Republique Algerienne Démocratique et populaire
Ministére de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Sadik Benyahia de Jijel
Faculté des Sciences Exactes et d'informatique
Département d'Informatique

Mémoire
de fin d'etude pour obtention du
diplôme Master de Recherche en
Informatique
Option : Systèmes d'Information et Aide à la Décision
Théme

Développement d'un système de gestion des méta-données dans les data lakes à base de sources NoSQL

Réaliser par :
Ouahab Racha

Encadré par :
Mr BOUKRAA Doulkifli

Année universitaire : 2020\2021

Remerciements

Je remercie Dieu le tout puissant de m’avoir donné le courage et la volonté d'achever ce travail et sans lequel il n’aurait jamais été accompli.

Mes remerciements les plus sincères, accompagnés de toute notre gratitude vont tout d’abord à notre encadreur **Mr. BOUKRAA Doulkifli**, pour nous avoir proposé ce sujet et pour ses précieux conseils et de nous avoir dirigé durant mon projet, et surtout pour la confiance qu’il m’a accordé pour la réalisation de ce projet.

Je remercie tous les enseignants de la faculté des Sciences Exactes et d’informatique et surtout ceux du département informatique.

Je remercie les membres de jury pour m’avoir fait l’honneur de juger mon travail.

Enfin, je remercie vont à toute personne ayant contribué, de près ou de loin, à l’aboutissement de ce travail.

Résumé

Les data lakes (lacs de données) permettent de stocker les données massives (BigData) à des fins d'utilisations diverses (décisionnelles, apprentissage automatique...). Les métadonnées jouent un rôle primordial dans le sens qu'elles permettent de connaitre et d'accéder aux données renfermées dans le data lake.

Le travail présenté dans ce mémoire a pour objectif de développer un système basé sur les graphes qui permet de gérer les métadonnées dans un data lake composé de sources NoSQL (graphes, documents et relationelles).

Mots-Clés : BigData , Lacs de données, Métadonnées , Gestion de métadonnées, NOSQL, Base de données orientée graphe.

Abstract

Data lakes are used to store massive data (BigData) for various purposes (decision-making, machine learning, etc.). Metadata play an essential role in the sense that they allow to know and access the data contained in the data lake.

The aim of the work presented is to develop a graph-based system that allows you to manage metadata in a data lake made up of NoSQL sources (graphs, documents and relational).

Keywords: BigData, DataLakes, Metadata, Metadata Management, NoSQL, Graph-oriented database.

ملخص

تستخدم بحيرات البيانات لتخزين البيانات الضخمة لأغراض مختلفة (إتخاذ القرار، والتعلم الآلي، وما إلى ذلك). تلعب البيانات الوصفية دورًا أساسيًا حيث أنها تسمح بمعرفة والوصول إلى البيانات الموجودة في بحيرة البيانات

الهدف من العمل المقدم في هذه الأطروحة هو تطوير نظام قائم على الرسم البياني يسمح لك بإدارة البيانات الوصفية في بحيرة البيانات المكونة من مصادرNoSQL(graphes, documents , relationelles).

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Depuis une vingtaine d'années, les données générées n'ont fait que s'accroître. Actuellement nous produisons annuellement une masse de données très importante. Cet accroissement des données touche tous les secteurs, tant scientifiques qu'économiques, ainsi que le la vie de tous les jours, caractérisé entre autres par un monde fortement connecté à travers les réseaux sociaux. L'explosion quantitative des données numériques a obligé les chercheurs et praticiens à trouver de nouvelles manières de voir et d'analyser le monde. Il s'agit de découvrir de nouveaux ordres de grandeur concernant la capture, la recherche, le partage, le stockage, l'analyse et la présentation des données. Ainsi est né le « Big Data ». Il s'agit d'un concept permettant de stocker un nombre indicible d'informations et de les gérer par de nouvelles technologies qui dépasseraient les capacités de celles destinées à gérer des données ordinaires.

Le Big Data ne fait pas seulement référence à des données mais aussi à leur analyse et leur utilisation. On essaye de trouver des modèles ainsi que ce qui les relie pour les placer dans un contexte réel. Le défi n'est pas seulement représenté par le grand volume de données mais aussi par la rapidité des traitements et la diversité des informations. Le flux est continu au sein de données non structurées.

Pour soutenir ces efforts et relever ces défis, une révolution est en cours dans la gestion des données autour de la façon dont les données sont stockées, traitées, gérées et fournies aux décideurs. La technologie du Big Data permet une évolutivité et une rentabilité des ordres de grandeur supérieurs à ce qui est possible avec une infrastructure de gestion de données traditionnelle. Le libre-service prend le relais des approches soigneusement conçues et exigeantes en main-d'œuvre du passé, où des armées de professionnels de l'informatique ont créé des entrepôts de données et des magasins de données bien ciblés, mais il a fallu un mois pour apporter des modifications.

La grande diversité des sources de données se traduit souvent par des silos d'informations, un ensemble de systèmes de gestion de données non intégrés avec des schémas, des langages de requête et des API hétérogènes. Les systèmes de lacs de données (Data Lake) ont été proposés comme solution à ce problème, en fournissant un référentiel sans schéma pour les données brutes

avec une interface d'accès commune. Le lac de données est une nouvelle approche audacieuse qui exploite la puissance de la technologie du Big Data et l'associe à l'agilité du libre-service.

Avec le data lake, les utilisateurs peuvent aujourd'hui rapidement intégrer des données dans un pool unique, de sorte qu'elles sont immédiatement disponibles à la fois pour les opérations et pour les analytiques. C'est ainsi que le data lake, en particulier lorsqu'il est déployé au-dessus de Hadoop, répond à la pression des entreprises pour disposer d'outils de traitement de la donnée qui leur apportent un avantage commercial et de prise de décision via la découverte et les analyses.

Cependant, le simple fait de déverser toutes les données dans un lac de données sans aucune gestion des métadonnées ne conduirait qu'à un « marais de données ». Le rôle des métadonnées est principalement de dévoiler le contenu d'un lac de données et de servir en outre différents objectifs, tels que l'exploration de données, l'analyse et la gouvernance, etc. Le catalogue qui organise les métadonnées est le premier point d'accès pour la plupart des utilisateurs. En particulier, pour accéder aux données sans schéma, les lacs de données doivent s'appuyer sur des métadonnées pour leur découverte et interrogation. La gestion de métadonnées est nécessaire pour garantir un accès efficace aux données. Néanmoins, elle pose de nombreux défis au sein des lacs de données en raison des caractéristiques du Big Data, connues sous le nom de Big Data V, conduisant au problème des Big Metadata. Il existe de nombreux travaux ayant proposé des systèmes de gestion des métadonnées. Cependant, les modèles sur lesquels il s'appuient manquent de mécanismes d'extensibilité au niveau de la conception des métadonnées.

Notre travail consiste à développer un système basé sur les graphes qui permet de gérer les métadonnées dans un data lake composé initialement de sources NoSQL (graphes, documents, et relationnelles). Le système est également ouvert et extensible pour prendre en compte d'autres types de sources à travers leur abstraction.

Ce mémoire est structuré en quatre (04) chapitres, suivis d'une conclusion générale qui présente une synthèse des travaux réalisés.

- Dans le chapitre 1 « Concepts de bases », nous développons les principaux concepts concernant les domaines de notre travail : Big Data, Bases de données NoSQL graphes,

Base de données NoSQL documents et Data lakes, à travers leurs définitions, leurs processus de fonctionnement, et leurs domaines d'application.

- Dans le chapitre 2, nous présentons en premier lieu une définition de métadonnées, puis leur rôle, leur type, la gestion de métadonnées et enfin les métadonnées pour les Data Lakes.

- Le chapitre 3 est consacré à l'analyse et la conception du système, où nous montrons l'application du processus 2TUP allégée (2-Track Unified Process). Nous y abordons l'étape de capture des besoins fonctionnels, l'étape d'analyse qui permet d'approfondir la capture des besoins fonctionnels, et la conception du système où nous regroupons les deux derniers types de conception de processus 2TUP.

- Enfin dans le chapitre 4, nous montrons la faisabilité de notre système à travers sa mise en œuvre par une application informatique. Nous présentons d'abord les outils utilisés pour la réaliser, les jeux de données et enfin l'application sous forme de prises d'écran avec des descriptions. Ce chapitre représente la dernière partie de ce mémoire.

Chapitre I

1 Concepts de base

1.1 Introduction

Nous vivons actuellement dans ce que beaucoup appellent l'ère de l'information, un moment où la quantité d'informations a augmenté plus rapidement que tout autre moment dans l'histoire humaine. Cette croissance rapide de la quantité d'informations disponibles a soulevé un nouveau défi pour les chercheurs ce qu'il les obligé à trouver de nouvelles manières de voir et d'analyser le monde. Il s'agit de découvrir de nouveaux ordres de grandeur concernant la capture, la recherche, le partage, le stockage, l'analyse et la présentation des données.

Dans ce chapitre, nous allons présenter les notions de base sur les Big Data, NoSQL, Base de données orienté graphe, Base données orienté document et Data lake.

1.2 Big Data

1.2.1 Histoire du Big Data

Selon Press, le terme « Big Data » est utilisé pour la première fois en 1997 par des scientifiques de la NASA en décrivant le problème auquel ils font face au niveau de la visualisation d'un très grand nombre de données sur leurs systèmes informatiques. Ils en concluent qu'ils ont un « problème de Big Data » quand les supports informatiques habituels ne suffisent plus pour contenir la quantité de données présente. [11]

1.2.2 Définition de Big Data

Littéralement, selon le journal officiel de la république française le terme « Big Data » signifie mégadonnées [1], (grosses données ou encore données massives).

Selon Beyer: « BigData est une information en haut volume, avec une haute vitesse de génération et une grande variété et qui requiert une nouvelles formes de traitement pour permettre une prise de décision améliorée, la découverte de la perspicacité et l'optimisation des processus ». [3]

En 2011, le McKinsey Global Institute propose la définition suivante : « Le Big Data se réfère à un ensemble de données dont la taille va au-delà de la capacité des logiciels de bases de données classiques à capturer, stocker, gérer et analyser ». [12] (traduction)

Selon les chercheurs de McKinsey, cette définition est intentionnellement subjective, dans le sens où tout est relatif. Ils définissent donc le Big Data comme ce qui sera toujours au-delà de la puissance de nos technologies.

En 2013, le terme « Big Data » entre officiellement dans l'Oxford English Dictionary. Il est défini comme « des données d'une très grande taille, dans la mesure où leur manipulation et leur gestion entrainent d'importants challenges logistiques » (traduction). [2]

1.2.3 Caractéristique du Big Data

IL existe trois composantes principales définissant une série de données comme étant du Big Data, dans la littérature on les appelle les « 3 V's » (voir la figure 1.1) : le Volume, la Vitesse et

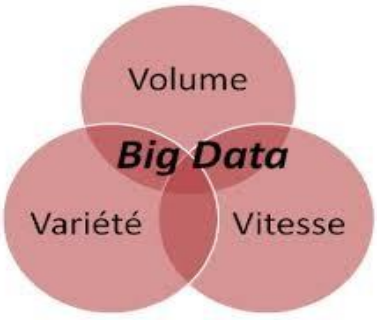


Figure 1.1 3V du Big Data

la Variété.

- **Volume :** Le nom Big Data lui-même contient le terme "énorme". La taille des données joue un rôle très important dans la détermination de la valeur des données. Le volume indique une quantité de données si grande que les outils informatiques standards n'ont pas les capacités nécessaires à leur traitement.

  Cependant, il ne suffit pas d'avoir une grande quantité d'information pour se trouver en face d'une base de donnée qui puisse s'appeler « Big Data », ce qui est vraiment important c'est d'avoir une collection de données dont on puisse extraire la majeure quantité d'information pour la prise de décision en faisant le moindre effort. En autres termes, en ce qui concerne le volume, on a réellement du Big Data quand on a une grande quantité de donnés exploitables. [4]
- **La variété :** Les données caractéristiques du Big Data proviennent d'une énorme quantité de sources : smartphones, ordinateurs, capteurs, codes de barres, etc. Cette diversité de sources fait qu'on ait une grande variété de données à exploiter, rendant impossible de les

analyser avec des outils habituels [4]. La variété caractérise le Big Data par la coexistence de données hétérogènes provenant de sources disparates.

- **La vélocité (la vitesse)** : la vélocité du traitement dans la mesure où la vitesse de traitement des données peut aller jusqu'au temps réel, grâce à « Internet Of Thing » (internet des objets) et la facilité pour avoir accès à l'internet. Aussi, la vélocité peut faire référence à la vitesse avec laquelle on crée des informations par les machines ou par les êtres humains (ex : dans les réseaux sociaux) [4].

Alors on peut définir le Big Data comme : Un ensemble des données avec un grand potentiel d'exploitabilité en occupant de grand volume provenant d'une variété de sources qui leur génèrent à une grande vitesse, rendant impossible leur analyse avec des outils d'analyse traditionnelle, et dont on est sûr de leur intégralité et origine. [4]

Certains auteurs définissent le concept de Big Data avec des caractéristiques supplémentaires telles que la valeur et la validité.

- **La valeur :** le caractère complémentaire « valeur » fait référence à la potentialité des données, en particulier en termes économiques. Il est ainsi associé à l'usage qui peut être fait de ces mégadonnées, de leur analyse, notamment d'un point de vue économique. [13]
- **La validité :** véracité ou validité fait référence à la qualité des données et/ou aux problèmes de valeurs aberrantes ou manquantes (ces problèmes pouvant être résolus par le volume de données), mais aussi à la confiance que l'on peut avoir dans les données. S'il existe des critères permettant de qualifier la qualité des données, dans le cas de Big Data, cette vérification de la qualité est rendue difficile voire impossible du fait du volume, de la variété et de la vélocité spécifique au Big Data. [13]

1.2.4 Classification du Big Data

On peut classer les informations en trois énormes types de donnés : Données structurées, semi-structurées et non-structurées.

- **Information structuré :** se trouvent, par exemple, dans les bases de données ou encore dans les langages informatiques. Ainsi, on les reconnait au fait qu'elles sont disposées de façon à être traitées automatiquement et efficacement par un logiciel, mais non nécessairement par un humain. D'après Alain Garnier, l'auteur du livre l'information non

structurée dans l'entreprise, « une information est structurée lorsqu'elle est répétable, systématique et calculable ». [5]

- **Information non structurée :** par opposition à la catégorie précédente, les informations non structurées représentent l'ensemble des informations pour lesquelles il est impossible de retrouver une structure prédéfinie. Elles sont toujours destinées à des humains et il s'agit donc essentiellement de documents textes et multimédias, comme des lettres, des livres, des rapports, des collections d'images ou de vidéos, des brevets, des images satellites, des offres de services, des CV, des appels d'offres… et la liste est encore longue. [5]
- **Information semi-structurée :** il est à noter que la frontière entre informations structurées et informations non structurées demeures assez floue et qu'il n'est pas toujours aisé de classer un document dans l'une ou l'autre des catégories. Dans ce cas précis, vous avez sans doute affaire à de l'information semi-structurée. [5]

1.2.5 Technologies de Big Data

La technologie Big Data peut être définie comme un logiciel-utilitaire conçu pour analyser, traiter et extraire les informations d'un ensemble de données extrêmement complexe et volumineux que le logiciel de traitement de données traditionnel ne pourrait jamais gérer. Nous exposons ici les principales technologies qui ont soutenu et soutiennent encore l'industrie du Big Data [30] :

a. **La plateforme Apache Hadoop :** Un des outils les plus importants quand on parle de Big Data est la plateforme Apache Hadoop, lancée en décembre 2011. C'est un système open source, écrit en Java (langage de programmation informatique) libre, qui permet le stockage et le traitement de très gros volumes de données, cela au travers d'un groupement d'ordinateurs utilisant des algorithmes simples. Pour illustrer à quel point cette plateforme est devenue incontournable pour les leaders de l'industrie du Big Data, il faut savoir que Hadoop est utilisé, pour la recherche et la production de données, par des entreprises comme Amazon, Facebook, Google, Twitter et Yahoo ! entre autre [7]. (Voir

la figure 1.2 qui représente les composants de corps du Hadoop).

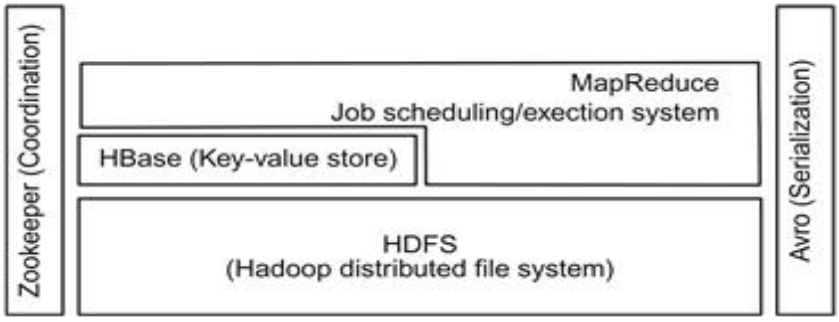


Figure 1.2 composants de corps du Hadoop

b. **Les bases de données non-relationnelles NOSQL :** Les bases de données relationnelles organisent et présentent les données selon un système de relations (tableaux à deux dimensions) et utilisent le langage de programmation informatique SQL (Structured Query Langage). Au contraire, les bases de données non-relationnelles, rebaptisées « NoSQL » (Not Only SQL) en 2009, sont des outils qui ne sont plus fondés sur l'architecture classique des bases relationnelles et qui permettent de faire face à deux des grands challenges du Big Data : son volume et sa variété. La plupart de ces bases de données sont incorporées dans des sites web comme Google, Yahoo!, Amazon et Facebook.

c. Le cloud computing : Littéralement « informatique dans le nuage », cette technologie permet des capacités importantes de stockage de données mais aussi de ressources informatiques, disponibles en ligne, à des prix fixes et avec de hauts niveaux de flexibilité et de fiabilité. Grâce au cloud computing, les entreprises actives dans l'industrie du Big Data disposent d'une quantité infinie de ressources informatiques de qualité.

d. **Les systèmes de data mining :** Ces systèmes d'exploration de données constituent un ensemble de méthodes, d'algorithmes et de techniques qui permettent de manipuler le Big Data en apportant des solutions efficaces face aux challenges du volume, de la variété et de la vitesse des données. PageRank[1] et MapReduce[2] sont des modèles informatiques de ce type qui sont utilisés par Google et qui sont les symboles d'une avance importante dans le domaine de la manipulation et de l'analyse des grands volumes de données.

[1] Le PageRank mesure la qualité et la quantité des liens reçus par une page sur un site Internet, afin d'affiner son classement.

[2] MapReduce est un modèle de programmation popularisé par Google. Il est principalement utilisé pour la manipulation et le traitement d'un nombre important de données au sein d'un cluster de nœuds.

L’exposé de ces quelques outils et techniques du Big Data nous montre la complexité informatique et technologique qui se cache derrière les géants de l’industrie des données. A présent, nous allons parcourir quelques applications concrètes du Big Data dans le monde de l’entreprise et dans notre vie quotidienne.

1.2.6 Applications du Big Data

Voici quelques exemples d'applications Big Data :

a. **E-santé :** l'analyse avancée des ensembles de données médicales a de nombreuses applications bénéfiques. Il permet de personnaliser les services de santé (par exemple, les médecins peuvent surveiller les symptômes des patients en ligne afin d'ajuster la prescription) ; adapter les plans de santé publique en fonction des symptômes de la population, de l'évolution de la maladie et d'autres paramètres ; il est également utile d'optimiser les opérations hospitalières et de diminuer les dépenses de santé. [6]

b. **Internet des objets (IoT) :** représente l'un des principaux marchés des applications Big Data. En raison de la grande variété d'objets, les applications de l'IoT sont en constante évolution. De nos jours, il existe diverses applications Big Data pour les entreprises logistiques. En fait, il est possible de suivre la position des véhicules avec des capteurs, des adaptateurs sans fil et un GPS. Ainsi, ces applications basées sur les données permettent aux entreprises non seulement de superviser et de gérer les employés, mais aussi d'optimiser les itinéraires de livraison. C'est en exploitant et en combinant diverses informations, y compris l'expérience de conduite passée. La ville intelligente est également un domaine de recherche à chaud basé sur l'application de données IoT. [6]

c. **Services publics :** les services publics tels que les organisations d'approvisionnement en eau placent des capteurs dans les canalisations pour surveiller le débit d'eau dans les réseaux complexes d'approvisionnement en eau. Il est rapporté dans la presse que le Bangalore Water Supply and Sewage Board met en œuvre un système de surveillance en temps réel pour détecter les fuites, les connexions illégales et contrôler à distance les vannes pour assurer un approvisionnement équitable en eau dans différentes zones de la ville. Cela permet de réduire le besoin d'opérateurs de vannes et d'identifier et de réparer en temps opportun les conduites d'eau qui fuient. [6]

1.3 Base de donnée NOSQL

Depuis les années 70, la base de données relationnelle était l'incontournable référence pour gérer les données d'un système d'information. Toutefois, face aux 3V (*Volume, Velocity, Variety*), le relationnel peut difficilement lutter contre cette vague de données. Le NoSQL s'est naturellement imposé dans ce contexte en proposant une nouvelle façon de gérer les données, sans reposer sur le paradigme relationnel, d'où le "**Not Only SQL**". Cette approche propose de relâcher certaines contraintes lourdes du relationnel pour favoriser la distribution (structure des données, langage d'interrogation ou la cohérence).

Dans un contexte bases de données, il est préférable d'avoir un langage de haut niveau pour interroger les données plutôt que tout exprimer en Map/Reduce. Toutefois, avoir un langage de trop haut niveau comme SQL ne facilite pas la manipulation. Et c'est en ce sens que l'on peut parler de "Not Only SQL", d'autres solutions peuvent être proposées pour résoudre le problème de distribution. Ainsi, le NoSQL est à la fois une autre manière d'interroger les données, mais aussi de les stocker.

- Types de base de données NoSql

Les besoins de stockage et de manipulation dans le cadre d'une base de données sont variables et dépendent principalement de l'application que vous souhaitez intégrer. Pour cela, différentes familles de bases NoSQL existent : *Clé/Valeur, colonnes, documents, graphes*. Chacune de ces familles répond à des besoins très spécifiques.

1. Base orientée Clé/ valeur

Le but de la famille clé-valeur est l'efficacité et la simplicité. Un système clé-valeur agit comme une énorme table de hachage distribuée sur le réseau. Tout repose sur le couple Clé/Valeur (voir la figure 1.3). La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données.

Le fait d'avoir n'importe quoi implique qu'il n'y ait ni schéma, ni structure pour le stockage. D'un point de vue de bases de données, il n'y a pas la possibilité d'exploiter ni de contrôler la structure des données et de fait, pas de langage (SQL = Structured Qyery Langage).

Les seules opérations de type CRUD peuvent être utilisées :

- Create (key, value)
- Read (key)
- Update (key, value)
- Delete (key)

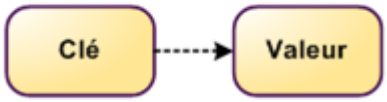


Figure 1.3 Base de données orientée clés/valeur [14]

2. Base orientée Colonnes

Traditionnellement, les données sont représentées en ligne, représentant l'ensemble des attributs. Le stockage orienté colonne change ce paradigme en se focalisant sur chaque attribut et en les distribuant (voir la figure 1.4). Il est alors possible de focaliser les requêtes sur une ou plusieurs colonnes, sans avoir à traiter les informations inutiles (les autres colonnes).

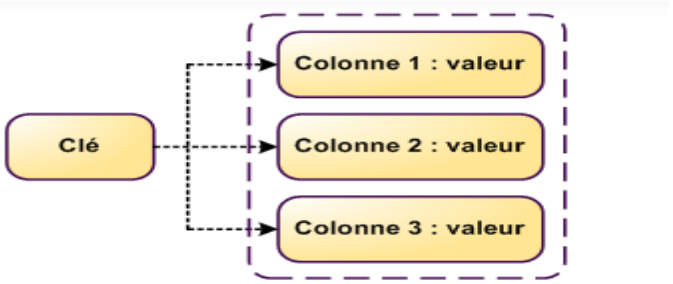


Figure 1.4 Base de données orientée colonnes [14]

3. Base orientée Documents

Les bases orientées documents ressemblent sans doute le plus à ce que l'on peut faire dans une base de données classique pour des requêtes complexes. Le but de ce stockage est de manipuler des documents contenant des informations avec une structure complexe (types, listes, imbrications). Il repose sur le principe du clé/valeur, mais avec une extension sur les champs qui composent de document (voir la figure 1.5).

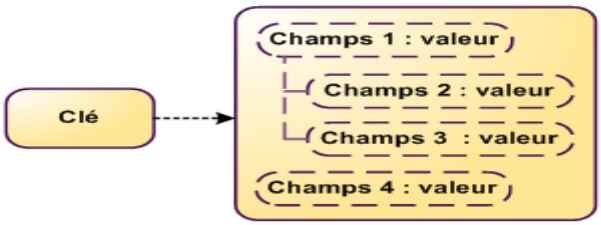


Figure 1.5 Base de données orientée document [14]

4. Base orientée Graphes

Dans la base orientée graphe, les données stockées sont : les nœuds, les liens et des propriétés sur ces nœuds et ces liens (voir la figure 1.6). Les requêtes que l'on peut exprimer sont basées sur la gestion de chemins, de propagations, d'agrégations, voire de recommandations. Toutefois, contrairement aux solutions précédentes la distribution des nœuds sur le réseau n'est pas triviale.

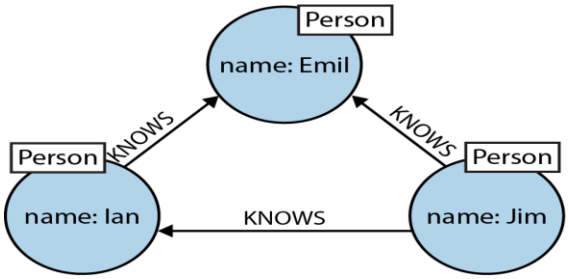


Figure 1.6 Base de données orientée graphes [15]

1.4 Emergence des bases de données orientées graphes

Les réseaux sociaux et l'émergence de Facebook, LinkedIn et Twitter ont accéléré l'émergence de la base de données NoSQL la plus complexe, la base de données **graphes**. Les trois autres familles NoSQL (*Clé/Valeur, colonnes, documents)* n'adressent pas le problème de corrélations entre les éléments. Prenons l'exemple d'un réseau social : dans certains cas, il devient très complexe de calculer la distance entre deux personnes non directement connectées. Et c'est ce type d'approche que résolvent les bases orientées Graphes. La base de données graphes est orientée vers la modélisation et le déploiement de données graphiques par construction. [7]

1.4.1 Définition de base de données graphes

Un système de base de données graphes (désormais, une base de données graphes) est un système de gestion de base de données en ligne avec méthodes CRUD (Create, Read, Update, and Delete) qui exposent un modèle de données graphes. Les bases de données Graphes sont généralement conçus en tenant compte de l'intégrité transactionnelle et de la disponibilité opérationnelle. [15]

1.4.2 Les propriétés des bases de données graphes

Il y a deux propriétés des bases de données graphes à prendre en compte lors de l'étude des technologies de bases de données graphes :

1.4.2.1 Le stockage sous-jacent

Certaines bases de données graphes utilisent un stockage de graphes natif optimisé et conçu pour stocker et gérer des graphes. Cependant, toutes les technologies de base de données graphes n'utilisent pas le stockage de graphes natif. Certains sérialisent les données du graphe dans une

base de données relationnelle, une base de données orientée objet ou un autre magasin de données à usage général. [15]

1.4.2.2 Le moteur de traitement

Certaines définitions exigent qu'une base de données graphes utilise une contiguïté sans index, ce qui signifie que les nœuds connectés se « pointent » physiquement les uns vers les autres dans la base de données. [15]

1.4.3 Représentation de Base de données graphes

Une base de données graphes représente chaque objet comme un nœud et les relations comme une arête.

Comme le modèle ER classique pour SGBDR, nous devons créer un modèle d'attribut pour une base de données graphes. Nous pouvons commencer par prendre le niveau le plus élevé dans une hiérarchie en tant que nœud racine (similaire à une entité) et connecter chaque attribut comme son sous-nœud. Pour représenter différents niveaux de la hiérarchie, nous pouvons ajouter une sous-catégorie ou une sous-référence et créer une autre liste d'attributs à ce niveau. Cela crée un modèle de parcours naturel comme un parcours d'arbre, qui est similaire au parcours d'un graphe. Selon la propriété cyclique du graphique, nous pouvons avoir un modèle équilibré ou asymétrique [7].

1.4.4 Type de base de données graphes [10]

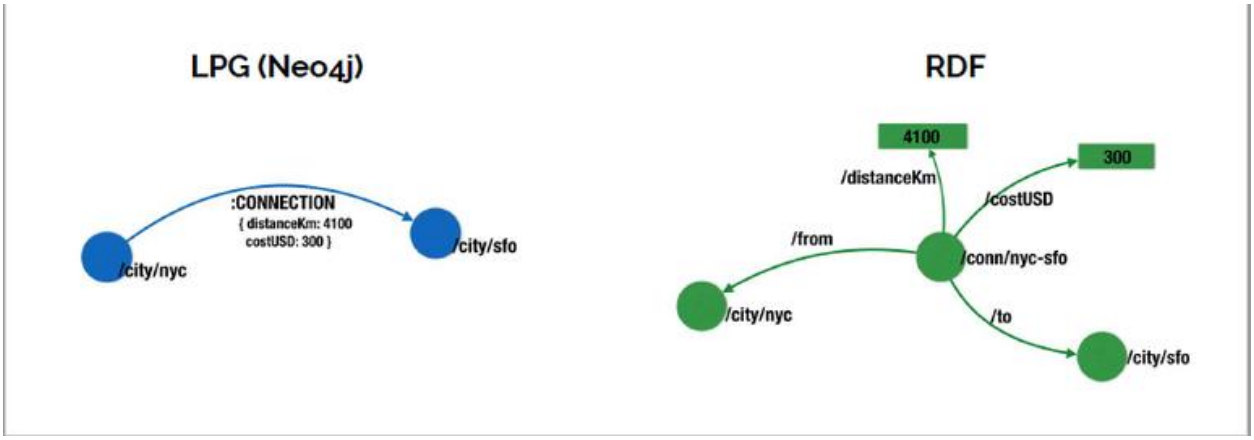


Figure 1.7 Représentation du modèle LPG et RDF

Les réseaux peuvent être représentés par des graphes très différents suivant la quantité d'informations que l'on veut conserver et l'utilisation de ces informations. Il existe donc deux

grands types de bases de données orientées graphe : le « Resource Description Framework » (RDF) et le « Labelled Property Graph » (LPG).

Le RDF reprend l'information de façon complète mais volumineuse. L'autre représentation des graphes qu'est le LPG qui est plus synthétique et donc moins volumineux.

Le LPG. Un « Property graph » est un graphe où les arcs et les nœuds peuvent chacun avoir une étiquette ainsi qu'un attribut spécifique. Il est possible d'avoir plusieurs étiquettes et attributs dans le cas de graphes qualifiés de « Multi-Valued Property ». Dans un graphe LPG :

-les nœuds sont interprétés comme des entités. Chaque nœud possède une clé-valeur, c'est à dire un identifiant unique à cette entité. Les nœuds ont aussi une ou plusieurs valeurs qui sont les attributs du nœud. Les nœuds peuvent être regroupés sur des couches et assimilés à un type. Dans le cas présenté ci-dessus (Figure 1.7), il s'agit du type « city ». Une fois les entités identifiées, les arcs représentent une relation entre deux entités.

-Les arcs possèdent une certaine direction, allant d'une entité à l'autre. Elles sont aussi définies par un identifiant unique et peuvent avoir un ou plusieurs attributs, quantitatifs ou qualitatifs.

