Business Intelligence : vers un ETL à la demande sur des bases de données orientées documents

Manel Souibgui

▶ **To cite this version:**

Manel Souibgui. Business Intelligence : vers un ETL à la demande sur des bases de données orientées documents. Algorithme et structure de données [cs.DS]. HESAM Université; Université Tunis El Manar. Faculté des Sciences Mathématiques, Physiques et Naturelles de Tunis (Tunisie), 2022. Français. NNT : 2022HESAC015 . tel-03938453

HAL Id: tel-03938453
https://theses.hal.science/tel-03938453

Submitted on 13 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HESAM UNIVERSITÉ

le cnam

ÉCOLE DOCTORALE Sciences des Métiers de l'Ingénieur
Centre d'études et de recherche en informatique et communications
ÉCOLE DOCTORALE Mathématiques, Informatique, Sciences et Technologie de la Matière
Laboratoire en informatique, programmation, algorithmique et heuristique

THÈSE

présentée par : **Manel SOUIBGUI**
soutenue le : **14 décembre 2022**

pour obtenir le grade de : **Docteur d'HESAM Université & de l'Université de Tunis El Manar**

Conservatoire national des arts et métiers & Faculté des sciences de Tunis

Discipline : **Informatique**

Spécialité : **Sciences pour l'ingénieur spécialité Informatique**

Business Intelligence: Towards On-demand ETL over Document Stores

THÈSE dirigée par :
Mme Samira SI-SAID CHERFI Professeure des Universités, CNAM
M. Sadok BEN YAHIA Professeur des Universités, FST

et co-encadrée par :
Mme Faten ATIGUI Maître de Conférences, CNAM

Jury

Mme Nedra MELLOULI	Maître de Conférences HDR, Université Paris 8	Présidente
M. Olivier TESTE	Professeur, Université Toulouse 2	Rapporteur
M. Jérôme DARMONT	Professeur, Université Lyon 2	Rapporteur
M. Mohamed Mohsen GAMMOUDI	Professeur, ISAMM, Université Manouba	Rapporteur
Mme Fatma ABDELHÉDI	Directrice du LR-CBI, France	Examinatrice

THÈSE

HESAM UNIVERSITÉ

UNIVERSITÉ DE TUNIS EL MANAR

FACULTE DES SCIENCES DE TUNIS

le cnam

ÉCOLE DOCTORALE Sciences des Métiers de l'Ingénieur
Centre d'études et de recherche en informatique et communications
ÉCOLE DOCTORALE Mathématiques, Informatique, Sciences et Technologie de la Matière
Laboratoire en informatique, programmation, algorithmique et heuristique

THÈSE

présentée par : **Manel SOUIBGUI**
soutenue le : **14 décembre 2022**

pour obtenir le grade de : **Docteur d'HESAM Université & de l'Université de Tunis El Manar**

Conservatoire national des arts et métiers & Faculté des sciences de Tunis

Discipline : **Informatique**

Spécialité : **Sciences pour l'ingénieur spécialité Informatique**

Business intelligence : vers un ETL à la demande sur des bases de données orientées documents

THÈSE dirigée par :
Mme Samira SI-SAID CHERFI Professeure des Universités, CNAM
M. Sadok BEN YAHIA Professeur des Universités, FST

et co-encadrée par :
Mme Faten ATIGUI Maître de Conférences, CNAM

Jury

Mme Nedra MELLOULI	Maître de Conférences HDR, Université Paris 8	Présidente
M. Olivier TESTE	Professeur, Université Toulouse 2	Rapporteur
M. Jérôme DARMONT	Professeur, Université Lyon 2	Rapporteur
M. Mohamed Mohsen GAMMOUDI	Professeur, ISAMM, Université Manouba	Rapporteur
Mme Fatma ABDELHÉDI	Directrice du LR-CBI, France	Examinatrice

THÈSE

Affidavit

Je soussignée, Manel Souibgui, déclare par la présente que le travail présenté dans ce manuscrit est mon propre travail, réalisé sous la direction scientifique de Pr. Sadok Ben Yahia et Pr. Samira Si-Said Cherfi, dans le respect des principes d'honnêteté, d'intégrité et de responsabilité inhérents à la mission de recherche. Les travaux de recherche et la rédaction de ce manuscrit ont été réalisés dans le respect de la charte nationale de déontologie des métiers de la recherche. Ce travail n'a pas été précédemment soumis en France ou à l'étranger dans une version identique ou similaire à un organisme examinateur.

Fait à Paris, le 10 janvier 2023

Signature 

Affidavit

I, undersigned, Manel Souibgui, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Pr. Sadok Ben Yahia and Pr. Samira Si-Said Cherfi, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with the French charter for Research Integrity. This work has not been submitted previously either in France or abroad in the same or in a similar version to any other examination body.

Paris, January 10^{th}, 2023

Signature

To my parents Beji and Nejma

Without your inspiration, endless love, drive, and encouragement, I might not be the person I am today. Thank you mom and dad! Everything I have and everything I am, I owe it all to you. Thank you, my two lifelines ...

To my lovely sister Hanen

You always encourage me with passion and endless support. Thank you for being the warm and caring person that you've always been.

To my beloved brothers Riadh and Bilel

I'm thankful to have a supportive and loving brothers like you.

To my brother-in-law Faycel and my sisters-in law Nassira and Salsabil

To my beautiful nieces Nada, Sirine, Saja and Lina

To my lovely nephews Yassine, Iyed, Yasser and Anas

To my family

To my friends

and all those who made this achievement possible.

Thank you all for your everlasting love and warm encouragement throughout my research.

Acknowledgments

I would like to express my deepest gratitude to my co-supervisors *Sadok BEN YAHIA* and *Samira SI-SAID CHERFI* for believing in me. I gratefully acknowledge their support, understanding and motivation during these past years.

I would also extend my sincere gratitude to my advisor *Faten ATIGUI* for her guidance, encouragement and continuing support. Her mentoring has contributed to both my professional and personal growth.

I would like to extend my sincere thanks to to the thesis monitoring committee members *Nicolas TRAVERS* and *Mohamed-Amine BAAZIZI* for their thoughtful comments and recommendations on this project.

I would like to acknowledge the help and assistance that I received from Mrs Saloua ZAMMALI.

I also thank Alexandre LESCAUT, Meryem HARA, and Sandra BOSSE for all the administrative help.

Thanks to Conservatoire National des Arts et métiers, Computer Science Department staff members, my friends and colleagues in the CEDRIC laboratory who gave me a lot of encouragement to finish this thesis.

Thanks to Faculty of Sciences of Tunis, Computer Science Department staff members, my friends and colleagues in the LIPAH laboratory, who supported me throughout my thesis.

For this dissertation, I would like to thank the members of my thesis committee. I thank Mrs. Nedra MELLOULI for examining and agreeing to chair my thesis committee. I would like to thank my reading committee members Mr. Olivier TESTE, Mr. Jérôme DARMONT and Mr. Mohamed Mohsen GAMMOUDI for accepting to review my thesis report. I would also like to thank Mrs. Fatma ABDELHÉDI for examining my thesis report.

Résumé

L'émergence des données issues du web et de ses corollaires entraîne une révolution digitale qui touche tous les aspects d'une entreprise. L'évolution de ces données en termes de volume, de variété et de vélocité implique la nécessité d'intégrer dans les systèmes existants des services axés sur le Big Data. Impactés par cette révolution, les systèmes Business Intelligence & Analytics (BI&A), qui visent à supporter la prise de décision grâce à l'extraction des données pertinentes à partir des bases de données de l'entreprise, nécessitent d'être revus. En effet, à l'ère du Big Data, les bases de données NoSQL sont devenues omniprésentes en tant que systèmes hautement extensibles et sans schéma pour stocker des données volumineuses ayant une variété et une vélocité importantes. Bien que les systèmes NoSQL sont largement utilisés aujourd'hui, leur exploitation reste limitée dans le contexte de la BI&A qui demeure, pendant de longues années, liée essentiellement aux systèmes de gestion de bases de données conventionnels telles que les bases de données relationnelles. En fait, dès les premiers jours de l'entreposage des données, le modèle relationnel a été fondamental dans la quête de la cohérence et de la qualité des données analytiques dans les systèmes BI&A.
Afin d'améliorer les performances, le modèle NoSQL offre une grande variété structurelle et une flexibilité accrue de schémas ; mais il a abandonné les règles phares définissant les données relationnelles, à savoir, la définition préalable d'un schéma et la définition des contraintes d'intégrité. Par conséquent, l'exploitation des données sans schéma, et souvent sans contraintes d'intégrité pour la prise de décision nécessite de revoir toutes les phases de l'architecture BI&A, notamment le processus Extract-Transform-Load (ETL) afin de pouvoir gérer le volume, la variété et la vélocité des données issues des systèmes NoSQL, tels que les systèmes orientés documents.

Dans ce contexte, l'objectif principal de cette thèse est d'explorer comment extraire, transformer et charger les bases de données orientées documents pour des fins décisionnelles ? et comment préparer ces données variées et volumineuses pour répondre aux besoins des décideurs ?

Pour ce faire, dans un premier temps, nous avons mené une étude approfondie de la littérature sur ces problématiques. Cette revue nous a conduits à introduire une nouvelle approche de Business Intelligence & Analytics permettant d'extraire, transformer et de charger à la demande les données requises pour la prise de décision à partir de bases de données orientées documents. Nous nous concentrons sur l'ETL à la demande où, contrairement aux travaux existants, nous considérons la dispersion des données sur deux ou plusieurs collections. En effet, il est courant que les données indispensables pour la prise de décision soient stockées dans des collections différentes. Dans ce cas, le problème qui se pose est comment joindre plusieurs collections en l'absence d'un schéma et des contraintes d'intégrité définis au préalable ? Assurer cette tâche manuellement s'avère laborieux, coûteux et infaisable dans les ensembles de données à grande échelle.

Ainsi, dans un second temps, nous étudions le problème de la découverte automatique des attributs clés qui serviront à la jointure de collections. Nous proposons un algorithme qui vise à détecter automatiquement les identifiants et les références composés et non composés à partir de plusieurs sources de données orientées documents. L'approche est basée sur des caractéristiques et des règles d'élagage qui visent à définir les identifiants candidats. Afin de détecter les pairs (identifiant, référence) entre chaque deux collections, nous avons adopté la technique de plongement de graphe : *node2vec*, qui offre des avantages significatifs grâce aux similarités syntaxiques et sémantiques.

Afin de valider notre approche, nous avons développé un prototype qui met en oeuvre les deux niveaux : *(i)* détection des identifiants candidats ; et *(ii)* identification des paires candidates des attributs clés. L'étude expérimentale est basée sur deux benchmarks : TPC-H[1] et TPC-E[2] et deux sources de données réelles : Twitter [DiScala and Abadi, 2016] et Musicians [3]. Les résultats obtenus montrent la faisabilité et la pertinence de notre approche en termes de précision et de cohérence.

Mots-clés : Business Intelligence & Analytics, ETL, Base de données orientées documents, Découverte d'identifiants, Découverte de références

[1] https://github.com/souibguimanel/TPCHjson
[2] https://github.com/souibguimanel/TPCEjson
[3] https://www.wikidata.org/wiki/Wikidata:Main_Page

Abstract

In today's world, data play a valuable role in organizations. Most notably, the volume and variety of the gathered data have expanded dramatically leading to a digital revolution that affects all aspects of a company. The evolution of these data explains the need to integrate big data techniques, analytics, and operations into existing systems. Impacted by this revolution, Business Intelligence & Analytics systems (BI&A), which aims to bolster decision-making by extracting pertinent data to gain insights of significant worth, likewise need to be reviewed.

In fact, in the big data era, NoSQL stores have become ever-present as a highly expandable and schema-free system that can better handle the volume, variety, and velocity of data. Although NoSQL systems are broadly used today, BI&A wields conventional database management systems such as relational databases. In fact, from the earliest days of data warehousing, the qualities of the relational model have been highly valued in the quest for data consistency and quality in BI&A.

To enhance performance, the NoSQL model offers a large structural variety and increased flexibility of schemata. But, it abandons the key rules defining relational data, namely predefined schema, and predefined integrity constraints.

Therefore, exploiting schema-free data for decision-making is challenging since it requires reviewing all the BI&A phases, particularly the Extract-Transform-Load (ETL) process, to fit the volume, variety, and velocity of NoSQL data like document stores data.

In this context, the main goal of this thesis is to explore how to extract, transform, and load document stores for decision-making? and how to prepare these data to fit the decision-makers' requirements?

For this, we start by conducting an in-depth literature review regarding these issues. Then, based on the underlined limits of these works, we introduce a new BI&A approach that extracts, transforms, and

loads on-demand the required data (on-demand ETL), for decision-making based on document stores. We particularly focus on the on-demand ETL where, unlike existing works, we consider the dispersion of data over two or more collections. Indeed, it is common for relevant data, used in decision-making, to be stored in different collections. This calls into question how to join dispersed data across several collections in the absence of integrity constraints that are not defined beforehand? A document may have hundreds of fields. Hence, fetchig join fields manually is time and effort-consuming and infeasible in large-scale datasets.

Thus, in a second step, we study the problem of automatic discovery of key fields that will be used to join collections. We propose an algorithm that aims to automatically detect both composite and non-composite *identifiers* and *references* on several document stores. The approach is based on scoring features and pruning rules to discover actual candidate *identifiers* from many initial ones efficiently. To find candidate pairs (identifier, reference) between several document stores, we put into practice *node2vec* as a graph embedding technique, which yields significant advantages while using syntactic and semantic similarity measures.

To validate our proposal, we have developed a prototype that implements the two phases of our approach: *(i)* candidate *identifiers* discovery for each collection; and *(ii)* identification of candidate pairs of key fields (*identifier* and *reference*) for every two collections. We base our experimental study on two benchmarks: TPC-H[4] and TPC-E[5], and two real-world datasets: Twitter [DiScala and Abadi, 2016] and Musicians[6]. The obtained results demonstrate the feasibility and the relevance of our approach in terms of precision and accuracy.

Keywords: Business Intelligence & Analytics, ETL, Document stores, Identifiers discovery, References discovery

[4]TPC-H: https://github.com/souibguimanel/TPCHjson
[5]https://github.com/souibguimanel/TPCEjson
[6]https://www.wikidata.org/wiki/Wikidata:Main_Page

Contents

List of Tables

List of Figures

Synthèse en français de la thèse

L'informatique décisionnelle (Business Intelligence & Analytics, i.e., BI&A) désigne les applications et les pratiques permettant d'analyser l'information et d'améliorer la prise de décision [Rizzi, 2009]. Dans une architecture BI&A, comme le montre la figure 1, les données sont extraites à partir de sources hétérogènes, de différents types et formats, pour être transformées et chargées dans un Entrepôt de Données (ED) par le biais de processus ETL (Extract-Transform-Load). Les processus ETL ont trois fonctions principales : *(i)* extraction des données à partir des sources; *(ii)* transformation des données (nettoyage, agrégation, conversion, etc.) pour qu'elles puissent être stockées sous le format ou la structure appropriés favorisant les analyses OLAP ; et *(iii)* chargement de l'ensemble des données résultant dans le système cible, typiquement, un ED. Les analyses s'appuient ensuite sur des requêtes OLAP et des outils de visualisation dédiés à la prise de décision.

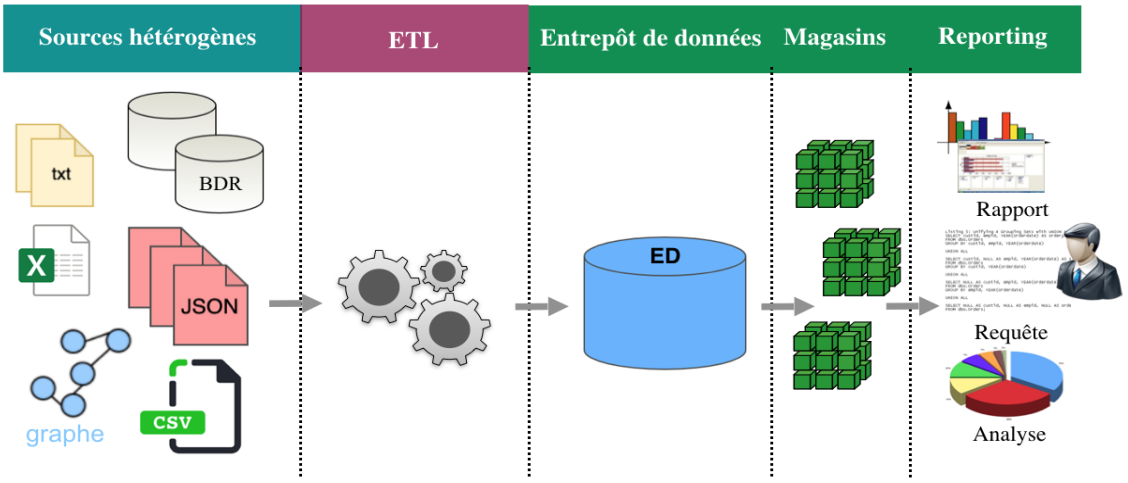

Figure 1: Architecture du système Business Intelligence & Analytics

Avec la transformation digitale et l'avènement des big data, les architectures BI&A subissent une évolution majeure. En effet, les données à analyser étant caractérisées par une Variété, une Vélocité et un Volume importants (3V), elles sont stockées dans des bases de données (BD) adaptées telles que les BD NoSQL qui s'avèrent de plus en plus utilisées. Ces BD, qui offrent une flexibilité de schéma,

regroupent quatre familles : clé/valeur, orientées colonnes, orientées documents et orientées graphes.

Afin d'exploiter ces BD pour la prise de décision, il est indispensable de fournir une architecture BI&A adaptée et évolutive. La phase de l'ETL étant l'une des phases les plus complexes et les plus coûteuses, et dont dépend fortement la réussite d'un projet BI&A [Souibgui et al., 2019], il est nécessaire de proposer une approche qui tient compte à la fois de la nature des sources de données et des besoins décisionnels afin d'exploiter de manière efficace et performante plusieurs sources de données variées telles que les BD NoSQL.

Pendant de longues années, les systèmes BI&A utilisent essentiellement les BD relationnelles [Ram et al., 2016]. En effet, dès les premiers jours de l'entreposage des données, le modèle relationnel a été fondamental dans la quête de la cohérence et de la qualité des données analytiques.

Avec l'avènement des big data, depuis plus d'une décennie, les BD NoSQL sont devenues couramment utilisées pour stocker des données volumineuses, variées et ayant une vélocité importante. La question qui s'est rapidement posée dans ce contexte, est de savoir comment les BD NoSQL se rapportent-elles à l'informatique décisionnelle et comment exploiter ces données variées et volumineuses pour la prise de décision ? Exploiter des données sans schéma défini au préalable, et souvent sans contraintes d'intégrité, pour la prise de décision nécessite de revoir toutes les phases de l'architecture BI&A, en particulier les processus ETL [Theodorou et al., 2017]. Nous nous intéressons particulièrement au modèle orienté document qui s'avère de plus en plus utilisé dans le monde industriel [Gartner, 2017], par exemple, le système MongoDB a dépassé les 26.2%[7] de taux d'utilisation dans les entreprises en 2021. Pour ce faire, il est indispensable de fournir une approche BI&A dédiée, capable d'extraire, de transformer et de charger des données variées et hétérogènes.

Voici en substance les questions de recherche auxquelles nous nous intéressons dans ce travail :

- Comment exploiter plusieurs sources de données orientées documents pour des fins décisionnelles ?

  L'idée est de revoir toutes les phases de l'architecture BI&A, en particulier le processus ETL, pour fournir des solutions capables d'exploiter des sources de données orientées documents pour la prise de décision. Il est essentiel de fournir une approche BI&A dédiée qui part de sources de données orientées documents, extrait, transforme et charge les données dans l'ED, et fournit les

[7] https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba

moyens nécessaires aux décideurs pour analyser facilement ces données.

- Comment extraire transformer et charger des données dispersées sur plusieurs sources orientées documents ?

  L'idée est de revoir le processus ETL afin de fournir de nouvelles solutions dédiées à des sources orientées documents et hétérogènes. En effet, dans une approche BI&A, on est souvent amené à utiliser plusieurs sources et plusieurs "tables" de la même source. Ainsi, le principal défi lors de l'utilisation de sources de données sans schéma consiste à extraire des données dispersées sur plusieurs collections. Ce défi conduit à une question cruciale : comment détecter les attributs clés de jointure dans des sources de données en l'absence de schéma défini au préalable et de contraintes d'intégrité ?

Pour répondre à ces questions, nous avons mené une étude bibliographique approfondie (Section). Dans la Section , nous avons proposé une nouvelle approche BI&A qui vise à extraire tansformer et charger des sources de données orientés documents. La Section présente notre solution pour détecter les paires d'attributs clés en vue de faire la jointure de deux BD orientées documents. Afin de valider nos contributions, nous avons développé un prototype que nous décrivons dans la Section . Nous présentons un bilan sur nos contributions et une esquisse de nos travaux futurs dans la Section .

Notre objectif étant d'extraire, de transformer et de charger des BD NoSQL pour la prise de décision, nous présentons dans cette section les travaux existants dans la littérature autour de cette problématique. Nous nous intéressons particulièrement aux travaux qui ont traité le problème de l'ETL en se concentrant sur la jointure. Nous présentons également les travaux qui ont proposé des contributions pour l'analyse OLAP de ces BD.

Peu de travaux ont abordé le problème de l'ETL dans le contexte des BD NoSQL, en particulier les BD orientées documents [Asanka, 2015, Yangui et al., 2017]. Dans [Mallek et al., 2018], les auteurs ont proposé un outil, appelé *BigDimETL*, traitant le processus ETL dans le contexte de BD NoSQL. Les données sont extraites à partir des BD orientées documents pour être transformées en BD orientées colonnes afin d'appliquer les techniques de partitionnement. L'approche vise à augmenter la performance en parallélisant le traitement des opérations de sélection, projection et jointure. Il est à noter que dans la littérature, de nombreux travaux se sont intéressés au problème d'extraction de

schéma des BD orientées documents. Étant une étape indispensable dans un processus ETL, nous étudions ces différentes contributions. Ainsi, Klettke et al. [Klettke et al., 2015] ont proposé une méthode d'extraction d'un schéma d'une collection de documents JSON représenté sous forme d'un graphe. D'autre part, Gallinucci et al. [Gallinucci et al., 2018] ont étendu le cadre d'extraction de schéma d'une collection de documents JSON, en utilisant des techniques de profilage de schémas afin d'expliquer la variété des structures des documents JSON en utilisant un arbre décisionnel.

Bien que de nombreux auteurs ont procédé à l'extraction de schémas, ce problème est encore insuffisamment exploré. L'inconvénient majeur réside dans le fait que ces approches ne couvrent pas la recherche des types réels cachés des attributs lors de l'extraction de schémas. Cette étape s'avère indispensable pour la manipulation et l'interrogation de ces BD. Cependant, pour détecter les paires d'identifiant et de référence correctement, il est nécessaire de l'enrichir par l'identification des types cachés des attributs, faute de quoi des résultats faux positifs seront obtenus.
D'autre part, même si la plupart de ces travaux s'est concentrée sur la première phase du processus ETL, c'est à dire la phase d'extraction de données ou la phase d'extraction de schémas, les contributions dans la phase de transformation restent encore limitées et nécessitent plus d'efforts. Dans la suite, nous étudions en particulier les travaux mettant l'accent sur l'opération de jointure dans le contexte des bases de données NoSQL.

Dans [Celesti et al., 2019], les auteurs ont discuté l'impact de l'exécution de l'opération de jointure entre deux collections de documents JSON. Ils ont proposé un algorithme qui effectue la jointure sur deux collections stockées dans MongoDB. L'algorithme nécessite d'avoir les deux clés de jointure en entrée. De plus, comme la jointure est une opération obligatoire pour interroger les BD orientées documents, nous avons également étudié les approches dédiées à l'interrogation [Kondylakis et al., 2019, Mami et al., 2019, Pokorný, 2020]. Toutes ces contributions reposent sur l'hypothèse que les clés de jointure sont connues à l'avance. À notre connaissance, aucun travail antérieur n'a proposé une méthode pour détecter automatiquement la paire de clés de jointure dans deux sources NoSQL, c'est-à-dire trouver les deux identifiants et leurs références respectives. Par conséquent, il est intéressant d'examiner les recherches antérieures menées dans le contexte des BD relationnelles [Wu et al., 2019]. Dans un travail récent, Jiang et Naumann [Jiang and Naumann, 2020] ont proposé une approche pour

découvrir automatiquement les clés primaires et les clés étrangères pour une base de données relationnelle. L'approche est basée sur les dépendances fonctionnelles, à savoir la *Combinaison de Colonnes Unique* et les *Dépendances d'Inclusion*. Les deux types de dépendances ont été utilisés pour, respectivement, détecter les clés primaires et les clés étrangères dans les bases de données relationnelles. Dans le contexte des BD orientées documents, un identifiant n'a pas exactement les mêmes caractéristiques que la clé primaire dans les BD relationnelles. En fait, une *Combinaison de Colonnes Unique* est l'ensemble des attributs dont la projection ne contient que des valeurs uniques et non nulles, alors que dans les BD orientées documents, le schéma n'est pas rigide. Les attributs peuvent être absents dans certains documents ou peuvent avoir des valeurs nulles.
Ainsi pour résoudre le problème de détection des identifiants et leurs références partant de sources de données orientées documents, il faut trouver d'autres alternatives adaptées à la nature sans schémas de ces BD. En effet, même si le problème de la détection des clés primaires et des clés étrangères dans les bases de données relationnelles est un problème connu et que des solutions dédiées ont été proposées dans la littérature [Jiang and Naumann, 2020], le problème de la détection automatique des identifiants et des références dans les BD orientées documents reste un défi auquel aucune contribution préalable n'a été faite.

OLAP est largement utilisé comme une approche d'analyse de données structurées pour faciliter la prise de décision. Avec la diffusion des BD NoSQL, contenant des données semi-structurées, il devient nécessaire de trouver des solutions permettant l'analyse multidimensionnelle (OLAP) sur des BD NoSQL. Les auteurs dans [Chouder et al., 2019] ont proposé une approche, qui s'intègre dans le contexte de self-service BI. Elle consiste à extraire le schéma d'une collection de documents JSON. Un schéma multidimensionnel initial est généré pour permettre à l'utilisateur de formuler sa requête. La validation de la requête se base sur la recherche des dépendances fonctionnelles approximatives. Au fur et à mesure qu'une dépendance fonctionnelle est trouvée, le schéma multidimensionnel est raffiné pour ajouter les niveaux de hiérarchies. La requête validée est traduite en langage natif du système MongoDB pour répondre aux analyses OLAP. Il est de même pour les approches dans [Francia et al., 2020], [Gallinucci et al., 2018], qui permettent l'application OLAP directement sur une collection de documents JSON. Elles diffèrent de l'approche de [Chouder et al., 2019] par le fait de chercher toutes les dépendances fonctionnelles, qui sont représentées sous forme d'un graphe au lieu de les chercher à

la demande.

Bien qu'il existe de nombreuses études, elles ne peuvent être considérées comme concluantes. En effet, les travaux existants qui ont traité de l'analyse OLAP sur les bases de données orientées documents, n'ont considéré qu'une seule collection. Cependant, dans le cadre de BI&A, les données sont souvent éparpillées sur plusieurs collections. À notre connaissance, il n'existe pas une approche globale de BI&A qui part de sources NoSQL, avec une multitude de sources hétérogènes et dispersées, et qui permet la création d'un entrepôt de données jusqu'à arriver aux analyses OLAP.

Dans un premier, nous proposons dans cette thèse une approche permettant d'extraire, transformer et charger les données dispersées sur plusieurs sources orientées documents. Étant donné que ce type de sources de données est extrêmement volumineux et évolutif, notre approche est pilotée à la fois par les sources de données et les besoins des utilisateurs finaux. Elle opère en deux étapes principales:

1. Étape 1 : *OnDemand ETL :* extraire, transformer et charger des BD orientées documents à la demande. Cette étape vise à récupérer des données pertinentes et nécessaires pour répondre aux besoins d'analyse en deux phases :

   - Phase 1 : *Extraction de schémas :* les BD orientées documents ont une structure variable et dynamique. Cette variété est explicitée par la présence ou l'absence de certains attributs avec une variété de types. Par conséquent, inférer le schéma de chaque collection devient primordial pour l'intégration de données. Cette étape consiste à extraire "à froid" le schéma plat pour chaque collection de documents. Le schéma plat représente un ensemble d'attributs avec leurs types, e.g., *JSONArray*, *Object*, *Integer*, etc. Chaque attribut est exprimé sous forme de son chemin d'accès à partir de la racine. Comme un type simple d'un attribut donné peut être caché sous un autre type simple, nous proposons une méthode qui vise à vérifier le type réel caché de chaque attribut pour détecter de tels cas. Par exemple, si les valeurs d'un attribut donné sont de type Float (exemple : 5.02), alors qu'elles sont représentées entre guillemets : "5.02", dans ce cas, le type réel doit être Float au lieu de String.
     Le nombre de schémas globaux (m schémas plats) extraits à partir des sources est inférieur ou égal au nombre initial des collections, à savoir n collections.

