusieurs entités se rapportent aux mêmes id, le dernier tuyau importé écrase les précédents. Quel que soit le type d'importation, le résultat ne correspondait pas aux attentes : « *insert* » ne permet pas l'utilisation d'un ID déjà existant, « *Upsert* » remplace tout le document, et « *merge* » ne permet introduction que d'un document, les autres se faisant écraser.

Pour répondre à cette problématique une solution en trois étapes a été trouvée. Une option `$push` existe pour ajouter un objet à un tableau lorsqu'on met un document à jour, elle conviendrait à notre situation, mais quelques manipulations doivent être mises en œuvre pour pouvoir l'utiliser.

1) Transformer les objets « tuyaux » en tableau et les importer simplement avec `mongoimport()`
2) Créer un programme pour sélectionner les « tuyaux doublons »
3) Créer un programme pour ajouter chacun de ces tuyaux aux regards correspondant via l'option `$push`.

Transformer les objets en tableau est assez simple, c'est uniquement une manipulation de JSON. Il suffit de transformer ce document JSON où « tuyaux entrant » est un objet :

```
{"_id":"64056-02RV006550","tuyaux_entrant":{"ref_noeud_amont":"64056-
02BO006560","suffixe_c":"FAUX","_id":"64056-02RV006550",……}}
```

Ceci en ajoutant des crochets autour de l'objet pour signifier qu'il pourrait en avoir plusieurs différents et donc créer un tableau (voir les modification en rouge) :

```
{"_id":"64056-02RV006550","tuyaux_entrant":[{"ref_noeud_amont":"64056-
02BO006560","suffixe_c":"FAUX","_id":"64056-02RV006550",…….}]}
```

On peut ensuite utiliser simplement `mongoimport()` pour importer ces fichiers (tuyaux_entrant et tuyaux_sortant).

```
mongoimport --db testmem --collection regards --mode merge --file
C:\Users\amand\Documents\Am\Master2\memoire\donnees\point_taques\tuyaux_entrant.j
son
```

Ensuite, un programme a été construit en Python (voir annexe 3) pour retrouver et stocker dans un csv tous les documents écrasés. Ce code sélectionne chacune des lignes dont l'ID est répété et les écrit tous dans un fichier à l'exception du dernier qui se retrouve déjà dans la base de données vu que c'est lui qui a écrasé tous les autres. Ce code peut être appliqué plusieurs fois dans le cas où des triplets existent.

Enfin, un deuxième code Python est mis en place pour importer les documents isolés dans les tableaux des documents correspondant (voir annexe 4). Une boucle parcourant le fichier CSV des tuyaux en double et les insère dans le tableau existant grâce à la fonction `update_many()` et `$push` de Pymongo. Voici un extrait de ce code précisant l'application de mise à jour avec `$push`.

```
db.regards_array.update_many({"_id":row[0]},

{'$push':{"tuyaux_sortant":{"_id":row[0],"ref_noeud_aval":row[1],"suffixe_c":row[
2],"index_releve":row[3],"prof_radier":
row[4],"dia_larg":row[5],"hauteur":row[6],"forme tuyau":row[7],"materiau
tuyau":row[8],"materiau_revetement":row[9],"code_etat":row[10],"type_assain":row[
11],"type_troncon":row[12],"demergement":row[13],"type_systeme":row[14],"type_noe
ud":row[15],"indice_tuyau":row[16]}}})
```

### 5.3.4 Données CSV

Premièrement, il faut spécifier que le format CSV qui est accepté par MongoDB doit être séparé par des virgules et non des points-virgules.

L'importation CSV est utilisée pour importer la table contenant les précisions techniques sur les tuyaux. Cette table n'est pas introduite dans la structure du document car elle est considérée comme secondaire et son utilisation ne vaut pas la duplication que son introduction dans un document existant provoquerait.

En effet, un tuyau se retrouve deux fois dans la collection « regard » car il se retrouve une fois pour le nœud amont et une fois pour le nœud aval. Ainsi, si on introduit les informations supplémentaires dans cette même collection, trop de réplications seront créées pour l'utilisation qui en sera faite. Créer trop de duplications pourrait également poser problème lors des mises à jour.

Voici un exemple de commande pour l'importation d'un CSV.

```
mongoimport --db testmem --collection tuyaux --headerline --type csv --file
C:/Users/amand/Documents/Am/Master2/memoire/donnees/point_taques/Infonet_TUYAUX.c
sv
```

### 5.3.5 Images

Le stockage des images est effectué par GridFS. C'est un choix stratégique pour tester cette fonction particulière de MongoDB, sachant que d'autre options moins lourdes existent pour notre type d'images.

#### *5.3.5.1 Réalisation*

L'exportation de GridFS est effectuée également par un programme python, ce qui permet d'utiliser les fonctions d'importation de GridFS une fois pour tous les fichiers. Gridfs est une extension dans Python, et c'est la fonction `put()` qui est utilisée pour introduire les fichiers.

Un problème de librairie fait que l'importation des JPG pose problème. Le script est donc complet mais non fonctionnel (voir annexe 9).

#### *5.3.5.2 Critique*

Fort lent, certains conseillent de plutôt charger les images dans le cloud que sur le Serveur MongoDB. GridFS à l'avantage de faciliter les choses, mais augmente le nombre d'opération par requête à cause de la création de ses deux collections. C'est pour cela que certains conseillent d'agréger les deux collections si les images ne sont pas trop lourdes. Vu les résultat obtenus, il est conseillé de garder les méthodes classiques de stockage des images dans les BD.

## 5.4 Requêtes (détail)

### 5.4.1 Type de requêtes

#### *5.4.1.1 Requêtes à effectuer*

- Trouver les regards proches d'une parcelle

- Trouver les parcelles proches d'un regard
- Trouver tous les nœuds en amont d'un regard

#### *5.4.1.2 Réalisation*

Les éléments du langage de requêtes sont récurrents selon les pilotes utilisés, mais certains détails varient, comme l'utilisation de guillemets, d'« *underscores* » ou de crochets.

Il faut donc être attentif lors du passage d'un langage à une autre. Le langage utilisé ici est Python avec son pilote Pymongo.

Script requête 1 :

La réalisation d'un script python pour effectuer les requêtes permet de les rendre modulables et ainsi de rassembler les requêtes semblables. Ce premier programme permet de rechercher n'importe quels points de la BD dans l'entourage d'un quelconque autre point, sélectionné par son identifiant.

Le script permet d'introduire la collection source et la collection cible ainsi que l'identifiant de l'élément à partir duquel les nœuds proches seront recherchés. La distance en radian (car calcul sphérique) doit également être introduite.

Ensuite la requête est articulée en deux parties. La première pour extraire la géométrie du point source et la seconde pour effectuer une recherche à partir de cette géométrie. L'extraction des coordonnées est assez longue car pour pouvoir être introduite dans la requête suivante, elles doivent être de type « liste », ce qui n'est pas le cas à la sortie de la requête (elles sont de type dictionnaire). Diverses manipulations doivent donc être mises en place pour isoler les coordonnées du point et les formuler dans le bon format (voir code en annexe 5).

Une fois les deux requêtes complétées, les résultats sont transformés et mis en page pour pouvoir être inscrit dans un fichier JSON pour la visualisation.

Script requête 2 :

Le script de la requête 2 se trouve également en annexe (voir Annexe 6). Il permet d'introduire en entrée l'identifiant du regard cible.

Une boucle de requêtes est ensuite effectuée pour rechercher progressivement les regards en amont. Un « *while* » et un « *for* » sont utilisés ensembles pour compenser l'inexistence de « *do… while* » en Python. Cette boucle est élaborée pour tourner tant qu'un tuyau en amont existe.

La procédure pour isoler l'identifiant du nœud en amont est de nouveau assez longue et demande une transformation de type et une isolation de la valeur qui n'est pas la même que le script précédent car on isole cette fois un identifiant et non des coordonnées. Cette solution est la seule ayant été trouvée, mais une technique plus rapide doit exister.

Les résultats sont également inscrits dans un fichier JSON pour pouvoir être afficher.

MongoDB possède une fonction `explain`() qui permet d'effectuer un rapport sur l'indexation des collections et sur les requêtes. Les producteurs de MongoDB conseillent fortement son utilisation lors des phases de développement et de production (MongoDB, 2015).

