N°d'ordre: .....

République Algerienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la Recherche scientifique

Université de Sidi Bel Abbès
Faculté des Sciences Exactes
Département d'informatique
Laboratoire EEDIS

Thèse de Doctorat

Domaine : Mathématiques - Informatique
Filière : Informatique
Spécialité : Informatique

Par

Mme Nesrine LEHIRECHE

INTEGRATION DES SYSTEMES D'INFORMATION DIRIGÉE PAR LES LINKED DATA

Soutenue le 31-01-2021 devant le jury :

Dr.	TOUMOUH Adil	Université Djillali Liabès	Président du jury
Pr.	BENSLIMANE Sidi Mohamed	ESI-SBA	Examinateur
Pr.	BOUKLI HACENE Sofiane	Université Djillali Liabès	Examinateur
Pr.	ADJOUDJ Réda	Université Djillali Liabès	Examinateur
Pr.	MALKI Mimoun	ESI-SBA	Directeur de thèse

Année Universitaire : 2020 - 2021

Remerciement

Il m'est particulièrement agréable de pouvoir remercier ici le Professeur MALKI Mimoun, Directeur de cette thèse, pour avoir bien voulu m'accueillir dans son équipe de recherche et m'avoir encadré durant ces années. Je lui exprime ma profonde gratitude. Ses encouragements me furent d'un constant appui sans lequel la persévérance m'aurait manqué.

Mes respects et ma gratitude vont également aux membres du jury qui m'ont fait l'honneur de juger ce travail. Je remercie tout d'abord Dr.TOUMOUH Adil. , pour avoir présidé le jury de cette thèse. Je remercie aussi les examinateurs Pr. BENSLIMANE Sidi Mohamed, Pr.BOUKLI-HACENE Sofiane. et Pr. ADJOUDJ Réda qui ont bien voulu accepter de juger mon travail.

Je remercie particulièrement Professeur LEHIRECHE Ahmed, mon père, pour toutes nos discussions et ses suggestions qui m'ont accompagné tout au long de mon cursus universitaire. Et pour les nombreux encouragements qu'il m'a prodiguée.

Je tiens également à remercier Pr. Sofiane BOUKLI HACENE, Chef du laboratoire EEDIS (Evolutionary Engineering and Distributed Information Systems) pour sa disponibilité permanente, son soutien et tout le temps qu'il a consacré pour m'aider à l'accomplissement de cette thèse.

J'adresse mes sincères remerciements à Mr HAMOU Reda Mohamed pour l'aide précieuse qu'il m'a apportée notamment dans la publication de mes travaux de recherches.

Résumé

L'intégration de SPARQL et RDF avec les SGBDRs est cruciale pour l'adoption des technologies du Web sémantique en entreprise. Cette importance se reflète dans la création du standard R2RML pour faire le mapping RDB-to-RDF, et dans toutes les recherches axées sur la réécriture des requêtes SPARQL-to-SQL au moyen de mappings. Les approches proposées sur l'exécution de requêtes SPARQL sur SGBDRs souffraient souvent de problèmes qui limitaient leur utilisation dans la pratique. Certaines techniques supposent un schéma relationnel fixe et ne prennent pas en charge les langages de mapping ; d'autres génèrent des requêtes SQL complexes qui ne fonctionnent pas bien. Ontology-Based Data Access (OBDA) est connu comme une approche populaire pour résoudre ces problèmes et permet d'interroger les sources de données relationnelles via une ontologie, cette dernière est connectée aux sources de données via une spécification déclarative donnée en termes de mappings qui relient les symboles de l'ontologie (classes et propriétés) aux vues (SQL) sur les données. L'ontologie avec les mappings expose un graphe RDF virtuel, qui peut être interrogé à l'aide de SPARQL. Ce graphe RDF peut être matérialisé, générant des triplets RDF à utiliser dans les triplestores RDF, ou bien il peut être conservé virtuel et interrogé uniquement pendant l'exécution de la requête. Nous sommes intéressés par les requêtes SPARQL contenant les patterns de type optionnels imbriqués introduits pour traiter les informations manquantes. Nous abordons dans cette thèse comment les requêtes SPARQL avec l'optionnels imbriqués peuvent être efficacement évaluées dans un SGBDR, en proposant une solution basée sur l'optimisation de l'algèbre SPARQL. Nous présentons également une évaluation comparative de notre proposition au système Ontop, qui est considéré comme le plus efficace dans la réécriture.

Mots Clés :Web Sémantique, Ontologie, BDDR,SQL,Linked Data, SPARQL, Mapping.

Abstract

The integration of SPARQL and RDF with RDBMs is crucial for the adoption of Semantic web Technologies in enterprise. This importance is reflected in the creation of the R2RML standard for mapping RDB-to-RDF, and in all the research focused towards translating SPARQL queries into efficient SQL over RDBMs by means of mappings. Previous techniques on execution of SPARQL queries over RDBMS often suffered from problems that limited their use in practice. Some techniques assume a fixed relational schema and do not support general mapping languages; others generate complex SQL queries that do not perform well. Ontology-Based Data Access (OBDA) is known as a popular approach for tackling these issues, and allows for querying relational data sources through an ontology, the later is connected to the data sources through a declarative specification given in terms of mappings that relate symbols in the ontology (classes and properties) to (SQL) views over the data. The ontology with the mappings expose a virtual RDF graph, which can be queried using SPARQL. This RDF graph can be materialized, generating RDF triples to be used with RDF triplestores, or alternatively it can be kept virtual and queried only during query execution. We are interested in SPARQL queries containing the nested optional patterns introduced to deal with missing information. We tackle in this thesis how SPARQL queries with the nested optional patterns can be efficiently evaluated in an RDBMS, proposing a solution based on the optimization of SPARQL Algebra. We also present a comparative evaluation of our proposals to the Ontop system, which is considered the most efficient in rewriting.
Keywords :Semantic Web,Ontology,RDB,SQL, Linked Data, SPARQL,Mapping.

الملخص

يعد دمج *SPARQL* و *RDF* مع *RDBMs* أمرًا بالغ الأهمية لاعتماد تقنيات الويب الدلالي في المؤسسات. تنعكس هذه الأهمية في إنشاء معيار *R2RML* للقيام بتحويل *RDB* إلى *RDF* ، وفي جميع الأبحاث التي تركز على ترجمة *SPARQL* إلى *SQL* باستخدام التعيينات (*mapping*) .

غالبًا ما عانت الأساليب المقترحة لتشغيل *SPARQL* على أنظمة *SGBDRs* من مشاكل استخدامها من الناحية العملية. تفترض بعض التقنيات مخططًا علاقيًا ثابتًا ولا تدعم لغات التطابق ؛ ينشئ البعض الآخر استعلامات *SQL* معقدة لا تعمل بشكل جيد. يُعرف (*OBDA*) بأنه نهج شائع لحل هذه المشكلات ويسمح بالاستعلام عن مصادر المعلومات العلائقية من خلال الانطلوجيا الذي يحدد المفردات المشتركة ونماذج المجال. ترتبط الأنطولوجيا بمصادر البيانات عبر مواصفات تعريفيّة تُعطى من حيث التعيينات التي تربط رموز الأنطولوجيا (الفئات والخصائص) بالعروض (*SQL*) على البيانات. تم إنشاء معيار *R2RML* لتحديد التطابق في منهجية *OBDA* . يعرض الانطلوجيا باستخدام التعيينات رسمًا بيانيًا افتراضيًا لـ *RDF* ، والذي يمكن الاستعلام عنه باستخدام *SPARQL* . يمكن تحقيق مخطط بياني *RDF* الافتراضي هذا ، وإنشاء عناصر *RDF* للاستخدام في مخزن *RDF* ، أو يمكن الاحتفاظ به افتراضيًا والاستعلام عنه فقط أثناء تنفيذ الاستعلام.

نحن مهتمون باستعلامات *SPARQL* التي تحتوي على أنماط الأنواع المتداخلة الاختيارية *Optional* المقدمة للتعامل مع المعلومات الناقصة. نناقش في هذه الأطروحة كيف يمكن تقييم استعلامات *SPARQL* ذات الخيارات المتداخلة بكفاءة في نظام *RDBMS* ، من خلال اقتراح حل يعتمد على تحسين الكتابة الجبرية ل *SPARQL* . نقدم أيضًا تقييمًا مقارنًا لاقتراحنا لنظام *SPARQL* . نقدم أيضًا تقييمًا مقارنًا لاقتراحنا لنظام *Ontop* ، والذي يعتبر الأكثر كفاءة في إعادة الكتابة .

Table des matières

Liste des figures

Liste des tableaux

Introduction Générale 1

1.1 Contexte

Au cours de la dernière décennie et plus, le Web sémantique (SW) est passé d'une vision futuriste abstraite, à une réalité toujours plus proche d'un web mondial de données liées et interconnectées avec le sens bien défini. Des langages et technologies standard ont été proposés et évoluent constamment afin de servir de briques de base pour ce Web de «nouvelle génération», des outils pertinents sont en cours de développement et atteignent progressivement leur maturité, tandis que de nombreuses applications du monde réel donnent déjà un avant-goût des avantages. Le Web sémantique est sur le point d'apporter divers domaines, aussi divers que les sciences de la vie, la surveillance de l'environnement, le patrimoine culturel, le gouvernement électronique et la gestion des processus commerciaux. Ces progrès évidents sont le résultat de recherches menées depuis deux décennies et il n'est pas surprenant que, de nos jours, le Web sémantique soit perçu comme un domaine de recherche multidisciplinaire, combinant et acquérant une expertise dans d'autres domaines scientifiques, tels que l'intelligence artificielle, les sciences de l'information, théorie des algorithmes et de la complexité, théorie des bases de données et réseaux informatiques, pour n'en nommer que quelques-uns.(Sequeda 2016)

La participation des bases de données et leur rôle dans cet environnement Web en évolution a été étudiée dès le début de la conception du Web sémantique, non seulement parce qu'elle était initialement comparée à «une base de données globale» (Berners-Lee *et al.* 1998), mais aussi parce que cela - nouveau au temps - le domaine de la recherche pourrait tirer parti de la grande expérience et de la maturité du domaine de la base de données. Cependant, la collaboration et l'échange d'idées entre ces deux domaines n'étaient pas unidirectionnels. La communauté des bases de données a rapidement reconnu les opportunités découlant d'une coopération étroite avec le domaine du Web sémantique (Sheth & Meersman 2002) et comment celui-ci pouvait offrir des solutions à des problèmes de longue date, et fournir inspiration pour plusieurs sous-communautés de bases de données, intéressées par l'intégration et l'interopérabilité de bases de données hétérogènes, les architectures distribuées, les bases de données déductives, la modélisation conceptuelle, etc.

Les tentatives de combiner ces deux mondes différents se concentraient à l'origine sur la réconciliation des écarts entre les deux technologies les plus représentatives et dominantes de chaque monde : les bases de données relationnelles et les ontologies. Ce problème est également connu sous le nom de problème de mapping de la base de données à l'ontologie, qui est subsumé par le problème de non-correspondance d'impédance

objet-relationnel plus large et est dû aux différences structurelles entre le modèle relationnel et le modèle orientés objet. Les correspondances entre le modèle relationnel et le modèle de graphe RDF, qui est un élément clé du Web sémantique, ont également été étudiées et un groupe de travail du W3C a été formé pour examiner cette question et proposer des standards connexes. Néanmoins, la définition théorique du mapping entre les modèles mentionnés n'est pas une fin en soi. La motivation entraînant la prise en compte des mappings entre les bases de données relationnelles et les technologies du Web sémantique est multiple, conduisant à des problèmes distincts, où les mappings sont découverts, définis et utilisés de manière différente pour chaque cas de problème.

À l'origine, les systèmes de bases de données étaient considérés par la communauté du Web sémantique comme un excellent moyen pour un stockage efficace des ontologies (Beckett & Grant 2003), en raison de leurs avantages de performances connus et bien mis en évidence. Cette considération a conduit au développement et à la production de plusieurs systèmes de base de données, spécialement optimisés pour le stockage persistant, la maintenance et l'interrogation des données du web sémantique (Spanos *et al.* 2012). De tels systèmes sont connus sous le nom de triplestore. Ces bases de données peuvent gérer des milliards d'enregistrements et prendre en charge l'inférence.

Les triplestores utilisent des URI [1], ce qui signifie qu'ils prennent en charge l'interrogation et le raisonnement sur le Web sémantique. Contrairement aux bases de données relationnelles qui stockent les données dans des tables, les triplestores stockent les données sous forme sujet-prédicat-objet, telles que «Ahmed enseigne l'informatique» ; chaque déclaration est appelée un triplet. Cette représentation des données utilise le Resource Description Framework (RDF) [2], un modèle standard de publication et de partage de données sur le Web. Le sujet, le prédicat et l'objet peuvent tous être des URI, ce qui permet de lier facilement les données. La collection de déclarations dans un triplestore forme une base de données sous forme graphe des faits. Chaque triplet peut avoir un nom, créant un graphe nommé. Le sujet et l'objet sont les nœuds du graphe et les prédicats présentent les feuilles. Il peut éventuellement y avoir un modèle de schéma, appelé ontologie, qui fournit une description formelle des données.

Les triplestores sont interrogés par un langage appelé SPARQL (SPARQL Protocol and RDF Query Language)) [3]. Comme SQL, les don-

1. https ://www.w3.org/Addressing/URL/uri-spec.html
2. http ://www.emse.fr/ zimmermann/W3C/RDF1.1Primer/
3. http ://www.w3.org/TR/2013/REC-sparql11-query-20130321/

nées peuvent être extraites sous forme de tableau à l'aide d'une requête Select. Les requêtes sont structurées de telle sorte qu'un préfixe définit l'espace de noms et une clause SELECT définit l'ensemble de résultats à renvoyer à partir de l'ensemble de données spécifié où le triplet correspond à un pattern de requête. Des clauses supplémentaires, telles que ORDER BY et DISTINCT, modifient l'ensemble de résultats. Les variables sont utilisées dans les clauses et utilisent la même variable dans plusieurs patterns de la requête pour définir des jointures.

SPARQL prend également en charge une requête CONSTRUCT qui génère les résultats au format RDF, une requête ASK qui répond aux questions par Oui / Non et une requête DESCRIBE qui décrit les ressources qui correspondent à la requête. Étant donné que les triplestores utilisent des URIs pour référencer les données, les requêtes peuvent être exécutées à partir de SPARQL endpoints publiques tels que : Virtuoso SPARQL Query, SPARQL Explorer. Ces endpoints publiques avec l'utilisation d'URIs signifient que toutes les données accessibles peuvent facilement être jointes à d'autres données accessibles.
Les triplestores offrent plusieurs avantages par rapport aux systèmes de gestion de base de données traditionnels. Ils sont connus par leurs flexibilités, ou il n'est pas nécessaire de définir un schéma à l'avance et pas besoin d'entités artificielles telles que des tables pour représenter une relation many-to-many. Les triplestores peuvent être interrogés en utilisant le langage SPARQL. Contrairement aux requêtes SQL, qui deviennent compliquées et inefficaces si la base de données n'a pas été conçue avec des colonnes à joindre et des index pour rendre la recherche efficace, les triplestores peuvent facilement gérer des requêtes complexes. L'utilisation des URI rend le partage de données simple. Il s'agit d'un avantage pour les programmes d'analyse qui doivent rassembler des données provenant de plusieurs sources.

Ce type de collaboration entre la base de données et le Web sémantique spécifie un flux de données et d'informations de ce dernier vers le premier, avec des données SW. Les triplestores sont également largement utilisés pour l'analyse sémantique, comme l'exploration de texte, les algorithmes d'apprentissage automatique qui peuvent lever l'ambiguïté et raisonner sur les entités. Les triplestores sont également utilisés pour établir un lien entre des bases de données structurées et des documents de données non structurées. Leur capacité à prendre en charge l'inférence fait des triplestores une technologie clé derrière les applications de recherche et de découverte.

1.2 Problématiques

L'importance des bases de données du point de vue du Web sémantique est évidente à partir des multiples avantages et cas d'utilisation dans lesquels un mapping de bases de données et d'ontologies est utilisée (Lehireche *et al.* 2017).Il est important d'identifier les différentes motivations et problèmes impliquant les interactions entre les bases de données relationnelles et les technologies logicielles, afin de réussir une séparation claire des objectifs et des défis.

Les SGBDRs utilisés dans l'entreprise, servant par exemple à la fabrication assistée par ordinateur, à la planification des ressources d'entreprise, systèmes de gestion de la chaîne d'approvisionnement et de gestion de contenu. Nous considérons l'intégration des données relationnelles comme une technique cruciale d'intégration des données d'entreprise et des bases de données relationnelles dans le web sémantique.

Pour comprendre la relation entre les bases de données relationnelles et le Web sémantique, nous adoptons une méthodologie où chaque technologie est décomposée en couches correspondantes (Xu *et al.* 2006).Cela nous permet d'identifier et d'exploiter les similitudes. Selon le contexte historique, il est facile de décomposer les bases de données relationnelles en une pile représentant une puissance expressive croissante et peut-être pas par coïncidence, celle qui correspond à la pile du Web sémantique.

La figure 1.1 montre la relation qui en résulte entre les bases de données relationnelles et le Web Sémantique. Les observations suivantes sont faites :

— La couche fondamentale des bases de données relationnelles et du Web sémantique est le modèle de données (modèle relationnel et RDF).

— La couche suivante fournit un langage pour décrire les schémas (définitions de table et RDFS [4]).

— Les couches supérieures augmentent l'expressivité du langage (contraintes, vues, récursivité et OWL [5]).

— Les deux technologies ont un langage de requête (SQL et SPARQL).

— Chaque couche a évolué au fil du temps.

La comparaison entre les deux piles de couches dans la figure 1.1 suggère qu'un cadre bien défini pour combiner les bases de données relationnelles et le web sémantique, et par rapport à la technologie actuelle on aborde la question suivante : Comment et dans quelle mesure

4. https ://www.w3.org/TR/rdf-schema/
5. https ://www.w3.org/OWL/

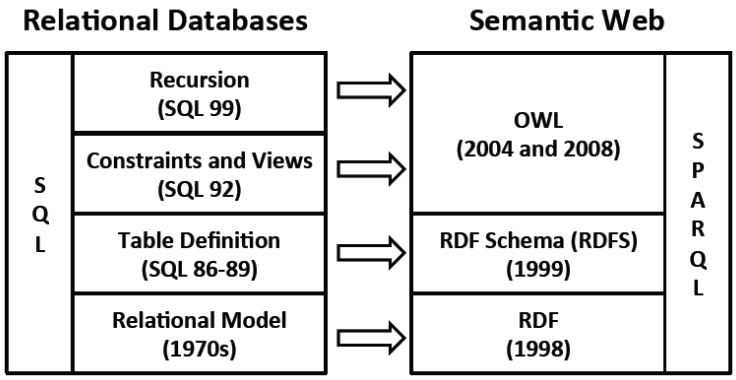

FIGURE 1.1 – *Couches des Bases de données relationnelles et du Web Sémantique (Sequeda 2016)*

les bases de données relationnelles peuvent-elles être intégrées au Web sémantique ? L'infrastructure de base de données relationnelle existante peut être réutilisée pour prendre en charge le Web sémantique.

Deux problèmes sont étudiés : Une base de données relationnelle peut-elle être automatiquement intégrée en tant que source de données Web sémantique ? Ce contexte comprend : une base de données relationnelle, un mapping, et une ontologie OWL (McGuinness *et al.* 2004) . La première solution est une approche de matérialisation utilisée pour dériver de nouveaux triplets RDF qui sont ensuite stockés dans un Triplestore, comme le montre la figure 1.2. Les données RDF dans le Triplestore sont considérées comme la matérialisation des données de la base de données relationnelle à l'aide du mapping et d'une ontologie.

La réponse à une requête SPARQL sur l'ontologie est calculée en posant directement la requête SPARQL sur le Triplestore. D'un schéma de base de données relationnelle et de données vers RDF et OWL. Avec cette approche, il n'y a pas de surcharge lorsqu'une requête SPARQL est évaluée sur le Triplestore par rapport à l'approche de réécriture. Cependant, un inconvénient est qu'une copie des données relationnelles d'origine doit être conservée. Si des mises à jour se produisent sur les données relationnelles d'origine, elles doivent être propagées vers la version RDF. De plus, selon l'ontologie, la taille des données RDF matérialisées peut être infinie.