- Phase 2 : *Intégration de schémas et de données :* cette étape permet d'extraire "à chaud" les données requises par les besoins d'analyse. Pour identifier l'ensemble de collections contenant les données pertinentes, cette phase consiste à faire un alignement entre les schémas et les attributs multidimensionnels exprimant le besoin du décideur. En effet, chaque attribut multidimensionnel peut se trouver dans une ou plusieurs collections, et comme il existe une myriade de collections, il serait coûteux de chercher les attributs d'analyse en parcourant la liste complète des données dans chaque collection. Ainsi, faire recours à l'extraction de schémas est indispensable pour relever les défis posés par la nécessité du passage à l'échelle. L'extraction des données pertinentes nécessite souvent d'accéder à plus qu'une collection de documents. Cet accès multiple exige de trouver les clés pour effectuer une jointure. Dans notre contribution, nous nous concentrons principalement sur la jointure car c'est une opération complexe à assurer, alors qu'aucune définition préalable des paires de clés de jointure n'a été faite.

2. Étape 2 : Analyse OLAP : à cause de l'explosion des données NoSQL qui sont caractérisées par la flexibilité de schéma, et la complexité d'intégration, l'objectif de cette phase est de garantir une analyse OLAP adaptée au contexte des BD NoSQL, particulièrement aux BD orientées documents, pour garantir des réponses aux attentes des décideurs en temps opportun. Nous notons que cette phase ne fait pas partie du cadre de cette thèse.

Nous avons ensuite étudier le problème de détection des paires d'identifiant et de référence candidates. Notre objectif est de détecter les paires (identifiant, référence) entre chaque deux collections de documents JSON. Comme le montre la figure 2, cette phase opère en deux étapes principales : *(i)* détection des identifiants candidats ; et *(ii)* identification des paires candidates des attributs clés.

Dans la première étape de détection des identifiants candidats, nous identifions une première liste d'identifiants candidats pour chaque collection. La recherche se limite aux attributs de types simples, fréquents et de valeurs uniques. Puisque, un identifiant ne peut pas être de type *JSONObject* ou *JSONArray*, nous avons limité la recherche uniquement aux attributs simples. En outre, en raison de la flexibilité du schéma, les documents d'une même collection peuvent présenter une variété structurelle. Certains attributs peuvent être présents dans un document et absents dans les autres. Ainsi, nous

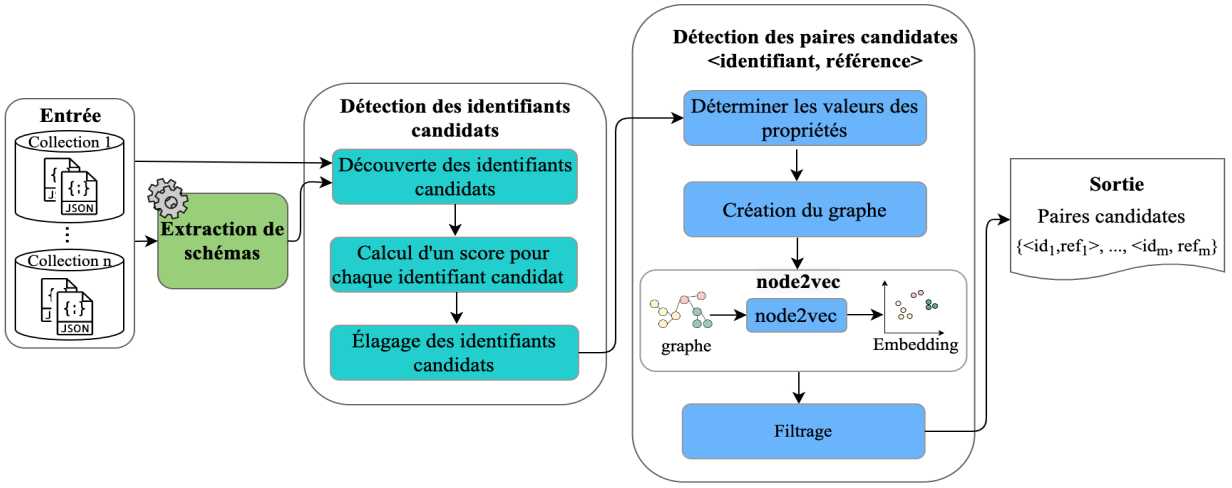

Figure 2: Architecture de la phase de détection des paires candidates des identifiants et des références

classons les attributs en fréquents et non fréquents. Pour trouver les identifiants candidats, nous considérons uniquement les attributs fréquents. Ensuite, dans la liste des attributs fréquents ayant des types simples, nous cherchons les attributs à valeurs uniques.

Ensuite, nous associons un score pour chaque attribut de la liste obtenue. Pour calculer le score, nous avons utilisé un ensemble de caractéristiques. Dans le contexte des BD relationnelles, plusieurs caractéristiques ont été proposées dans la littérature pour identifier les clés primaires [Jiang and Naumann, 2020]. Nous adaptons ces caractéristiques aux BD orientées documents et nous avons introduit de nouvelles caractéristiques. Ces caractéristiques sont définies comme suit :

- Cardinalité : en pratique, les concepteurs de schémas montrent une tendance à utiliser moins d'attributs pour la définition de l'identifiant : moins d'attributs permettent une meilleure compréhension et maintenabilité. La fonction du score est définie par $\frac{1}{|IDc|}$, où IDc est un identifiant candidat.

- Suffixe/préfixe de nom : les identifiants sont généralement identifiés par leur suffixe/préfixe de noms. Nous considérons cette liste des suffixes/préfixes possibles pour les identifiants : "id", "key", "nr", "no", "pk", "num" et "code". La fonction du score est définie par $\frac{prefixSuffix(IDc)}{|IDc|}$, où $prefixSuffix(IDc)$ compte le nombre d'attributs dans IDc dont le nom contient l'un des préfixes/suffixes mentionnés ci-dessus.

- Longueur de la valeur : les attributs qui sont utilisés comme *identifiants* sont supposés avoir une longueur de valeur courte, car généralement ils sont des *identifiants* non sémantiques. La fonction du score est définie par $\frac{1}{max(1,\overline{LengthMax(f_i)}-n)}$ où
  - $\overline{LengthMax(f_i)}$ est la longueur moyenne des valeurs les plus longues associées aux attributs de IDc. Cette fonction est définie par $\overline{LengthMax(f_i)} = \frac{1}{|IDc|}\sum_{i=1}^{|IDc|} LengthMax(f_i)$.
  - n est le paramètre utilisé pour pénaliser les valeurs longues.
- Profondeur : Les identifiants ont souvent une profondeur faible. En fait, des attributs imbriqués ont moins de chance d'être un *identifiant* pour toute la collection. Nous définissons la fonction du score comme suit : $\frac{1}{|IDc|}(\sum_{i=1}^{|IDc|}\frac{1}{depth(f_i)+1})$, où $f_i \in IDc$.
- Type de données : Dans la pratique, un attribut est susceptible d'être un *identifiant* si son type de données est *Integer* ou *String*. La fonction est définie par $\frac{1}{|IDc|}(\sum_{i=1}^{|IDc|} type(f_i))$ où $type(f_i)$ est une fonction binaire qui renvoie un si l'attribut a un type *String* ou *Integer* ou zero dans le cas contraire.

Après avoir associé un score à chaque identifiant candidat et vu que la liste initiale est large, nous la filtrons en utilisant une technique d'élagage appelée *Cliff* [Jiang and Naumann, 2020].

L'objectif de la deuxième étape est de trouver les paires (identifiant, référence) entre chaque deux collections, en se basant sur l'ensemble des identifiants candidats fournis par la phase précédente. Étant donné deux collections, nous cherchons les paires candidates dans les deux sens : *(i)* pour chaque identifiant $IDc(C_1)$ de la première collection C_1, nous cherchons la référence la plus similaire, si elle existe, dans la liste des attributs $F(C_2) = f_1, \ldots, f_n$ de la deuxième collection C_2 ; et *(ii)* pour chaque identifiant $IDc(C_2)$ de la deuxième collection C_2, nous cherchons la référence la plus similaire, si elle existe, dans la liste des attributs $F(C_1) = f_1, \ldots, f_n$ de la première collection C_1.
Le degré de similarité dépend des valeurs des propriétés communes entre un identifiant et sa référence. Afin d'examiner ces similitudes, nous modélisons la relation entre $IDc(C_i)$ et $F(C_j) = f_1, \ldots, f_n$ sous forme de graphe.

Les techniques basées sur les graphes sont devenues omniprésentes car elles sont essentielles pour l'extraction des connaissances [Cui et al., 2019]. Cependant, les graphes avec leur représentation traditionnelle ne permettent pas de tirer pleinement parti des approches et des techniques existantes de

l'apprentissage automatique. De nos jours, il y a eu un intérêt considérable pour le *graph embedding* qui vise à associer à chaque sommet du graphe une représentation vectorielle [Cai et al., 2018]. Récemment, plusieurs techniques de *graph embedding* ont été proposées tel que l'algorithme *node2vec* [Grover and Leskovec, 2016]. Nous utilisons ce dernier qui effectue une représentation locale du voisinage des sommets d'un graphe. Le graphe que nous avons défini est simple[8], hétérogène[9], non orienté et non pondéré. Nous le notons par *G(V,E)* où :

- $\boldsymbol{V}$**=** $\{IDc(C_i), F(C_j), PV\}$: l'ensemble des sommets qui sont de trois types :
  - $IDc(C_i)$: un identifiant candidat de la collection C_i
  - $F(C_j)$: l'ensemble des attributs de la collection C_j
  - PV : l'ensemble des propriétés communes entre les identifiants et les références. Par exemple *Integer* est une valeur de la propriété *type de données*.
- $\boldsymbol{E}$: l'ensemble des arêtes représentant les relations entre les sommets. Une arête peut être établie entre *(i)* un sommet d'un identifiant $IDc(C_i)$ et un sommet d'une référence candidate de $F(C_j)$ s'il existe une similarité syntaxique ou sémantique ; et *(ii)* un attribut dans $\{IDc(C_i), F(C_j)\}$ et une valeur de propriété.

Nous proposons un ensemble de règles pour identifier les propriétés liées aux identifiants et références :

- Compatibilité des types de données : l'identifiant, et sa référence doivent avoir le même type ou des types compatibles. Par exemple, si nous avons un identifiant de type *String* et une référence avec un type *Integer*, et ils ne sont pas convertibles, cette paire ne sera pas considérée comme candidate dans ce cas.
- Similarité syntaxique : nous utilisons la mesure de similarité syntaxique *Fuzzy-Token* [Wang et al., 2011]. La similarité combine à la fois la similarité basée sur les tokens et la similarité des chaînes de caractères. Pour utiliser cette fonction de similarité, les chaînes d'entrée n_1 et n_2 sont divisées en tokens. Nous considérons les deux cas pour la division : *(i)* avoir un délimiteur, par exemple, "_" et / ou une lettre majuscule; *(ii)* les chaînes sont attachées sans délimiteur, par exemple "LINESTATUS".

[8] sans arête multiple
[9] avec des sommets de différents types

- Similarité sémantique : l'utilisation de la similarité syntaxique uniquement ne suffit pas pour couvrir tout les cas possibles, par exemple "Customer" et "Client". Par conséquent, nous proposons une étape pour mesurer de la similarité sémantique. Pour ce faire, nous utilisons la base de données lexicale *Wordnet*.

Comme preuve de concept de notre approche, nous avons développé des prototypes java des phases principales de notre approche afin d'expérimenter nos contributions. Ces prototypes fournissent l'ensemble des mécanismes permettant de: *(i)* extraire le schéma d'une collection de documents JSON contenant la liste des attributs avec leurs types ; et *(ii)* détecter les paires candidates d'attributs clés de jointure (identifiant, référence) qui sont relatives à chaque deux collections de documents JSON. Notre étude expérimentale est basée sur deux benchmarks (TPC-H et TPC-E) et deux sources de données réelles (Twitter et Musicians).

Les expérimentations que nous avons menées visent à valider notre approche, en termes de pertinence des résultats. La validation de l'approche implique les deux niveaux : *(i)* détection des identifiants candidats ; et *(ii)* identification des paires candidates des attributs clés. Pour chaque niveau, nous avons utilisé quatre métriques : precision, recall, accuracy et percentage decrease. Les résultats de l'évaluation de notre approche donnent un aperçu de la faisabilité de la détection des attributs clés de jointure dans les bases de données orientées documents contenant des données dispersées sur plusieurs collections.

Les travaux présentés dans ce mémoire de thèse se situent dans le contexte de l'informatique décisionnelle avec l'objectif d'exploiter des données sans schéma et dispersées sur plusieurs sources NoSQL pour la prise de décision.
Nous avons présenté une nouvelle approche qui vise à extraire, transformer et charger des sources de données NoSQL à la demande. Cette approche est hybride qui prend en compte à la fois la nature sans schéma des sources de données et les besoins décisionnels afin d'explorer plusieurs bases de données NoSQL de manière efficace. Nous nous sommes concentrés, en particulier, sur les bases de données orientées documents et avons abordé l'impact de la nature sans schéma et de l'hétérogénéité de ces données. Contrairement aux travaux existants, notre approche ne se limite pas à une seule collection, car nous considérons la dispersion des données sur plusieurs collections.

Dans les systèmes décisionnelles, l'extraction de données pertinentes qui répondent aux exigences du décideur nécessite souvent d'accéder à plusieurs collections de documents JSON. Cet accès multiple augmente la complexité puisqu'il nécessite de trouver les attributs de liaison "pivot". Par conséquent, nous étudions le problème de la découverte automatique des attributs de jointure. Nous proposons un algorithme qui vise à détecter automatiquement à la fois les identifiants et les références à partir de plusieurs bases de données orientées documents. Notre approche met en évidence trois étapes principales : *(i)* Extraction de schémas ; *(ii)* La découverte d'identifiants candidats ; et *(iii)* Identifier les paires (identifiant, référence) candidates. L'approche est basée sur des caractéristiques et des règles fondées sur les similarités syntaxique et sémantique pour trouver les identifiants candidats (identifiant, référence) entre chaque deux collections.

Les résultats des recherches que nous avons menées pourraient être utiles pour :

- Les systèmes Business Intelligence & Analytics en particulier les processus ETL traitant des bases de données orientées documents : aujourd'hui, l'exploitation des données NoSQL pour des fins analytiques manque de maturité, de traçabilité et de gestion des métadonnées. Dans le cadre de la BI&A, les données sont souvent éparpillées sur plusieurs collections de documents. Par conséquent, l'extraction des données pertinentes qui correspondent aux besoins des décideurs nécessite souvent d'accéder à plusieurs sources de données, d'où l'utilisation de l'opération de jointure. En effet, les outils ETL open source et commerciaux offrent des composants de jointure. Cependant, il appartient à l'utilisateur de choisir les clés avant d'appliquer la jointure. Dans un contexte NoSQL, un document peut avoir des centaines d'attributs ce qui rend difficile, voire infaisable l'identification manuelle de l'identifiant d'une collection.

- Interrogation des données : la jointure est une opération primordiale pour l'interrogation des données. Par conséquent, il est obligatoire d'avoir les clés de jointure au préalable. À titre d'exemple, MongoDB utilise le MQL (MongoDB Query Language) pour interroger les données stockées dans des bases de données orientées documents. Les développeurs confirment que les opérations de jointure entre collections sont essentielles [Celesti et al., 2019]. La communauté open source de MongoDB a récemment introduit l'opérateur $lookup en version 3.2. Cet opérateur effectue une jointure entre deux collections en utilisant un localField et un foreignField spécifiés par l'utilisateur.

- Conception de bases données : identifier les interrelations dans les bases de données héritées nécessite certainement la découverte de clés de jointure. Même si l'absence de schéma rigide caractérise les bases de données NoSQL, les développeurs devraient également avoir une vue d'ensemble des données et prendre les mesures et décisions appropriées pendant le processus de conception de l'application. Ces décisions peuvent avoir un effet significatif sur l'efficacité de l'application et la lisibilité du code [Mior and Salem, 2018].

Les travaux menés dans cette thèse peuvent être poursuivis dans différentes directions :

- En premier lieu, nous avons l'intention d'étudier l'impact du rafraîchissement incrémental des données [Qu and Deßloch, 2017] et comment planifier en conséquence le processus de découverte des identifiants et des références dans les bases de données orientées documents (cf. Chapitre 4). Dans notre approche, la découverte des identifiants et des références est traitée lorsque l'utilisateur est hors ligne (traitement à froid). Si de nouvelles modifications sont produites au sein d'une ou plusieurs collections qui sont déjà traitées, le processus est relancé afin d'identifier les nouveaux identifiants et références, s'ils en existent. Le traitement étant effectué "à froid", la difficulté de ce processus n'impacte pas l'utilisateur ; néanmoins, nous considérons le traitement "à chaud" comme l'un de nos objectifs futurs.
  En second lieu, nous envisageons d'intégrer notre approche en tant que composant dans un outil ETL commercial, par exemple, Talend Big Data et Talend Data Integration.

- En ce qui concerne notre approche globale (cf. Chapitre 3), en premier lieu, nous envisageons de l'enrichir en y intégrant une phase de modélisation multidimensionnelle basée sur plusieurs collections. Depuis plusieurs décennies, OLAP est largement utilisé comme approche d'analyse de données pour aider à la prise de décision sur des données structurées. Avec l'utilisation répandue des bases de données NoSQL contenant des données semi-structurées, il est également de plus en plus nécessaire d'appliquer OLAP sur ces bases de données, afin d'aider les utilisateurs, qui ne sont pas experts, à prendre de meilleures décisions. Les travaux existants se concentrent sur l'extraction d'un schéma multidimensionnel à partir d'une seule collection [Chouder et al., 2019, Gallinucci et al., 2019], alors que dans BI&A, la récupération des données pertinentes nécessite souvent d'accéder à plus d'une collection de documents JSON.
  En second lieu, nous avons l'intention d'étendre notre approche pour supporter des modèles

de sources de données NoSQL supplémentaires puisque notre objectif ultime est de fournir une approche BI&A dédiée à NoSQL.

Chapter 1

Context and Challenges

Contents

1.1 Context

In today's world, data play a valuable role in organizations. Most notably, the volume and variety of gathered data have expanded dramatically. Analyzing these data aids in gaining insights of significant worth that bolster decision-making. This is ensured through Business Intelligence & Analytics (BI&A) systems. The term Business Intelligence & Analytics refers to a wide range of applications and technologies that collect, manage, and analyze data to assist the end-user in making better decisions [Watson, 2009].

In traditional BI&A architecture, data are extracted from heterogeneous sources of different types and formats. These scattered data are transformed using the Extract-Transform-Load (ETL) process. The latter performs three basic functions: *(i)* data extraction from data sources; *(ii)* data transformation where the data are converted and structured for decision-making (e.g., data cleansing, aggregation, reformatting, matching, etc.); and *(iii)* the resulting dataset is loaded into the target system, typically a Data Warehouse (DW) [Kimball and Ross, 2002, Vassiliadis, 2009].

With the emergence of big data, data management systems based on non-relational models such as key-value, column-oriented, document-oriented, and graph-oriented are widely accepted as efficient systems to store NoSQL data. In fact, NoSQL adopt a schemaless representation: the schema is flexible, and is not predefined with upfront constraints. NoSQL data models offer greater flexibility over data types and heterogeneity consisting in schema variants and different possible structures [Rizzi, 2022, TruicÄ et al., 2021]. However, in operational applications, the absence of unique schema grants flexibility but adds complexity to analytical applications [Golfarelli and Rizzi, 2018].

Data analytics are gaining increasing attention in BI&A solutions that become data-driven rather than business and intuition driven [McAfee and Brynjolfsson, 2012]. Without delving into these data, bias and false assumptions may lead to poor decision-making. Consequently, there is an obvious need to be able to cope with high technology standards to face big data-specific complexity (Volume, Variety, Velocity, etc.) [Bizer et al., 2011, Chen et al., 2012].
Exploiting NoSQL data for decision-making lacks maturity, governance, traceability, and metadata management. Therefore, it is essential to provide a dedicated comprehensive BI&A approach that starts from NoSQL sources, extracts, transforms, and loads the data into the DW, and provides the means required for decision-makers to analyze this data quickly. Other issues related to data arise, such

as data quality, data understanding, and availability [Kim et al., 2018, Thota, 2017]. These problems related to data will generate performance degradation going against the expected reactiveness. Indeed, proceeding as in traditional BI solutions by loading all the data before analysis is rather unfeasible as data is leading to performance degradation. Moreover, data evolve and are highly heterogeneous. This means that some of the data may be useless or irrelevant to the analysis needs. Based on the above observations, an ETL framework needs to be on-demand regarding the data streams it has to consider [Baldacci et al., 2017].

1.2 Background

This section presents the key concepts that form the Business Intelligence foundation. To begin with, we define Business Intelligence & Analytics with an overview of its typical architecture, detailing the main parts, mainly the data sources, the ETL process, the Data Warehouse, and the OLAP analysis phase. Secondly, we present the basic concepts related to big data and NoSQL stores.

1.2.1 Business Intelligence & Analytics

Business Intelligence & Analytics is defined as "a broad category of applications, technologies, and processes for gathering, storing, accessing, and analyzing data to help business users make better decisions" [Watson, 2009]. As depicted in Figure 1.1, BI&A systems use data gathered from company internal operational databases and external data. These data are transformed and integrated before being loaded into the Data Warehouse. The relevant data for the decision-making process are collected from data sources using the ETL process [Vassiliadis, 2009].

Data Sources

Data sources are heterogeneous at different levels, including data type, format, syntactic, and semantic aspects. Data can be ambiguous and of poor quality due to missing values, high data redundancy, and lack of truthfulness.
The sources from which data are extracted, transformed, and loaded into the Data Warehouse can be:

- **Internal**: data are generated by the company, namely transactional databases about sales, human

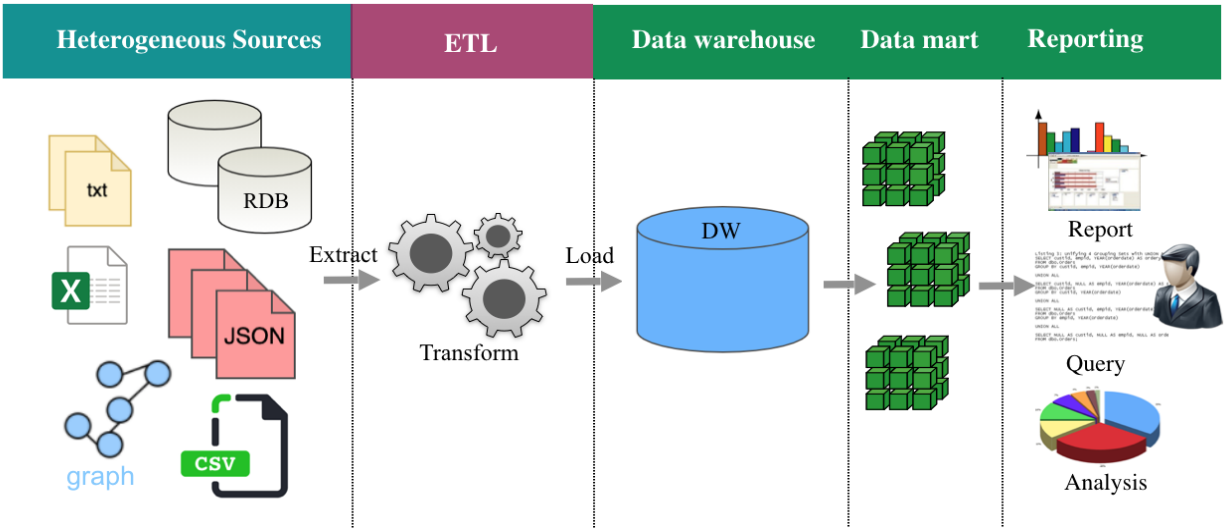

Figure 1.1: Business Intelligence & Analytics system architecture

resources, etc.

- **External**: data are generated from outside the company, namely websites, sensors, emails, etc.

ETL Process

The ETL process is the core stage in a BI&A architecture. It is a three-fold process that performs these basic functions in data movement:

- Extract: before moving data to a new destination, it is mandatory to retrieve these data from data sources that are typically heterogeneous.
- Transform: this step deals with data structuring and formatting. Data are converted to be stored in the proper format or structure for querying and analysis (e.g., data cleansing, reformatting, matching, standardizing, aggregation, etc.)
- Load: the transformed data are loaded into the target system, typically a Data Warehouse.

Many reasons stand behind the need for the data integration phase within the decision system: (i) heterogeneous formats; (ii) data formats can be difficult to be interpreted or ambiguous; (iii) legacy

systems using obsolete databases, and (iv) data source's structure is changing over time [Bodziony et al., 2022, Theodorou et al., 2016]. All these features of data sources make data quality uncertain. A variety of studies were conducted to identify different quality issues within the data integration process. Most of these studies agree that data quality faces numerous challenges [Batini et al., 2009, Warth et al., 2011]. Indeed, ETL is a crucial part of the data warehousing process where most data cleansing and curation are carried out. Hence, we propose a classification of typical data quality issues according to ETL stages, i.e., extract, transform and load. As depicted in Table 1.2, each one of these stages is prone to different quality problems at both schema and instance levels.

Even if many efforts have been made to deal with data quality issues in the traditional ETL process, there are still many new challenges to overcome. Indeed, extending existing ETL solutions is very important to cope with the volume, variety, and velocity of nowadays data.

A global analysis of the quality defects observed at the instance level in Table 1.2 shows that most of the problems have solutions at the schema level in traditional relational databases through, for example, constraints definition. The defects at the schema level also show the importance of a schema vision over the data to firstly help to identify the defects and secondly to solve them.

Table 1.1: Examples of ETL data quality problems

		Problem	Description
E	**Schema**	Lack or absence of integrity constraints	Rules that define the consistency of a given data or dataset in the database (e.g., key and referential integrity constraints, uniqueness). Example of uniqueness violation: Two customers having the same SSN number customer 1= (name="John", SSN="12663"), customer 2= (name="Jane", SSN="12663").
		Poor schema design	Imperfect schema level definition [Debbarma et al., 2013, Rahm and Do, 2000]. Example 1: Attributes names are not significant: "FN" stands for First Name and "Add" stands for Address. Example 2: Source without a schema: "John;Doe;jd@gmail.com;USA"
		Embedded values	Multiple values entered in one attribute [Debbarma et al., 2013]. Example: name=" John D., Wisconsin Ave NW, Washington, DC 20036, US".
	Instance	Duplicate records	Duplicate records means two records that inadvertently share the same data [Boufares and Salem, 2012]. We distinguish two types of duplication: • Different identifiers referring to the same entity: In this case, records contains same terms that are expressed differently. In fact, misspellings and different ways of writing names can all lead to duplicate entries [van Gennip et al., 2018]. For instance, a phone number can be written in different format: Local phone number (e.g., 458-7801), or international phone number (+1-212-458-7801 or (212)-458-7801) • Same identifier referring to different entities: Another form of duplication is the conflicts of entities when inserting a record having the same id as an existing record [Yang et al., 2017].
		Missing values	Missing values are classified into two types: Data in one field appear to be null or empty [Gill and Singh, 2014, Yang et al., 2017] (i.e., Direct incompleteness), and missing values caused by data operations such as update (i.e., Indirect incompleteness).

Table 1.2: Examples of ETL data quality problems

		Problem	Description
T	**Schema**	Variety of data types	Different data types between the source and the target schema. Different data types for the same field.
		Naming conflicts	If we have two data sources that have two synonymous attributes (e.g., Gender/Sex), then the union of the sources mentioned above requires schema recognition [Gill and Singh, 2014, Rahm and Do, 2000, van Gennip et al., 2018].
	Instance	Syntax inconsistency (Structural conflicts)	The data retrieved from the source have not the same format as the data of the Warehouse [Yang et al., 2017]. There are different syntactic representations of attributes whose type is the same [Boufares and Salem, 2012]. Example 1: French date format (i.e., dd/mm/yyyy) is different from that of the US one (i.e., mm/dd/yyyy). Example 2: Gender attribute is represented differently in the two data sources, e.g., 0/1, F/M.
L		Wrong mapping of data	Linking a data source to the incorrect destination results in the spread of inaccurate data.
		Wrong implementation of the slowly changing dimension	Slowly changing dimension is a task used to track both current and historical data in the Data Warehouse due to versioning of data after every load and update operation [Gill and Singh, 2014]. It is critical to reflect the dimensions attributes changes without affecting the Data Warehouse consistency and traceability.