## 5.5 Visualisation des résultats

### 5.5.1 Via Compass

Compass offre une visualisation uniquement des points, et ne permet d'afficher que des requêtes uniquement réalisées sur Compass. Malheureusement ces types de requêtes sont assez limités et ne correspond pas à nos besoins. Une ligne d'exemple de requêtes est introduite dans l'annexe 5 (elle correspond à l'équivalent de la requête Pymongo effectuées) pour pouvoir tester et comparer les requêtes. Compass offre une possibilité de créer de manière interactive des requêtes de recherche dans un périmètre sphérique, mais l'offre se limite à cette opération.

### 5.5.2 Via QGIS

Aucune des extensions ne fonctionne correctement. Elles sont soit incapables d'effectuer la connexion avec MongoDB ou soit n'arrivent pas à importer les couches quand la connexion est effectuée correctement.

De plus ces extensions ne sont pas des plus utiles dans notre cas, vu qu'elles sont créées dans le but d'exporter les couches présentes dans la base de données. Ce qui est recherché dans ce cas-ci est l'affichage des résultats des requêtes effectuées, qui elles ne sont pas enregistrées dans la base de données.

### 5.5.3 Via un explorateur

La solution qui est validée est l'utilisation d'une API Javascripts de Google nommée *« Data Layer: Drag and Drop GeoJSON »* (Google, 2017). Cette application est présentée sous forme de fichier HTML (voir annexe 7). Son utilisation est aisée. Il suffit de modifier l'URL source pour l'utilisation de l'API et d'y insérer une clé fournie simplement par Google.

Il suffit ensuite de glisser le contenue d'un fichier GéoJSON dans le fichier généré par un explorateur pour pouvoir afficher les géométries. Mais les documents exportés lors de la manipulation de la base de données MongoDB sont en format JSON. Il faut donc effectuer la manœuvre inverse que pour l'importation d'un fichier GeoJSON dans MongoDBpour pouvoir afficher les données.

Il faut donc, pour rappel, insérer des virgules entre chaque documents JSON et ajouter

```
{
  "type": "FeatureCollection",
  "features": [
```

au début du code et « `]}` » à la fin de celui-ci.

Voici une capture d'écran de l'application de visualisation utilisée. Les données affichées sont les données représentant le résultat de l'exemple de la requête 2 (annexe 6). Une exemple de ces données sont disponibles sous format GeoJSON dans l'annexe 8 (seul deux points sont affichés avec leur structure complète car les données possèdent un grand nombre de champs).

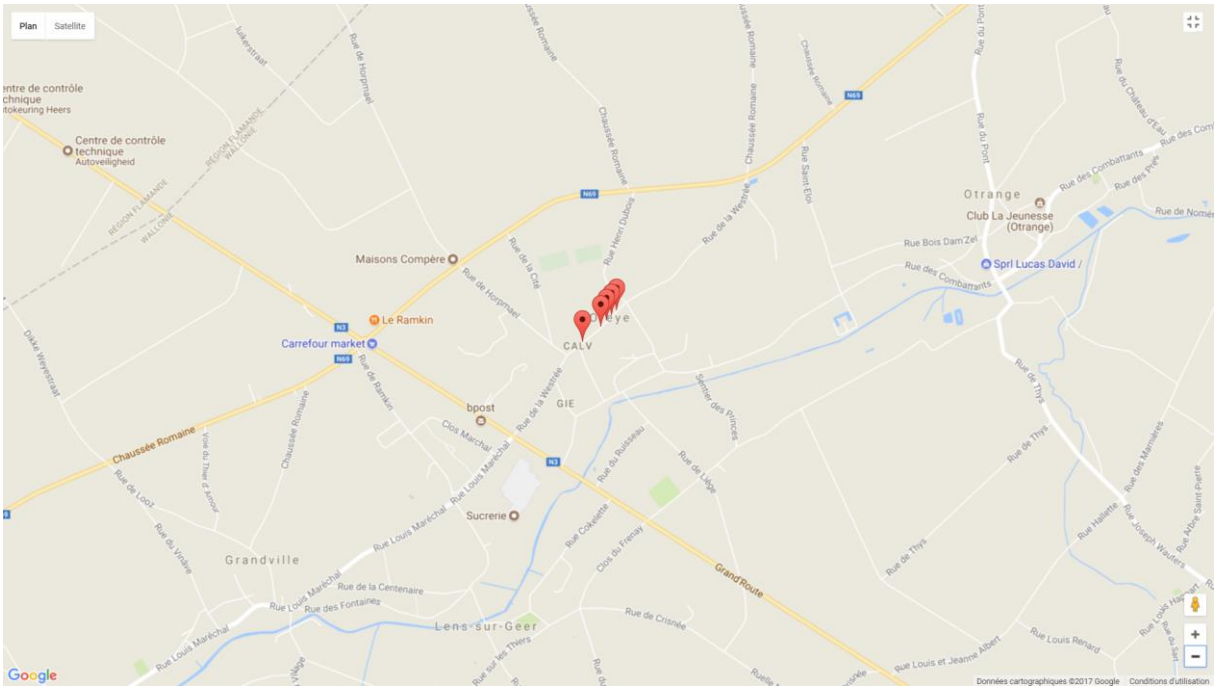



*Figure 22 Visualisation du résultat de la requêtes 2 avec l'API google maps*

# 6. CONCLUSION

L'implémentation de la base de données MongoDB fonctionne et donne des résultats. Voici une discussion sur les différentes conclusions qui peuvent être tirées.

## 6.1 Schéma

Les données de l'AIDE sélectionnées préalablement ont toutes pu être introduites dans la base de données NoSQL, les données relationnelles comme les données au schéma plus complexe et variable. Les données relationnelles peuvent être introduites dans le même schéma que les données non-structurées. Certaines relations fonctionnent même bien en tant que documents intégrés. Ce schéma s'adapte bien au type de données, mais aussi à la phase exploratoire du travail, car il permet d'être modifié et adapté tous le long de l'implémentation selon les changements de stratégies.

Un schéma non structuré est facile à utiliser, mais appréhender ce type de conception peut prendre du temps. Car c'est une différente manière de penser, et il faut l'adopter entièrement pour pouvoir profiter des avantages des modèles de base de données orientés document. Il faut également prendre conscience des besoins de l'application pour adapter correctement le modèle. Utiliser un schéma trop complexe peut pourtant avoir des points négatifs. En effet, pour appliquer les requêtes, il faut connaître le schéma et savoir quels champs et quels documents interroger.

## 6.2 Géospatial

Les données géospatiales sont introduites sans difficulté dans la base de données. Le format GeoJSON est facile à manipuler et sa conception permet une bonne compréhension des données par la simple lecture.

Par contre les requêtes géospatiales de MongoDB sont limitées. Premièrement, les requêtes simples faisant appel à deux couches géospatiales ne peuvent pas être implantées rapidement et doivent être établies en plusieurs étapes. MongoDB conseille d'introduire les données reliées dans les mêmes documents, mais cette option n'est pas disponible pour les relations spatiales. Un outil de jointure existe quand des documents ne sont pas intégrés mais de nouveau, cet outil n'existe pas pour les données géolocalisées.

Enfin, les outils de visualisation disponibles ne sont encore assez évolués. Compass, une interface de MongoDB, offre une visualisation, mais limitées aux points et offrant de créer des requêtes interactives avec `geowithin()` uniquement

La solution finalement utilisée demande déjà une partie de développement d'application pour pouvoir juger les résultats de requêtes. Son utilisation n'est pas du tout typique à l'utilisation de MongoDB, ni des bases de données NoSQL. De plus, la visualisation n'est pas directe et demande des traitements intermédiaires.

## 6.3 Images

MongoDB offre un outil intéressant, GidFS. Les bases de données n'offrent pas souvent la possibilité de stocker des fichiers importants. C'est une option à prendre qui a été prise en compte et qui a été testée, mais son exploration n'a pas été approfondie et demande plus d'attention pour pouvoir la juger entièrement. Stocker les images dans une URL reste une solution convenable.

## 6.4 Requêtes

En ce qui concerne l'interrogation de la base de données, l'étude de l'efficacité des requêtes n'a pas été effectuée en détail. Pour cela, il aurait premièrement fallu pouvoir confirmer que les requêtes ont été exprimée d'une manière la plus efficace possible. Et de plus, l'étude de la performance serait plus intéressante en parallèle avec un système de gestion de bases de données relationnelles.