La deuxième solution est une méthode capable d'évaluer les requêtes SPARQL sur la base de données relationnelle. C'est une approche basée sur la réécriture qui a été au centre des préoccupations des chercheurs au cours des dernières années. Trois étapes sont exécutées, comme le montre la figure 1.3. Étant donné une requête SPARQL et une ontologie OWL, une nouvelle requête SPARQL est générée qui contient les connaissances de

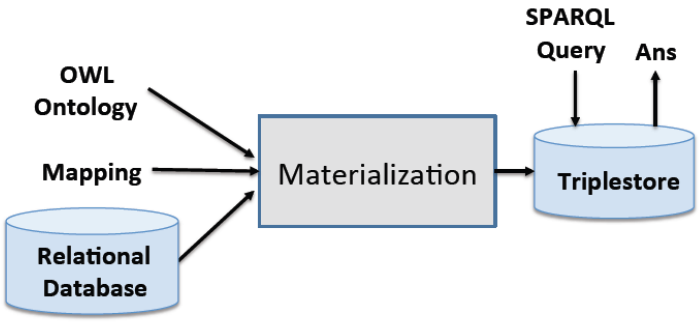

FIGURE 1.2 – *Approche de matérialisation (Sequeda 2016)*

l'ontologie. La nouvelle requête SPARQL est considérée comme la requête réécrite. Dans la deuxième étape, le mapping est utilisé pour compiler la requête SPARQL réécrite en une requête SQL qui peut ensuite être évaluée sur la base de données. Enfin, la requête SQL est évaluée sur la base de données relationnelle, ce qui donne une réponse à la requête SPARQL initiale.

L'avantage de l'approche de réécriture est que le raisonnement se fait sur la base de données relationnelle d'origine. Par conséquent, les résultats des requêtes reflètent les dernières données à jour. Comme décrit précédemment, l'inconvénient est la surcharge engendrée par la traduction de la requête SPARQL.

Ce travail se concentre principalement sur l'intégration des bases de données relationnelles par la réécriture de requête SPARQL en SQL. Étant donné une base donnée relationnelle, une ontologie OWL, des mappings de la base de données relationnelle à l'ontologie OWL. Comment exécuter des requêtes SPARQL sur une base de données relationnelle, conformément au mapping défini ?, Les requêtes SPARQL évaluées sur la représentation Web sémantique sont converties en requêtes SQL et ensuite évaluées sur les bases de données relationnelles.

Dans cette thèse, nous avons soulevé deux principaux défis pour exécuter des requêtes SPARQL sur une base de données relationnelle. Tout d'abord, comment définir un mapping entre la base de données et l'ontologie, afin de répondre directement aux requêtes SPARQL. Ensuite, décrire les optimisations nécessaires sur le modèle relationnel résultant pour que les requêtes SPARQL soient efficacement exécutées sur la base de données relationnelle.

- Le premier défi consiste à identifier un mapping pouvant servir de moyen automatique et par défaut pour traduire un schéma de base de données relationnelle en une ontologie OWL et des instances de bases de

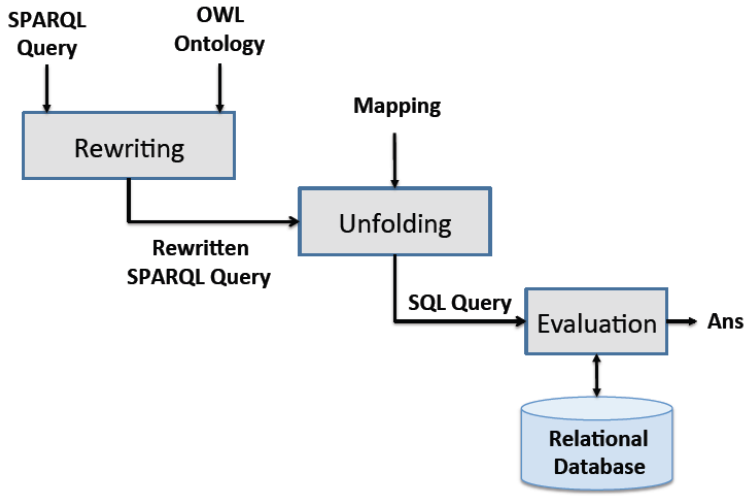

FIGURE 1.3 – *Approche de réécriture de requête (Sequeda 2016)*

données relationnelles en RDF. Plusieurs aspects touchant ce problème de mapping, citant par exemple, comment créer une nouvelle ontologie à partir d'informations extraites d'une base de données relationnelle, comment générer des instructions RDF conformes à une ou plusieurs ontologies prédéfinies et reflétant le contenu d'une base de données relationnelle, découvrir des correspondances entre une base de données relationnelle et une ontologie.

- Le deuxième défi pousse l'optimisation des requêtes dans l'infrastructure SQL existante. En revanche, les requêtes SQL générées peuvent déjà être exécutées par le moteur de base de données, mais elles sont inefficaces : elles contiennent souvent des sous-requêtes, des auto-jointures redondantes et des jointures sur des expressions complexes telles que les concaténations de chaînes de caractères. Les approches existantes se concentrent principalement sur l'utilisation des optimisations déjà incorporées dans les bases de données relationnelles Structurelles et sémantique, qui indique systématiquement que les performances d'exécution des requêtes SPARQL sont comparables aux performances des requêtes SQL écrites directement pour la représentation relationnelle des données. De telles optimisations sont connues pour être relativement coûteuses.

1.3 OBJECTIF

L'objectif du présent travail est de proposer une approche de mapping de bases de données relationnelles vers RDF, ainsi qu'une approche d'optimisation de requête SQL résultante. Notre approche de mapping permet

de découvrir des correspondances entre la base de données relationnelle et le modèle RDF, afin de générer des données RDF conformes à une ou plusieurs ontologies prédéfinies. Pour cela, nous cherchons à définir des règles de mapping qui peuvent être facilement encodées dans un programme sans avoir besoin d'interpréter un nouveau langage de mapping, éviter la création manuelle de document de mapping, et par conséquent réduire le temps d'exécution du processus de mapping. Notre approche d'optimisation de traduction de requête est basée sur le traitement de l'algèbre SPARQL. Il s'agit ici de définir la sémantique de la requête SPARQL.

Dans ce processus de définition de sémantique, l'algèbre SPARQL définit une syntaxe sous formes de graphes patterns, des expressions et des opérateurs dans des clauses (similaires dans certains cas à celles du langage SQL) telles que PREFIX (spécifie l'adresse exploitée dans la construction de la requête), SELECT ... [FROM] ... WHERE (requête interrogative), CONSTRUCT (requête constructive), UNION, OPTIONNAL (jointures, conditions optionnelles), FILTER (conditions obligatoires) et DESCRIBE, ASK (description d'une ressource, évaluation d'une requête). Cette syntaxe concrète fournit un langage alternatif orienté machine pour SPARQL qui Fournit une référence pour tester l'analyse et la simplification de SPARQL, simplifie les constructions de requêtes spécifiques et contrôle les solutions à partir des résultats d'évaluation sur le moteur SPARQL. Donc, notre objectif porte sur l'évaluation de l'algèbre SPARQL et la traduction des expressions et opérateurs complexes, afin d'améliorer le temps d'exécution et le résultat de la traduction de la requête SPARQL sur la base de donnée.

1.4 Principales Contributions

Le travail présenté dans cette thèse ne prétend pas fournir une solution complète répondant à tous les différents aspects du problème de la réécriture de requête SPARQL en SQL. Toutefois, ce travail a apporté une contribution considérable au champ de recherches sur l'évaluation de la requête SPARQL sur la base de données relationnelle. Et c'est à partir des objectifs décrit ci-dessus, que s'inscrivent nos deux principales contributions.

Pour atteindre notre premier objectif défini dans la section précédente. Nous avons proposé une approche formelle basée sur un ensemble d'algorithmes qui génère automatiquement un mapping en combinant des techniques et règles bien connues. Cette formalisation nous permet de prendre en charge des mappings complexes vers des bases de données,

et même à automatiser partiellement la génération de mappings appropriés à partir des relations vers le modèle RDF. Cependant, De notre point de vue, une des principales raisons du manque de déploiement des language de mapping réside dans la complexité a générer manuellement les mappings, qui peut entrainer de nombreuses erreurs dans le document de mapping. Notre formalisation est dynamique, elle nous permet d'intégrer plusieurs sources de données hétérogènes, et fournir suffisamment d'information pour qu'un processus prenne en charge à la fois la réécriture et la matérialisation des données, et enfin, améliorer le temps d'exécution.

La deuxième contribution consiste à effectuer une optimisation sur l'algèbre SPARQL avant que la requête ne soit transformée en une forme relationnelle. Ainsi, nous souhaiterons répondre exclusivement à nos deux principaux objectifs d'optimisation qui sont : 1. générer une seule requête unifiée pour réduire la taille et la complexité de la requête SQL résultante, 2. Améliorer le temps d'exécution. Pour cela, nous avons approfondi la solution proposée par (Xiao *et al.* 2018). Qui effectue les optimisations sur l'algèbre SPARQL d'une requête. Ensuite, la requête est traduite dans le modèle relationnel et d'autres optimisations sont effectuées. Comme nous le montrerons dans la section évaluation, cette approche est plus efficace. Nous étondons l'algorithme de (Calvanese *et al.* 2017)de traduction pour transformer les requêtes SPARQL en requêtes SQL. L'algorithme vise à générer des requêtes optimisées. La principale différence est la façon dont nous traitons l'algèbre SPARQL. Nous utilisons une algèbre modifiée qui contient les informations de mapping. Par conséquent, nous pouvons effectuer certaines optimisations avant qu'une requête ne soit transformée en une forme relationnelle. De plus, nous sommes en mesure de transformer la requête SPARQL sous une forme mieux adaptée à la transformation. Grâce à cela, les optimisations sur le modèle relationnel seront plus efficaces.

1.5 Organisation de la thèse

Ce manuscrit s'organise en six chapitres : en partant de la problématique telle qu'elle est traitée dans le premier chapitre, et en allant vers la description des solutions proposées et développées afin de résoudre les problèmes liées d'une part au mapping RDB-to-RDF, et d'autre part, à l'optimisation de la réécriture. Puis en terminant par une expérimentation et une évaluation des solutions dans le cadre d'une traduction efficace de requête SPARQL.

Le Chapitre 2 décrit un ensemble de différents standards concernés par notre problématique, Il aborde l'utilisation des technologies du Web

sémantique (i.e. l'ontologie OWL, RDF, SPARQL ...) en vue d'intégrer une base de données relationnelle dans le Web sémantique pour faciliter l'accès, la manipulation et la réutilisation des données relationnelles.

Le Chapitre 3 est consacré à l'intégration des bases de données dirigées par les données liées, ou nous décrivons dans un premier temps les challenges de l'intégration et ensuite l'utilisation de l'approche linked data pour intégrer les données sur tous les axes d'un système d'information.

chapitre 4 présente une vue d'ensemble des différents champs de recherche concernés par notre problématique. Nous présentons les différentes approches, outils, et systèmes de réponse aux requêtes SPARQL les plus populaires, qui peuvent être classés en deux types principaux : les triplestores et les systèmes OBDA [6] . Ce chapitre présente aussi une synthèse pour comparer les approches sur la base des éléments de la réécriture (Algorithme de traduction, langage de mapping, types d'optimisations, Algèbre SPARQL, raisonnement) et leurs avantages et lacunes en vue de l'intégration.

cinquième chapitre détaille les deux approches que nous avons proposées afin de trouver un meilleur scénario d'intégration par la réécriture. Nous décrirons, d'abord la solution proposée pour définir un mapping afin de répondre à toutes les questions concernant les types de mapping qui peuvent être exprimés, la génération de document de mapping et les données produites par le mapping. Nous détaillons par la suite notre contribution basée sur le traitement sémantique de l'algèbre SPARQL, l'expression algébrique, les variables et les opérateurs qui affectent le temps d'exécution. Nous y présentons ensuite l'algorithme de traduction de requête SPARQL en SQL en incluant le traitement algébrique de la requête SPARQL, en suite, nous décrivons le processus d'implémentation des solutions. Nous poursuivons par une expérimentation du prototype sur un ensemble de requêtes de test. Finalement, nous présentons les résultats de l'expérience menée en particulier sur l'optimisation de l'algèbre SPARQL de la requête. Nous présentons également une évaluation des résultats obtenues et mettons en évidence les avantages et performances de cette solution par rapport aux approches existantes. Nous terminons cette thèse par une conclusion générale dans le chapitre 6 et les perspectives ouvertes de travaux futurs.

6. https ://www.obdasystems.com/obda

2 Web sémantique et Linked Data

2.1 Introduction

La forte utilisation des données touche tous les secteurs socio-économiques à savoir : des données sur la performance de nos écoles locales, l'efficacité énergétique de nos voitures, une multitude de produits de différents fournisseurs ou la façon dont nos taxes sont dépensées. En nous aidant à prendre de meilleures décisions, ces données jouent un rôle de plus en plus central dans nos vies et stimulent l'émergence d'une économie de données (Heath & Bizer 2011). De plus en plus d'individus et d'organisations contribuent en choisissant de partager leurs données avec d'autres, y compris des sociétés web telles que Amazon et Yahoo !, des journaux tels que Le soir d'Algérie et Le quotidien, des organismes publics. Gouvernements et des initiatives de recherche dans diverses sources comme le Centre de Recherche sur l'Information Scientifique et Technique (CERIST), Centre de Développement des Technologies Avancées (CDTA). Et d'autre consomment ces données pour créer de nouvelles entreprises, accélérer les progrès scientifiques, etc. qui démontre la demande d'accès aux données et les partager, Cela soulève trois questions clés :

— Quelle est la meilleure façon de fournir l'accès aux données afin qu'elles puissent être réutilisées plus facilement.

— Comment permettre la découverte des données pertinentes dans la multitude des ensembles de données disponibles.

— Comment permettre aux applications d'intégrer des données provenant d'un grand nombre de données auparavant inconnues. L'enchainement des données distribuées à travers le web, la réutilisation de ces données et la publication exige un mécanisme standard pour spécifier l'existence et la signification des liens entre les entités. Ce mécanisme est fourni par le langage RDF (Ressource Description Framework). Ce qui a fait que le web a évolué d'un web de document liés à un web de données liées, ce dernier vise à permettre l'accès à ces données, en les rendant disponibles dans des formats lisibles par machine(par ex. RDF, XML . [1] et JSON [2]) et en les connectant à l'aide d'identificateurs de ressources uniformes (URI), permettant ainsi aux utilisateurs et aux machines de collecter ces données,et de les assembler afin de consulter et interpréter les informations présentées sur le web .

Le reste du chapitre est organisé comme suit : Nous allons donner la définition d'une Ontologie dans la section 2, ainsi les différents composants utilisés pour modéliser une ontologie. Dans la section 3 nous introduisons

1. https ://www.w3.org/XML/
2. https ://www.json.org/json-en.html

le Web sémantique. La section 4 nous décrivons la définition des données liées, et présentons aussi les principes de base des données liées. Dans la dernière section, nous présentons brièvement les langages du web sémantique à savoir Le modèle de données RDF, RDFS, le language OWL, et le SPARQL.

2.2 Web Sémantique

Dans les dernières années, le développement impressionnant de l'Internet a renforcé l'apparition d'une énorme quantité de données disponibles sur le Web. La richesse et la croissance exponentielle de ce volume de données promet d'être une mine d'or pour les entreprises. Cependant, cette croissance d'informations donnera lieu aussi à de vrais obstacles si les données ne sont pas bien structurées et bien représentées. Cela montre les challenges du Web du point de vue de la recherche, le partage, l'accès et la réutilisation de ces données, selon les exigences de l'utilisateur.

La naissance de la nouvelle génération du Web (i.e. le Web sémantique) est un des efforts pour pallier à ces exigences. L'idée du Web sémantique est de modéliser le contenu des ressources du Web en ajoutant de la sémantique sous forme de métadonnées en vue de rendre les ressources compréhensibles par des machines (Berners-Lee *et al.* 2001) Autrement dit, il s'agit de décrire ces ressources selon une représentation formelle avec une sémantique clairement définie et qui soit conçue pour une interprétation par des programmes. La base de l'infrastructure du Web sémantique s'appuie sur l'explicitation de la conceptualisation d'un domaine, partagée par une communauté et représentée dans une ontologie du domaine concerné.

L'objectif du Web sémantique est de rendre explicite le contenu sémantique des ressources dans le Web (documents, pages web, services, etc.). Les machines et les agents logiciels pourraient "comprendre" les contenus décrits dans les ressources et faciliter les tâches de traitement des informations de façon plus automatique et plus efficace.

Les recherches actuellement réalisées dans le domaine du Web Sémantique s'appuient sur un existant riche provenant de différents domaines. Le Web sémantique est non seulement appliqué dans les recherches intelligentes d'information, mais il est aussi étudié dans les recherches sur l'ingénierie des connaissances, les systèmes de représentation des connaissances, le traitement automatique de la langue naturelle, l'apprentissage, les agents intelligents, le raisonnement automatique, etc.

2.2.1 Architecture du web sémantique

Le Web Sémantique est un concept visant à permettre aux machines de comprendre la signification de l'information sur le Web. Le but est ainsi de mettre en place, en plus du réseau des hyperliens entre les pages Web classiques, un réseau de liens entre les données structurées. Les machines accèdent plus intelligemment aux différentes sources de données contenues sur le Web et peuvent effectuer des traitements plus précis pour les utilisateurs. L'objectif du Web sémantique est, tout simplement, d'apporter la sémantique formelle nécessaire pour que des machines, elles aussi, puissent consulter et interpréter les informations présentées sur le Web.

Autour du Web Sémantique gravite une très grande quantité de technologies, voici un schéma résumant les différentes couches et langages du Web Sémantique :
Les couches les plus bases assurent l'interopérabilité syntaxique : la notion d' **URI (Uniform Resource Identifier)** fournit un adressage standard universel permettant d'identifier les ressources tandis que Unicode est un encodage textuel universel pour échanger des symboles.

XML (Extensible Markup Language) fournit une syntaxe pour décrire la structure du document, créer et manipuler des instances des documents. Il utilise l'espace de nommage (namespace) afin d'identifier les noms des balises (tags) utilisées dans les documents XML. Le XML schéma permet de définir les vocabulaires pour des documents XML valides. Cependant, XML n'impose aucune contrainte sémantique à la signification de ces documents, l'interopérabilité syntaxique n'est pas suffisante pour qu'un logiciel puisse "comprendre" le contenu des données et les manipuler d'une manière significative.

Les couches **RDF (Resource Description Framework)** et RDF Schéma sont considérées comme les premières fondations de l'interopérabilité sémantique. Elles permettent de décrire les taxonomies des concepts et des propriétés (avec leurs signatures). RDF fournit un moyen d'insérer de la sémantique dans un document, l'information est conservée principalement sous forme de déclarations RDF. Le RDF schéma décrit les hiérarchies des concepts et des relations entre les concepts, les propriétés et les restrictions domaine/co-domaine pour les propriétés.

La couche suivante **OWL (Web Ontology Language)** est un langage de représentation des connaissances construit sur le modèle de données de RDF. Il fournit les moyens pour définir des ontologies web structurées. Il décrit des sources d'information hétérogènes, distribuées et semi-structurées en définissant le consensus du domaine commun et partagé par plusieurs personnes et communautés.

Les règles sont aussi un élément clé de la vision du Web sémantique,

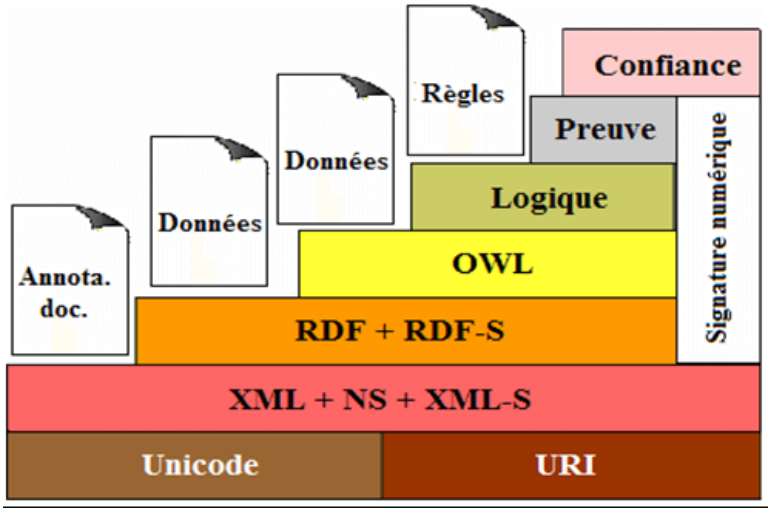

FIGURE 2.1 – *Couches du Web Sémantique (Charlet & Kembellec 2016)*

la couche Règles offre la possibilité et les moyens de l'intégration, de la dérivation, et de la transformation de données provenant de sources multiples, etc.

La couche **Logique** se trouve au-dessus de la couche OWL. Certains considèrent ces deux couches comme étant au même niveau, comme des ontologies basées sur la logique et permettant des axiomes logiques. En appliquant la déduction logique, on peut inférer de nouvelles connaissances à partir d'une information explicitement représentée.