Il est aussi possible de mettre en avant une structure en triplet dans un graphe LPG. En effet, la structure est constituée de deux entités composées d'attributs, un nœud de départ et un nœud d'arrivée. Le nœud de départ est relié au nœud d'arrivée par un arc également constitué d'attributs. Ce type de graphe possède une structure flexible, il est aisé d'ajouter ou de supprimer des entités ou des arcs. Il stocke la même information qu'un graphe RDF mais de façon plus synthétique, ce qui permet d'effectuer des recherches de nœuds et d'arcs plus rapidement. On appelle parcours transversal du graphe la capacité de retrouver le nœud voulu en minimisant le nombre d'entités et d'arc consultés.

1.4.5 Langages de requête

Il existait un nombre important de langages de requête pour interagir avec les bases de données orientées graphes. Voici quelques exemples des langages de requête de base de données orientée graphes les plus populaires :

- Cypher : c'est un langage de requête déclaratif très populaire qui a été inventé par Neo4j. Sa popularité provient de sa ressemblance avec SQL. Les personnes ayant une expérience du SQL se sentiront comme à la maison. Neo4j participe à présent aux activités de mise en place du nouveau standard GQL ;

- SPARQL : c'est un langage de requête déclaratif de type SQL créé par le W3C pour interagir avec les bases de données orientées graphes ;
- GraphQL : créé par Facebook, c'est un langage de requête pour les API qui n'est pas spécifique aux bases de données graphes. Les utilisateurs définissent la structure des données dont ils ont besoin et obtiennent exactement ce qu'ils ont demandé. Les requêtes GraphQL sont organisées en types et en champs, et non en points de terminaison. En modifiant l'objet de requête, vous pouvez déterminer ce qui est renvoyé par le serveur ;
- Gremlin : il a été créé en 2009 et est le langage de requête pour Apache TinkerPop. C'est un DSL (Domain Specific Language) de parcours de graphe qui peut être déclaratif ou impératif. Il peut être utilisé pour les bases de données orientées graphes OLTP et OLAP. Gremlin est basé sur Groovy, mais possède de nombreuses variantes permettant aux développeurs d'écrire des requêtes Gremlin de manière native dans de nombreux langages de programmation modernes tels que Java, JavaScript, Python, Scala, Clojure et Groovy.

1.4.6 Les avantage et les inconvénients de base de données graphes

Les bases de données graphes fournissent moins de schéma et un stockage efficace des données semi-structurées. Les requêtes sont exprimées sous forme de traversées, rendant ainsi les bases de données graphes plus rapides que les bases de données relationnelles. Il est facile à mettre à l'échelle et compatible avec les tableaux blancs. Les bases de données Graphes sont compatibles ACID et offrent une prise en charge de la restauration. Mais il est très difficile de réaliser le «sharding[3]» dans les bases de données Graphes. Les bases de données graphes sont difficiles à regrouper. [8]

1.4.7 Domaines d'applications des bases de données graphes

Les bases de données Graphes peuvent être utilisées pour une variété d'applications telles que les applications de réseautage social, les logiciels de recommandation, la bio-informatique, la gestion de contenu, la sécurité et le contrôle d'accès, la gestion du réseau et du cloud, etc… [8]

[3] Le Sharding : partitionnement des données

1.4.8 Bases de données orienté graphe les plus populaires

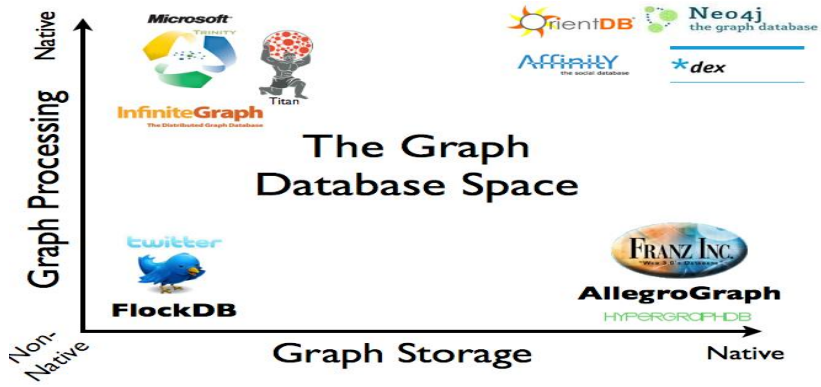


Figure 1.8 Base de données Graphes [15]

Neo4J : est une base de données graphes native open source NoSQL qui fournit un backend transactionnel compatible ACID pour vos applications. Le développement initial a commencé en 2003, mais il est accessible au public depuis 2007. Le code source, écrit en Java et Scala. [16]

infiniteGraph : est une base de données graphes distribuée d'entreprise implémentée en Java, et provient d'une classe de technologies de base de données NOSQL (ou Not Only SQL) qui se concentrent sur les structures de données de graphe. Les développeurs utilisent Infinitegraph pour trouver des relations utiles et souvent cachées dans des ensembles de Big Data hautement connectés. InfiniteGraph est multiplateforme, évolutif, compatible avec le cloud et est conçu pour gérer un débit très élevé. L'adoption est observée dans les applications gouvernementales, télécoms / réseaux, soins de santé, cyber sécurité, fabrication, finance, CRM et réseaux sociaux. [17]

GraphDB : est un système de gestion de base de données NoSQL open source écrit en Java. Il s'agit d'une base de données multi modèle, prenant en charge les modèles de graphes, de documents, de clés / valeurs et d'objets, mais les relations sont gérées comme dans les bases de données graphes avec des connexions directes entre les enregistrements. [17]

AllegroGraph : est un triple store à code source fermé conçu pour stocker des triplets RDF, un format standard pour les données liées. AllegroGraph est actuellement utilisé dans les projets Open source, les projets commerciaux et les projets du ministère de la Défense. C'est également le composant de stockage du projet TwitLogic qui amène le Web sémantique aux données Twitter.[17]

1.5 Base de données orientée document

1.5.1 Qu'est-ce qu'une base de données orientée Document ?

Une base de données de documents est un type de base de données non relationnelle conçu pour stocker et interroger des données sous forme de documents de type XML, PDF, JSON. Elles sont quelque peu similaires aux enregistrements des bases de données relationnelles, mais sont beaucoup plus flexibles car elles sont moins schématiques. Les bases de données de documents permettent aux développeurs de stocker et d'interroger une base de données en utilisant le même format de modèle de document que celui qu'ils utilisent dans leur code d'application. La nature souple, semi-structurée et hiérarchique des documents et des bases de données de documents leur permet d'évoluer avec les besoins des applications. [8]

1.5.2 Représentation de base de données orientée documents

Les bases de données orientées documents créent une paire simple : une **clé unique (**Ces clés peuvent être une simple chaîne ou une chaîne faisant référence à l'URI ou au chemin) est affectée à un document spécifique. Ce document, qui peut par exemple être formaté avec XML, JSON contient les informations à proprement parler. Étant donné que la base de données n'exige pas de schéma déterminé (les documents peuvent avoir une structure complètement différente les uns des autres.), il est également possible d'intégrer différents types de documents dans un même document store, Un "document" a une structure arborescente qui contient une liste de champs qui ont une valeur et qui peuvent à leur tour contenir une structure avec une liste de champs et ainsi de suite. [8]

1.5.3 Caractéristiques de conception des bases de données orientées document

Parmi de nombreuses caractéristiques on peut citer : [7]

- Absence de schéma : il n'y a aucune restriction sur la structure et le format de la façon dont les données doivent être stockées. Cette flexibilité permet à un système évolutif d'ajouter plus de données et permet aux données existantes d'être conservées dans la structure actuelle.
- Magasin de documents : les objets peuvent être sérialisés et stockés dans un document, et il n'y a pas d'intégrité relationnelle à appliquer et à suivre.

- Facilité de création et de maintenance : une simple création du document permet de créer des objets complexes une seule fois et la maintenance est minimale une fois le document créé.
- Aucune application de relation : les documents sont indépendants les uns des autres et il n'y a pas de relation de clé étrangère à craindre lors de l'exécution de requêtes. Les effets de la concurrence et des problèmes de performances liés à la même chose ne sont pas un problème ici.
- Formats ouverts : les documents sont décrits en utilisant JSON, XML ou un dérivé, ce qui rend le processus standard et propre dès le départ.
- Gestion des versions intégrée : les documents peuvent devenir volumineux et compliqués avec les versions. Pour éviter les conflits et maintenir l'efficacité du traitement, la gestion des versions est implémentée par la plupart des solutions disponibles aujourd'hui.

1.5.4 Les avantages de base de donnée orientées documents

Les avantages de ce modèle comprennent : [7]

- Possibilité de stocker des données dynamiques dans des formats non structurés, semi-structurés ou structurés.
- Possibilité de créer des vues persistantes à partir d'un document de base et de les stocker pour analyse.
- Capacité de stocker et de traiter de grands ensembles de données.

1.5.5 Les base de données orienté document les plus populaires

MongoDB : a été développé par 10gen et a été initialement publié en 2009. Elle a été développé en C ++. C'est une base de données performante et efficace. Elle fournit des fonctionnalités telles que la tolérance aux pannes de cohérence, la persistance. MongoDB fournit des fonctionnalités supplémentaires comme l'agrégation, les requêtes ad hoc, l'indexation, le partage automatique, etc. Dans MongoDB, les documents sont principalement stockés au format BSON (Binary JSON). MongoDB utilise GridFS comme spécification pour stocker des fichiers volumineux. MongoDB est bien adapté aux applications telles que les systèmes de gestion de contenu, l'archivage, l'analyse en temps réel, etc. MongoDB est actuellement utilisé par les réseaux MTV, Foursquare, The Guardian etc. Il est également utilisé dans des projets comme le LHC du CERN, UIDAI Aadhaar qui est Projet d'identification unique de l'Inde. Les inconvénients sont qu'il peut ne pas être fiable et que l'indexation prend beaucoup de RAM. [8]

CouchDB : a été développé par Apache Software Foundation et a été initialement publié en 2005. Il a été développé en C ++. Il utilise des documents JSON pour stocker des données et fournit une API HTTP RESTful pour créer et mettre à jour des documents de base de données. Il fournit JavaScript comme langage de requête. Il fournit une application Web intégrée appelée FULTON qui peut être utilisée pour l'administration. Il est hautement disponible, tolérant aux pannes et persistant. Il implémente le contrôle d'accès concurrentiel multi-version (MVCC) offrant ainsi un accès simultané aux utilisateurs. CouchDB a de grandes capacités de réplication et de synchronisation. Certains des inconvénients de CouchDB sont que les vues temporaires dans CouchDB sur de grands ensembles de données sont vraiment lentes, pas bonnes pour traiter les données relationnelles, pas de support pour les requêtes ad hoc. [8]

1.6 Lacs de données (Data Lakes)

Aujourd'hui l'avènement du Big Data et ses évolutions constantes créent de plus en plus de besoins en technologies performantes d'analyse des données. Le Data Lake (ou lac de données) fait son apparition pour répondre à ces besoins.

1.6.1 Définitions

Un Data Lake peut être conçu comme un référentiel de stockage qui contient une grande quantité de données dans son format natif (les données peuvent être des données structurée, semi-structurée et non structurée), jusqu'à ce qu'elles soient nécessaires. Au sein des Data Lakes, des données hétérogènes sont stockées et consultées ; cependant, le traitement de ces données est fastidieux, long et inefficace, en raison des différents modèles de données et formats de fichiers. De plus, l'accès à ces données est entravé par la variété des interfaces, des services et des applications. [18] Les lacs de données sont principalement associés à hadoop, ce qui permet de stocker des données avec réplication à faible cout et de traiter les données avec MapReduce. [19]

1.6.2 A quoi sert un Data Lake ?

Grâce au data lake, l'utilisateur va pouvoir matérialiser son besoin, extraire les différentes données liées à ce besoin et les combiner à sa guise pour en faire sens. La grande évolution du data lake est qu'il rend le traitement des données plus opérationnel, car il est capable de réagir aux données en temps réel. La flexibilité rendue possible par le lac de données ainsi que la dimension opérationnelle vont permettre aux entreprises de se concentrer sur leur proposition de valeur et les solutions innovantes qu'elles peuvent mettre en place, leur cycle d'innovation étant ainsi optimisé et accéléré. [28]

Le data lake est une structure permettant de traiter de très grandes volumétries de données hétérogènes même si on n'en connaît pas encore l'usage. La donnée brute ingérée reste vierge et permet ainsi d'ouvrir le champ des possibilités quant à son analyse. De plus, grâce au data lake, il est possible de coupler la donnée interne de l'entreprise avec des données externes telles que la météo, la pollution, le trafic, etc… pour en faire un outil puissant de prédiction des comportements.

1.6.3 Lacs de données et métadonnées

Les lacs de données sont généralement construits pour intégrer de très grands volumes de données non structurées de manière rapide. Ils ne se limitent toutefois pas à une technologie de stockage (la plupart du temps HDFS – Hadoop Distributed File System), mais proposent un nouvel écosystème de données permettant rapidement, à la demande, de croiser des données, sans besoin de prétraitements coûteux tels que la construction d'un entrepôt de données. Les données sont immédiatement accessibles, contrairement, de nouveau, aux entrepôts de données qui sont rafraîchis périodiquement via une phase d'ETL (extraction, transformation, chargement) qui peut être coûteuse [23].

Les lacs de données ont une architecture « plate », où chaque élément de données possède un identifiant unique et un ensemble d'étiquettes (tags) le caractérisant [23]. Par conséquent, ils ne nécessitent pas de schéma rigide comme les entrepôts de données. De plus, les étiquettes caractérisant les données sont des métadonnées indispensables à la compréhension des données et à leur accès, sans même parler de l'efficacité des requêtes. L'avantage de l'étiquetage des données est que de nouvelles données, de nouvelles sources, peuvent être insérées au fil de l'eau. Une fois les données étiquetées, elles doivent simplement être connectées aux données déjà stockées. Ce dispositif permet aux utilisateurs de formuler des requêtes directement, sans besoin de recourir à l'aide d'un expert en informatique décisionnelle.

1.6.4 Fonctionnalités des lacs de données

1. L'acquisition

Cette fonctionnalité est celle par laquelle les sources de données, internes, externes, structurées, non structurées vont être "ingérées" par le lac de données. On peut faire l'analogie avec le système décisionnel (d'entreprise), et son composant ODS (Operation Data Store).

2. Catalogue de données

Chaque fois que des données provenant de différentes sources, contextes et avec divers schémas sont rassemblées, des métadonnées sont nécessaires pour suivre ces données. Cela s'applique également aux lacs de données, où la gestion des métadonnées est un élément crucial. Les métadonnées capturent des informations sur les données réelles, par exemple des informations de schéma, une sémantique ou une lignée. Ils garantissent que les données peuvent être trouvées, fiables et utilisées. Il existe un grand nombre d'approches différentes pour la gestion des métadonnées. Selon la littérature Data Lake, les catalogues de données sont utilisés pour stocker les métadonnées. [22]

3. Le stockage

Ce composant est celui où les données acquises et cataloguées vont pouvoir être stockées avant, pendant et après (potentiellement) la phase d'exploitation (ou d'exploration) de ces données. C'est à ce composant qu'est souvent réduit un lac de données [26] associé à la technologie Apache Hadoop. Russom [27] emploie d'ailleurs le mot "Hadoop Data lake". Mais le lac de données ne doit pas être réduit à cette fonction (ou composant) mais bien être considéré comme un concept d'architecture.

4. L'exploitation ou l'exploration

Ce composant est l'objectif des lacs de données : permettre l'exploitation et l'exploration des données de l'organisation.

5. La gouvernance

La gouvernance du lac de données a en son cœur la gestion du catalogue des métadonnées mais ce n'est pas le seul élément nécessaire pour gouverner un lac de données. La gestion de la sécurité, du cycle de vie et de sa qualité sont les fondamentaux.

1.6.5 Architecture du lac de données

L'architecture du lac de données décrit comment les données sont organisées conceptuellement dans un lac de données. Il facilite l'utilisation d'un lac de données [20] en définissant où les données peuvent être trouvées dans la condition nécessaire pour une utilisation particulière. Il existe deux variantes pour l'architecture globale des lacs de données, à savoir les architectures de zone et d'étang (pond).

1.6.5.1 Architecture de zone

Diverse alternative existe pour les architectures de zone, ils diffèrent par de multiples aspects, tel que le nombre et les caractéristiques des zones. Bien qu'il n'y ait pas d'architecture de zone communément acceptée, l'idée reste toujours la même :

Les données sont affectées à une zone en fonction du degré de traitement qui leur a été appliqué. Lorsque les données son ingérées dans le lac de données, elles sont stockées au format brut dans la zone brute, qui est commune à toutes les architectures de zone. D'autres zones conditionnent ensuite davantage ces données. Certaines zones normalisent les données pour s'adapter à un format commun, d'autres nettoient les données. Certaines architectures de zone incluent même une zone de type data-mart, où les données sont préparées pour s'adapter à certains cas d'utilisation et outils. L'avantage des architectures de zone est que même si les données sont disponibles dans un format transformé et prétraité, elles sont toujours accessibles en tant que données brutes dans la zone brute. [22]

[25] propose une architecture DL fonctionnelle (voir Figure 1.9), qui contient quatre zones essentielles, et chaque zone, à l'exception de la zone de gouvernance, a une zone de traitement (rectangle en pointillé) et une zone de stockage de données qui stocke le résultat des processus (rectangle gris) :

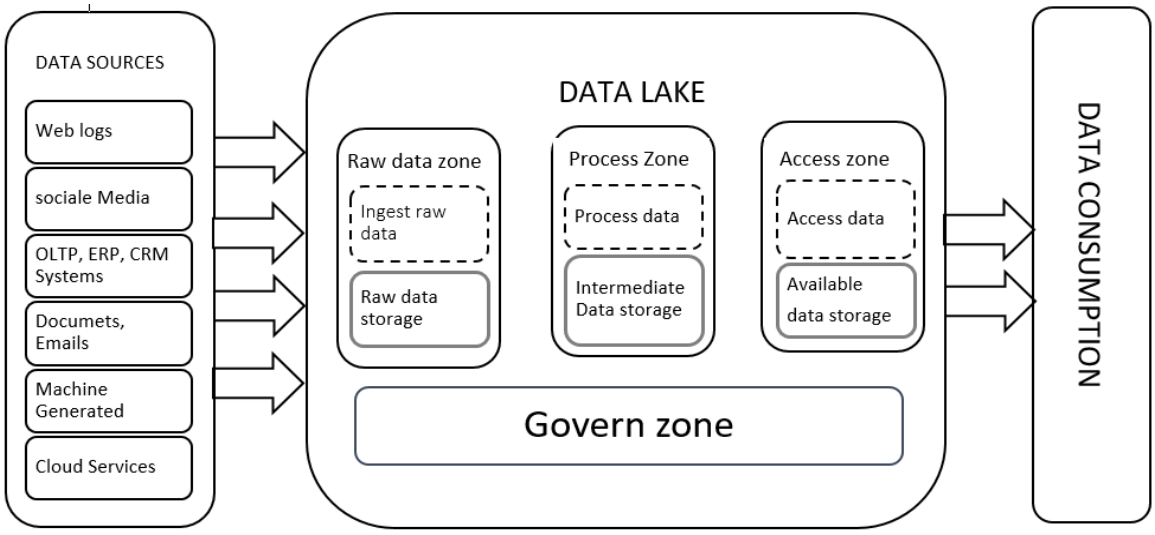


Figure 1.9 architecture de zone du data lake. [25]

1.6.5.2 Architecture des bassins (pond)

L'architecture des bassins est une autre variante de l'architecture globale des lacs de données. Les données de lac de données sont réparties sur cinq étangs différents. Cependant, contrairement aux

architectures de zone, les données ne sont disponibles que dans le bassin à un moment donné. Lors de l'ingestion, les données sont stockées dans le bassin de données brutes. Seuls les données non utilisées et les données qui ne rentrent dans aucun des autres bassins restent dans le bassin de données brutes, toutes les autres données circulent dans le bassin de données analogiques, d'application ou textuelles. Le bassin vers lequel ils se dirigent dépend des caractéristiques des données. Le bassin de données analogiques contient des données de mesure, telles que des fichiers journaux ou des données IoT. Dans le bassin de données d'application, toutes les données générées par les applications sont stockées. Le bassin de données d'application, toutes les données générées par les applications sont stockées. Le bassin de données textuelles contient des données textuelles. D'autres données comme des images et des vidéos, restent dans le bassin de données brutes. Lorsque les données ne sont plus utilisées, ils quittent leur bassin respectif et se dirigent vers le bassin de données d'archives. Au fur et à mesure que les données circulent dans les étangs, ils sont transformés en fonction de l'étang auquel ils appartiennent actuellement. Par exemple, les valeurs aberrantes peuvent être supprimées du bassin de données analogiques et les données textuelles peuvent être structurées.

En plus de ces deux architectures de lac de données générales, la littérature suggère l'architecture lambda [21] pour organiser les données par lots et en continu. Les données entrantes sont copiées dans deux branches différentes. Sur une succursale, les données sont stockées en permanence et périodiquement traitées par lots. Sur l'autre branche, les données entrantes sont traitées en temps réel pour fournir des résultats rapides.

1.6.6 Les éléments essentiels d'une solution data lake et analytiques

Au fur et à mesure que les organisations créent des lacs de données et une plate-forme d'analyse, elles doivent prendre en compte un certain nombre de fonctionnalités clés, notamment :

a. Mouvement de données **:** Les lacs de données vous permettent d'importer n'importe quelle quantité de données pouvant venir en temps réel. Les données sont collectées à partir de plusieurs sources et déplacées dans le lac de données dans leur format d'origine. Ce processus vous permet de mettre à l'échelle des données de toute taille, tout en économisant du temps lors de la définition des structures de données, du schéma et des transformations. [24]

b. Analytique **:** Les lacs de données permettent à divers rôles de votre organisation, tels que les scientifiques des données, les développeurs de données et les analystes commerciaux, d'accéder aux données avec leur choix d'outils et de cadres d'analyse. Cela inclut les Framework open source tels qu'Apache Hadoop, Presto et Apache Spark [24].
c. Stocker et cataloguer les données en toute sécurité : Les Data Lakes vous permettent de stocker des données relationnelles telles que des bases de données opérationnelles et des données issues d'applications métier, et des données non relationnelles telles que des applications mobiles, des appareils IoT et des réseaux sociaux [24].
d. Apprentissage automatique **:** Les Data Lakes permettront aux organisations de générer différents types d'informations, y compris des rapports sur les données historiques, et de faire du machine Learning où des modèles sont construits pour prévoir les résultats probables, et suggérer une gamme d'actions prescrites pour obtenir le résultat optimal. [24]

1.6.8 Les lacs de données vis-à-vis des systèmes décisionnels

Dans le tableau 1.1 nous présentons un positionnement des lacs de données dans les système d'information et vis-à-vis du système décisionnel. En tant que composant du système d'information le lac de données peut donc être étudié selon son architecture via une démarche d'urbanisme.

	Lac de données	Entrepôt de données
Système de stockage	Hadoop, NoSQL, Base de données Relationnelle	Base de données Relationnelle
Qualification de la donnée	Non	Oui
Valeur de la données	Haute	Haute
Granularité de la donnée	Brute	Agrégée
Connaissance de la donnée	Non	Oui
Préparation de la donnée	A la volée	Avant intégration

Intégration	Aucun traitement, Donnée brute	Contrôle de la qualité, filtrage, agrégation
Transformation	Aucune puis ETL	ETL
Schéma	On read	On write
Architecture d'information	Horizontale	Verticale
Modélisation	A la volée	Etoile ou flocon
Métadonnées	Oui	Optionnel
Modèle de conception	Data driven	Information driven
Méthode d'analyse de donnée	Unique	Répétitive
Utilisateur	Informaticien, data scientiste, Analyste, Développeur.	Ligne métier
Fréquence de mis à jour	Temps réel et mode batch	Mode batch
Gouvernance		
Architecture	Centraliser ou fédérée ou hybride	Centraliser

Tableau 1.1: Comparaison lac de données entrepôt de données [29]

1.7 Conclusion

Dans ce chapitre, nous avons illustré le concept du Big Data, ses caractéristiques, ses technologies connexes et ses domaines d'applications ainsi que les outils et les techniques de stockage utilisées. A ce stade, nous pouvons dire que le Big Data est un écosystème large et complexe qui surpassent les capacités des systèmes traditionnels de stockage et de traitement des données. C'est dans ce contexte que le concept de lac de données est introduit, en guise de solution aux problèmes induits par l'hétérogénéité des mégadonnées. Un lac de données propose un stockage intégré des données sans schéma prédéfini.

Pour éviter que le lac de donnée soit inutilisable, un système de métadonnées efficace devient alors essentiel pour rendre les données interrogeables. Et c'est exactement notre sujet de deuxième chapitre « les métadonnées ».

Chapitre II

2. Gestion des Métadonnées

2.1 Introduction

Les métadonnées, sont devenues un terme largement utilisé mais encore souvent sous-spécifié qui est compris de différentes manières par les diverses communautés professionnelles qui conçoivent, créent, décrivent, préservent et utilisent les systèmes et ressources d'information. C'est une construction qui existe depuis aussi longtemps que les humains organisent l'information, bien que de manière transparente dans de nombreux cas, et aujourd'hui nous la créons et interagissons avec elle de manière de plus en plus numérique. Au cours des cent dernières années au moins, la création et la gestion des métadonnées incombent principalement aux professionnels de l'information engagés dans le catalogage, la classification et l'indexation, mais comme les ressources d'information sont de plus en plus mises en ligne par le grand public, les considérations relatives aux métadonnées ne relèvent plus uniquement des professionnels de l'information. [32]

Dans ce chapitre nous allons présenter la définition des métadonnées, la gestion des métadonnées et les métadonnées dans le lac de données.

2.2 Définition de métadonnées

Dans son acception première, le terme signifie « données sur les données ou données qui renseignent sur certaines données et qui permettent ainsi leur utilisation pertinente » (Vocabulaire de la géomatique publié par l'office de la langue française de 1993).

Les métadonnées sont des informations sur le contexte des données, en d'autres termes, des données sur les données [33]. Elles constituent des déclarations émises par un niveau d'abstraction supérieur concernant un niveau inférieur. Les métadonnées sont définies dans [19] comme « les informations relatives aux données collectées elles-mêmes et contiennent des informations concernant la structure et la sémantique des données ». De plus, les métadonnées ne sont pas là par accident : elles ont été explicitement ajoutées (manuellement ou automatiquement) pour aider quelqu'un, quelque part, à un moment donné à trouver les informations associées. Pour faciliter la localisation des objets. Ils fournissent des informations sur chaque ensemble de données, comme la taille, le schéma d'une base de données, le format, l'heure de la dernière modification, les listes de contrôle d'accès, l'utilisation, etc. [38]

Il donne aux consommateurs de données, y compris les applications, une meilleure compréhension des données et de leurs propriétés, améliorant ainsi leur convivialité [7]. Les métadonnées ont toujours joué un rôle clé en favorisant la coopération de sources de données hétérogènes [34]. Ce rôle était déjà pertinent dans les architectures passées (par exemple, les systèmes d'information coopératifs et les entrepôts de données) mais il est devenu beaucoup plus crucial avec l'avènement des lacs de données [36].

2.3 Rôle de métadonnées

Les rôles que peuvent jouer les métadonnées sont variés et dépendent de la manière dont elles sont créées et utilisées [39]. On peut citer ses 4 rôles principaux suivants :

2.3.1 Classification

Les données peuvent avoir de nombreuses caractéristiques par lesquelles elles peuvent être regroupées ou classées. Ces catégories nous permettront de l'organiser et de le gérer. Les données peuvent être classées selon chacun des éléments suivants :

- Sujet – Ex : données financières, données sur les étudiants, données de collecte de fonds, données sur la santé, données sur les produits, etc.
- Utilisation – Ex : transactionnel, analytique, réglementaire, etc.
- Heure – Ex : données en direct et actuelles, historiques, prédictives, etc.
- Contenu – Ex : données géospatiales, données machine, données structurées vs non structurées, etc.
- Champ d'application – Ex : entreprise, externe, départemental, données de base, etc.

La gestion des données en fonction de la classification ou des groupes vous permet d'appliquer les mêmes normes, procédures et processus, ainsi que les gestionnaires et les propriétaires des données. Bien que vous puissiez avoir les mêmes données dans plusieurs groupes, cela ajoute une autre couche de complexité à la façon dont elles doivent être gérées. Par exemple, vous pouvez avoir les mêmes métadonnées indiquant qu'elles sont transactionnelles, qui relèvent du RGPD (règlement général sur la protection des données), qu'elles concernent l'ensemble de l'entreprise, ainsi que des données de santé en direct et non structurées. Habituellement, ces groupes peuvent être placés dans une hiérarchie pour déterminer quelle classification doit avoir la priorité sur les

autres. Dans le web, par exemple, les métadonnées ont clairement un rôle d'identification et de spécification de la donnée "référence ". Elles peuvent ainsi servir de base aux moteurs de recherche. Elles peuvent de plus faciliter ou améliorer la diffusion de données de base en établissant une sorte de "standard d'échange" [39].

2.3.2 Description

Le principal rôle des métadonnées est d'aider à la structuration et à la recherche d'information, Les métadonnées facilitent la recherche d'information, elles décrivent le contenu et les relations entre les données, et permettent des actions efficaces sur les données en leur fournissant un contexte. Elles promettent la possibilité de l'accès à des types de données plus nombreux et différents à un nombre croissant de consommateurs de données au sein d'une organisation [38]. La description des données nous aide à comprendre à la fois leurs aspects logiques et physiques. Les données décrites doivent inclure :

- Signification des données - Définitions commerciales, entités de modélisation de données et attributs
- Structure de données - Description des objets de données (entités, tables, enregistrements, etc.), leurs regroupements logiques et leurs relations
- Contenu des données - Les types de données tels que la date, le texte, le nombre, etc.
- Valeurs de données - Quelles valeurs sont autorisées, quelles données de référence sont disponibles, quels modèles ou plages de valeurs doivent-elles suivre, quelles contraintes doivent-elles respecter, etc.
- Lignage des données - Quelle est la source de données, comment les données ont-elles été créées, dérivées et / ou calculées, comment ont-elles été transformées, etc.

Les métadonnées descriptives se rapportent à une information et vous donnent des détails supplémentaires à ce sujet, par exemple un titre d'image, tandis que la structure se rapporte à de nombreuses choses et vous indique comment des informations similaires sont stockées et ce qu'elles signifient. Sans cette description, vous serez perdu et sa devient très difficile de collecter des données, les intégrer aux systèmes internes ou externes, les entretenir ou en tirer des informations utiles. Le suivi des modifications des données et la détection de tout processus qui

pose des problèmes lorsqu'on effectue une analyse de données est difficile si on ne dispose pas d'informations sur les données et le processus de déplacement des données [38].

2.3.3 Orientation

Les métadonnées peuvent servir de guide à tout utilisateur technique ou professionnel pour trouver les données dont il a besoin, via les moteurs de recherche ou d'autres processus. Ces métadonnées d'orientation peuvent être constituées de :

- Mots clés - Il peut s'agir de toutes les métadonnées décrites jusqu'à présent
- Taxonomies - Encore un autre exemple de la façon dont la classification aide
- Horodatage (Timestamp) - généralement ajoutés automatiquement au niveau de la table ou de la ligne.
- Rapports, processus, personnes associés - Savoir où les données apparaissent, qui sont les utilisateurs ou les gestionnaires de données, comment les données sont capturées et transformées pourrait servir de bon point de départ pour trouver ce dont vous avez besoin
- Synonymes, alias, termes apparentés.
- Fournir à vos parties prenantes les conseils nécessaires pour trouver les données dont elles ont besoin pour le rapport, l'analyse, les tests, le prototypage, le dépannage, etc. permet de gagner du temps et d'utiliser au mieux les ressources disponibles.

2.3.4 Contrôle

Les métadonnées peuvent fournir les connaissances nécessaires pour déterminer quels contrôles doivent être appliqués sur les données et quelles données doivent être contrôlées, gérer et protéger les droits. Le contrôle via les métadonnées permet d’appliquer des contraintes dues à : (1) Conformité réglementaire et politiques internes, (2) Conservation et archivage (Les métadonnées facilitent la gestion et l’archivage), (3) Confidentialité et sécurité (authentification), (4) Niveaux de service et exigences commerciales, (5) les prérequis techniques [23]. Ces contrôles permettent de garantir le respect des règles et réglementations internes et externes, des politiques et des exigences commerciales et techniques.