Les manipulations démontre néanmoins, les larges possibilités de requêtes offertes par les pilotes et soulignent l'importance de connaître les particularités des syntaxes de requêtes selon les pilotes.

## 6.5 Global

Globalement, utiliser MongoDB en réponse à une grande quantité de données et pour des meilleures performances semble une bonne option et offre quelques avantages significatifs.

L'installation du serveur est rapide et sans coût. Même si quelques options payantes existent et pourraient faciliter la création et la gestion des bases de données. En effet, des options comme les ETL ou les compléments de sécurité se retrouvent dans des packages plus évolués et payants, destinés à des objectifs plus larges que le développement.

La prise en main du logiciel peut être rapide pour quelqu'un qui a l'habitude de programmer, grâce à l'existence de nombreux pilotes. Mais la présence d'un grand choix d'outils peut être positif tout comme négatif. En effet, d'un côté, choisir le langage qui nous convient peut faciliter la manipulation. Mais le choix peut devenir difficile quand plusieurs langages ont leurs avantages. Le choix du pilote est donc un choix crucial et doit être basé sur une justification fondée. Certains pilotes peuvent offrir des fonctionnalités différentes et donc, devoir jongler entre deux pilotes quand ces options sont nécessaires n'est pas idéal.

La documentation offerte par MongoDB est très riche en explications et en exemples. Ces exemples sont malheureusement souvent orientés vers la création de nouvelles données et non de l'importation. De plus, peu de rubriques concernent le géospatial.

La communauté de MongoDB est très importante par rapport à d'autres SGBD NoSQL, mais elle est relativement moins développée que pour les SGBDR comme PostgreSQL. De plus certaines solutions proposées ne semblent pas être fixes et les avis divergent souvent pour des problématiques comme l'importation d'images.

Il ne faut pas s'arrêter sur les points négatifs de MongoDB, car ces systèmes sont en progressions permanentes et leur adaptation au type données géospatiales de plus en plus utilisé ne fait que

commencer. Le type de travail effectué permet de comprendre que ce type de domaine demande une étude continue de la progression des techniques et des particularités.

Les avis divergent fortement sur l'avantage du NoSQL, mais MongoDB offre définitivement des énormes avantages en combinant son langage de requêtes avec des langages de programmation. Cela offre des nombreuses options et permet de rendre les requêtes beaucoup plus compréhensive et instinctives que le SQL

Ces BD NoSQL ne sont pas crées afin de remplacer le SQL mais plutôt pour offrir une meilleure option pour un certain type de données et de requêtes.

Travailler sur le NoSQL est intéressant, car cela offre une nouvelle vision sur la façon de gérer les données et montre que même si une technologie est généralisée, cela ne prouve pas qu'elle est parfaite et qu'elle correspond entièrement aux cas d'utilisation auxquels elle fait face.

Ce travail ne donne pas les possibilités d'explorer la totalité des avantages et défauts de MongoDB, il ne représente qu'une esquisse de l'implémentation et de l'utilité d'une base de données NoSQL.

## 6.6 Perspectives

Il est donc important de tester le système de gestions avec une plus grande quantité de données pour pouvoir constater l'entièreté des capacités de la base de données dans ces cas. Regarder les performances d'une requête n'a pas beaucoup de sens lorsqu'une petite quantité de données est utilisée, car l'avantage de MongoDB par rapport au SQL augmente en fonction de la quantité de données. Travailler avec le « *sharding* » est également une des spécialités de MongoDB qui n'a pas été exploitée. L'application mobile est aussi un fort de MongoDB.

Tester et prévoir l'introduction de nouvelles données pourrait également être intéressant et l'importation de nouveau documents risque de varier grandement par rapport à l'importation directe de collections. Cela permettrait également de profiter d'avantages des points forts de MongoDB.

De plus, comparer les requêtes de MongoDB avec des requêtes SQL ainsi que la création de nouvelles requêtes serait plus que bénéfique pour l'analyse des performances et des avantages des deux systèmes.

Trouver de meilleures options de visualisation pourrait aussi être une perspective intéressante, car une API google, n'est peut-être pas la solution optimale.

Certains défauts de MongoDB pourraient être détournés en essayant d'implanter ce type de réseau géolocalisé plutôt avec un SGBD NoSQL orienté graphe, par exemple Néo4J. Ce SGBD offre une palette de requêtes plus large que celle de MongoDB.

Ensuite, étudier la question de sécurité plus en détail pourrait également constituer une problématique à elle seule. Ce point n'est pas encore fonctionnel pour la plupart des bases de données NoSQL et pose encore des problèmes aujourd'hui comme cela a été mentionné dans ce travail.

Enfin réétudier plus en détails les besoins de l'AIDE serait également un plus. Car les schémas non structurés des bases de données NoSQL peuvent offrir de nombreuses possibilités et avantages

différents selon le système et le modèle. Ainsi, la schématisation préalable de données n'est pas primordiale mais c'est surtout les objectifs de l'application qui doivent être fixés pour permettre l'adoption du système le plus performant dans le cas étudié. Enfin, l'utilisation dans une situation réelle est le meilleur test pour une base de données.

# BIBLIOGRAPHIE


Abraham, S. (2016). Comparative Analysis of MongoDB Deployments in Diverse Application. *International Journal of Engineering and Management Research*, 6, 21-24.

AIDE. (2016). *La société*. http://www.aide.be/accueil/la-societe. Consulté le 6 août 2017

Amirian, P., Basiri, A., & Winstanley, A. (2014). Evaluation of Data Management Systems for Geospatial Big Data. *Computational Science and Its Applications, ICCSA* Guimaraes: Springer International Publishing Switzerland, 678–690.

Bulk, D. (2011). *Big Data Impacts Data Management: The 5 Vs of Big Data*. Dave Bulk Experienced DB2 Consulting and Training: http://davebeulke.com/big-data-impacts-data-management-the-five-vs-of-big-data/. Consulté le 12 août 2017

Carolin, C. (2014). Bases de données NO SQL et SIG: d'un existant restraint à un avenir prometteur. *Géomatique Expert*, 100, 44-48.

Cattell, R. (2010, Décembre). Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record*, 39, 12-27.

Chodorow, K. (2013). *MongoDB, The Definitive Guide (second edition).* Sebastopol: O-Reilly Media, 432 p.

de Souza Baptista, C., Santos Pires, C. E., Farias Batista Leite, D., Guimares de Oliveira, M., & de Lima Junior, O. F. (2014). NoSQL Geographic Databases: An Overview. Dans E. Pourabbas, *Geographical Information: Trends and Technologies*. Rome: CRC Press, 73-103.

Digora. (s.d.). *Qu'est-ce qu'une base NoSQL? Les cas Datastax et MongoDB.* Digora: http://www.digora.com/fr/blog/definition-base-nosql-datastax-mongodb. Consulté le 17 août 2017.

Flo. (2016, mars 13). *Mongodb and geodata part 1 – from shapefile to mongodb 3.2*. iSticktoit.net: http://isticktoit.net/?p=1444. Consulté le 3 août 2017

Fotache, M., & Cogean, D. (2013). NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL. *Informatica Economica*, 17, 41-57.

Funck, R., & Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. *Cloud and Service Computing (CSC), 2011 International Conference on.* Hong Kong: IEEE, pp. 336-341.

Huang, Q., & Xu, C. (2014). A data-driven framework for archiving and exploring social media data. *Annals of GIS*, 20, 265-277.

Hurst, N. (2010). *Visual Guide to NoSQL Systems.* Récupéré sur Nathan Hurst's Blog, Thoughts on Software, Technology, and Startups: http://blog.nahurst.com/visual-guide-to-nosql-systems. Consulté le 13 août 2017

IBM. (2012). *IBM*. Bringing Big Data to the enterprise: https://www-01.ibm.com/software/in/data/bigdata/. Consulté le 29 juillet 2017.

Johnson, N. (2009, novembre 9). *Spatial indexing with Quadtrees and Hilbert Curves*. Nick's Blog: http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves. Consulté le 18 août 2017

JSON. (s.d.). *Présentation de JSON*. JSON: http://www.json.org/json-fr.html. Consulté le 6 août 2017

Laloux, M. (2011). *Le NoSQL dans le domaine géospatial : MongoDB avec JavaScript ou Python, ArcGIS et Quantum Gis.* PortailSIG: http://www.portailsig.org/content/le-nosql-dans-le-domaine-geospatial-mongodb-avec-javascript-ou-python-arcgis-et-quantum-gis. Consulté le 27 juillet 2017

Lee, J.-G., & Kang, M. (2015). Geospatial Big Data: Challenges and Opportunities. *Big Data Research*, 2, 74-81.