Les couches **Preuve (Proof)** et Confiance (Trust) sont les couches restantes qui fournissent la capacité de vérification des déclarations effectuées dans le Web Sémantique. On s'oriente vers un environnement du Web sémantique fiable et sécurisé dans lequel nous pouvons effectuer des tâches complexes en sûreté. D'autre part, la provenance des connaissances, des données, des ontologies ou des déductions est authentifiée et assurée par des signatures numériques, dans le cas où la sécurité est importante ou le secret nécessaire, le chiffrement est utilisé. Les trois langages RDF et RDFS, OWL seront présentés plus en détail dans la section 2.4.

2.2.2 Composants principaux du Web sémantique

Dans cette section, nous allons voir les deux composants essentiels et importants du Web sémantique que sont l'ontologie et l'annotation sémantique. Les ontologies sont la technologie dorsale pour le Web sémantique et pour le management des connaissances formalisées décrivant les ressources du Web. Elles fournissent la "sémantique" exploitable par machine des données et des sources d'informations qui peuvent être communiquées entre différents agents (logiciel et humaines), tandis que les annotations sémantiques décrivent les ressources en utilisant la "sémantique"

définie dans l'ontologie. Les ressources annotées par les méta-données faciliteront la recherche, l'extraction, l'interprétation et le traitement de l'information d'une manière plus efficace.

Ontologie

L'importance des ontologies est reconnue dans divers domaines de recherche comme la représentation des connaissances, l'ingénierie des connaissances, la conception de bases de données, l'intégration des données, les systèmes d'information... Les ontologies sont aussi essentielles pour le Web sémantique qui, d'une part, cherche à s'appuyer sur des modélisations de ressources du Web à partir de représentations conceptuelles des domaines concernés et, d'autre part, a pour objectif de permettre à des programmes de faire des inférences dessus. Nous allons présenter par la suite dans la section 2.3 la définition et les composants de l'ontologie ainsi que certains outils de construction de l'ontologie qui sont largement utilisés d'aujourd'hui.

Méta-données et Annotations sémantiques

L'information du Web actuel devient plus fortement distribuée, extrêmement volumineuse, évolutive, très hétérogène et souvent très peu structurée. Afin de mieux utiliser l'information et les ressources du Web, il est nécessaire de proposer des méthodes et des outils pour représenter, manipuler et exploiter des ressources. Nous allons discuter dans cette section certains moyens et outils qui visent à améliorer la communication et l'interopérabilité des applications ainsi qu'à partager et exploiter l'information sur le Web.

Une **annotation** peut être considérée d'une manière simple comme une information graphique ou textuelle attachée à un document et le plus souvent placée dans ce document. En ce qui concerne le Web, les annotations les plus courantes sont des annotations en langage naturel (informelles), des symboliques (surlignée, soulignage, italique), des notes textuelles en marge, des images, des sons, etc.

Une annotation est toujours associée à l'objet qui a été annoté. Dans ce sens, les annotations sont considérées comme des méta-données. Les **méta-données** peuvent être définies comme étant des données relatives à d'autres données : données sur des données. La méta-donnée est une information interprétable par machine sur des ressources d'information du Web ou d'autres sources de données. Si une méta-donnée est une donnée sur une donnée, une annotation constitue un cas particulier d'une métadonnée puisqu'elle représente une nouvelle donnée attachée à une

ressource documentaire. D'un point de vue plus lié à la pratique de l'annotation et de métadonnées.

Puisque les données actuelles du Web sont destinées essentiellement aux humains, elles ne sont pas très bien structurées et n'ont pas de sémantique formelle. Par conséquent, un des objectifs, dans l'environnement du Web Sémantique, est de décrire le contenu des ressources en les annotant avec des informations non ambigües afin de favoriser l'exploitation de ces ressources par des agents logiciels. Cet objectif est considéré comme la tâche d'annotation consistant donc à prendre en entrée une ressource documentaire et fournir en sortie le même contenu enrichi par des annotations sémantiques basées sur des représentations de la connaissance plus ou moins formelles.

2.3 Ontologie

Les ontologies occupent désormais une place importante dans la recherche en informatique et sont utilisées ou proposées à l'utilisation, dans de nombreux domaines (intégration de données, Web sémantique, Web services, E-commerce, bases de données, etc.). Le caractère formel et consensuel d'une ontologie permet de la diffuser dans une communauté et contribue à pouvoir rendre interopérable les applications en représentant explicitement la sémantique des données.

L'utilisation croissante d'ontologies dans divers domaines (techniques ou documentaires) fait qu'aujourd'hui, certaines données sont déjà représentées comme des instances de classes d'ontologies et ont donc leur sémantique définie à partir d'ontologies. Nous appellerons ces données des données à base ontologique. Le besoin de représenter les ontologies et les données à base ontologique dans de vraies bases de données plutôt qu'en mémoire centrale ou dans des fichiers ordinaires se fait de plus en plus sentir.

L'ontologie, qui est considérée comme un ensemble structuré de termes et de concepts d'un domaine particulier en précisant les relations entre ces termes et leurs propriétés. Chaque terme d'une ontologie doit posséder une définition pour être sûr de la signification qui y est associée. Une telle représentation a la capacité à structurer les connaissances explicites du domaine étudié et d'en déduire d'autres connaissances implicites intéressantes. Cela comprend l'utilisation des outils de raisonnement sur une ontologie. En effet, en plus des éléments purement descriptifs que constituent les termes et leurs relations, une ontologie comprend des règles de raisonnement et il existe des algorithmes permettant de parcou-

rir le graphe en suivant les relations et les contraintes entre les termes.

2.3.1 Définition

Une ontologie est « **une spécification explicite d'une conceptualisation**» (Gruber 1995). Le terme « conceptualisation », dans la définition, fait référence à un système de concepts, autrement dit à un ensemble structuré de concepts. L'expression « spécification explicite » signifie, pour sa part, que la conceptualisation est représentée dans un langage. Ce langage peut être une langue naturelle (ex : français, anglais) ou un langage formel (ex : logique du 1er ordre, réseau sémantique).

2.3.2 Composants d'ontologie

Les modèles d'ontologies sont basées sur cinq sortes de composants principaux : 1. Les classes, 2. Les propriétés, 3. Les types de valeurs, 4. Les axiomes, 5. Les instances. 6. les relations (Dehainsala 2007)

Les classes

Une classe est la description abstraite d'un ou plusieurs objets semblables. Une classe correspond à ce qui a une existence propre, matérielle ou immatérielle, dans le domaine étudié du monde réel. Les classes sont toujours organisées en graphe acyclique à l'aide de relations de subsomption. Une classe possède toujours un identifiant. Sa définition comporte toujours une partie textuelle, qui permet de la rattacher à une connaissance préexistante de l'utilisateur, et une partie formelle constituée de relations avec d'autres composants de l'ontologie. Par exemple, les concepts Personne et Doctorant représentent des classes différentes des objets humains en réalité dans lesquelles Personne est un super-concept de Doctorant, le concept Manuscrit décrit une classe des objets du type de document, etc.

Les propriétés

Les propriétés sont des éléments qui permettent de caractériser et de distinguer les instances d'une classe. Comme les classes, les propriétés possèdent toujours un identifiant et une définition comportant une partie textuelles et une partie formelle. Les propriétés peuvent être fortement typées, i.e., associées un domaine et un co-domaine précis qui permettent respectivement de spécifier les classes susceptibles d'initialiser une propriété et de contraindre les domaines de valeurs des propriétés.

Une propriété peut prendre ses valeurs soit dans des types simples, soit dans des classes. Le terme propriété regroupe donc les notions parfois distinguées (par exemple dans le modèle entité-association) d'attribut et d'association. Dans l'exemple ci-dessus, une propriété rédiger peut indiquer une relation entre les concepts Doctorant et Manuscrit dans laquelle Doctorant est le domaine et Manuscrit est le co-domaine de la propriété rédiger.

Les différents langages de définition des ontologies permettent d'exprimer des caractéristiques spécifiques sur les propriétés. Par exemple, OWL permet d'exprimer des caractéristiques algébriques (symétrique, transitive, etc.)

Les types de valeurs

Les types de valeurs définissent des ensembles de valeurs dans lesquels les propriétés doivent prendre leurs valeurs, ainsi le cas échéant, que les opérations pouvant porter sur ces valeurs. Les types autorisés incluent toujours (1) des types simples (en particulier : entier, réel, caractère, booléen, etc.), (2) les classes (de telles propriétés représentent alors des associations) et (3) des collections, en général associées à des cardinalités minimum et maximum.

Les axiomes

Les axiomes sont des prédicats qui s'appliquent sur les classes ou les instances des classes de l'ontologie et qui permettent de restreindre les interprétations possibles d'une ontologie et/ou de déduire de nouveaux faits à partir des faits connus.
Pratiquement tous les langages permettent d'exprimer des axiomes de typage (pour des instances, des valeurs ou des liens), de subsomption et de cardinalité. Les autres axiomes exprimables dépendent de chaque modèle particulier d'ontologie et des prédicats particuliers définis au niveau du langage. Par exemple, OWL permet de contraindre une propriété à avoir une valeur dans une certaine classe (SOME).

Les instances

Les instances ou objets ou individus représentent des éléments spécifiques d'une classe. Chaque instance est caractérisée par son appartenance à une (ou éventuellement plusieurs) classes(s) et par des valeurs de propriétés. Selon les langages, les instances peuvent, ou non, être associées à un identifiant unique qui permet de distinguer une instance à une autre.

C'est l'hypothèse d'unicité de nom (Unique Name Assumption UNA) faite par certains langages (OWL Flight) et non par d'autres (OWL). Il est possible dans certains langages de définition d'ontologie qu'une instance appartienne à plusieurs classes en même temps. C'est la notion de multi-instanciation. Par exemple, l'instance "LEHIRECHE" est du type concept Doctorant, "ISEDAW" est une instance du concept Equipe_Recherche, etc.

Les relations

Représentent des liens sémantiques de la connaissance du domaine. Une relation peut être distinguée comme une propriété ou un attribut. Les relations décrivent des interactions entre concepts.

2.3.3 Typologie des ontologies

Les ontologies sont classées sur différents niveaux, selon le but pour lequel elles sont conçues, nous pouvons distingués : (Guarino *et al.* 2009)

— **Ontologies de domaine** : les plus connues, elles expriment des conceptualisations spécifiques à un domaine, elles sont réutilisables pour des applications sur ce domaine.

— **Ontologies d'application** : elles contiennent des connaissances du domaine nécessaires à une application donnée ; elles sont spécifiques et non réutilisables.

— **Ontologies génériques** : appelées aussi ontologies de haut niveau, elles expriment des conceptualisations très générales tels que le temps, l'espace, l'état, le processus, les composants, elles sont valables dans différents domaines ; les concepts figurant dans une ontologie du domaine sont subsumés par les concepts d'une ontologie générique, la frontière entre les deux étant floue.

— **Ontologies de représentation ou méta-ontologies** : indiquent des formalismes de représentation de la connaissance ; les ontologies génériques ou du domaine peuvent être écrites en utilisant des primitives d'une telle ontologie.

2.3.4 Outils de développement des ontologies

Il existe plusieurs outils pour développer et maintenir des ontologies. Nous introduisons par la suite certains de ces outils qui sont largement utilisés par la communauté des ontologistes.

Protégé [3] est un outil d'édition de l'ontologie développé à l'Université de Stanford, qui est utilisé largement aujourd'hui pour élaborer des ontologies en RDF(S) et OWL. Protégé fournit un environnement de développement graphique et interactif pour aider les ingénieurs et les experts du domaine à réaliser des tâches plus facilement. Le modèle de connaissances de Protégé est compatible avec l'OKBC (Open Knowledge Base Connectivity). Un des avantages de Protégé est son architecture ouverte et modulaire, il facilite le développement de nouvelles fonctionnalités à travers des plugs-in pour effectuer des opérations différentes.

NeOn Toolkit [4] est un éditeur d'ontologie initialement développé dans le cadre du projet NeOn et maintenant pris en charge, avec d'autres technologies de NeOn, par la Fondation NeOn.
NeOn Toolkit est un environnement d'ingénierie d'ontologie multiplateforme open source à la pointe de la technologie, qui fournit une prise en charge complète du cycle de vie de l'ingénierie d'ontologie. L'outil est basé sur la plate-forme Eclipse, et fournit un ensemble complet de plug-ins (actuellement 45 plug-ins sont disponibles) couvrant une variété d'activités d'ingénierie d'ontologie, y compris l'annotation et la documentation, le développement, acquisition de connaissances, gestion, modularisation et personnalisation.

2.4 Langages du Web sémantique

Dans le contexte du Web Sémantique, plusieurs langages ont été développés. La plupart de ces langages reposent sur XML ou utilisent XML comme syntaxe. Nous allons présenter brièvement certains langages principaux RDF, RDF Schéma, OWL, et SPARQL.

2.4.1 Ressource Description Framework (RDF)

Définition

Le langage RDF permet d'exprimer de l'information au sujet de ressources. Les ressources peuvent être quelconques, notamment des documents, des personnes, des objets physiques ou des concepts abstraits.
En particulier, RDF [5] peut être utilisé pour publier et interconnecter des données sur le Web. Par exemple, récupérer http ://www.example.org/majid#me pourrait fournir des données sur Arnaud, y compris le fait qu'il connait Béatrice, via son IRI (un IRI

3. https ://protege.stanford.edu/
4. http ://neon-toolkit.org/wiki/Main*page.html*
5. https ://www.emse.fr/ zimmermann/W3C/RDF1.1Primer/

est un Identifiant de Ressource Internationalisé Récupérer l'IRI de Majid pourrait alors fournir plus de données sur elle, y comprit des liens vers d'autres ensembles de données pour ses amis, ses centres d'intérêt, etc. Une personne ou un processus automatique peut alors suivre ces liens et agréger des données sur ces diverses choses. De telles utilisations de RDF sont souvent qualifiés de données liées (Linked Data) (Lefrançois & Zimmermann 2017)

Le modèle de données RDF

Triplet RDF nous permet de faire des déclarations sur les ressources. Le format de ces déclarations est simple. Une déclaration a toujours la structure suivante :

<sujet> <prédicat> <objet>

Une déclaration RDF exprime une relation entre deux ressources. Le sujet et l'objet représentent les deux ressources en relation ; le prédicat représente la nature de leur relation. La relation est formulée de manière directionnelle (depuis le sujet vers l'objet) et est appelée dans RDF une propriété. Puisque les déclarations RDF consistent en trois éléments, elles sont appelées triplets.
Exemple 1 échantillon de triplets (informels)

<Majid> <est une> <personne>.
<Majid> <est un ami de> <Ali>.
<Majid> <est né> <le 14 juillet 1990>.
<Majid> <s'intéresse à> < Les Hirondelles de Kaboul>.
< Les Hirondelles de Kaboul> <a été écrit par> < Yasmina Khadra>.
< Les Hirondelles de Kaboul > <est adaptée en > < film d'animation >.

La même ressource est souvent référencée dans plusieurs triplets. Dans l'exemple ci-dessus, Majid est le sujet de quatre triplets, et Les Hirondelles de Kaboul est le sujet de deux triplets et l'objet d'un seul triplet. Cette capacité à avoir la même ressource dans la position de sujet d'un triplet et la position d'objet d'un autre permet de trouver des connexions entre les triplets, qui sont une partie importante de la puissance de RDF.
Nous pouvons visualiser les triplets comme un graphe connexe. Les graphes sont constitués de nœuds et d'arcs. Les sujets et les objets des triplets forment les nœuds du graphe ; les prédicats forment des arcs. La figure 2.2 montre le graphe associé aux triplets de l'exemple 1.

IRI L'abréviation IRI signifie «International Resource Identifier» (identifiant de ressources internationnalisé). Un IRI identifie une ressource. Les

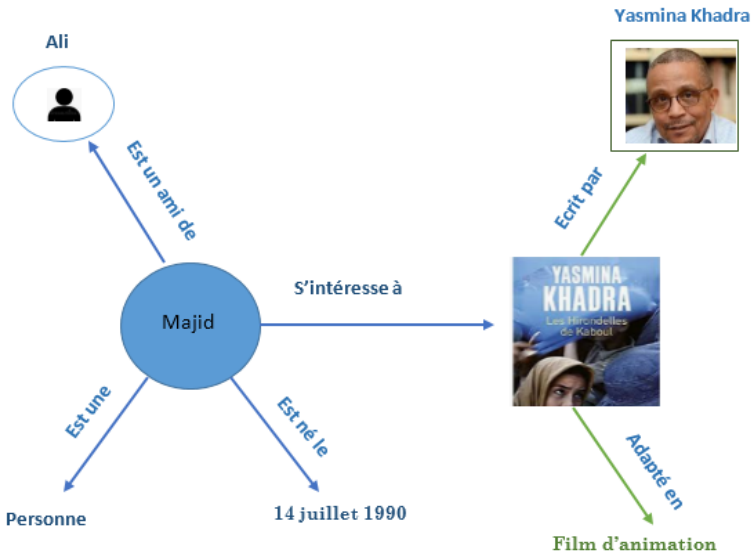

FIGURE 2.2 – *Graphe associé à l'échantillon de triplets de l'exemple 1.*

URL (Uniform Resource Locators ou localisateurs de ressource uniformes) que les gens utilisent comme adresses Web sont une forme d'IRI. D'autres formes d'IRI fournissent un identifiant pour une ressource sans implication sur leur localisation ni la manière d'y accéder. La notion d'IRI est une généralisation des URI (Uniform Resource Identifier ou identifiant de ressources uniforme) qui permet d'utiliser des caractères non-ASCII dans les chaînes de caractères des IRI. Par exemple, l'IRI pour Léonard de Vinci dans DBpedia est : http ://fr.dbpedia.org/resource/Léonard_de_Vinci.
Les IRI sont des identifiants globaux, donc d'autres personnes peuvent réutiliser cet IRI pour identifier la même chose. Par exemple, l'IRI suivant est utilisé par de nombreuses personnes comme propriété RDF pour indiquer une relation de connaissance entre personnes : http ://xmlns.com/foaf/0.1/knows.

Littéraux Les littéraux sont des valeurs de base qui ne sont pas des IRI. Comme exemples de littéraux, on trouve les chaines de caractères tels que «livre», des dates telles que «le 14 juillet 1990» et des nombres tels que «3.14159». Les littéraux sont associés à un type de données permettant à ces valeurs d'être traitées et interprétées correctement. Les littéraux de type chaines de caractères peuvent éventuellement être associées à une balise pour la langue. Par exemple, «Yasmina khadra» pourrait être associé à la balise de langue «dz»
Les littéraux ne peuvent apparaître qu'en position d'objet d'un triplet.

Nœud vide Les IRI et les littéraux constituent la matière première

pour rédiger des déclarations en RDF. En outre, il est parfois pratique de pouvoir parler de ressources sans avoir besoin d'utiliser un identificateur global. Par exemple, nous pourrions vouloir exprimer que le livre « Les Hirondelles de Kaboul » sur sa couverture à l'arrière-plan il y'a une femme non identifiée. Une ressource sans identificateur global, telle que le cyprès de la peinture, peut être représenté en RDF par un nœud vide. Les nœuds vides sont comme des simples variables en algèbre : ils représentent quelque chose sans dire quelle est leur valeur.
Les nœuds vides peuvent apparaître en position de sujet et d'objet d'un triplet. Ils peuvent être utilisés pour dénoter des ressources sans les nommer explicitement avec un IRI.

Graphe RDF Un certain nombre de formats de sérialisation existent pour écrire des graphes RDF. Cependant, différentes façons d'écrire le même graphe conduisent exactement aux mêmes triplets, et sont donc logiquement équivalents. Les formats sont les suivants.
1. la famille Turtle de langages RDF (N-Triples[6], Turtle[7], TriG[8] et N-Quads[9]) ;
2. JSON-LD[10](syntaxe RDF fondée sur JSON) ;
3. RDFa[11](pour l'incorporation dans HTML et XML) ;
4. RDF/XML[12](syntaxe XML pour RDF).