2.4 Type de métadonnées

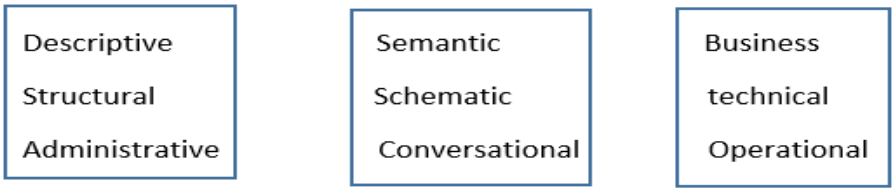


Figure 2.1 Types de classification des métadonnées

Suite à ce qui est dit dans [41], les métadonnées peuvent être divisées en trois catégories (voir le tableau 2.1), à savoir :

1. Métadonnées métier (business) : qui incluent des règles d'entreprise (par exemple, la limite supérieure et inférieure d'un champ, contraintes d'intégrité).
   Les métadonnées métier capturent ce que les données signifient pour l'utilisateur final afin de faciliter la recherche et la compréhension des champs de données, par exemple les noms d'entreprise, les descriptions, les balises, la qualité et les règles de masquage. Celles-ci sont liées à la définition des attributs métier afin que chacun interprète de manière cohérente les mêmes données par un ensemble de règles et de concepts définis par les utilisateurs métier. Un glossaire métier est un emplacement central qui fournit une description métier pour chaque élément de données grâce à l'utilisation d'informations de métadonnées. [38,41]
2. Métadonnées opérationnelles : qui incluent les informations générées automatiquement lors du traitement des données. Elles capturent la lignée, la qualité, le profil et la provenance (par exemple, quand les éléments de données sont-ils arrivés, où se trouvent-ils, d'où viennent-ils, quelle est la qualité des données, etc). Elle peut également contenir le nombre d'enregistrements rejetés pendant la préparation des données ou l'exécution d'une tâche, ainsi que le succès ou l'échec de cette exécution elle-même. Les métadonnées opérationnelles identifient également la fréquence à laquelle les données peuvent être mises à jour ou actualisées. [41]
3. Métadonnées techniques : Qui comprennent des informations sur la forme et la structure de chaque ensemble de données, telles que la taille et la structure du schéma ou le type de données (texte, images, JSON, etc.). La structure du schéma comprend les noms des champs, leurs types de données, leurs longueurs, s'ils peuvent être vides, etc. La structure est généralement fournie par une base de données relationnelle ou par l'en-tête d'une

feuille de calcul, mais peut également être ajoutée lors de l'ingestion et de la préparation des données [33, 41]. Certaines métadonnées techniques de base peuvent être obtenues directement à partir des ensembles de données (c'est-à-dire la taille), mais d'autres types de métadonnées sont dérivés. [38]

Catégorie	Exemple
Métier	Titre de livre, éditeur de livre , auteur
Technique	Taille, type de donnée, nom de la table , nom de la colonne
Opérationnelles	Droit d'accès, provenance, exigence de qualité

Tableau 2.1 classification de métadonnées.

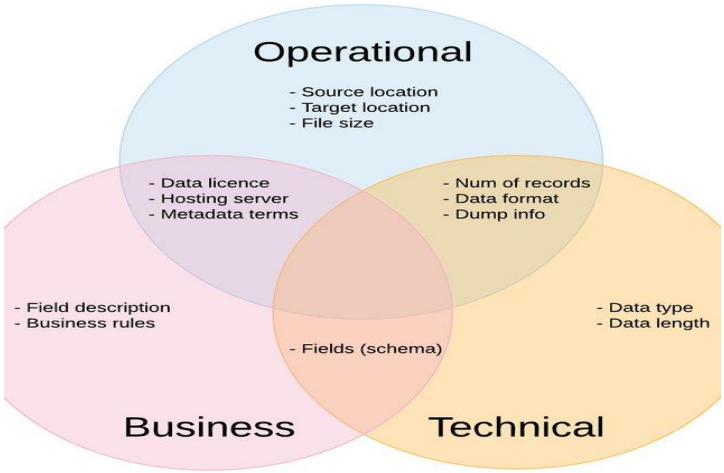


Figure 2.2 Classification des métadonnées [42]

Selon la figure 2.2, nous observons que ces trois sous-ensembles se croisent. Par exemple, puisque les métadonnées métier (Business) contiennent toutes les règles métier qui sont principalement exprimées en termes de champs de données et que le schéma de données est inclus dans les métadonnées techniques, nous pouvons conclure que les champs de données représentent l'intersection parfaite entre ces deux sous-ensembles. De manière analogue, les métadonnées techniques contiennent le type et la longueur des données, la possibilité qu'un champ puisse être NULL ou auto-incrémenté, le nombre d'enregistrements, le format des données et certaines informations de vidage. Ces trois derniers éléments sont communs aux

métadonnées opérationnelles, qui contiennent des informations telles que les sources et l'emplacement cible, ainsi que la taille du fichier. Enfin, l'intersection entre les métadonnées opérationnelles et commerciales représente des informations sur la licence de l'ensemble de données, le serveur d'hébergement, etc. (par exemple, voir les conditions de métadonnées de Dublin Core Metadata Initiative DCMI).

Il existe d'autre approches pour catégoriser les métadonnées (voir figure 2.1). L'Organisation nationale de normalisation de l'information (NISO) distingue trois types de métadonnées : les métadonnées descriptives, structurelles et administratives [33].

1. Métadonnées descriptives : Ce type de métadonnées (informations sur le contenu d'une ressource qui aide à la trouver ou à la comprendre). Les métadonnées descriptives sont la façon dont les données sont identifiées. Cela inclut des éléments tels qu'un titre, une date ou des mots-clés. Un exemple de métadonnées descriptives serait si un fichier audio était stocké dans un ordinateur, le temps d'exécution de l'audio serait des métadonnées descriptives [33]. Les métadonnées descriptives sont des métadonnées qui fournissent une description d'un objet. [40]
2. Métadonnées administratives : Les métadonnées administratives sont un terme générique faisant référence aux informations nécessaires à la gestion d'une ressource ou liées à sa création. Dans la sphère des métadonnées administratives se trouvent les métadonnées techniques, les informations sur les fichiers numériques nécessaires pour les décoder et les restituer, comme le type de fichier; la préservation des métadonnées prenant en charge la gestion à long terme et la future migration ou émulation de fichiers numériques, par exemple une somme de contrôle ou un hachage; et les métadonnées des droits, telles qu'une licence Créative Commons, qui détaille les droits de propriété intellectuelle attachés au contenu. [33]

   Les métadonnées administratives donc donnent des instructions importantes sur un fichier. Elles indiquent le type de restrictions à placer sur le fichier, par exemple qui peut y accéder ou non. Un exemple de métadonnées administratives serait un fichier d'entreprise identifié comme un type de fichier spécifique. Les métadonnées administratives fournissent des informations sur l'origine et la maintenance d'un objet [40]

3. Métadonnées structurelles : Les métadonnées structurelles font référence à la façon dont les données sont formatées et assemblées. Pensez-y comme une table des matières dans un livre. Elles expliquent comment les données sont liées les unes aux autres. Un exemple de métadonnées structurelles serait si un auteur incluait des notes dans son livre indiquant que ce ne serait que la première version de nombreux autres livres à venir. [33]

Le document « Data Wrangling » classe les métadonnées comme schématiques, sémantiques ou conversationnelles [35].

Les métadonnées schématiques couvrent les informations sur le schéma et la mise en forme, contenant ainsi une partie des métadonnées administratives et structurelles des NISO.

Les métadonnées sémantiques sont équivalentes aux métadonnées descriptives des NISO avec des informations supplémentaires sur les associations, couvrant ainsi une partie des métadonnées structurelles.

Enfin, les métadonnées conversationnelles couvrent toutes les informations sur les personnes qui ont précédemment accédé aux données, leurs idées et leurs expériences avec les données. L'idée étant que les connaissances et la compréhension des données ne doivent être acquises qu'une seule fois et sont ensuite partagées.

2.5 Gestion des métadonnées

La gestion des métadonnées (**Metadata Management)** n’est pas seulement une stratégie en plein essor. En effet, avec l’augmentation du volume de données, les métadonnées ont aussi été multipliées de façon exponentielles impliquant une nécessaire gestion de ces informations, véritable moteur de la gouvernance des données.

La gestion des métadonnées implique des activités centrées sur l'organisation et le traitement des actifs d'information avec des métadonnées [43]. Son objectif principal est de maximiser la valeur des données stockées [44].

Généralement, la gestion des métadonnées est effectuée au niveau d'un projet, d'un programme unique ou d'une initiative [45]. Si la portée s'étend à l'ensemble de l'organisation, elle s'appelle Enterprise Metadata Management (EMM) [43]. Comme le montre la figure 2.3, la gestion des métadonnées commence par la création d'un inventaire de données [45]. Une fois que les données existantes sont claires, elles peuvent être évaluées et analysées. Les données sont

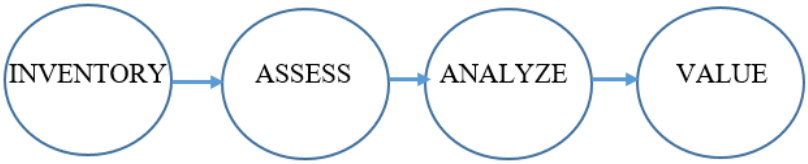


Figure 2.3 Pratiques de gestion des métadonnées [45]

analysées pour trouver des ensembles de données supplémentaires ou des métadonnées. S'ils sont trouvés, ils sont utilisés pour enrichir les données, ajoutant ainsi de la valeur.

Gartner cartographie les pratiques illustrées dans la figure 2.3 à quatre cas d'utilisation de la gestion des métadonnées, répertoriés dans la figure 2.4 [43]. La pratique d'inventaire est mappée au cas d'utilisation de la gouvernance des données. Le lien avec la gouvernance réside dans le fait qu'un inventaire est le résultat de métadonnées et de données maîtrisées. Il permet la création d'un glossaire métier, d'une piste d'audit, d'une analyse d'impact et plus encore.

Selon Gartner, le cas d'utilisation de la gouvernance des données se résume à soutenir la mise en œuvre de politiques et de règles.

Le deuxième cas d'utilisation, le risque et la conformité, est mis en correspondance avec la pratique d'évaluation. Les éléments de données doivent être cartographiés et identifiés, le traitement et les risques associés doivent être évalués et la lignée ainsi que l'analyse d'impact doivent être gérées en continu pour permettre la réalisation des exigences en matière de risque et de conformité. Ce cas d'utilisation inclut également la gestion de l'accès aux données en fonction des risques et des besoins de sécurité.

Le troisième cas d'utilisation identifié par Gartner est l'analyse des données. Il donne un aperçu des performances antérieures de l'organisation. L'objectif est d'apprendre des métadonnées précédemment exploitées afin de s'améliorer à l'avenir.

Enfin, la pratique « valeur » constitue également un cas d'utilisation de la gestion des métadonnées. Contrairement au cas d'utilisation de l'analyse de données, l'attention se porte sur

Figure 2.4 Cas d'utilisation de la gestion des métadonnées

les perspectives futures, les cas d'utilisation dépendent chacun des précédents.

Alors la gestion des métadonnées, est l'administration de données qui décrit d'autres données, impliquant la mise en place de politiques et de processus qui garantissent que les informations peuvent être intégrées, consultées, partagées, liées, analysées et maintenues de manière optimale dans toute l'organisation. Les métadonnées sont générées dès lors que des données sont créées, collectées, partagées ou encore supprimées. Ajoutons, qu'elles ne se limitent pas à être « des données sur les données » mais sont des informations structurées qui permettent de décrire, d'expliquer, localiser, préciser des données tout en facilitant l'utilisation et la gestion des informations.

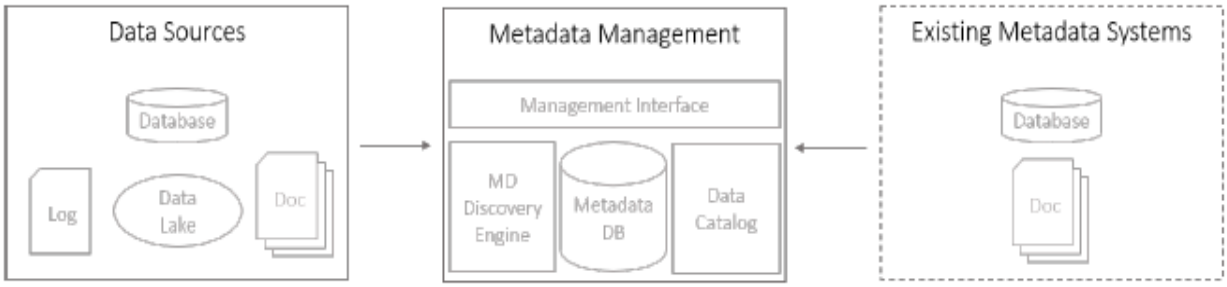


Figure 2.5 gestion des métadonnées [44]

2.6 Métadonnées pour les lacs de données

Les lacs de données sont généralement construits pour intégrer de très grands volumes de données non structurées de manière rapide. Parmi ses caractéristiques un catalogue de métadonnées qui renforce la qualité des données [51]. Chaque fois que des données provenant de différentes sources, contextes et avec divers schémas sont rassemblées, des métadonnées sont nécessaires pour assurer le suivi de ces données.

2.6.1 Spécificité des métadonnées dans les lacs de données

Les métadonnées ont toujours joué un rôle clé en favorisant la coopération de sources de données hétérogènes [45,26]. Ce rôle était déjà pertinent dans les architectures passées (par exemple, les systèmes d'information coopératifs et les entrepôts de données) mais il est devenu beaucoup plus crucial avec l'avènement des lacs de données [26]. En effet, dans cette nouvelle architecture, les métadonnées représentent la seule possibilité de garantir une gestion efficace et efficiente de l'interopérabilité des sources de données. Les métadonnées sont utilisées par l'analyste pour déchiffrer les données brutes trouvées dans le lac de données, ou en d'autres termes, les métadonnées sont la feuille de route de base des données qui résident dans le lac de données. [31]. Elles capturent des informations sur les données réelles, par exemple des informations de schéma, une sémantique ou une lignée [35, 36]. Elles garantissent que les données peuvent être

trouvées, fiables et utilisées. Il existe un grand nombre d'approches différentes pour la gestion des métadonnées. La qualité des données est fournie par un ensemble de métadonnées [51]. Lorsque seules les données brutes sont stockées dans le lac de données, l'analyste qui doit utiliser ces données est paralysé. Imaginez essayer de rechercher sur Wikipédia si aucun des articles n'avait de titre. Les données brutes en elles-mêmes ne sont tout simplement pas très utiles. Maintenant, lorsque les données brutes sont correctement étiquetées avec métadonnées et stockées ensemble dans le lac de données, vous disposez désormais d'un service incroyablement utile. [31]

2.6.2 Rôle des métadonnées dans le lac de données

Contrairement aux entrepôts de données ou aux SGBD, les lacs de données peuvent ne pas être accompagnés au départ de catalogues de données descriptifs et complets. Sans informations de métadonnées explicites, un lac de données peut facilement devenir un marais de données. Les catalogues de données sont essentiels pour la découverte et l'intégration à la demande dans les lacs de données ainsi que pour le nettoyage des données brutes. Castenado et Gidely [38] ont résumé le besoin de gestion des métadonnées en six éléments :

• **Visibilité des données :** Les lacs de données sont généralement construits pour intégrer de très grands volumes de données de manière rapide, ils ont la particularité de pouvoir ingérer automatiquement des données sans que les utilisateurs sachent quelles données sont disponibles. La visibilité des données permet à l'utilisateur d'être conscient des données existantes dans le lac de données. La visibilité des données peut être assurée en exposant les métadonnées associées à différentes catégories d'utilisateurs (professionnels de l'informatique, autorités de gouvernance, entreprises analystes, etc.), puis en leur permettant d'agir. Le suivi des modifications des données et la détection de tout processus qui pose des problèmes lorsque vous effectuez une analyse de données est difficile si vous ne disposez pas d'informations sur les données et le processus de déplacement des données.

• **Fiabilité :** Un lac de données peut stocker toutes les données, quelle que soit la source, quelle que soit la structure, et généralement aussi quelle que soit la taille. Chaque fois que des données provenant de différentes sources, contextes et avec divers schémas sont rassemblées, Pour garantir la fiabilité des données, les métadonnées de provenance peuvent être utilisées pour évaluer la qualité des données [56]. En outre, les métadonnées garantissent la fiabilité des

données non seulement pour les autorités de gouvernance internes, mais également pour les autorités externes en fournissant un lignage de données [53].

• **Archivage et cycle de vie :** les données dans les lacs de données peuvent atteindre des volumes énormes, une bonne pratique pour une meilleure gestion des données est d'archiver régulièrement les données inutilisées [55]. Dans ce contexte, les métadonnées peuvent être utilisées pour classer les données par nature (données de base, données transactionnelles, données sociales) ou selon leur âge, puis pour prioriser l'archivage.

• **Sécurité et confidentialité :** les lacs de données sont caractérisés par le nombre croissant des utilisateurs qui accèdent aux données pour différents objectifs. Cette situation peut entraîner de graves problèmes de sécurité, en particulier pour les données brutes accessibles en mode batch. Pour s'assurer que les mesures de sécurité appropriées sont prises, les métadonnées peuvent être utilisées par exemple pour baliser des éléments de données spécifiques puis en utilisant ces balises pour accorder différents modes d'accès [54]. Le problème de sécurité est encore plus grave dans les lacs de données publics car ils exposent leurs données à des utilisations ou des utilisateurs à haut risque [53].

• **Accès démocratisé aux données** : selon [53], et afin de réaliser des analyses en libre-service, les utilisateurs finaux peuvent accéder aux données par eux-mêmes. Pour cette raison, les métadonnées peuvent être utilisées pour aider les analystes commerciaux à mener des activités analytiques en leur fournissant des artefacts de métadonnées analytiques, tels que des schémas, des requêtes. [61] [55]

• **Optimisation :** les métadonnées permettent différents types d'optimisation. Par exemple, lorsque les données résident sur une lagune de données (data lagoons) [60], les métadonnées sont utiles pour connaître les données, les ressources, la sécurité et le modèle de coût que la lagune de données expose aux consommateurs de services de données, tels que les lacs de données. De même, si le lac de données stocke les données dans le cloud, les métadonnées peuvent également concerner les prix, les restrictions d'utilisation et peuvent permettre une optimisation des coûts [52].

D’autres éléments peuvent être ajoutées à partir de la littérature étudiée :

• **Accès aux données :** les métadonnées fournissent des informations sur l'accès aux données et éléments de données. Selon le cas d'utilisation du lac de données, les métadonnées peuvent décrire les données à différents niveaux. Par exemple, lors du déplacement ou de la transformation des

données dans différentes zones du lac de données, les métadonnées fournissent des détails techniques pour l'accès aux données, tels que

- Noms techniques, types de données, tailles…
- Emplacements physiques, [58] ou des ensembles de données [59], un dispositif de stockage, par ex. nœud de réseau [54],
- Mappages source-cible [53]
- Liens entre les vues intégrées du lac de données et ses sources [60].

Les métadonnées aident à réduire le temps nécessaire pour obtenir des informations en fournissant un accès facile à la découverte des données disponibles et en conservant une carte de suivi des données complète (lignage des données).

• **Pré-traitement des données :** le rôle des métadonnées est de fournir des informations statistiques sur les données [57] telles que le ratio de valeurs manquantes ou de dévoiler les incohérences et diversités des données en termes de types de données, de longueurs, de formats lorsqu'ils se rapportent aux mêmes éléments de données [41].

• **Intégration de données :** Le rôle des métadonnées consiste à rassembler des morceaux des mêmes données à un haut niveau de description tout en résolvant les similitudes et autres problèmes d'intégration ([42], [57]).

2.6.4 Gestion des métadonnées dans le lac de données

La gestion des métadonnées est un élément crucial aux lacs de données [46]. En plus d'extraire les métadonnées des sources et d'enrichir les données avec des métadonnées significatives (telles que la description détaillée des données et les contraintes d'intégrité), les systèmes de gestion des métadonnées doivent prendre en charge un stockage efficace des métadonnées (surtout lorsqu'elles deviennent volumineuses) et des requêtes répondant aux métadonnées. L'extraction automatique des métadonnées est un sujet important dans les environnements de lac de données, car de grandes quantités de données sont ingérées et stockées. Certains concepts de lac de données fournissent même un système de gestion des métadonnées complet pour stocker et interroger les métadonnées [47]. Pour extraire des métadonnées schématiques, même à partir de sources de données sans schéma, le profilage de schéma peut être utilisé [48].

Il existe divers modèles de métadonnées pour les lacs de données. Certains d'entre eux ne fournissent que peu de description et de détails de réalisation [49], tandis que d'autres sont conçus pour une application spécifique [50]. Outre la structure et la sémantique des données, les

métadonnées sur l'origine des données sont tout aussi importantes [41]. Les métadonnées de lignage décrivent d'où proviennent les données, comment elles ont été produites et comment elles ont été traitées. Cependant, les métadonnées de lignage ne sont pas ou sont insuffisamment prises en compte dans tous les modèles de métadonnées étudiés pour les lacs de données. Un seul modèle de métadonnées mentionne la provenance [50], mais il n'y a aucune autre explication sur le stockage ou l'utilisation des informations de provenance. Bien que la gestion des métadonnées soit cruciale pour les lacs de données, aucune stratégie complète de gestion des métadonnées couvrant toutes les métadonnées des lacs de données n'est disponible.

2.6.5 Système de métadonnées pour le lac de données

Les auteurs dans [51] identifient six fonctionnalités principales qui devraient idéalement être fournies par le système de métadonnées d'un lac de donnée :

- L'enrichissement sémantique : également appelé annotation sémantique ou profilage sémantique, consiste à générer une description du contexte des données, par exemple avec des balises, pour les rendre plus interprétables et compréhensibles. L'annotation sémantique joue un rôle clé dans l'exploitation des données, en résumant les jeux de données contenus dans le lac afin qu'ils soient plus compréhensibles pour l'utilisateur. Il peut également être utilisé comme base pour identifier les liaisons de données.
- L'indexation des données : consiste à mettre en place une structure de données pour récupérer des ensembles de données en fonction de caractéristiques spécifiques (mots-clés ou modèles). L'indexation permet d'optimiser l'interrogation des données dans le lac grâce au filtrage des mots clés. Il est particulièrement utile pour la gestion des données textuelles, mais peut également être utilisé dans un cadre semi-structuré ou structuré contexte des données.
- La génération et la conservation de liens : est le processus de détection de relations de similarité ou d'intégration de liens préexistants entre des ensembles de données. L'intégration de liaisons de données peut être utilisée pour élargir l'éventail des analyses possibles du lac en recommandant des données liées à celles qui intéressent l'utilisateur. Les liaisons de données peuvent également être utilisées pour identifier des grappes de données, c'est-à-dire des groupes dans lesquels les données sont fortement liées les unes aux autres et sensiblement différentes des autres données.

- Le polymorphisme des données : comme le stockage de plusieurs représentations des mêmes données. Chaque représentation correspond aux données initiales, modifiées ou reformatées pour un besoin spécifique. Par exemple, un document textuel peut être représenté sans mots vides ou comme un sac de mots. Il est essentiel dans le contexte des lacs de données de structurer au moins partiellement les données non structurées pour permettre leur analyse automatisée. Le stockage simultané de plusieurs représentations des mêmes données évite notamment de répéter les prétraitements et accélère ainsi les analyses.
- Le contrôle de version des données : fait référence à la capacité du système de métadonnées à prendre en charge les modifications de données tout en conservant les états précédents. Cette capacité est essentielle dans les lacs de données, car elle garantit la reproductibilité des analyses et prend en charge la détection et la correction d'éventuelles erreurs ou incohérences. Le versionnage permet également de prendre en charge une évolution ramifiée des données, notamment dans leur schéma.
- Le suivi de l'utilisation : enregistre les interactions entre les utilisateurs et le lac de données. Les interactions sont généralement des opérations de création, de mise à jour et d'accès aux données. L'intégration de ces informations dans le système de métadonnées permet de comprendre et d'expliquer les éventuelles incohérences dans les données. Il peut également être utilisé pour gérer des données sensibles, en détectant des intrusions. Le suivi de l'utilisation et la gestion des versions des données sont étroitement liés, car les interactions conduisent dans certains cas à la création de nouvelles versions ou représentations des données. Cependant, ces caractéristiques ne sont pas systématiquement proposées ensemble.

2.6.6 Quelques Exemples de système de gestion des métadonnées

- Un exemple de système de gestion des métadonnées est Google Dataset Search (GOODS) qui extrait et collecte des métadonnées pour les ensembles de données générés et utilisés en interne par Google [35]. Les métadonnées collectées vont des informations spécifiques à l'ensemble de données telles que les propriétaires, les horodatages et le schéma aux relations entre plusieurs ensembles de données telles que leur similitude et leur provenance. GOODS rend les ensembles de données accessibles et consultables en exposant leurs métadonnées collectées dans des profils d'ensembles de données.

- *Constance* est un autre exemple qui, en plus d'extraire des métadonnées à partir de sources, enrichit les sources de données en annotant les données et les métadonnées avec des informations sémantiques [26]. Constance rend les métadonnées générées accessibles dans un environnement de réponse aux requêtes basé sur un modèle. En revanche, le projet *Ground* collecte le contexte des données qui comprend les applications, les comportements et les changements de données et stocke les métadonnées dans des structures graphiques interrogeables [46].
- *Skluma* extrait des métadonnées profondément intégrées, des sujets latents et des métadonnées contextuelles à partir de fichiers dans divers formats dans un lac de données [54] et permet une découverte basée sur des sujets [54]. La découverte de métadonnées fournit l'abstraction des données qui est cruciale pour la compréhension et la découverte des données, mais il reste des opportunités pour mieux extraire et relier les connaissances des lacs et incorporer ces connaissances dans les bases de connaissances existantes (générales ou spécifiques au domaine).

2.7 Conclusion

Au cours de ce chapitre, nous avons abordé le concept des métadonnées, qui sont des données sur les données. Nous avons présenté la définition des métadonnées, ses rôles, ses types, la gestion des métadonnées, et le concept métadonnées pour le lac de données ; plus précisément nous avons défini les métadonnées dans le lac de données, le rôle des métadonnées dans le lac de données, la gestion des métadonnées dans le lac de données, les six fonctionnalités de système de métadonnées pour le lac de données et aussi nous avons cité quelques exemples de gestion de métadonnées dans le lac de données.

Chapitre III

3. Analyse et Conception du système

3.1 Introduction

Dans ce chapitre, nous détaillons l'application du processus 2TUP (2-Track Unified Process) à notre travail. Notre application de 2TUP est allégée vues les spécificités de notre travail dans le sens qu'il s'agit d'un système destiné à des concepteurs de métadonnées et d'exploitant de ces métadonnées par leur visualisation. Le processus 2TUP respecte le modèle en Y du développement et qui sépare les aspects techniques des aspects fonctionnels. Dans ce qui suit, nous allons présenter l'étape de capture des besoins fonctionnels, c'est-à-dire l'énoncé de l'ensemble des besoins liés au métier et domaine traités par le système ainsi que leur modélisation. Ensuite, l'étape d'analyse permet d'approfondir la capture des besoins fonctionnels en détaillant l'aspect *données* par le développement du modèle statique et en détaillant l'aspect *traitements* en développant le modèle dynamique. Dans le processus 2TUP, la conception du

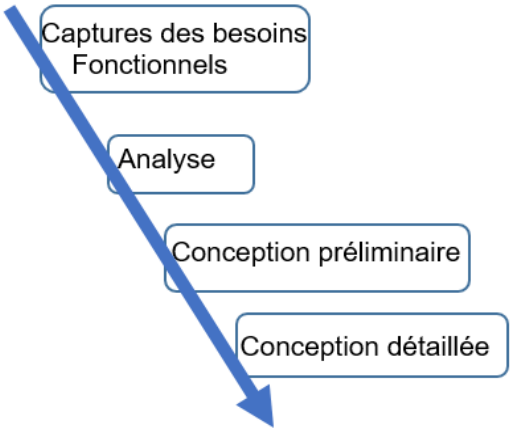


Figure 3.1 Procesus 2TUP allégé

système est divisée en trois étapes : conception générique qui est indépendante du métier, la conception préliminaire qui adapte la conception générique aux besoins métier et la conception détaillée qui détaille la conception préliminaire. Dans ce chapitre nous avons regroupé les deux dernières étapes de conception. L'adaptation du processus 2TUP est schématisée en figure 3.1.

3.2 Les besoins fonctionnels

Les besoins fonctionnels de notre système sont les suivants :

a. Création des métadonnées d'un lac de données

b. Ajout des métadonnées des sources du lac de données : les sources qui alimentent le lac de données peuvent être diverses. Des exemples sont

- Sources relationnelles : il s'agit d'ajouter des métadonnées des sources représentant des données relationnelles, telles que des bases de données, Des schémas relationnels. Le terme source fait abstraction et dépend du SGBDR utilisé et doit contenir directement les tables relationnelles.
- Sources graphes : il s'agit d'ajouter des métadonnées des sources de type graphe de propriétés (source NoSQL). La source doit contenir directement les nœuds et les relations avec leurs propriétés.
- Sources Documents : il s'agit d'ajouter des métadonnées de type documents (source NoSQL). La source doit contenir directement les items qui constituent la collection des documents

L'ajout des métadonnées sur les sources peut être manuelle pour les sources graphes et documents. Par contre, elle peut être automatique ou manuelle pour celles décrivant des sources relationnelles, et ce vu qu'en général, les sources relationnelles sont naturellement munies de leurs métadonnées dans le dictionnaire de données.

Aussi, le système doit permettre d'enrichir la description de chaque source par des métadonnées structurelles, comme suit :

- Pour les sources relationnelles, le système doit permettre l'ajout des métadonnées sur les tables et pour chaque table, ses colonnes, avec la possibilité d'altérer la structure des tables ;
- Pour les sources graphes, le système doit permettre l'ajout des métadonnées sur les nœuds et les relations entre eux ;
- Pour les documents, le système doit permettre l'ajout des métadonnées sur les items des documents et sur les relations entre eux ;

c. Accès aux données : il s'agit de pouvoir à travers les métadonnées, visualiser le contenu du lac de données, notamment en exploitant les métadonnées d'emplacement physique de chaque élément du lac. Par exemple, il doit être possible d'afficher le contenu d'une table, d'une colonne, d'un nœud, d'un sous-graphe (i.e. nœud et relations), d'un item individuel de document ou d'une sous-document.

d. Ajout de nouveaux types de sources : ce besoin répond à un souci d'extensibilité du système. Il s'agit de pouvoir ajouter de nouveaux types de sources de données telles que les feuilles de calcul Excel, du contenu multimédia, du texte non structuré, etc. tout en associant à chaque source les métadonnées adéquates.

3.3 Identification des acteurs

Notre système comprend deux types d'acteurs : l'architecte des métadonnées qui ajoute ces dernières et l'exploitant du lac de données qui visualise et accède aux données du lac. La figure 3.2 décrit la responsabilité de chaque acteur.

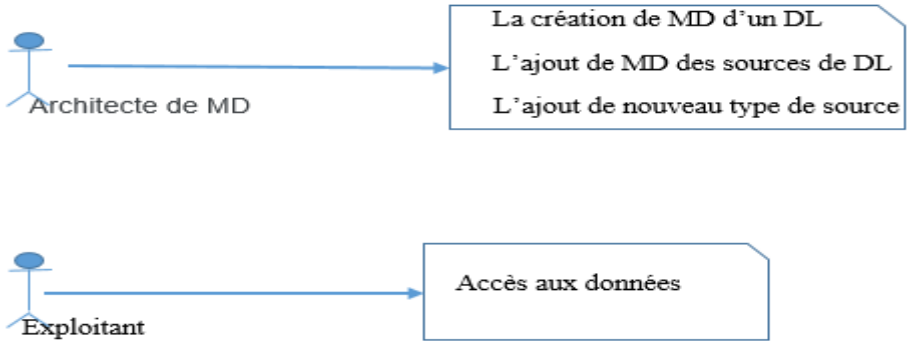
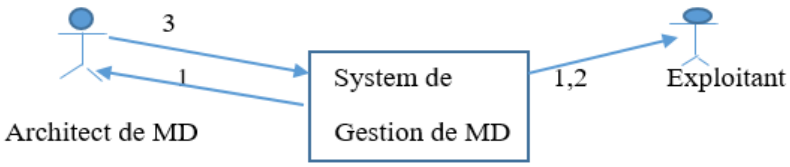


Figure 3.2 responsabilité des acteurs du système

3.4 Identification des messages et modélisation du contexte

La figure 3.3 représente le diagramme de contexte dynamique de notre système.