Losson, O. (S.d.). *Introduction aux Systèmes de Gestion de Bases de Données Relationnelles.* Université de Lille: http://lagis-vi.univ-lille1.fr/~lo/ens/commun/bd/bd_cours.pdf. Consulté le 12août 2017.

Lourenço, J., Cabral, B., Carreiro, P., Vieira, M., & Bernardino, J. (2015). Choosing the right NoSQL database for the. *Journal of Big Data*, 2, 18.

McKnight Consulting Group. (2014). *NoSQL evaluator's guide.* 16p.

Menge, C. (2015). *Storing (small) Images in MongoDB*. Christoph Menge, notes on software engineering: http://menge.io/2015/03/24/storing-small-images-in-mongodb/. Consulté le 20 août 2017

Minelli, M., Chambers, M., & Dhiraj, A. (2013). *Big Data, big analytics.* New Jersey: John Wiley & Sons, Inc. 224 p.

MongoDB. (2015). *Guide de migration d'un système RDMS vers MongoDB.* MongoDB: https://www.mongodb.com/fr/collateral/rdbms-mongodb-migration-guide. Consulté le 27 août 2017.

MongoDB. (s.d-a). *GridFS*. MongoDB Documentation: https://docs.mongodb.com/manual/core/gridfs/#gridfs. Consulté le 13 août 2017.

MongoDB. (s.d.-b). *MongoDB documentation*. Récupéré sur MongoDB documentation: https://docs.mongodb.com/. Consulté le 13 août 2013

Nilsson, A. (2017). *Comment éviter la prise en otage de vos données*. MongoDB: https://www.mongodb.com/fr/blog/post/how-to-avoid-a-malicious-attack-that-ransoms-your-data. Consulté le 19 août 2017.

NoSQL. (2011). *Définition du NoSQL*. NoSQL: http://nosql-database.org/. Consulté le 28 juillet 2017

Oussous, A., Benjelloun, F.-Z., Ait Lahcen, A., & Belfkih, S. (2015, Mai). Comparison and Classification of NoSQL Databases. *International conference on Big Data, Cloud and Applications*. Tetouan, Maroc. 1-6

Parker, Z., Poe, S., & Verbsky, S. (2013). Comparing NoSQL MongoDB to an SQL DB. *Proceedings of the 51st ACM Southeast Conference* New York: ACM., 5:1-5:6.

Python. (2017). *Python*. Python: https://www.python.org/. Consulté le 17 août 2017

Sarthak, A., & KS, R. (2015). Performance Analysis of MongoDB Vs. PostGIS/PostGreSQL Databases. *Free and Open Source Software For Geospatial Conférence*. Seoul. 37-43

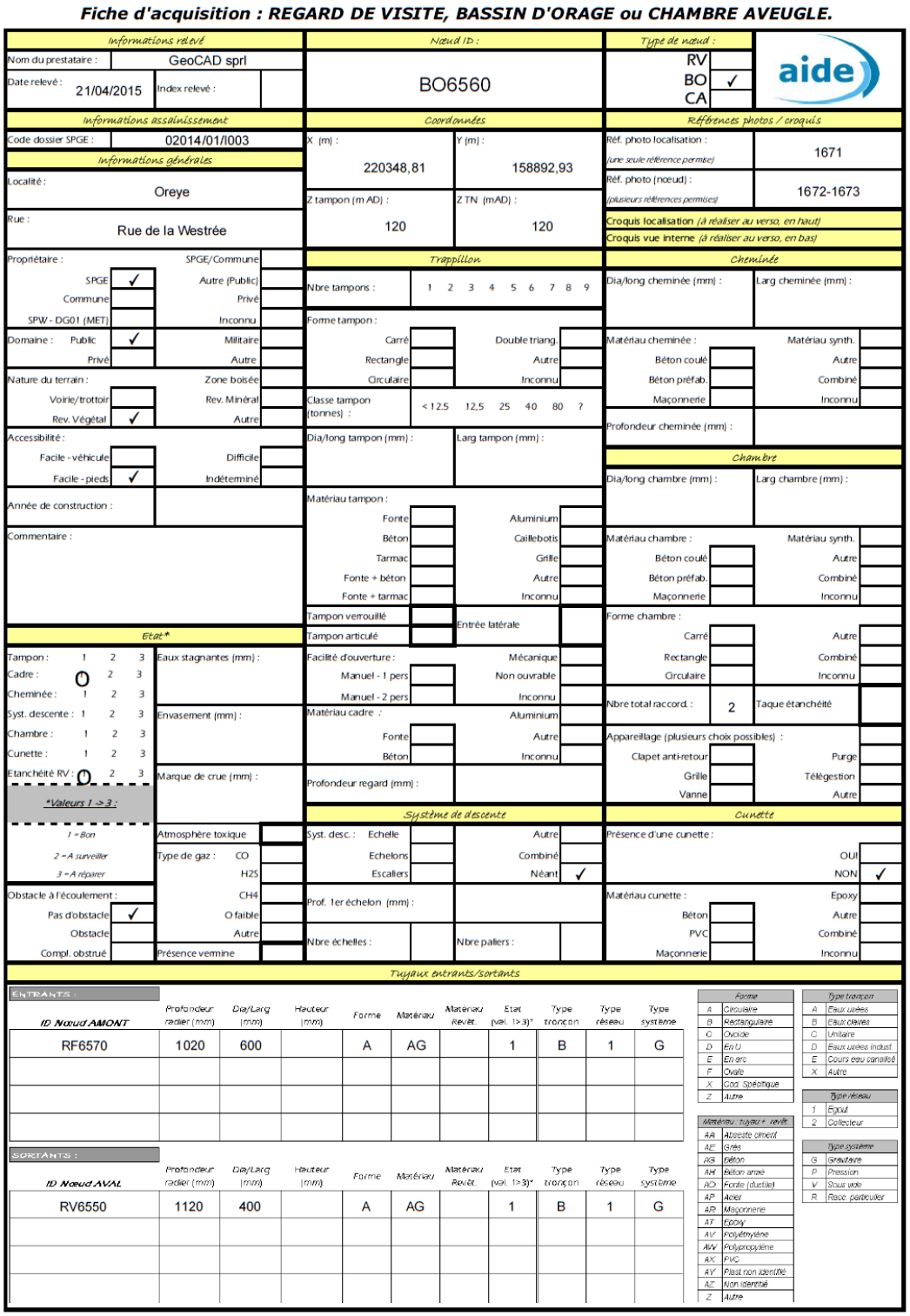
Sharma, S. (2015). An Extended Classification and Comparison of NoSQL Big Data Models.

Winand, M. (s.d.). *L'arbre de recherche (B-Tree) rend l'index rapide*. Use the Index, Luke. A guide to database performance for developers: http://use-the-index-luke.com/fr/sql/anatomie-dun-index/le-b-tree. Consulté le 16 août 2017.

## Liste des annexes

# ANNEXES

## Annexe 1. Exemple de fiche de regard



**Fiche d'acquisition : REGARD DE VISITE, BASSIN D'ORAGE ou CHAMBRE AVEUGLE.**

| Informations relevé | | Nœud ID : | Type de nœud : |
|---|---|---|---|
| Nom du prestataire : | GeoCAD sprl | BO6560 | RV / BO ✓ / CA |
| Date relevé : | 21/04/2015 | Index relevé : | |

| Informations assainissement | | Coordonnées | | Références photos / croquis | |
|---|---|---|---|---|---|
| Code dossier SPGE : | 02014/01/I003 | X (m) : | Y (m) : | Réf. photo localisation : (une seule référence permise) | 1671 |
| | | 220348,81 | 158892,93 | Réf. photo (nœud) : (plusieurs références permises) | 1672-1673 |
| Informations générales | | Z tampon (m AD) : | Z TN (mAD) : | Croquis localisation (à réaliser au verso, en haut) | |
| Localité : | Oreye | 120 | 120 | Croquis vue interne (à réaliser au verso, en bas) | |
| Rue : | Rue de la Westrée | | | | |