Graphes multiples RDF fournit un mécanisme pour grouper des déclarations RDF dans des graphes multiples et associer de tels graphes à un IRI. Les graphes multiples sont une extension récente du modèle de données RDF. En pratique, les concepteurs du RDF et de gestionnaires de données avaient besoin d'un mécanisme pour parler d'un sous ensemble d'une collection de triplets. Les graphes multiples ont été d'abord introduits dans le langage de requête SPARQL. Le modèle de données RDF a alors été étendu avec une notion de graphes multiples étroitement alignée avec SPARQL.
Des graphes multiples dans un document RDF constituent un ensemble de données RDF. Ce dernier peut avoir plusieurs graphes nommés et au moins un graphe sans nom («par défaut).
Par exemple, les déclarations de l'exemple 1 pourraient être groupées en deux graphes nommés. Un premier graphe pourrait être fourni par un

6. https ://www.w3.org/TR/n-triples/
7. https ://www.w3.org/TR/turtle/
8. https ://www.w3.org/TR/trig/
9. https ://www.w3.org/TR/n-quads/
10. https ://json-ld.org/
11. https ://www.w3.org/TR/rdfa-primer/
12. https ://www.w3.org/TR/rdf-syntax-grammar/

site de réseau social et identifié par http ://example.org/majid :

Exemple 2 : premier graphe de l'échantillon de l'ensemble de données

<Majid> <est une> <personne>.
<Majid> <est un ami de> <Ali>.
<Majid> <est né> <le 14 juillet 1990>.
<Majid> <s'intéresse à> < Les Hirondelles de Kaboul>.

L'IRI associé au graphe est appelé le nom du graphe.
Un second graphe pourrait être fourni par Wikidata et identifié par https ://www.wikidata.org/wiki/Special :EntityData/Q12418 :

Exemple 3 : second graphe de l'échantillon de l'ensemble de données

< Les Hirondelles de Kaboul> <a été écrit par> < Yasmina Khadra>.
< Les Hirondelles de Kaboul > <est adaptée en > < film d'animation >.

2.4.2 RDF Schéma (RDFS)

Le modèle de données RDF fournit un moyen de faire des déclarations sur des ressources. Ce modèle de données ne fait pas d'hypothèse sur ce que les IRI des ressources signifient. En pratique, RDF est typiquement utilisé en association avec des vocabulaires ou d'autres conventions qui fournissent des informations sémantiques sur ces ressources.
Pour permettre la définition de vocabulaires, RDF propose le langage RDF Schema. Ce langage permet de définir des caractéristiques sémantiques des données RDF. Par exemple, on peut déclarer que l'IRI http ://www.example.org/amiDe peut être utilisé comme propriété et que les sujets et objets des triplets ayant http ://www.example.org/amiDe comme prédicat doivent être des ressources de la classe http ://www.example.org/Personne.
RDF Schema utilise la notion de classe (classe) pour spécifier les catégories qui peuvent être utilisées pour classer des ressources. La relation entre une instance et sa classe est déclarée via la propriété type. Avec RDF Schéma, on peut créer des hiérarchies de classes et de sous-classes et de propriétés et sous-propriétés. Des restrictions de types sur les sujets et objets de triplets particuliers peuvent être définies via les restrictions de domaine (domain) et de co-domaine (range). Un exemple de restriction de domaine était donné ci-dessus : les sujets des triplets "amiDe" devraient être de la classe "Personne".

L'un des premiers vocabulaires RDF utilisé mondialement était le vocabulaire "Friend of a Friend" (FOAF) [13] pour décrire des réseaux sociaux. D'autres exemples de vocabulaires RDF sont :

Dublin Core [14] : L'initiative des métadonnées Dublin Core (Dublin Core Metadata Initiative) maintient un ensemble d'éléments de métadonnées pour décrire un large éventail de ressources. Ce vocabulaire offre des propriétés telles que "creator" (créateur), "publisher" (éditeur) et "title" (titre).

schema.org [15] : Schema.org est un vocabulaire développé par un groupe de grands fournisseurs de moteurs de recherche. L'idée est que les webmestres peuvent utiliser ces termes pour baliser des pages Web, afin que les moteurs de recherche comprennent de quoi parlent ces pages.

SKOS [16] : SKOS est un vocabulaire pour publier des schémas de classification tels que des terminologies et des thésaurus sur le Web. SKOS est depuis 2009 une recommandation du W3C et est largement utilisé dans le monde des bibliothèques.

2.4.3 Web Ontology language(OWL)

OWL est un langage pour représenter des ontologies dans le Web sémantique. C'est une extension du vocabulaire de RDF(S). Le langage OWL offre aux machines de plus grandes capacités d'interprétation du contenu Web que celles permises par XML, RDF et RDF schéma (RDFS), grâce à un vocabulaire supplémentaire et une sémantique formelle. Inspiré des logiques de descriptions, OWL fournit un grand nombre de constructeurs permettant d'exprimer de façon très fine les classes de manière plus complexe correspondant aux connecteurs de la logique de description équivalente (intersection, union, restrictions diverses, etc.), les propriétés des classes définies (telles que la disjonction), la cardinalité (par exemple "exactement un"), plus des types des propriétés (propriétés d'objet ou d'annotation...), des caractéristiques des propriétés (par exemple la symétrie, la transitivité), et des classes énumérées.

Le langage d'ontologie Web OWL définit et instancie des ontologies Web. Une ontologie OWL peut contenir des descriptions de classes, de propriétés et de leurs instances. Pour une telle ontologie, la sémantique formelle OWL indique comment déduire ses conséquences logiques, c'est-à-dire les faits non pas littéralement présents dans l'ontologie mais qui découlent

13. http ://xmlns.com/foaf/spec/
14. https ://dublincore.org/
15. https ://schema.org/
16. https ://www.w3.org/2004/02/skos/

```
<rdf:RDF
 <xmlns: xsd "http://www.w3.org/2001/XMLSchema#">
 < xmlns:  rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 < xmlns:  rdfs "http://www.w3.org/2000/01/rdf-schema#">
 < xmlns:  owl "http://www.w3.org/2002/07/owl#">

<owl:Ontology  rdf:about= "Herbivore"
 <rdfs:subClassOf rdf:resource="#Animal"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty  rdf:resource  ="#mange">
<owl:allValuesFrom  rdf:resource  ="#plante">
</owl:Restriction>
   </rdfs:subClassOf >
 </owl:Class>
</rdf:RDF>
```

FIGURE 2.3 – *Exemple de document OWL*

de la sémantique.

Ces inférences peuvent être fondées sur un seul document ou sur plusieurs documents répartis combinés à l'aide de mécanismes OWL définis. Une ontologie formalisée en OWL comprend Un espace de nom. L'entête pour décrire l'ontologie. La définition des classes, des propriétés et des instances.

Le langage OWL offre trois sous-langages d'expression croissante conçus pour des communautés de développeurs et d'utilisateurs spécifiques.

— Le langage OWL Lite concerne les utilisateurs ayant principalement besoin d'une hiérarchie de classifications et de mécanismes de contraintes simples. Par exemple, OWL Lite gère des contraintes de cardinalité, il ne permet que des valeurs de cardinalité de 0 ou 1. Mettre en œuvre des outils pour OWL Lite devrait être plus simple que pour ses parents d'expression plus grande, tout comme tracer un chemin de migration rapide pour des thésaurus et autres taxonomies.

— Le langage OWL DL concerne les utilisateurs souhaitant une expressivité maximum sans sacrifier la complétude de calcul (toutes les inférences sont sûres d'être prises en compte) et la décidabilité (tous les calculs seront terminés dans un intervalle de temps fini) des systèmes de raisonnement. Le langage OWL DL comprend toutes les structures de langage de OWL avec des restrictions comme la sé-

paration des types (une classe ne peut pas être en même temps un individu ou une propriété, une propriété être un individu ou une classe). OWL DL se nomme ainsi pour sa correspondance avec la logique de description, un champ de la recherche portant sur un fragment décidable particulier de la logique de premier ordre.

— Le langage OWL Full est destiné aux utilisateurs souhaitant une expressivité maximum et la liberté syntaxique de RDF sans garantie de calcul. Par exemple, dans OWL Full, on peut simultanément traiter une classe comme une collection d'individus et comme un individu à part entière. Une autre différence significative par rapport à OWL DL réside dans la possibilité de marquer un objet owl :DatatypeProperty comme étant un objet owl :InverseFunctionalProperty. Le langage OWL Full permet à une ontologie d'augmenter la signification du vocabulaire prédéfini (RDF ou OWL). Un système de raisonnement ne pourra probablement pas mettre en œuvre toutes les caractéristiques d'OWL Full.

2.4.4 SPARQL Query Language

Définiton

SPARQL (Sparql Protocol and RDF Query Langage) est le langage standard pour interroger les données représentées par des triplets, C'est l'un des trois normes fondamentales du Web sémantique, avec RDF et OWL. Il n'est pas limité à interroger des données stockées dans l'un des formats RDF.
La partie «Protocole» du nom de SPARQL fait référence aux règles régissant la manière dont un programme client et un serveur de traitement SPARQL échangent des requêtes et des résultats SPARQL. Ces règles sont spécifiées dans un document distinct du document de spécification de la requête et constituent principalement un problème pour les développeurs de processeurs SPARQL (DuCharme 2013)

Structure d'une requête SPARQL

La syntaxe d'une requête SPARQL est inspirée du SQL, une requête est un graphe RDF (graphes de triplets : sujet, prédicat, objet) avec des variables, où les ressources sont présentées par des URI (ou IRI) abrégées par des préfixes. Et les objets peuvent être : littéraux, chaines de caractères, booléens, entiers. SPARQL recherche des valeurs des variables qui sont des sous-graphes du graphe représentant les données.

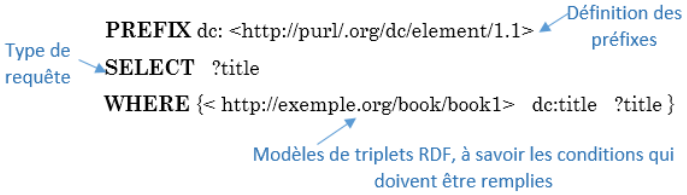

FIGURE 2.4 – *Structure d'une requête SPARQL*

Les **PREFIXES** permettent de définir des espaces de noms (préfixes : pour abréger les URI).
La clause **SELECT** projection et identification de variables et valeurs retournées par la requête (* toutes les variables utilisées dans la clause WHERE).
FROM : source (graphe de données) interrogée.
La clause **WHERE** contient des triplets qui définissent le motif (pattern) recherché.

Types de requêtes SPARQL

SPARQL a quatre formes de requête. Ces formes de requête utilisent les solutions de matching de patterns pour former les ensembles de résultats ou les graphes RDF. Les formes de requête sont :

SELECT Retour d'une table de tous X, Y, etc satisfaisant aux conditions suivantes ... (comme dans l'exemple en dessous : renvoyer le nom d'une organisation avec un URI spécifique). Des variables spécifiques et leurs liaisons sont renvoyés lorsqu'une liste de noms de variables est donnée dans la clause SELECT. La syntaxe SELECT * est une abréviation qui sélectionne toutes les variables qui sont dans la portée à ce point de la requête. Il exclut les variables uniquement utilisées dans FILTER, dans la partie droite de MINUS, et prend en compte les sous-requêtes.

La clause SELECT peut également introduire de nouvelles variables. Les règles d'affectation dans l'expression SELECT sont les mêmes que pour l'affectation dans BIND. L'expression combine des liaisons de variables déjà présentes dans la solution de requête, ou définies précédemment dans la clause SELECT, pour produire une liaison dans la solution de requête.

CONSTRUCT Trouvez tous X, Y, etc répondant aux conditions sui-

Données d'échantillon

```
comp:A rov:haslegalName "Niké" .
comp:A org:hasRegisteredSite site:1234 .

Comp:B rov:haslegalName "BARCO" .

site:1234 locn:fullAddress "Dahliastraat 24, 2160 Wommelgem .
```

Requête

```
PREFIX comp: < http://example/org/org/>
PREFIX org: < http://www.w3.org/TR/vocab-regorg/ >
PREFIX site: <http://example.org/site/>
PREFIX rov: <http://www.w3.org/TR/vocab-regorg/>

SELECT ?name

WHERE
 { ?x org:hasRegisteredSite site:1234 .
   ?x rov:haslegalName ?name  .}
```

Résultat

name
"Niké"

FIGURE 2.5 – *la requête SELECT*

Données d'échantillon

```
comp:A rov:haslegalName "Niké" .
comp:A org:hasRegisteredSite site:1234 .

comp:B rov:haslegalName "BARCO" .

site:1234 locn:fullAddress "Dahliastraat 24, 2160 Wommelgem" .
```

Requête

```
PREFIX comp: < http://example/org/org/>
PREFIX org: < http://www.w3.org/TR/vocab-regorg/ >
PREFIC rdfs: <http://www.w3.org/TR/rdf-schema/>

CONSTRUCT {?comp rdfs:label ?name}

WHERE
 { ?comp org:haslegalName ?name. }
```

Graphe résultant

```
@prefix comp: <http://example/org/> .
@prefix rdfs: <http://www.w3.org/TR/rdf-schema/>

comp:a rdfs:label "Niké" .
comp:b rdfs:label "BARCO" .
```

FIGURE 2.6 – *la requête Construct*

vantes ... et les remplacer dans le modèle ci-dessous afin de générer des (potentiellement nouvelles) déclarations RDF, créant ainsi un nouveau graphe. (l'exemple : créer un nouveau graphe avec une autre étiquette pour le nom) La requête CONSTRUCT renvoie un seul graphe RDF spécifié par un modèle de graphique. Le résultat est un graphe RDF formé en prenant chaque solution de requête dans la séquence de solutions, en remplaçant les variables dans le modèle de graphe et en combinant les triplets en un seul graphe RDF par union définie.

DESCRIBE Trouvez toutes les déclarations dans l'ensemble de données qui fournissent des informations sur la ou les ressource(s) suivante(s) (identifiées par leurs noms ou descriptions). (exemple pour renvoyer tous les triplets des organisations inscrites sur un site particulier).

La forme DESCRIBE renvoie un graphe RDF à résultat unique conte-

FIGURE 2.7 – *la requête Describe*

nant des données RDF sur les ressources. Ces données ne sont pas prescrites par une requête SPARQL, où le client de requête devrait connaître la structure du RDF dans la source de données, mais est plutôt déterminé par le processeur de requêtes SPARQL. Le pattern de requête est utilisé pour créer un ensemble de résultats. DESCRIBE prend chacune des ressources identifiées dans une solution, ainsi que toutes les ressources directement nommées par IRI, et assemble un seul graphe RDF en prenant une "description" qui peut provenir de toute information disponible, y compris l'ensemble de données RDF cible. La description est déterminée par le service de requête. La syntaxe DESCRIBE * est une abréviation qui décrit toutes les variables d'une requête.

ASK Y a-t-il une quelconque X, Y, etc qui satisfait les conditions suivantes. Les applications peuvent utiliser la forme ASK pour tester si un modèle de requête a une solution. Aucune information n'est renvoyée sur les solutions de requête possibles, que la solution existe ou non....
(Dans l'exemple la requête permet de vérifier s'il y a des organisations ayant "1234" comme leur site enregistré ?).

Motifs de graphe SPARQL

SPARQL est basé sur la correspondance des motifs de graphe (graph pattern). Des gp plus complexes peuvent être formés en combinant des motifs plus petits de différentes manières.
Dans cette section, nous décrivons les deux formes qui combinent les patterns par conjonction : basic graph pattern (bgp), qui combine les triples patterns, et le group graph pattern, qui combinent tous les autres graphs patterns.

FIGURE 2.8 – *la requête ASK*

Motifs de graphe élémentaires (basic graph pattern bgp)

Les Motifs de graphe élémentaires sont des ensembles de motifs de triplet. La correspondance de motif de graphe SPARQL est définie en termes de combinaison des résultats de la correspondance de bgp.
Lorsque vous utilisez des nœuds vides de la forme _ : abc, les étiquettes des nœuds vides sont étendues au bgp. Une étiquette peut être utilisée dans un seul bgp dans n'importe quelle requête. SPARQL évalue le bgp en utilisant la correspondance de sous-graphes, qui est définie pour une implication simple.

Motifs de graphe de groupe (Group graph pattern)

Dans une chaîne de requête SPARQL, un group graph pattern est délimité par des accolades : . Par exemple, le pattern de cette requête est un group graph pattern comprenant un seul motif de graphe élémentaire.

PREFIX foaf : <http ://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE {
?x foaf :name ?name .
?x foaf :mbox ?mbox .
}
Les mêmes solutions seraient obtenues à partir d'une requête qui regroupait les triples patterns en deux bgp. Par exemple, la requête ci-dessous a une structure différente mais donnerait les mêmes solutions que la requête précédente :

```
PREFIX foaf : <http ://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { { ?x foaf :name ?name . }
{ ?x foaf :mbox ?mbox . }
}
```

Motif de groupe vide (Empty Group Pattern)

Le group pattern { }, fait correspondre n'importe quel graphe (y compris le graphe vide) avec une solution qui ne lie aucune variable. Par exemple :

```
SELECT ?x
WHERE {}
```

Correspond à une solution dans laquelle la variable x n'est pas lié.

Portée des filtres (Scope of Filters))

Une contrainte, exprimée par le mot-clé FILTER, est une restriction sur les solutions sur l'ensemble du groupe dans lequel le filtre apparaît. Les patterns suivants ont tous les mêmes solutions :

```
{ ?x foaf :name ?name .
?x foaf :mbox ?mbox .
FILTER regex( ?name, "Ali")
}
{ FILTER regex( ?name, "Ali")
?x foaf :name ?name .
?x foaf :mbox ?mbox .
}
{ ?x foaf :name ?name .
FILTER regex( ?name, "Ali")
?x foaf :mbox ?mbox .
}
```

Motifs de graphe optionnel (Optional Graph pattern)

Les bgp permettent aux applications d'effectuer des requêtes où le pattern de requête entier doit correspondre pour qu'il y ait une solution. Pour chaque solution d'une requête contenant uniquement des motifs de graphe de groupe avec au moins un bgp, chaque variable est liée à un terme RDF dans une solution. Cependant, des structures régulières et

complètes ne peuvent pas être supposées dans tous les graphes RDF. Il est utile de pouvoir disposer de requêtes permettant d'ajouter des informations à la solution là où elles sont disponibles, mais ne rejette pas la solution car une partie du pattern de requête ne correspond pas. La correspondance Optional fournit cette fonctionnalité : si la partie optional ne correspond pas, elle ne crée aucune liaison mais n'élimine pas la solution. Les parties Optional du graph pattern peuvent être spécifiées syntaxiquement avec le mot clé OPTIONAL appliqué à un graph pattern :

pattern OPTIONAL { pattern }

Les graphs patterns sont définis de manière récursive. Un graph pattern peut avoir zéro ou plusieurs optional graph patterns, et toute partie du pattern de requête peut avoir une partie optional. Dans cet exemple, il existe deux optional graph patterns.

```
PREFIX foaf : <http ://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE { ?x foaf :name ?name .
OPTIONAL { ?x foaf :mbox ?mbox } .
OPTIONAL { ?x foaf :homepage ?hpage }
}
```

Correspondance des alternatives (Matching Alternatives)

SPARQL fournit un moyen de combiner des graphs patterns afin que l'un des plusieurs graph pattern alternatifs puisse correspondre. Si plusieurs alternatives correspondent, toutes les solutions du pattern possibles sont trouvées.
Les alternatives de pattern sont spécifiées syntaxiquement avec le mot-clé UNION.

```
PREFIX dc10 : <http ://purl.org/dc/elements/1.0/>
PREFIX dc11 : <http ://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10 :title ?title } UNION { ?book dc11 :title ?title } }
```

2.5 Données Liées

2.5.1 définition

"Les données liées sont un ensemble de principes de conception pour le partage de données lisibles par machine sur le Web pour une utilisation par les administrations publiques, les entreprises et les citoyens."(Heath & Bizer 2011)

2.5.2 Principes des Données liées

Comment les données liées sont intégrés dans l'architecture technique du web. • Utiliser URI (Uniform Ressource Identifier) comme des noms des entités.
• Utiliser HTTP URI, pour chercher ces noms (ces adresses).
• Quand quelqu'un cherche une URI, fournir des informations utiles en utilisant les standards (RDF, SPARQL).
• Inclure des liens vers d'autres URI, pour découvrir des nouvelles entités.

Un document Web est établi sur un ensemble de standards :
URI : mécanisme global et unique d'identification.
HTTP : mécanisme d'accès universel.
HTML : un format de contenu largement utilisé.
Hyperlinks entre les documents Web qui réside dans différents serveurs Web.
Les principes du Linked data sont :
1. URIs, 2. HTTP URIs, 3. RDF , 4. RDF Links : hyperliens dans le contexte des Linked Data.

Nommer les entités par des URI

Pour publier les données sur le web, les ressources d'un domaine donné doivent être identifiées par HTTP URI. (Décrit les propriétés et les relations).

Fabrication d'URI déréférencé

Chaque HTTP URI doit être déréférencé, les clients peuvent chercher l'URI en utilisant le protocole HTTP et récupérer la description de la ressource identifiée par l'URI.
Les descriptions qui sont prévues pour être lue par des humains, sont souvent représentées en HTML.