Data Warehouse

Data Warehouses stand as the cornerstone of BI&A systems. Inmon [Inmon, 1992] defines the DW as "a subject-oriented, integrated, time-variant, non-volatile collection of data in support of management's decision-making process."

Data Mart

A data mart is the access layer of the data warehouse environment that is used to get data out to the users. The data mart is a subset of the data warehouse and is usually oriented to a specific business line or team.

OLAP

OLAP(On-Line Analytical Processing) systems allow decision-makers to improve their management by consulting and analyzing aggregated multidimensional data, i.e., Data cubes.

1.2.2 Big data and NoSQL Stores

The exponential growth of digital data brings up massive big data datasets [Dhaouadi et al., 2022]. Processing these data in a reasonable amount of time using traditional methods is difficult or even infeasible. Indeed, several basic rules are used to define big data and their characteristics in terms of properties number called V's. The number of V's has evolved from 3 V's in 2001 to 17 V's in 2017 [Panimalar et al., 2017]. Each rule is defined by a set of properties characterizing big data.

- **3 V's rule**: this rule was initially introduced by Doug Laney[1] in 2001. In the description, Big data refer to high **volume**, **velocity** and/or **variety** (3 V's) of data resources that require cost-effective, ingenious data processing to empower enhanced insights, decision-making and process automation [Laney, 2001].
- **4 V's rule**: the *International Data Corporation* (IDC) has described Big data as a new generation of technologies and architectures, designed to economically extract **value** from very large **volumes** of wide **variety** of data, by enabling high **velocity** capture, discovery and/or analysis [Balti et al., 2021, Gantz and Reinsel, 2011].

[1]Former Distinguished Vice President and Analyst with Gartner's Chief Data Officer Research team [Laney, 2001]

- **5 V's rule**: Two new V's appeared in [Demchenko et al., 2013] where the new definition refers to Big data as: "technologies aim to process high-**volume**, high-**velocity**, high-**variety** data (sets/assets) to extract intended data **value** and ensure high-**veracity** of original data and obtained information that require cost-effective, innovative forms of data and information processing (analytics) for enhanced insight, decision making, and processes control".

  The five Big data properties used in the above defined rules are as follows:

  - **Volume**: refers to the tremendous amount of the generated data from different sources.
  - **Velocity**: refers to the rate at which the massive volume of data is created, gathered, and analyzed.
  - **Variety**: refers to the variety of data formats that can be structured, semi-structured or unstructured.
  - **Veracity**: refers to the quality and credibility of the collected data.
  - **Value**: refers to how useful is the gathered data for decision-making.

These characteristics of big data entail several strategic stakes:

- makes the best use of this valuable data to extract the foremost possible information for decision-making.
- better managing the storage in databases to make them highly performant and easy to use.

Hence, to suit the data volume, velocity, and variety, NoSQL stores provide new cost-effective and schema-free systems that are based on four families:

- **Key-value oriented model**: data are stored in the form of key/value pairs where each value is associated with a unique key. Each key allows only strings, while its value can be String, JSON, XML, etc. In general, key-value stores have no query language. It uses the put, get and delete commands to store, retrieve and update data [Angadi et al., 2013].
- **Column-oriented model**: wide column data stores combine some features of relational databases with the key-value model. Data are stored in column families as rows with several columns connected with a row key. Rows can contain different columns which are the fundamental data

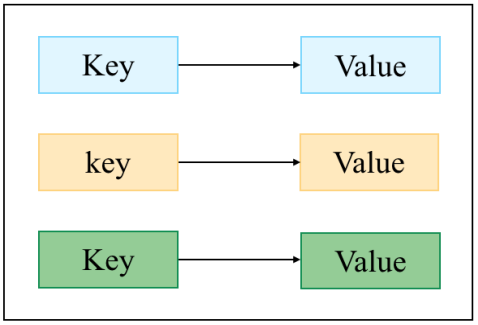

Figure 1.2: Structure of the key-value model

structure unit. Each column holds three values: key name, timestamp, and value [Kuznetsov and Poskonin, 2014]. Unlike relational databases where the schema is fixed, the columns in rows can be deleted or added at will.

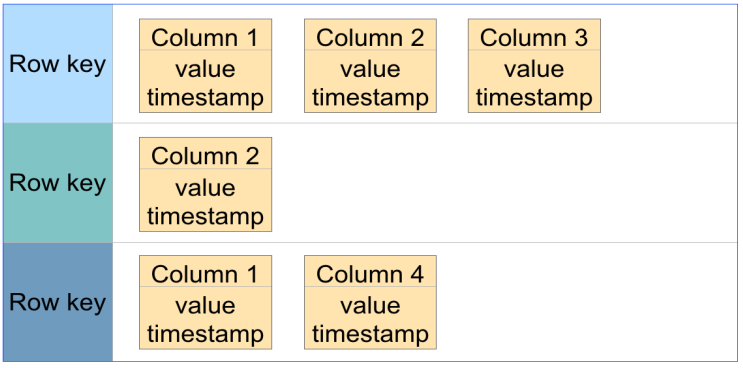

Figure 1.3: A column family in a column-oriented database

- **Graph-oriented model**: data are organized into entities and relationships. The entity is represented as a node. A relationship between two nodes is represented as an edge. This model is dedicated to storing and querying tightly connected data. Graph databases are advantageous for use cases where links between data must be created and easily queried such as recommendation engines, social networking, and fraud detection.

- **Document-oriented model**: document stores are an evolution of key-value stores where values are no longer opaque to the system: they are generally stored in JSON-like document format with a mandatory unique identifier. Data are organized into collections of documents where

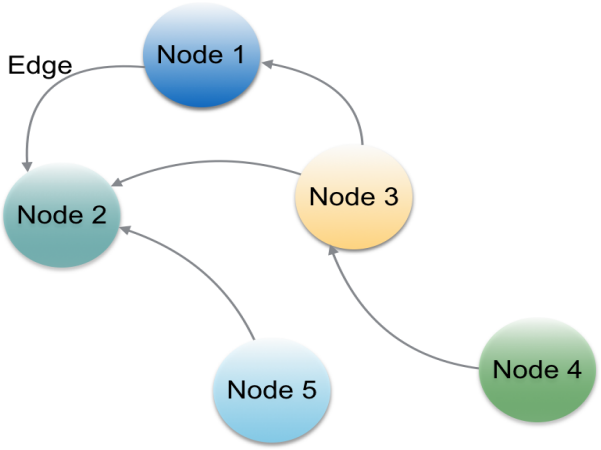

Figure 1.4: Structure of the graph-oriented model

each document contains a set of key-value pairs. Document stores generally allow the definition

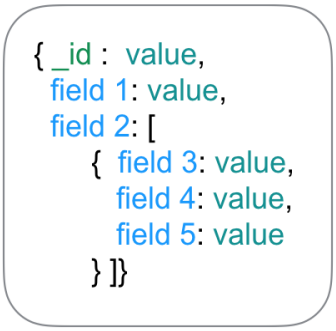

Figure 1.5: Structure of the document-oriented model

of indexes on document fields. This comes with much richer querying capabilities than key-value stores. Like other NoSQL models, documents have a flexible schema, which is not defined beforehand, thereby allowing fast storage and retrieval of vast collections of documents. Within the same collection, documents can be of different structures. This model is increasingly used in the industrial world. For instance, the SGBD MongoDB has exceeded 26.2% as a rate of use in companies in 2021[2].

[2] https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba

1.3 Challenges and Research Questions

For more than a decade, NoSQL datastore became commonly used to store big data. These systems are schema-free and built upon distributed systems, making them easy to scale and shard. They provide data storage and retrieval methods that differ from relational databases. Better performance scaling and no requirement for a unique schema are among the potential advantages of NoSQL stores [Rizzi, 2022]. However, in the rush to solve the challenges of big data and large numbers of concurrent users, NoSQL abandoned some of the core features of relational databases, which make them highly scalable and easy to use Abdelhédi et al. [2021], Mali et al. [2020]. Unlike relational databases, the NoSQL model is designed to eschew constraints and schemas: adopt a schemaless representation where the schema is not fixed beforehand and without integrity constraints. However, the schemaless nature adds complexity to data integration and analytical application as it amplifies quality problems within these databases.

Although the use of NoSQL systems is widely accepted today, Business Intelligence & Analytics wields relational data sources [Ram et al., 2016]. In fact, from the earliest days of data warehousing, the qualities of the relational model have been highly valued in the quest for data consistency and quality that are ensured through the structured and rigid data model.

The accuracy and relevance of Business Intelligence & Analytics rely on bringing high data quality to the data warehouse from both internal and external sources using the ETL process. The latter is complex and time-consuming as it manages data with heterogeneous content and diverse quality problems. Ensuring data quality requires tracking quality defects along the ETL process. Authors in [LaValle et al., 2011] conducted a study based on 3 000 business executives and managers. This survey showed that 50% of the respondents consider improving data management and BI&A as their priorities. It also revealed that 20% of them cited concerns with data quality as a primary obstacle to BI&A systems.

Data analytics has been used for many years to support business decision-making. Several studies uphold that poor data quality has direct and indirect impacts on the underlying business decisions [Batini et al., 2009, Warth et al., 2011]. According to Redman [Redman, 1998], at least three proprietary studies have provided estimates of poor data quality costs between 8 and 12% of the revenue range.

The problem of quality in the ETL process still requires a lot of effort, mainly if the BI&A system is

based on non-relational data sources as NoSQL databases [Abdelhédi et al., 2017]. Today, data management systems based on non relational models as key-value, column-oriented, document-oriented, and graph-oriented, which adopt a schemaless representation are widely accepted as efficient systems to store data. Therefore, exploiting varied and voluminous data for decision-making without a previously defined schema and often without integrity constraints requires reviewing all the phases of the BI&A architecture, particularly the ETL process [Theodorou et al., 2017]. There has been a growing interest in volume and velocity, however, handling variety issues are similarly important and needs more efforts in ETL over NoSQL stores. It is essential to provide a dedicated BI&A capable of extracting, transforming, and loading heterogeneous data as document stores, the ones in which we are particularly interested in our work. Therefore, we are particularly interested in the document-oriented model.

Our research questions are as follows:

- How to leverage multiple document stores for decision-making?
  Today, exploiting NoSQL document stores for decision-making lacks maturity, traceability, and metadata management. Therefore, the idea is to review all the phases of the BI&A architecture, particularly the ETL process, to provide solutions capable of exploiting document stores for decision-making. It is essential to provide a dedicated comprehensive BI&A approach that starts from document stores, extracts, transforms, and loads the data into the DW, and provides the means required for decision-makers to easily analyze this data.

- How to extract, transform and load scattered data over multiple document stores?
  In a BI&A architecture, the main challenge when dealing with schemaless data, as document stores, is extracting data that is dispersed across multiple document stores. This challenge leads to the question of how to identify joinable fields from several collections without integrity constraints. In fact, fetching relevant data that meet the decision-makers requirements often needs to access more than one document store. While joining tables in relational data sources is straightforwardly owed to the availability of a precise join key, in document stores, collections are a long way away from having defined join keys because of the absence of integrity constraints. So, identifying the ”joinable” fields to stick to two document stores is a tricky challenge. Despite its importance, no previous work has paid close attention to detecting join key pairs in NoSQL

stores, particularly in document-oriented stores.

1.4 Contributions

To answer the research questions mentioned above, we provide the following contributions:

- In a first step, we introduce a global BI&A approach that aims to extract, transform, and load document stores in an on-demand fashion. This approach is hybrid as it considers both data sources' schemaless nature and analytical needs to explore several document stores efficiently. Unlike existing works, our approach is not limited to one collection, as we consider the dispersion of data across several collections.

- Secondly, we shed light on the ETL process, where dealing with the extraction of dispersed data across several document stores, which did not have integrity constraints, constitutes a challenge. Thus, we study the problem of discovering join fields to automatically detect both *identifiers* and *references* on several document stores. The *modus operandi* of our approach underscores three core stages: *(i)* schema extraction; *(ii)* discovery of candidate *identifiers*; and *(iii)* identifying candidate pairs of *identifier* and reference fields. We use scoring features and pruning rules to discover actual candidate *identifiers* from many initial ones efficiently. To find candidate pairs between several document stores, we put into practice *node2vec* as a graph embedding technique, which yields significant advantages while using syntactic and semantic similarity measures for pruning pointless candidates.

- To validate our contributions, we have developed a prototype to support the main phases of our approach. The validation comprises both levels: *(i)* candidate *identifiers* discovery for each collection; and *(ii)* identification of candidate pairs of key fields (*identifier* and *reference*) for every two collections. We base our experimental study on two benchmarks: TPC-H and TPC-E, and two real-world datasets: Twitter and Musicians. We report the results that show the feasibility of our approach and discuss the challenges to be addressed in our future work.

1.5 List of Publications

- **Manel Souibgui**, Faten Atigui, Saloua Zammali, Samira Si-Said Cherfi, Sadok Ben Yahia: Data quality in ETL process: A preliminary study, in: Proceedings of the 23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems **KES 2019**: 676-687 [Souibgui et al., 2019] (Rank B).

- **Manel Souibgui**, Faten Atigui, Sadok Ben Yahia, Samira Si-Said Cherfi: Business Intelligence and Analytics: On-demand ETL over Document Stores, in: Proceedings of the 14th International Conference on Research Challenges in Information Science **RCIS 2020**: 556-561 [Souibgui et al., 2020] (Rank B).

- **Manel Souibgui**, Faten Atigui, Sadok Ben Yahia, Samira Si-Said Cherfi: IRIS-DS: A New Approach for Identifiers and References Discovery in Document Stores, in: Proceedings of the 54th Hawaii International Conference on System Sciences **HICSS 2021**: 1-10 [Souibgui et al., 2021] (Rank A).

- **Manel Souibgui**, Faten Atigui, Sadok Ben Yahia, Samira Si-Said Cherfi: An embedding driven approach to automatically detect identifiers and references in document stores. Data & Knowledge Engineering Journal **DKE 2022**, Volume 139, 102003 [Souibgui et al., 2022] (Quartile: Q2, Impact Factor: 1.99)

1.6 Dissertation Outline

This thesis is organized as follows:

Chapter 2 scrutinizes the related literature. We survey existing works that paid attention to how to exploit NoSQL stores for decision-making? and how to prepare these data to fit the analysis requirements? Hence, we identify three major streams of approaches: *(i)* the first one dealt with the ETL process over NoSQL stores; *(ii)* the second one treated the schema matching issues; and *(iii)* the third stream of approaches have tackled the OLAP analysis over document stores.

Chapter 3 presents an overview of our approach that aims to extract, transform and load NoSQL data sources, particularly document stores, on-demand. Then, we introduce the general architecture of the global approach with the different BI&A phases. This approach is hybrid as it considers both data sources' schemaless nature and decision-makers needs to explore several document stores efficiently. We mainly focus on the ETL phase starting from document-oriented sources, as ETL is one of the most complex phases, on which strongly depends the success of the BI&A project.

Chapter 4 details our main contribution IRIS-DS to automatically discover identifiers and references in document stores as a fundamental step to prepare the ETL process. The works presented in this chapter represent a crucial component in the overall architecture presented in Chapter 3.

Chapter 5 reports our experimental findings. The validation of our contributions comprises both levels: *(i)* candidate *identifiers* discovery for each collection; and *(ii)* identification of candidate pairs of key fields (*identifier* and *reference*) for every two collections. In this chapter, we, firstly, describe the data collection and preparation. We base our experimental study on two benchmarks: TPC-H and TPC-E, and two real-world datasets: Twitter and Musicians. Then, we describe the evaluation protocol, and report the experimental results, which prove the worthiness of our contributions.

The dissertation concludes by summarizing our contributions and sketching possible future directions.

Chapter 2

State of the Art

Contents

2.1 Introduction

In this chapter, we survey existing works that paid attention to how to exploit NoSQL stores for decision-making and how to prepare these data to fit the analysis requirements. We identify several major streams of approaches. Firstly, in Section 2.2, we reviewed the approaches that dealt with NoSQL data integration, including ETL process over NoSQL stores and NoSQL schema extraction. Indeed, the main challenge when dealing with schemaless data is extracting data that is dispersed across multiple NoSQL stores. This challenge leads to the question of how to identify joinable fields from several collections without integrity constraints. Thus, we, secondly, present the approaches that dealt with the problem of joining dispersed data across NoSQL stores, joinable table discovery in tabular data (cf., Section 2.3), and the approaches that have treated the ontology and the schema matching issues (cf., Section 2.4). Furthermore, in Section 2.5, we review the approaches that tackled the OLAP analysis over NoSQL stores in the context of self-service Business Intelligence.

2.2 NoSQL Data Integration

In this Section, we, firstly, scrutinize the works regarding the ETL process over NoSQL stores. Since NoSQL stores are schema-free, we, additionally, scrutinize the related works that have proposed approaches to extract the schema from these sources. Finally, we present the works addressing the problem of joining dispersed data across several NoSQL stores.

2.2.1 ETL over NoSQL Stores

Few researchers have addressed the problem of ETL in the context of NoSQL stores [Dumin Sahiet, 2015, Yangui et al., 2017]. Most of the existing works have proposed contributions to improve the performance of the ETL process. As the problem of performance is not the core of our work, we present, in this section, four representative works.

In [Ali, 2018], the authors conducted a thorough investigation of the existing methodologies for designing, developing, and optimizing ETL workflows in a big data context. They highlight that there is a very limited support for semi-structured and unstructured data, nevertheless, the variety of data formats is expanding rapidly. Designing ETL process for big data is challenging since typical ETL operators are not adequate for handling large amounts of data. For this, the authors proposed a

theoretical ETL framework that allows the developer to expand the functionality of an ETL tool with new kinds of cleaning operators, as outlier detection or de-duplication, to fit these data.

In [Mehmood and Anees, 2022], the authors proposed a real-time ETL architecture for unstructured data. The main goal is to accelerate stream-disk joins, which almost require frequent disk access, during the transformation phase in the ETL process. Thus, they attempt to overcome the problems of disk overhead and stream data loss. In distributed disk-data, they perform stream-disk join by combining data from stream and disk data based using a common attribute that is defined by the user.
In [Mallek et al., 2020], the authors proposed a tool called *BigDimETL*, that extracts, transforms, and load NoSQL stores. This tool aims to minimize ETL time consuming by parallelizing the treatment of *select*, *project*, and *join* operations. The authors point out that document stores as JSON, have a complex structure to be handled in the ETL process. Hence, they convert the input data extracted from a document store into a column-oriented model in order to apply partitioning techniques.
In [Kathiravelu et al., 2018], the authors proposed a hybrid ETL approach that combines eager and lazy ETL. Eager ETL is the traditional ETL process, whereas lazy ETL processes data only when necessary. The approach aims to reduce the execution time for repetitive scientific research queries by putting away the previously integrated and loaded data into an integrated data repository.

In the literature review presented in [Petricioli et al., 2021], the authors underlined that some glaring issues regarding NoSQL technologies used in the decision support systems still require more effort. They draw attention to the fact that nowadays, NoSQL technologies have problems, particularly, with join operation and aggregate functions due to the schemaless nature of NoSQL stores. The users frequently have to write their own custom programs to be able to perform the ETL process on a NoSQL store.

Most of the present studies on ETL over NoSQL stores have focused on volume and velocity, but handling variety issues are similarly important and needs more efforts.

2.2.2 Schema Extraction

The schema extraction aims to find a list of document fields with their types [Abdelhédi et al., 2021]. Most new data models undertake a schemaless representation which does not imply that these data are stored without a schema. Alternatively, the schema is a soft concept where instances in the

same data store referring to the same concept might be stored using distinct schemas to match specific characteristics of each instance [Gallinucci et al., 2018]. In fact, when accessing NoSQL data through a written program, it is critical to have a solid understanding of its structure, or schema. Since it is a crucial step in an ETL process dealing with document stores, we have studied these contributions.

The authors in [Klettke et al., 2015] have proposed a method to extract the global schema of a collection of JSON documents using a graph representation. The nodes represent JSON fields, nested objects, or arrays while the edges reflect the JSON documents' hierarchical structure. Since NoSQL systems do not check any structural constraints, the approach reveals structural data outliers using similarity measures to capture the degree of heterogeneity of JSON data.

In [Wang et al., 2015], the authors have proposed a schema management framework to extract distinct schemas in a collection. To have one single view of collection data, they offer a new concept called *skeleton* used as a relaxed form of the schema. The approach supports queries by allowing developers to find a suitable collection to persist a new document. In the same direction, the authors in [Izquierdo and Cabot, 2016] have proposed a tool called *JSONDiscoverer* that aims to represent implicit structures of a given set of JSON documents as a UML class diagram. This step is followed by an advanced discovery that infers the global schema of a set of JSON documents. Baazizi et al. [Baazizi et al., 2019] were interested in schema inference of massive JSON datasets. The approach aims to infer the structural features of JSON data that describe JSON objects and arrays, and consider nested values and optional keys. The distinguishing feature of their approach is that it is parametric and allows the user to specify the degree of preciseness and conciseness of the inferred schema. Besides, Gallinucci et al. [Gallinucci et al., 2018] have extended the schema extraction level of a JSON documents' collection with schema profiling techniques to capture the hidden rules explaining schema variants. These rules are represented using a decision tree as a schema profile. The proposed algorithm implements a divisive approach to deliver accurate and succinct schema profiles. It combines value-based and schema-based criteria to better capture the rules underlining the use of different schemas within a collection.

Although many authors have conducted schema extraction, this problem is still insufficiently explored. The major disadvantage of the existing approaches is that they do not explore how to find the hidden type of each field while extracting the schema. In fact, the types are identified in a naive way according to the representation of the value. For example, if values are of numeric type, and

represented as a string, the real type will not be noticed. On the other hand, most of the above approaches have focused on the first phase of the ETL process, i.e., the extraction phase, here, extracting document schema. Contributions in the transformation phase remain limited and require more effort.

2.2.3 Joining Dispersed Data Across Several NoSQL Stores

In this section, we present the works that have addressed the issue of joining dispersed data across NoSQL stores. In [Celesti et al., 2019], the authors discussed the impact of performing the join operation in document stores. They underline that some operations, which are trivial in relational databases management systems, may become more complex in NoSQL management systems, particularly the join operation, which is not explicitly available in NoSQL stores. The approach provides an algorithm to perform, at the application layer, an inner-join operation on two collections, which is experimented with MongoDB, a document-oriented database management system. However, the algorithm requires to be fueled with join keys, which are generally not indicated in document stores.

Besides, since the join is mandatory for querying tasks, we have also studied the dedicated querying approaches. In [Mami et al., 2019], the authors proposed *Squerall*, a framework that enables querying heterogeneous data on the fly without prior data transformation. *Squerall* supports MongoDB (document-oriented system) and Cassandra (column-oriented system). In addition, the framework allows the user to declare modifications for altering join keys during query time to make data joinable.

Authors in [Kondylakis et al., 2019] have proposed a data management solution allowing joins over NoSQL Cassandra databases where the primary keys are considered as partition keys. They offer an implementation of query execution and optimization module to find optimal join algorithms using basic heuristics. The approach proposed in [He et al., 2015] inputs two sets of values from join columns and produces a predicted join relationship using an extensive table corpus. They employ statistical co-occurrence to quantify semantic correlation at the row and column levels. However, when working on a spreadsheet with several tables, it is up to the user to select two join columns from two distinct tables. In fact, they just know the tables but not the precise columns that got to be joined.

The previous works have all addressed the problem of joining NoSQL stores. All of them rely on a strong assumption: having the join keys beforehand. It is worth mentioning that, in NoSQL

stores, no prior works have proposed a method to find the pair of join keys, i.e., both *identifiers* and their respective *references*. Hence, it would benefit from examining prior research carried out within a relational database context, particularly the contributions that aim to detect primary keys and foreign keys in relational databases and joinable table discovery as presented in the following section 2.3.

2.3 Data Discovery

Data discovery is the process of exploring data to automatically detect similar, unionable, or joinable attributes among heterogeneous datasets. We review the related works that have tackled these issues. We distinguish two main streams. The first one is related to primary key and foreign key discovery in relational databases. The second stream of approaches has treated the problem of joinable table discovery.

2.3.1 Primary Key and Foreign Key Discovery in Relational Databases

This subsection presents the contributions that have dealt with detecting primary keys and foreign keys in relational databases. Many efforts were made, particularly, for foreign keys detection [Chen et al., 2014, Memari et al., 2015, Wu et al., 2019, Zhang et al., 2010]. Here, we present three representative works among them.

In [Chen et al., 2014], the purpose of the proposed approach is to automatically discover a set of foreign keys connecting a given fact and dimension tables. The foreign key relationships are discovered in PowerPivot. The latter is an Excel add-in that can be used to generate pivot tables from different datasets, perform data analysis, and build data models. The approach is based on a set of pruning rules to filter out candidates, and a scoring function based on a string similarity.

In [Memari et al., 2015], the proposed approach aims to profile unary and multicolumn foreign keys from incomplete data. It proposes three different algorithms according to the ways of handling the occurrences of null values: full, simple, and partial semantics that depends on whether the foreign key columns match all values in the referenced tuple.

In [Wu et al., 2019], Wu et al. have proposed a hybrid-machine framework to detect foreign keys on web tables. Due to the poor quality of web tables, which can contain noisy data, the authors underline that discovering foreign keys required human intervention. Their approach is based on two

phases. They, firstly, find candidate foreign keys using the proposed algorithm. Secondly, the candidates are validated using crowdsourcing. Crowdsourcing is the process of gathering work, information, or views from a large group of people who submit their opinions via social media or mobile applications.

The previous works have proposed approaches to profile foreign keys in relational databases and web tables. However, we note that all of them assume the presence of already known primary keys.

On the other hand, quite freshly, Jiang and Naumann [Jiang and Naumann, 2020] have proposed an approach to automatically discover both primary keys and foreign keys in a relational database. The approach is based on the functional dependencies that describe the characteristics of a table or relationships between tables, namely unique column combinations and inclusion dependencies. Both dependencies have been used to detect primary keys and foreign keys. A unique column combination is a set of attributes whose projection contains only the column combinations having unique and non-null values. Their work is based on the set of inclusion dependencies given as input. However, this assumption couldn't pertain to the context of document stores. Even if this previous work [Jiang and Naumann, 2020] is the closest one to our problem, we cannot apply it out of the context of relational databases. Thus, it is essential to rethink the problem using alternative methods adapted to document stores' schemaless nature.

2.3.2 Joinable Table Discovery

In [Bogatu et al., 2020], the authors proposed an approach that aims to detect if attribute values coming from different sources belong to the same domain. Based on this, whether the sources are candidates to be joined or unioned to populate a target is decided. In [Zhu et al., 2019], given a table and one join column, the authors aim to find joinable tables in data lakes by formulating the problem as an overlap set similarity search. Finding a joining table is based on a given join column regardless of whether it is a primary/foreign key. Additionally, their approach is based solely on values, which is unhandy in a NoSQL context. Indeed, they ignore numeric values since they create casual joins that are not meaningful. However, numeric values are very important to discover identifiers and references in document stores in our work.

Similarly, in [Fernandez et al., 2018], Fernandez et al. propose an approach that aims to find objects

that are semantically related. However, to identify semantic links, their approach requires domain-specific knowledge encoded in an ontology, which is not always available.

These works aim to identify relatedness between tables when explicit relationships are missing. The relatedness can be for joinability or unionability. In addition, they search for significant datasets to populate a target table given a set of tables. The results generated by these approaches are in the form of similar tables or similar attributes regardless of whether among these attributes there are primary/foreign keys or not. However, our objective is to determine identifiers and references among several attributes that can be similar between the two collections. On the other hand, even if these works are interesting, they are dedicated to tabular data. Hence, it is not suitable for schemaless or schema variant data stores as in document stores where we have different levels of object nesting, null and missing values. Besides, by and large, these approaches are based solely on values, which is unhandy in a NoSQL context.
To look for a more practical solution to this problem, we have investigated, in the remainder, the works regarding ontology and schema matching that could be a suitable way to avoid the use of inclusion dependencies.

2.4 Ontology and Schema Matching Approaches

A schema refers to a structure of metadata presenting a blueprint of how the data is stored and accessed by applications [Aumueller et al., 2005]. Schema and ontology matching is a crucial task in the schema integration process since it aims to identify semantic correspondences between metadata, thereby reducing manual treatment. In this section, we report our review findings that we broadly classified as follows: document stores matching, ontology alignment, and graph embedding.

2.4.1 Document Stores Matching

Schema matching aims to identify correspondences between the elements of heterogeneous schemas. These elements are semantically related [Bellahsene et al., 2011]. In the literature, approaches are broadly classified into three categories: *(i)* schema-based approaches: consider only schema information; *(ii)* instance-based approaches: rely on data to retrieve relevant insights that can boost the

schema matching results; and *(iii)* hybrid approaches: combine several matching techniques. Most of these approaches rely on manual human intervention. Furthermore, few researchers have considered schema matching for document stores [Blaselbauer and Josko, 2020].