| Propriétaire : | | SPGE/Commune | | Trappillon | | | | | | | | | Cheminée | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SPGE | ✓ | Autre (Public) | | Nbre tampons : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Dia/long cheminée (mm) : | | Larg cheminée (mm) : | |
| Commune | | Privé | | Forme tampon : | | | | | | | | | | | | | |
| SPW - DGO1 (MET) | | Inconnu | | Carré | | | | Double triang. | | | | | | Matériau cheminée : | | Matériau synth. | |
| Domaine : Public | ✓ | Militaire | | Rectangle | | | | Autre | | | | | | Béton coulé | | Autre | |
| Privé | | Autre | | Circulaire | | | | Inconnu | | | | | | Béton préfab. | | Combiné | |
| Nature du terrain : | | Zone boisée | | Classe tampon (tonnes) : | < 12.5 | 12,5 | 25 | 40 | 80 | ? | | | | Maçonnerie | | Inconnu | |
| Voirie/trottoir | | Rev. Minéral | | | | | | | | | | | | Profondeur cheminée (mm) : | | | |
| Rev. Végétal | ✓ | Autre | | | | | | | | | | | | | | | |
| Accessibilité : | | | | Dia/long tampon (mm) : | | | | Larg tampon (mm) : | | | | | | Chambre | | | |
| Facile - véhicule | | Difficile | | | | | | | | | | | | Dia/long chambre (mm) : | | Larg chambre (mm) : | |
| Facile - pieds | ✓ | Indéterminé | | Matériau tampon : | | | | | | | | | | | | | |
| Année de construction : | | | | Fonte | | | | Aluminium | | | | | | Matériau chambre : | | Matériau synth. | |
| Commentaire : | | | | Béton | | | | Caillebotis | | | | | | Béton coulé | | Autre | |
| | | | | Tarmac | | | | Grille | | | | | | Béton préfab. | | Combiné | |
| | | | | Fonte + béton | | | | Autre | | | | | | Maçonnerie | | Inconnu | |
| | | | | Fonte + tarmac | | | | Inconnu | | | | | | | | | |
| | | | | Tampon verrouillé | | | | Entrée latérale | | | | | | Forme chambre : | | | |
| | | | | Tampon articulé | | | | | | | | | | Carré | | Autre | |
| | | | | Facilité d'ouverture : | | | | Mécanique | | | | | | Rectangle | | Combiné | |
| | | | | Manuel - 1 pers | | | | Non ouvrable | | | | | | Circulaire | | Inconnu | |
| | | | | Manuel - 2 pers | | | | Inconnu | | | | | | Nbre total raccord. : | 2 | Taque étanchéité | |
| | | | | Matériau cadre : | | | | Aluminium | | | | | | | | | |
| | | | | Fonte | | | | Autre | | | | | | Appareillage (plusieurs choix possibles) : | | | |
| | | | | Béton | | | | Inconnu | | | | | | Clapet anti-retour | | Purge | |
| | | | | Profondeur regard (mm) : | | | | | | | | | | Grille | | Télégestion | |
| | | | | | | | | | | | | | | Vanne | | Autre | |

| Etat* | | | | Eaux stagnantes (mm) : | Système de descente | | | Cunette | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Tampon : | 1 | 2 | 3 | | Syst. desc. : Echelle | | Autre | Présence d'une cunette : | | |
| Cadre : | 0 | 2 | 3 | | Echelons | | Combiné | | OUI | |
| Cheminée : | 1 | 2 | 3 | | Escaliers | | Néant ✓ | | NON | ✓ |
| Syst. descente : | 1 | 2 | 3 | Envasement (mm) : | Prof. 1er échelon (mm) : | | | Matériau cunette : | | Epoxy |
| Chambre : | 1 | 2 | 3 | | | | | Béton | | Autre |
| Cunette : | 1 | 2 | 3 | | Nbre échelles : | | Nbre paliers : | PVC | | Combiné |
| Etanchéité RV : | 0 | 2 | 3 | Marque de crue (mm) : | | | | Maçonnerie | | Inconnu |

*Valeurs 1 -> 3 :*
1 = Bon
2 = A surveiller
3 = A réparer

| Obstacle à l'écoulement : | | Atmosphère toxique | |
|---|---|---|---|
| Pas d'obstacle | ✓ | Type de gaz : CO | |
| Obstacle | | H2S | |
| Compl. obstrué | | CH4 | |
| | | O faible | |
| | | Autre | |
| | | Présence vermine | |

Tuyaux entrants/sortants

ENTRANTS :

| ID Noeud AMONT | Profondeur radier (mm) | Dia/Larg (mm) | Hauteur (mm) | Forme | Matériau | Matériau Revêt. | Etat (val. 1>3)* | Type tronçon | Type réseau | Type système |
|---|---|---|---|---|---|---|---|---|---|---|
| RF6570 | 1020 | 600 | | A | AG | | 1 | B | 1 | G |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

SORTANTS :

| ID Noeud AVAL | Profondeur radier (mm) | Dia/Larg (mm) | Hauteur (mm) | Forme | Matériau | Matériau Revêt. | Etat (val. 1>3)* | Type tronçon | Type réseau | Type système |
|---|---|---|---|---|---|---|---|---|---|---|
| RV6550 | 1120 | 400 | | A | AG | | 1 | B | 1 | G |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

| Forme | |
|---|---|
| A | Circulaire |
| B | Rectangulaire |
| C | Ovoïde |
| D | En U |
| E | En arc |
| F | Ovale |
| X | Coq. Spécifique |
| Z | Autre |

| Matériau tuyaux + revêt. | |
|---|---|
| AA | Absente d'mont |
| AE | Grès |
| AG | Béton |
| AH | Béton armé |
| AO | Fonte (ductile) |
| AP | Acier |
| AR | Maçonnerie |
| AT | Epoxy |
| AV | Polyéthylène |
| AW | Polypropylène |
| AX | PVC |
| AY | Plast non identifié |
| AZ | Non identifié |
| Z | Autre |

| Type tronçon | |
|---|---|
| A | Eaux usées |
| B | Eaux claires |
| C | Unitaire |
| D | Eaux usées indust. |
| E | Cours eau canalisé |
| X | Autre |

| Type réseau | |
|---|---|
| 1 | Egout |
| 2 | Collecteur |

| Type système | |
|---|---|
| G | Gravitaire |
| P | Pression |
| V | Sous vide |
| R | Rejet particulier |

## Annexe 2. Programme JavaScript de transformation d'un CSV en JSON

```
// Source: http://www.bennadel.com/blog/1504-Ask-Ben-Parsing-CSV-Strings-With-
Javascript-Exec-Regular-Expression-Command.htm
// This will parse a delimited string into an array of
// arrays. The default delimiter is the comma, but this
// can be overriden in the second argument.

function CSVToArray(strData, strDelimiter) {
    // Check to see if the delimiter is defined. If not,
    // then default to comma.
    strDelimiter = (strDelimiter || ",");
    // Create a regular expression to parse the CSV values.
    var objPattern = new RegExp((
    // Delimiters.
    "(\\" + strDelimiter + "|\\r?\\n|\\r|^)" +
    // Quoted fields.
    "(?:\"([^\"]*(?:\"\"[^\"]*)*)\"|" +
    // Standard fields.
    "([^\"\\" + strDelimiter + "\\r\\n]*))"), "gi");
    // Create an array to hold our data. Give the array
    // a default empty first row.
    var arrData = [[]];
    // Create an array to hold our individual pattern
    // matching groups.
    var arrMatches = null;
    // Keep looping over the regular expression matches
    // until we can no longer find a match.
    while (arrMatches = objPattern.exec(strData)) {
        // Get the delimiter that was found.
        var strMatchedDelimiter = arrMatches[1];
        // Check to see if the given delimiter has a length
        // (is not the start of string) and if it matches
        // field delimiter. If id does not, then we know
        // that this delimiter is a row delimiter.
        if (strMatchedDelimiter.length && (strMatchedDelimiter != strDelimiter))
{
            // Since we have reached a new row of data,
            // add an empty row to our data array.
            arrData.push([]);
        }
        // Now that we have our delimiter out of the way,
        // let's check to see which kind of value we
        // captured (quoted or unquoted).
        if (arrMatches[2]) {
            // We found a quoted value. When we capture
            // this value, unescape any double quotes.
            var strMatchedValue = arrMatches[2].replace(
            new RegExp("\"\"", "g"), "\"");
        } else {
            // We found a non-quoted value.
            var strMatchedValue = arrMatches[3];
        }
        // Now that we have our value string, let's add
        // it to the data array.
        arrData[arrData.length - 1].push(strMatchedValue);
    }
    // Return the parsed data.
    return (arrData);
```

```
}

function CSV2JSON(csv) {
    var array = CSVToArray(csv);
    var objArray = [];
    for (var i = 1; i < array.length; i++) {
        objArray[i - 1] = {};
        for (var k = 0; k < array[0].length && k < array[i].length; k++) {
            var key = array[0][k];
            objArray[i - 1][key] = array[i][k]
        }
    }

    var json = JSON.stringify(objArray);
    var str = json.replace(/},/g, "},\r\n");

    return str;
}

$("#convert").click(function() {
    var csv = $("#csv").val();
    var json = CSV2JSON(csv);
    $("#json").val(json);
});

$("#download").click(function() {
    var csv = $("#csv").val();
    var json = CSV2JSON(csv);
    window.open("data:text/json;charset=utf-8," + escape(json))
});
```