Numéro du message	Nature de message
1	Métadonnées du DL
2	Données
3	Ajout la structure du DL (MD)

Figure 3.3 diagramme de contexte dynamique

3.5 Identification de cas d'utilisations

A partir de l'analyse des besoins fonctionnels, nous avons identifié les cas d'utilisation ci-dessous. Notons que dans la suite, nous omettons de reprendre le terme métadonnées à chaque fois qu'on fait référence à un élément du lac de données (exemple, ajouter un lac de données veut implicitement dire ajouter les métadonnées d'un lac de données)

- **Cas n°1:** Gérer un lac de données

  But : Décrire un lac de données (DL)

  Acteur : Architecte des métadonnées

  Description : il s'agit d'ajouter un lac de données et ses métadonnées et le gérer

  Enchaînements

  - Enchaînement 1: ajouter un DL
  - Enchainement 2 : consulter un DL (sa structure et ses métadonnées)
  - Enchainement 3 : modifier un DL (métadonnées)
  - Enchainement 4 : supprimer un DL. La suppression d'un lac de données entraîne la suppression de toute sa structure.

- **Cas n°2 :** Ajouter et gérer une source relationnelle

  But : Prendre en compte une source relationnelle dans le lac de données

  Résumé : Ce cas permet d'ajouter une source relationnelle en l'attachant à un DL existant et de la gérer.

  Acteur : Architecte des métadonnées

  Précondition : l'existence d'au moins un DL (c'est-à-dire que le 1er cas d'utilisation doit être déjà exécuté au moins une fois).

  Enchaînements

  - Enchaînement 1 : ajouter une source relationnelle
  - Enchainement 2 : consulter une source relationnelle (sa structure et ses métadonnées)
  - Enchainement 3 : modifier une source relationnelle (métadonnées)
  - Enchainement 4 : supprimer une source relationnelle. La suppression d'une source relationnelle entraîne la suppression de toute sa structure.

- **Cas n°3 :** Ajouter et gérer une source graphe

  But : Prendre en compte une source graphe dans le lac de données

Résumé : Ce cas permet d'ajouter une source graphe en l'attachant à DL existant et la gérer

Acteur : Architecte des métadonnées

Précondition : l'existence d'au moins un DL (c'est-à-dire que le 1er cas d'utilisation doit être déjà exécuté au moins une fois).

Enchaînements

- Enchaînement 1: ajouter une source graphe
- Enchainement 2 : consulter une source graphe (sa structure et ses métadonnées)
- Enchainement 3 : modifier une source graphe (métadonnées)
- Enchainement 4 : supprimer une source graphe. La suppression d'une source graphe entraîne la suppression de toute sa structure.

- **Cas n°4 :** Ajouter et gérer une source document

  But : Prendre en compte une source document dans le lac de données

  Résumé : Ce cas permet d'ajouter une source de donnée document, d'ajouter les items du document et ajouter les relations entre ces items tout en attachant le document à un DL existant et de le gérer

  Acteur : Architecte des métadonnées

  Précondition : l'existence d'au moins un DL (c'est-à-dire que le 1er cas d'utilisation doit être déjà exécuté au moins une fois).

  Enchaînements

  - Enchaînement 1 : ajouter une nouvelle source document
  - Enchainement 2 : ajouter les items de document
  - Enchainement 3 : ajouter les relations entre les items de document si elles existent
  - Enchainement 4 : consulter une source document (sa structure et ses métadonnées)
  - Enchainement 5 : supprimer une source document. La suppression d'une source document entraîne la suppression de toute sa structure.

- **Cas n°5** : Ajouter et gérer une table relationnelle

  But : Prendre en compte la structure de tables d'une source relationnelle dans le lac de données, d'ajouter les colonnes de la table tout en attachant cette dernière à un DL existant et de gérer la table.

Résumé : Ce cas permet d'ajouter une table et ajouter ses colonnes (au moins une colonne) à une source relationnelle

Acteur : Architecte des métadonnées

Précondition : l'existence d'au moins une source relationnelle (schéma) où se trouve la table (c'est-à-dire que le 2ème cas d'utilisation doit être déjà exécuté).

Enchaînements

- Enchaînement 1 : ajouter une nouvelle table
- Enchainement 2 : ajouter les colonnes de la table
- Enchainement 3 : ajouter les relations entre les colonnes des tables si elles existent
- Enchainement 4 : consulter une table relationnelle (sa structure et ses métadonnées)
- Enchainement 5 : supprimer une table. La suppression d'une table entraîne la suppression de toute sa structure en termes de colonnes.

- **Cas n°6** : Ajouter et gérer un nœud de graphe

  But : Prendre en compte la structure d'une source graphe dans le lac de données en lui ajoutant des nœuds, tout en les attachant à un DL existant et de gérer le nœud.

  Résumé : Ce cas permet d'ajouter un nœud et de l'associer à une source graphe.

  Acteur : Architecte des métadonnées

  Précondition : l'existence d'au moins une source graphe pour attacher le nœud (c'est-à-dire que le 3ème cas d'utilisation été déjà exécuté avant son exécution).

  Enchaînements

  - Enchaînement 1 : ajouter un nouveau noeud
  - Enchainement 2: modifier un noeud
  - Enchainement 3 : consulter un noeud (ses métadonnées)
  - Enchainement 4 : supprimer un nœud. La suppression d'un nœud entraîne la suppression des relations avec les autres nœuds et du graphe s'il s'agit du dernier nœud.

- **Cas n°7** : Ajouter et gérer une relation (arcs) entre noeuds

  But : Prendre en compte la structure d'une source graphe dans le lac de données en lui ajoutant des relations entre nœuds et de gérer ces relations

Résumé : Ce cas permet d'ajouter une relation entre nœuds et de l'associer aux nœuds et à une source graphe

Acteur : Architecte des métadonnées

Précondition : l'existence d'au moins un nœud de graphe (c'est-à-dire que le 6ème cas d'utilisation été déjà exécuté au moins une fois avant son exécution).

Enchaînements

- Enchaînement 1 : ajouter une nouvelle relation entre noeuds
- Enchainement 2 : modifier une relation entre noeuds (ses métadonnées)
- Enchainement 3 : consulter une relation entre noeuds (ses métadonnées)
- Enchainement 4 : supprimer une relation entre noeuds.

- **Cas n°8** : Ajouter et gérer un item de document

  But : Prendre en compte la structure d'une source document dans le lac de données en lui ajoutant des items supplémentaires.

  Résumé : Ce cas permet d'ajouter un item et de l'associer à une source document.

  Acteur : Architecte des métadonnées

  Précondition : l'existence d'au moins une source document pour attacher l'item (c'est-à-dire que le 3ème cas d'utilisation été déjà exécuté avant son exécution).

  Enchaînements

  - Enchaînement 1 : ajouter un nouvel item
  - Enchainement 2 : modifier un item de document (les métadonnées)
  - Enchainement 3 : consulter un item de document (les métadonnées)
  - Enchainement 4 : supprimer un item de document. Cette suppression entraîne la suppression du document s'il s'agit du seul item du document.

- **Cas n°9** : Ajouter et gérer une relation entre items

  But : Prendre en compte la structure d'une source document dans le lac de données en lui ajoutant des relations entre items

  Résumé : Ce cas permet d'ajouter une relation entre items et de l'associer aux items et à une source document

  Acteur : Architecte des métadonnées

  Précondition : l'existence d'au moins un item de document (c'est-à-dire que le 6eme cas d'utilisation été déjà exécuté au moins une fois avant son exécution).

Enchaînements

- Enchaînement 1 : ajouter une nouvelle relation entre items
- Enchainement 2 : modifier une relation entre items (ses métadonnées)
- Enchainement 3 : consulter une relation entre items (ses métadonnées)
- Enchainement 4 : supprimer une relation entre items.

- **Cas n°10** : Gérer les propriétés de métadonnées

  But : permettre de prendre en compte différents types de métadonnées (techniques, métier, opérationnelles)

  Résumé : Ce cas permet d'ajouter les propriétés des métadonnées indépendamment de leurs associations aux éléments du lac de données

  Acteur : Architecte des métadonnées

  Précondition :/

  Enchaînements

  - Enchaînement 1 : ajouter une nouvelle propriété des métadonnées
  - Enchainement 2 : modifier une propriété des métadonnées (type ou nom)
  - Enchainement 3 : consulter une propriété des métadonnées
  - Enchainement 4 : supprimer une propriété des métadonnées. La suppression d'une propriété de métadonnées entraîne la suppression des liens entre elle et tout élément du lac de données.

- **Cas n°11**: Gérer les concepts

  But : Ce cas d'utilisation répond au besoin d'extensibilité des métadonnées du lac de données. Ce cas d'utilisation peut être vu comme un méta-cas d'utilisation.

  Résumé : Ce cas permet d'ajouter de nouveaux concepts selon le format des données (fichier excel, image, vidéo…)

  Acteur : Architecte des métadonnées

  Précondition : le concept ne doit pas exister auparavant. La consultation des concepts existants est nécessaire au préalable.

  Enchaînements

  - Enchaînement 1 : ajouter un nouveau concept
  - Enchainement 2 : modifier un concept (son nom). Cela entraîne la modification au niveau des instances de ce concept

- Enchainement 3: consulter un concept
- Enchainement 4 : supprimer un concept existant. Cela entraîne la suppression des instances de ce concept.

- **Cas n°12** : Gérer les liens entre concepts

  But : Ce cas permet de prendre en compte la structuration des concepts du lac de données.

  Résumé : ce cas permet d'ajouter, modifier ou supprimer une relation entre les concepts du lac de données.

  Acteur : Architecte des métadonnées

  Précondition : Le cas d'utilisation n°11 doit être exécuté au moins une fois.

  Enchaînements

  - Enchaînement 1 : ajouter une relation entre concepts.
  - Enchainement 2 : modifier une relation entre concepts (son nom). Cela entraîne la modification de la même relation entre les instances correspondantes du lac de données.
  - Enchainement 3 : supprimer une relation entre concepts. Cela entraîne la suppression de la relation entre les instances correspondantes du lac de données.

- **Cas n°13** : Gérer une instance de concept

  But : ce cas d'utilisation est une abstraction des cas d'utilisation de 1 à 9. Il permet de prendre en compte les instances d'un concept dans le lac de données selon les changements produits dans les sources du lac.

  Résumé : Ce cas d'utilisation permet d'ajouter, modifier, consulter ou supprimer une instance de concept du lac de données.

  Acteur : Architecte des métadonnées

  Précondition : /

  Enchaînements

  - Enchaînement 1 : ajouter une instance de concept
  - Enchainement 2 : modifier une instance de concept (les métadonnées).
  - Enchainement 3 : supprimer une instance de concept

- **Cas n°14** : Accéder aux données du DL

But : permettre aux exploitants du lac de données d'accéder à ses données (exemple : contenu d'une table, d'un graphe, d'un document, d'un nœud, …)

Résumé : ce cas d'utilisation permet de choisir certains éléments du lac et de voir leur donnée si la visualisation s'applique à cet élément.

Acteur : Exploitant

Enchaînements

- Enchaînement 1 : choisir un élément du lac de données
- Enchainement 2 : afficher les données de l'élément choisi

3.6 Diagramme de cas d'utilisation

La figure 3.4 représente le diagramme de cas d'utilisation de notre système.

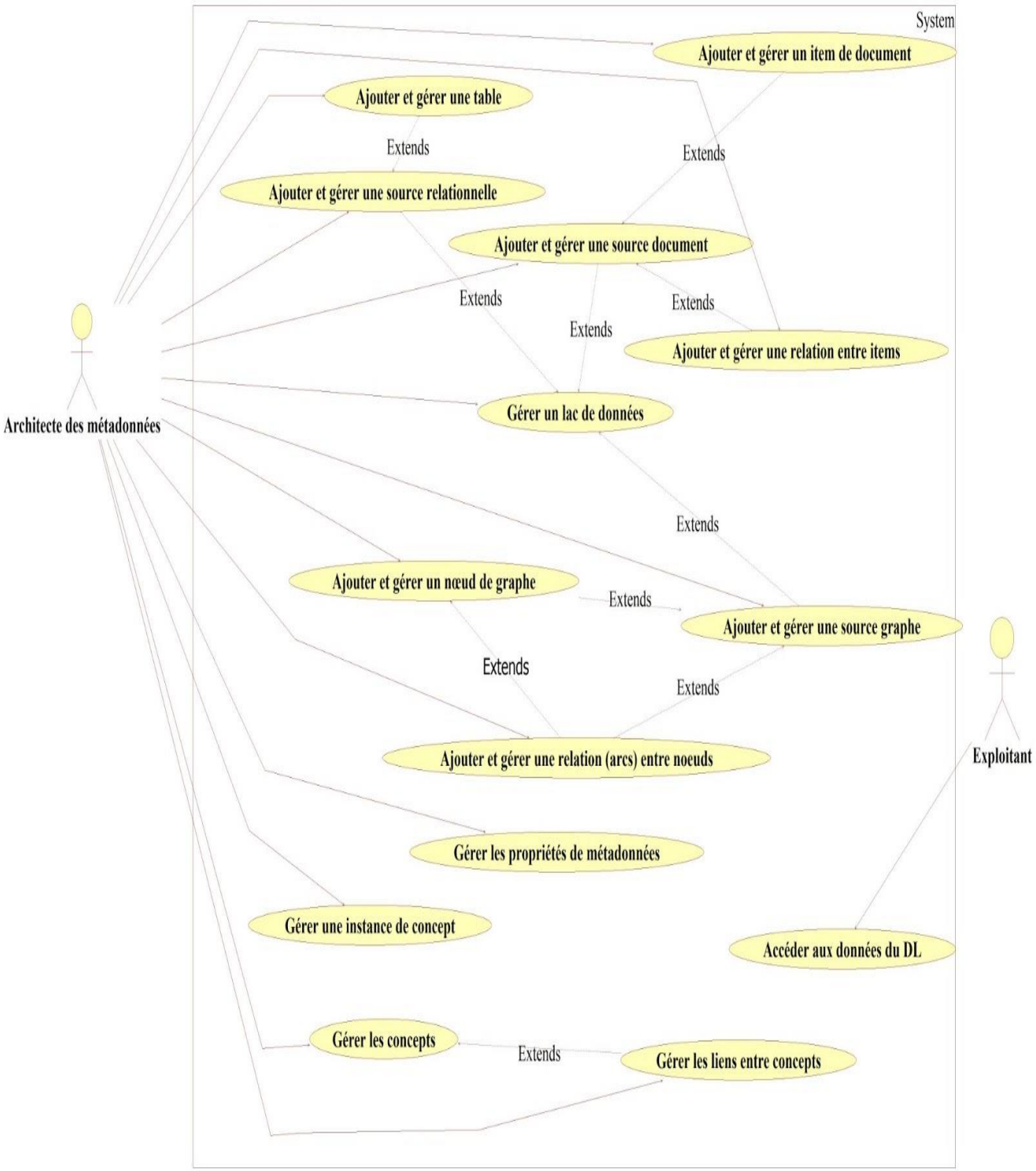


Figure 3.4 diagramme de cas d'utilisation de notre système

3.7 Identification des classes candidates

1. Cas d'utilisation Gérer un lac de données : on identifie la classe DataLake et Properiété_MD (voir figure 3.5).

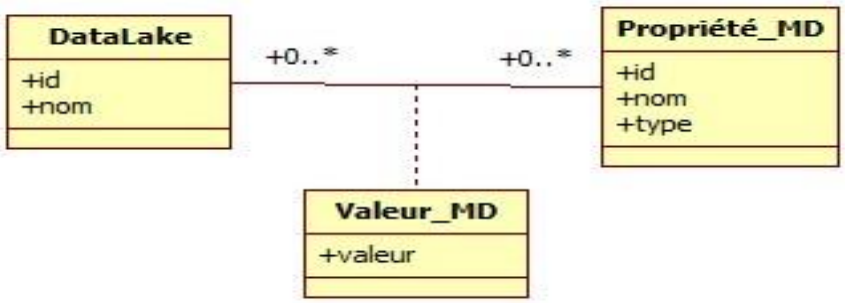


Figure 3.5 diagramme de classes candidates de cas d'utilisation gérer un lac de données.

2. Cas d'utilisation Ajouter et gérer une source relationnelle : on identifie les classes DataLake, Relationnelle_Ds et Properiété_MD

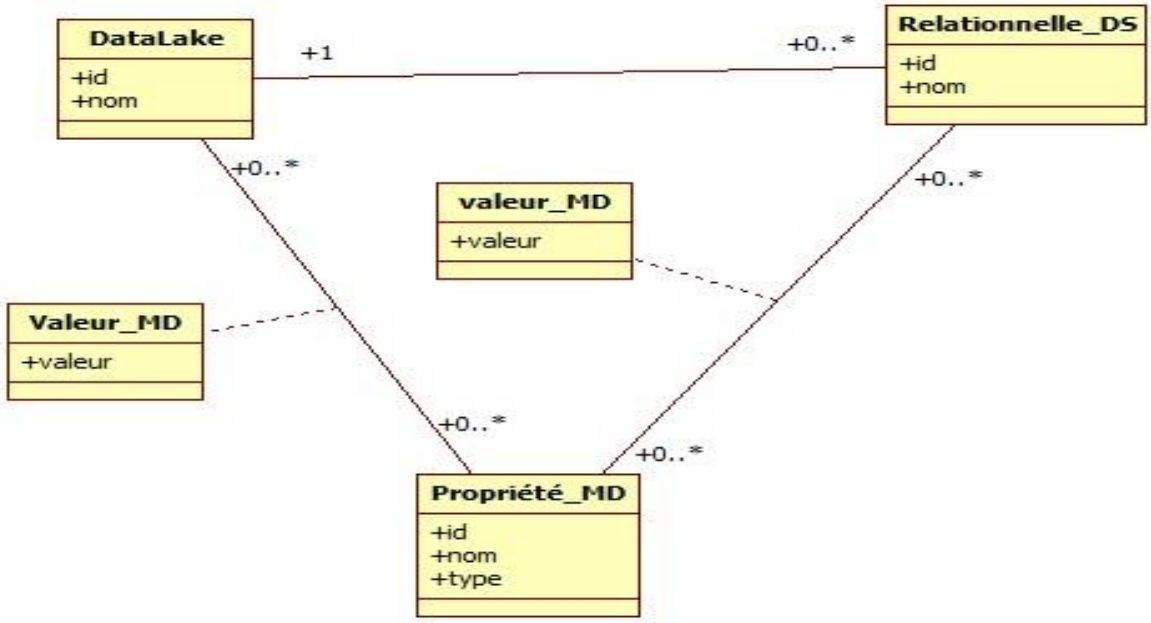


Figure 3.6 diagramme de classes candidates de cas d'utilisation ajouter et gérer une source relationnelle.

3. Cas d'utilisation Ajouter et gérer une source graphe : on identifie les classes DataLake, Graph_DS et Propriété_MD

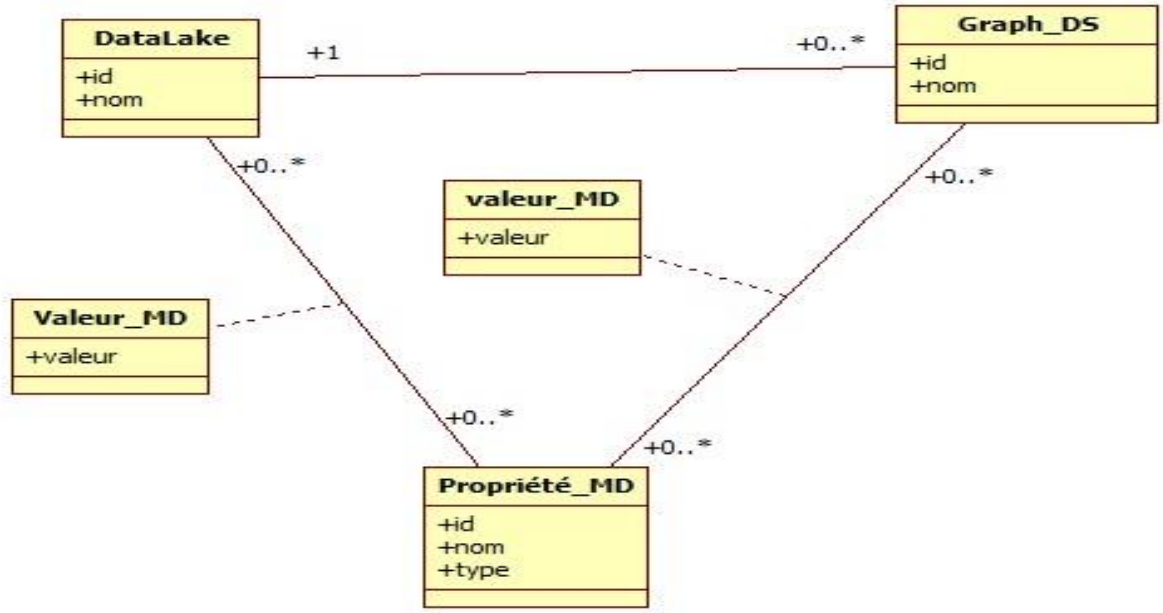


Figure 3.7 diagramme de classes candidates de cas d'utilisation Ajouter et gérer une source graphe

4. Cas d'utilisation Ajouter et gérer une source document : on identifie les classes DataLake, Document_DS, Item, RelationD et Propriété_MD

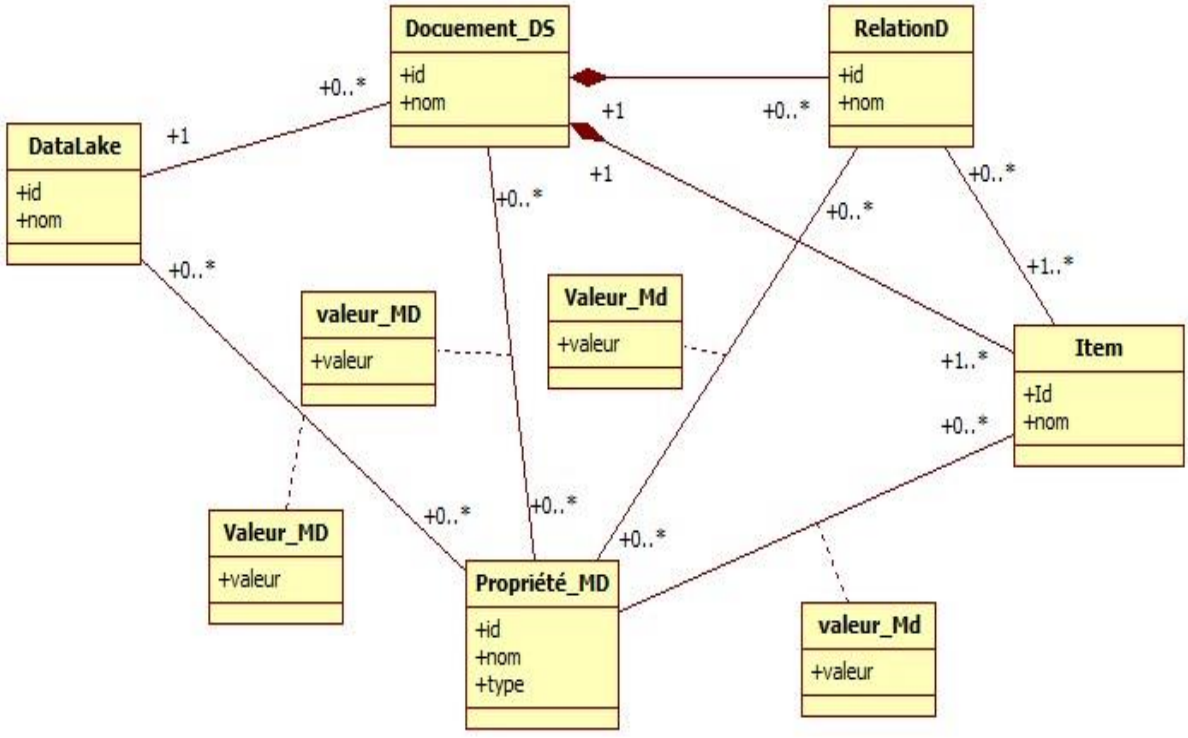


Figure 3.8 diagramme de classe candidates de cas d'utilisation Ajouter et gérer une source document.

5. Cas d'utilisation Ajouter et gérer une table relationnelle : Relationnelle_DS, Table, columun et Propriété_MD (voir la figure 3.9)

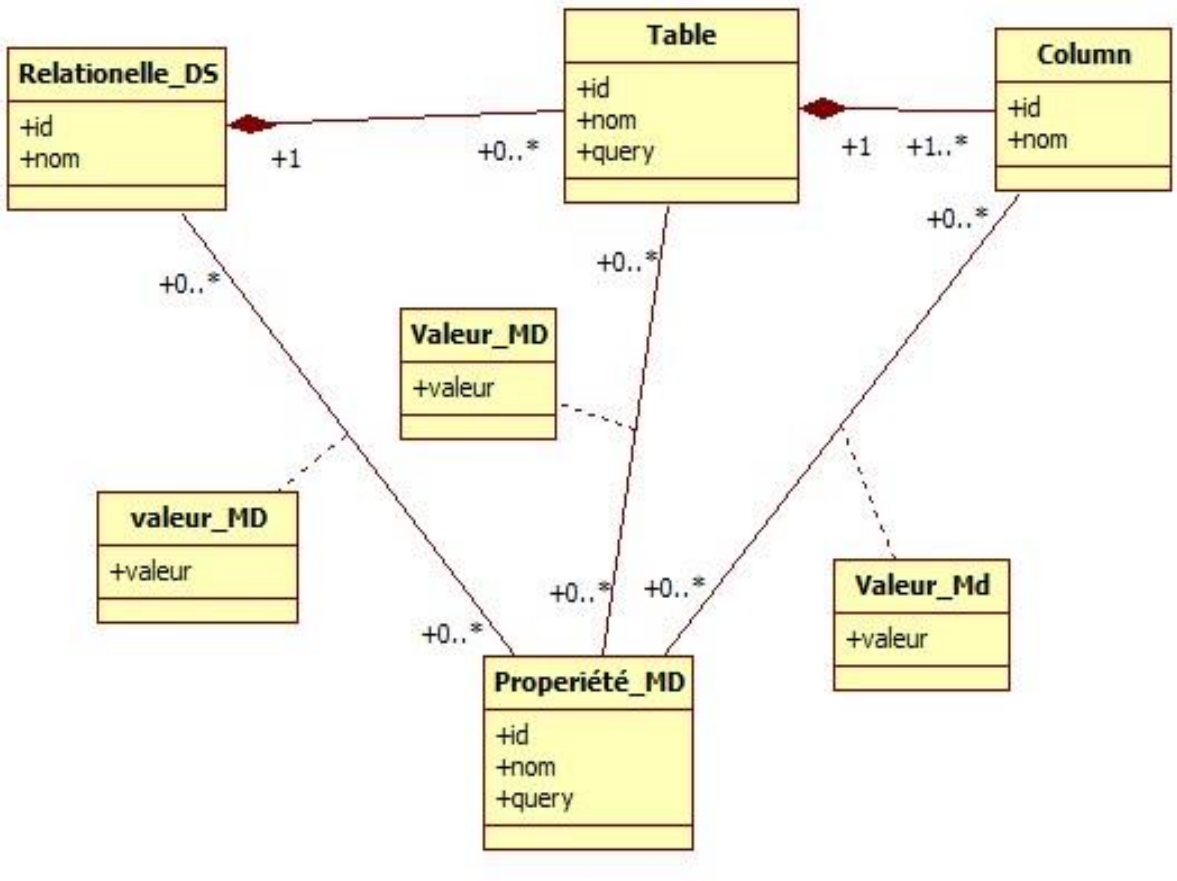


Figure 3.9 diagramme de classes candidates pour le cas d'utilisation ajouter et gérer une table relationnelle

6. Cas d'utilisation Ajouter et gérer un nœud de graphe : Graph_DS, Node et Propriété_MD (voir la figure 3.10)

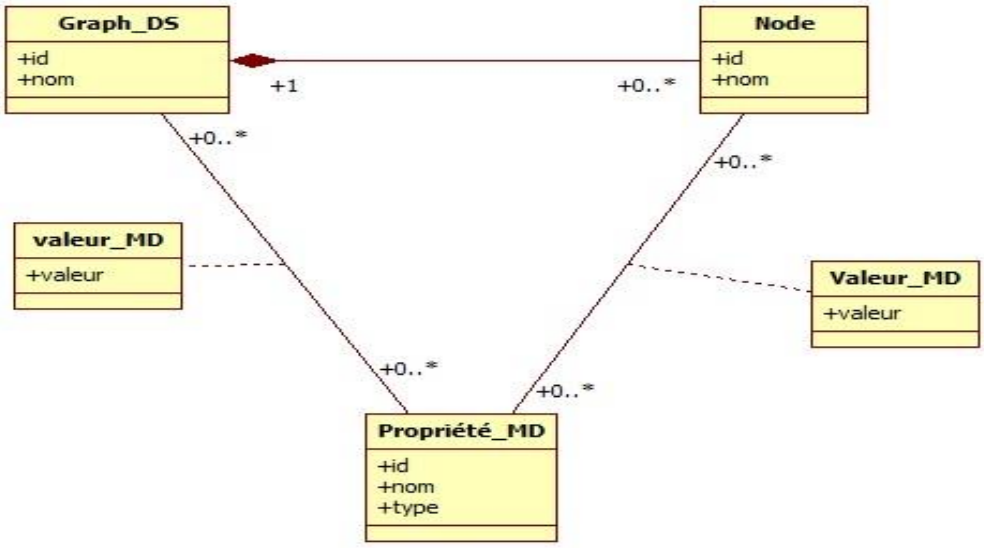


Figure 3.10 diagramme de classes candidates pour le cas d'utilisation ajouter et gérer un noeud de graphe.