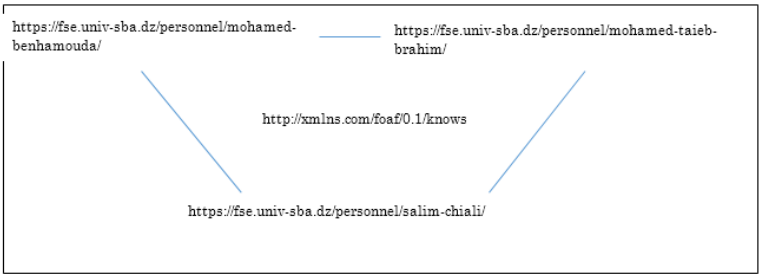

FIGURE 2.9 – *Utilisation des URI pour les noms des ressources.*

Les descriptions qui sont prévues pour la consommation par des machines, sont représentées comme des données RDF. Il est pratique d'employer des URI différents pour identifier l'objet réel, et le document qui le décrit.

Fournir des informations RDF

RDF fournit un modèle de données. Les deux formats les plus connus pour publier les Linked data sur le web sont RDF/XML et RDFa.
Dans RDF, une description d'une ressource est représentée par un nombre de triplets (subject, predicate, object), un triplet reflète la structure d'une phrase simple.
Exemple

Mohamed Reda has nick name moh
Subject Predicate Object

Considérer le triplet RDF comme un graphe RDF, sujet et objet comme des nœuds et un arc (prédicat) qui les relie. Il est possible d'imaginer toutes les Linked Data comme un seul graphe global géant.

Formats de sérialisation de RDF

La syntaxe RDF/XML est largement utilisé pour publier les Linked data sur le Web. **RDFa** : un format qui place les triplets RDF dans les documents HTML. **Turtle** : supporte les préfixes des espaces des noms et d'autres sténographies. **Ntriple**s : sont des sous-ensembles du Turtle. **RDF/JSON** : incluant les langages de programmation Web comme JavaScript et PHP.

Incluant des liens vers d'autres entités

Les RDF links pointent vers d'autres sources de données sur le web. Comme : External RDF links : qui sont fondamentaux pour permettent

aux applications de découvrir des nouvelles données.
3 types importants de RDF Links :
Relationship Links : pointent dans les entités des autres sources de données.
Identify Links : pointent dans les alias d'URI utilisés par d'autres sources de données, pour identifier la même entité.
Vocabulary Links : à partir des données, les liens pointent vers les définitions du vocabulaire des termes qui sont utilisés pour représenter les données.

2.6 Conclusion

Nous venons de présenter la notion du Web sémantique, la nouvelle génération du web actuel, avec une nouvelle infrastructure qui a pour ambition d'ajouter de la sémantique aux données pour permettre de les traiter automatiquement par machine. Afin de comprendre la signification de l'information sur le Web. Le but est ainsi de mettre en place, en plus du réseau des hyperliens entre les pages Web classiques, un réseau de liens entre données structurées. Les machines accèdent plus intelligemment aux différentes sources de données contenues sur le Web et peuvent effectuer des traitements plus précis pour les utilisateurs. Et comment mettre en place le Web Sémantique et quelles sont les technologies qui gravitent autours. Et par la suite nous avons présenté des principes de l'architecture du Web sémantique, des technologies et langages comme le RDF, RDFS, OWL et le langage d'interrogation SPARQL, ainsi que des composants principaux (i.e. ontologie, annotation sémantique.

Nous avons donné aussi une vue générale sur le concept et les principes de base des données liées, ainsi que des aperçus des technologies de support telles que les URI, HTTP et RDF. Ces technologies permettent un style de publication qui introduit des données au cœur du Web - une caractéristique unique qui expose les données liées au potentiel rigoureux et illimité du Web en général. Cette intégration absolue avec le Web, lui-même basé sur des standards ouverts et axés sur la communauté, constitue également une fonction de protection pour les éditeurs de données.

Les données liées ont été adoptées par un nombre important d'éditeurs de données qui, collectivement, ont construit un Web de données d'une échelle impressionnante. Ce faisant, ils ont démontré sa faisabilité en tant qu'approche de la publication de données sur le Web et la maturité croissante des plates-formes logicielles et des ensembles d'outils qui le supportent. Le développement d'outils est toujours informé non seulement par des normes et des spécifications, mais aussi par les meilleures

pratiques qui sont développées et adoptées dans la communauté concernée.

3 Intégration des Systèmes d'Information dirigés par Les Données liées

3.1 Introduction

L'intégration des données dans les grandes entreprises est un problème crucial mais en même temps coûteux, durable et difficile. Tandis que les informations stratégiques sont souvent déjà collectées dans des systèmes d'information intégrés tels que les systèmes ERP, CRM et SCM, l'intégration de ces systèmes ainsi que l'intégration des autres sources d'information restent un défi majeur.

Les grandes entreprises opèrent souvent avec des centaines voire des milliers de différents systèmes d'informations et bases de données. Par exemple, il est estimé que Volkswagen a environ 5000 systèmes d'informations différents déployés. Daimler même après une décennie d'efforts de consolidation - le nombre d'indépendants Systèmes IT atteint encore 3000 données. (Lehireche *et al.* 2017)

Différentes approches, techniques et méthodes ont été introduites afin de résoudre le défi de l'intégration de données. Dans les deux dernières décennies, les approches les plus répandue dans le domaine de l'intégration de données sont principalement basées sur XML, les services Web et les architectures orientées services (SOA). XML définit une syntaxe standard pour la représentation des données, les services Web fournissent des protocoles d'échange de données et la SOA est une approche pour l'architecture des systèmes distribués.

Les approches récentes, par exemple, considèrent l'intégration de données basées sur l'ontologie, où les ontologies sont utilisées pour décrire les données, les requêtes et les mapping entre eux. Le problème de l'intégration de données basé sur l'ontologie nécessite de hautes compétences (sémantiques intentionnelles) pour développer les ontologies afin de modéliser et capturer la dynamique de l'entreprise. Une approche connexe, mais légèrement différente est l'utilisation du paradigme Linked Data pour l'intégration de données d'entreprise. De même, comme le web de données a émergé en complétant aux Web de documents, les intranets de données peuvent compléter les intranets d'architecture SOA peuplant actuellement les grandes entreprises.

L'acquisition de Freebase [1] par Google et Powerset par Microsoft sont les premiers indicateurs, que les grandes entreprises non seulement utilisent le paradigme Linked Data pour l'intégration de leurs milliers de systèmes d'informations, mais ils ont également pour objectif de construire une bases de connaissances de l'entreprise (similaires à ce que Freebase est maintenant pour Google).

Dans ce chapitre nous explorons les défis auxquels sont confrontées les

1. https ://developers.google.com/freebase

grandes entreprises en matière d'intégration de données. Ceux-ci incluent, mais ne sont pas limités au développement, la gestion et l'interconnexion des taxonomies d'entreprise, des bases de données de domaine, des wikis et d'autres sources d'informations d'entreprises. Nous discutons de l'intégration des bases de données relationnelles avec l'approche de données liées qui fait appel aux mapping RDB-to-RDF, et les langages R2RML [2] et D2RQ [3] pour spécifier les règles de mapping afin de transformer le contenu relationnel en RDF.

3.2 Challenges de l'intégration des données dans l'entreprise

La figure 3.1 montre une vision globale sur le Web de données d'entreprise et l'architecture des systèmes informatiques d'entreprise sémantiquement liés qui en résulte. Les taxonomies d'entreprise existantes soient le noyau des centres de liaison et d'intégration dans les grandes entreprises, puisque ces taxonomies reflètent déjà une grande partie de la terminologie du domaine et de la culture organisationnelle et corporative. Afin de transformer les taxonomies d'entreprise en bases de connaissances complètes, des ensembles de données pertinents supplémentaires provenant du Linked Open Data Web doivent être intégrés et liés aux taxonomies internes et aux structures de connaissances. Par la suite, la base de connaissances d'entreprise émergente peut être utilisée (1) pour l'interconnexion et l'annotation de contenu dans des wikis d'entreprise, des systèmes de gestion de contenu et des portails ; (2) comme un ensemble stable de concepts et d'identifiants réutilisables ; et (3) en tant que connaissances de base pour les applications intranet, extranet et de recherche de site. Par conséquent, on cherche que les intranets actuels orientés documents dans les grandes entreprises soient complétés par un intranet de données, ce qui facilite l'intégration sémantique légère des systèmes d'information et de bases de données dans les grandes entreprises.

Dans la représentation Les lignes pleines montrent comment les systèmes informatiques peuvent être actuellement connectés dans un scénario typique. Les lignes pointillées visualisent comment les systèmes informatiques peuvent être liés en utilisant un cloud de données interne. L'EDW (Enterprise Data Web) comprend également une base de connaissances d'entreprise (EKB), qui comprend des définitions de vocabulaire, des copies des linked open data pertinentes, ainsi que des liens internes et externes entre les ensembles de données. Les données du cloud LOD

2. https ://www.w3.org/TR/r2rml/
3. http ://d2rq.org/d2rq-language

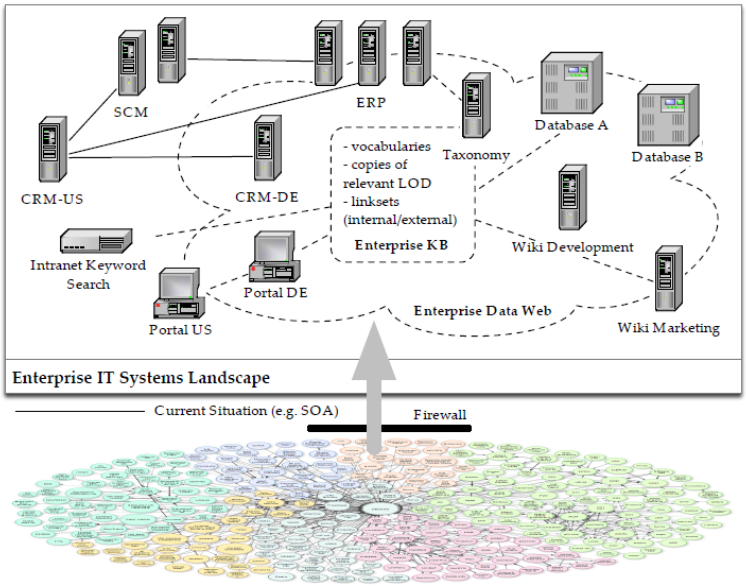

FIGURE 3.1 – *Présentation du système informatique du Web de Données d'Entreprise* (Frischmuth *et al.* 2012)

peuvent être réutilisées dans l'entreprise, mais les données internes sont sécurisées à partir d'un accès externe, comme dans les intranets habituels.

Les domaines identifiés où les défis de l'intégration des données se posent dans les grandes entreprises sont nombreux, et sont cités dans le tableau 3.1. En considérant d'abord la situation actuelle, et ensuite les avantages de l'utilisation des technologies de données liées afin de relever les défis respectifs. Nous décrivons à titre d'exemple les défis des taxonomies qui sont utilisées presque dans toutes les grandes entreprise pour fournir un modèle linguistique partagé visant à structurer les grandes quantités de documents, courriels, descriptions de produits, directives d'entreprise, etc. qui sont produits quotidiennement. Il existe de nombreux défis à relever pour que les taxonomies fonctionnent correctement.

Un problème évident est que la création de métadonnées pour les objets numériques est un travail supplémentaire qui présente peu d'avantages directs pour les créateurs des contenus correspondants. Ce problème peut cependant être résolu par une explication appropriée des avantages pour les parties intéressées. Idéalement, les créateurs de métadonnées devraient être en mesure de consommer les métadonnées d'autres créateurs afin qu'ils puissent voir directement les avantages de la description correcte des métadonnées. Un problème plus important et de longue durée, en particulier dans les grandes entreprises, est que différents créateurs de métadonnées utilisent des terminologies différentes et que le même objet peut recevoir différentes descriptions de métadonnées par différentes per-

sonnes.

TABLE 3.1 – *Vue d'ensemble des Challenges d'intégration de données. (Frischmuth et al. 2012)*

Information Integration Domain	Current state of the art	Linked Data benefit
Enterprise Taxonomies	proprietary, centralized, no relationships between terms, multiple independent terminologies (dictionnaires)	open standards (e.g. SKOS), distributed, hierarchical, multilingual, re-usable in other scenarios
XML Schema Governance	multitude of XML schemas, no integrated documentation	relationships between entities from different schemas, tracking/documentation of XML schema evolution
Wikis	text-based wikis for teams or internal-use encyclopedias	re-use of (structured) information via data wikis (by other applications), interlinking with other data sources, e.g. taxonomies
Web Portal and Intranet Search	keyword search over textual content	sophisticated search mechanisms employing implicit knowledge from different data sources
Database Integration	DataWarehouses, schema mediation, query federation	lightweight data integration through RDF layer
Enterprise Single Sign–On	consolidated user credentials, centralized SSO	no passwords, more sophisticated access control mechanisms (arbitrary metadata attached to identities)

Afin de pallier aux problèmes cités, il faut appliquer la transition vers les nouvelles technologies des données liées. Pour cela, nous présentons les taxonomies d'entreprise dans RDF en utilisant le vocabulaire SKOS (Miles & Bechhofer 2009) standardisé et largement utilisé ainsi que de publier des définitions de termes via les principes de données liées. Cette approche implique les principaux avantages suivants :

Grâce à cette approche, la quantité de temps et d'efforts nécessaires pour construire et maintenir une taxonomie d'entreprise puisse être répartie sur de nombreux postes, les coûts initiaux pour construire une taxonomie de base peuvent être assez élevés. Le problème de trouver des termes initiaux (et leurs définitions) peut être résolu en intégrant un service comme DBpedia Spotlight (Mendes *et al.* 2011) dans des outils déjà utilisés (par exemple des systèmes de gestion de contenu). Avec ce service, les textes peuvent être annotés avec DBpedia. Puisque DBpedia est dérivé de Wikipédia, les termes peuvent déjà contenir une description détaillée dans plusieurs langues dans de nombreux cas. Une telle description peut être utilisée comme point de départ pour le raffinement.

Pour les termes spécifiques aux entreprises, lorsqu'une description n'est pas disponible via DBpedia (ce qui sera souvent le cas dans les grandes entreprises telles que les entreprises automobiles avec une terminologie spécifique), un service d'extraction de mots clés comme FOX [4]peut être utilisé. Un tel service renverra également une URI bien connu si possible, mais retournera aussi les URI pour les autres mots-clés trouvés (avec ou sans métadonnées). Ainsi, un ensemble initial d'URI de terme peut être rassemblé, qui peut ensuite être annoté dans une seconde étape.

Un autre défi majeur identifié dans la plupart des entreprises aujourd'hui, où XML est utilisé pour l'échange de message, l'intégration de données, la publication et le stockage, souvent sous la forme de services Web et de bases de données XML. Pour pouvoir utiliser efficacement les documents XML, nous devons savoir quel type de données nous pouvons nous attendre à y trouver. Pour ce faire, les schémas XML [5] doivent être présents pour chaque format XML utilisé. Les schémas XML décrivent la structure autorisée d'un document XML.

Dans une entreprise typique, il existe des centaines, voire des milliers de schémas XML, chacun peut être écrit dans un langage de schéma

4. http ://aksw.org/Projects/FOX.html
5. https ://www.w3.org/TR/xmlschema11-1/

XML différent. De plus, à mesure que l'entreprise et son environnement évoluent, les schémas doivent s'adapter. Par conséquent, de nouvelles versions de schémas sont créées, ce qui entraîne une prolifération de schémas XML. La gouvernance de schéma XML est maintenant le processus de mise en ordre d'un grand nombre de schémas XML générés et utilisés dans de grandes organisations.

Il existe différents outils qui nous permettent de visualiser des schémas individuels (comme Oxygen xml editor[6]) et il existe des outils qui nous permettent de découvrir les relations entre les schémas. Ensuite, il existe des produits d'intégration qui permettent d'établir des mappings entre des schémas XML ainsi que des solutions serveur qui fournissent de grandes bibliothèques de schémas XML utilisés dans des standards d'entreprise très répandus. Ils nous permettent d'exécuter les mapping et de gérer les transformations de messages entre différents formats. Même si XML est disponible depuis 1998, il existe encore des zones de gestion de schéma XML qui restent non résolues ou qui sont encore en cours de recherche. L'un des domaines est la modélisation conceptuelle des schémas XML (Nečaskỳ *et al.* 2012) et l'évolution des schémas XML implémentés dans un outil de gestion de schéma XML eXolutio.(Klímek & Necaskỳ 2011) Le modèle conceptuel peut aider à la gouvernance de schéma XML lorsqu'il existe de nombreux schémas décrivant le même problème de domaine.

Dans ce cas, il est utile d'avoir un modèle conceptuel en place qui interconnecte les divers schémas XML et fournit une documentation visuelle et une vue d'ensemble des concepts utilisés dans les schémas XML. L'interconnexion est faite par le mapping des concepts présents dans les schémas XML aux concepts du modèle conceptuel. En outre, avoir le modèle conceptuel peut faciliter le processus de publication de données XML en tant que données liées. Nous pouvons mapper le modèle conceptuel à une ontologie, nous pouvons ensuite créer automatiquement des scripts XSLT[7] pour transformer les données XML valides contre nos schémas XML modélisés en RDF selon l'ontologie et revenir en arrière.

L'approche du modèle conceptuel n'est viable que pour un ensemble de schémas dans un problème de domaine commun. Cependant, dans les grandes entreprises, il existe souvent plusieurs problèmes de domaines dans lesquels les solutions XML sont déployées. Dans ce cas, nous sou-

6. https ://www.oxygenxml.com/
7. https ://www.w3.org/TR/xslt-30/

haitons avoir au moins une vue d'ensemble des schémas existants, des langages et des espaces de noms utilisés, des conventions de naming des éléments et des attributs, ainsi que des liens entre les schémas existants. A cet effet, un référentiel de schémas XML serait utile. Nous avons fait les premiers pas vers un tel référentiel et créé un référentiel de schémas XML ontology, qui est une approche de données liées pour décrire les schémas XML. Une partie de la description peut être générée à partir du schéma XML lui-même, une autre partie (comme le contact responsable du schéma) peut être ajoutée manuellement. L'avantage est que, comme avec d'autres données liées, les utilisateurs peuvent interroger le référentiel à l'aide de requêtes SPARQL standard, ils peuvent collaborer sur les outils d'utilisation de descriptions comme OntoWiki [8] et ils peuvent lier aux descriptions de schéma (ou élément ou attribut) d'autres données liées. En utilisant leurs URIs. Ceux-ci peuvent être des concepts de taxonomie, des discussions de développement ou de conception dans des wikis ou même des emails ou d'autres documents texte pouvant contenir des URIs comme de la documentation, des présentations, etc. interconnectés et intégrés avec d'autres sources d'informations et de données dans l'entreprise.

3.3 Intégration des Bases de données

Le développement exponentiel d'échanges d'informations par le Web a mis à jour les difficultés pour retrouver l'information pertinente désirée par un utilisateur final. En effet, les informations sont représentées et stockées dans une multitude de sources de données et cela de façon très hétérogène. Les premières approches d'intégration de ces sources de données pour les faire coopérer ont été réalisées dans le cadre de système de bases de données relationnelles, objets/relationnelles ou objets au travers de la mise en place d'une fédération de base de donnée. Cependant, dans le contexte du Web, les sources de données ont pour but principal de fournir des capacités d'interrogation et non d'exportation des données. Ce contexte nous oblige à repenser la façon dont nous devons intégrer ces différentes sources d'informations pour connaître au plus tôt la sémantique des données mémorisées. Connaître cette sémantique, c'est pouvoir « filtrer » les sources qui sont les plus adéquates pour répondre à une demande d'information. L'hétérogénéité des données est l'un des problèmes de longue date le plus populaire dans le domaine de la recherche sur les bases de données, qui reste, dans une large mesure, non résolu. L'hétérogénéité se produit entre deux ou plusieurs systèmes de bases de données lorsqu'ils utilisent une infrastructure logicielle ou matérielle

8. http ://ontowiki.net/

différente, suivent des conventions syntaxiques et des modèles de représentation différents, ou lorsqu'ils interprètent différemment des données identiques ou similaires. La résolution d'hétérogénéité permet d'intégrer plusieurs bases de données et d'interroger uniformément leur contenu. Dans les architectures d'intégration de base de données typiques, un ou plusieurs modèles conceptuels sont utilisés pour décrire le contenu de chaque base de données source, des requêtes sont posées par rapport à un schéma conceptuel global et, pour chaque base de données source, un wrapper est responsable de reformuler la requête et de récupérer les données appropriées .