## Annexe 3. Sélection des documents écrasées lors de l'importation

```
import csv
from os import chdir
chdir ("C:/Users/amand/Documents/Am/Master2/memoire/donnees/point_taques")
fname="tuyaux_sortant_2.csv"
file = open (fname,"rb")

reader=csv.reader(file)
write=csv.writer(open("tuyaux_sortant_3.csv","wb"))
id_double=[]    #création de la liste qui comprendra les id traités
rang=1  #création d'un rang pour l'écriture des id dans la liste
x=0
for row in reader:      #première boucle qui parcourt toutes les ID
    id=row[0]  #on nomme les id concerné et en amont
    id_am=row[1]

    rang=rang+1 #incrementation du rang
    ok=id in id_double
    id_double.append(id)    #on inscrit l'id traité pour ne pas le traiter deux
fois


    if ok == True:      #si cet indice a déjà été traité passer au suivant
        pass


    else:
        fname2="tuyaux_sortant_2.csv"   #nouvelle ouverture du fichier
        file2 = open (fname2,"rb")  #pour pouvoir créer une nouvelle boucle
        prev_row=row    #on stocke la colonne car ce sont les premières qui ne
sont pas dans mongodb
        reader2=csv.reader(file2)
        for row in reader2:
            id_2=row[0]             #On stock l'id du regard concerné mais aussi
celui à l'opposé du tuyau
            id_am2=row[1]

            if id== id_2 and id_am!=id_am2:
                write.writerow(prev_row)
                print  prev_row
                prev_row=row

            #Si cet id est bien le même et que le tuple est bien différent que le
premier
            #la ligne est écrite dans le csv
print x
file.close()
file2.close()
```

## Annexe 4. Code python pour l'ajout des tuyaux supplémentaires

```
import csv
from os import chdir
import pymongo
from pymongo import MongoClient

import sys
client = MongoClient()  #connexion à la base de données
db=client.testmem
a=pymongo.version       #vérification des versions pour assurer le fonctionnement
de la suite
print (a)
b=sys.version
print (b)
chdir ("C:/Users/amand/Documents/Am/Master2/memoire/donnees/point_taques")

fname="tuyaux_sortant_2.csv"        #import du CSV contenant les documents
sélectionné par le code précédent
file = open (fname,"rt",encoding='utf8')
c=db.regards_array.find_one()           #affichage d'un document pour
illustration
print (c)
reader=csv.reader(file)                 #ouverture du CSV
for row in reader:

    manip=db.regards_array.update_many({"_id":row[0]},

{'$push':{"tuyaux_sortant":{"_id":row[0],"ref_noeud_aval":row[1],"suffixe_c":row[
2],"index_releve":row[3],"prof_radier":
row[4],"dia_larg":row[5],"hauteur":row[6],"forme tuyau":row[7],"materiau
tuyau":row[8],"materiau_revetement":row[9],"code_etat":row[10],"type_assain":row[
11],"type_troncon":row[12],"demergement":row[13],"type_systeme":row[14],"type_noe
ud":row[15],"indice_tuyau":row[16]}}})
    print (row[0])
# boucle qui permet d'insérer chacun des documents du CSV dans le tableau de la
collection


file.close()
#fermeture du fichier
```

## Annexe 5. Code python requete 1 et écriture dans fichier

```
import json
import pymongo
from pymongo import MongoClient
from bson.json_util import dumps
from bson import Binary, Code
from bson import json_util

import sys
client = MongoClient()
db=client.testmem
collection=db.cadastre_point            #définition collection source
collection_2=db.regards_array           #définition collection cible
id="64056B0552/00W003"                  #définition id source
dist=0.00006032673383261347             #définition distance en radian

#selection d'un point selon son id
#exemple équivalent en javascript (fonctionne dans compasse pour affichage)
#{"_id":"64056B0552/00W003","geometry":{"$geoWithin":{"$centerSphere":[[5.3559020
209440575,50.72806354784038],0.00006032673383261347]}}}

point_rech=collection.find_one({"_id":id},{"geometry.coordinates":1,"_id":0})
#affiche le document correspondant
print(point_rech)
geom=point_rech['geometry']
print(geom)
coord=geom['coordinates']                   #extraction des coordonnées du point
print("coord=",type(coord))                 #vérification du type



query={"geometry":{'$within':{'$centerSphere':[coord,dist]}}}       #recherche
des regards autour du point en radian
regard_proche=collection_2.find(query)
print(regard_proche)
x=0
#for doc in regard_proche:                          #affichage du resultat
    # print(doc)
     #x=x+1                                        #compte

json_objs=[]
for doc in regard_proche:                          #affichage du resultat
    json_obj=json.dumps(doc,default=json_util.default,indent=4)
    json_objs.append(json_obj)
    x=x+1                                          #compte

chn = " ".join(json_objs)                          #transformation de la liste en
chaine de caractère
print(chn)                                         #affichage résultat

f_read=
open("C:/Users/amand/Documents/Am/Master2/memoire/donnees/point_taques/result.jso
n", "w")
f_read.write(chn)                                  #écriture du résultat dans un
fichier JSON
print(type(json_objs))
print(x)                                           #écriture du compte
```

## Annexe 6. Code python requête 2 et écriture dans fichier

```
import json
import pymongo
from pymongo import MongoClient
from bson.json_util import dumps
from bson import Binary, Code
from bson import json_util
import re

import sys
client = MongoClient()
db=client.testmem
collection=db.regards_array         #définition collection source
collection_2=db.regards_array           #définition collection cible
id="64056-02CA006590"                   #définition id source
json_objs=[]
#selection d'un point selon son id
#exemple équivalent en javascript (fonctionne dans compasse pour affichage)
#{"_id":"64056B0552/00W003","geometry":{"$geoWithin":{"$centerSphere":[[5.3559020
209440575,50.72806354784038],0.000060326733832613 47]}}}

print(collection.find({"_id":id},{'tuyaux_entrant':{'$exists':1}}).count(True))

while collection.find({"_id":id},{'tuyaux_entrant':{'$exists':1}}).count(True) >
0:

point_rech=collection.find_one({"_id":id},{"tuyaux_entrant.ref_noeud_amont":1,"_i
d":0})         #affiche le document correspondant
    print((point_rech))

    ref=point_rech['tuyaux_entrant']        #affiche contenu tuyaux entrant
    ref_str=str(ref)                #transformation de la liste en caractère
    list_ref=(ref_str.split (':'))          #division en champ et valeur
    id_ref= list_ref[1]
    m=re.search("'(.+?)'}]",id_ref)         #isolation de la valeur
    if m:
        id_amont=m.group(1)
    print('ID AMONT',id_amont)
    id=id_amont
    print(collection.find({'$and':
[{"_id":id},{'tuyaux_entrant':{'$exists':1}}]}).count(True))
    doc=collection.find_one({"_id":id})
    json_obj=json.dumps(doc,default=json_util.default,indent=4)
    json_objs.append(json_obj)
    if collection.find({'$and':
[{"_id":id},{'tuyaux_entrant':{'$exists':1}}]}).count(True) == 0:
        break                   #s'arreter si plus de tuyaux en amont


chn = " ".join(json_objs)                       #transformation de la liste en
chaine de caractère
print(chn)
f_read=
open("C:/Users/amand/Documents/Am/Master2/memoire/donnees/point_taques/result.jso
n", "w")
f_read.write(chn)
```