7. Cas d'utilisation Ajouter et gérer une relation (arcs) entre noeuds : Graph_DS, Node, RelationG et Propriété_MD (voir la figure 3.11)

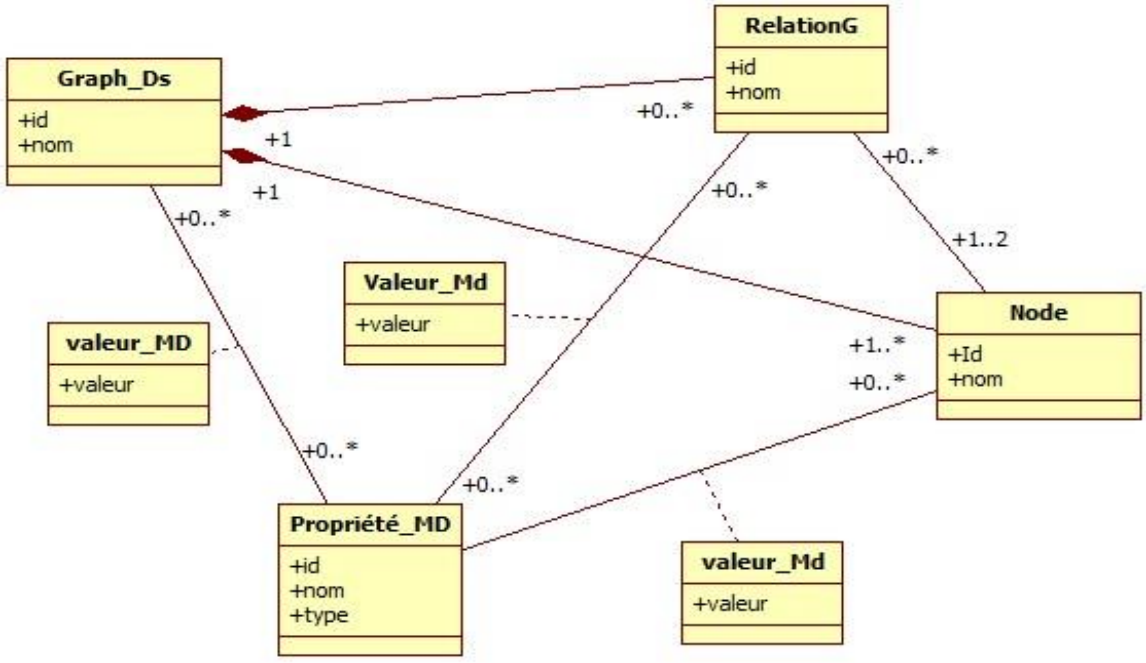


Figure 3.11 diagramme de classes pour le cas d'utilisation Ajouter et gérer une relation (arcs) entre nœuds

8. Cas d'utilisation Ajouter et gérer un item de document : Document_DS, Item, RelationD et Propriété_MD (voir la figure 3.12)

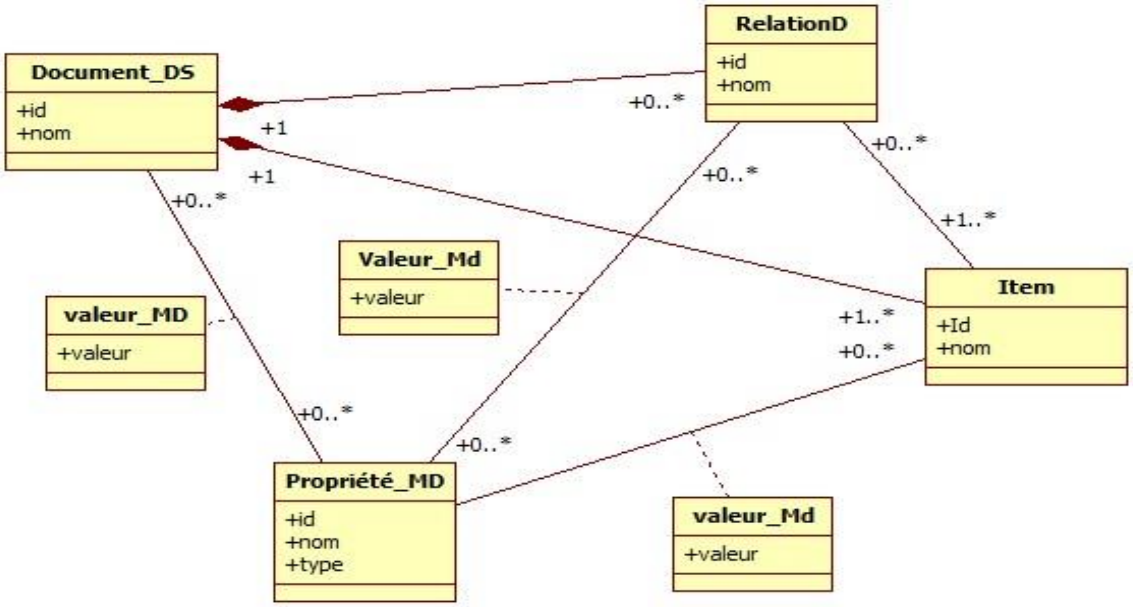


Figure 3.12 digramme de classes candidates pour le cas d'utilisation Ajouter et gérer un item de document

9. Cas d'utilisation Ajouter et gérer une relation entre items : Document_DS, Item,RelationD et Propriété_MD (voir la figure 3.13)

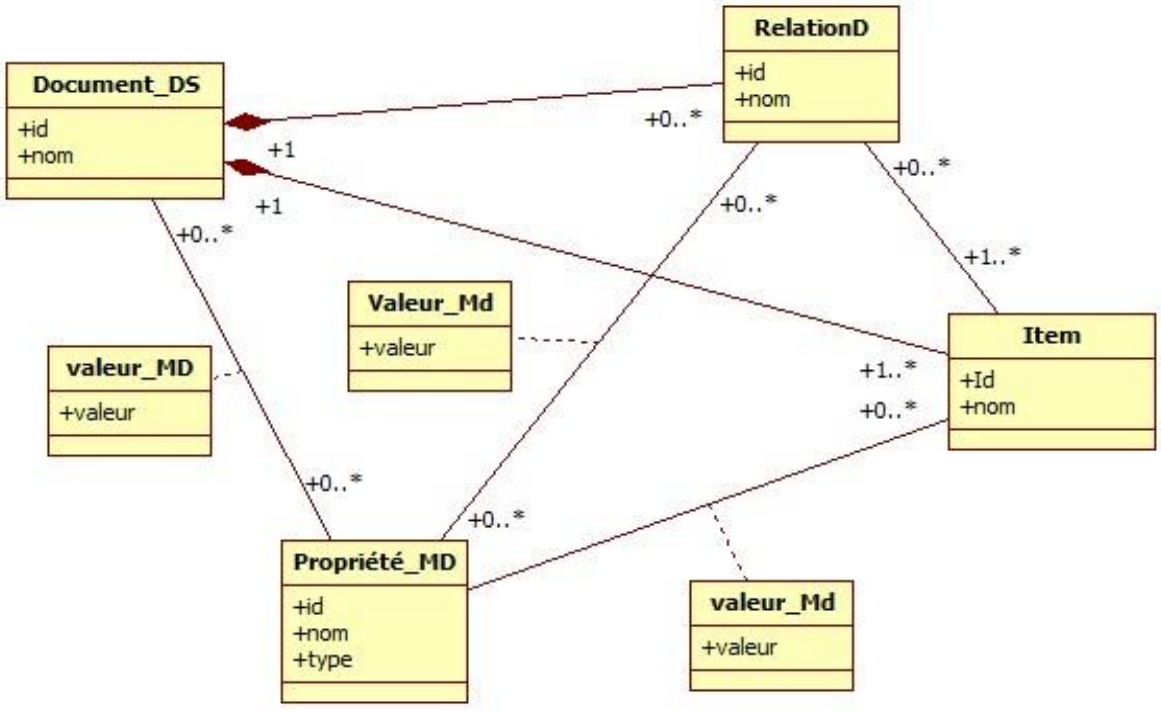


Figure 3.13 diagramme de classes candidates pour le cas d'utilisation Ajouter et gérer une relation entre items

10. Cas d'utilisation Gérer les propriétés de métadonnées : la classe propriété (voir la figure 3.14)

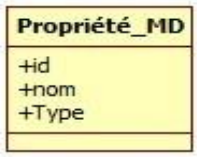


Figure 3.14 diagramme de classes candidates pour le cas d'utilisation gérer les propriétés de métadonnées

11. Cas d'utilisation Gérer les concepts : la classe concepts (voir la figure 3.15)

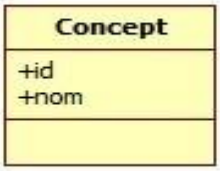


Figure 3.15 diagramme de classes candidates pour le cas d'utilisation gérer les concepts

12. Cas d'utilisation Gérer les liens entre concepts : la classe concept (voir la figure 3.16)

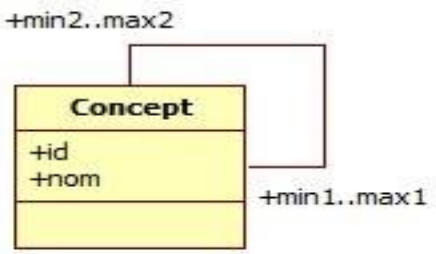


Figure 3.16 diagramme de classes candidates pour le cas d'utilisation Gérer lien entre concepts

13. Gérer une instance de concept : la classe instance (voir la figure 3.17)

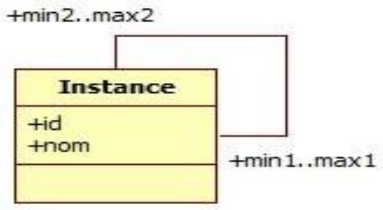


Figure 3.17 diagramme de classes candidates pour le cas d'utilisation gérer une instance de concept

14. Cas d'utilisation Accéder aux données du DL : les classes Relationnelle_DS, Document_DS, Item, Table, Column, Graph_DS et Node (voir la figure 3.18)

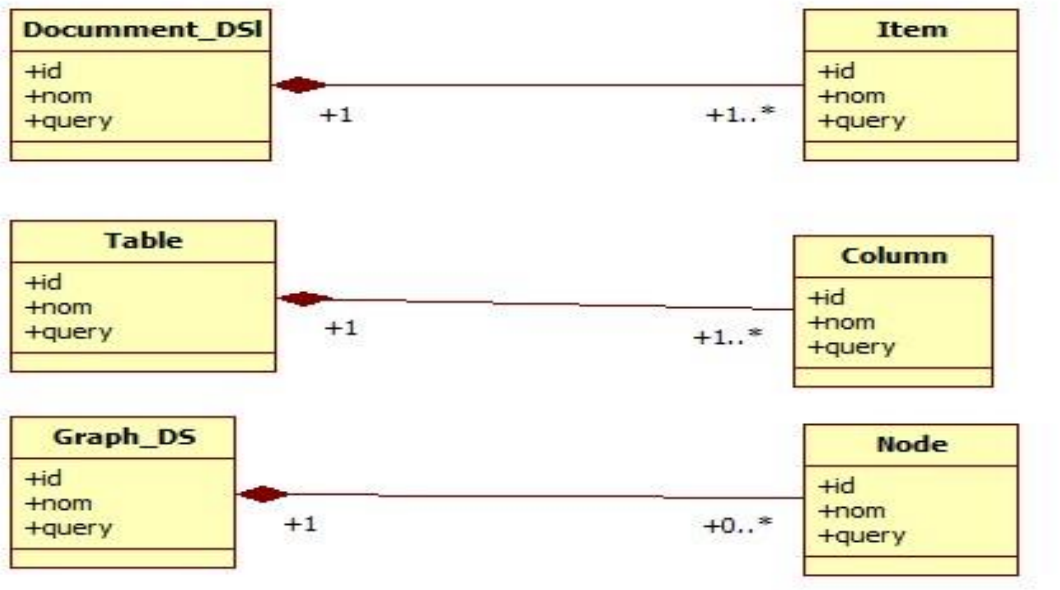


Figure 3.18 diagramme de classes candidates pour le cas d'utilisation accéder aux données du DL

3.8 Diagramme de classes global du schéma du lac de données

La figure 3.19 représente le diagramme de classe global de notre système. Notons que ce diagramme correspond au niveau "schéma" du lac de données dans le sens qu'il concerne des lacs de données concrets par opposition au modèle de lac qui contient les concepts et les liens entre eux.

Remarque : La classe d'association valeur_MD est dupliquée entre chaque classe et la classe Propriété_MD , et pour des raisons de clarté, nous avons omis de la dupiquer à chaque fois.

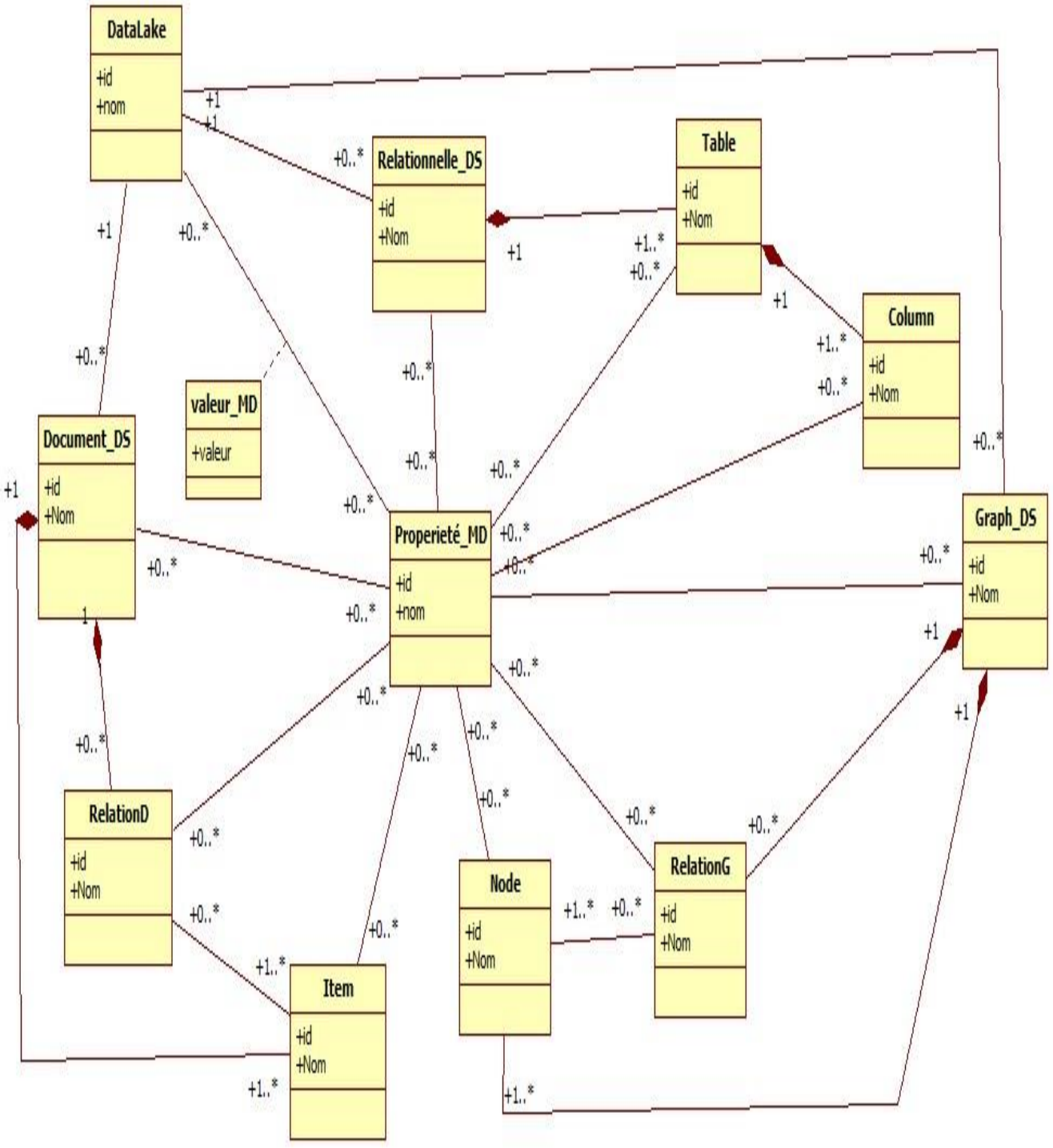


Figure 3.19 diagramme de classe globale de notre système

3.9 Diagramme de classes Modèle de métadonnées

La figure 3.20 représente le diagramme de classe de modèle de métadonnées de notre système.

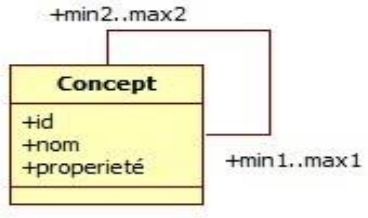


Figure 3.20 diagramme de classe de modèle de métadonnées

3.10 Diagramme de classes Instance d'un concept de lac de données

La figure 3.21 représente le diagramme de classe des instances de concepts d'un lac de données de notre système.

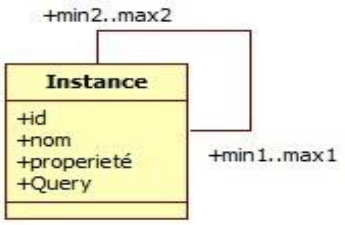


Figure 3.21 diagramme de classe des instances de notre système

3.11 Diagrammes de séquences : les diagrammes de séquences permettent de montrer l'intéraction entre les acteurs et les objets concrets du diagramme de classes.

1. Cas d'utilisation *Gérer un lac de données* : la figure 3.22 représente le diagramme de séquence du cas d'utilisation gérer un lac de données.

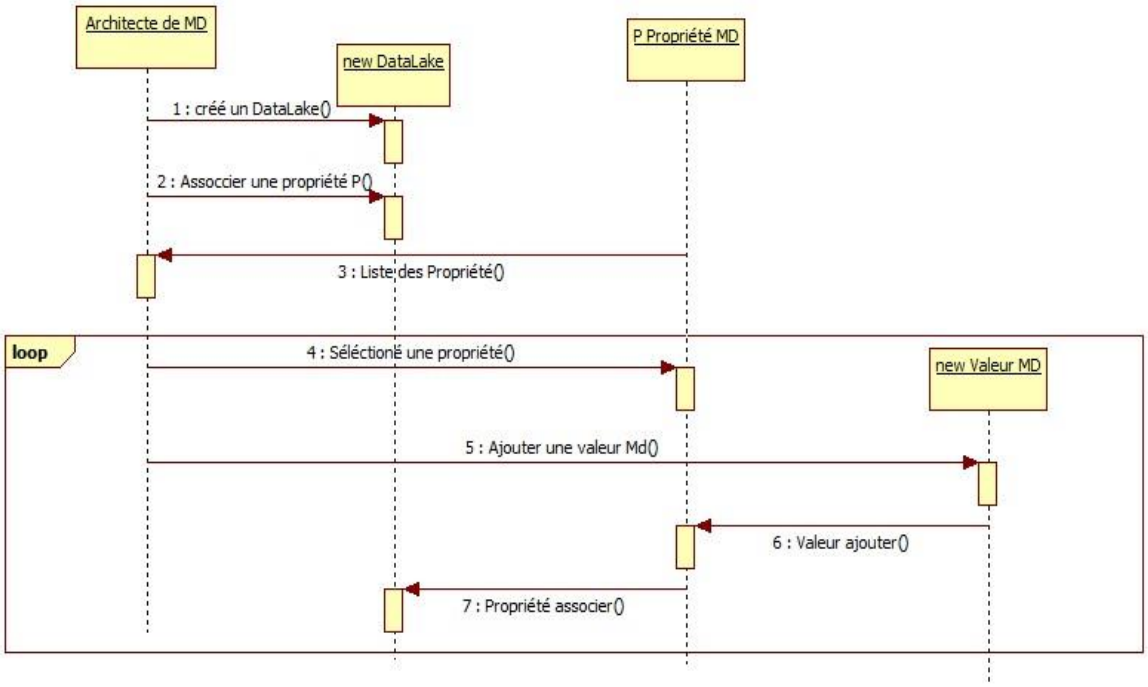


Figure 3.22 diagramme de séquence du cas d'utilisation gérer un lac de données.

Remarque : Pour tous les diagrammes de séquence qui suivent : l'association des éléments aux propriétés de métadonnées se fait de la même manière.

2. Cas d'utilisation *Ajouter et gérer une source relationnelle* : la figure 3.23 représente le digramme de séquence du cas d'utilisation ajouter et gérer une source relationnelle.

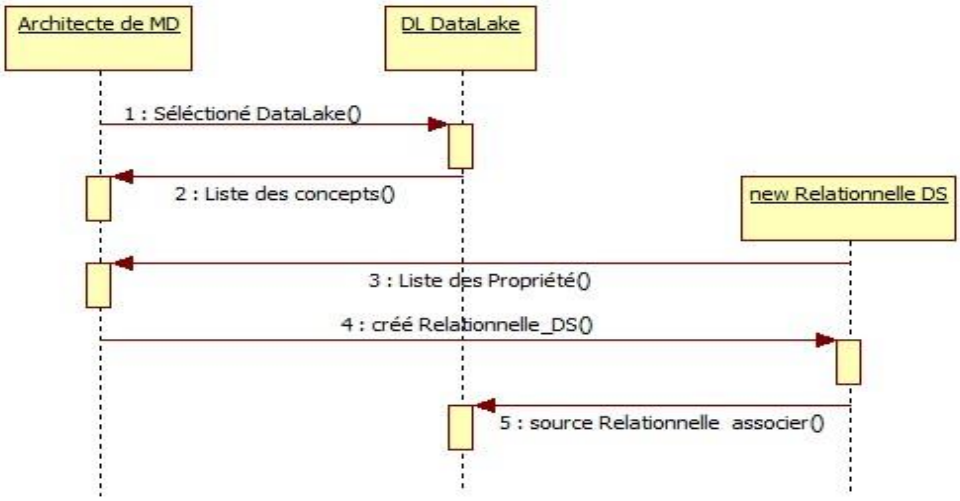


Figure 3.23 digramme de séquence du cas d'utilisation ajouter et gérer une source relationnelle.

3. Cas d'utilisation *Ajouter et gérer une source graphe* : la figure 3.24 représente le diagramme de séquence de cas d'utilisation ajouter et gérer une source graphe

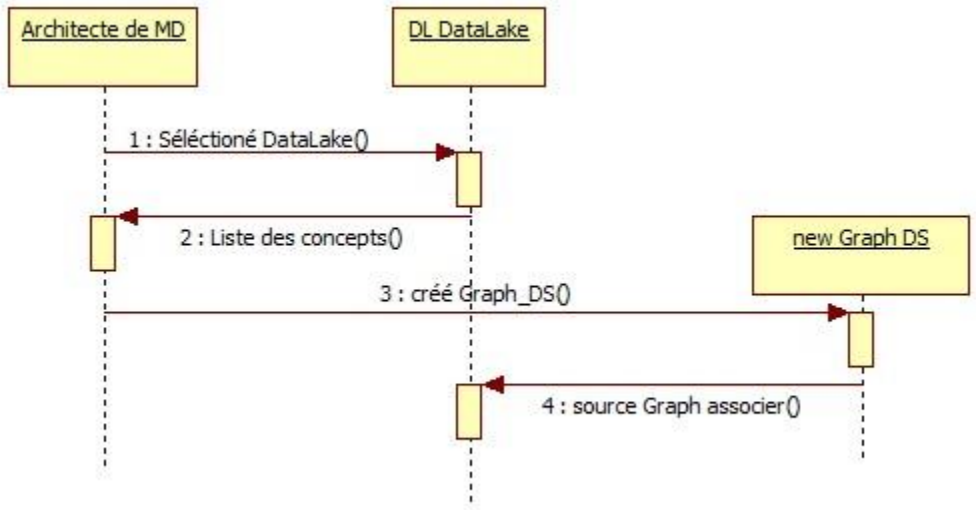


Figure 3.24 diagramme de séquence du cas d'utilisation ajouter et gérer une source graphe.

4. Cas d'utilisation *Ajouter et gérer une source document* : la figure 3.25 montre le diagramme de séquence du cas d'utilisation ajouter et gérer une source document

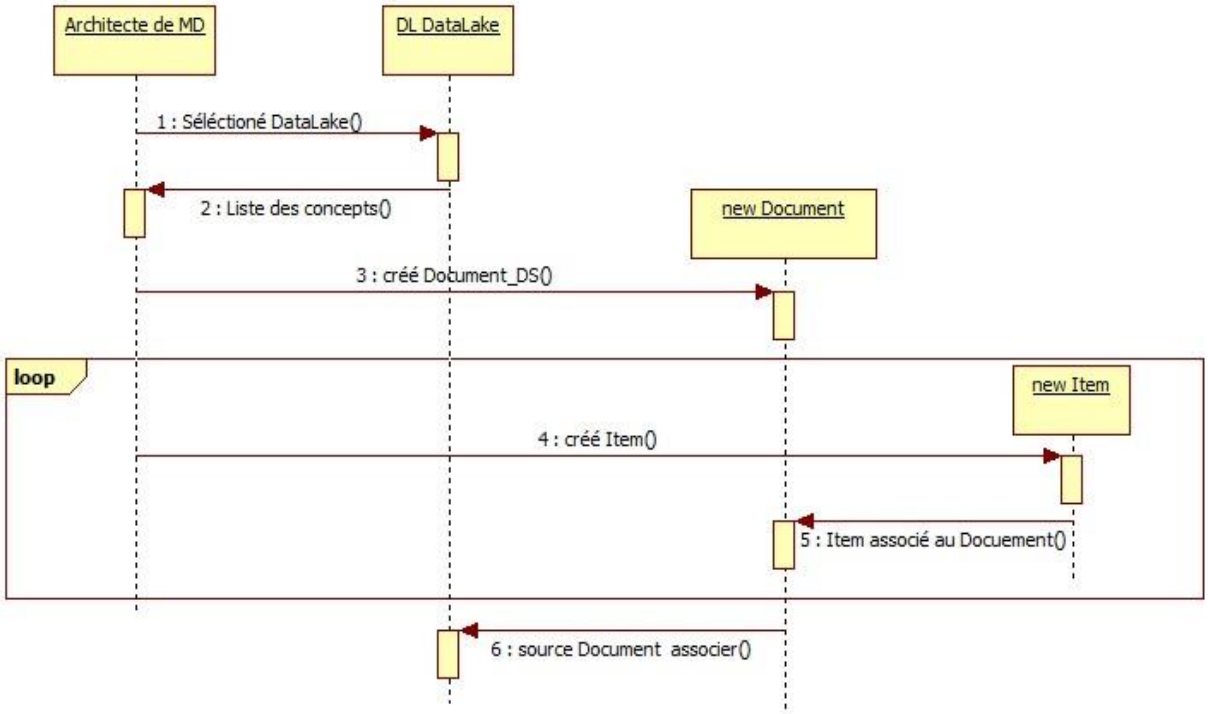


Figure 3.25 diagramme de séquence du cas d'utilisation ajouter et gérer une source document.

5. Cas d'utilisation *Ajouter et gérer une table relationnelle* : la figure 3.26 illustre le diagramme de séquence du cas d'utilisation ajouter et gérer une table relationnelle

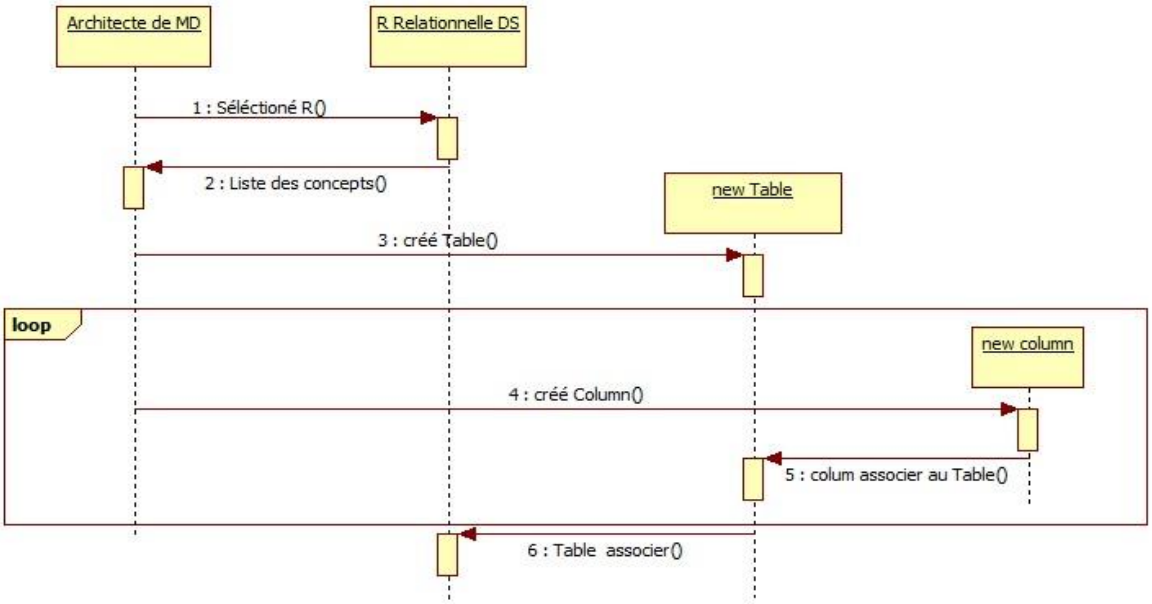


Figure 3.26 diagramme de séquence du cas d'utilisation ajouter et gérer une table relationnelle.

6. Cas d'utilisation *Ajouter et gérer un nœud de graphe* :la figure 3.27 représente le diagramme de séquence du cas d'utilisation ajouter et gérer un nœud de graphe

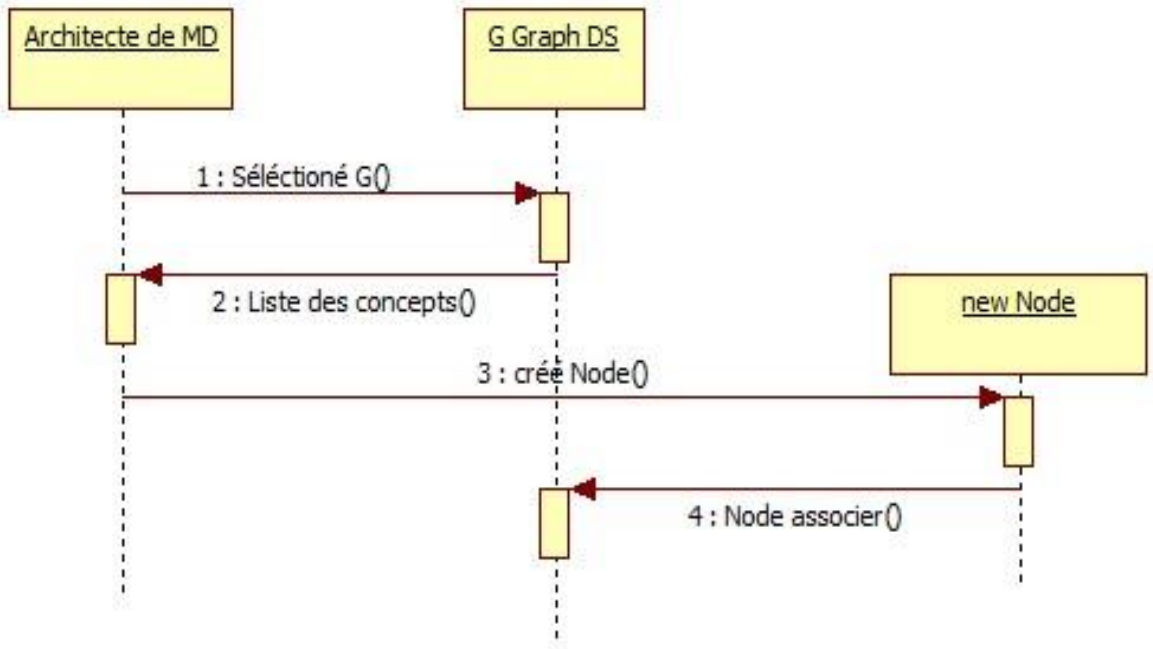


Figure 3.27 diagramme de séquence du cas d'utilisation ajouter et gérer un nœud de graphe.

7. Cas d'utilisation *Ajouter gérer une relation (arcs) entre nœuds* : la figure 3.28 illustre le diagramme de séquence du cas d'utilisation ajouter gérer une relation (arcs) entre nœuds.