L'intégration basée sur l'ontologie emploie des ontologies au lieu de schémas conceptuels et, par conséquent, les correspondances entre les bases de données source et une ou plusieurs ontologies doivent être définies. Ces correspondances consistent en des formules de mapping qui expriment les termes d'une base de données source comme une requête conjonctive contre l'ontologie (Local comme vue ou mapping LAV), expriment les termes d'ontologie comme une requête conjonctive contre la base de données source (Global comme vue ou mapping GAV), ou indiquer une équivalence de deux requêtes par rapport à la fois à la base de données source et à l'ontologie (Global Local as view ou mapping GLAV). Le type de mappings utilisé dans une architecture d'intégration influence à la fois la complexité du traitement des requêtes et l'extensibilité de l'ensemble du système (par exemple, dans le cas des mappings GAV, le traitement des requêtes est trivial, mais l'ajout d'une nouvelle base de données source nécessite une redéfinition de tous les mappings ; l'inverse est valable pour les mappings LAV). Ainsi, la découverte et la représentation de mappings entre schémas de bases de données relationnelles et ontologies constituent une partie intégrante d'un scénario d'intégration de base de données hétérogène.

3.3.1 Approche Données liées

En tant que technologie clé pour mettre en œuvre le Web sémantique, les données liées sont progressivement devenues une préoccupation académique et industrielle. Les données liées représentent une pratique des technologies sur le Web et des données de structure liées. L'objectif des données liées est de permettre aux utilisateurs de partager des données structurées sur le Web aussi facilement qu'ils peuvent partager des documents aujourd'hui. Sur le Web de données structurées avec des données liées, les utilisateurs peuvent passer d'un ensemble de données à un autre. Par rapport au Web de document qui permet de passer d'un document

à un autre, le Web de données fournit beaucoup plus de liens et possède une sémantique. Le mapping de bases de données relationnelles vers RDF est un problème fondamental pour le développement de données liées.

À l'origine, les systèmes de bases de données étaient considérés par la communauté du Web sémantique comme un excellent moyen pour un stockage efficace des ontologies, en raison de leurs avantages de performances connus et bien mis en évidence. Cette considération a conduit au développement et à la production de plusieurs systèmes de base de données, spécialement optimisés pour le stockage persistant, la maintenance et l'interrogation des données du web sémantique (Spanos *et al.* 2012). De tels systèmes sont connus sous le nom de triplestore, car ils sont spécialement conçus pour le stockage des déclarations RDF, également appelées triplets. Ce type de collaboration entre la base de données et le Web sémantique spécifie un flux de données et d'informations de ce dernier vers le premier, sous la forme d'une population de bases de données spécialisées, qui ont souvent une structure prédéfinie, avec des données.

Une ligne de recherche différente, peut-être plus intéressante, prend comme point de départ une base de données relationnelle existante et entièrement fonctionnelle et cherche des moyens d'extraire des informations et de les rendre utilisables du point de vue du Web sémantique. Dans ce cas, la motivation passe du stockage et de l'interrogation efficaces des structures ontologiques existantes à des problèmes, tels que l'intégration de bases de données, ontology Learning, l'accès aux données basées sur l'ontologie et l'annotation sémantique des pages Web dynamiques.

Le mapping des données relationnelles au modèle de données RDF adopte des techniques d'intégration de base de données relationnelles et les augmente. En utilisant un mapping des données relationnelles vers RDF, les données peuvent être intégrées dans un cloud de données interne ou externe. En utilisant des URI pour identifier les ressources, l'intégration avec des données non relationnelles et externes est facilitée. L'intégration de la base de données suivant le cycle de vie des données liées, concerne les étapes d'extraction et de stockage / interrogation. L'approche générale du mapping d'une base de données relationnelle en RDF utilisant le RDB to RDF Mapping Language (R2RML) par le groupe de travail W3C RDB2RDF est illustrée à la figure 3.2. Essentiellement, le standard R2RML décrit comment une base de données relationnelle peut être transformée en RDF par des moyens de term maps et triple maps (1). La base de connaissances RDF qui en résulte peut être matérialisée dans un

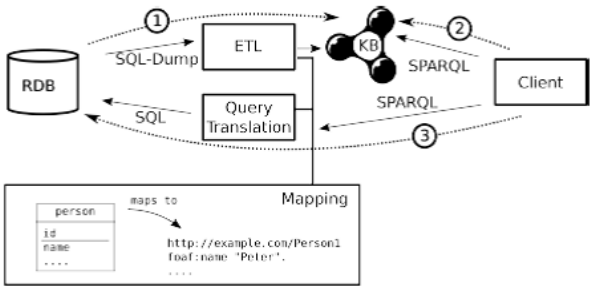

FIGURE 3.2 – *Intégration de base de données basée sur le mapping avec traduction de requête et Extract Transform Load (ETL)*
(Frischmuth *et al.* 2012)

triple store et ensuite interrogée en utilisant SPARQL (2). Afin d'éviter une étape de matérialisation coûteuse, les implémentations R2RML peuvent mapper dynamiquement une requête SPAQRL en entrée dans une requête SQL correspondante (3), ce qui donne exactement les mêmes résultats que la requête SPARQL en cours d'exécution sur le RDF dump matérialisé. R2RML définit comment les triplets sont créés à partir des relations, mais pour faciliter une intégration efficace des données d'entreprise, il est primordial de fournir des interfaces RDF et SPARQL au SGBDR. En évitant une matérialisation coûteuse des données relationnelles dans un triple store dédié, une intégration légère dans les architectures existantes est possible. Par conséquent, les wikis sémantiques, les outils de query federation et les outils d'interconnexion peuvent fonctionner avec les données des bases de données relationnelles.

L'utilisation du SPARQL 1.1 query federation[9] permet d'intégrer des bases de données relationnelles dans des systèmes de query federation avec des requêtes couvrant plusieurs bases de données. Cette fédération permet par exemple des portails en combinaison avec un EKB une vue intégrée sur les données d'entreprise.

3.3.2 Challenges

L'importance des bases de données du point de vue du Web sémantique est évidente à partir des multiples avantages et cas d'utilisation dans lesquels un mapping de bases de données vers RDF peut être utilisée qui signifie la transition d'un modèle de représentation à un autre. A cet égard, il est important d'identifier les différentes motivations et problèmes impliquant les interactions entre les bases de données relationnelles et les technologies logicielles, afin de réussir une séparation claire des objectifs

9. https ://www.w3.org/TR/sparql11-federated-query/

et des défis.

Deux défis sont identifiés. Tout d'abord, définissez un mapping qui peut servir de moyen automatique et par défaut pour traduire un schéma de base de données relationnelle en une ontologie OWL et des instances de bases de données relationnelles en RDF. Deuxièmement, définissez une méthode pour que les requêtes SPARQL soient efficacement évaluées par la base de données relationnelle. L'objectif est qu'une telle méthode utilise les optimisations déjà intégrées dans les bases de données relationnelles.

Les coûts de génération de données RDF est plus élevés afin que les requêtes SPARQL puissent être évaluées par le triple store. Cependant, il existe plusieurs cas où la matérialisation associée à l'approche ETL n'est pas adéquate ou possible, par exemple si la base de données relationnelle sous-jacente est fréquemment mise à jour. Dans ces cas, les données RDF sont mieux gardées virtuelles, et par conséquent les requêtes SPARQL sur elles doivent être traduites en SQL vers le système de base de données relationnelle sous-jacent étant donné que le processus de traduction doit prendre en compte les mappings R2RML spécifiés. Rajoutant à ça le problème de combler le gap entre les triples stores et les bases de données relationnelles est également présent dans les mappeurs SPARQL-to-SQL et stimule la recherche.

Un autre défi pour le mapping des données relationnelles dans RDF est le manque actuel de meilleures pratiques et supports d'outils pour la création de mapping. La standardisation du langage de mapping RDB vers RDF (R2RML) qui spécifie des règles pour transformer le contenu d'une base de données relationnelle en RDF. Toutefois, sans une formalisation et une optimisation appropriées, les requêtes SQL générées peuvent ne pas être correctes ni suffisamment efficaces pour être évaluées sur la base de données relationnelle sous-jacente. Ainsi, un manque d'outils matures pour la création et l'application des mappings R2RML. Le défi ici consiste à créer des interfaces conviviales et à établir les meilleures pratiques pour la création, l'intégration et la maintenance de ces mappings. Dernièrement, pour la lecture-écriture, les mises à jour d'intégration sur les données mappées doivent être propagées dans le SGBDR sous-jacent. Une première solution est présentée dans.(Eisenberg & Kanza 2012) Dans le contexte des données d'entreprise, une intégration avec des mécanismes de contrôle d'accès est d'une importance vitale.

3.3.3 Langages de Mapping

La majorité des données d'aujourd'hui sont stockées dans des bases de données relationnelles. Pour rendre ces données disponibles sur le Web sémantique, elles doivent être mappées en RDF. Le mapping des données relationnelles vers RDF peut se faire de plusieurs manières. Une approche simple consiste à utiliser une mapping directe qui mappe directement les données, sans aucune possibilité de personnaliser le RDF. Une autre approche consiste à utiliser un langage de mapping qui permet à l'utilisateur de spécifier la relation entre les deux modèles de données. Cela permet de personnaliser le RDF de sortie.

Les langues de mapping peuvent être classées en quatre catégories. Ce sont : les mappings directs, les mappings à usage général en lecture seule, les mappings à usage général en lecture-écriture et les mappings à usage spécial. Les langages de mapping à usage spécial sont conçus pour des scénarios d'utilisation spécifiques. Un exemple est le langage de mapping appelé Triplify (Auer *et al.* 2009), qui est une application légère pour publier des données liées à partir de bases de cit relationnelles.

Un mapping direct mappe la base de données relationnelle directement sur RDF, sans aucune interaction de l'utilisateur. Cela signifie que le RDF reflétera le modèle de données exacte des données relationnelles, plutôt que le domaine des données. Un mapping direct fonctionne généralement en mappant chaque table à une classe. Chaque ligne de la table sera mappée à un individu qui sera membre de la classe de la table. Chaque colonne sera mappée à une propriété. Les clés étrangères seront mappées avec une propriété qui lie un individu à un autre. Le co-domaine des autres propriétés sera littéral. Le mapping directe des données relationnelles n'est généralement pas ce qui est nécessaire. Il fournit plutôt une base pour définir et comparer des transformations plus complexes. Il a plusieurs inconvénients. Par exemple, les relations plusieurs-à-plusieurs dans une base de données relationnelle sont généralement modélisées en ayant une table qui lie les entités d'une table aux entités d'une autre table. Cela amène le mapping à créer une classe supplémentaire pour cette table, même si elle est facilement exprimée en RDF avec une propriété simple. Un autre inconvénient est que le mapping direct ne réutilisera pas le vocabulaire des ontologies existantes. Tous les URI sont générés en fonction du schéma et des données de la base de données.

L'exemple 3.1 (sur l'annexe) donne un aperçu du mapping directe. Comme on peut le voir, la structure du RDF généré reflète directement

la structure de la base de données. Un meilleur mapping utiliserait, par exemple, le vocabulaire FOAF pour décrire les personnes. Il peut également supprimer la propriété <Person #OwnsCar> et ne conserve que <Person # ref-OwnsCar>. La propriété ID des personnes peut également être supprimée, car elle était destinée à être utilisée uniquement dans la base de données relationnelle. Les ID ont été utilisés pour construire les URI des personnes.

Un langage de mapping à usage général est beaucoup plus utile qu'un mapping direct. Un langage comme celui-ci fonctionne en spécifiant un mapping qui décrit la relation entre la base de données relationnelle et le RDF. Un langage de mapping à usage général en lecture seule est généralement assez expressif. Cependant, cela peut rendre le language plus difficile à comprendre et à apprendre. L'expressivité élevée augmente également la complexité des mappings, ce qui rend difficile l'ajout de la prise en charge de l'écriture. Des exemples de langages de mappage à usage général en lecture seule incluent R2RML et D2RQ.

Les langages de mapping à usage général en lecture-écriture sont généralement moins expressifs que les mappings en lecture seule. Cela est principalement dû au problème de mise à jour de vue, qui rend difficile la mise à jour de la base de données basée sur une mise à jour d'une vue. La raison en est que plusieurs mises à jour différentes de la base de données peuvent entraîner la même modification de la vue. En SQL, cela est résolu en ajoutant des restrictions aux vues pouvant être mises à jour. Afin d'avoir un support général pour l'accès en écriture, il est donc nécessaire d'ajouter des restrictions au langage de mapping. R3M (Hert *et al.* 2011) est un exemple de langage de mapping à usage général en lecture-écriture.

R2RML Mapping Language

Définition

R2RML a été créé par le groupe de travail RDB2RDF du W3C. En octobre 2007, le W3C a formé un groupe d'incubateurs pour explorer le domaine de l'accès RDF aux bases de données relationnelles. R2RML (RDB to RDF Mapping Language) est un langage de mapping des bases de données relationnelles aux triplets RDF. Un mapping R2RML spécifie la relation entre la base de données et les triplets RDF. Le mapping se compose de triples map qui, à l'aide de ses sous-parties, peut spécifier les triplets RDF. R2RML offre plusieurs façons de spécifier des triplets RDF. Cela le rend utile dans de nombreuses situations différentes. Ceci est en contraste avec les direct mappings, ce qui ne permet aucun contrôle du graphe RDF

résultant.
R2RML permet à l'utilisateur de spécifier les mappings utilisés pour créer des vues RDF en lecture seule des données relationnelles. Les mappings eux-mêmes sont également sérialisés en tant que RDF. Il est indépendant de toute application pouvant utiliser les mappings. Cela signifie qu'une application qui interroge une vue RDF n'a pas besoin de se soucier que la vue RDF a été créée par un mapping R2RML. Pour l'application, l'interrogation d'une vue RDF créée par un mapping R2RML sera exactement la même que si les données étaient stockées directement en tant que RDF.

Mapping RDB to RDF avec R2RML
La Figure 3.3 illustre le processus de mapping d'une base de données relationnelle vers RDF à l'aide de R2RML. En commençant par un ensemble de tables de base de données, la table logique sélectionne l'une des tables (ou crée une vue) qui sera utilisée par le mapping. La Subject map spécifie le mapping entre les lignes de la table et les nœuds de ressources dans le graphe RDF. Dans ce cas, il mappe la deuxième ligne sur le nœud de ressource ex : Person # 2.

Predicate map mappe une colonne de la table à un prédicat dans le graphe RDF. Il peut y avoir plusieurs maps de prédicat afin de spécifier plus de prédicats. Dans ce cas, predicate map mappe les colonnes Name, Age et Car sur, respectivement, les prédicats foaf : name, foaf : age et ex : hasCar.

Object map mappe les valeurs de la base de données aux objects dans le graphe RDF. Un mapping d'objet de référence est utile pour mapper des relations de clé étrangères. Deux objects normales et un referencing object map sont utilisées dans la Figure 3.3. Les deux mappings d'objets normaux mappent les valeurs Ali et 38 aux valeurs littérales dans le graphe RDF. Referencing object map mappe la valeur GF42 au nœud de ressource ex : Car # GF42. Ce nœud de ressource doit avoir été spécifié par un subject map d'un autre mapping. Les objets qui sont connectés à quels prédicats sont déterminés par des mappings d'objets de prédicat.

Le modèle de données R2RML
Une mapping R2RML est composée de plusieurs parties. Comme on peut le voir sur la Figure 3.4, il y a une partie principale appelée triples map, qui contient une table logique, subject map et un certain nombre de predicate object maps. Cette section décrira toutes les différentes parties

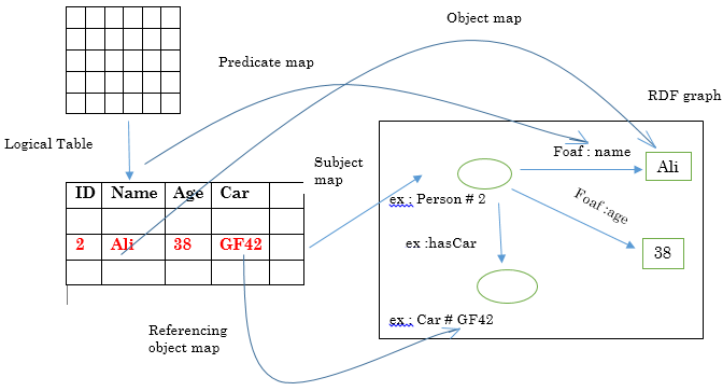

FIGURE 3.3 – *Le processus de mapping RDB to RDF*

du modèle de données R2RML.

Cartes de terme (Triples Map)

Le triple map est la partie principale d'un mapping R2RML. Triple map indique des triplets RDF correspondant à une table logique. Une table logique peut être, par exemple, une table dans une base de données. Le triples map contient également un subject map et un certain nombre de predicate object maps, qui spécifie comment les triplets devraient être. Il y a deux triples maps dans l'exemple (voir Annexe), ex : CarMap et ex : PersonMap. L'ex : CarMap mappe les voitures dans la base de données à RDF, tandis que ex : PersonMap mappe les personnes.

Table logique (Logical Table)

La table logique est la table SQL à partir de laquelle les triplets RDF seront générés. La table logique peut être une table SQL, une vue SQL ou une vue R2RML. Une vue R2RML est une requête SQL similaire à une vue SQL normale. Cependant, une vue R2RML permet au mapping de spécifier une vue sans avoir l'ajouter à la base de données actuelle. Une table logique a une requête SQL efficace qui produira le contenu de la table logique lorsqu'elle est exécutée sur la base de données d'entrée. La requête SQL efficace d'une vue R2RML est essentiellement sa requête SQL. Pour une table ou une vue SQL, c'est la requête :
SELECT * FROM table ;
Dans l'exemple, la table logique de ex : CarMap a le nom de la table Car. Cela signifie que la table logique d'ex : CarMap sera identique à la table Car dans la base de données. La table logique d'ex : CarMap est donnée par la requête SQL suivante :

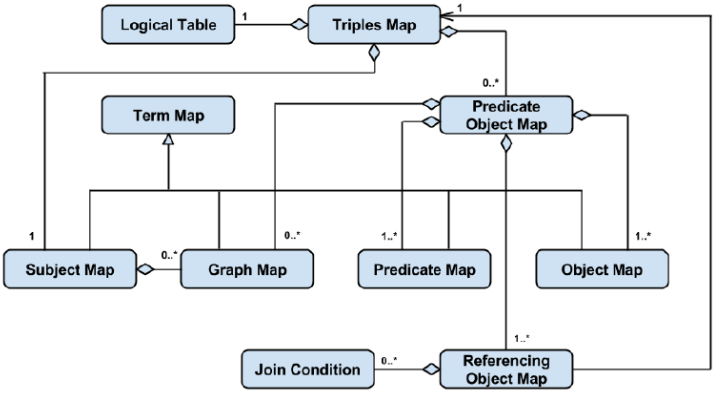

FIGURE 3.4 – *Une vue d'ensemble du modèle de données R2RML*

```
SELECT * FROM Car ;
```

Carte de prédicat objet (Predicate Object Map)

Predicate object map spécifie les prédicats et les objets des triplets RDF qui seront générés. Un predicate object map a au moins un predicate map et un object map. Un predicate map spécifie les prédicats tandis qu'un object map spécifie des objets. Un object map peut être un mapping d'objet normal, qui spécifie des objets directement en fonction des valeurs de la base de données ou un mapping d'objet de référence utilisé pour mapper des clés étrangères.

Dans l'exemple, ex : CarMap et ex : PersonMap contient deux predicate object maps chacun. Le premier predicate object map ex : CarMap (ligne 13 à 16) est utilisée pour ajouter des numéros d'immatriculation aux voitures.

Carte de terme (Term Map)

Term map spécifie les termes RDF de la table logique. Il existe quatre types de term maps, un pour chaque partie d'un triplet RDF : subject maps, predicate maps, object maps et graph maps. Chacun d'entre eux sont utilisés dans différentes parties du mapping.

Term map peut avoir une valeur constante, une valeur de colonne ou une valeur de modèle. Si elle est à valeur constante, tous les termes auront la même valeur spécifiée. Si elle est évaluée par une colonne, les valeurs de la colonne spécifiée dans la table logique seront utilisées pour créer les termes. Si le modèle est évalué, les termes seront spécifiés en fonction du modèle donné. Un modèle est une chaîne pouvant contenir plusieurs noms de colonnes. Un modèle est utile pour créer des URIs uniques en fonction des valeurs d'une ou de plusieurs colonnes. Par exemple, à la ligne 10 de l'exemple, le modèle suivant est spécifié :

http ://www.example.org/car/ RegNum

La création d'URIs à partir de ce modèle est effectuée en remplaçant RegNum par des valeurs de la colonne RegNum de la table logique.
Le term map peut également avoir un type de terme spécifié. Le type de terme détermine quel type de terme RDF il est. Les types de termes possibles sont les URIs, les nœuds vides et les littéraux. Le type de terme ne peut être défini que pour les term maps valables pour les colonnes et les modèles. Le type du terme RDF spécifié par un terme à valeur constante est déterminé directement par la valeur constante donnée. Par exemple, si la valeur constante est définie sur ex : hasReg (ligne 14 dans l'exemple 3.2), le type de terme sera défini sur URI.