## Annexe 7. Code HTML visualisation fichier JSON

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1.0" />
    <title>Drag and Drop GeoJSON</title>
    <style>
      html { height: 100% }
      body { height: 100%; margin: 0; padding: 0; overflow: hidden; }
      #map { height: 100% }
      #drop-container {
        display: none;
        height: 100%;
        width: 100%;
        position: absolute;
        z-index: 1;
        top: 0px;
        left: 0px;
        padding: 20px;
        background-color: rgba(100, 100, 100, 0.5);
      }
      #drop-silhouette {
        color: white;
        border: white dashed 8px;
        height: calc(100% - 56px);
        width: calc(100% - 56px);
        background-image:
url('
IArs4c6QAAAAZiS0dEAGQAZABkkPCsTwAAAAlwSFlzAAALEwAACxMBAJqcGAAAAAd0SU1FB90LHAIvICW
dsKwAAAAZdEVYdENvbW1lbnQAQ3JlYXRlZCB3aXRoIEdJTVBXgQ4XAAACdklEQVR42u3csU7icBzA8Xp3
GBMSeRITH8JHMY7cRMvmVmXoE9TAcJubhjD4ApoiopgqDMWAKAgIcSAiCfxuwhwROVJbkPD9rP23ob8vp
ZCQKgoAAAAAAAAAAPDYyiK/eNM05bNtr6+vSjgcXiHxDMkE1WpVFvGcfpCVICAIQUAQgoAgBAFBCAKCgC
AEAUEIAoIQBAQhCAgCghAEBCEICEIQEIQgIAgIQhAQhCAgCEFAEIKAICAIQUAQgoAgBAFBCDIzhmFINBo
9/K6D0XVddnd3ZaneDY7jSCqVcn3SfjyeKRKJbJ2dnYllWbKUl2i5XJaXlxdJJBIy7yDHx8fy9vYm6XR6
OWMM3d/fi4hIqVSSWCwmsw5ycHAgrVZLRETOz8+XO8ZQpVJ5H2Y6nRZN0/b9DqLruhSLxfd9MpkMMT6L0
uv1JJlMih9BhveJwWDwvv7i4oIY4zw8PIwMtt1uSzweF6+CHB0dSbfbHVmbzWaJMcnj4+OHAd/d3cne3p
64DWKapjw/P39Yd3l5SYxpVKvVsYO2LEtUVd2ZNoiu6+I4ztg1V1dXxPAiSq/Xk5OTk0k9pNVqyenp6ch
94l+5XI4YbtRqNfHa9fX1t43xcwGa/Nnc3PwdDAY9OZht28rGxgZPvP6KSCSy9fT09OUrw7ZtPqa8jFKv
113HuLm5IYbXVFXdcRPl9vaWGH5GaTQaU8fI5/PE8JumafvNZvO/MQqFAjFmJRqNHk6Ksqgx5vr1zzAM2
d7edr3/6uqqsra2NnZbp9NR+v2+62OHQqG5zObXPIMEAgFlfX3dl2N79btl1viTA0FAEIKAIAQBAAAAAA
AAsMz+Ai1bUgo6ebm8AAAAAElFTkSuQmCC');
        background-repeat: no-repeat;
        background-position: center;
      }
    </style>
  </head>
  <body>
    <div id="map"></div>
    <div id="drop-container"><div id="drop-silhouette"></div></div>
    <script>
      /* Map functions key = AIzaSyDOanklVzed_PB6qP_GzLmu_NU8-hz9D5s*/

      var map;

      function initMap() {
        // set up the map
        map = new google.maps.Map(document.getElementById('map'), {
```

```
        center: new google.maps.LatLng(0, 0),
        zoom: 2
      });
    }

    function loadGeoJsonString(geoString) {
      var geojson = JSON.parse(geoString);
      map.data.addGeoJson(geojson);
      zoom(map);
    }

    /**
     * Update a map's viewport to fit each geometry in a dataset
     * @param {google.maps.Map} map The map to adjust
     */
    function zoom(map) {
      var bounds = new google.maps.LatLngBounds();
      map.data.forEach(function(feature) {
        processPoints(feature.getGeometry(), bounds.extend, bounds);
      });
      map.fitBounds(bounds);
    }

    /**
     * Process each point in a Geometry, regardless of how deep the points may
lie.
     * @param {google.maps.Data.Geometry} geometry The structure to process
     * @param {function(google.maps.LatLng)} callback A function to call on
each
     *     LatLng point encountered (e.g. Array.push)
     * @param {Object} thisArg The value of 'this' as provided to 'callback'
(e.g.
     *     myArray)
     */
    function processPoints(geometry, callback, thisArg) {
      if (geometry instanceof google.maps.LatLng) {
        callback.call(thisArg, geometry);
      } else if (geometry instanceof google.maps.Data.Point) {
        callback.call(thisArg, geometry.get());
      } else {
        geometry.getArray().forEach(function(g) {
          processPoints(g, callback, thisArg);
        });
      }
    }


    /* DOM (drag/drop) functions */

    function initEvents() {
      // set up the drag & drop events
      var mapContainer = document.getElementById('map');
      var dropContainer = document.getElementById('drop-container');

      // map-specific events
      mapContainer.addEventListener('dragenter', showPanel, false);

      // overlay specific events (since it only appears once drag starts)
      dropContainer.addEventListener('dragover', showPanel, false);
```

```
        dropContainer.addEventListener('drop', handleDrop, false);
        dropContainer.addEventListener('dragleave', hidePanel, false);
      }

      function showPanel(e) {
        e.stopPropagation();
        e.preventDefault();
        document.getElementById('drop-container').style.display = 'block';
        return false;
      }

      function hidePanel(e) {
        document.getElementById('drop-container').style.display = 'none';
      }

      function handleDrop(e) {
        e.preventDefault();
        e.stopPropagation();
        hidePanel(e);

        var files = e.dataTransfer.files;
        if (files.length) {
          // process file(s) being dropped
          // grab the file data from each file
          for (var i = 0, file; file = files[i]; i++) {
            var reader = new FileReader();
            reader.onload = function(e) {
              loadGeoJsonString(e.target.result);
            };
            reader.onerror = function(e) {
              console.error('reading failed');
            };
            reader.readAsText(file);
          }
        } else {
          // process non-file (e.g. text or html) content being dropped
          // grab the plain text version of the data
          var plainText = e.dataTransfer.getData('text/plain');
          if (plainText) {
            loadGeoJsonString(plainText);
          }
        }

        // prevent drag event from bubbling further
        return false;
      }


      function initialize() {
        initMap();
        initEvents();
      }
    </script>
    <script async defer

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDOanklVzed_PB6qP_GzLmu_NU8
-hz9D5s&callback=initialize"></script>
  </body>
</html>
```

## Annexe 8. Extrait résultat requête 2


```
{
  "type": "FeatureCollection",
  "features":
[
{
    "_id": "64056-02JO006600",
    "type": "Feature",
    "properties": {
        "code_incre": 6600,
        "type_noeud": "JO",
        "nom_presta": "GeoCAD sprl",
        "date_relev": "21-04-15",
        "index_rele": 1,
        "code_dossi": "02014/01/I003",
        "resetat": "SEE",
        "step_code": "64056/02",
        "step_statu": 1,
        "cours_d_ea": null,
        "sous_bassi": "Meuse aval",
        "AggloCode": null,
        "type_syste": null,
        "territoire": 1,
        "localite": "OREYE",
        "commune": "Oreye",
        "code_ins": 64056,
        "postalcode": 4360,
        "rue": "RUE DE LA WESTR\ufffdE",
        "proprietai": "SPGE",
        "domaine": "PU",
        "nature_ter": "VO",
        "accessibil": "FV",
        "annee_cons": null,
        "commentair": null,
        "x": 219713.47,
        "y": 158284.0,
        "z": 104.63,
        "z_tampon": 0.0,
        "nbr_tampon": null,
        "forme_tamp": null,
        "classe_tam": null,
        "dia_long_t": null,
        "largeur_ta": null,
        "materiau_t": null,
        "tampon_a_g": "FAUX",
        "tampon_ver": "FAUX",
        "entree_lat": "FAUX",
        "facilite_o": null,
        "materiau_c": null,
        "prof_radie": null,
        "systeme_de": null,
        "prof_premi": null,
        "nbr_echell": null,
        "nbr_palier": null,
        "forme_chem": null,
        "dia_long_c": null,
        "largeur_ch": null,
        "materiau_1": null,
```