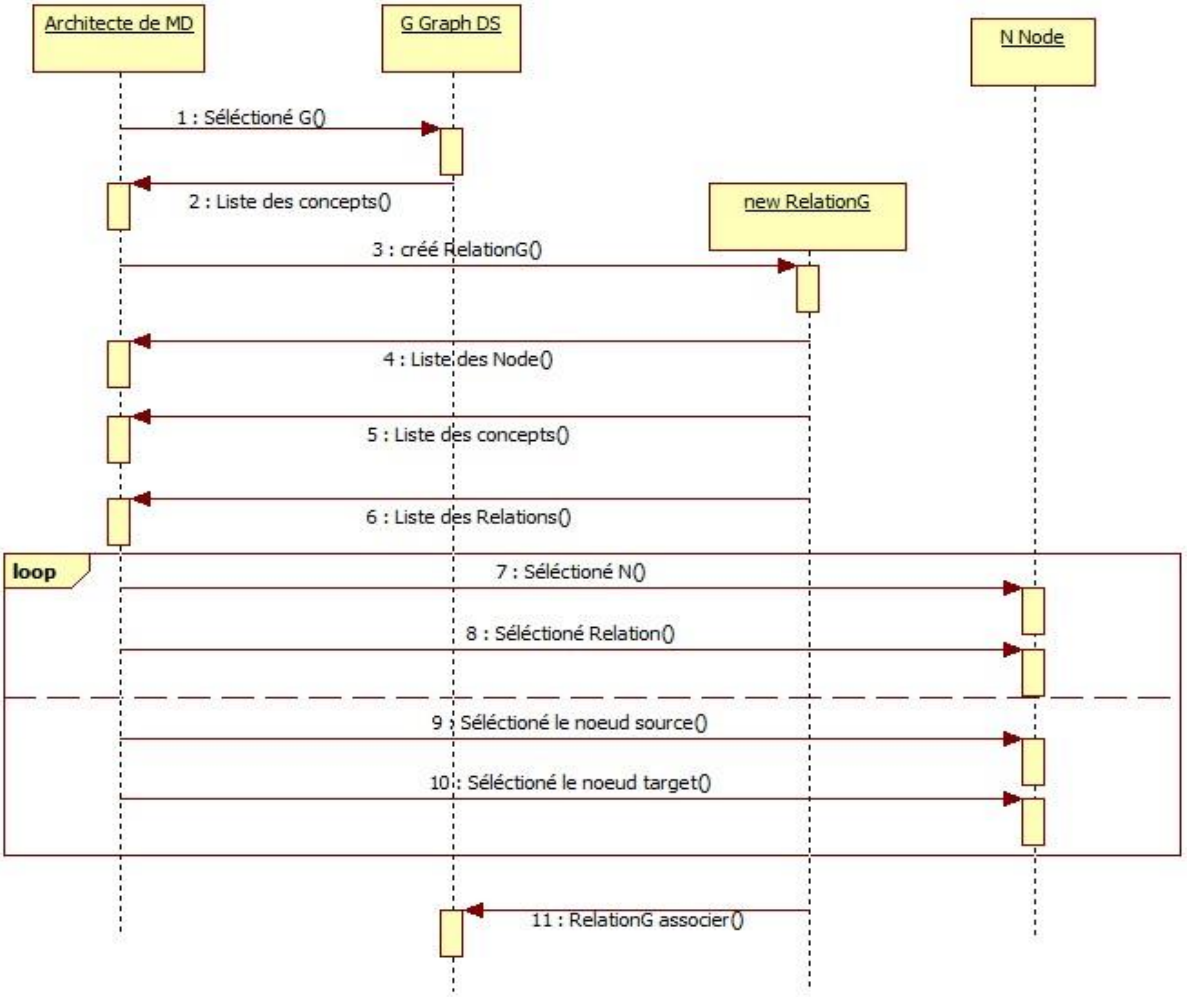


Figure 3.28 diagramme de séquence du cas d'utilisation ajouter gérer une relation (arcs) entre nœuds.

8. Cas d'utilisation *Ajouter et gérer un item de document* : la figure 3.29 montre le diagramme de séquence du cas d'utilisation ajouter et gérer un item de document.

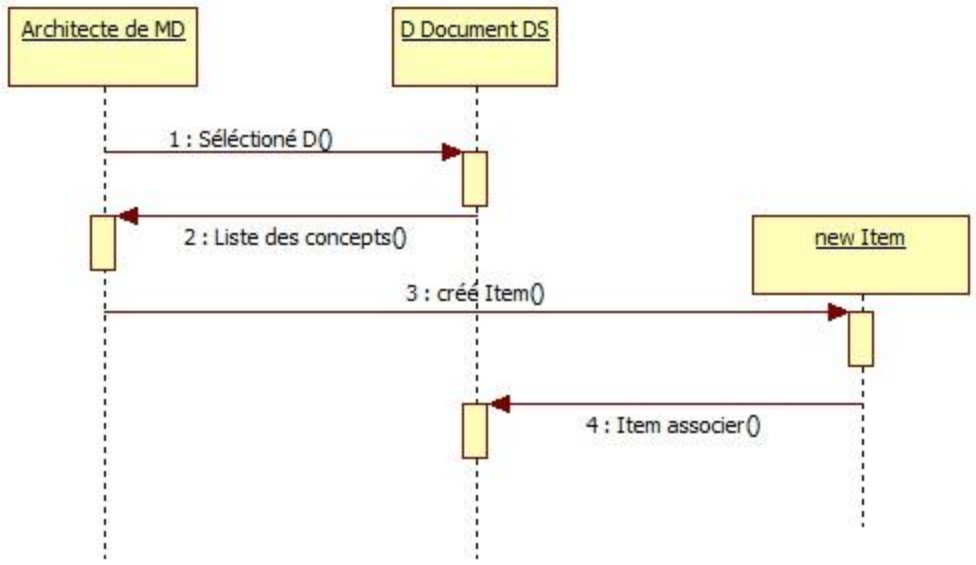


Figure 3.29 diagramme de séquence du cas d'utilisation ajouter et gérer un item de document.

9. Cas d'utilisation *Ajouter et gérer une relation entre items* : la figure 3.30 représente le diagramme de séquence du cas d'utilisation ajouter et gérer une relation entre items.

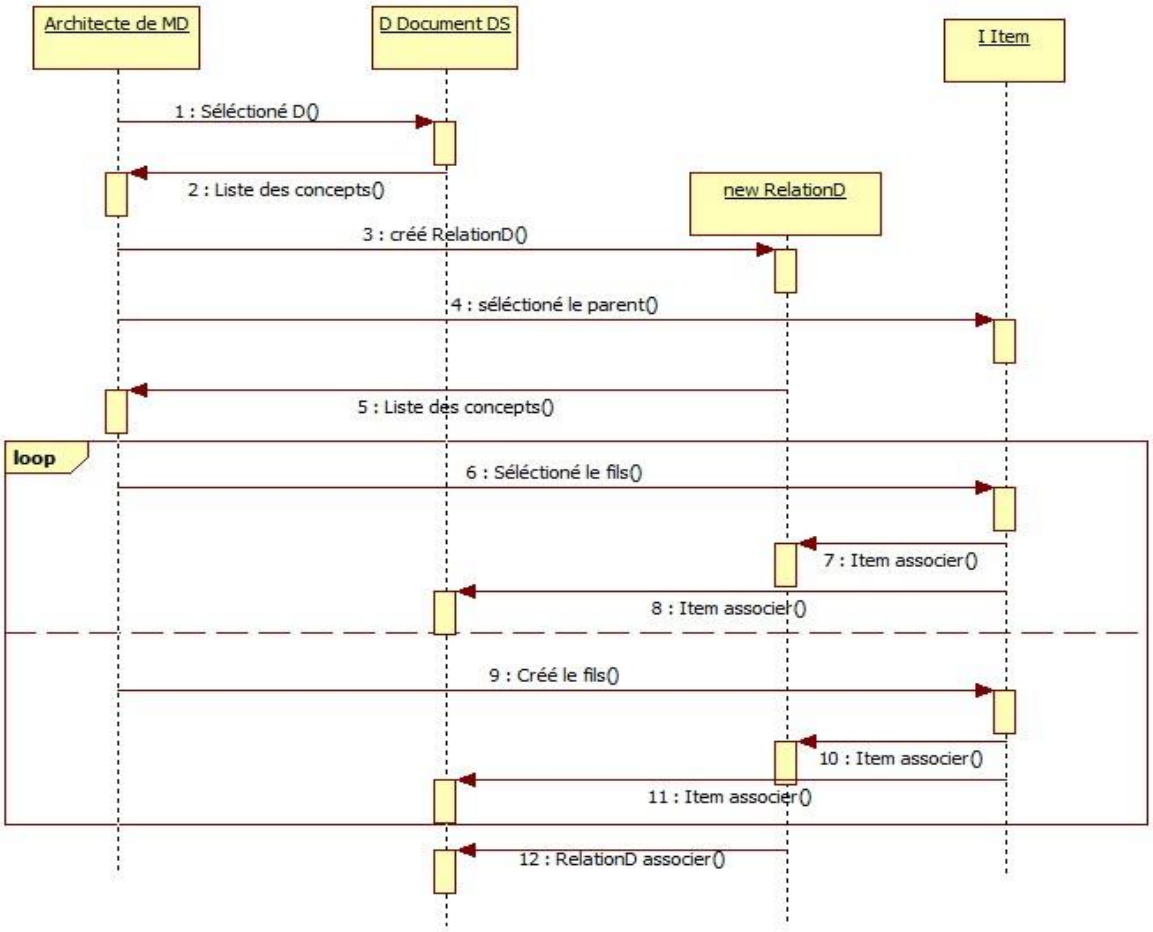


Figure 3.30 diagramme de séquence du cas d'utilisation ajouter et gérer une relation entre items.

3.12 Diagramme de déploiement

Le diagramme de déploiement est une vue statique du système qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux. La figure 3.31 représente le diagramme de déploiement de notre système.

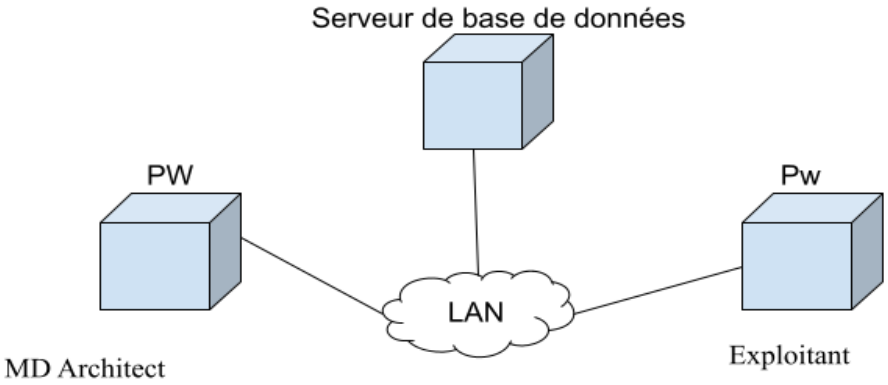


Figure 3.31 Diagramme de déploiement.

3.13 Identification des composants distribués

A ce niveau de conception, on s'intéresse aux composants métiers, issus du découpage en catégories. Puisque qu'une application reflète un regroupement de cas d'utilisation interdépendants, il en découle des classes regroupées en catégories de cas d'utilisation interdépendants. Par ailleurs, il est possible que certaines catégories soient partagées entre plusieurs applications. Dans notre système on trouve que la visualisation de schéma est partagée entre l'exploitant et l'architecte de métadonnées. La figure 3.32 représente le découpage de notre système en catégories.

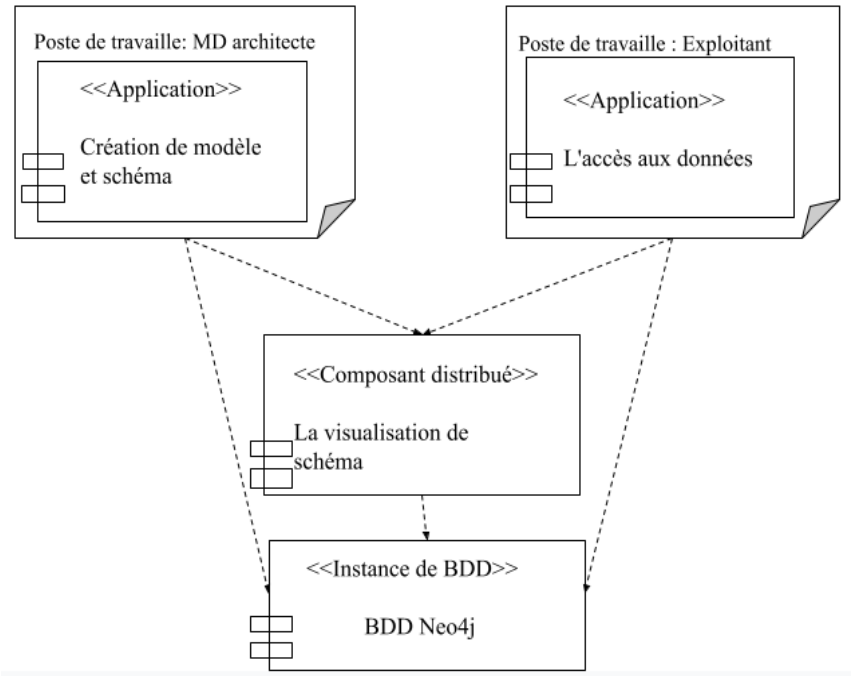


Figure 3.32 composant distribué de notre système

3.14 Enumération des interfaces utilisateurs (IHM)

Par interface, on signifie les interfaces de présentation des applications par opposition aux interfaces objet. La définition des interfaces objets peut être prévue dès l'étape de capture de besoins fonctionnels et étape d'analyse, mais c'est à ce niveau-là que les maquettes peuvent être dessinées et distribuées sur les applications. La définition des interfaces liées aux applications permet, plus tard, d'identifier les objets de la couche présentation. Les IHM peuvent être décrits textuellement dans un tableau comme illustré dans le tableau 3.1.

Vue IHM	**Description**

créer le modèle des métadonnées	Initier à créer un modèle. Ajouter et éditer un concept. Ajouter une liaison entre concepts
créer le schéma (instance du modèle)	Instancier un concept Instancier une liaison Editer un concept Visualiser le schéma
Accéder aux données	Visualiser le schéma Accéder aux données d'un concept

Tableau 3.1- Enumération d'interfaces homme-machine de notre système.

Dans ce tableau, nous présentons juste la vue IHM des méta-cas et celui de la visualisation et de l'accès. Les vues concrètes de chaque cas d'utilisation sont similaires.

3.15 Le passage de modèle objet au modèle graphe

Le passage du modèle objet au graphe suit un ensemble de règles illustrées dans le tableau suivant (tableau 3.2) :

Modèle objet	**Modèle graphe (espace modèle)**	**Modèle graphe (espace schéma)**
La classe Relationnelle_DS	Un nœud unique avec un nom «Relationnelle_DS»	Nœud avec un label « Relationnelle_DS » qui peut avoir

		plusieurs occurrences avec différents noms
La classe Table	Un nœud unique avec un nom « Table »	Nœud avec un label « Table » qui peut avoir plusieurs occurrences avec différents noms
La classe Colonne	Un nœud unique avec un nom « Column »	Nœud avec un label « Column » qui peut avoir plusieurs occurrences avec différents noms
La classe Graphe_DS	Un nœud unique avec un nom « GRAPH_DS »	Nœud avec un label « GRAPH_DS» qui peut avoir plusieurs occurrences avec différents noms
La classe RelationG	Un nœud unique avec un nom « RelationG»	Nœud avec un label « RelationG» qui peut avoir plusieurs occurrences avec différents noms
La classe Node	Un nœud unique avec un nom «Node»	Nœud avec un label «Node» qui peut avoir plusieurs occurrences avec différents noms
La classe Docuement_DS	Un nœud unique avec un nom «Document_DS»	Nœud avec un label «Document_DS» qui peut avoir plusieurs occurrences avec différents noms

La classe Item	Un nœud unique avec un nom «Item»	Nœud avec un label «Item» qui peut avoir plusieurs occurrences avec différents noms
La classe RelationD	Un nœud unique avec un nom «RelationD»	Nœud avec un label «RelationD» qui peut avoir plusieurs occurrences avec différents noms
La classe DataLake	Un nœud unique avec un nom «DL»	Nœud avec un label «DL» et un nom ”DataLake”
La classe Properties	Un nœud unique avec un nom «Properties»	Nœud avec un label «Properties » qui peut avoir plusieurs occurrences
Association de type composition	Une relation entre nœuds avec un nom «composedOf»	Une relation entre nœuds avec un label «composedOf»
Association entre la classe « properties » et les autres classe	Une relation à un types « has properties» entre le nœud « properties » et les autres nœuds	Une relation à un label de types « has properties» entre le nœud « properties » et les autres nœuds. La valeur de la propriété pour chaque noeud figure comme propriété de la relation (l’arc)
Association simple	Une relation de type «has»	Une relation de type « has»

Tableau 3.2 : passage de modèle objet au modèle graphe.

3.16 Conclusion

Dans ce chapitre, nous avons détaillé l’application du processus 2TUP allégée à notre travail, où nous avons présenté l’étape de capture des besoins fonctionnels en identifiant les acteurs, les

messages et nous avons présenté la modélisation du contexte, le diagramme de cas d'utilisation et les classes candidates de chaque cas. Ensuite, dans l'étape d'analyse, nous avons présenté le diagramme de classes global de notre système, et enfin dans les deux dernières étapes de conception du processus 2TUP (conception préliminaire et conception détaillée) nous avons présenté le diagramme de déploiement, l'identification des composants distribués, l'énumération des interfaces utilisateurs (IHM) et le passage de modèle objet au modèle graphe. Dans le chapitre suivant, nous présentons les détails de mise en œuvre de notre système

Chapitre IIII

4. Mise en œuvre

4.1 Introduction

Ce chapitre représente la dernière partie de ce travail ; il traite la phase d'implémentation de l'application. Nous commençons par la description de l'environnement utilisé pour développer l'application. Ensuite, nous présentons le jeu de données utilisé pour implémenter un lac de données et montrer l'accès à ses données. Finalement, nous présentons un aperçu de l'ensemble des fonctionnalités qu'offre notre application sous forme de prises d'écrans avec des descriptions.

4.2 Environnement de développement

L'environnement de développement et les différents outils utilisés sont les suivants :

4.2.1 Framework en front-end

On appelle framework front-end tout ensemble de classes, fonctions et utilitaires qui nous facilitent la création d'applications riches pour les navigateurs (et, de plus en plus, pour les mobiles). Un tel framework vise à nous isoler des différences techniques entre les navigateurs, et à nous éviter de réinventer la roue pour tous les besoins classiques de nos applications : gestion de l'interface utilisateur, des événements, du DOM, des formulaires, de l'évolution dans le temps des données manipulées par l'interface, etc.

Dans cet « espace » des frameworks front-end (ou simplement « frameworks front »), on trouve Angular, Ember, React, Vue.js et bien d'autres.

React (aussi appelé React.js ou ReactJS) : React est une bibliothèque JavaScript déclarative libre développée par Facebook depuis 2013[62], efficace et flexible pour construire des interfaces utilisateurs (UI). Elle permet de composer des UI complexes à partir de petits morceaux de code isolés appelées « composants ». [63]

React se concentre sur le cœur du problème : la gestion de l'interface utilisateur. Les autres briques applicatives, comme le routage côté client, le stockage des données, etc. sont laissées aux innombrables solutions complémentaires de son écosystème (par exemple, React-Router, Redux, Redux-Offline…)[9]

React a fait voler en éclat les dogmes établis du développement web, tels que la séparation stricte entre la structure (HTML), l'aspect (CSS) et le comportement (JS), classiquement écrits dans des fichiers bien distincts, pour revenir à la notion fondamentale de composant autonome, cohérent et complet. [9]

1. L'Encapsulation dans React [9] : Les composants React se comportent vis-à-vis du reste de l'application comme une boîte noire, avec une interface de programmation externe (une API) clairement définie. Ils contiennent en eux tout le nécessaire à leur bon fonctionnement : la structure, les styles, le comportement.
2. La composition dans React [9] : React nous encourage à structurer nos interfaces comme une arborescence de composants, et à rendre ces composants éminemment réutilisables. La majorité de nos composants sont eux-mêmes créées en combinant d'autres composants plus simples : c'est le principe fondamental de la composition, un outil primordial de structuration de code et d'application.
3. Le DOM virtuel [9] : React a inauguré la notion de DOM virtuel : React lui-même ne manipule pas directement le DOM du navigateur. Cela serait coûteux en performance et empêcherait de décliner cette approche sur d'autres supports tels que les applications mobiles natives, les terminaux textuels, les fichiers PDF, etc. À la place, React nous fait décrire un DOM virtuel, absolument distinct du DOM des navigateurs. Au moment venu il réconcilie ce DOM virtuel avec la couche de rendu réelle (par exemple, le DOM du navigateur, ou, si on est côté serveur, la production du texte HTML à renvoyer côté client), en prenant soin de minimiser le nombre d'opérations nécessaires.

Nous avons utilisé la bibliothèque de java script « React.JS » comme un langage de programmation, afin de développer le côté front-end (l'interface) de l'application.

4.2.2 Framework BackEnd

4.2.2.1 NodeJS [64]

Node.js est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau événementielles hautement concurrentes qui doivent pouvoir monter en charge. Elle utilise la machine virtuelle V8, la librairie libuv pour sa boucle d'évènements, et implémente sous licence MIT les spécification CommonJS. Il est également possible d'écrire NodeJS = Runtime Environment + JavaScript Library.

Parmi les modules natifs de Node.js, on retrouve *http* qui permet le développement de serveur HTTP. Il est donc possible de se passer de serveurs web tels que Nginx ou Apache lors du déploiement de sites et d'applications web développés avec Node.js.

Node.js est un environnement bas niveau permettant l'exécution de JavaScript côté serveur. Grâce à son fonctionnement non bloquant, il permet de concevoir des applications en réseau performantes, telles qu'un serveur web, une API ou un job CRON.

Node.js est utilisé pour faire des applications cross-plateform avec des framework comme Ionic pour les applications mobiles ou encore Electron pour les applications desktop. Les géants comme Discord ou encore Slack utilisent ce système.

Node.js est aussi beaucoup utilisé pour faire des serveurs de bot informatique. Certaines API Rest pour l'authentification sont faites avec Node.js.

4.2.2.2 Express [65]

Express est une infrastructure d'applications Web (framework) écrite en JavaScript et hébergée dans l'environnement d'exécution node.js, minimaliste et flexible qui fournit un ensemble de fonctionnalités robuste pour les applications Web et mobiles.

Coder des serveurs Web en Node.js pur est possible, mais long et laborieux. En effet, cela exige d'analyser manuellement chaque demande entrante. L'utilisation du framework Express simplifie ces tâches, en nous permettant de déployer nos API beaucoup plus rapidement.

Dans notre travail, nous avons utilisé Node.js et Express pour la création du serveur web qui permet d'interagir avec les bases de données.

4.2.3 Environnement de stockage

4.2.3.1 PostgreSQL

PostgreSQL s'appelait à l'origine POSTGRES, faisant référence à ses origines en tant que successeur de la base de données Ingres développée à l'Université de Californie à Berkeley. En 1996, le projet a été renommé PostgreSQL pour refléter sa prise en charge de SQL [66]. PostgreSQL est un système de gestion de bases de données relationnel et objet (SGBDRO) robuste et puissant, aux fonctionnalités riches et avancées, capable de manipuler en toute fiabilité de gros volumes de données, mêmes dans des situations critiques.[67]

Caractéristique de PostgreSQL

Ce SGBDRO utilise des types de données modernes, dits composés ou enrichis suivant les terminologies utilisées dans le vocable informatique usuel. Ceci signifie que PostgreSQL peut stocker plus de types de données que les types simples traditionnels entiers, caractères, etc. L'utilisateur peut créer des types, des fonctions, utiliser l'héritage de type, etc.

PostgreSQL est plus avancé que ses concurrents dans la conformité aux standards SQL. Il est largement reconnu pour son comportement stable, proche de *Oracle*, mais aussi pour ses possibilités de programmation étendues, directement dans le moteur de la base de données, via PL/pgSQL. Le traitement interne des données peut aussi être couplé à d'autres modules externes compilés dans d'autres langages [66]. PostgreSQL prend en charge SQL et JSON pour les requêtes relationnelles et non relationnelles pour l'extensibilité et la conformité SQL. PostgreSQL prend en charge les types de données avancés et les fonctionnalités d'optimisation des performances, qui ne sont disponibles que dans les bases de données commerciales coûteuses, comme Oracle et SQL Server. [66]

La figure 4.1 représente l'environnement de travail de PostgreSQL.

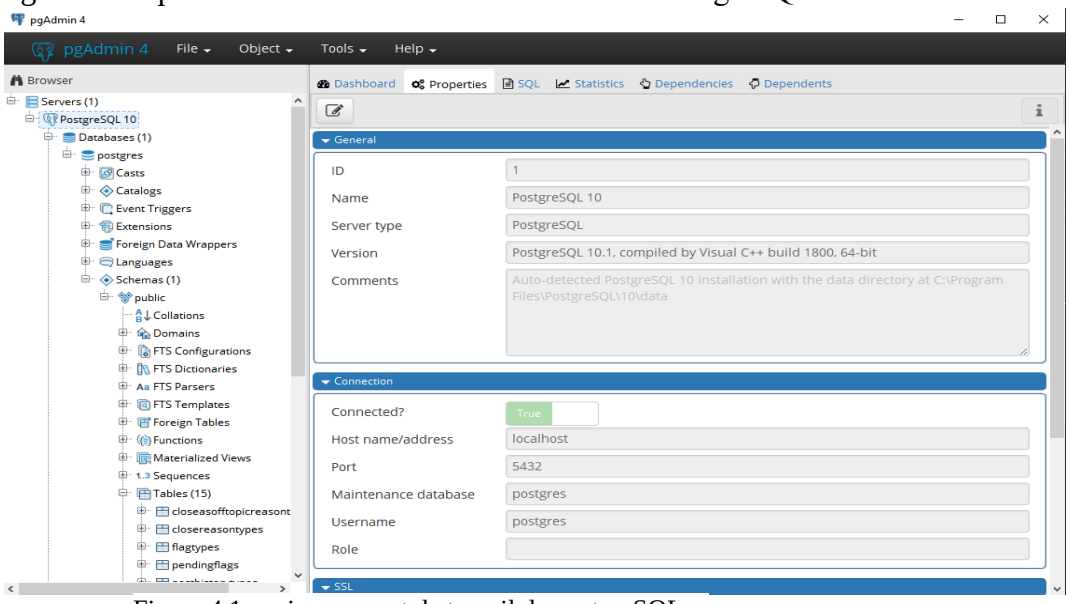


Figure 4.1 environnement de travail de postgreSQL.

Nous avons utilisé PostgreSQL version 10.1, pour stocker les données relationnelles, c'est-à dire les données des sources relationnelles d'un lac de données.

4.2.3.2 Neo4j

Neo4j est un SGBD graphe natif, qui permet de représenter les données en tant que nœuds reliés par un ensemble d'arcs, ces objets possédant leurs propres propriétés. Les propriétés sont constituées d'un couple de clé-valeurs de type simple tel que chaînes de caractères ou numérique ;

celles-ci peuvent être indexées. La modélisation est très proche des concepts métier, il n'est pas nécessaire d'utiliser de clés dans Neo4j, car les relations ont une existence propre. L'absence de modélisation rigide rend Neo4j bien adapté à la gestion de données changeantes et de schémas évoluant fréquemment.

La base de données Neo4j est construite pour être extrêmement performante pour traiter les liens entre nœuds. Ces performances sont dues au fait que Neo4j pré-calcule les jointures au moment de l'écriture des données, comparativement aux bases de données relationnelles qui calculent les jointures à la lecture en faisant appel aux Index et à la logique de clés. Ce qui fait de Neo4j une technologie adaptée à de larges ensembles de données connectées.

Les bases de données de graphes sont des outils puissants pour répondre à des requêtes faisant intervenir des relations entre objets. La recherche du plus court chemin entre deux points du graphe permet par exemple de mettre en place des profils utilisant liens sociaux, géographie et analyse d'impact. [68]

La figure 4.2 représente l'interface Neo4j version 1.4.3 de Neo4j Desktop et la version 4.2.1 pour la base de données utilisées dans notre travail. Notons que nous avons fait une utilisation double de Neo4j dans notre travail :

- Pour représenter les métadonnées (voir chapitre 2)
- Pour représenter et stocker les sources graphes d'un lac de données.

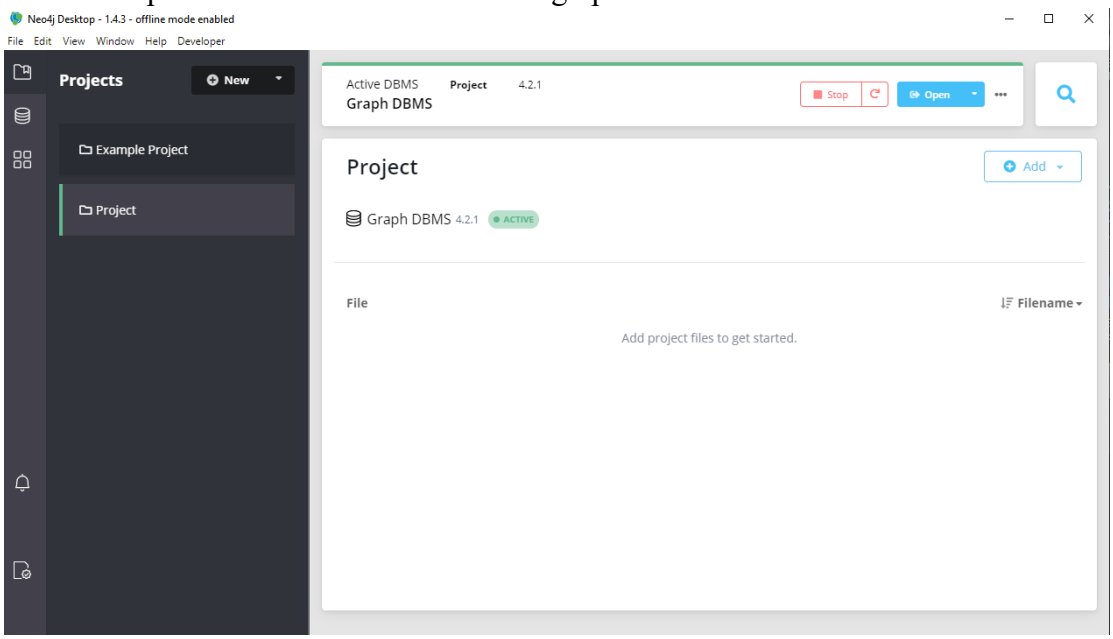


Figure 4.2 Interface Neo4j version 1.4.3 Desktop

4.2.3.3 MongoDB [69]

MongoDB est un système de base de données orienté document multiplateforme disponible à la source. Classé comme système de base de données NoSQL, MongoDB utilise des documents de type JSON avec des schémas facultatifs. MongoDB est développé par MongoDB Inc. et licencié sous la licence publique côté serveur (SSPL). MongoDB permet de manipuler des objets structurés au format BSON (JSON binaire), sans schéma prédéterminé. En d'autres termes, des clés peuvent être ajoutées à tout moment « à la volée », sans reconfiguration de la base.

Les données prennent la forme de *documents* enregistrés eux-mêmes dans des *collections*, une collection contenant un nombre quelconque de documents. Les collections sont comparables aux tables, et les documents aux enregistrements des bases de données relationnelles. Contrairement aux bases de données relationnelles, les champs d'un enregistrement sont libres et peuvent être différents d'un enregistrement à un autre au sein d'une même collection. Le seul champ commun et obligatoire est le champ de clé principale ("id"). Par ailleurs, MongoDB ne permet ni les requêtes très complexes standardisées, ni les *JOIN*, mais permet de programmer des requêtes spécifiques en JavaScript.

La figure 4.3 représente l'interface NoSQLBooster for MongoDB version 5.2.12 et la version 4.0.23 for MongoDB pour la base de données document utilisées dans notre travail.