Carte de sujet (Subject Map)
Subject map spécifie les termes sujets de la table logique. La subject map peut avoir une classe dont chaque sujet spécifié sera membre. Il peut également avoir un graph map. Le type de terme du subject map peut être soit une URI soit un nœud vide, car les littéraux sont illégaux dans la position du sujet d'un triplet RDF. Il est très fréquent qu'un subject map soit évalué par un modèle.
Dans l'exemple, ex : CarMap contient le subject map :

ex : CarMap
rr : subjectMap [
rr : template "http ://www.example.org/car/ RegNum" ;
rr : class ex : Car ;
]

Carte de prédicat (Predicate Map)
Predicate map spécifie les termes de prédicat pour le triplet RDF. Un predicate map peut avoir une valeur constante, une valeur de colonne ou une valeur de modèle comme des term maps habituels. Dans la plupart des cas, le predicate map est à valeur constante. Les prédicats générés à l'aide de predicate map seront associés à des objets par predicate object de prédicat. Le type de terme de predicate map doit être une URIs.
La ligne 14 de l'exemple donne un exemple de predicate map à valeur constante. Il est spécifié comme suit :

rr : predicate ex : hasReg ;

Ce predicate map a la valeur constante ex : hasReg. Notez que le prédicat rr : predicate est utilisé. C'est un prédicat de raccourci qui peut être utilisé

uniquement pour les predicate maps à valeurs constantes.

Carte d'objet (Object Map)

Object map spécifie les termes d'objet pour les triplets RDF. Le type de terme d'un object map peut être une URI, vide ou littéral. L'object map peut également avoir un type de données ou une balise de langue si le type de terme est littéral. Comme avec le predicate map, les objets générés par l'object map seront associés à des prédicats de predicate object map. Un exemple d'un object map peut être vu à la ligne 15 dans l'exemple.

rr : objectMap [rr : tableName "RegNum"];

Carte de graphe (Graph Map)

Graph map est un term map qui permet aux triplets RDF d'être stockés dans un graphe nommé spécifié. Un subject map et un predicate object map peuvent tous les deux avoir un graph map spécifiée. Si le graph map est spécifié dans le subject map, alors tous les triplets seront stockés dans ce graphe. S'il est spécifié dans predicate object map, alors tous les triplets générés seront stockés dans le graphe. Le type de terme d'un graph map doit être une URI. Comme d'habitude, un graph map peut être évalué par une colonne, par un modèle ou par une valeur constante. Si le graph map est de valeur constante, alors tous les triplets seront dans le même graphe. Des graphs maps à valeur de colonne ou par un modèle peuvent être utilisées pour stocker les triplets dans des graphes nommés par les valeurs de la base de données.
Une chose qui peut être faite avec un graph map dans l'exemple, est de stocker toutes les voitures du même modèle dans le même graphe. Cela peut être fait en ajoutant un graph map à valeur de modèle à un subject map d'ex : CarMap. Le graphe map peut ressembler à ceci :

[
rr : template "http ://www.example.org/graph/ CarModel"
]

Carte d'objet de référence (Referencing Object Map)

Referencing object map n'est pas un term map. Cependant, il peut être utilisé à la place d'un object map. Un referencing object map permet à des sujets d'un autre triples map d'être utilisés comme objets. Ceci est utile pour mapper, par exemple, des clés étrangères dans la base de données relationnelle. Le referencing object map à une référence à des tiples map appelés des triples map parents. Le subject map des triples map parents

est utilisée pour récupérer les objets. Le triple map dans lequel se trouve le referencing object map s'appelle child triples map.
Si la table logique des parents triples map est différente de la table logique du child triples map, les tables logiques doivent être jointes. Ceci est fait en utilisant les conditions de jointure. Une condition de jointure est spécifiée par deux colonnes. Un du child triples map et un de parent triples map. Plusieurs conditions de jointure peuvent être spécifiées afin de rejoindre plus d'une colonne.
Referencing object map a une requête SQL commune utilisée lors de la génération des objets. Dans cette requête, toutes les conditions de jointure existantes sont vérifiées. Le subject map des parents triples map est utilisée sur le résultat de cette requête afin de générer les objets du referencing object map. Dans cette requête les requêtes enfant et parent sont les requêtes SQL effectives des tables logiques, respectivement, le child et parent triples map.

D2RQ Mapping language

D2RQ est un langage pour mapper des bases de données relationnelles vers RDF. Il fait partie du système D2RQ, qui permet de visualiser les bases de données relationnelles sous forme de graphes RDF virtuels en lecture seule. Le projet D2RQ a été fondé par Christian Bizer, et est actuellement maintenu par Richard Cyganiak (Cyganiak *et al.* 2012). D2RQ visait à combler le vide entre les données ouvertes liées (RDF) et les bases de données relationnelles. Afin de rendre disponibles les données dans les bases de données relationnelles sous forme de données ouvertes liées.(Priyatna 2015)

Le langage D2RQ est utilisé pour faire des mapping qui décrivent la relation entre la base de données et le RDF. Le mapping D2RQ lui-même est écrit en RDF. Un mapping se compose de cartes de classe (class map) et de ponts de propriété (property bridges). Un class map représente une classe d'une ontologie OWL ou d'un schéma RDFS, et spécifie comment les instances de cette classe peuvent être identifiées dans la base de données. Une property bridge est utilisé pour ajouter des propriétés aux objets à partir d'une carte de classe. (Chen *et al.* 2013)

Dans un mapping de classes, la propriété d2rq : class spécifie l'URI de la classe. Les instances de la classe peuvent être spécifiées de plusieurs manières. La manière la plus normale consiste à utiliser un modèle d'URI qui est créé en insérant des valeurs de la base de données dans le modèle. Un modèle d'URI se compose d'un squelette d'URI et des noms de colonne d'une table dans la base de données. Si les valeurs de la base de données

sont déjà des URI valides, elles peuvent être utilisées directement à l'aide de la propriété d2rq : uriColumn. Il est également possible de générer des nœuds vides (Strandhaug 2014).

Les property bridges sont utilisés pour ajouter des propriétés aux objets d'une classe. Il a une propriété appelée d2rq : AppartToClassMap qui spécifie à quelle classe la propriété est destinée. Les instances de la classe spécifiée seront les sujets des triplets générés. La propriété d2rq : property spécifie la propriété qui sera utilisée. Il est également possible d'utiliser d2rq : dynamicProperty pour générer l'URI de la propriété au moment de l'exécution (à l'aide d'un modèle). Si plusieurs propriétés sont spécifiées, un triple pour chaque propriété sera généré.

La propriété d2rq : referToClassMap permet d'utiliser les sujets d'une autre class map, en tant qu'objets. Ceci est utile pour exprimer les clés étrangères dans la base de données. Si les mappings de classes obtiennent leurs valeurs à partir de différentes tables, une jointure doit être spécifiée (d2rq : join). Une jointure est spécifiée par deux colonnes dans des tables différentes qui doivent être égales (pour un exemple, voir la ligne 20 dans l'annexe).

L'exemple 3.4 du mapping D2RQ, mappe une partie de la base de données illustrée dans l'exemple 3.1, à RDF. Il contient les class maps ex : CarMap et ex : PersonMap. Les instances de ces classes proviennent respectivement des tables Car et Person. Chaque instance aura soit le type ex : Car, soit ex : Person. Les URI des instances sont générés par un modèle d'URI. Par exemple, une personne dont l'ID est défini sur 1 obtiendra l'URI : http : //www.example.org/ID=1. Le pont de propriété ex : CarOwner relie les personnes à leurs voitures. Dans la base de données, la table Person possède une clé étrangère vers la table Car. Il est donc possible d'utiliser la propriété d2rq : referToClassMap pour utiliser des instances de la carte de classe ex : CarMap comme objets dans les triplets générés. Pour ce faire, nous avons également besoin d'une condition de jointure. La condition de jointure indique que la colonne OwnsCar de la table Person doit être égale à la colonne RegNum de la table Car.

3.4 Conclusion

Dans ce chapitre, nous avons identifié plusieurs défis d'intégration de données qui se posent dans l'environnement de l'entreprise et plus largement dans le web sémantique. En particulier, nous avons examiné deux domaines très essentiels pour l'intégration, à savoir les taxonomies d'entreprise, XML schema governance, et nous avons discuté de l'utilisation des technologies de données liées. Les expériences, et le déploiement des

approches de données liées dans les scénarios d'entreprise a un énorme potentiel et peut entraîner des avantages considérables. En outre, nous considérons que l'intégration de la base de données est un sous-problème des défis de l'intégration des données dans les entreprises. Une couche de données liées au-dessus des bases de données, qui implique l'extraction des étapes du cycle de vie des données liées (stockage RDF), l'interrogation et la liaison, peut faciliter ces défis. Afin d'établir les données liées en tant que stratégie d'intégration des données d'entreprise.

L'utilisation du paradigme de données liées pour l'intégration des bases de données relationnelles nécessite La transformation de la base de données en un modèle de description universel, qui est le RDF, cette transformation est appelé mapping dans lequel le contenu d'une base de données relationnelle est transformé en RDF selon des règles spécifiques aux langages du mapping utilisé. Il existe plusieurs façons d'accéder aux données RDF mappées. Un moyen consiste à matérialiser les données et à les stocker, par exemple, dans un triple store. Cela peut être utilisé pour stocker toute de la base de données en RDF, mais ce n'est pas très pratique si le RDF doit être mis à jour avec la base de données. Ce sera également peu pratique si la base de données est très volumineuse. Ce qui est souvent fait à la place, c'est de créer uniquement une vue RDF des données relationnelles accessibles via un point de terminaison SPARQL. Les requêtes SPARQL peuvent être traduites en SQL, à l'aide des mappings. Les données sont stockées telles quelles dans la base de données. Cela signifie que la base de données peut être mise à jour normalement, car les mappings sont évalués au moment de la requête.

Nous avons met l'accent sur deux langages de mapping les plus populaires. Nous parlons ici du R2RML (Relational databases to RDF Mapping Language) un langage recommandé par le W3C pour exprimer des mappings personnalisés de la base de données vers des ensembles de données RDF. Et le D2RQ qui est utilisé pour faire des mapping qui décrivent la relation entre la base de données et le RDF.

Dans le chapitre suivant, nous discutons de façon plus détaillée de l'état de l'art de l'intégration des bases de données, les approches de mapping RDB-to-RDF et les approches de la réécriture SPARQL-to-SQL.

Etat de l'art 4

4.1 Introduction

Ce chapitre est consacré à une étude bibliographique qui concerne étroitement notre travail de recherche. Puisque le contexte de travail s'intéresse à l'intégration des bases de données relationnelles dans le web sémantique, l'état de l'art présenté dans ce manuscrit va aborder les approches actuelles et les outils de mapping des bases de données relationnelles en RDF. Qu'ils soient académiques ou industriels. En particulier, nous prenons en compte les implémentations R2RML récemment apparues. Nous mettons un accent particulier sur les spécificités de chaque technique, et nous décrivons, autant que possible, les capacités des différents outils étudiés. En tenant compte des préoccupations pratiques, et des projets qui ont fourni ces outils opérationnels pour mettre en œuvre le processus RDB-to-RDF. Dans ce chapitre, nous présentons les approches les plus populaires qui peuvent être classés en deux types principaux : Approches de matérialisation et approches de réécriture.(Lehireche *et al.* 2017)

La combinaison d'une base de données relationnelle, une ontologie OWL et de mappings, afin de répondre aux requêtes SPARQL sur ces trois composants, a récemment pris le nom d'Ontology-Based Data Access (OBDA). En règle générale, les chercheurs ont adopté l'une des deux approches pour développer des systèmes OBDA : une approche basée sur la réécriture ou une approche basée sur la matérialisation.

Dans l'approche basée sur la matérialisation, les entrées sont la base de données relationnelle, une ontologie OWL et un mapping. Ces derniers sont utilisés pour générer de nouvelles données sous forme de RDF qui sont ensuite stockés dans un triplestore. Les données RDF dans le Triplestore sont considérées comme la matérialisation des données de la base de données relationnelle compte tenu du mapping et de l'ontologie. La réponse à une requête SPARQL sur l'ontologie cible est calculée en interrogeant directement le Triplestore. Cependant, ce processus de matérialisation peut être coûteux, en particulier lorsque les sources de données sont fréquemment mises à jour.

L'approche basée sur la réécriture a été le principal objectif de la communauté de recherche ces dernières années. Trois étapes sont exécutées, étant donné une requête SPARQL et une ontologie OWL, une nouvelle requête SPARQL est générée qui contient les connaissances de l'ontologie. La nouvelle requête SPARQL est considérée comme la requête réécrite. La majorité de la littérature OBDA se concentre sur cette étape. Dans la deuxième étape, le mapping est utilisé pour compiler la requête SPARQL réécrite en une requête SQL qui peut ensuite être évaluée sur la base de

données. Peu de recherches ont été effectuées sur cette étape. Enfin, la requête SQL est évaluée sur la base de données relationnelle, qui donne la réponse à la requête SPARQL initiale. L'avantage de l'approche de réécriture est que les résultats de la requête reflètent les dernières données à jour.

Le but dans ce chapitre, est de présenter un état de l'art sur les approches de mapping RDB-to-RDF, qui permettent de répondre aux requêtes SPARQL sur des données relationnelles. Dans la section 4.2, nous présentons les approches qui fournissent un triplestore qui sera interrogé avec le SPARQL. Dans la section 4.3, nous nous intéressons à l'approche OBDA de réécriture qui expose un SPARQL endpoint virtuel pour interroger des données relationnelles à l'aide du mapping et une ontologie, Nous discutons plus en détail dans la section 4.3.9 des avantages et inconvénients de chaque approche, ainsi nous faisons une comparaison entre les systèmes dans un tableau. Cette analyse nous permet, en conclusion, d'identifier les insuffisances des approches existantes.

4.2 Approches de matérialisation RDB-to-RDF (Triplestore)

Face à la grande variété d'initiatives de mapping RDB-to-RDF, plusieurs études ont été menées pour comparer les approches et les techniques. En 2007, le W3C a créé le groupe de travail RDB2RDF pour standardiser les langages de mapping des schémas de bases de données relationnelles en RDF et OWL. Sahoo et al(Sahoo *et al.* 2009) ont mené une vaste étude, portant sur des articles théoriques, des preuves de concept, des projets spécifiques à un domaine ainsi que des outils de mapping génériques. Le but de cette étude n'était pas d'entrer dans les détails de chaque approche, mais de fournir au groupe de travail RDB2RDF un aperçu complet des différentes approches qui avaient été étudiées jusqu'à présent, afin de servir de base à la définition de R2RML.

Hert et al (Hert *et al.* 2011) ont proposé un cadre de comparaison basé sur les fonctionnalités qu'ils ont appliqué à l'état de l'art des langages de mapping. Il est dérivé des cas d'utilisation décrits par le groupe de travail W3C RDB2RDF (Sahoo *et al.* 2009). Les langages de mapping sont classés en quatre catégories : direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, specialpurpose mapping.

En 2010, Spanos et al (Spanos *et al.* 2012) a rédigé une revue complète des méthodes RDB-to-RDF, avec un effort particulier pour les classer dans des catégories disjointes : création d'une ontologie de schéma de base de données, création d'une ontologie spécifique à un domaine utilisant soit

la rétroingéniere de modèle de base de données, soit des connaissances de base de données d'experts , et définition ou découverte de mappings entre une base de données relationnelle et une ontologie existante. Ensuite, les fonctionnalités sont explorées dans chaque catégorie, telles que l'accessibilité des données ou le langage d'ontologie utilisé.

Sequeda et al (Sequeda *et al.* 2011) ont étudié les méthodes qui appliquent les principes du mapping directe pour traduire automatiquement une base de données relationnelle en RDF. Ils ont étudié les méthodes existantes pour extraire les connaissances ontologiques sous forme de données RDFS ou OWL. Cela va des approches simples (table à classe, colonne à propriété) à des approches plus avancées qui tentent de découvrir des relations telles que many-to-many, la subsomption, les relations symétriques et transitives et des fonctionnalités SQL telles que les contraintes d'intégrité. Enfin, les auteurs étendent les travaux de Tirmizi et al (Tirmizi *et al.* 2008), qui exploite toutes les combinaisons possibles de clés primaires et étrangères dans les tables relationnelles. L'une de leurs conclusions est que la qualité d'une ontologie résultante d'un mapping direct dépend fortement de la richesse du schéma SQL par rapport à la sémantique des domaines.
Ces études académiques examinés précédemment explorent des approches spécifiques pour mapper RDB à RDF avec un prototype ou une mise en œuvre de preuve de concept. Ces travaux ont conduit vers les approches industrielles et à la publication des outils / des application à la communauté comme :

4.2.1 Virtuoso Universal Server

Virtuoso Universal Server [1] est un ensemble d'outils complets commerciaux et open source développée par OpenLink, conçue pour répondre aux besoins de gestion, d'accès et d'intégration des données d'entreprise. Il est livré avec des fonctionnalités de classe de production telles qu'une base de données relationnelle, une réplication de données, un triplestore RDF, des capacités de raisonnement, l'intégration de plusieurs sources de données (SQL, RDF, XML, CMS, flux d'agrégation ...). Virtuoso Open-Source Edition22 est un sous-ensemble de Universal Server. L'édition open source de la suite d'outils RDF et SPARQL fournit des fonctionnalités telles que : Base de données relationnelle objet pour SQL, XML, RDF et texte libre ; RDF store et SPARQL endpoint ; serveur d'applications Web. En particulier, la fonctionnalité Vues RDF des données SQL implémente la fonctionnalité RDB vers RDF. Cette édition prend uniquement en charge la base de données relationnelle basée sur Virtuoso tandis que l'édition commer-

1. https ://virtuoso.openlinksw.com/

ciale prend en charge la plupart des systèmes relationnels bien connus. (Erling & Mikhailov 2010)

La matérialisation des données est possible mais ce n'est pas le but de Virtuoso. La méthode de récupération des données liées est également disponible et expose toutes les données hébergées par Virtuoso (sources de données SQL et XML) sous forme d'URI déréférencable. Les données RDF peuvent être récupérées sous forme de syntaxes RDF / XML, JSON ou N3.

Du point de vue de la réponse aux requêtes SPARQL, Virtuoso est principalement utilisé comme un triplestore. Il prend en charge SPARQL 1.1 et, dans ce mode, il offre des capacités de chaînage-arrière (par défaut) et chaînage-avant pour des sous-ensembles limités de RDFS et OWL. Lorsque Virtuoso est utilisé comme un SGBD standard, il peut être transformé en un système OBDA en configurant des mappings dans son propre langage de mapping. Cependant, son mode OBDA a plusieurs limitations : aucune capacité de raisonnement n'est disponible et seul un petit fragment de R2RML est pris en charge. Virtuoso est accessible via les API Sesame et Jena.

4.2.2 Oracle Database 12c

Oracle Spatial and Graph est une option d'Oracle Database Enterprise Edition (EE) [2]. La version 12c, publiée en juillet 2013, est fournie avec les fonctionnalités de gestion et d'analyse des graphes de données sémantique RDF, qui prennent en charge la traduction RDB-en-RDF.

Le graphe sémantique RDF se concentre principalement sur le stockage, l'interrogation simultanée et le raisonnement sur des données relationnelles et RDF. La base de graphes RDF peut évoluer jusqu'à des milliards de triplets, prend en charge la gestion des versions de graphes et l'indexation sémantique des documents. Les modèles de graphes SPARQL peuvent être inclus dans une requête SQL afin de rejoindre RDF et des données relationnelles. La traduction RDB-RDF est prise en charge depuis la version 12c en fournissant des vues RDF sur les tables relationnelles, les vues SQL et les résultats de requêtes SQL. R2RML et le mapping direct du W3C sont pris en charge. La vue RDF peut être interrogée via SPARQL 1.1 [Perry, et al. 2015].

4.2.3 Stardog

Stardog [3] est un triplestore commercial développé par Complexible Inc16. Il prend en charge SPARQL 1.1 et plusieurs niveaux de raison-

2. https ://www.oracle.com/database/12c-database/
3. https ://www.stardog.com/

nement. Stardog évite une matérialisation rapide et son moteur de raisonnement est en partie basé sur la réécriture des requêtes (en fait, le niveau de raisonnement peut être choisi par l'utilisateur au moment de la requête). Stardog est accessible via l'API Sesame. Depuis la version 4 publiée en novembre 2015, Stardog a intégré le code Ontop pour prendre en charge les requêtes SPARQL sur les graphes RDF virtuels. Par conséquent, il peut désormais être également classé comme système OBDA. (Taelman *et al.* 2019)

4.2.4 RDFox

RDFox [4] est un triplestore en mémoire développé à l'Université d'Oxford. Il implémente un nouvel algorithme de raisonnement Datalog parallèle en mémoire partagée et prend en charge le raisonnement OWL2 RL par matérialisation (Motik *et al.* 2014). Le système est un logiciel multiplateforme écrit en C ++ et est livré avec un wrapper Java prenant en charge l'API OWL.