```
        "prof_chemi": null,
        "dia_long_1": null,
        "largeur__1": null,
        "materiau_2": null,
        "forme_cham": null,
        "nbr_racc_e": null,
        "taque_etan": "FAUX",
        "appareilla": null,
        "materiau_3": null,
        "forme_cune": null,
        "etat_tampo": null,
        "etat_cadre": null,
        "etat_chemi": null,
        "etat_syste": null,
        "etat_chamb": null,
        "etat_cunet": null,
        "etancheite": null,
        "obstacle_e": null,
        "eaux_stagn": null,
        "envasement": null,
        "marque_cru": null,
        "atmosphere": "FAUX",
        "type_gaz": null,
        "vermine": "FAUX",
        "stockage_p": "A",
        "ref_photo_": null,
        "ref_photo": null,
        "ref_suppor": null,
        "ref_croqui": null,
        "ref_croq_1": null,
        "presence_t": null,
        "z_radier_t": null,
        "etat_ta": null,
        "notes": null,
        "NoTampon": "VRAI",
        "NoCunette": "VRAI",
        "NoSystemDe": "VRAI",
        "NoCheminee": "VRAI",
        "NoChambre": "VRAI",
        "user_text_": null
    },
    "geometry": {
        "type": "Point",
        "coordinates": [
            5.356232436947709,
            50.73091531471972
        ]
    },
    "tuyaux_entrant": [
        {
            "ref_noeud_amont": "64056-02RV006610",
            "suffixe_c": "FAUX",
            "_id": "64056-02JO006600",
            "index_releve": "1",
            "prof_radier": "0",
            "dia_larg": "150",
            "hauteur": "150",
            "forme tuyau": "A",
            "materiau tuyau": "AX",
```

```
            "materiau_revetement": "",
            "code_etat": "1",
            "type_assain": "1",
            "type_troncon": "G",
            "type_systeme": "C",
            "demergement": "0",
            "resexutoire": "0",
            "type_noeud": "JO",
            "indice_tuyau": "1"
        }
    ],
    "tuyaux_sortant": [
        {
            "_id": "64056-02JO006600",
            "ref_noeud_aval": "64056-02CA006590",
            "suffixe_c": "FAUX",
            "index_releve": "1",
            "prof_radier": "0",
            "dia_larg": "300",
            "hauteur": "300",
            "forme tuyau": "A",
            "materiau tuyau": "AG",
            "materiau_revetement": "",
            "code_etat": "1",
            "type_assain": "1",
            "type_troncon": "G",
            "demergement": "0",
            "type_systeme": "C",
            "type_noeud": "JO",
            "indice_tuyau": "1"
        }
    ]
} {
    "_id": "64056-02RV006610",
    "type": "Feature",
    "properties": {
        "code_incre": 6610,
        "type_noeud": "RV",
        "nom_presta": "GeoCAD sprl",
        "date_relev": "21-04-15",
        "index_rele": 1,
        "code_dossi": "02014/01/I003",
        "resetat": "SEE",
        "step_code": "64056/02",
        "step_statu": 1,
        "cours_d_ea": null,
        "sous_bassi": "Meuse aval",
        "AggloCode": null,
        "type_syste": null,
        "territoire": 1,
        "localite": "OREYE",
        "commune": "Oreye",
        "code_ins": 64056,
        "postalcode": 4360,
        "rue": "RUE DE LA WESTR\ufffdE",
        "proprietai": "SPGE",
        "domaine": "PU",
        "nature_ter": "VO",
        "accessibil": "FV",
```

```
        "annee_cons": null,
        "commentair": null,
        "x": 219782.38,
        "y": 158343.0,
        "z": 107.62,
        "z_tampon": 107.62,
        "nbr_tampon": 1,
        "forme_tamp": "CA",
        "classe_tam": "40",
        "dia_long_t": 600,
        "largeur_ta": 600,
        "materiau_t": "F",
        "tampon_a_g": "FAUX",
        "tampon_ver": "FAUX",
        "entree_lat": "FAUX",
        "facilite_o": "M1",
        "materiau_c": "F",
        "prof_radie": 390,
        "systeme_de": null,
        "prof_premi": null,
        "nbr_echell": null,
        "nbr_palier": null,
        "forme_chem": null,
        "dia_long_c": null,
        "largeur_ch": null,
        "materiau_1": null,
        "prof_chemi": null,
        "dia_long_1": 500,
        "largeur__1": 500,
        "materiau_2": "BP",
        "forme_cham": "CA",
        "nbr_racc_e": 2,
        "taque_etan": "FAUX",
        "appareilla": null,
        "materiau_3": "P",
        "forme_cune": null,
        "etat_tampo": 1,
        "etat_cadre": 1,
        "etat_chemi": null,
        "etat_syste": null,
        "etat_chamb": 1,
        "etat_cunet": 1,
        "etancheite": 1,
        "obstacle_e": 1,
        "eaux_stagn": null,
        "envasement": null,
        "marque_cru": null,
        "atmosphere": "FAUX",
        "type_gaz": null,
        "vermine": "FAUX",
        "stockage_p": "A",
        "ref_photo_": "RV6610_01_1679.JPG",
        "ref_photo": "RV6610_02_1680.JPG",
        "ref_suppor": null,
        "ref_croqui": null,
        "ref_croq_1": null,
        "presence_t": null,
        "z_radier_t": null,
        "etat_ta": null,
```

```
        "notes": null,
        "NoTampon": "FAUX",
        "NoCunette": "FAUX",
        "NoSystemDe": "VRAI",
        "NoCheminee": "VRAI",
        "NoChambre": "FAUX",
        "user_text_": null
    },
    "geometry": {
        "type": "Point",
        "coordinates": [
            5.357219554925059,
            50.7314374165587
        ]
    },
    "tuyaux_entrant": [
        {
            "ref_noeud_amont": "64056-02RV006620",
            "suffixe_c": "FAUX",
            "_id": "64056-02RV006610",
            "index_releve": "1",
            "prof_radier": "380",
            "dia_larg": "150",
            "hauteur": "150",
            "forme tuyau": "A",
            "materiau tuyau": "AX",
            "materiau_revetement": "",
            "code_etat": "1",
            "type_assain": "1",
            "type_troncon": "G",
            "type_systeme": "C",
            "demergement": "0",
            "resexutoire": "0",
            "type_noeud": "RV",
            "indice_tuyau": "1"
        }
    ],
    "tuyaux_sortant": [
        {
            "_id": "64056-02RV006610",
            "ref_noeud_aval": "64056-02JO006600",
            "suffixe_c": "FAUX",
            "index_releve": "1",
            "prof_radier": "400",
            "dia_larg": "150",
            "hauteur": "150",
            "forme tuyau": "A",
            "materiau tuyau": "AX",
            "materiau_revetement": "",
            "code_etat": "1",
            "type_assain": "1",
            "type_troncon": "G",
            "demergement": "0",
            "type_systeme": "C",
            "type_noeud": "RV",
            "indice_tuyau": "1"
        }
    ]
}
```

## Annexe 9. Script python GridFS

```
import gridfs
import pymongo
from pymongo import MongoClient
import os.path
from os import chdir
import glob
chemin="C:/Users/amand/Documents/Am/Master2/memoire/donnees/Photos"

chdir ("C:/Users/amand/Documents/Am/Master2/memoire/donnees/Photos")
list=(os.listdir('.'))
paths=glob.glob("C:/Users/amand/Documents/Am/Master2/memoire/donnees/Photos/*")
# client = MongoClient()
# db=client.testmem
# fs = gridfs.GridFS(db)
print (type(list[1]))

# for p in list:                #option pour entrer tous les fichiers en un coup
    # print (p)
    # fichier=open(p, "r")
    # stored=fs.put(fichier.read(),content_type="image/jpeg",filename=p)
# for fichier in os.walk(path):
    # print(os.path.join(fichier))



# from element in os.listdir('')

# read in the image.
filename =
"C:/Users/amand/Documents/Am/Master2/memoire/donnees/Photos/BO6560_01_1671.JPG"
datafile = open(filename,"r+");
thedata = datafile.read()

# connect to database

connection = pymongo.Connection("localhost",27017);
database = connection['testmem']

# create a new gridfs object.
fs = gridfs.GridFS(database)

# store the data in the database. Returns the id of the file in gridFS
stored = fs.put(thedata, filename="testimage")

# retrieve what was just stored.
outputdata =fs.get(stored).read()

datafile.close()
```