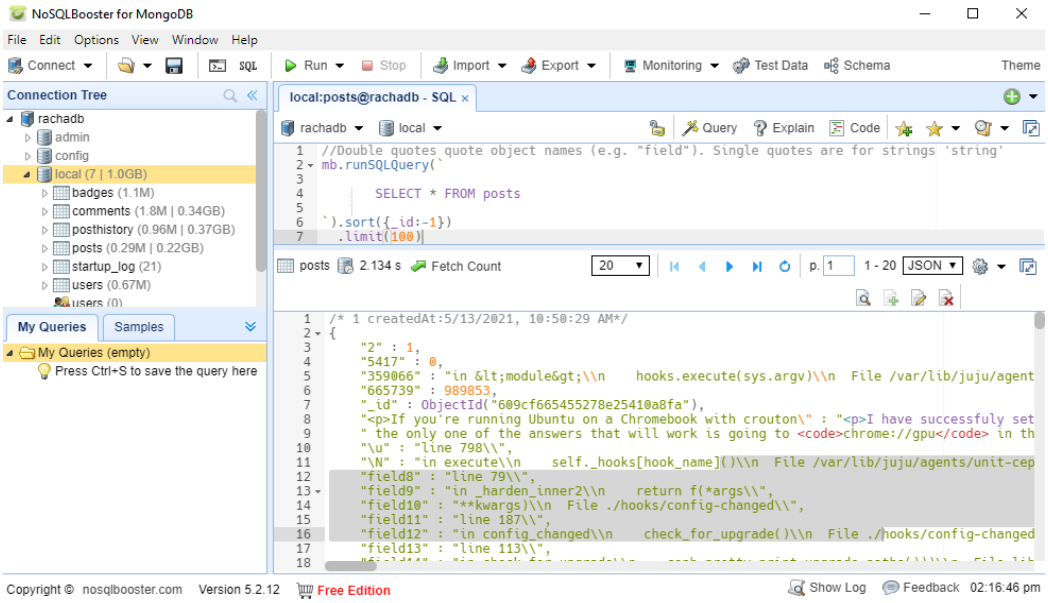


Figure 4.3 Interface NoSQLBooster for MongoDB version 5.2.12

On peut résumer l'environnement de développement dans le schéma illustré dans la figure 4.4

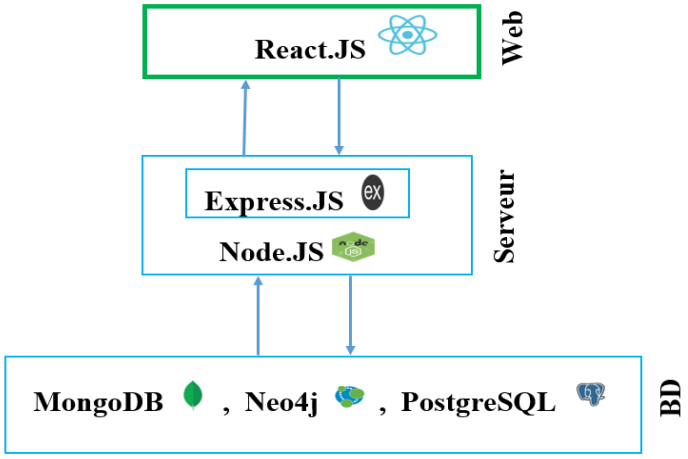


Figure 4.4 Environnement de développement de l'application

4.3 Jeux de données

Dans notre travail, nous avons implémenté un lac de données pour le réseau de questions/réponses StackExchange. Le lac de données est diversifié dans le sens qu'il contient trois sources de données NoSQL, à savoir une source relationnelle, une source documents et un graphe. Le lac de données est également possible à étendre à d'autres sources des trois types cités, ou bien à de nouveaux types de sources, dans la mesure où les données correspondantes sont disponibles. Les trois sources actuelles du lac de données sont décrites comme suit :

1- **Source relationnelle :** pour stocker les données relationnelles, nous avons utilisé PostgreSQL. Notre base de données relationnelle contient un schéma relationnel nommé Public composé de 15 table, avec les nombres suivants d'enregistrements de chaque table (tableau 4.1).

Table	**Enregistrements**
closeasofftopicreasontypes	12
closereasontypes	13
flagtypes	3
pendingflags	337
posthistorytypes	33

postnoticetypes	15
posts	726299
posttags	47381
posttypes	8
reviewrejectionreasons	19
reviewtaskresulttypes	23
reviewtaskstates	3
reviewtasktypes	9
users	669235
votetypes	14

Tableau 4.1 Nombre de tables et enregistrement de schéma relationnel.

- Etapes d'alimentation de la base de données relationnelle

La 1ère étape est la création de schéma « public » sous PostgreSQL, la 2ème est la création des tables et la dernière c'est l'importation des données depuis les fichiers .csv exportés depuis la base de données en ligne de StackExchange. La création du schéma relationnel et importation vers PostgresSQL est assurée par l'exécution des requêtes :

-Requête de la création de schéma « public » sous PostgreSQL :

CREATE SCHEMA "public";

-Exemple de requête de la création de la table Posts sous PostgreSQL :

```
CREATE TABLE "public".posts (
        id              integer  NOT NULL ,
        posttypeid      smallint  NOT NULL ,
        acceptedanswerid    integer   ,
        parentid        integer   ,
```

```
    creationdate        timestamp  NOT NULL ,
    deletiondate        timestamp   ,
    score               integer   ,
    viewcount           integer   ,
    owneruserid         integer   ,
    lasteditoruserid    integer   ,
    lasteditdate        timestamp   ,
    lastactivitydate    timestamp   ,
    answercount         integer   ,
    commentcount        integer   ,
    favoritecount       integer   ,
    closeddate          timestamp   ,
    communityowneddate  timestamp   ,
  CONSTRAINT posts_pkey PRIMARY KEY ( id ),
 CONSTRAINT fk1_posts_posts FOREIGN KEY ( acceptedanswerid ) REFERENCES "public".posts( id),
  CONSTRAINT fk1_posts_users FOREIGN KEY ( lasteditoruserid ) REFERENCES "public".users( id ) ,
  CONSTRAINT fk_posts_users FOREIGN KEY ( owneruserid ) REFERENCES "public".users( id));
```

-Exemple de requête pour l'importation de données depuis le fichier Posts.cv :

\COPY Posts from 'D:\CSV-20210429T101435Z-001\CSV\Posts.csv' DELIMITER ',' ;

2- **MongoDB** : nous avos utilisé MongoDB pour stocker les documents. L'alimentation de la base passe par deux étapes : la 1ère étape est la création des collections et la 2ème consiste, pour chaque collection, à importer les données (documents) de fichier .csv correspondant à partir NoSQLBooster for MongoDB. En tout, nous avons obtenu dix (10) collections dans notre source documents du lac de données.

3- **Neo4j** : nous avons utilisé Neo4j dans notre travail pour créer trois graphes :

-le1er graphe est pour stocker et manipuler le modèle et le schéma de métadonnées créé par l'architecte de métadonnées,

-le 2ème graphe est pour stocker le graphe utilisé par l'exploitant et qui est synchronisé avec le graphe de l'architecte des métadonnées,

-le 3ème graphe est utilisé pour stocker les données StackExchange importées depuis les fichiers .csv, en utilisant les requêtes Cypher :

```
:auto USING PERIODIC COMMIT
LOAD CSV FROM 'file:///users.csv' AS row
create(:Users {
id: toInteger(row[0]),
creationdate: row[1],
displayname: row[2],
accountid: toInteger(row[3])
})
```

4.4 Présentation de l'application

L'application développée est un système basé sur les graphes qui permet de gérer les méta-données dans un lac de données composé de sources NoSQL (graphes, documents, et relationelles) et extensible à d'autres types de sources. Le système permet aux utilisateurs d'ajouter des métadonnées concernant les sources du data lake et d'accéder aux données.

L'application développée est composée de trois espaces (schéma, modèle et visualisation) en passant par une étape d'authentification par login/password. Le schéma d'application est présenté dans la figure 4.5.

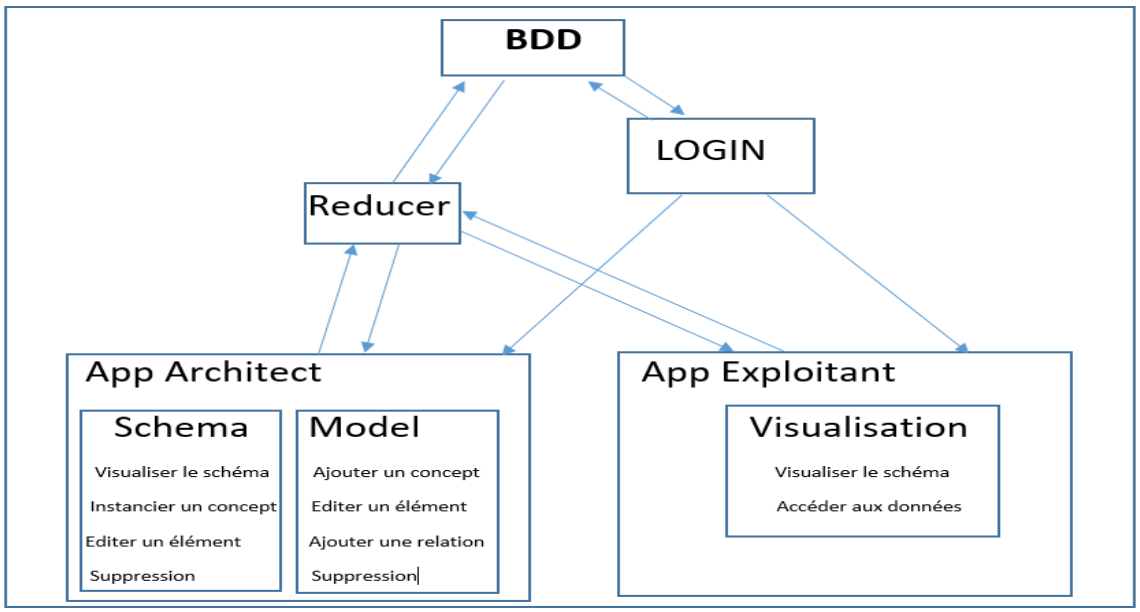


Figure 4.5 Schéma de l'application développé.

Description d'application

1. **Page de connexion :** la première page qui s'affiche après l'exécution de projet est celle présentée en figure 4.6. L'utilisateur doit se connecter en mentionnant son email et password. Si ces derniers sont invalides (n'existent pas dans la BD), la connexion est refusée (voir figure 4.7), sinon, le système retourne le type de l'utilisateur (Architect ou bien Exploitant), et selon son type, l'application affiche une page d'accueil spécifique.

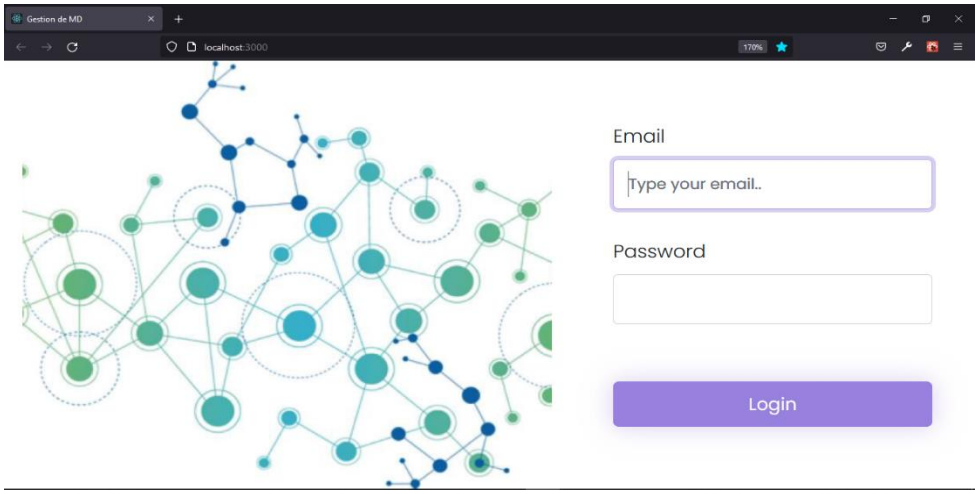


Figure 4.6 Page de connexion (LOGIN)

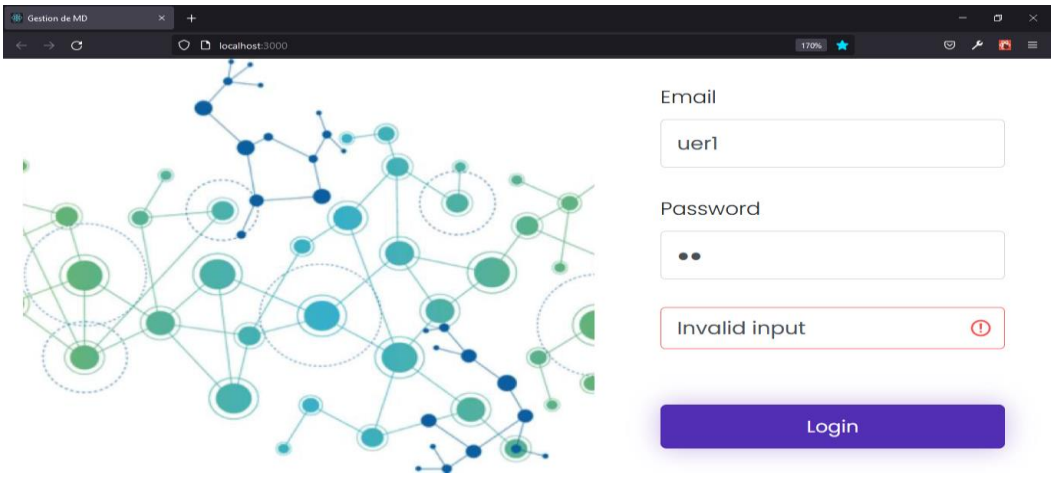


Figure 4.7 Connexion refusée

2. Espace d'Architecte de MD

a. **Page d'accueil pour l'Architecte de métadonnées :** la fenêtre principale de l'architecte de MD et celle qui lui permet de visualiser et manipuler le schéma. Cette fenêtre est présentée dans la figure 4.8.

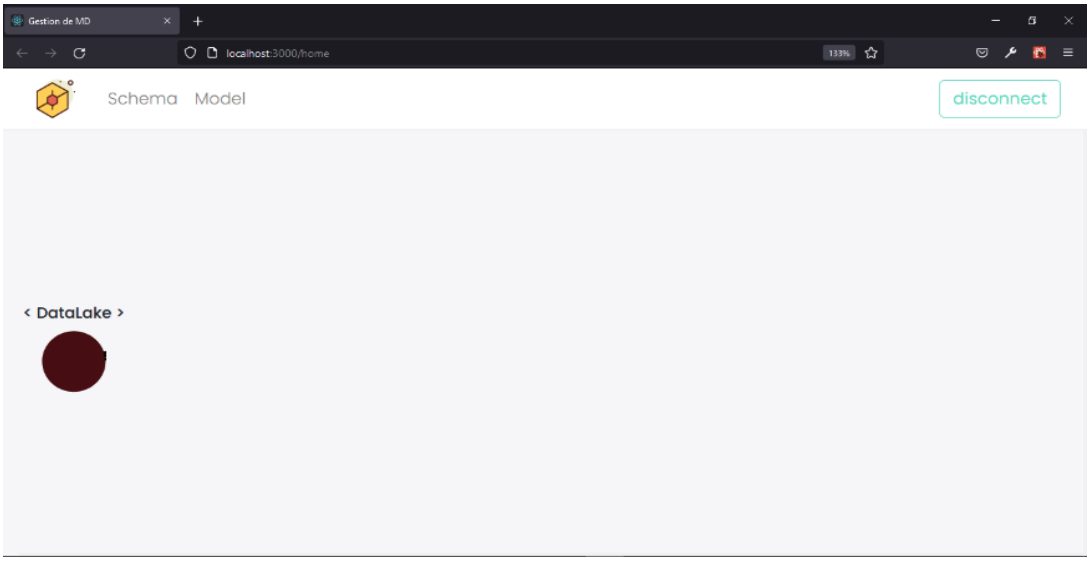


Figure 4.8 Page d'accueil pour l'architecte de MD.

Cette fenêtre est composée de :

1- La barre de navigation (en haut de la page en blanc) : elle contient de gauche à droite un logo, deux liens (Schema qui représente la fenêtre illustrée dans la figure 4.8, et Model présenté dans les sections suivante) et à droite, un bouton « disconnect » qui permet à l'utilisateur de se déconnecter de l'application.

2- Une aire en bas (en gris clair) : c'est dans cette surface où l'architecte peut construire son graphe de métadonnées en respectant le modèle (graphe des concepts représenté dans la page Model). La construction de schéma est détaillée dans les sections suivantes.

b. **Espace Model de l'architecte :** la figure 4.9 illustre l'espace model de l'architecte avec un seul concept qu'est le DL. Cette fenêtre lui permet de construire le graphe des concepts.

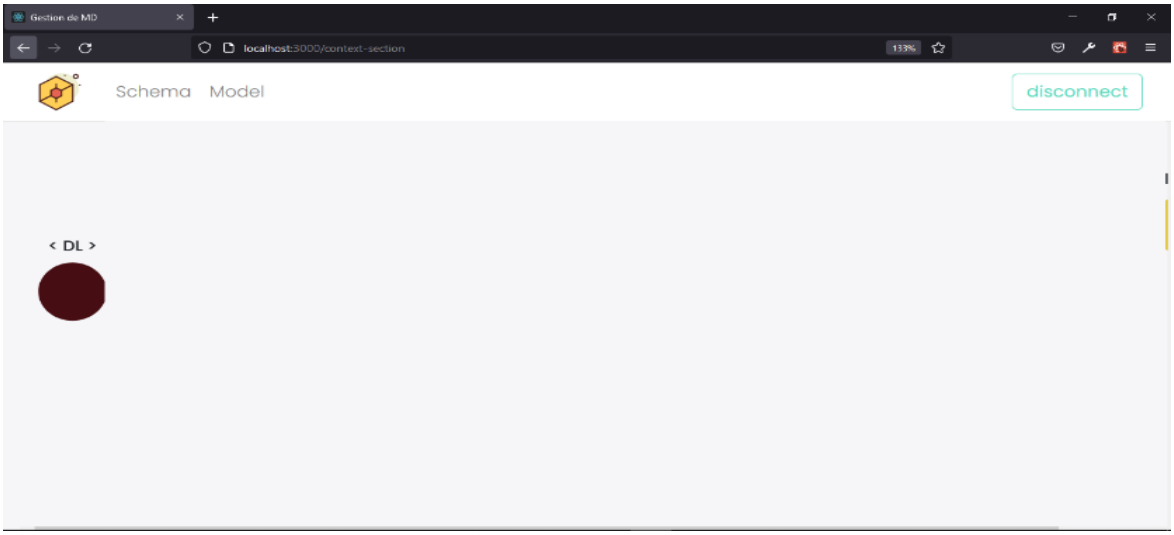


Figure 4.9 espace Model de l'Architect (la deuxième fenêtre de l'Architect)

c. **Fonctionnalités de l'architecte de MD dans l'application**

Avant la construction de schéma de métadonnées, l'architecte de MD doit avant tout ajouter des concepts donc il faut construire le graphe des concepts (le modèle) au moins un concept.

L'ajout d'un concept : après que l'architecte se soit connecté, bascule de l'espace Schema à l'espace Model en cliquant sur Model. L'architecte peut alors ajouter de nouveaux concepts selon les étapes suivantes (exemple d'ajout de concept nommé Rel_DS) :

Etape 1 Appel de menu : la figure 4.10 représente le menu pour ajouter un nouvel élément**.**

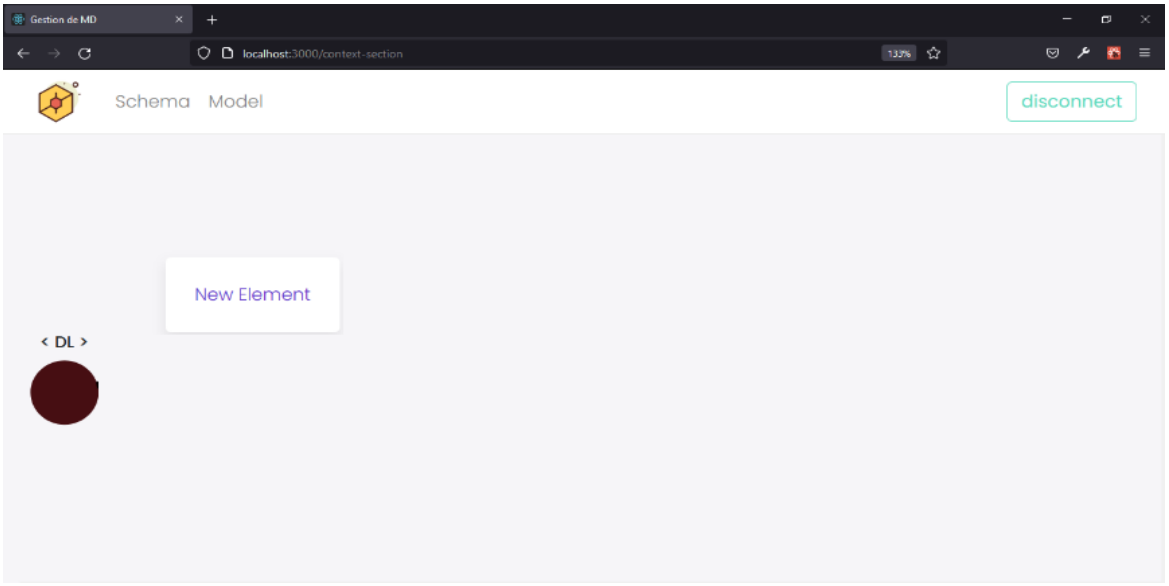


Figure 4.10 Menu pour ajouter un nouvel élément.

Etape 2 Ajout d'élément : en sélectionnant « New Element » un cercle en jaune s'affiche avec un texte (unnamed) en haut c'est le nom actuel du concept. (Voir figure 4.11)

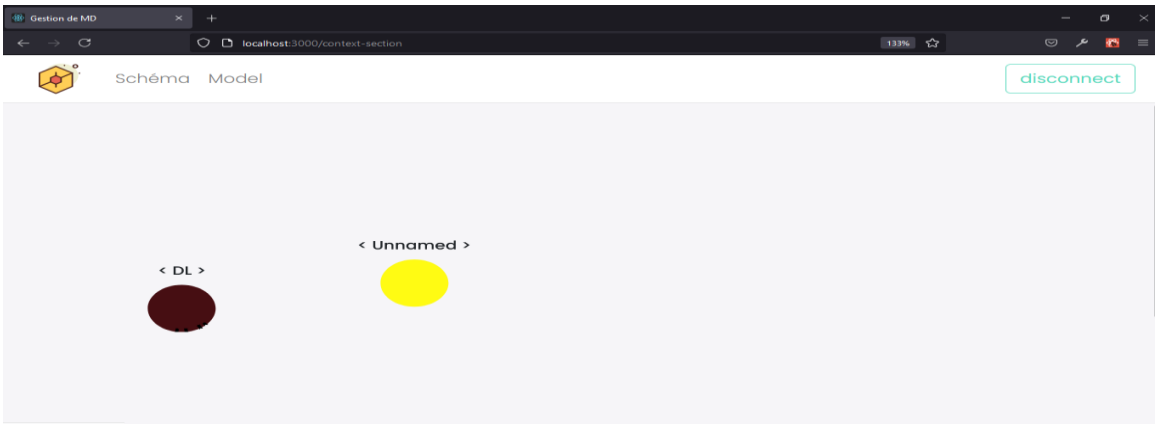


Figure 4.11 nouvel élément

pe 3 Modification de nom de concept : sélectioné «Edit Data» de menu illustrer dans la figure 4.12 pour afficher le component Model de React « Edit Element » (voir figure 4.13) qui sert à modifier le nom de cercle dans notre cas son nouveau nom est « Rel_DS » et on le stocke dans la base de donnée Neo4j.

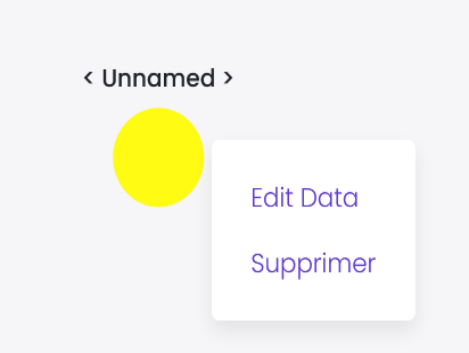


Figure 4.12 capture d'écran du menu affiché après le clic droit de la souris sur un élément dans la section Model

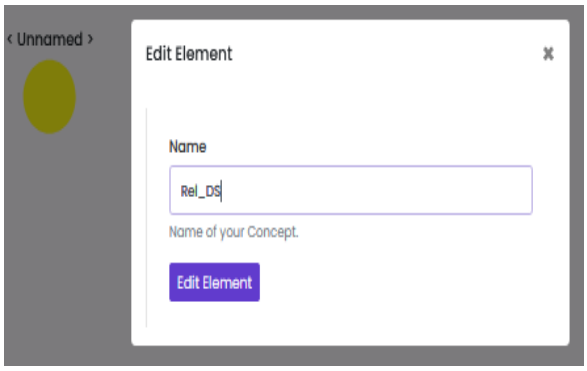


Figure 4.13 le modal affiché après la sélection de «Edit Element »

Etape 4 Ajout d'une relation : la figure 4.14 montre le modal «Add constrainte» qui s'affiche après l'ajout d'une relation entre deux concepts (dans notre cas entre DL et Rel_DS), pour reseigner un nom et des contraintes à cette relation.

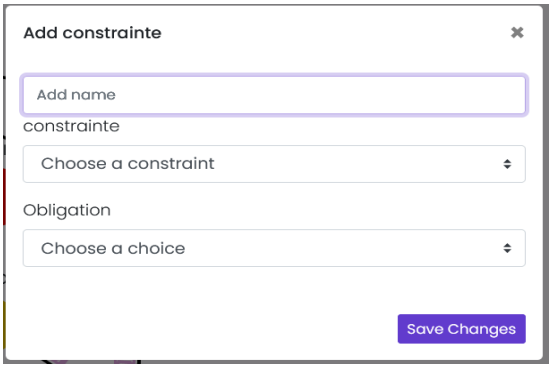


Figure 4.14 le modal affiché après qu'on relie deux concepts avec une flèche (relation)

Explication des différents composants de modal présentés dans la figure 4.14 :

Le 1er input sert à ajouter le nom de la relation dans notre cas « has »

Le 2ème input est une liste déroulante pour sélectionner une contrainte de la relation. On a trois possibilités (voir figure 4.15):

1- [1,1] qui veut dire que cette relation doit être instanciée une seule fois entre deux concepts qui portent le nom DL et Rel_DS.
2- [1, *] qui veut dire que cette relation doit être instanciée au moins une seule fois.
3- [0, *] : la relation à de 0 à plusieurs occurrences.

Le 3ème input est une autre liste déroulante pour spécifier si la relation est obligatoire ou non, la valeur est soit *true* ou *false* (voir figure 4.16).

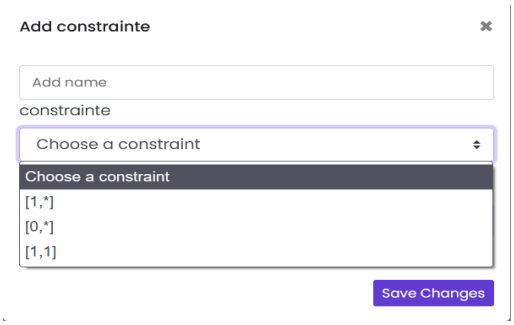


Figure 4.15 liste déroulante des contraintes possibles.

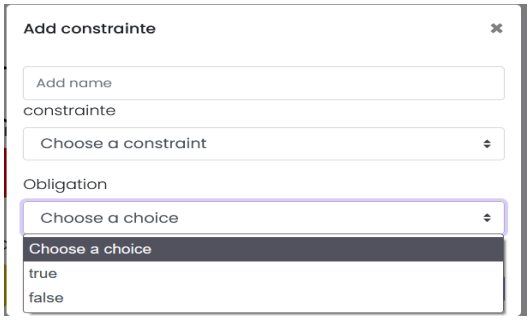


Figure 4.16 liste déroulant des choix possibles pour la propriété Obligation de relation.

Dans notre exemple, nous avons fait les choix suivantes : « has » comme un nom de relation, [0,*] comme une contrainte et « true » pour obligation.

Le bouton « Save change » sert à sauvegarder et créer la relation dans la base de données Neo4j.

A la fin de ces étapes, nous obtenons le résultat en figure 4.17.

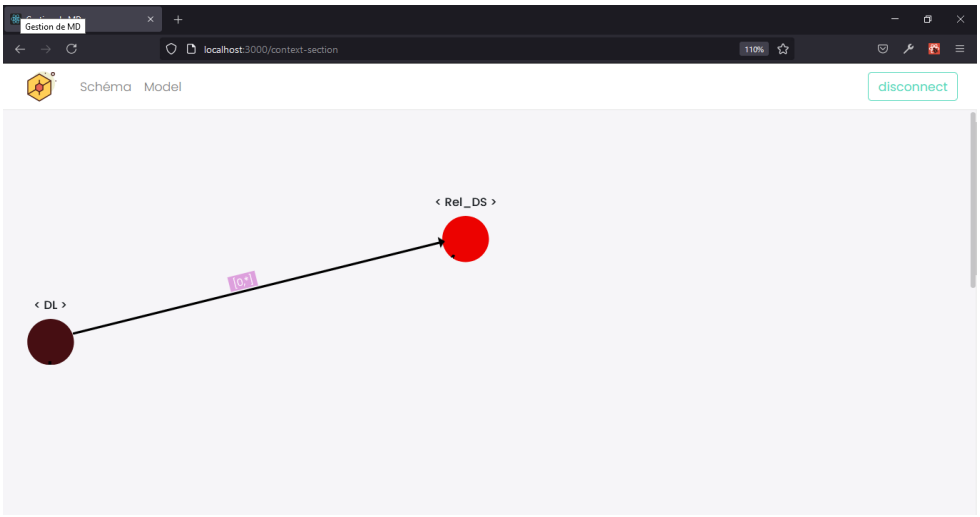


Figure 4.17 résultat d'ajout d'une relation

En suivant les mêmes étapes, on peut ajouter d'autres concepts en spécifiant chaque chemin avec une couleur différente (la couleur est donnée aléatoirement aux concepts liés avec « DL » et tout le chemin aura la même couleur). La figure 4.18 montre un modèle avec différents concepts.

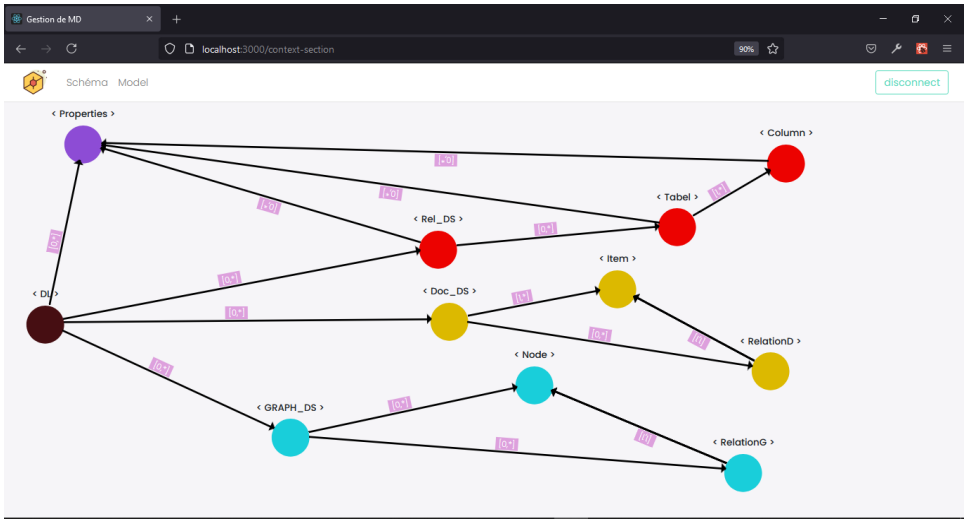


Figure 4.18 Modèle avec différent concept.

Les différentes étapes pour instancier un concept : On prend l'exemple du concept « Rel_DS » déjà créé (voir la section précédente). Pour ajouter une instance (instancier un concept), l'architecte doit basculer de la page Model à la page Schéma. Dans ce qui suit, nous expliquons les différentes étapes pour instancier le concept Rel_DS. Autrement dit, pour associer un concept de type Rel_DS au concept de type DL :

Etape 1 : il faut d'abord choisir le nœud source (le nœud DataLake dans notre cas), un panneau s'affiche à droite de la page (voir la figure 4.19).