In [Waghray, 2020], the author presents an empirical study on matching JSON files using existing tools. This study aims to survey and evaluate the state of the art to check whether existing data integration approaches and tools can handle the JSON format. The conducted study demonstrates that these tools do not readily bear JSON and that additional effort is needed to understand the underlying issues properly and develop systems that natively support JSON. The second step consists in cleansing data where the user chooses a column.

In [Knoblock and Szekely, 2015], the author proposed a semi-automatic approach, called *Karma*, for the semantic schema mapping starting from heterogeneous sources, including JSON. To resolve the data format problem, *Karma* starts by converting all the data formats into a nested relational data model using existing methods. Then, the user chooses a column to transform by giving examples of data transformation for particular rows. The system learns the transformation and applies it to the rest of the data. For example, the user inputs the first name and the last name in the correct order: {*Kerry James, Marshall* } instead of {*Marshall, Kerry James*}. Then, the system learns the transformation made to apply it to the rest of the data. To cope with various data formats, the user models each dataset as a semantic description using a domain ontology to be then represented by *Karma* in a common schema, which can be in RDF or CSV format. However, the data integration using a semantic mapping method is limited to the cultural heritage domain. Moreover, it requires an expert user to model the input datasets using a domain ontology.

Similarly, authors in [Blaselbauer and Josko, 2020] proposed an approach to match multiple heterogeneous JSON schemas using linguistic, semantic, and instance-based techniques. This is done in the same collection of documents, where the documents' attributes are related to the same concepts.

Schema matching is typically performed using techniques attempting to approximate the meaning encoded in the schema. On the other hand, ontology matching systems use the knowledge explicitly ciphered in ontologies [Shvaiko and Euzenat, 2005].

2.4.2 Ontology Alignment

Ontologies encode the concepts and properties of a subject area and the relationships between them. Ontology alignment is the process of finding correspondences between concepts in ontologies [Shvaiko and Euzenat, 2013]. Roughly speaking, it aims at identifying semantic similarities between concepts. Several ontology alignment systems are proposed in the literature.

The *Alignment API*, proposed in [David et al., 2011], aims to represent correspondences between two given ontologies. It performs the Cartesian product of possible pairs of entities related to two OWL[1] schemas. In addition, each pair of entities is associated with a similarity measure.

Authors in [Kachroudi et al., 2018] proposed *Kepler*, an ontology alignment system which addresses the key challenges related to heterogeneous ontologies in the semantic web. the system is designed to compute alignments in a multilingual context using a translator.

Authors in [Djeddi et al., 2018] proposed a matching algorithm called eXtended mapping (*XMap*) dealing with large-scale ontology matching. The application of the *XMap* algorithm requires an RDF[2] ontology format. The algorithm is based on string, linguistic and structural similarity measures.

On the other hand, the alignment procedure proposed in the *AML* [Faria et al., 2018] tool is essentially based on the lexical and structural aspects. Indeed, it uses four sources of background knowledge, which are limited to the medical field.

A wealthy number of works proposed semantic matching. They used either a field description [Zhang et al., 2021] or an external domain knowledge encoded in knowledge graphs, ontologies, thesauri, or pre-trained word embedding on corpora. However, these approaches can not be applied when such knowledge is unavailable for the considered dataset. Therefore, recent works opt for embedding techniques to capture semantic similarities without the need for an external domain-specific knowledge.

2.4.3 Graph Embedding

Graph embedding is a technique that converts graph nodes into a low-dimensional vector. The embedding encapsulates the graph topology by preserving the neighborhood similarity between nodes

[1] Web Ontology Language
[2] Resource Description Framework

in the embedded space [Cai et al., 2018]. Recently, various graph embedding techniques have been proposed. Adopting one of these techniques depends on the problem settings and the type of the graph embedding input, i.e., homogeneous graph, heterogeneous graph, or graph with auxiliary information and knowledge.

In [Koutras et al., 2020] authors propose a schema matching approach to generate column similarities based on graph embedding in relational databases. REMA creates an undirected graph using columns and values as nodes connected with edges that reflect the input table. However, the embedding is based on instances, which are unhandy in a NoSQL context.

In [Azmy et al., 2019], authors investigate the problem of entities matching across knowledge graphs. They used graph embedding to benefit from the RDF model's graph nature. Using the graph embedding for learning representations with RDF2VEC carries out entity matching with higher accuracy. The authors outline that resolving ambiguous entities can not be worked out by NLP techniques grounded on a text since the source entities and target entities are syntactically the same. Therefore the disambiguation process is performed through graph embedding, which explores the context of entities.
Another advantage of using embedding, which is highlighted by the *node2vec* technique, is that it detects similarities without a requirement for external knowledge, and it can be based on graphs containing heterogeneous nodes. Using a traditional matching technique does not take advantage of all accessible features, e.g., throwing away data types while processing attributes and using the names of the attributes solely.

To investigate how to exploit NoSQL stores for analytical purposes, we have also scrutinized, in the remainder, the related approaches that aim to enable OLAP analysis over document stores.

2.5 OLAP Analysis over Document Stores

Online analytical processing (OLAP) is widely used as a structured data analysis approach dedicated to decision-making. With the diffusion of NoSQL stores, a great effort should be devoted to finding solutions for OLAP analysis on NoSQL stores, particularly on document-oriented ones. However, to the best of our knowledge, few works have tried to find solutions for OLAP analysis on document stores. In [Hamadou et al., 2019], the authors proposed a preliminary approach that

performs OLAP queries across heterogeneous data models, i.e., relational, document-oriented, and column-oriented, using a dataspace layer on top of the underlying databases. The dataspace is a set of features. Each feature is a representation of a set of attributes modeling semantically the same concept. However, authors assume that the *SameAs* or foreign key relationships between attributes are already known and that all keys are not composite. They provide an execution plan for a given query launched on different databases. Then, the query defined on the dataspace will be translated into a set of queries to be performed on separate databases.

Chouder et al. [Chouder et al., 2019] have proposed an approach to enable OLAP on document stores in the context of self-service BI. This approach extracts the global schema of one collection of nested JSON documents and generates a multidimensional draft schema. To build the multidimensional schema, the dimensions and measures are first identified among the set of fields contained in the JSON collection. Then, the decision-maker can define his query, which is validated by mining approximate functional dependencies. Once a functional dependency is found, the multidimensional schema is refined to add multidimensional hierarchies. Finally, the validated query is translated into the native language of MongoDB.

Similarly, the approach proposed in [Gallinucci et al., 2018] enables OLAP directly on a JSON collection. But, unlike [Chouder et al., 2019] where the research for functional dependencies is done on-demand, this approach looks for all the approximate functional dependencies. Given a collection, the approach consists in integrating the distinct local schemas extracted from documents to propose a global schema. Then, it provides a multidimensional view of the global schema. The identification of different hierarchies is based on approximate functional dependencies.Most stages require user interaction.

These approaches focus on querying a single collection of JSON documents and do not address the problem of integrating several collections.

2.6 Discussion

Our goal is to extract, transform, and load data, which is scattered over several document stores, to be queried for decision-making. In our literature study, we reviewed the approaches that propose solutions for the ETL process over NoSQL stores. The majority of prior research that has been carried

out in a big data context, has focused on volume and velocity, nevertheless addressing variety issues is similarly crucial in tackling several real-world problems.
On the other hand, even though many authors have conducted schema extraction, this problem remains under-explored. They extract the set of document fields with their types. The types are identified in a naive way according to the representation of the value. For example, if values are of numeric type, and represented as a string, the real type will not be noticed. This can have major consequences in a BI system because the measures that can be analyzed are often numeric, and an important measure for the decision-maker can be missed. Besides, identifying the hidden types is extremely important in determining the identifier candidates.

Hence, the major downside of these approaches is that they do not try to find the hidden type of each field while extracting the schema. It would be of special interest to infer the real types of values since it is very important in data discovery and data integration.
Moreover, although most of the above approaches have only focused on the first phase of the ETL process, i.e., the extraction phase or extracting documents schemata, contributions in the transformation phase remain briefly addressed in the literature and require more effort. Some key questions and notions are still not discussed. In fact, we investigate the focus on the problem of joining scattered data in the context of NoSQL stores. Indeed, even if the problem of the detection of primary keys and foreign keys in relational databases is a known problem and dedicated solutions have been proposed in the literature, the issue of automatic detection of identifiers and references in document stores remains a challenge to which no previous contribution has been made.
Additional studies to understand more completely the above-mentioned concerns are required. Therefore, we have also investigated other data discovery methods that seek to identify joinable or unionable tables. These works attempt to find relatedness between tables when explicit relationships are lacking. The relatedness may be for joinability or unionability. In other words, given a set of tables, they look for meaningful datasets to populate a target table. These approaches generate results in the form of similar tables or similar attributes regardless of whether there are primary/foreign keys among these features or not. Additionally, though these works are interesting, they are devoted to tabular data. As a result, it is not appropriate for schema variant data stores as document stores, where we have different levels of object nesting, null and missing values. Furthermore, these approaches are generally based solely on values which is unhandy in a NoSQL context.

On the other hand, we have reviewed works on ontology and schema matching. In order to deeply study the advantages and disadvantages of ontology-based approaches and investigate how to exploit them for the matching of NoSQL data sources, we have explored the different ontology alignment systems presented in Subsection 2.4.2. For this, we propose assimilating document stores schemas alignment problem to an ontology alignment problem. We consider JSON as a document-oriented format and Ontology Web Language as the ontology format. As a first step, we transform a JSON schema into an OWL format. Then, we align two OWL schemas that we have created starting from two initial schemas of JSON collections. The result given by *Kepler* [Kachroudi et al., 2018] is not following our goal. This is mainly due to the fact that this algorithm prioritizes the topological aspects more than the terminological and syntactic ones. As for the *AML* tool, presented in [Faria et al., 2018], it relies on four sources of background knowledge that are restricted to the medical area. Additionally, the obtained alignment results from the algorithm, proposed in [Djeddi et al., 2018], are incomplete since it aligns the classes without considering object properties. Therefore, if objects nested in a JSON document are transformed into ontology properties to guarantee the topological aspect of the document-oriented model, they will not be considered when applying the proposed algorithm.

The formal semantics is more grounded in ontologies than in document stores since ontologies represent the meaning rather than the data. The ontology alignment process is challenging, particularly when the ontologies are retrieved from heterogeneous sources leading to inherent differences. Nearly all of the alignment techniques rely on string-based similarities, which cannot handle the vocabulary mismatch issue. Therefore, finding out the suitable similarity measures and how to viably combine them to be used in alignment solutions is still a challenge [Nkisi-Orji et al., 2018].
On the other hand, since a large number of works call for external domain knowledge, which is almost unavailable, current researches opt for embedding techniques to capture semantic similarities without the need for external domain-specific knowledge. Embedding techniques are very interesting to explore, nevertheless, the existent works that used embedding for semantic matching are based on instances, which is impractical in a NoSQL context.

On the other hand, existing works that have dealt with the document stores OLAP analysis focus on querying a single collection of JSON documents and do not address the problem of integrating several collections. Besides, as document stores are characterized by their volume and variety, it is

important to provide an on-demand ETL that extracts solely the data that meet the decision-makers' requirements. On-demand approaches are worth interest and are not yet used for the ETL in the context of document stores [Baldacci et al., 2017, Yang et al., 2015]. To the best of our knowledge, no prior works have proposed a BI&A approach based on the on-demand methodology that starts from schema extraction of document stores (with a multitude of varied and dispersed data over more than one collection), performs ETL operations, and reaches OLAP analysis.

2.7 Conclusion

In this chapter, we have carried out a literature review of the related works regarding our objective: extracting, transforming, and loading NoSQL stores for decision-making, and preparing these data to fit the decision-makers' requirements. We identify several key lines of approaches dealing with: *(i)* ETL process over NoSQL stores; *(ii)* problem of joining dispersed data across NoSQL stores and tabular data in the absence of integrity constraints; *(iii)* ontology and the schema matching issues; and *(iii)* OLAP analysis over NoSQL stores in the context of the self-service Business Intelligence.

In the next chapter, we present our global BI&A approach to extract, transform and load scattered data across document stores.

Chapter 3

Extract Transform and Load Scattered Data across Document Stores

Contents

3.1 Introduction

While NoSQL databases are extensively used nowadays, BI&A has traditionally been associated with relational systems since they have been highly esteemed for data quality and consistency. Leveraging NoSQL stores for decision-making lacks maturity, traceability, and metadata management. Thus, to cope with the volume, variety, and velocity of these data, it is essential to review all the phases of the BI&A architecture, particularly the ETL process, to provide solutions capable of exploiting NoSQL stores for decision-making.

For this, we introduce a new comprehensive approach that aims to extract, transform, and load document stores on-demand (on-demand ETL). This approach is hybrid as it considers both data sources schemaless nature and analytical needs in order to explore several document stores in an efficient way. Over the last few years, on-demand ETL has changed the way data analytics and business intelligence are performed. Instead of extracting, transforming, and loading data in bulk, on-demand ETL processes data as needed. The main characteristics of an on-demand ETL are that:

- ETL processes are not processed until the data is required for a query.
- only the data required to execute the query is targeted by the ETL processes.

By incorporating the on-demand ETL, a BI&A architecture is thus underpinned by real-world business requirements.

In this chapter, we particularly focus on the ETL phase starting from document-oriented sources, as ETL is one of the most complex phases, on which strongly depends the success of the BI&A project Theodorou et al. [2017].
Worthy of mention is that we introduce a global BI&A architecture where our contributions are focused on the ETL process. Thus, the OLAP phase is outside the scope of this thesis. Specifically, the distinguishing features of our approach are as follows:

- To the best of our knowledge, there does not exist a global BI&A approach dedicated to NoSQL data sources that extracts, transforms, and loads the required data on-demand for decision-making.
- Here, we focus on document stores, and unlike existing works, our approach is not limited to one

collection, as we consider the dispersion of data across several collections.

The remainder of this chapter is organized as follows: in Section 3.2, we recall the basic concepts related to document stores. In Section 3.3, we give an overview of our approach. In Sections 3.4, 3.5, and 3.6 we present the core stages of our approach. To roll out and validate our approach before implementation, we propose, in Section 3.7, a case study based on the TPC-H benchmark. Finally, Section 3.8 draws the conclusion.

3.2 Preliminaries

Document stores, *aka document-oriented databases*, are one of the four families of NoSQL stores. A document (cf., Definition 1) is the basic concept of document stores. A document has a schemaless nature: it does not have up-front constraints or a strictly predefined schema. For instance, some attributes in documents can be missed entirely, have null values, or have different data types.
To elucidate the basic concepts, we briefly present, in Table 3.1, the terminology related to document stores and their associated concepts in relational databases.

JavaScript Object Notation (JSON[1]) is currently the most commonly adopted format that we will use in the remainder. As shown in Figure 3.1, syntactically, keys and values are separated by colons, while commas separate key-value pairs. Objects and arrays can be embedded inside a document. Objects use curly braces symbols and contain an unordered set of key-value pairs, while arrays use square bracket symbols and include an ordered collection of values.

Definition 1 *(***Document and Collection***) A document d is an object. Each object contains an unordered set of key-value pairs (keys are also called names [Chouder et al., 2019]); a key is a string, while a value can be a primitive value (i.e., Number, String, or Boolean), an object, an array of values, or null. A collection C is an array of documents.*

[1] https://www.json.org

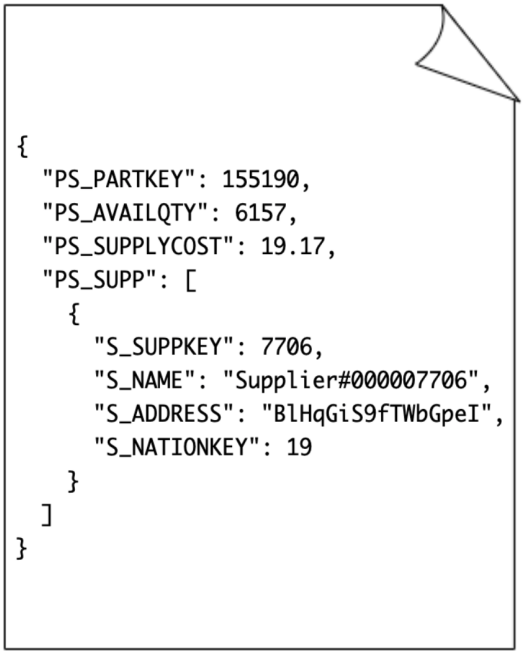

Figure 3.1: Example of JSON document

Table 3.1: Main terminology of document stores and its equivalent in relational databases

Document stores	**Relational databases**
database/document stores	database
collection	table
document	row
field	column
object id	surrogate key

3.3 Overview of our Approach

We introduce a new approach that aims to extract, transform, and load several document stores in an on-demand fashion and provide dedicated solutions for decision-making. It considers both data sources' schemaless nature and analytical needs to explore several document stores efficiently.

As shown in Figure 3.2, our approach operates in two main stages: *(i)* on-demand ETL where ETL operations are only performed on the pertinent data that meets the decision-maker requirements; and *(ii)* OLAP analysis of this data (This stage is out of the scope of this thesis).

In our work, a first objective is to consider a document-oriented data model taking into account

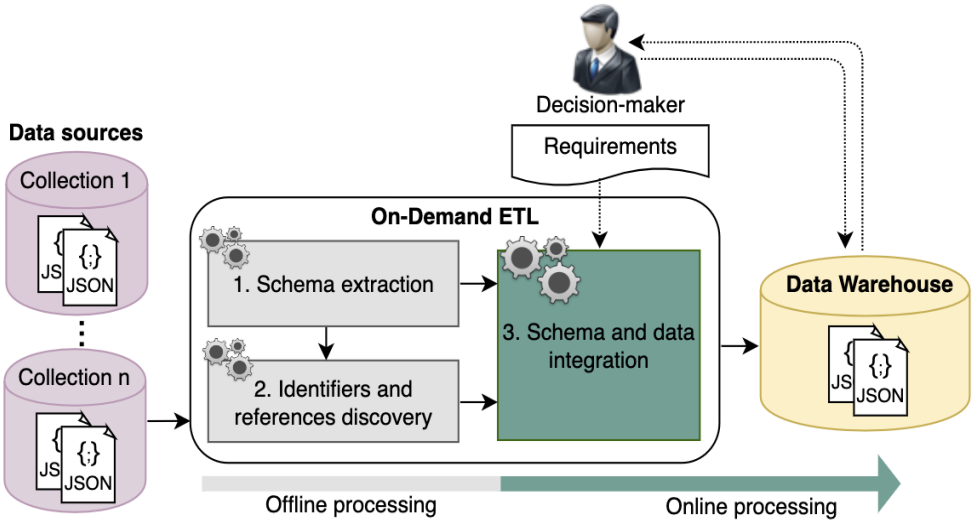

Figure 3.2: Our approach general architecture as a glance

schema variety. We consider scattered data over several collections. Since in documents it is common to nest one object into another object Pokorný [2020], we regard the different cases of nesting objects:

- The document-related objects are represented separately from the original document (i.e., in another collection) and are referenced using identifiers.
- All the document-related objects are nested in the original document with different nesting depths.

Starting from dispersed data across several document stores, our approach is built upon two main stages:

- **Stage 1: On-demand ETL** aims to extract, transform and load only, the relevant data for each specific analysis. It operates in three phases as follows:
  - **Phase 1: Schema extraction:** extracts the schema of each collection, in order to deal with the schemaless and variety nature of document stores. This phase is automated and processed when the user is offline.
  - **Phase 2: Identifiers and references discovery:** fetching relevant data often needs to access more than one collection of JSON documents. This multiple access requires finding the

"pivot" connecting fields to perform a join operation. The latter is a complex operation to ensure, while no prior definition of join keys. While joining tables in relational data sources is assured by dint of a precise join key, in document stores, collections are unlikely to have an exact join key due to the absence of integrity constraints. So, identifying key fields that are necessary for joining two collections is a real challenge that we detail separately in the next chapter (cf., Chapter 4). This phase is automated and processed when the user is offline.

- **Phase 3: Schema and data integration:** starting from several collections, this phase aims to:
  - perform a mapping between the decision-maker requirements and collections schemas.
  - perform ETL operations.
  - create the DW which is a collection of JSON documents.

It is worth mentioning that this phase is processed when the user is online, since it requires interaction with him, as depicted in Figure 3.2.

- **Stage 2: OLAP analysis**: the aim of this stage is to ensure an OLAP analysis adapted to document stores. The main difference with respect to other approaches is that OLAP analysis can be performed on several collections which is very important in BI&A applications.

In the remainder, we present the core stage of our approach, which is divided into two phases: *(i)* schema extraction of document stores; and *(ii)* schema and data integration.

3.4 Schema Extraction

Document stores have a dynamic schema, mainly evident through the presence or absence of specific fields with various types. Although this schemaless nature guarantees some perks, the lack of schema information makes data processing complex and difficult. Hence, it is critical to extract the exact schema of each collection in order to perform the data integration successfully. For this, we propose extracting a flat document schema (cf., Definition 2) and the schema of each collection (cf., Definition 3) focusing on inferring real types.

Definition 2 (Document flat schema) *A list of fields with their associated types. Let* $S_D = \{(p,t)_i / 1 < i < k\}$ *be the schema associated to a JSON document* D*. It consists of* k *pairs* (p,t)*, such that:*

p: the field path from the document root. It is the unique identifier of each field.

t: the field type. Since a field can have different types from one document to another, we consider the most frequent type.

Example 1 *Figure 3.3 depicts a JSON document on the left and shows its associated flat schema on the right, where $ symbol represents the document root.*

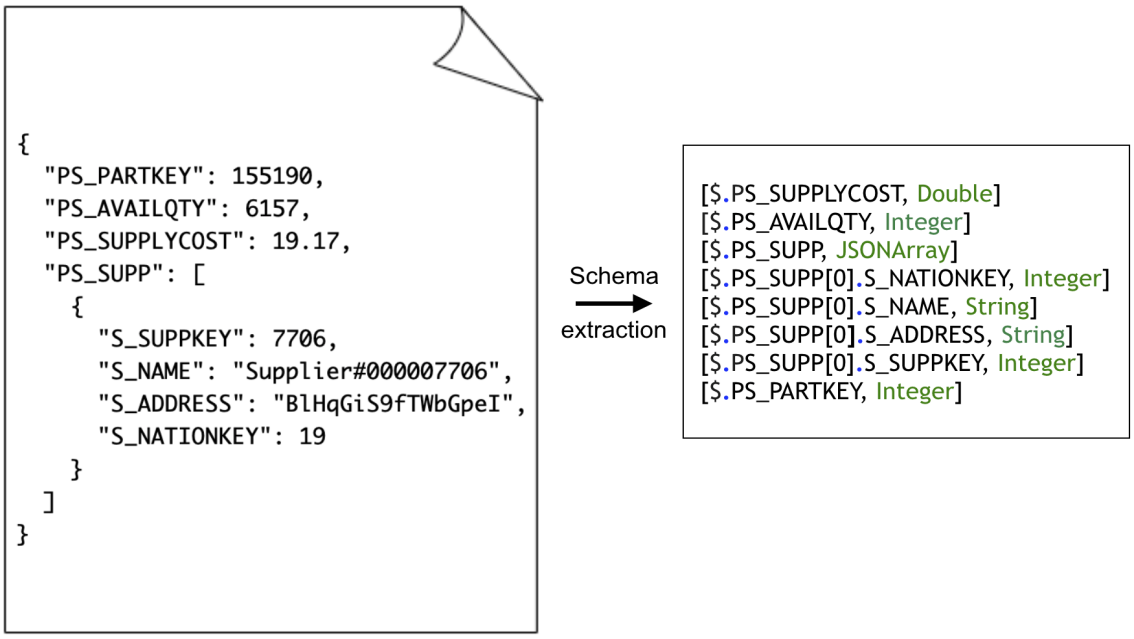

Figure 3.3: Example of document flat schema generation

Definition 3 (Collection schema) *the schema of a collection* C *is* $S(C) = \bigcup_{j=1}^{l} S_{D_j}$*, where* S_{D_j} *is the flat schema associated with a JSON document* D_j *that belongs to the collection* C *and* l *denoting the number of distinct documents schemas that exist in a collection* C*.*

Similarly to data, the quality of the extracted schema influences the data integration process and therefore leads to inaccurate analysis. It is important to overhaul the extracted schemas from document stores.

As mentioned in the related work, we report several methods in the literature to address the schema extraction issue. The proposed solutions generate a schema as a set of document fields with their associated types. Oddly, the types are not thoroughly described. As far as we know, no previous work has provided a schema that yields the real hidden type for document fields. Since we can hide a real primitive type under another primitive type, we propose a new method that aims to check the type of each field to detect such cases. This method avoids misleading results generated by a wrong data type. For instance, if the values of a given field f_i are of Float type, e.g., 19.17 (cf., Example 3.3), while they are represented between quotes: "19.17", in this case, the real type must be Float instead of String.

Checking the actual type requires access to the values, making it awkward to use in a NoSQL context as the volume of data is important. We thus suggest using the random sampling technique (cf., Definition 4). The latter is an unbiased form to collect a subset of data using randomness [Nguyen et al., 2020]. Hence, given a sample of values of a field f_i belonging to a collection C, we check the actual type using a set of regular expressions. For instance, in order to check if the type is really String, Float, or Date, we have used these regular expressions; The above list of regular expressions can be extended and completed gradually.

- check if the type is String: "$.*[a-zA-Z].*$"
- check if the type is Float: "$[-+]?[0-9]*\\.?[0-9]+$".
- check if the type is Date :
  - "$M/dd/yyyy$"
  - "$dd.M.yyyy$"
  - "$M/dd/yyyy\ hh:mm:ss\ a$"
  - "$dd.M.yyyy\ hh:mm:ss\ a$"
  - "$dd.MMM.yyyy$"
  - "$dd-MMM-yyyy$"
  - "$yyyy-dd-M$"

Definition 4 (Simple Random Sampling) *Simple random sampling is one of the most widely used category of probability sampling techniques in the statistic literature [Zhao et al., 2018]. Each item in*

the underlying population has an equal probability of being selected in an unbiased fashion. Random numbers are generated in order to select items to constitute the random sample.

Identifying the hidden types is extremely important in determining the identifier candidates, as detailed in the next chapter (cf., Chapter 4).

3.5 Schema and Data Integration

This phase aims to ensure the mapping between decision-maker requirements (cf., Definition 5) and collections schemas, to perform ETL operations and, to create the DW which is a collection of JSON documents.

3.5.1 Mapping

The goal of this step is to ensure the mapping between collections schemas and the decision-maker requirements so as to unveil the collections of interest, which meet the decision-maker requirements and will be included as input in the rest of the phases.

Definition 5 (Decision-maker requirements) *decision-maker requirements are expressed as a multidimensional schema. The latter involves the subject of analysis, which is described by numerical attributes called measures (cf., Definition 7) and the axes of analysis called dimensions (cf., Definition 6) [Thenmozhi and Vivekanandan, 2012]. A multidimensional schema, denoted with MS, is a triple $MS = (D, M, f)$ where:*

- $D = \{d_i, 1 < i < l\}$*: a finite set of dimensions* d_i*. Each dimension is associated to a finite set of hierarchy levels* $h_j(d_i)$*.*
- $M = \{(m, o, a)_i, 1 <= i < n\}$*: the set of measures to be analysed, where:*

  m: label of the measure to be analysed.

  o :

  $$o = \begin{cases} \textit{Computation formula}, \\ \emptyset, \qquad\qquad\qquad \textit{otherwise} \end{cases}$$

  a: aggregation operator associated with each measure m_i *(SUM, AVG, COUNT, etc.).*

- *f: a function that associates each measure to a finite set of grouping fields, such that* $f : M \rightarrow G$ *where* G *is a set of grouping fields.*

Definition 6 (Dimension) *A dimension is a qualitative value representing an axe of analysis. It is used to categorize and reveal details in data, e.g., region and date.*

Definition 7 (Measure) *A measure is a quantitative value that can be aggregated against dimensions, e.g., turnover.*

Example 2 *A decision-maker intends to analyze the order revenue measure according to product type and order year. The multidimensional schema is* $MS = (D, M, f)$ *where:*

- D =*{product type, date={month, year}}*
- M =$\{m_1 = (revenue, [price * quantity], SUM)\}$
- $f(m_1)$= *{year, product type}*

Definition 8 (Mapping) *given the multidimensional schema and the set of collections schemas, a mapping is defined by the function:* $\varphi : \{G, M\} \rightarrow S$
$x \mapsto \varphi(x)$

where:

- $\{G, M\}$ *is a finite set of multidimensional attributes, i.e., dimensions and measures.*
- S *is a a finite set of collections' schemas.*

This phase is semi-automatic since it requires interaction with the decision-maker. In fact, a multidimensional attribute can be found in more than one collection. In this case, the decision-maker has to validate manually the mapping proposed by the system.

3.5.2 Performing ETL Operations

ETL is a crucial part of the BI&A chain where most of the data curation is carried out. The latter is made up of a set of operations [Atigui et al., 2012]. Hence, we present an extensible list of ETL operations that we have adapted to document stores. As shown in Table 3.2, we introduce a

formalization for each operation and a notation assigned to each one. For each ETL operation, we provide an example of its counterpart in the Talend Big Data (TBD) tool. The latter is a free and open source tool. It offers a development environment dealing with a variety of big data sources and targets. We note that, unfortunately, commercial ETL tools, almost provide the same components for different data sources format since they transform the data source format to a flat representation. For instance, Talend offers almost the same ETL components in different products as Talend Data Integration and Talend Big Data.

After pinpointing the collections that meet the decision-maker requirement, the first operation is to join these collections based on the detected identifiers and references. Thus, we obtain a single collection where relevant data are centralized in one document store, which represent the data warehouse after performing other ETL operations.

Table 3.2: The main ETL operations for document stores

ETL operations	**Description**	**Notation**	**Examples of Talend Big Data Components**
Input/Wrapper	Return a collection C_2 containing key-value pairs that are constructed from a collection C_1.	$C_2 \leftarrow I_{f_1,...,f_n}(C_1)$	tMongoDBInput
Project	Pass along the documents of the collection C to acquire only the given fields $f_1, ..., f_n$ with their values.	$\prod_{f_1,...,f_n}(C)$	tExtractJSONFields
Filter	Select a subset of documents within a collection C that match a given criteria cr.	$\sigma_{cr}(C)$	tFilterRow
Cartesian product	Return a collection containing all possible combinations of the documents belonging to the collections C_1 and C_2.	$C_1 \times C_2$	tMap*
Join	Combine data from two collections based on one or more fields. Given two collections C_1 and C_2, we denote: - $d_i^{C_1}$, $d_j^{C_2}$: documents belonging to the collections C_1 and C_2, respectively; - $A(d_i^C)$: the list of (field, value) pairs included in the document d_i^C; - $A\_key(C_1, C_2) = (a_1, a_2)$: function applied to search the key fields a_1 and a_2 that link the collections C_1 and C_2; - CF: the resulting collection of the C_1 and C_2 join; CF is an array of documents d_{ij} where $d_{ij} = \{A(d_i^{C_1}) + A(d_j^{C_2})\}$ such as $\{a_1.value = a_2.value\}$ where $a_1.value$, $a_2.value$ are the values respectively associated to a_1 and a_2.	$C_1 \bowtie_{(a_1,a_2)} C_2$	tMap*

* In Talend Big Data, it is up to the user to select manually the join attributes to combine data from two collections.

ETL operations	**Description**	**Notation**	**Examples of Talend Big Data Components**
Aggregation	Group fields from multiple documents based on one or more document fields. We denote with: - $G = \{g_j / 1 < j <= n\}$: a set of fields used for grouping values; - f_i: field that will be aggregated; - F_i: aggregate function;	$Agg(F_i(f_i), G)$	tAggregateRow
Set a default value	Set default value for a document field within a collection C.	$\psi_{f \leftarrow c}(C)$	Edit the schema using built-in settings
Rename	Rename a document field in a collection C.	$\rho_{f_1 \leftarrow f_2}(C)$	Edit the schema using built-in settings
Arithmetic conversion	Convert numerical fields $\{n_1, n_2, ...\}$ by applying an arithmetic operation $O_t(n_1, n_2, ...)$, where t is the arithmetic operation type.	$\{n_1, n_2, ...\} \Rightarrow O_t(n_1, n_2, ...)$	tMap*
Format conversion	Convert field value format $f_1(v)$ to another given format $f_2(v)$.	$f_1(v) \Rightarrow f_2(v)$	tMap*

* In Talend Big Data, it is up to the user to select manually the join attributes to combine data from two collections.

3.6 OLAP Analysis

For decades, OLAP has been widely utilized as a data analysis approach to bear decision-making on structured data within an organization. With the widespread adoption of NoSQL databases that store semi-structured data, there is an increasing need for making OLAP operational on document stores, so that even inexperienced users can gain knowledge and can use a NoSQL-based data warehouse. The decision-maker expresses OLAP queries that are then translated into a document store native query language. In fact, even if data are stored and extracted from document stores, the data processing is kept transparent for the end user.

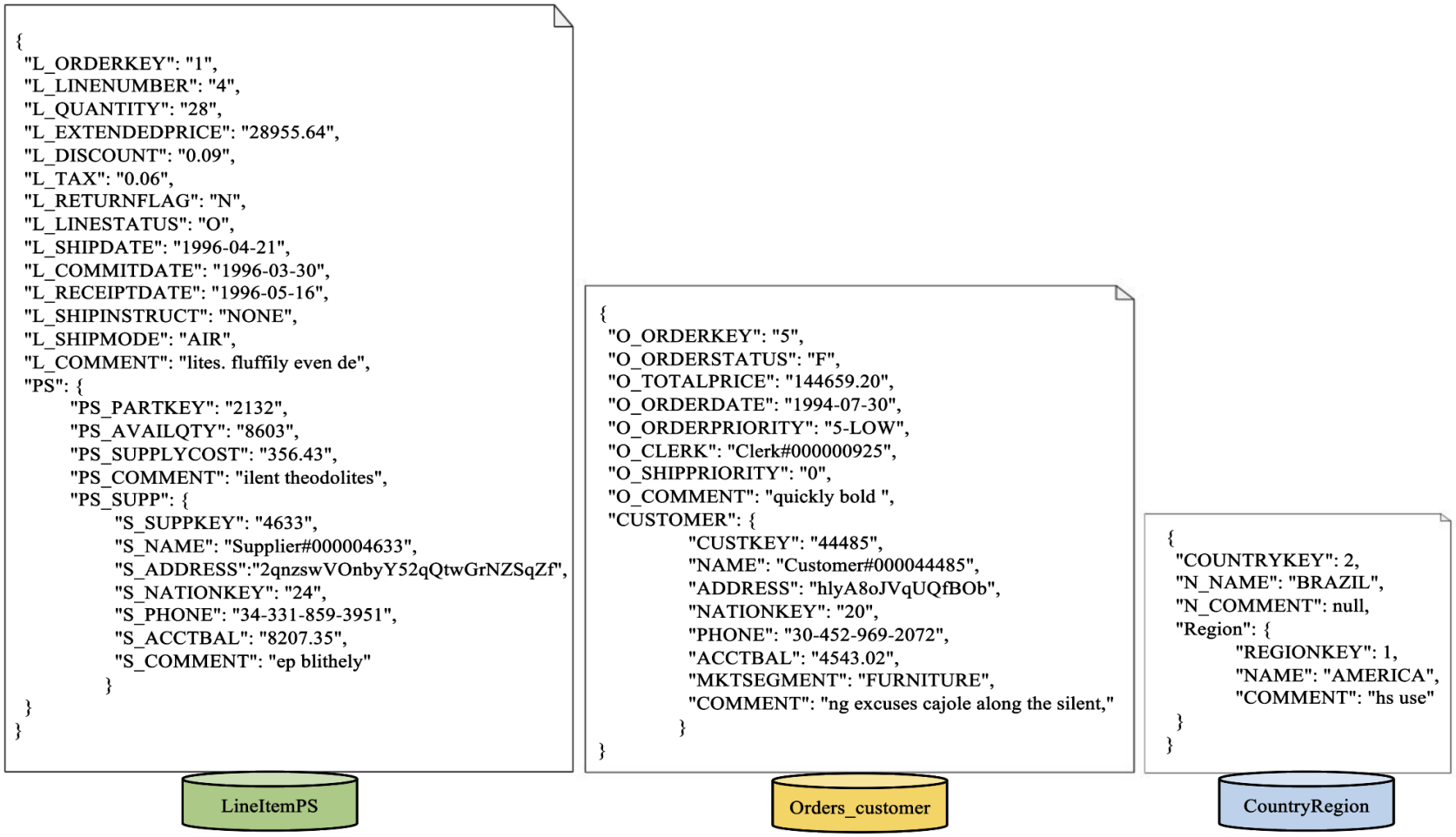

Figure 3.4: Sample JSON documents of each collection created from the TPC-H benchmark

3.7 Case study

In this section, our objective is to study the feasibility of our approach. We present a case of study based on JSON collections. We roll out the approach using Talend Open Studio for Big Data to perform ETL operations, and MongoDB to store JSON collections.

3.7.1 Data Sources

Figure 3.4 shows three JSON collections, i.e, `LineItemPS, Orders_customer and CountryRegion` that are based on the TPC-H[2] benchmark. This benchmark consists of relational sources that we have transformed into JSON collections[3]. In order to have different storage models, we have denormalised *(i)* the `LineItem` collection by embedding documents from the `PartSupp` and the `Supplier` collections; *(ii)* the `Orders` collection by embedding documents from the `Customer` collection ; and *(iii)* the `Country` collection by embedding documents from the `Region` collection. For the sake of legibility, we have presented an excerpt of one document from each collection. The used collections of JSON documents are stored on MongoDB as a document-oriented DBMS.

3.7.2 Decision-maker Requirements

Based on this benchmark, our basic scenario is an on-demand ETL, which starts by extracting the schema for each collection as an upfront processing. Secondly, we start from the decision-maker requirements expressed as a multidimensional schema $MS = (D, M, f)$ where:

- D ={*date*={*year*}, *nation*={*n_name*}}
- M ={m_1 = (*profit, [(l_extendedprice*(1 - l_discount)) - (ps_supplycost * l_quantity)], SUM*)}
- *f*(m_1)= {*year, nation*}

The above multidimensional schema is defined based on the query `Q9` proposed in the TPC-H benchmark. This query identifies the profit made on a particular parts line, categorized by supplier nation and year. The functional query definition as described in the TPC-H documentation is represented in Listing 3.1. The decision-maker intends to analyse the profit measure according to the order

[2]Decision support benchmark: `http://www.tpc.org/tpch/`
[3]`https://github.com/souibguimanel/TPC-H_JSON/`

year and supplier nation. The profit is defined as the sum of `[(l_extendedprice * (1 - l_discount)) - (ps_supplycost * l_quantity)]`.

```
select nation,
       o_year,
       sum(amount) as sum_profit
from (
 select n_name as nation,
        extract(year from o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as
            amount

 from part,
      supplier,
      lineitem,
      partsupp,
      orders,
      nation
 where s_suppkey = l_suppkey and
       ps_suppkey = l_suppkey and
       ps_partkey = l_partkey and
       p_partkey = l_partkey and
       o_orderkey = l_orderkey and
       s_nationkey = n_nationkey
) as profit

group by
nation,
o_year
order by
nation,
o_year desc;
```

Listing 3.1: TPC-H: Q9 Query

```
select supp_nation,
       cust_nation,
       l_year, sum(volume) as revenue
from
  (select n1.n_name as supp_nation,
          n2.n_name as cust_nation,
          extract(year from l_shipdate) as l_year, l_extendedprice * (1 -
              l_discount) as volume
   from supplier,
        lineitem,
        orders,
        customer,
        nation n1,
        nation n2
   where s_suppkey = l_suppkey
         and o_orderkey = l_orderkey
```

```
        and c_custkey = o_custkey
        and s_nationkey = n1.n_nationkey
        and c_nationkey = n2.n_nationkey
        and ((n1.n_name = '[NATION1]'
            and n2.n_name = '[NATION2]')
            or (n1.n_name = '[NATION2]'
            and n2.n_name = '[NATION1]'))
        and l_shipdate between date '1995-01-01' and date '1996-12-31') as
            shipping
group by supp_nation,
        cust_nation,
        l_year
order by supp_nation,
        cust_nation,
        l_year;
```

Listing 3.2: TPC-H: Q7 Query

3.7.3 Mapping and ETL Operations

To extract pertinent data that meet the decision-maker requirements, we perform a mapping step in order to identify collections of interest, namely `LineItem_PS, CountryRegion, and Orders`. Each multidimensional attribute is associated to the collection to which it belongs. For instance, the multidimensional attribute `n_name` belongs to the `CountryRegion` collection, while `o_orderdate` belongs to the `Orders_customer` collection. The fields that are required to compute the profit measure belong to the `LineItem_PS` collection. Therefore, a join operation should be performed, in order to combine data from the above-mentioned collections. For example, a join operation should be performed between `LineItem_PS` and `CountryRegion` collections. To illustrate the operations presented in Table 3.2, we propose a simplified ETL workflow designed using Talend Open Studio for Big Data[4] (TBD) as depicted in Figure 3.5.

[4] *Talend Big Data* is a free and open source tool. It offers a development environment dealing with a variety of big data sources and targets.

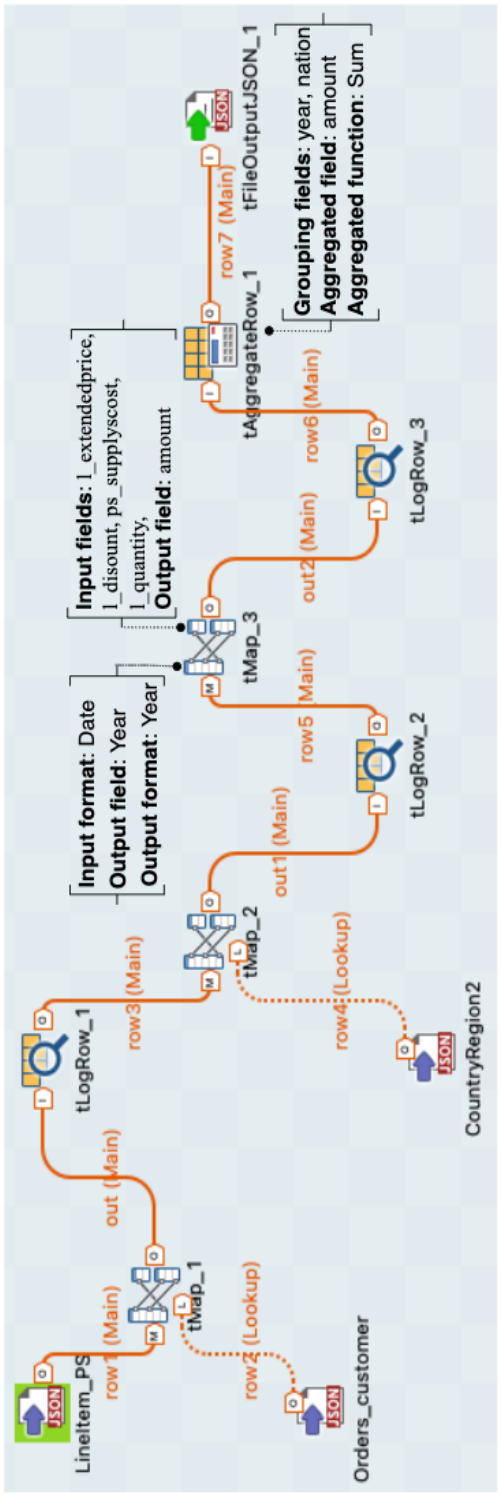

Figure 3.5: ETL workflow example

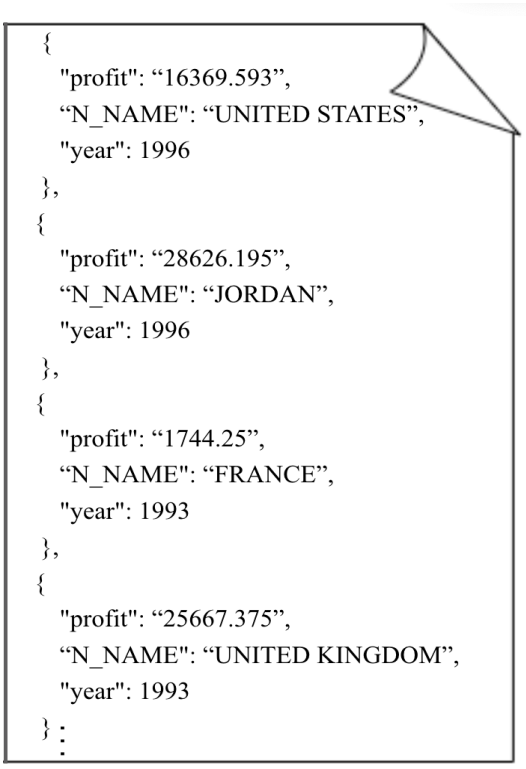

Figure 3.6: The ETL workflow result

3.7.4 Accuracy and Completeness Evaluation

The final result of the workflow presented in Figure 3.5, is expressed in JSON format containing the analysis requirements: *year*, *nation* and the *profit* measure, i.e., Sum (*amount*). We study the feasibility of our approach by ensuring that the collected data return a correct result during the querying process compared to the relational databases result.

To do so, we have considered separately the `Q7` and `Q9` TPC-H queries as decision-maker requirements. According to each query, we have performed the different phases of our approach to extract relevant data that will be stored in a JSON collection. To execute the queries on the collections containing relevant data, we have translated them into the MongoDB query language as shown in Table 3.3.

To check the accuracy and completeness of the obtained results, we execute the queries and compare each query result obtained from our approach against the result obtained when we execute the same

query on the TPC-H relational sources using MySQL[5]. The comparison between the two results proves both accuracy and completeness of the result generated by our approach as shown in Table 3.3.

[5]an open source relational database management system

Table 3.3: Example of queries execution to evaluate our approach in terms of completeness and accuracy of the generated analysis results

TPC-H query translated to the MongoDB query language	**Excerpt of the querying result using MySQL and MongoDB**
Q9: db.collection.aggregate([{$group : {_id:{ nation: "$nation", year: "$year"}, profit: {$sum: "$amount"}}}, { $sort: {year: −1}}])	**MySQL** +---+---+---+ | nation | o_year | profit | +---+---+---+ | ALGERIA | 1996 | 27378.2400 | | CANADA | 1996 | 55567.1720 | **MongoDB** { "_id" : { "nation" : "ALGERIA", "year" : "1996" }, "profit" : 27378.239999999998 } { "_id" : { "nation" : "CANADA", "year" : "1996" }, "profit" : 55567.172000000006 }
Q7: db.collection2.aggregate([{$group : { _id: {supp_nation: "$supp_nation", cust_nation: "$cust_nation", l_year : "$l_year" }, revenue: {$sum: "$volume"} }}, {$sort: {supp_nation: −1 , cust_nation: −1, l_year: −1}}])	**MySQL** +---+---+---+---+ | supp_nation | cust_nation | l_year | revenue | +---+---+---+---+ | UNITED KINGDOM | JORDAN | 1996 | 20321.5008 | **MongoDB** _id" : {"supp_nation" : "UNITED KINGDOM", "cust_nation" : "JORDAN", "l_year" : "1996"}, "revenue" : 20321.500799999998}

3.7.5 Summary of Existing Tools

In the case of study, we have used a set of components provided by the Talend Big Data tool to create the ETL workflow. Unfortunately, most of these components rely on manual human intervention. For the tMap component, it is up to the user to select manually the join attributes to combine data from two collections, since TBD does not propose neither a syntactic matching nor a semantic matching between two given data sources. Indeed, the variety, schemaless nature, and the absence of integrity constraints problems are not considered in TBD. Besides, regardless of the nature of the data source, TBD provides a flat representation of the schema, which does not assist the user to easily explore NoSQL stores.

On the other hand, the data preparation phase is provided by some data quality and data profiling tools as Talend Data Quality[6] (TDQ). Thus, we conduct a set of experiments on TDQ to evaluate their assistance to quality [Souibgui et al., 2019]. The tool generates statistics about data sources and creates matching analysis, especially, dedicated to analyze duplicates. Since the elimination of duplicate records requires similarity detection, TDQ creates a matching analysis, which is based solely on syntactic similarities. Whereas, addressing semantic similarity is very important to conduct a more precise matching analysis. Hence, commercial data profiling tools and data integration tools, do not consider semantic similarity, so they are unreliable to suggest join keys or even matching attributes, especially when the data are varied and dispersed.

3.8 Conclusion

In this chapter, we presented the different phases of our BI&A approach that extracts, transforms and loads the required data for decision-making (on-demand ETL) from document stores. We focus on the on-demand ETL stage where, unlike existing works, we consider the dispersion of data over two or more collections. We have presented a case of study, based on JSON collections, which shows the feasibility of our approach.

Fetching relevant data that meet the decision-makers requirements often needs to access more than one document store. While joining tables in relational data sources is straightforwardly owed to the availability of a precise join key and integrity constraints, in document stores, collections are a long

[6] an open source profiling tool

way away from having an exact join key because of the absence of integrity constraints. So, identifying the ”joinable” fields to stick to two document stores is a tricky challenge. Despite its importance, no previous work has paid close heed to automatically detect join key pairs in the context of NoSQL stores, particularly in document-oriented stores. We address this issue in the next chapter (cf., Chapter 4).

Chapter 4

IRIS-DS: Automatic Detection of Identifiers and References in Document Stores

Contents

4.1 Introduction

In Chapter 3, we have proposed a hybrid BI&A approach that considers both schemaless data sources and analytical needs to explore several document stores efficiently. Carrying out primary ETL operations, especially how to correctly join two different document stores, i.e., collections, still being a challenge [Celesti et al., 2019, Souibgui et al., 2019].

Fetching relevant data that meet the decision-maker's requirements often needs to access more than one document store. While joining tables in relational data sources is straightforwardly owed to the availability of a precise join key, in document stores, collections are a long way away from having the same join key because of the absence of integrity constraints. So, identifying the "joinable" fields to stick to two document stores is a tricky challenge.

For this, in this chapter, we introduce IRIS-DS (**I**dentifiers and **R**eferences D**IS**covery in **D**ocument **S**tores), a new approach that aims to automatically discover the pairs of join keys (*identifier*, *reference*) starting from more than two document stores.

The chapter's outline is as follows: in Section 4.2, we present a motivating example that emphasises the main challenges. In Section 4.3, we introduce the core stages of our approach. In Section 4.4 and Section 4.5 we detail each stage. Then, in Section 4.6, we explain the overall algorithm. Finally, in Section 4.7, we present a case study of our approach.

4.2 Motivating Example

Here, we present a motivating example that smoothly sheds light on the significant challenges of detecting the join keys in the context of document stores. We consider n collections denoted $C_1, \ldots, C_n$, that store two main topics, to wit `orders` made on marketplaces like *Amazon* and *Cdiscount*, and `deliveries` insured by brands like *Bosch* and *Moulinex*. For the sake of simplicity, Figure 4.1 shows two collections, C_1 for `orders` and C_2 for `deliveries`. Suppose that we are interested in analyzing the deliveries' delay, called `DD`. We need to compute the delay as the difference between the actual delivery date versus the expected one. C_1 contains all the orders made on *Amazon* marketplace and C_2 contains the deliveries done by the Bosch brand to different marketplaces.

As `DD` = *deliveryDate* − *expDeliveryDate* where `deliveryDate` $\in C_1$ and `expDeliveryDate` $\in C_2$,

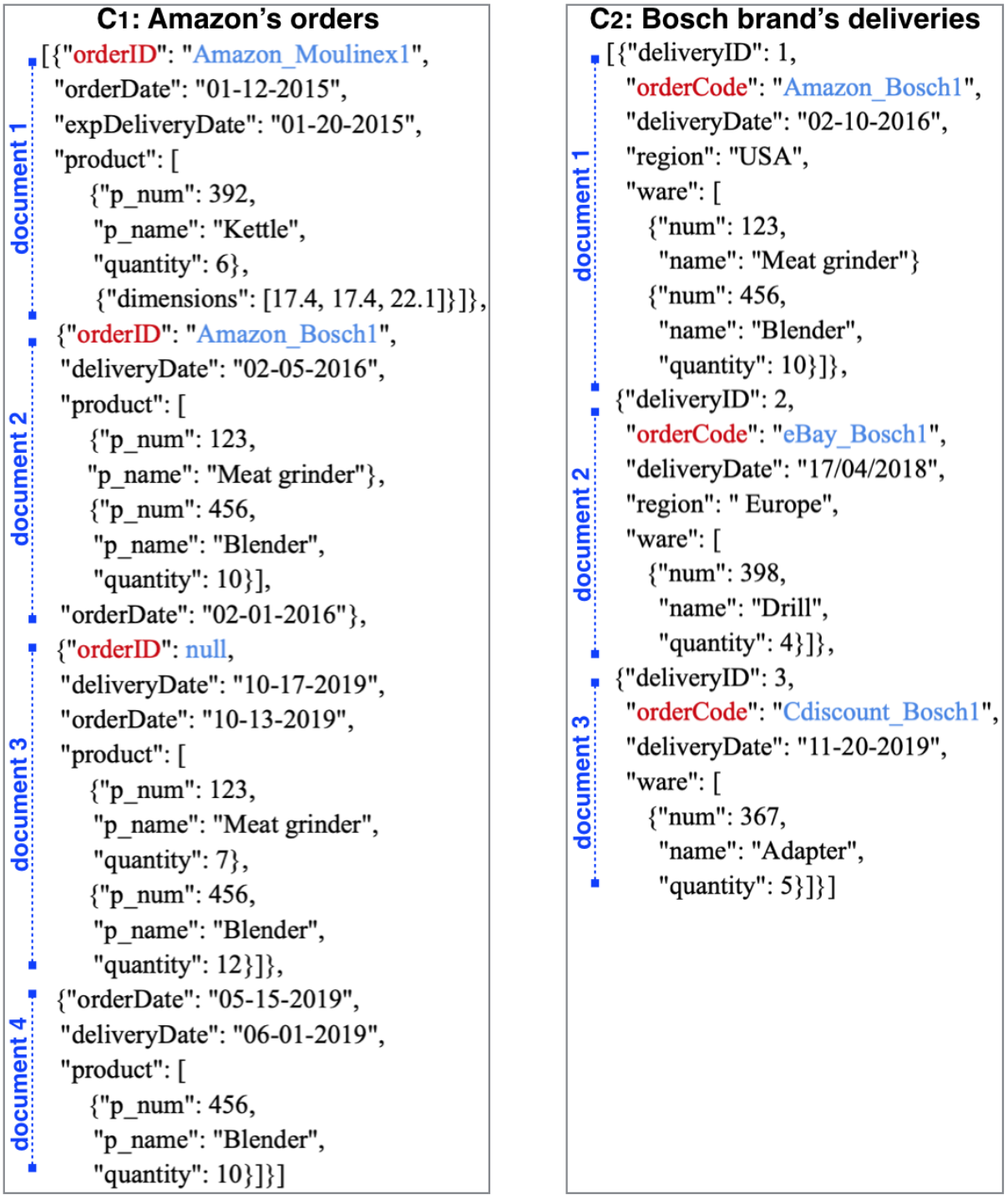

Figure 4.1: An excerpt of two collections

it is of paramount importance to correctly join C_1 and C_2 in order to compute the DD metric. The key fields that join C_1 and C_2 are orderID as an *identifier* in C_1 and orderCode as a *reference* in C_2. If we use existing algorithms dedicated to relational databases in order to automatically detect join keys, it would be unfitting. In fact, the orderID in C_1 has a null value in the third document and is absent in the fourth one. Additionally, the set of orderCode values: {Amazon_Bosch1, eBay_Bosch1, Cdiscount_Bosch1} is not included in the set of orderID values: {Amazon_Moulinex1, Amazon_Bosch1, Amazon_KenWood1}.

By and large, document stores have different aspects of heterogeneity that could be:

- **Intra-document**: the variety is mainly related to heterogeneous structures within a document. For instance, a JSON array can hold objects with different fields. As shown in Figure 4.1 (document 1 of C_1), the product field is a JSON array that contains different objects in terms of fields number and types. The first object comprises three fields, while the second contains an array of simple values (i.e., [17.4, 17.4, 22.1]).

- **Inter-documents**: the variety is mainly related to documents in different fields. Fields can be present in some documents and absent in others. For instance, the product field in the first document of C_1 does not contain the same fields as in the second document. Moreover, fields do not have the same order (e.g., orderDate), which is an intrinsic characteristic of an object.

- **Inter-collections**: the variety lies in having different schemas with different knowledge domains.

In document stores, joining two collections is a thriving challenge because of their clueless schemaless nature. In fact, *(i)* unlike the primary key which is unique and not null, *identifier* as all the other fields, can be missing in some documents or can, normally and not exceptionally, have null values; *(ii)* document stores do not have "precise" join keys beforehand due to the absence of integrity constraints; and *(iii)* unlike a relational database, *reference* values are not included in the *identifiers*' values, so it is impossible to use the inclusion dependencies in order to automatically detect *identifiers* and *references*.