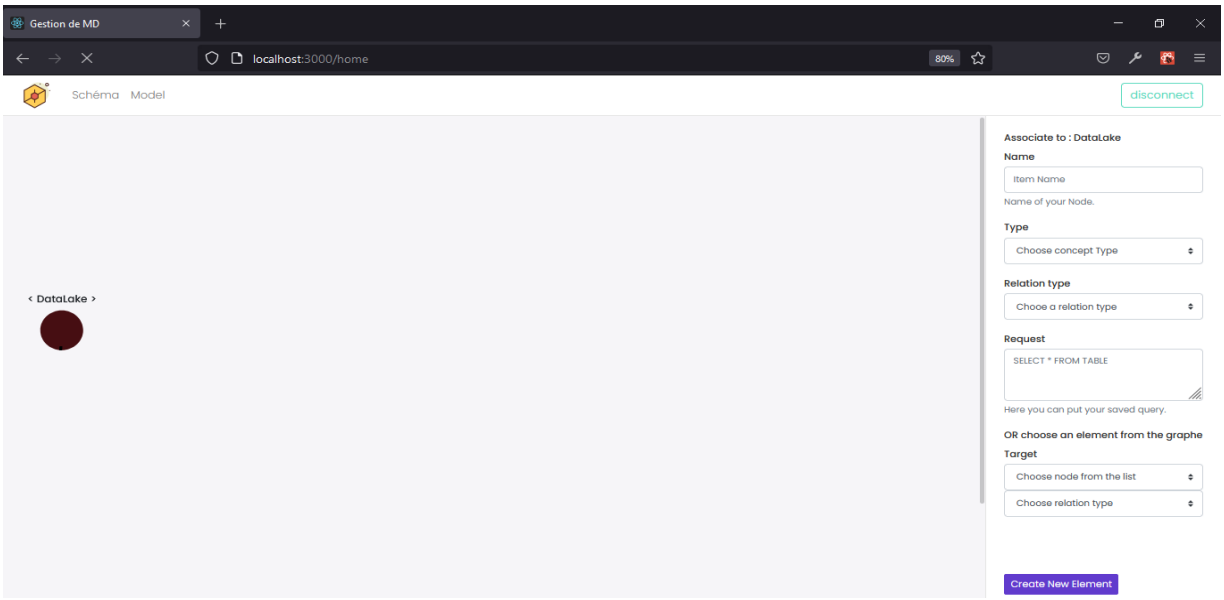


Figure 4.19 espace schéma avec le panneau à droit qui s'affiche après le double clic sur un élément

Ce panneau contient de haut en bas :

-un label qui affiche le nom de nœud source « Associate to : DataLake ».

-un Input pour donner un nom au nouveau nœud.

- une liste déroulant pour choisir le type de nouveau nœud, au moment de la sélection de l'élément source, une requête Cypher s'exécute pour avoir les types de concepts liés avec le concept DL pour les afficher dans cette liste. Dans notre exemple on peut trouver les types : GRAPH_DS, Doc_DS et Rel_DS (voir la figure 4.20).

-une autre liste déroulante pour choisir le type de relation pour ces deux concepts et de la même manière les types de relation son récupérés de la base de donnée à travers une requête Cypher.

-un textArea spécialement pour stocker une requête dans le nouveau nœud, s'il peut en avoir une.

Pour les éléments qui restent : si par exemple le nœud qu'on veut associer au DataLake existe déjà dans le graphe, on le choisit directement, sans le recréer une deuxième fois. On trouve en bas du panneau une liste déroulante pour afficher le nom des éléments qui existe déjà dans le graphe que l'on peut les associer au DataLake. En fait, pas tous les éléments existant dans le graphe peuvent être associés au nœud DataLake, mais seulement les nœuds qui sont de type GRAPH_DS, Doc_DS et Rel_DS , et cela dépend du modèle et ses contraintes.

Etape2 : après le remplissage des inputs et le choix des élément List déroulante ou bien le choix d'un élément dans le graphe, le bouton «Create» permet soit d'associer l'élément source avec un élément qui existe dans le graphe, c'est-à-dire de créer une relation entre eux ou bien créer un nouvel élément et le relier avec l'élément déja sélectionné.

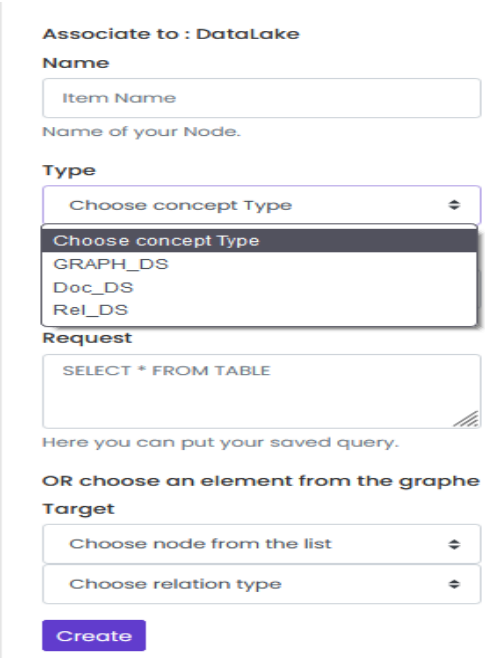


Figure 4.20 les types de concept possible qu'on peut l'associer à l'élément « DatLake »

Dans notre exemple, nous avons nommé le nœud « public », avec un type « Rel_DS », un nom de relation « has» ce nœud n'a pas de requête par ce qu'il s'agit de nœud qui représente un schéma relationnel. La figure 21 montre le résultat de ces étapes.

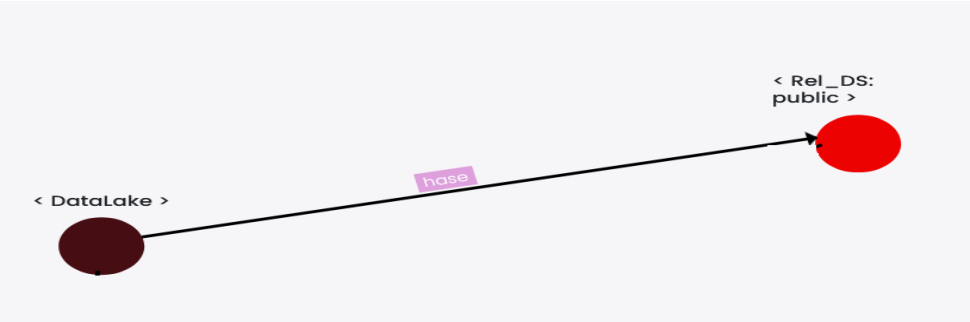


Figure 4.21 résultat de l'instanciation de concept « Rel_DS »

Association des propriétés de métadonnées à un élément :

A travers l'application et dans l'espace Schema, l'architecte peut associer à n'importe quel élément une propriété de métadonnées. Par exemple si on veut associer une propriété de métadonnées au nœud « Table : posts », l'architecte suit les étapes suivantes :

1.Afficher le menu présenté dans la figure 4.22

Figure 4.22 menu affiché après le choix d'un élément dans l'espace schéma.

2.Sélectionner le dernier choix « MD Properties » le panneau à droite s'affiche une autre fois mais avec un différent formulaire (voir figure 4.23)

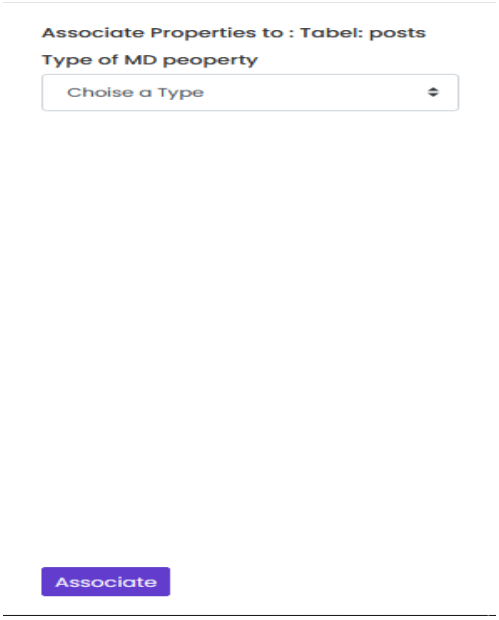


Figure 4.23 panneau affiché après la sélection de MD Properties de menu.

A cette étape, l'architecte doit choisir un type de propriété de métadonnées existant dans la liste déroulante (Technical , Business et Operational).

Après le choix du type de propriété, la liste des noms des propriétés de MD du type choisi s'affiche (dans notre exemple Business) selon celles qui existent dans la base de données Neo4j. Si la liste est vide ou bien la propriété n'existe pas, on peut l'ajouter en sélectionnant le choix « Add new properties » (voir figure 4.24)

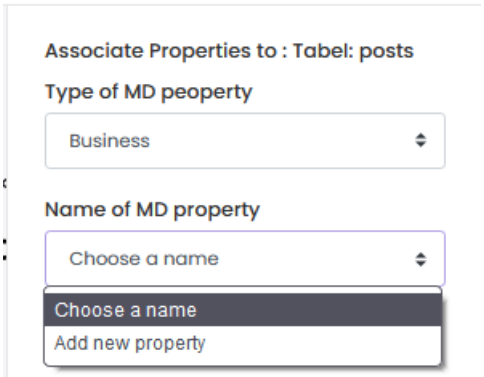


Figure 4.24 Liste des noms de propriétés existante dans la base.

Ce choix permet d'afficher un autre formulaire dans le panneau qui contient le type de propriété de metadonnées, un input pour le nom et un autre input pour la valeur (figure 4.25)

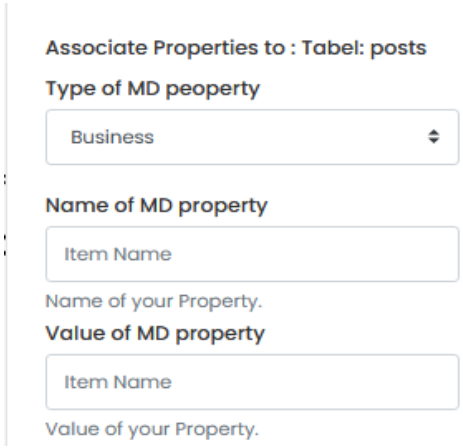


Figure 4.25 panneau pour ajouter une nouvelle propriété de métadonnées.

L'architecte peut aussi à travers l'application modifier le nom et/ou la requête stockée d'un élément, en sélectionnant le choix « Edit element » de menus déjà présentés dans la figure 4.22, cela conduit à l'apparition de même panneau toujours à droite avec un autre formulaire qui contient un input pour le nom, textArea pour la requête et le bouton « Edit Element » (voir la figure 4.26).

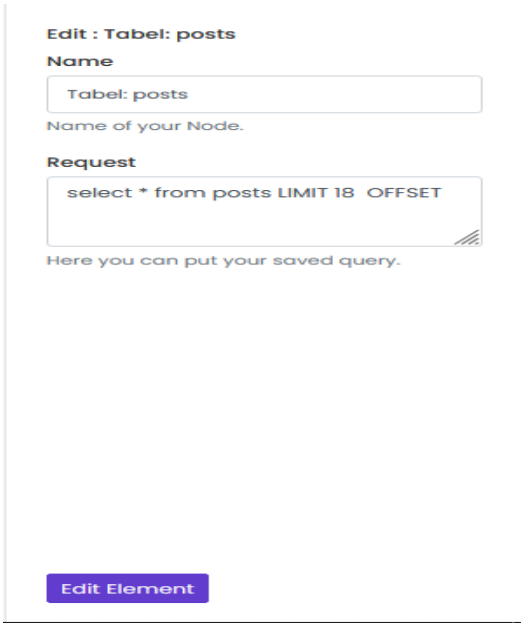


Figure 4.26 panneau qui sert à modifier un élément.

L'architecte peut également effectuer d'autres opération à travers l'application, comme la suppression d'un nœud (ou un chemin), l'édition d'une relation...

3.Espace d'Exploitant

Page d'accueil pour l'exploitant : contrairement à l'architecte de MD, l'exploitant a un seul espace qui lui permet de visualiser le schéma de DL et d'accéder à travers l'application aux données. Son espace est présenté dans la figure 4.27.

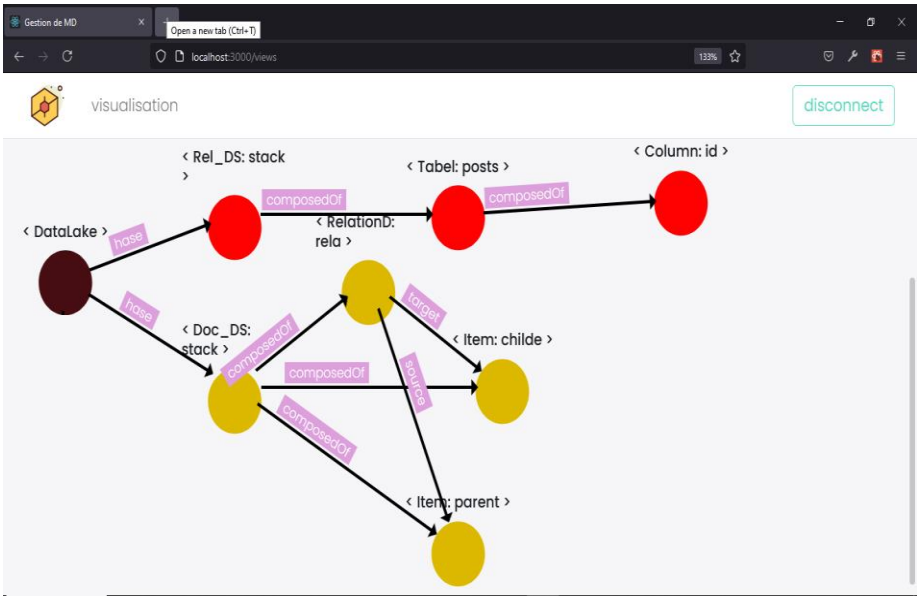
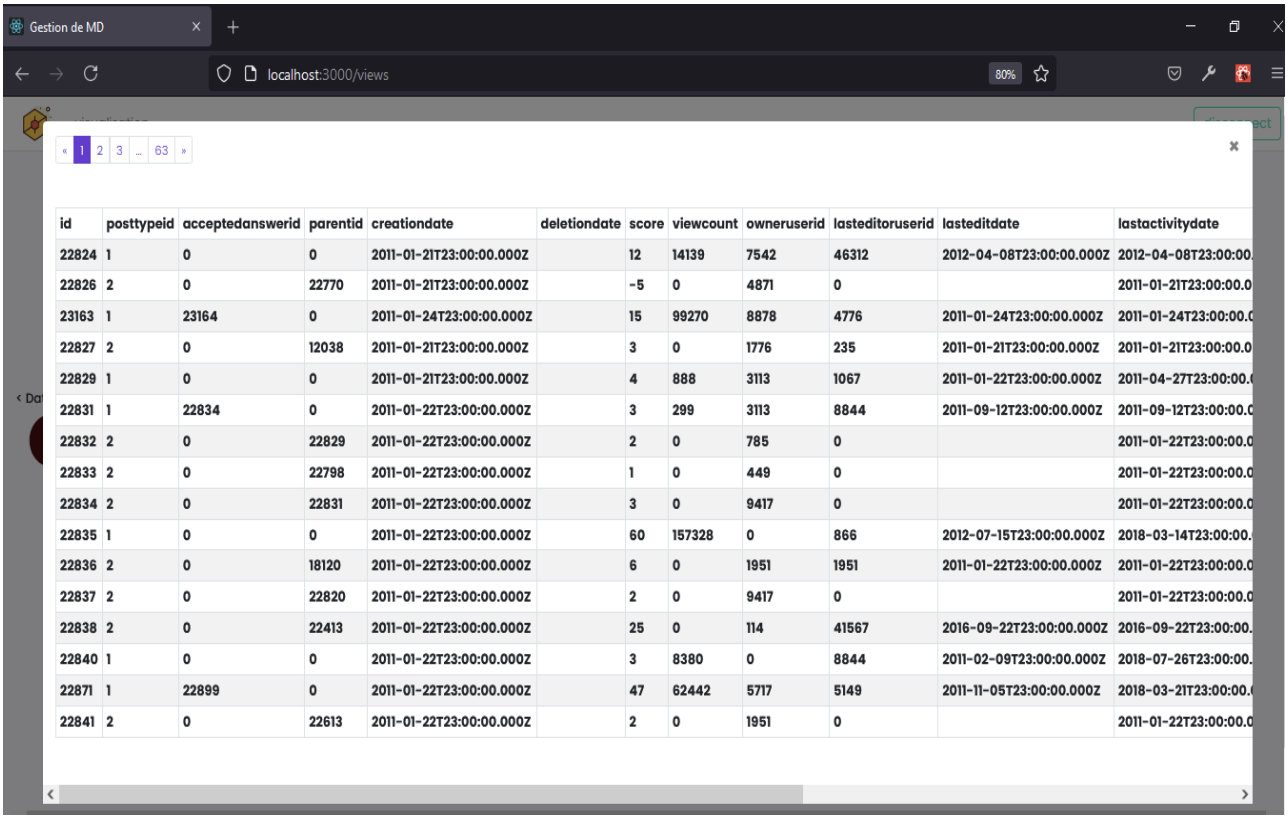


Figure 4.27 espace d'Exploitant.

L'exploitant peut choisir un élément pour afficher ses données. Par exemple, pour l'élément « Table : posts » on a stocké la requête « select * from posts » pour afficher ses données. La figure 4.28 montre le résultat de la requête.



id	posttypeid	acceptedanswerid	parentid	creationdate	deletiondate	score	viewcount	owneruserid	lasteditoruserid	lasteditdate	lastactivitydate
22824	1	0	0	2011-01-21T23:00:00.000Z		12	14139	7542	46312	2012-04-08T23:00:00.000Z	2012-04-08T23:00:00.
22826	2	0	22770	2011-01-21T23:00:00.000Z		-5	0	4871	0		2011-01-21T23:00:00.0
23163	1	23164	0	2011-01-24T23:00:00.000Z		15	99270	8878	4776	2011-01-24T23:00:00.000Z	2011-01-24T23:00:00.0
22827	2	0	12038	2011-01-21T23:00:00.000Z		3	0	1776	235	2011-01-21T23:00:00.000Z	2011-01-21T23:00:00.0
22829	1	0	0	2011-01-21T23:00:00.000Z		4	888	3113	1067	2011-01-22T23:00:00.000Z	2011-04-27T23:00:00.0
22831	1	22834	0	2011-01-22T23:00:00.000Z		3	299	3113	8844	2011-09-12T23:00:00.000Z	2011-09-12T23:00:00.0
22832	2	0	22829	2011-01-22T23:00:00.000Z		2	0	785	0		2011-01-22T23:00:00.0
22833	2	0	22798	2011-01-22T23:00:00.000Z		1	0	449	0		2011-01-22T23:00:00.0
22834	2	0	22831	2011-01-22T23:00:00.000Z		3	0	9417	0		2011-01-22T23:00:00.0
22835	1	0	0	2011-01-22T23:00:00.000Z		60	157328	0	866	2012-07-15T23:00:00.000Z	2018-03-14T23:00:00.
22836	2	0	18120	2011-01-22T23:00:00.000Z		6	0	1951	1951	2011-01-22T23:00:00.000Z	2011-01-22T23:00:00.0
22837	2	0	22820	2011-01-22T23:00:00.000Z		2	0	9417	0		2011-01-22T23:00:00.0
22838	2	0	22413	2011-01-22T23:00:00.000Z		25	0	114	41567	2016-09-22T23:00:00.000Z	2016-09-22T23:00:00.
22840	1	0	0	2011-01-22T23:00:00.000Z		3	8380	0	8844	2011-02-09T23:00:00.000Z	2018-07-26T23:00:00.
22871	1	22899	0	2011-01-22T23:00:00.000Z		47	62442	5717	5149	2011-11-05T23:00:00.000Z	2018-03-21T23:00:00.0
22841	2	0	22613	2011-01-22T23:00:00.000Z		2	0	1951	0		2011-01-22T23:00:00.0

Figure 4.28 l'affichage de données de la tables posts dans l'espace visualisation de l'exploitant.

4.5 Conclusion

Tout au long de ce chapitre, nous avons d'abord présenté l'environnement de travail. Par la suite, nous avons expliqué l'architecture de notre application et avons finalement présenté les principales fonctionnalités de notre application.

Conclusion générale

Ce projet de fin d'études avait pour ambition de mettre en place un système de gestion de métadonnées dans les lacs de données.

Notre travail a été conduit comme suit. Nous avons présenté les Concepts de Big data, de bases de données NoSQL, les lacs de données, ainsi que les métadonnées et leur rôle particulier dans les lacs de données. Nous avons conduit notre travail selon le processus 2TUP allégé où nous avons abordé les étapes d'analyse, de conception et de codage.

Ce travail nous a été très formateur, puisqu'il nous a permis de découvrir de nouvelles technologies innovantes, notamment le concept de lac de données qui permet de centraliser les données en vue de leur réutilisation par des analystes ou opérationnels. La technologie des graphes de propriétés nous a également permis de découvrir une façon plus flexible pour gérer les données qui permet de pallier la rigidité et contraintes du modèle relationnel.

Ce travail a également été riche en matières de contraintes à gérer, notamment les contraintes techniques, mais aussi les contraintes de temps, contraintes d'expérience et de technologie. En outre, ce projet a nécessité un pouvoir d'abstraction élevé, car il a été question de gérer deux niveaux d'abstraction (modèle et schéma) et d'appliquer les contraintes du modèle au schéma et de répondre à un besoin d'extensibilité par l'abstraction des éléments du lac de données par les concepts et liens entre concepts.

Comme futur travail, nous pensons à améliorer notre application, par exemple la lecture automatique des sources relationnel, ajouter d'autre type de source (comme les base de données Clés/Valeur par exemple). Nous pensons également à introduire les notions de web sémantique à notre projet pour permettre une intégration sémantique.

Références

[1] journal officiel du 22 aout 2014 : www.legifrance.gouv.fr (consulté le 26-01-2021)
[2] Dictionnaire anglais d'oxford : www.oed.com (consulté 26-01-2021)
[3] M. A. Beyer et D. Laney, « The Importance of "Big Data": A Definition », 2012.
[4] O. Polo1, 2, Étudiant Master KIMP 1École National d'Ingenieurs de Metz, France 2Universidad del Norte, Barranquilla, Colombie Octobre 2015, page 3
[5] Jean-Louis Monino, Soraya Sedkaoui , Big Data, Open Data et valorisation des données ,first published 2016 in Great Britain by ISTE Editions Ltd 27-37 St George's Road London SW19 4EU UK
[6] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, Samir Belfkih , October 2018,, Big Data technologies : A survey, Volume 30, Issue 4, Pages 431-448.
[7] Krish Krishnan, Data warehousing in the age of Big Data, 2013.
[8]International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5– No.4, March 2013 – www.ijais.org 16 type of NOSQL Databases ans its comparaison with relational Databases.
[9] Openclasseroom : https://openclassrooms.com (consulté le 27-05-2021)
[10] Déom ,Paul, Analyse des solutions de spatialisation dans les graphes NoSQL afin d'exploiter la topologie arc noeud des réseaux routier, liége université , 2018-2019.
[11] George Anadiotis ,Big on Data. Graph databases and RDF: It's a family affair. 2017. https://www.zdnet.com/article/graph-databases-and-rdf-its-a-family-affair (accès le 11/01/2021) .
[12] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Angela Hung Byers, McKinsey Global Institute [MGI]. (2011). Big Data : The next frontier for innovation, competition, and productivity. McKinsey&Company. Consulté en ligne le 12/01/2021 : https://www.mckinsey.com
[13] Bernard Espinasse and Patrice Bellot. Introduction au big data : Opportunités, stockage et analyse des mégadonnées. Techniques de l'Ingènieur, France, 2017.
[14]developpez.com : https://big-data.developpez.com/tutoriels/apprendre-faire-choix-architecture-big-data/ (consulter le 28-01-2021)
[15] lan robinson, jim webber et emil eifrem , graph databases,2eme edition2015, O'Reilly Media, INC,1005 Gravenstein Highway North, Srbastopol, CA 95472.
[16] Neo4j : https://neo4j.com
[17] semantic scholar consulté enligne le 12/01/2021 https://www.semanticscholar.org/topic/OrientDB/1558555
[18] Doctoral Symposium Paper, A Semantics-enabled approach for Data Lake Exploration Services, Massimiliano Garda, 2019 IEEE World Congress on Services (SERVICES)
[19] H. Mehmood et al., "Implementing big data lake for heterogeneous data sources," in Proceedings - 2019 IEEE 35th International Conference on Data Engineering Workshops, ICDEW 2019, 2019, pp. 37–44.
[20] Madsen, M.: How to Build an Enterprise Data Lake: Important Considerations before Jumping In. Third Nature Inc. (2015).
[21] Marz, N., Warren, J.: Big Data - Principles and best practices of scalable real-time data systems. Manning Publications Co. (2015).
[22] Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang ,Leveraging the Data Lake Current State and Challenges , In: Proceedings of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2019)
[23] Miloslavskaya, N. et A. Tolstoy (2016). Big Data, Fast Data and Data Lake Concepts. Procedia Computer Science 88, 300–305.
[24] what is data lake : https://aws.amazon.com (Consulté le 14/03/2021)

[25] Ravat, Franck and Zhao, Yan Data Lakes: Trends and Perspectives. (2019) In: International Conference on Database and Expert Systems Applications (DEXA 2019), 26 August 2019 - 29 August 2019 (Linz, Austria)
[26] Fang, H. , 2015, Managing data lakes in big data era: what's a data lake and why has it became popular in data management ecosystem. In: Proceedings of the International Conference on Cyber Technology in Automation (CYBER 2015), Shenyang, China, pp. 820–824. IEEE (2015)
[27] Russom. Data Lakes : Purposes, Practices, Patterns, and Platforms, 2017. https://tdwi.org/ Research
[28] DataOps.Roks par Saagie https://www.saagie.com/ (consulté le 15-03-2021)
[29] Cédrine Madera. L'évolution des systèmes et architectures d'information sous l'influence des données massives : les lacs de données. Base de données [cs.DB]. Université Montpellier, 2018. Français. ffNNT : 2018MONTS071ff. fftel-02138983f
[30] edureka! : https://www.edureka.co/ (consulté le 15-03-2021)
[31] Bill Inmon, Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump, 2016, edition R A Peters. Consulter enligne.
[32] J. Paul Getty Trust ,2008, Introduction to Metadata,2eme Edition, Paris: Firmin-Didot, 1869–1887, volume 1.
[33] J. Riley, 2017, Understanding metadata: What is metadata, and what is it for. National Information Standards Organization NISO Primer series. Baltimore, MD: National Information Standards Organization, isbn: 978- 1-937522-72-8. url: http://marciazeng.slis.kent.edu/metadatabasics/types.htm (cit. on pp. 17, 22).
[34] Bergamaschi, S., Castano, S., Vincini, M., Beneventano, D., 2001, Semantic integration and query of heterogeneous information sources. Data Knowl. Eng. 36(3), 215–249
[35] Terrizzano, I., Schwarz, P., Roth, M., Colino, J.E. ,2015, Data Wrangling: The Challenging Journey from the Wild to the Lake. In: Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR)
[36] IBM Analytics , (2016), The governed data lake approach. IBM.
[38] F. Castanedo and S. Gidley, 2017, "Understanding Metadata: Create the foundation for a Scalable Data Architecture," O'reilly.
[39] Coulondre S., Libourel T., Spéry L., 1998. Metadata and GIS: a classification of Metadata for GIS. GIS Planet'98, International Conférence and Exhibition on Geographic Information. Lisbonne, Portugal.
[40] Jeffrey Pomerantz, ntz, 2015, Metadata, Cambridge, MA: MIT Press, 2227–2230
[41] Oram, A. (2015), "Managing the Data Lake," O'Reilly, vol. 91, no. 1, p. 24.
[42] C. Diamantini, P. Lo Giudice, L. Musarella, D. Potena, E. Storti, and D. Ursino, 2018, A new metadata model to uniformly handle heterogeneous data lake sources, vol. 909, no. October. Springer International Publishing.
[43] G. de Simoni, A. Dayley, R. Edjlali, 2018, Magic Quadrant for Metadata Management Solutions. (cit. on pp. 17, 22–25).
[44] G. de Simoni., 2012, Five Ways to Use Metadata Management to Deliver Business Value for Data. (cit. on pp. 23–25).
[45] G. de Simoni, A. Dayley, R. Edjlali., 2018, 4 Use Cases That Drive Critical Capabilities in Metadata Management. (cit. on pp. 23, 24, 33).
[46] Quix, C., Hai, R., Vatov, I., 2016, Metadata Extraction and Management in Data Lakes With GEMMS. Complex Syst. Informatics Model. Q. 9 (9), 67–83.
[47] Hai, R., Geisler, S., Quix, C., 2016, Constance: An Intelligent Data Lake System. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD).
[48] Gallinucci, E., Golfarelli, M., Rizzi, S., 2018, Schema profiling of document-oriented databases. Information Systems 75, 13–25.
[49] Walker, C., Alrehamy, H., 2015, Personal Data Lake with Data Gravity Pull. In: Proceedings of the 2015 IEEE Fifth International Conference on Big Data and Cloud Computing (BDCloud) IEEE.

[50] Nogueira, I., Romdhane, M., Darmont, J., 2018, Modeling Data Lake Metadata with a Data Vault. In: Proceedings of the 22nd International Database Engineering Applications Symposium (IDEAS).
[51] Pegdwendé N. Sawadogo, Étienne Scholly, Cécile Favre, Éric Ferey, Sabine Loudcher, and Jérôme Darmont, Metadata Systems for Data Lakes: Models and Features, Université de Lyon, Lyon 2, ERIC EA 3083
[52] E. Zagan and M. Danubianu, 2020, "Data Lake Approaches: A Survey," in 2020 15th International Conference on Development and Application Systems, DAS 2020 – Proceedings.
[53] A. Oram and S. Gidley, 2019, Data Lake Maturity Model, O'Reilly.
[54] V. Theodorou, R. Hai, and C. Quix, 2019, "A Metadata Framework for Data Lagoons," in Communications in Computer and Information Science.
[55] J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen, 2018, "Analytical metadata modeling for next generation BI systems," J. Syst. Softw., vol. 144, pp. 240–254, Oct.
[53] Y. H. Chen, H. H. Chen, and P. C. Huang, 2018, "Enhancing the data privacy for public data lakes," Proc. 4th IEEE Int. Conf. Appl. Syst. Innov. 2018, ICASI 2018, pp. 1065–1068.
[54] A. Whitepaper, 2019, "Building Big Data Storage Solutions (Data Lakes) for Maximum Flexibility,"
[55] S. Gupta and V. Giri, 2018, "Practical enterprise data lake insights: Handle data-driven challenges in an enterprise big data lake," Pract. Enterp. Data Lake Insights Handle Data-Driven Challenges an Enterp. Big Data Lake, pp. 1–327.
[56] O. Hartig and J. Zhao, 2009, "Using Web data provenance for quality assessment," in CEUR Workshop Proceedings.
[57] A. Maccioni and R. Torlone, 2018, "KAYAK: A framework for just-in-time data preparation in a data lake," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).
[58] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, 2010, "The Hadoop distributed file system," in 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010.
[59] A. Halevy et al., 2016, "Goods: Organizing Google's datasets," in Proceedings of the ACM SIGMOD International Conference on Management of Data, vol. 26-June-2016.
[60] M. N. Mami, D. Graux, S. Scerri, H. Jabeen, and S. Auer, 2019, "Querying data lakes using spark and presto," in The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019.
[61] B. Bilalli, A. Abelló, T. Aluja-Banet, and R. Wrembel, 2016, "Towards intelligent data analysis: The metadata challenge," IoTBD 2016 - Proc. Int. Conf. Internet Things Big Data, pp. 331–338.
[62] Wikipedia : https://fr.wikipedia.org/wiki/React (consulté le 27-05-2021)
[63] React : https://fr.reactjs.org/ (consulté le 27-05-2021)
[64] Wikipedia : https://fr.wikipedia.org/wiki/Node.js (consulté le 27-05-2021)
[65] MDN Web Docs : https://developer.mozilla.org (consulter le 27-05-2021)
[66]Wikipedia: https://en.wikipedia.org/wiki/PostgreSQL (consulté le 27-05-2021)
[67] D-BOOKER: https://www.d-booker.fr (consulté le 27-05-2021)
[68] Wikipedia : https://fr.wikipedia.org/wiki/Neo4j (consulté le 27-05-2021)
[69] Wikipedia : https://en.wikipedia.org/wiki/MongoDB (consulté le 27-05-2021)