4.3 Overview of our Approach

We summarize our approach in Figure 4.2, where we present the sequential order of the three stages. The primary input is the heterogeneous collections on which the IRIS-DS is processed. Collections are document stores in JSON format. The output of the IRIS-DS is a set of candidate pairs of identifiers and references. The core stages of our approach are:

- schemas extraction
- discovery of candidate *identifiers*
- identifying candidate pairs of key fields

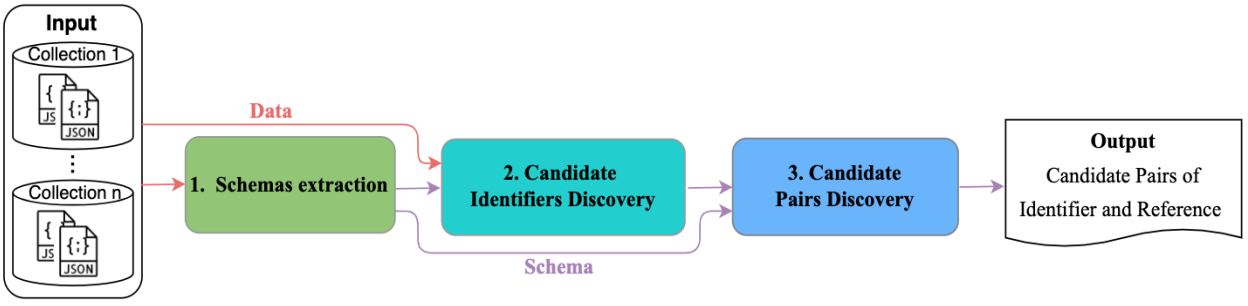

Figure 4.2: Our approach overview

4.4 Discovery of Candidate Identifiers

In this stage, we restrict our focus on the discovery of candidate *identifiers* on which depends the identification of the pairs (*identifier*, *reference*) afterwards. Hence, this stage aims to start with identifying an initial list of candidate *identifiers* for each collection and come out with a refined list after the scoring and the pruning phases.

4.4.1 Identifying the Initial List of Candidate Identifiers

Let us consider a collection C, and its schema $S(C) = T_{cx} \cup T_s$, where T_{cx} is the set of fields with complex types (JSON object or JSON array) and T_s the ones with simple types (primitive types).

Since an *identifier* can not, probably, be a *JSON object* nor a *JSON array*, then we limited the search space of candidate *identifiers* to the ones having simple types (T_s). In JSON format, each element in an array can be of three types: objects (set of key-value pairs), arrays, and/or simple values (Integer, String, Boolean, etc.). Since we are looking for candidate *identifiers*, we are interested in JSON elements represented as key-value pairs, whether nested in objects or arrays. For instance, in document 1 of C_1 (cf., Figure 4.1), *product* is an array of objects (complex structure), we consider its nested key-value pairs, whereas "dimensions" is an array of simple values that can not be a candidate identifier, so they are not considered.

Moreover, due to schema flexibility, documents within the same collection may present some structural variety. Some fields are not present in all documents or may have null values. Thus, we classify fields in T_s as being required (F_r) or optional (F_o) (cf., Definition 9). We limited the search space of candidate *identifiers* to the required fields within F_r. Then, within F_r, we look for single fields and combinations of fields having unique values. Throughout this chapter, we use the acronym IDc to refer to a candidate *identifier*(cf., Definition 10) which can be constituted of one or more fields. We note that we regard only minimal unique fields' combinations. To put it another way, let $\texttt{Comb} = \{f_1 \dots f_n\}$ be a combination of fields. `Comb` is considered as a minimal unique combination if $\forall f_i \in \texttt{Comb}$ is not unique. The generation of these combinations is done steadily. Firstly, we look for the IDc made up of single fields. Secondly, we generate combinations of two fields from the remaining list of non-unique fields that are frequent and of simple types. We repeat the same step for the triad combinations.

Checking unique values brings us back to evoke the problems related to duplicate detection in case of missing values. To better understand this point, consider for example two instances' values of a composite identifier: ("1", "2") and ("1", "null"). This case calls into question some past assumptions: *(i)* assume that the two instances' values are identical; or *(ii)* assume that the two instances' values are different. Multiple strategies have also been proposed to deal with missing values.

By and large, an identifier must be unique and not null [Pejcoch, 2014]. However, owing to schema flexibility, any field in each collection can be missing in some documents or have null values. Thus, we apply the uniqueness checking only to required fields (cf., Definition 9). If the field is required (with a high frequency of appearance with values different from null and missing), we verify the uniqueness constraint. The latter is verified based on non-null values [Berti-Équille et al., 2019]. If so, the field is retained as a candidate *identifier*, and its score is computed in the following stage.

Definition 9 *(**Required Field**) A field f_i is required whenever its frequency is greater than or equal to a threshold ε. The frequency is computed as $freq(f_i) = \frac{|\widetilde{k}_c|}{|D_c|}$ [Chouder et al., 2019], where $|\widetilde{k}_c|$ is the number of documents in which the key in the given field is not missing and has a not null value, and $|D_c|$ stands for the total number of documents within the collection C.*

Definition 10 *(**Candidate Identifier**) Given a collection C, a candidate identifier (IDc) is one or more fields that are of simple types, required, and form a minimal combination of unique values.*

4.4.2 Scoring Candidate Identifiers

In the context of relational sources, we explored several primary key features in the literature [Jiang and Naumann, 2020, Papenbrock and Naumann, 2017] to distinguish valid primary keys from spurious ones. We reuse some of these features that we have adapted to the context of document stores in our proposal, and we introduce extra features: **depth**, **data type**, and **name prefix**. We describe these features in the following.

- **Cardinality**: in practice, schema designers show a tendency to use fewer fields for the identifier definition: fewer fields enable better understandability and maintainability. The score function is defined as $\frac{1}{|IDc|}$.
- **Name prefix/suffix:** *identifiers* are generally identified by their field name prefix/suffix. We consider the list of possible names' prefixes/suffixes for identifiers as: "id", "key", "nr", "no", "pk", "num", and "code". We define the score function as $\frac{prefixSuffix(IDc)}{|IDc|}$, where $prefixSuffix(IDc)$ counts the number of fields in the IDc whose name contains one of the prefixes/suffixes mentioned above.
- **Depth:** *identifiers* often have a shallow depth. In fact, nested fields has a lower chance to be an *identifier* for the entire collection. We define the score function as $\frac{1}{|IDc|}(\sum_{i=1}^{|IDc|} \frac{1}{depth(f_i)+1})$, where $f_i \in IDc$.
- **Data type:** hands-on hints show that a field is likely to be an *identifier* whenever its data type is Integer or String.

  We define the data type score as $\frac{1}{|IDc|}(\sum_{i=1}^{|IDc|} type(f_i))$ where $type(f_i)$ is a binary function that returns one if the field f_i has a *String* or an *Integer* type or zero otherwise.

- **Value length:** fields that are used as *identifiers* are supposed to have a short value length, as they are typically non-semantic *identifiers*. The score function is defined as $\frac{1}{max(1,\overline{LengthMax(f_i)}-n)}$, where
  - $\overline{LengthMax(f_i)}$ is the average length of the longest values associated to the IDc fields. This function is defined as $\overline{LengthMax(f_i)} = \frac{1}{|IDc|}\sum_{i=1}^{|IDc|} LengthMax(f_i)$.
  - n is a parameter used to penalize long values.

  Value length is a value-based feature, making it cumbersome to use in a NoSQL context. To take advantage of this critical feature, we use the random sampling technique as reported earlier (cf., Subsection 3.4).

We use these features to score each candidate *identifier* related to each collection. We use the overall average of the computed scores for the total score.

4.4.3 Pruning Candidate *Identifiers*

Expectedly, the set of the initial candidate *identifiers* is very large. Filtering techniques are essential to get rid of irrelevant candidate *identifiers*. For this, for each collection, we score each candidate *identifier* using the above-described features. In our approach, we use the *Cliff* technique [Jiang and Naumann, 2020] (cf., Definition 11). As described in Example 3, the set of candidate *identifiers* is split into two parts: *(i) Upper*: it contains the candidates before the *Cliff*; and *(ii) Lower*: it contains the remaining candidates. Since the candidates that appear in the *Upper* part do have the highest scores, we prune the candidates belonging to the *Lower* part. We note that in case of multiple instances of *Cliff*, we retain all candidates before the last *Cliff* (cf., Definition 12).

Definition 11 *(**Cliff** [**Jiang and Naumann, 2020**]) Given $S = \{S_1, S_2, \ldots, S_n\}$, the sorted score list of candidate identifiers belonging to one collection, and their corresponding score difference list, $SD = \{SD_1, SD_2, \ldots, SD_{n-1}\}$, where a score difference is defined as $SD_i = S_i - S_{i+1}$ of each pair of adjacent candidates, the Cliff is the pair of adjacent candidates S_i and S_{i+1} having the largest SD score.*

Definition 12 *(**Multiple instances of the *Cliff***) Given $SD = \{SD_1, SD_2, \ldots, SD_{n-1}\}$ the set of score differences where SD_i is the largest score difference. $\forall SD_j \in SD$, if $\exists SD_j \mid SD_j = SD_i$ such that $j \neq i$ then we prune the lower part of SD_k, where*

$$k = \begin{cases} i\ , & S_i < S_j \\ j, & otherwise \end{cases} \tag{4.1}$$

Example 3 *As depicted in Figure 4.3, we suppose having a list S of candidate identifiers' scores, which are decreasingly sorted as follows: $S = \{1.0, 0.6, 0.58, 0.39, 0.23, 0.1\}$. We generate the score difference list $SD = \{0.4, 0.02, 0.19, 0.16, 0.13\}$. The Cliff is the largest score difference value in SD, i.e., 0.4. The green and the pink squares, shown in Figure 4.3, respectively illustrate, the Upper and the Lower parts.*

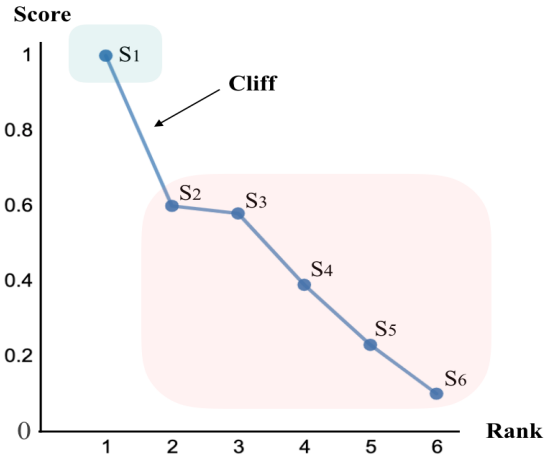

Figure 4.3: Example of the *Cliff* method applied to the ranked scores of candidate *identifiers*

The pruning phase is dedicated to refining the initial list of candidate *identifiers* for each collection. Furthermore, the refined list identifies candidate pairs of key fields as detailed in the remainder.

4.5 Identifying Candidate Pairs of Identifiers and References

This stage aims to constitute the pairs of *identifier* and *reference* fields related to every two document stores. Given two collections, we search the candidate pairs in both directions. Roughly speaking, we firstly find for each candidate *identifier* $IDc(C_1)$ of the first collection C_1 the most similar candidate *reference*, if it exists, from the set of fields $F(C_2) = f_1, \ldots, f_n$ of the second collection C_2. Secondly, we find for each $IDc(C_2)$ the most similar candidate *reference*, if it exists, from $F(C_1) = f_1, \ldots, f_n$. We filter the obtained candidate pairs afterward.

The degree of similarity depends on the shared properties' values between an *identifier* and its *reference*. In order to inspect these similarities, we model the relation between $IDc(C_i)$ and $F(C_j) = f_1, \ldots, f_n$ as a graph.

Graph-based techniques are becoming ubiquitous since they are essential to yield new insights into data. However, graphs with their traditional representation do not allow entirely take benefit from the existing machine learning approaches and techniques [Cui et al., 2019].

In recent years, there has been considerable interest in graph embedding that aims to convert graph nodes into a lower-dimensional space in which the neighborhood similarity between nodes is preserved in the embedded space [Cai et al., 2018]. To this end, using embedding and vector spaces offers a richer toolset and machine learning approaches. Our objective aligns with the graph embedding goal, particularly the node embedding one.

Hence, we uptake a graph embedding technique, called *node2vec* [Grover and Leskovec, 2016]. The latter learns feature representations for the nodes across a graph for different machine learning tasks. *Node2vec* explores network neighborhood. It designs a flexible neighborhood sampling, called a random walk, by interpolating between Breadth-first and Depth-first sampling strategies. This algorithm is of quadratic complexity.
The graph we defined is simple (do not allow multiple edges), heterogeneous (with nodes of different types), undirected and unweighted.

We denoted it as $G(V, E)$ where:

- $\boldsymbol{V}$**=** $\{IDc(C_i), F(C_j), PV\}$: the set of vertices (aka nodes) with three types, where
  - $IDc(C_i)$: a candidate *identifier* of the collection C_i.
  - $F(C_j)$: the set of fields of the collection C_j.
  - PV: the set of properties' values related to both identifiers annode2d references, e.g., String is a value of the data type property.
- $\boldsymbol{E}$: the set of edges that link and define the present relationships between nodes. An edge can be established between:
  - a node of an identifier $IDc(C_i)$ and a node of a candidate *reference* from $F(C_j)$ if this pair has a syntactic or semantic similarity greater than a threshold.

– a field in $\{IDc(C_i), F(C_j)\}$ and a property value.

Example 4 *Figure 4.4 shows an example of the input graph where the blue nodes represent the fields belonging to both collections* `CountryRegion` *and* `Orders_customer` *and the green nodes represent their properties' values. The candidate identifier of the collection* `CountryRegion` *is* `CountryKey`*. The colored edge linking the two nodes* `CountryKey` *and* `NationKey` *shows the existence of a syntactic or semantic relationship between the two nodes. We note that for simplicity, we use only a simple label for each node, whereas in the schema, each field is identified by its full path from the document root.*

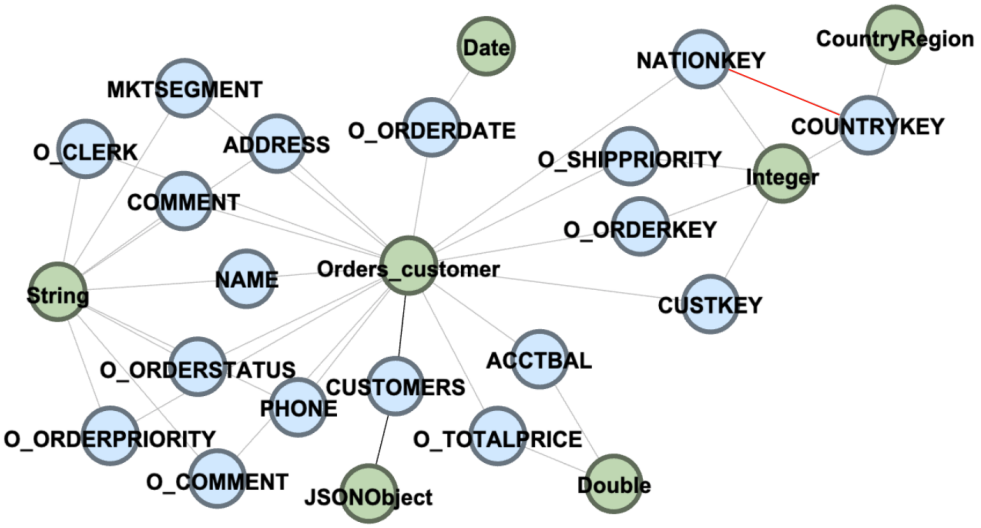

Figure 4.4: Example of the input graph for *node2vec*

We propose a set of rules to identify the properties related to identifiers and references:

- **Rule 1: Compatibility of data type:** the identifier and its reference must have the same type or compatible types. For example, if we have an *identifier* with a *String* type and a *reference* with an *Integer* type, and they are not convertible, this pair will not be considered in this case. Our approach covers all possible combinations of primitive types, e.g., (*String*, *String*), (*String*, *Double*), (*Integer*, *Double*), (*Short*, *Double*). As mentioned earlier in the global schema extraction stage, since an actual primitive type can be hidden under another primitive type, we check the type of each field pair to detect such cases.

- **Rule 2: Syntactic similarity-based pruning:** in many instances, fields' names are not randomly

assigned for better understanding. Hence, taking into account the similarity between the fields' names of each pair could be a kick-off beacon. To this end, we use a syntactic similarity measure, and we opt for the *Fuzzy-Token* similarity since it is the most suitable for our case [Wang et al., 2011]. Thus, the similarity combines both token-based similarity and string similarity. To use this similarity function, the input strings s_1 and s_2 are tokenized. We consider both cases for the tokenization: *(i)* having a delimiter, e.g., "_" and/or uppercase letter; *(ii)* strings are attached without a delimiter, e.g., "LINESTATUS". The function is defined as $syntac(s_1, s_2) = \frac{|T_1 \tilde{\cap}_\sigma T_2|}{|T_1|+|T_2|-|T_1 \tilde{\cap}_\sigma T_2|}$, where s_1 is the *reference* name and s_2 is either the *identifier* name or the collection name of that identifier. Then, we retain the maximum value obtained between the two similarity measures. We note that s_1 and s_2 are amended compared to [Jiang and Naumann, 2020][1]. In addition, T_1 and T_2 are the tokens' sets related to s_1 and s_2 respectively and σ is the edit distance threshold used to penalize lower similarities. To compute this similarity, a weighted bigraph should be constructed using T_1 and T_2. The weight, i.e., edit distance measure, is assigned to each edge. Then, we keep only the edges with a weight larger than σ. The fuzzy overlap, denoted by $|T_1 \tilde{\cap}_\sigma T_2|$, is used to define the maximum weight matching of the constructed graph. Note that if $|T_1|$ or $|T_2|$ are more significant than one, we filter them from the set of possible suffixes or prefixes such as "key" and "id." For instance, by considering the two fields "CountryKey" and "CustomerKey," the syntactic similarity measure may rise due to the common suffix "Key," which becomes misleading to get the correct result and enhance the probability of getting false-positive results. On the other hand, the case of fields "Id" and "UserId" reveals the necessity to keep T_1 and T_2 without the filtering step.

- **Rule 3: Semantic similarity based pruning:** Using only the syntactic similarity between two fields is not sufficient to cover all cases, e.g., `customer` and `client`. It indeed leads to generating some false-positive and false-negative results. To this end, we propose a filtering step based on semantic similarity. To do so, we use the *Wup* semantic similarity measure (cf., Definition 13), which is based on the lexical database WordNet[2]. Like the syntactic measure, we use tokenization to divide the attached words into meaningful separated words. Given two fields f_1 and f_2,

[1] In [Jiang and Naumann, 2020], the authors have concatenated the table name for both primary key and foreign key presented in an inclusion dependency. However, it remains unclear to concatenate table names for both of them because, generally, the foreign key is likely to be similar to the name of the referenced table, but the inverse rarely happens.

[2] `https://wordnet.princeton.edu/`

we split each of them into a set of tokens, T_1 and T_2 respectively. We consider f_1 and f_2 semantically similar if exists at least a semantic similarity between elements of a pair (t_1, t_2), where $t_1 \in \{T_1 \setminus SP\}$ and $t_2 \in \{T_2 \setminus SP\}$. We denote with SP the set of possible suffixes or prefixes such as "key" and "id". Likewise the syntactic similarity, we deal differently with a unary set of tokens (T_1 or T_2) by considering all of the tokens without a filtering step.

Definition 13 *(**Wup similarity [Pedersen et al., 2004, Wu and Palmer, 1994]**) Wup is a path-based semantic similarity. Given two concepts, it finds the path length to root from the Least Common Subsumer (LCS), the most specific concept they share as an ancestor. The Wup similarity is computed as follows:* $sim_{Wup} = \frac{2 \times depth(LCS(C_1,C_2))}{depth(C_1)+depth(C_2)}$ *where* $depth(C)$ *is the depth of the concept in the WordNet hierarchy.*

Based on the rules defined above, we define the properties that concern both identifiers and references, namely: collection name, data type, IsSyntacticallySimilar, and IsSemanticallySimilar.

4.6 The IRIS-DS Algorithm

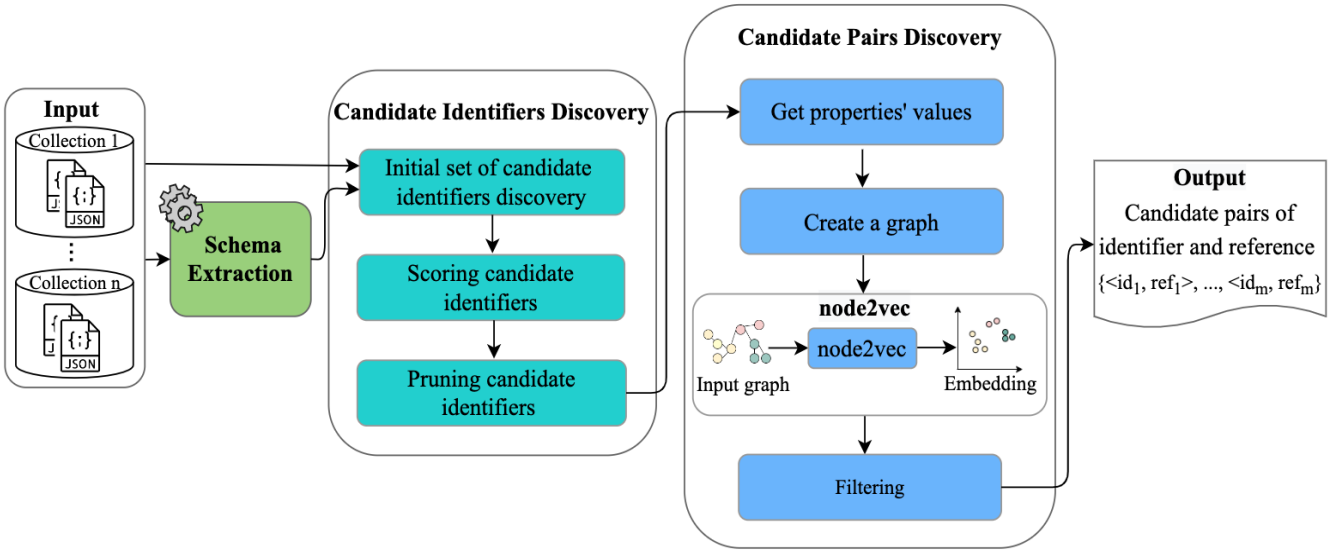

Figure 4.5: IRIS-DS general architecture

Figure 4.5 shows the overall process of our proposed algorithm IRIS-DS. Starting from several collections, it firstly extracts the collections' schemas. Secondly, it discovers the initial set of candidate

identifiers that will be refined after the scoring and the pruning steps.

Then, it performs a graph embedding technique to track down the set of candidate pairs of *identifier* and *reference*.

The pseudo-code is sketched in Algorithm 1, which in turn invokes various methods that are detailed separately.
In line $A_1.L_2$, i.e., Algorithm 1, Line 2, we start with the extraction of collections' global schemas. In line $A_1.L_3$, we search candidate *identifiers* from the set of fields presented in collections' global schemas. This step is explained separately in Algorithm 2.

In line $A_1.L_4$, the list of collections' pairs, denoted as L, is generated using the Cartesian product while keeping only pairs with different elements, i.e., each collection pair (CP) is defined as $CP = (C_i, C_j)$ where $C_i \neq C_j$. The cardinality of L is defined as $|L| = \frac{|\mathbb{C}|(|\mathbb{C}|-1)}{2}$, where $|\mathbb{C}|$ is the number of distinct collections.

Algorithm 1 IRIS-DS

Input: Collections $\mathbb{C}$
Output: Pairs of candidate *identifier* and *reference* for each collections' pair $IRcand$

```
1  LCP1 ← ∅, LCP2 ← ∅
2  SG_C ← GenerateCollectionsSchemas(C)
3  SearchCandidateIDs(C, SG_C)
4  L ← GetListOfCollectionsPairs()
5  foreach CP=(C_i, C_j) in L do                          ▷ |L| = |C|(|C|−1)/2
6      if GetID(C_i) ≠ ∅ then
7          LCP1 ← discoverPairs(C_i, C_j)
8      end
9      if GetID(C_j) ≠ ∅ then
10         LCP2 ← discoverPairs(C_j, C_i)
11     end
12     IRcand ← filter(LCP1, LCP2)
13     Store(C_i, C_j, IRcand)          ▷ Store: store the IRcand for each collection pair
14 end
15 return IRcand
```

The idea is to iterate over L to find the set of pairs of candidate join keys (*identifier*, *reference*), if they exist, between every two collections (lines $A_1.L_{5-14}$). We consider both directions: a field in C_i can refer to another field in C_j (lines $A_1.L_{5-8}$) or vice versa (lines $A_1.L_{9-11}$). The pairs discovery assured in line $A_1.L_7$ and line $A_1.L_{10}$ are detailed in Algorithm 3 where the loop from line $A_3.L_1$ to

Algorithm 2 SearchCandidateIDs()

Input: Collections $\mathbb{C}$, Collections schemas $SG_{\mathbb{C}}$
Output: Initial list of candidate *identifiers* IL

```
1  I ← ∅, R ← ∅, U ← ∅, IDs ← ∅, L ← ∅
2  foreach C in ℂ do
3  |   I ← GetFieldsWithSimpleTypes(SG_ℂ)
4  |   R ← GetRequiredFields(I)
5  |   U ← GetUnique(R)
6  |   S ← Score(U)
7  |   IDs ← Cliff(S)
8  |   L ← L ∪ IDs
9  end
10 return L
```

line $A_3.L_{14}$ iterates over the list of *identifiers* of the current collection C_i, which are generated with *GetID(C_i)*.

In line $A_3.L_3$, we start by creating the input graph G for the embedding. Building the graph has a linear time complexity O(n) as execution time depends on the size of the collection schema. The properties values are among the constituting nodes of the input graph. We detail the extraction of the properties values in Algorithm 4. In lines $A_3.L_{4-5}$, we apply the *node2vec* algorithm, and we generate the model after learning feature representations for the nodes across the graph. The loop in lines $A_3.L_{6-10}$, iterates over the field(s) of an *identifier* and search the most similar field(s) that will be considered as a candidate *reference*.

In Algorithm 4, based on the rules mentioned above, we search the properties' values related to an *identifier* of the first collection and the set of fields of the second collection. In line $A_4.L_3$, we find the data type of the field *f*. In line $A_4.L_5$, we check if the current field and the *identifier* have a syntactic similarity measure equal to or greater than a given threshold. If they are syntactically similar, we assign to f the name of the identifier as a property value. In this way, an edge will be established between the *identifier*'s node and the field's node. Similarly, we verify the semantic similarity between the *identifier* and the current field.

Algorithm 3 discoverPairs()

Input: Collection C_i, collection C_j
Output: list of candidate pairs where the *identifiers* in C_i and reference in C_j LCP

foreach *IDc in GetID*(C_i) **do**
  $PV \leftarrow GetPV(IDc) \cup GetPV(GetFields(SG(C_j)))$ ▷ PV: properties values
  $G \leftarrow CreateGraphForEmbedding(IDc, GetFields(SG(C_j)), PV)$ ▷ G: input graph for node2vec
  $n \leftarrow node2vec(G)$
  $model \leftarrow LearnEmbedding(n)$ ▷ Learning node representations $R \leftarrow \emptyset$
  foreach *part in IDc* **do**
    if $model.GetSimilar(part) \neq \emptyset$ **then**
      $R \leftarrow R \cup$ model.GetSimilar(part) ▷ list of the most similar fields to IDc
    end
  end
  if $|IDc| = |R|$ **then**
    $LCP \leftarrow LCP \cup (IDc, R)$ ▷ LCP: list of candidate pairs
  end
end
return LCP

Algorithm 4 GetPV()

Input: List of fields L, candidate *identifier* IDc, collections' names CN(L) and CN(IDc)
Output: properties values PV

$PV \leftarrow \{CN(L), CN(IDc)\}$ ▷ add collections' names as properties values
foreach f *in* L **do**
  $t \leftarrow dataType(f)$ ▷ get the data type of the field f according to Rule 1
  $PV_f \leftarrow PV_f \cup t$
  if *CheckSyntacticSimilarity()*$=$ *true* **then** $PV_f \leftarrow PV_f \cup IDc$ ▷ check if the field f is syntactically similar to IDc
  else $PV_f \leftarrow PV_f \cup null$
  if *CheckSemanticSimilarity()*$=$ *true* **then** $PV_f \leftarrow PV_f \cup IDc$
  else $PV_f \leftarrow PV_f \cup null$
  $PV \leftarrow PV \cup PV_f$
end
return PV

4.7 Case Study

Figure 4.6 shows two JSON collections, i.e., `Orders_customer` and `CountryRegion`, that are based on the TPC-H[3] benchmark. This benchmark comprises relational sources that we have transformed into JSON collections[4]. For readability, we only present an excerpt of one document from

[3] Decision support benchmark: http://www.tpc.org/tpch/
[4] https://github.com/souibguimanel/TPCHjson

each collection. Based on this benchmark, our basic scenario is to perform a join operation between `Orders_customer` and `CountryRegion`. In doing so, we should perform *identifiers* and *references* discovery between the two aforementioned collections.

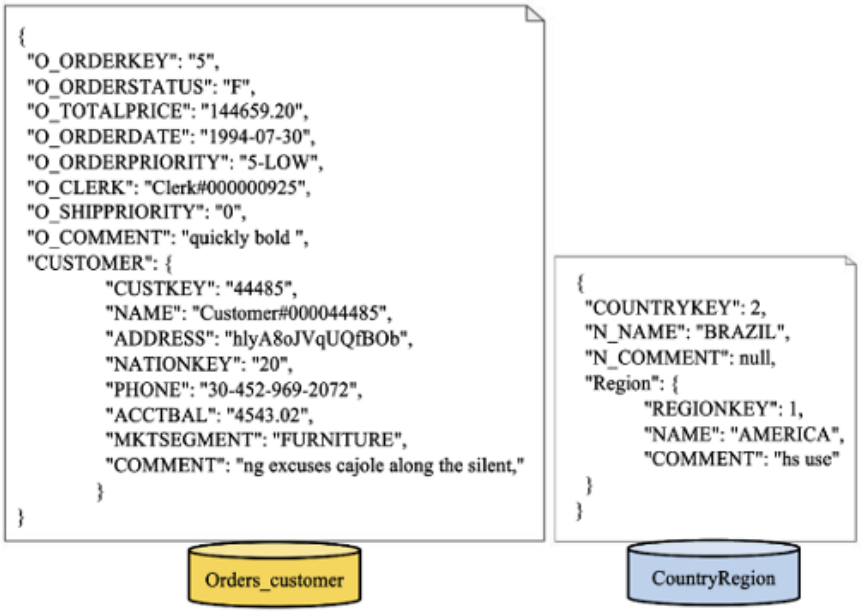

Figure 4.6: Sample JSON documents of the TPC-H benchmark

Figure 4.6 shows two JSON collections, i.e., `Orders_customer` and `CountryRegion`, that are based on the TPC-H[5] benchmark. This benchmark comprises relational sources that we have transformed into JSON collections[6]. For readability, we only present an excerpt of one document from each collection. Based on this benchmark, our basic scenario is to perform a join operation between `Orders_customer` and `CountryRegion`. In doing so, we should perform *identifiers* and *references* discovery between the two aforementioned collections. We start by identifying an initial list of candidate *identifiers* for each collection. Then, we present the obtained candidate *identifiers* for each collection in the second column of Table 4.1. We note that we present a field using its path from the document root, where *$* symbol represents the document root. For each candidate *identifier*, we compute its score based on the features described above. To prune irrelevant candidates for each collection, we search the cliff value after ranking the scores. We split the candidate *identifiers* into *Upper* and *Lower* parts. The *Upper* part of the collection `CountryRegion` contains `[$.COUNTRYKEY]`, and the *Upper* part

[5]Decision support benchmark: `http://www.tpc.org/tpch/`
[6] `https://github.com/souibguimanel/TPCHjson`

of the collection `Orders_customer` contains `[$.O_ORDERKEY] and [$.customer.CUSTKEY]`. The remaining candidate identifiers related to each collection are pruned since they are in the *Lower* part.

Table 4.1: Initial list of candidate *identifiers* with their scores

Collection	**Candidate *identifiers***	**Score**
`CountryRegion`	`[$.COUNTRYKEY]`	1.00
	`[$.N_NAME]`	0.63
`Orders_customer`	`[$.O_ORDERKEY]`	1.00
	`[$.Customers.CUSTKEY]`	0.90
	`[$.O_CLERK, $.customer.NATIONKEY]`	0.75
	`[$.customer.MKTSEGMENT, $.customer.NATIONKEY]`	0.70
	`[$.O_ORDERSTATUS, $.customer.NATIONKEY, $.O_ORDERPRIORITY]`	0.70
	`[$.O_ORDERSTATUS, $.O_CLERK]`	0.70
	`[$.O_TOTALPRICE]`	0.60
	`[$.O_COMMENT]`	0.60
	`[$.Customers.PHONE]`	0.52
	`[$.customer.NAME]`	0.52
	`[$.O_CLERK, $.O_ORDERPRIORITY]`	0.52
	`[$.customer.ACCTBAL]`	0.50
	`[$.O_ORDERDATE]`	0.50
	`[$.customer.COMMENT]`	0.50
	`[$.customer.ADDRESS]`	0.50
	`[$.customer.MKTSEGMENT, $.O_CLERK]`	0.49

We start by preparing the graph embedding input. As shown in Figure 4.7, we create three graphs:

- G_1: presents the relationships between the candidate *identifier* `[$.COUNTRYKEY]` with the different fields of the collection `Orders_customer`.
- G_2: presents the relationships between `[$.O_ORDERKEY]` with the different fields of the collection `CountryRegion`.
- G_3: presents the relationships between `[$.customer.CUSTKEY]` with the different fields of the collection `CountryRegion`.

The green nodes denote the candidate *identifiers* the first collection. The blue nodes indicate the fields of the second collection that can contain the candidate *reference*. Finally, the nodes of properties' values are presented with a pink color. The red edge in G_1 marks the existence of a semantic similarity between the two fields `[$.customer.NATIONKEY]` and `[$.COUNTRYKEY]`.

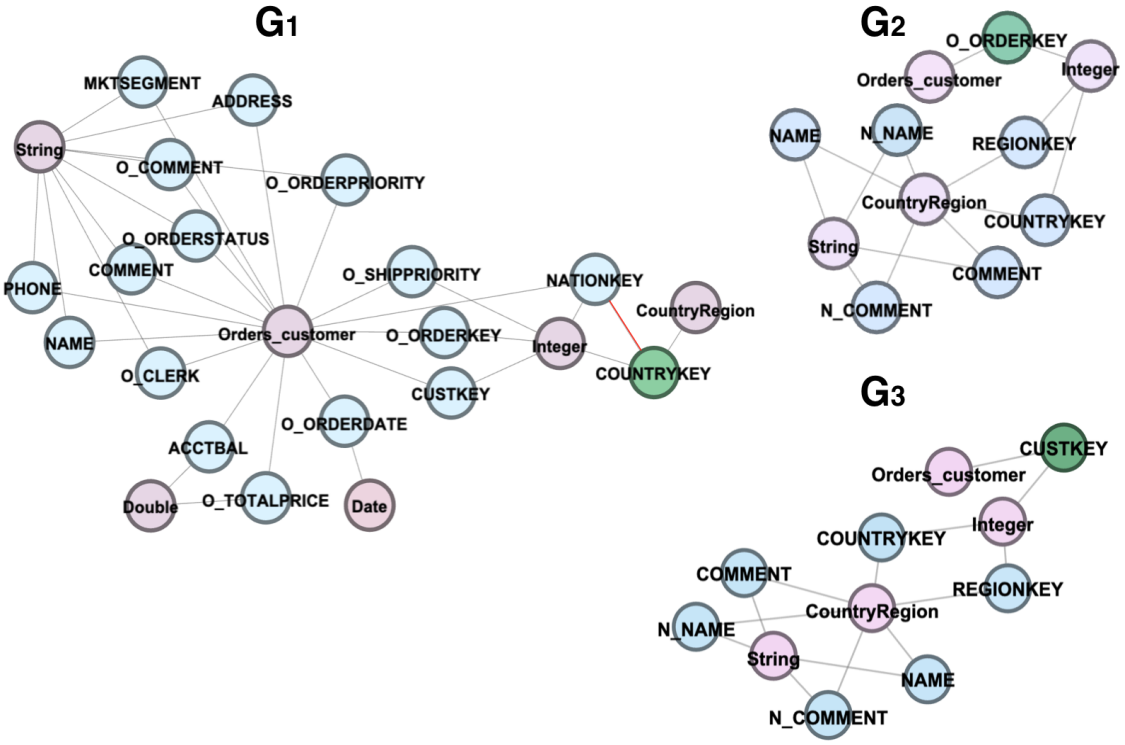

Figure 4.7: Node2vec input

To establish the relationships between nodes in each graph, we identify the values of the properties: collection name, data type, syntactic similarity measure, and semantic similarity measure. Since the two last properties require pairs of fields, we create pairs using the Cartesian product between a candidate *identifier* of the first collection and the group of fields of the second collection. Then, we compute the syntactic and semantic similarity measures for each pair.

The pair with a similarity measure greater than a threshold will have an edge between their representing nodes in the graph.

Each graph obtained in the previous step is taken as an input of the `node2vec` algorithm[7] to seek the key pairs based on the candidate *identifiers*. By performing the `node2vec` algorithm, we get the most similar fields for each identifier with the corresponding probabilities, which approximate the similarities between nodes. Regarding probabilities values, we keep the highest ones to identify the candidate pairs of key fields. We hosted the python code samples that are used in this step in a

[7] https://snap.stanford.edu/node2vec/

GitHub repository[8]. As a result, we obtain `[$.customer.NATIONKEY]` as the most similar field to the candidate *identifier* `[$.COUNTRYKEY]` with a probability measure in the region of 0.994.

4.8 Conclusion

Document stores have a variable schema, where fields can be missing in some documents or have null values. Because of the lack of integrity constraints and inclusion dependencies, a document store is a long way away from having an exact join key. In the literature, existing works have provided dedicated solutions to relational databases. For NoSQL data stores, current contributions rely on a strong assumption: having the join key pairs beforehand. We seldom know the pair of identifiers and references in document stores.
To this end, we have proposed a new approach for *identifiers* and *references'* discovery based on several document stores. We have introduced the IRIS-DS algorithm that discovers *identifier* candidates for each collection, and then identifies the candidate pairs of *identifier* and *reference* for every two collections. The sighting features of our approach are:

- to the best of our knowledge, no former approach has been dedicated to joining keys discovery in the context of document stores. We consider both composite and non-composite join keys.
- we adapt existing features, which identify candidate *identifiers*, to the context of document stores and introduce new ones.
- to detect the candidate pairs of *identifier* and *reference*, we uptake a graph embedding technique, called *node2vec* [Grover and Leskovec, 2016], which yields significant advantages.
- unlike existing works, we use both syntactic and semantic similarity measures for pruning pointless candidates.

[8] `https://github.com/souibguimanel/Get_similarities_node2vec`

Chapter 5

Experimental Study

Contents

5.1 Introduction

This chapter reports our experimental findings regarding discovering identifiers and references in document stores. As proof of concept of our approach, we have implemented java prototypes to support the main phases in order to validate and evaluate the relevance of our approach. We, firstly, present the technical architecture of our prototype in Section 5.2. Secondly, we describe the data collection and preparation in Section 5.3. We base our experimental study on two benchmarks and two real-world datasets. In Section 5.4, we describe the evaluation protocol, and finally, we discuss the experimental results in Section 5.5.

5.2 Technical Architecture of our Prototype

As proof of the concept of our approach, we have developed a prototype to support the main phases and tested them under macOS High Sierra machine, Processor Intel Core i5, 2.7 GHz, and 8 GB of DDR3 RAM. As shown in Figure 5.1, our prototype is implemented in java 8 using the integrated development environment Eclipse Java EE. We store the used collections of JSON documents on MongoDB as a document-oriented DBMS. To access the document stores, we use the MongoDB java driver. The latter establishes a connection with the MongoDB database by creating a MongoClient. The rest of the java code implements the core stages of our approach, namely document stores schema extraction, discovery of candidate identifiers for each collection in the MongoDB database, and creation of the graph that will be used as an input of the graph embedding algorithm. The creation of the graph requires computing syntactic and semantic similarities. Since, we use a token-based syntactic similarity, we use the python Wordninja library[1] to split the attached words into tokens. As for the semantic similarity, we use the WS4J API to exploit the large lexical database WordNet in java.
Finally, to discover candidate pairs of *identifiers* and *references*, we use node2vec as a graph embedding algorithm. This algorithm is implemented in python 3 and available on GitHub. We ran this algorithm on Google Colab[2], which provides free access to computer resources such as GPUs, making it suitable for machine learning and data analysis.

[1] https://pypi.org/project/wordninja/
[2] https://colab.research.google.com/

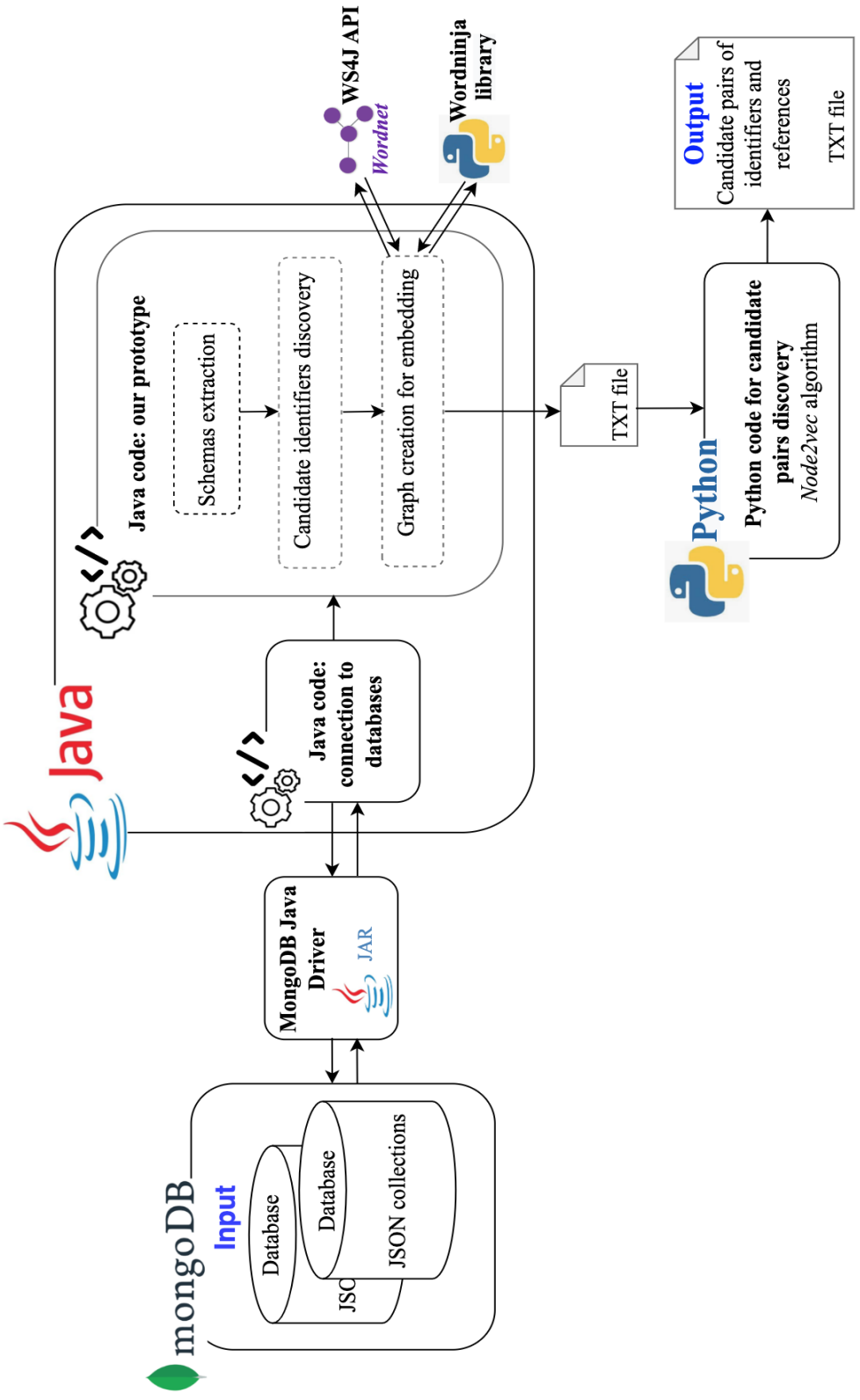

Figure 5.1: Technical architecture of our prototype

5.3 Data Collection and Preparation

We base our experimental study on two benchmarks and two real-world datasets:

- **TPC-H**: a benchmark for decision support systems. It represents the activity of any industry that manages, sells or distributes a product worldwide.

- **TPC-E**: a benchmark offering a set of flat files that models a brokerage firm with customers who start transactions concerning trades, account inquiries, and market research.

- **Twitter**: these datasets, which comprise users and their related tweets, are scraped from Twitter's API. We consider two collections: `Tweets` and `Users`. Each document of the `Tweets` collection contains nested objects such as the coordinates object that gives the geographic location. These datasets are initially in JSON format. They are heterogeneous in terms of schema variety, different nesting levels (i.e., field depth), missing values, different types, etc.

- **Musicians**: are datasets extracted from Wikidata[3], a real-world data source. These datasets are implemented in Valentine [Koutras et al., 2021] where the authors proposed a well-thought-out dataset creation process that is tailored to the scope of matching techniques. The datasets represent data about American singers. It contains around 11k tabular data that we converted into JSON documents. To resemble a real-life scenario, authors in [Koutras et al., 2021] have varied the names of the attributes in the source and the target sources included in Valentine. In addition, they provide with the source and the target dataset a JSON file containing the possible matching that we use as ground truth to check the accuracy of our results as shown in Figure 5.2.

Collections within considered datasets contain fields having distinct maximal depth values. For example, as shown in Table 5.1, the maximal depth in the Twitter datasets is three, whereas, in the TPC-H datasets, the maximal depth is two.

The idea is to start from sources for which we know the various links (identifier, reference) to verify the accuracy and consistency of the results generated by our algorithm. Thus, we used the two benchmarks TPC-H and TPC-E to compare the result obtained with those of the gold standard.

[3] https://www.wikidata.org/wiki/Wikidata:Main_Page

```
{
  "matches": [
      {
         "source": "MusiciansSource",
         "source_attribute": "musician",
         "target": "MusiciansTarget",
         "target_attribute": "musicianID"
      }

  ]
}
```


Figure 5.2: Example of a ground truth file for the Musicians datasets

Table 5.1: Minimal and maximal depth within the considered datasets

Datasets	Minimal depth	Maximal depth
TPC-H	1	2
TPC-E	1	3
Twitter	1	3
Musicians	1	1

Since our approach deals with document stores, we carried out a data preparation phase to use the two benchmarks TPC-H and TPC-E, in our experimental process as input to our prototype. We have implemented a transformation phase to convert their generated flat files to JSON ones. We consider each record in the flat file as a document in the JSON collection. We perform the data preparation stage regarding the document-oriented model features, e.g., randomly assigning null or missing values. Furthermore, to have different storage models, we have denormalized data in both benchmarks: *(i)* **TPC-H**, we have denormalized the `Orders` collection by embedding documents from `Customer`. Similarly, we have denormalised the `Nation` collection by embedding documents from `Region`; and *(ii)* **TPC-E**, we have denormalized the `Trade` collection by embedding documents from `TradeType`. Similarly, we have denormalized `CustomerAccounts` by embedding documents from both `Address` and `Customer` collections. This is done by replacing each foreign key with its full object. We host the generated data in a GitHub repositories[4,5] to make them openly available.

[4] https://github.com/souibguimanel/TPCHjson
[5] https://github.com/souibguimanel/TPCEjson

5.4 Evaluation Protocol

The experiments we conduct aim to validate our approach in terms of result relevance. The approach validation comprises both levels: *(i)* candidate *identifiers* discovery for each collection; and *(ii)* identification of candidate pairs of key fields (*identifier* and *reference*) for every two collections. To this end, for each level, we use four metrics:

- **precision**: the fraction of the predicted true *identifier*/pairs among the predicted *identifiers*/pairs.
- **recall**: the fraction of the predicted true *identifier*/pairs among *identifiers*/pairs of the gold standard.
- **accuracy**: the number of correct results returned by our algorithm.
- **percentage decrease**: rate the reduction of the number of candidates that will be proposed to the end-user. This metric aims to assess the relevance of the cliff technique, which prunes poor candidate identifiers. This metric is computed as $\frac{(OriginalNumber - NewNumber)}{OriginalNumber} * 100$.

  We distinguish two cases to compute the percentage decrease:

  - Discovery of candidate *identifiers* for each collection:
    - original number: the schema size of the given collection.
    - new number: the number of the detected candidate *identifiers*.
  - Identification of candidate pairs of key fields (identifier and reference) for every two collections:
    - original number: given two collection schemas, the original number is the sum of the cardinality of the Cartesian product between the candidate *identifiers* $IDc(C_1)$ of the first collection and the set of fields of the second collection $F(C_2) = f_1, \ldots, f_n$ and the cardinality of the Cartesian product between $IDc(C_2)$ and $F(C_1) = f_1, \ldots, f_m$.
    - new number: the number of the discovered pairs of identifier and reference.

5.5 Experimental Results

As shown in Tables 5.2 and 5.4, we compare the output sets of both candidate *identifiers* and candidate pairs (*reference*, *identifier*) with the gold standard of the Twitter collections, Musicians

collections, TPC-H benchmark, and TPC-E benchmark, and we report the precision, recall, accuracy, and the percentage decrease. The results show that our approach reaches a high precision and accuracy without diminishing the recall. Furthermore, the percentage decrease metric yields increasingly excellent results by reducing the number of candidates proposed to the end user.
We note that in Table 5.2, our algorithm shows a decrease in the precision to 0.25 when detecting the identifier in the `Financial` collection. This decrease is because of several non-composite unique fields, which generate false-positive results. However, this error accounts for only a tiny portion of the used collections, and fortunately, the percentage decrease is still high.

Although our approach is dedicated to heterogeneous document stores (consisting of diverse data contents), we have also used the two Musicians datasets as part of our experimental study, which encompasses homogeneous data devoted to evaluating matching techniques.

Table 5.2: IRIS-DS results for the candidate *identifiers* discovery in TPC-H, TPC-E and Twitter collections

	Collection	**# documents**	**Precision**	**Recall**	**Accuracy**	**Percentage decrease %**
TPC-H	`Orders`	1k	1	1	1	94.11
	CountryRegion	24	1	1	1	85.71
	Supplier	1k	1	1	1	94.73
TPC-E	`Trade_TT`	10k	1	1	1	94.73
	Financial	20k	0.25	1	0.73	71.42
	CustomersAccounts	10k	1	1	1	97.14
	Holding	10k	1	1	1	83.33
	AccountPermission	10k	1	1	1	80
Twitter	Tweets	1k	1	1	1	96.42
	TwitterUsers	10k	1	1	1	96.66
	Tweets	2M	1	1	1	97.61
	TwitterUsers	2M	1	1	1	97.29
Musicians	MusiciansSource	11k	1	1	1	92.30
	MusiciansTarget	11k	1	1	1	92.30

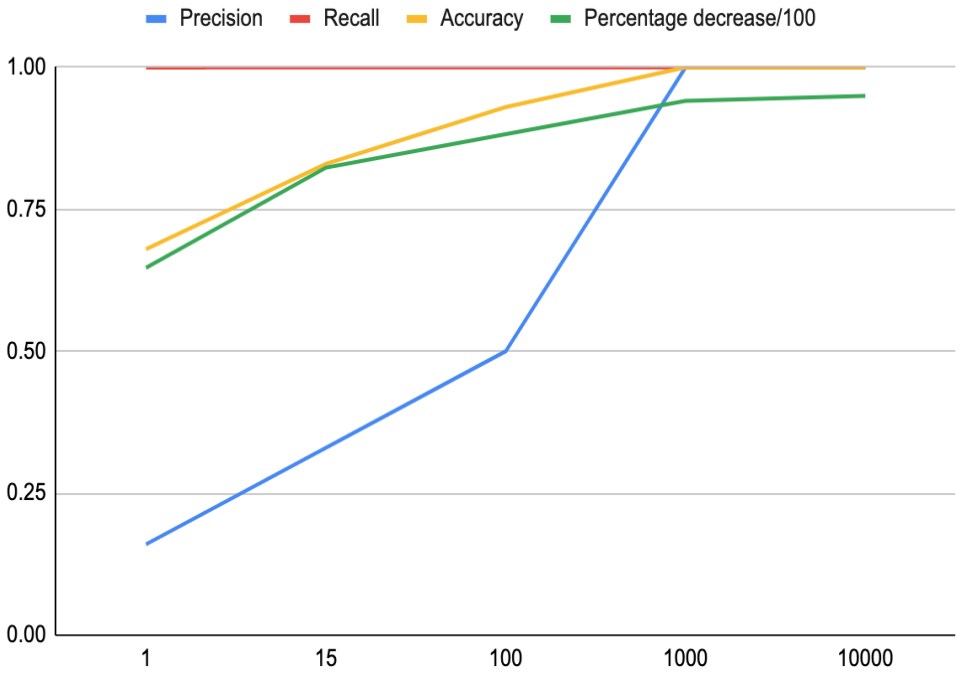

Figure 5.3: Impact of the dataset size: `Orders` collection

We carry out further tests that corroborated the correlation between the collection size and the values of the evaluation metrics. To gauge this effect, we have varied the `Orders` collection size, as shown in Figure 5.3. The result demonstrates that the more we increase the collection size, the better precision, recall, accuracy, and percentage decrease. The number of fields with unique values increases if we reduce the number of documents in the same collection. Although the precision often decreases when using a collection with few documents, the percentage decrease remains significantly high, reflecting the reduction of the final number of candidates proposed to the end user.

On the other hand, since we introduced new features to discover candidate *identifiers*, we perform additional tests to emphasize the importance of these features. For instance, the depth feature is intrinsic to document stores since collections' fields have different depths due to the embedding of objects inside documents. Thus, as shown in Table 5.3, by omitting this feature while computing the score for the `CustomersAccounts` collection, we realize the remarkable decrease in the precision, accuracy, and percentage decrease. Similarly, we remark a significant difference between their absence and presence for the data type and field name prefix features while computing scores and discovering candidate *identifiers*.

Table 5.3: Importance of the depth feature: `CustomersAccounts` collection

	Precision	**Recall**	**Accuracy**	**Percentage decrease**
Without depth	0.05	1	0.51	48.57
With depth	1	1	1	97.14

Table 5.4: IRIS-DS results for candidate pairs discovery in TPC-H, TPC-E and Twitter collections

	Collection 1	**Collection 2**	**Precision**	**Recall**	**Accuracy**	**Percentage decrease %**
TPC-H	`Orders`	`Nation`	1	1	1	95.83
	`Supplier`	`Orders`	N/A	N/A	1	100
	`CountryRegion`	`Supplier`	1	1	1	92.85
TPC-E	`Trade_TT`	`CustomersAccounts`	1	1	1	98.14
	`Holding`	`Trade_TT`	1	1	1	96
	`Financial`	`CustomersAccounts`	N/A	N/A	1	100
	`CustomersAccounts`	`Holding`	1	1	1	97.56
	`AccountPermission`	`CustomersAccounts`	1	1	1	97.50
	`Holding`	`AccountPermission`	N/A	N/A	1	100
Twitter	`Tweets`	`Users`	1	1	1	98.50
Musicians	`MusiciansSource`	`MusiciansTarget`	1	1	1	96.15

Table 5.5: Comparison of IRIS-DS with HoPF algorithm [Jiang and Naumann, 2020] applied on the TPC-H and the TPC-E benchmarks

		Identifier		(Reference, Identifier)	
	Algorithm	**Precision**	**Recall**	**Precision**	**Recall**
TPC-H	**HoPF**	1	1	0.88	0.88
	IRIS-DS	1	1	1	1
TPC-E	**HoPF**	0.80	0.80	0.72	0.91
	IRIS-DS	0.85	1	1	1

Since our approach considers several collections, we apply key pair discovery to every two collections. Indeed, there is at least one pair of collections that are not joinable, so they did not have a relationship (*reference*, *identifier*). Our approach can handle such cases. In fact, as depicted in Table 5.4, the pairs' discovery performed between the collections' pairs *(i)* `Supplier` and `Order`; *(ii)* `AccountPermission` and `Holding`; and *(iii)* `Financial` and `CustomersAccounts` returns no join key pairs. This implies that the precision and recall are N/A, i.e., Not Applicable because the number of true-positive values is null. For example, this might occur where the gold standard does not contain join key fields, and our algorithm returns no pairs correctly. We note that the parameter settings (*dimensions* and *walk-length*) of the *node2vec* algorithm that we used in our tests gives a greater result when they are set to respectively 10 and 30.

Since we are the first to propose an approach for *identifier* and *reference* discovery in document stores, we compare our algorithm against the most recent work [Jiang and Naumann, 2020] proposed in the context of relational databases, as shown in Table 5.5. Our approach and the HoPF approach share the same objectives: joining multiple sources without having the join keys beforehand. Our approach treats document stores, while the HoPF approach [Jiang and Naumann, 2020] is applied to relational databases. Relational databases have functional dependencies: (i) the primary key values are unique and not null; (ii) the foreign key values are included in the primary key values. However, document stores miss all these features. Roughly speaking, unlike the HoPF approach, we can not apply functional dependencies in the context of document stores. Hence, we have undergone a rethinking of the problem by using alternative methods adapted to document stores' schemaless nature, namely, adapting existing features and providing new features and pruning rules.

The results of our approach evaluation prove the viability of detecting automatically identifiers

and references pairs in document stores with dispersed data across many collections.

5.6 Conclusion

In this chapter, the experiments we conducted aim to validate our approach, in terms of result relevance. The approach validation involves both levels: *(i)* candidate *identifiers* discovery for each collection; and *(ii)* identification of candidate pairs of key fields (*identifier* and *reference*) for every two collections. For each level, we have used four metrics: precision, recall, accuracy, and percentage decrease. Outcomes from our approach's evaluation give insight into the feasibility of detecting join key fields in document stores containing scattered data over several collections. The carried-out experiments on the TPC-H and the TPC-E benchmarks, and the Twitter and Musicians datasets underscore that our approach fulfills the accuracy of the generated results.

Conclusion and Perspectives

Thesis summary

In this thesis work, we contribute to the field of Business Intelligence & Analytics, intending to exploit schema-free data scattered over several document stores for analytical purposes. There are three challenges for which we proposed a novel approach to deal with:

1. schemaless data sources that have to be extracted and transformed to fit decisions needs;
2. data sources that are numerous, scattered, and not known in advance by decision-makers;
3. The lack of integration among these heterogeneous data sources as they are usually produced by independent producers.

To deal with the two first points, we have introduced a new approach that aims to extract, transform, and load NoSQL data sources in an on-demand fashion. This approach is hybrid as it considers both data sources' schemaless nature and analytical needs to explore several NoSQL stores efficiently. We first start with schema extraction from the sources. We focused, particularly, on document stores and addressed the impact of the schemaless nature and heterogeneity within these data. Notably, unlike existing works, our approach is not limited to one collection, as we consider the dispersion of data across several collections.

Indeed, in Business Intelligence & Analytics, fetching relevant data that meet the decision-maker requirement often needs to access more than one data source. This implies facing mainly two major difficulties. The first is related to **independence between production context and usage context** of data. The consequence is an absence of complete match between the decision makers' needs and the data used. To avoid loading unnecessary data for the envisaged analyses, our approach proposes to reduce the loaded data by **mapping** the multidimensional attributes reflecting the decision makers' needs and the collections schemas. The second difficulty is related to the evolution of data sources. As it is crucial for analysis to be up-to-date regarding the data sources it is necessary to offer an approach allowing the revision of the aligned schema **on-demand**.

To deal with the lack of integration of data sources, our approach proposes to join sources by helping to discover the connecting fields. Unlike commercial tools and existing approaches that propose to perform the join on attributes manually using the key fields proposed by the analyst, our approach is based on an automatic algorithm that we have developed.

Hence, we study the problem of discovering join fields automatically. We propose an algorithm that aims to automatically detect both *identifiers* and *references* on several document stores. The *modus operandi* of our approach underscores three core stages: *(i)* global schema extraction; *(ii)* discovery of candidate *identifiers*; and *(iii)* identifying candidate pairs of *identifier* and *reference* fields. We use scoring features and pruning rules to discover actual candidate *identifiers* from many initial ones efficiently. To find candidate pairs between several document stores, we put into practice *node2vec* as a graph embedding technique, which yields significant advantages while using syntactic and semantic similarity measures for pruning pointless candidates.

We claim that our findings might be useful for:

- **Business Intelligence & Analytics systems**, particularly ETL processes dealing with document stores: today, exploiting NoSQL data for analytical purposes lacks maturity, traceability, and metadata management. In the context of BI&A, data are often scattered over distinct sources and several collections of documents. Hence, fetching relevant data that fits the decision-maker's needs often require access to several document stores, thence needs to use the join operation.

  Moreover, open-source and commercial ETL tools offer to join components. However, it is up to the user to choose the join keys before applying the join operation. In a NoSQL context, a document may have hundreds of fields, making it tricky to fetch the identifier of a given collection manually.

- **Querying tasks**: The join is mandatory for querying tasks. Hence, it is compulsory to have the join keys beforehand. For instance, MongoDB uses the MQL (MongoDB Query Language) to query stored data in document databases using different operators. Furthermore, since developers know well that joining operations among collections is essential [Celesti et al., 2019], MongoDB, an open-source community, has recently introduced the *$lookup* operator in version 3.2. This operator performs a left join between two collections using a *localField* and a *foreignField* must be specified by the user.

- **Database design**: identifying the interrelationships in legacy databases certainly requires the discovery of join keys. Even though the lack of a rigid schema characterizes NoSQL frameworks, developers would also need to overview the data and take appropriate steps and decisions during the application design process. These decisions can have a significant effect on application

efficiency and readability of the code [Mior and Salem, 2018].

Future directions

Some headings are still worth exploring in our work:

- Regarding our contribution in chapter 4, we intend to study the impact of the data incremental refresh [Qu and Deßloch, 2017] and how to schedule the process of identifiers and references discovery in document stores accordingly. In our approach, the discovery of identifiers and references is processed when the user is offline. If new changes are produced within one or more collections that have already been processed, the process is relaunched again in order to identify the new identifiers and references, which can remain the same. As the processing is done offline, the laboriousness of this process does not impact the user; nevertheless, we consider online processing one of our future goals.
  Secondly, we intend to integrate our approach as an ETL component into a commercial ETL tool, e.g., Talend Big Data and Talend Data Integration.

- In the long run, we plan to enrich our global approach, described in chapter 3, by incorporating a multidimensional modeling phase based on several document stores. For several decades, OLAP has been widely utilized as a data analysis approach to help decision-making on structured data. With the widespread use of NoSQL databases containing semi-structured data, there is an increasing need for supporting OLAP on document stores as well, to help non-expert users gain insights and make better decisions. Current works focus on extracting a multidimensional schema from a single collection [Chouder et al., 2019, Gallinucci et al., 2019], whereas in BI&A fetching relevant data, often needs to access more than one collection of JSON documents.
  Secondly, we intend to extend our approach to support additional NoSQL data models since our ultimate aim is to provide a NoSQL-dedicated BI&A approach.

Bibliography

Abdelhédi, F., Brahim, A. A., Atigui, F., and Zurfluh, G. Mda-based approach for nosql databases modelling. In *Big Data Analytics and Knowledge Discovery - 19th International Conference, DaWaK 2017, Lyon, France, August 28-31, 2017, Proceedings*, pages 88–102, 2017.

Abdelhédi, F., Brahim, A. A., Rajhi, H., Ferhat, R. T., and Zurfluh, G. Automatic extraction of a document-oriented nosql schema. In *Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, Online Streaming, April 26-28, 2021, Volume 1*, pages 192–199. SCITEPRESS, 2021.

Ali, S. M. F. Next-generation ETL framework to address the challenges posed by big data. In *Proceedings of the 20th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with 10th EDBT/ICDT Joint Conference (EDBT/ICDT 2018), Vienna, Austria, March 26-29, 2018*, volume 2062 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

Angadi, A., Angadi, A., Gull, K., and Lee, D. Growth of new databases analysis of nosql datastores. 06 2013.

Asanka, P. D. ETL framework design for NoSQL Databases in Dataware housing. *IJRCAR*, 3:67–75, 2015.

Atigui, F., Ravat, F., Teste, O., and Zurfluh, G. Using OCL for automatically producing multidimensional models and ETL processes. In *Data Warehousing and Knowledge Discovery - 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings*, volume 7448 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2012.

Aumueller, D., Do, H. H., Massmann, S., and Rahm, E. Schema and ontology matching with

COMA++. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 906–908, 2005.

Azmy, M., Shi, P., Lin, J., and Ilyas, I. F. Matching entities across different knowledge graphs with graph embeddings. *CoRR*, abs/1903.06607, 2019.

Baazizi, M. A., Colazzo, D., Ghelli, G., and Sartiani, C. Parametric schema inference for massive JSON datasets. *VLDB J.*, 28(4):497–521, 2019.

Baldacci, L., Golfarelli, M., Graziani, S., and Rizzi, S. QETL: an approach to on-demand ETL from non-owned data sources. *Data Knowl. Eng.*, 112:17–37, 2017.

Balti, H., Mellouli, N., Ben Abbes, A., Farah, I. R., Sang, Y., and Lamolle, M. Enhancing big data warehousing and analytics for spatio-temporal massive data. In *Information Systems Development: Crossing Boundaries between Development and Operations (DevOps) in Information Systems (ISD2021 Proceedings), Valencia, Spain, September 8-10, 2021*. Universitat Politècnica de València / Association for Information Systems, 2021.

Batini, C., Cappiello, C., Francalanci, C., and Maurino, A. Methodologies for data quality assessment and improvement. *The journal of ACM computing surveys CSUR*, 41(3):16, 2009.

Bellahsene, Z., Bonifati, A., Duchateau, F., and Velegrakis, Y. On evaluating schema matching and mapping. In *Schema Matching and Mapping*, pages 253–291. 2011.

Berti-Équille, L., Harmouch, H., Naumann, F., Novelli, N., and Thirumuruganathan, S. Discovery of genuine functional dependencies from relational data with missing values. In *Actes du XXXVIIème Congrès INFORSID, Paris, France, June 11-14, 2019*, pages 287–288, 2019.

Bizer, C., Boncz, P. A., Brodie, M. L., and Erling, O. The meaningful use of big data: four perspectives - four challenges. *Journal of Special Interest Group on Management of Data SIGMOD*, 40(4):56–60, 2011.

Blaselbauer, V. M. and Josko, J. M. B. Jsonglue: A hybrid matcher for json schema matching. In *Proceedings of the Brazilian Symposium on Databases*, 2020.

Bodziony, M., Morawski, R., and Wrembel, R. Evaluating push-down on nosql data sources: experiments and analysis paper. In *BiDEDE '22: Proceedings of The International Workshop on Big*

Data in Emergent Distributed Environments, in conjunction with the 2022 ACM SIGMOD/PODS Conference in Philadelphia, PA, USA, June 12, 2022, pages 4:1–4:6. ACM, 2022.

Bogatu, A., Fernandes, A. A. A., Paton, N. W., and Konstantinou, N. Dataset discovery in data lakes. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 709–720, 2020.

Boufares, F. and Salem, A. B. Heterogeneous data-integration and data quality: Overview of conflicts. In *Proceedings of the 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications SETIT*, pages 867–874, Sousse, Tunisia, 2012.

Cai, H., Zheng, V. W., and Chang, K. C. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Trans. Knowl. Data Eng.*, 30(9):1616–1637, 2018.

Celesti, A., Fazio, M., and Villari, M. A study on join operations in mongodb preserving collections data models for future internet applications. *Future Internet*, 11(4):83, 2019.

Chen, H., Chiang, R. H. L., and Storey, V. C. Business intelligence and analytics: From big data to big impact. *Journal of Management Information Systems MIS*, 36(4):1165–1188, 2012.

Chen, Z., Narasayya, V., and Chaudhuri, S. Fast foreign-key detection in microsoft sql server powerpivot for excel. *Proc. VLDB Endow.*, 7(13):1417–1428, aug 2014. ISSN 2150-8097.

Chouder, M. L., Rizzi, S., and Chalal, R. Exodus: Exploratory OLAP over document stores. *Inf. Syst.*, 79:44–57, 2019.

Cui, P., Wang, X., Pei, J., and Zhu, W. A survey on network embedding. *IEEE Trans. Knowl. Data Eng.*, 31(5):833–852, 2019.

David, J., Euzenat, J., Scharffe, F., and Trojahn dos Santos, C. The alignment api 4.0. *Semantic web*, 2(1):3–10, 2011.

Debbarma, N., Nath, G., and Das, H. Analysis of data quality and performance issues in data warehousing and business intelligence. *The International Journal of Computer Applications IJCA*, 79(15), 2013.

Demchenko, Y., Ngo, C., and Membrey, P. Architecture framework and components for the big data ecosystem. 2013.

Dhaouadi, A., Bousselmi, K., Gammoudi, M. M., Monnet, S., and Hammoudi, S. Data warehousing process modeling from classical approaches to new trends: Main features and comparisons. *Data*, 7 (8):113, 2022.

DiScala, M. and Abadi, D. J. Automatic generation of normalized relational schemas from nested key-value data. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 295–310, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450335317.

Djeddi, W. E., Ben Yahia, S., and Khadir, M. T. Xmap: results for OAEI 2018. In *Proceedings of the 13th International Workshop on Ontology Matching co-located with the 17th International Semantic Web Conference, OM@ISWC 2018, Monterey, CA, USA, October 8, 2018.*, pages 210–215, 2018.

Dumin Sahiet, P. D. A. ETL framework design for NoSQL Databases in Dataware housing. *IJRCAR*, 3:67–75, 2015.

Faria, D., Pesquita, C., Balasubramani, B. S., Tervo, T., Carriço, D., Garrilha, R., Couto, F. M., and Cruz, I. F. Results of AML participation in OAEI 2018. In *Proceedings of the 13th International Workshop on Ontology Matching co-located with the 17th International Semantic Web Conference, OM@ISWC 2018, Monterey, CA, USA, 2018*, pages 125–131, 2018.

Fernandez, R. C., Mansour, E., Qahtan, A. A., Elmagarmid, A. K., Ilyas, I. F., Madden, S., Ouzzani, M., Stonebraker, M., and Tang, N. Seeping semantics: Linking datasets using word embeddings for data discovery. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 989–1000, 2018.

Francia, M., Gallinucci, E., Golfarelli, M., and Rizzi, S. OLAP querying of document stores in the presence of schema variety. In *Proceedings of the 28th Italian Symposium on Advanced Database Systems, Villasimius, Sud Sardegna, Italy (virtual due to Covid-19 pandemic), June 21-24, 2020*, pages 128–135, 2020.

Gallinucci, E., Golfarelli, M., and Rizzi, S. Schema profiling of document-oriented databases. *Inf. Syst.*, 75:13–25, 2018.

Gallinucci, E., Golfarelli, M., and Rizzi, S. Approximate OLAP of document-oriented databases: A variety-aware approach. *Inf. Syst.*, 85:114–130, 2019.

Gantz, J. and Reinsel, D. Extracting value from chaos. In *IDC's Digital Universe Study, sponsored by EMC, 2011*, 2011.

Gartner. Mongodb debuts in gartner's magic quadrant for data warehouse, 2017. URL https://www.mongodb.com/blog/post/mongodb-debuts-in-gartner-s-magic-quadrant-for-data-warehouse-and-data-management-solutions-for-analytics.

Gill, R. and Singh, J. "An Open Source ETL Tool - Medium and Small Scale Enterprise ETL MaSSEETL". *International Journal of Computer Applications IJCA*, 108(4):15–22, 2014.

Golfarelli, M. and Rizzi, S. From star schemas to big data: 20+ years of data warehouse research. In *A Comprehensive Guide Through the Italian Database Research*, volume 31 of *Studies in Big Data*, pages 93–107. Springer International Publishing, 2018.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864, 2016.

Hamadou, H. B., Gallinucci, E., and Golfarelli, M. Answering GPSJ queries in a polystore: A dataspace-based approach. In *Conceptual Modeling - Proceedings of the 38th International Conference, ER, Salvador, Brazil*, pages 189–203, 2019.

He, Y., Ganjam, K., and Chu, X. SEMA-JOIN: joining semantically-related tables using big table corpora. *PVLDB*, 8(12):1358–1369, 2015.

Inmon, W. H. *Building the Data Warehouse*. QED Information Sciences, Inc., Wellesley, MA, USA, 1992. ISBN 0-89435-404-3.

Izquierdo, J. L. C. and Cabot, J. Jsondiscoverer: Visualizing the schema lurking behind JSON documents. *Knowl.-Based Syst.*, 103:52–55, 2016.

Jiang, L. and Naumann, F. Holistic primary key and foreign key detection. *J. Intell. Inf. Syst.*, 54(3): 439–461, 2020.

Kachroudi, M., Diallo, G., and Ben Yahia, S. KEPLER at OAEI 2018. In *Proceedings of the 13th International Workshop on Ontology Matching co-located with the 17th International Semantic Web Conference, OM@ISWC 2018, Monterey, CA, USA*, pages 173–178, 2018.

Kathiravelu, P., Sharma, A., Galhardas, H., Roy, P. V., and Veiga, L. On-demand big data integration: A hybrid ETL approach for reproducible scientific research. *CoRR*, abs/1804.08985, 2018.

Kim, M., Zimmermann, T., DeLine, R., and Begel, A. Data scientists in software teams: state of the art and challenges. In *Proceedings of the 40th International Conference on Software Engineering, ICSE*, page 585, Gothenburg, Sweden, 2018.

Kimball, R. and Ross, M. *The data warehouse toolkit: the complete guide to dimensional modeling, 2nd Edition*. Wiley, 2002. ISBN 9780471200246. URL https://www.worldcat.org/oclc/49284159.

Klettke, M., Störl, U., and Scherzinger, S. Schema extraction and structural outlier detection for json-based NoSQL data stores. In *Datenbanksysteme für Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Hamburg, Germany. Proceedings*, pages 425–444, 2015.

Knoblock, C. A. and Szekely, P. A. Exploiting semantics for big data integration. *AI Mag.*, 36(1): 25–38, 2015.

Kondylakis, H., Fountouris, A., Planas, A., Troullinou, G., and Plexousakis, D. Enabling joins over cassandra NoSQL databases. In *Big Data Innovations and Applications - 5th International Conference, Innovate-Data 2019, Istanbul, Turkey, Proceedings*, pages 3–17, 2019.

Koutras, C., Fragkoulis, M., Katsifodimos, A., and Lofi, C. REMA: graph embeddings-based relational schema matching. In *Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference, Copenhagen, Denmark, March 30, 2020*, 2020.

Koutras, C., Siachamis, G., Ionescu, A., Psarakis, K., Brons, J., Fragkoulis, M., Lofi, C., Bonifati, A., and Katsifodimos, A. Valentine: Evaluating matching techniques for dataset discovery. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 468–479, 2021.

Kuznetsov, S. D. and Poskonin, A. V. Nosql data management systems. *Programming and Computer Software*, 40(6):323–332, 2014.

Laney, D. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

LaValle, S., Lesser, E., Shockley, R., Hopkins, M. S., and Kruschwitz, N. Big data, analytics and the path from insights to value. *MIT sloan management review*, 52(2):21–32, 2011.

Mali, J., Atigui, F., Azough, A., and Travers, N. ModelDrivenGuide: An Approach for Implementing NoSQL Schemas. In *Database and Expert Systems Applications - 31st International Conference, DEXA, Bratislava, Slovakia, Proceedings,*, pages 141–151, 2020.

Mallek, H., Ghozzi, F., Teste, O., and Gargouri, F. BigDimETL with NoSQL database. In *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference KES, Belgrade, Serbia*, pages 798–807, 2018.

Mallek, H., Ghozzi, F., and Gargouri, F. Towards extract-transform-load operations in a big data context. *Int. J. Sociotechnology Knowl. Dev.*, 12(2):77–95, 2020.

Mami, M. N., Graux, D., Scerri, S., Jabeen, H., Auer, S., and Lehmann, J. Squerall: Virtual ontology-based access to heterogeneous and large data sources. In *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, Proceedings, Part II*, pages 229–245, 2019.

McAfee, A. and Brynjolfsson, E. Big data: The management revolution. *Harvard business review*, 90: 60–6, 68, 128, 10 2012.

Mehmood, E. and Anees, T. Distributed real-time etl architecture for unstructured big data. *Knowledge and Information Systems*, pages 1–27, 2022.

Memari, M., Link, S., and Dobbie, G. SQL data profiling of foreign keys. In *Conceptual Modeling - 34th International Conference, ER 2015, Stockholm, Sweden, Proceedings*, pages 229–243, 2015.

Mior, M. J. and Salem, K. Renormalization of nosql database schemas. In *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, pages 479–487, 2018.

Nguyen, T. D., Shih, M., Parvathaneni, S. S., Xu, B., Srivastava, D., and Tirthapura, S. Random sampling for group-by queries. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 541–552, 2020.

Nkisi-Orji, I., Wiratunga, N., Massie, S., Hui, K., and Heaven, R. Ontology alignment based on word embedding and random forest classification. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part I*, pages 557–572, 2018.

Panimalar, A., Shree, V., and Kathrine, V. The 17 v's of big data. *International Research Journal of Engineering and Technology*, 4:329–333, 09 2017.

Papenbrock, T. and Naumann, F. Data-driven schema normalization. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, 2017*, pages 342–353, 2017.

Pedersen, T., Patwardhan, S., and Michelizzi, J. Wordnet: : Similarity - measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004, Boston, Massachusetts, USA, May 2-7, 2004*, 2004.

Pejcoch, D. Critical evaluation of validation rules automated extraction from data. *Journal of Systems Integration*, 5:32–46, 2014.

Petricioli, L., Humski, L., and Vrdoljak, B. The challenges of nosql data warehousing. In *International conference on E-business technologies (EBT)*, 2021.

Pokorný, J. JSON functionally. In *Advances in Databases and Information Systems - 24th European Conference, ADBIS 2020, Lyon, France, August 25-27, 2020, Proceedings*, pages 139–153, 2020.

Qu, W. and Deßloch, S. Incremental ETL pipeline scheduling for near real-time data warehouses. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, pages 299–308, 2017.

Rahm, E. and Do, H. H. Data cleaning: Problems and current approaches. *Journal of IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

Ram, J., Zhang, C., and Koronios, A. The implications of big data analytics on business intelligence: A qualitative study in china. *Procedia Computer Science*, 87:221–226, 2016.

Redman, T. C. The impact of poor data quality on the typical enterprise. *Communications of the ACM Journal*, 41(2):79–82, 1998.

Rizzi, S. Business intelligence. In *Encyclopedia of Database Systems*, pages 287–288. 2009.

Rizzi, S. Olap and nosql: Happily ever after. In *Advances in Databases and Information Systems*, pages 35–44, Cham, 2022. Springer International Publishing.

Shvaiko, P. and Euzenat, J. A survey of schema-based matching approaches. *J. Data Semant.*, pages 146–171, 2005.

Shvaiko, P. and Euzenat, J. Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176, 2013.

Souibgui, M., Atigui, F., Zammali, S., Si-Said Cherfi, S., and Ben Yahia, S. Data quality in ETL process: A preliminary study. In *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES-2019, Budapest, Hungary*, pages 676–687, 2019.

Souibgui, M., Atigui, F., Ben Yahia, S., and Si-Said Cherfi, S. Business intelligence and analytics: On-demand ETL over document stores. In *Research Challenges in Information Science - 14th International Conference, RCIS 2020, Limassol, Cyprus, September 23-25, 2020, Proceedings*, pages 556–561, 2020.

Souibgui, M., Atigui, F., Ben Yahia, S., and Si-Said Cherfi, S. IRIS-DS: A new approach for identifiers and references discovery in document stores. In *54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021*, pages 1–10, 2021.

Souibgui, M., Atigui, F., Ben Yahia, S., and Si-Said Cherfi, S. An embedding driven approach to automatically detect identifiers and references in document stores. *Data Knowledge Engineering*, 139:102003, 2022. ISSN 0169-023X.

Thenmozhi, M. and Vivekanandan, K. A framework to derive multidimensional schema for data warehouse using ontology. In *Proceedings of National Conference on Internet and WebSevice Computing, NCIWSC*, 2012.

Theodorou, V., Abelló, A., Lehner, W., and Thiele, M. Quality measures for ETL processes: from goals to implementation. *Concurr. Comput. Pract. Exp.*, 28(15):3969–3993, 2016. doi:10.1002/cpe.3729. URL https://doi.org/10.1002/cpe.3729.

Theodorou, V., Jovanovic, P., Abelló, A., and Nakuçi, E. Data generator for evaluating ETL process quality. *Journal of Information Systems JIS*, 63:80–100, 2017.

Thota, S. *Big Data Quality*, pages 1–5. Springer International Publishing, Cham, 2017.

TruicÄ, C.-O., Apostol, E.-S., Darmont, J., and Pedersen, T. B. The forgotten document-oriented database management systems: An overview and benchmark of native xml dodbmses in comparison with json dodbmses. *Big Data Research*, 25:100205, 2021. ISSN 2214-5796.

van Gennip, Y., Hunter, B., Ma, A., and Moyer, D. Unsupervised record matching with noisy and incomplete data. *International Journal of Data Science and Analytics IJDSA*, 6(2):109–129, Sep 2018.

Vassiliadis, P. A survey of extract-transform-load technology. *International Journal of Data Warehousing and Mining IJDWM*, 5(3):1–27, 2009. doi:10.4018/jdwm.2009070101.

Waghray, K. JSON schema matching: Empirical observations. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 2887–2889, 2020.

Wang, J., Li, G., and Feng, J. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proceedings of the 27th International Conference on Data Engineering, ICDE, Hannover, Germany*, pages 458–469, 2011.

Wang, L., Hassanzadeh, O., Zhang, S., Shi, J., Jiao, L., Zou, J., and Wang, C. Schema management for document stores. *PVLDB*, 8(9):922–933, 2015.

Warth, J., Kaiser, G., and Kügler, M. The impact of data quality and analytical capabilities on planning performance: insights from the automotive industry. In *Proceedings of the Wirtschaftsinformatik*, page 87, Zurich, 2011.

Watson, H. J. Business intelligence: Past, present and future. In *Proceedings of the 15th Americas Conference on Information Systems AMCIS*, page 153, San Francisco, California, USA, 2009.

Wu, X., Wang, N., and Liu, H. Discovering foreign keys on web tables with the crowd. *Comput. Informatics*, 38(3):621–646, 2019.

Wu, Z. and Palmer, M. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, ACL '94, page 133–138, USA, 1994. Association for Computational Linguistics.

Yang, Q., Ge, M., and Helfert, M. Guidelines of Data Quality Issues for Data Integration in the Context of the TPC-DI Benchmark. In *Proceedings of the 19th International Conference on Enterprise Information Systems*, volume 1, pages 135–144, Portugal, 2017.

Yang, Y., Meneghetti, N., Fehling, R., Liu, Z. H., and Kennedy, O. Lenses: An on-demand approach to ETL. *PVLDB*, 8(12):1578–1589, 2015.

Yangui, R., Nabli, A., and Gargouri, F. ETL based framework for nosql warehousing. In *Information Systems - 14th European, Mediterranean, and Middle Eastern Conference, EMCIS, Coimbra, Portugal, Proceedings*, pages 40–53, 2017.

Zhang, J., Shin, B., Choi, J. D., and Ho, J. C. SMAT: an attention-based deep learning solution to the automation of schema matching. In *Advances in Databases and Information Systems - 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24-26, 2021, Proceedings*, pages 260–274, 2021.

Zhang, M., Hadjieleftheriou, M., Ooi, B. C., Procopiuc, C. M., and Srivastava, D. On multi-column foreign key discovery. *Proc. VLDB Endow.*, 3(1):805–814, 2010.

Zhao, Z., Christensen, R., Li, F., Hu, X., and Yi, K. Random sampling over joins revisited. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1525–1539, 2018.

Zhu, E., Deng, D., Nargesian, F., and Miller, R. J. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 847–864, 2019.

le cnam

Manel SOUIBGUI

Business Intelligence: Towards On-demand ETL over Document Stores

HESAM UNIVERSITÉ

Résumé : L'exploitation des bases de données orientées documents, qui sont sans schéma, et souvent sans contraintes d'intégrité, pour la prise de décision nécessite de revoir toutes les phases de l'architecture Business Intelligence, particulièrement le processus ETL afin de pouvoir gérer le volume, la variété et la vélocité de ces données. L'étude de la littérature nous a conduits à introduire une nouvelle approche permettant d'extraire, transformer et de charger à la demande les données requises pour l'analyse OLAP en considérant la dispersion des données sur plusieurs sources orientées documents. Afin d'extraire plusieurs sources, nous proposons une approche qui vise à détecter automatiquement les identifiants et les références composés et non composés à partir de plusieurs sources orientées documents. Notre approche est basée sur des caractéristiques et des règles d'élagage qui permettent de détecter les identifiants candidats et les paires candidates (identifiant, référence) entre chaque deux collections. Pour illustrer notre étude, nous présentons les résultats expérimentaux qui montrent la pertinence de notre approche et nous discutons les défis à relever dans nos travaux futurs.

Mots clés : Business Intelligence & Analytics, ETL, Base de données orientées documents, Découverte d'identifiants, Découverte des références

Abstract: Exploiting schema-free data for decision-making is challenging since it requires reviewing all the Business Intelligence phases, particularly the ETL process, to fit big data sources as document stores. After conducting a literature review regarding these issues, we introduce a new Business Intelligence approach that extracts, transforms, and loads the required data for OLAP analysis (on-demand ETL) from document stores. Secondly, as our goal is to extract disparate data, we study the problem of discovering join fields automatically. We introduce a new approach that aims to automatically detect identifiers and references on several document stores. The approach is based on scoring features and pruning rules to discover true candidate identifiers and candidate pairs (identifier, reference) between several document stores. To illustrate our study, we report our experimental findings that show encouraging results and we identify and discuss the challenges to be addressed in our future research.

Keywords: Business Intelligence & Analytics, ETL, Document stores, Identifiers discovery, References discovery