4.2.5 RDF-RDB2RDF

RDF-RDB2RDF Perl RDF [5] est une bibliothèque de logiciels open source livrée sous licence GPL. Il s'agit d'une bibliothèque très complète comprenant les fonctionnalités suivantes : RDF store en mémoire, prise en charge des bases de données relationnelles (MySQL, PostgreSQL) et SQLite, processeur de requête SPARQL 1.1 endpoint, serveur de données liées, analyseur RDFa, WebID, HTML5.

RDF-RDB2RDF [6] est une bibliothèque supplémentaire, développée audessus de Perl RDF, qui implémente R2RML et le direct mapping, il implémente la méthode de matérialisation des données. Les données RDF matérialisées peuvent être chargées dans le SPARQL endpoint fourni par Perl RDF, tombant ainsi dans l'approche ETL.

4.2.6 Triplify

Triplify [7] Le but de Triplify est de permettre aux applications Web populaires (comme les applications de blog) de publier le contenu de leur base de données relationnelle au format données liées. Étant donné que de nombreuses instances de ces applications Web sont déployées sur Internet, les aider à exposer rapidement leur base de données devrait se traduire par l'adoption du Web sémantique (Auer *et al.* 2009).

4. https ://www.oxfordsemantic.tech/product
5. http ://librdf.org/docs/perl.html
6. https ://metacpan.org/pod/RDF : :RDB2RDF
7. http ://aksw.org/Projects/Triplify.html

Triplify est une approche simple mais efficace à utiliser comme plug-in léger et facile à apprendre pour les applications Web existantes. Il est basé sur le mapping des requêtes URI HTTP vers la base de données relationnelle. Par rapport à l'approche du mapping direct, Triplify prend le problème dans l'autre sens : il se concentre d'abord sur les données relationnelles qui devraient être nécessaires, au lieu de traduire l'ensemble du schéma relationnel en une ontologie ad hoc. Les mappings sont implémentés comme des instructions SQL intégrées dans des scripts PHP, par conséquent, toute construction SQL ou fonction d'agrégation peut être utilisée. Les fonctions de transformation écrites en PHP peuvent à leur tour être appliquées aux données renvoyées par les requêtes SQL. Les auteurs soutiennent que Triplify facilite la création de moteurs de recherche personnalisés ciblant certaines niches, par exemple la recherche de contenu spécifique dans divers blogs, wikis ou forums".

Triplify fournit les implémentations à la demande et de matérialisation des données. En mode à la demande, les données sont récupérées en tant que données liées RDF ou JSON (les deux via HTTP GET), mais aucun processus de réécriture de requête n'est pris en charge.

4.2.7 Discussion

La section précédente montre une diversité d'approches parmi les outils conçus pour traduire des bases de données relationnelles en RDF. Dans l'ensemble, bien que les différentes approches soient motivées par diverses motivations, il n'existe actuellement aucune méthode standard de représentation des mappings entre RDB et RDF. Les projets examinés dans ce chapitre utilisaient divers formats de représentation et langages spécifiques aux outils (par exemple, langage de mapping D2RQ ou R2RML). Les mappings, en particulier sont créés par des experts de domaine ou utilisent des ontologies de domaine, qui devraient être disponibles pour la réutilisation.

Le résultat de l'approche de matérialisation est les triplestores qui fournissent un modèle logique générique flexible pour stocker n'importe quel ensemble de triplets RDF. Cependant, si les triplets sont générés à partir de sources externes (par exemple, des bases de données relationnelles), un processus ETL intermédiaire (extraction, transformation et chargement) est requis pour transférer les données entre ces sources externes et le triplestore. Le processus ETL peut être coûteux, en particulier lorsque les sources de données sont fréquemment mises à jour.

Certains travaux (Chebotko *et al.* 2009) se sont concentrés sur l'utilisation de bases de données relationnelles comme base pour les triplestores, avec l'idée d'exploiter les performances fournies par les systèmes de bases

de données relationnelles, et ne prennent pas en charge les langages de mapping tels que R2RML. Récemment, la plupart des approches nécessitent un mapping défini par l'utilisateur pour permettre la création d'un SPARQL endpoint virtuel sur n'importe quelle structure relationnelle. Le langage standard pour définir un tel mapping est R2RML.

Plusieurs triplestores RDF, tels que Virtuoso, utilisent une seule table pour stocker les triplets. Cette approche a l'avantage d'être intuitive, flexible et les mappings entre la couche conceptuelle et la couche de données (si nécessaire) sont pertinents. D'autre part, une telle approche ne peut pas utiliser les optimisations connues développées pour la base de données relationnelle normalisée - dont beaucoup sont actuellement utilisées dans -ontop-(section 4.3.6).

TABLE 4.1 – *Approches de mapping RDB-RDF*

System	Mapping	Algorithm	Optimizations	Query Implementation
Virtuoso	Native, R2RML	X	X	SPARQL → *RDF*
Oracle Database 12c	R2RML	X	X	SPARQL → *RDF*
Stardog	Native,R2RML	X	X	SPARQL → *RDF*
RDFox	X	X	X	SPARQL → *RDF*
RDF-RDB2RDF	R2RML	X	X	SPARQL → *RDF*
Triplify	Native,R2RML	X	X	SPARQL → *RDF*

Avec cette approche, il n'y a pas de surcharge lorsqu'une requête SPARQL est évaluée sur le Triplestore par rapport à l'approche de réécriture. Cependant, un inconvénient est qu'une copie des données relationnelles d'origine doit être conservée. Si des mises à jour se produisent sur les données relationnelles d'origine, elles doivent être propagées vers la version RDF. En outre, en fonction de l'ontologie, la taille des données RDF matérialisées peut être infinie.

4.3 Approches OBDA de réécriture SPARQL –to- SQL

Ontology-based Data Access (OBDA) vise à faciliter l'accès au contenu de la base de données en conservant la sémantique entre les besoins d'information (ce que les utilisateurs veulent savoir) et leur formulation sous

forme de requêtes exécutables (généralement en SQL). Cette approche cache la complexité de la structure de la base de données aux utilisateurs en leur fournissant une représentation de haut niveau des données sous forme de graphe RDF.
Le graphe RDF peut être considéré comme une vue de la base de données définie par un mapping RDB-RDF (par exemple, suivant la spécification R2RML) et enrichi au moyen d'une ontologie. Les utilisateurs peuvent ensuite formuler leurs besoins d'informations directement sous forme de requêtes SPARQL sur le graphe RDF. Nous nous concentrons sur l'approche OBDA, où le graphe RDF n'est pas matérialisé (et s'appelle un graphe RDF virtuel), et la base de données est relationnelle et prend en charge le SQL. De nombreuses techniques et outils ont été proposés ces dernières années pour permettre la publication de données relationnelles sur le web en RDF qui sont :

4.3.1 D2R Server

D2R Server [8] est l'un des pionniers des systèmes OBDA, développé à l'Université de Berlin et DERI. Est un projet académique open source. Il fournit un environnement intégré avec plusieurs options pour accéder aux données relationnelles en utilisant différentes méthodes telles que le SPARQL endpoint, les données liées, RDF dump. (Bizer & Cyganiak 2006)

Ce système de réécriture de requêtes implémente certaines optimisations de requêtes mais celles-ci ont souvent été signalées comme insuffisantes : par exemple, les requêtes SQL générées peuvent contenir un nombre excessif de jointures (Priyatna 2015). Il fournit son propre langage de mapping, D2RQ, Le langage de mapping déclaratif DR2Q (Cyganiak *et al.* 2012) est formellement défini par un schéma RDFS. Il est le successeur du langage D2R MAP basé sur XML. Les mappings sont exprimés en RDF, mais reposent également largement sur des fragments SQL pour exprimer des instructions SELECT ou pour utiliser des fonctions d'agrégation. Le D2RQ direct mapping crée automatiquement une ontologie locale. Ce mapping direct peut être personnalisé manuellement. Il ne prend en charge qu'un fragment de R2RML. Aucun mécanisme d'inférence n'est inclus.

Les performances varient en fonction de la méthode d'accès et sont censées fonctionner raisonnablement bien pour les triples patterns de base, mais il existe des limitations pour les graphs patterns avec des filtres et des modificateurs de séquence de solution. Ce logiciel est disponible sous licence open-source.

8. http ://d2rq.org/

4.3.2 Mastro

Mastro [9] est un système OBDA qui partage des points communs avec Ontop. Ce système de réécriture de requêtes prend en charge le raisonnement sur les ontologies. Contrairement aux autres systèmes OBDA. Il ne prend en charge qu'un fragment restreint de SPARQL qui correspond aux requêtes conjonctives. Mastro est disponible uniquement à des fins de démonstration, de test et d'évaluation. (Calvanese *et al.* 2011)

4.3.3 Ultrawrap

Ultrawrap [10] est un système OBDA, initialement développé par l'Université du Texas à Austin, Ultrawrap est commercialisé par Capsenta, fondée en 2011 en tant que spin-off de l'Université du Texas. Il est basé sur des vues SQL pour présenter les données relationnelles sous forme de triplets RDF. Il a été récemment étendu pour soutenir l'inférence sur une extension de RDFS avec des propriétés inverses et transitives (Sequeda *et al.* 2014). Ultrawrap utilise un analogue des T-mapping de Ontop, appelés mappings saturés et utilisés pour créer des vues régulières et matérialisées dans la base de données relationnelle.

Une ontologie locale est définie par le mapping direct. La représentation RDF est implémentée comme une vue SQL à trois colonnes (sujet, prédicat, objet), définie comme l'union de toutes les requêtes qui matérialisent tous les triplets RDF tels que définis par l'ontologie locale .(Sequeda *et al.* 2009) Par conséquent, une requête SPARQL peut être simplement réécrite dans une requête SQL sur la vue SQL.

Dans (Sequeda & Miranker 2013), les auteurs expliquent la nécessité de deux colonnes supplémentaires dans les vues SQL : les clés primaires du sujet et de l'objet. Cette modification permet à l'optimiseur de requête natif de tirer parti des index existants. Avec deux autres optimisations (l'élimination de l'auto-jointure et la détection de conditions non satisfaisantes), les auteurs montrent que le temps d'évaluation des requêtes est comparable à celui des requêtes SQL écrites directement pour la représentation relationnelle. Ultrawrap prend également en charge la méthode de récupération des données liées (requêtes HTTP GET). Récemment, la prise en charge des langages de mapping R2RML et D2RQ a été ajoutée. Il n'y a cependant aucune description de la façon dont la description de mapping est dérivée dans la vue SQL. Nous faisons l'hypothèse qu'un document R2RML / D2RQ est compilé dans une vue SQL qui reflète chaque triple map. La prise en charge des graphes nommés R2RML ne semble pas facile en utilisant uniquement la triple vue SQL. Une interface graphique

9. https ://www.obdasystems.com/mastro
10. http ://www.cs.utexas.edu/ miranker/studentWeb/UltrawrapHomePage2.html

qui fait partie de la suite d'outils permet d'aligner l'ontologie locale avec une ontologie de domaine.

4.3.4 Morph-RDB

Morph-RDB [11] est une implémentation de R2RML, développée par les développeurs de R2O et ODEMapster. Il prend en charge la matérialisation des données (appelé mode de mise à niveau des données) et le On-demand mapping à l'aide de requêtes SPARQL. Dans (Priyatna *et al.* 2014), les auteurs décrivent la méthode qu'ils ont développée pour activer la traduction de requête SPARQL en SQL basée sur R2RML.

Morph-RDB est développé en Scala et Java et est disponible sous la licence open source Apache 2.0. Le développement est en cours. Les auteurs travaillent également sur deux autres versions de Morph, qui s'appuient sur R2RML bien qu'elles s'adressent à des bases de données non relationnelles : Morph-streams traite des données diffusées généralement produites par des capteurs, et Morph-GFT interroge Google Fusion Tables comme des tables relationnelles.

Ce système met en œuvre un certain nombre de techniques d'optimisation des requêtes telles que l'élimination de l'auto-jointure. Cependant, il n'a aucune capacité d'inférence. Et les développeurs ont l'intention de continuer le support et l'évolution de Morph-RDB, en l'étendant éventuellement audelà de la spécification R2RML officielle, par ex. en ajoutant la prise en charge de Google Fusion Tables.

4.3.5 SparqlMap

SparqlMap [12] est un SPARQL-to-SQL basé sur les spécifications du groupe de travail W3C R2RML. La raison est d'exécuter la requête SPARQL sur les bases de données relationnelles existantes en traduisant la requête SPARQL en exactement une requête SQL correspondante basée sur le mapping R2RML. (Unbehauen *et al.* 2013) Grâce à une large gamme d'optimisations de traduction de requêtes SPARQL vers SQL, SparqlMap est en mesure d'atteindre des performances proches du cas SQL natif. La bibliothèque d'optimisations comprend : filter Optimization, self–join, optional–self–join and self–union elimination aussi bien que le filter et projection pushing. L'évaluation approfondie du système démontre l'impact de ces optimisations et montre que SparqlMap surpasse les outils de mapping RDB2RDF.
Avec l'extension Update du système SparqlMap (Unbehauen & Martin 2017),

11. https ://www.oeg-upm.net/index.php/en/technologies/334-morph/
12. http ://aksw.org/Projects/SparqlMap.html

les auteurs fournissent un accès en lecture / écriture à des sources de données structurées pour permettre une intégration plus étroite des systèmes sources dans le processus d'amélioration des connaissances. Et ont étendu SparqlMap avec un SPARQL endpoint qui exploite l'intégration des systèmes hérités dans le cycle de vie des données liées. Cette intégration bidirectionnelle permet en fin de compte au système hérité de bénéficier des avantages des processus de gestion des connaissances Linked Data. Le language de mapping exprimés est le R2RML, pour les instructions SPARQL de mise à jour vers le langage de manipulation de données SQL (DML). La mise à jour de SparqlMap s'adapte bien à l'augmentation de la taille des inserts et que d'autres optimisations ne sont pas nécessaires.

4.3.6 Ontop

Ontop [13] est un système open source d'accès aux données à base d'ontologie (OBDA) qui permet d'interroger les sources de données relationnelles via une représentation conceptuelle du domaine d'intérêt, fournie en termes d'ontologie, à laquelle les sources de données sont mappées. Les principales caractéristiques d'Ontop sont ses fondements théoriques solides, une approche virtuelle de l'OBDA, qui évite de matérialiser des triplets, et elle implémenté par la réécriture de requêtes. De nombreuses optimisations exploitent tous les éléments de l'architecture OBDA, Ontop est connu par sa conformité à toutes les recommandations pertinentes du W3C (y compris les requêtes SPARQL, Mappings R2RML et ontologies OWL et RDFS), et sa prise en charge pour toutes les principales bases de données relationnelles. (Calvanese *et al.* 2017)

La première version d'Ontop (2010), s'appelait Quest qui ne fait désormais référence qu'au moteur de traitement des requêtes. Quest peut fonctionner en (i) le mode virtuel, qui prend en charge les graphes RDF virtuels via des mappings, et (ii) le mode triplestore, qui stocke les triplets RDF directement dans une base de données relationnelle. Ensuite, l'algorithme de réécriture de requête a remplacé PerfectRef pour réduire considérablement la taille des réécritures et tirer parti des T-mappings et de l'index sémantique (Rodriguez-Muro 2010).

Les performances d'Ontop dépendent plus de la complexité de la combinaison de l'ontologie et des mappings que de la taille de l'ensemble de données. D'un autre côté, Ontop peut effectuer efficacement des inférences ontologiques en mode graphe RDF virtuel sans besoin de matérialisation.

Ontop a été adopté dans plusieurs cas d'utilisation universitaires et industriels. Par exemple la plateforme Optique. Ontop est le cœur cette plateforme et prend en charge le module de transformation des requêtes.

13. https ://ontop-vkg.org/

La plate-forme peut exploiter le parallélisme massif dans le back-end dans la mesure du possible, et elle prend également en charge le streaming de données (dans le scénario de streaming, Ontop est utilisé uniquement pour la réécriture de requêtes). (Giese *et al.* 2015)

4.3.7 EVI

les auteurs proposent un algorithme pour interroger les données RDF stockées dans une base de données relationnelle avec un mapping défini par l'utilisateur. Pour générer des requêtes SQL qui peuvent être exécutées efficacement sur les moteurs relationnels. Par rapport aux approches existantes, ils ne reposent pas uniquement sur les optimisations de la requête relationnelle, mais d'abord sur la requête SPARQL. Le mécanisme permet de créer un virtual SPARQL endpoint sur une base de données relationnelle. Il n'est pas nécessaire de modifier la base de données relationnelle. Il suffit de spécifier le mapping entre la représentation relationnelle et la représentation des données RDF.

L'approche vise à générer les requêtes qui utilisent des optimisations comme la sélection des candidats et l'optimisation de l'auto-jointure (comme le système -ontop-). La principale différence est de savoir comment traiter l'algèbre SPARQL. Ils utilisent une algèbre modifiée, qui contient les informations de mapping. Par conséquent, il est capable d'effectuer certaines optimisations avant qu'une requête ne soit transformée en une forme relationnelle. De plus, la transformation de la requête SPARQL sera sous une forme mieux adaptée à la réécriture. Pour cela, les optimisations sur le modèle relationnel sont plus efficaces.(Chaloupka & Nečaskỳ 2016)

4.3.8 Efficient Handling of SPARQL OPTIONAL for OBDA

Optional est une fonctionnalité clé de SPARQL pour traiter les informations manquantes. Bien que cet opérateur soit largement utilisé, il est également connu pour sa complexité, ce qui peut rendre l'évaluation des requêtes avec OPTIONAL difficile. Les auteurs abordent ce problème dans l'approche OBDA (Ontology-Based Data Access), où les données sont stockées dans une base de données relationnelle SQL et exposées sous forme de graphe RDF virtuel au moyen d'un mapping R2RML. Ils commencent par une traduction succincte d'un fragment SPARQL en SQL, qui respecte pleinement la sémantique des valeurs et s'appuie sur l'utilisation extensive de l'opérateur LEFT JOIN et de la fonction COALESCE. ils proposent ensuite des techniques d'optimisation pour réduire la taille et améliorer la structure des requêtes SQL générées. Les optimisations capturent les

interactions entre JOIN, LEFT JOIN, COALESCE et les contraintes d'intégrité telles que les attributs nuls, l'unicité et les contraintes de clé étrangère. Ces optimisations sont efficaces et toutes les traductions optimisées ont de meilleures performances (évaluées en moins d'une seconde) (Xiao *et al.* 2018).

4.3.9 Discussion

L'état de l'art met en évidence différents aspects du processus de mapping RDB to RDF par la réécriture, Certains de ces aspects doivent être pris en compte pour permettre la standardisation du processus de réécriture

L'un des aspects importants de la réécriture de requête est la possibilité de modéliser explicitement des informations qui ont été modélisées implicitement ou pas du tout représentées dans RDB, telles que la sémantique de domaine. Ainsi, de nombreux outils et applications génériques ont noté l'importance d'utiliser une ontologie de domaine, en plus des informations provenant du schéma de la base de données relationnelles, pour générer du RDF. Un autre aspect important du mapping RDB to RDF est le potentiel d'intégration de données en représentant des données de plusieurs sources RDB sous la forme d'un graphe RDF unique.

L'approche de réécriture de requêtes doit produire des requêtes SQL «raisonnables», c'est-à-dire ni trop volumineuses ni trop complexes pour être optimisées efficacement par le moteur de base de données. Ainsi, la technique de réécriture de requête doit résoudre deux problèmes différents : (i) un problème de traduction de requête qui implique des mappings RDB-RDF sur des schémas relationnels, et (ii) un problème d'optimisation de requête. Il existe un certain nombre de systèmes et de techniques liés à ce problème, tels que ceux décrits dans SparqlMap et Ultrawrap. Cependant, chacune de ces approches a des limitations qui affectent les aspects critiques de la réponse aux requêtes sur RDF virtuel. Ces limitations incluent la génération de requêtes SQL inefficaces ou même incorrectes, le manque d'un plan formel et de mauvaises implémentations. De plus, certains d'entre eux ne prennent pas en charge les schémas de base de données arbitraires car ils ne prennent pas en charge les langages de mapping RDB vers RDF, tels que R2RML.

Les systèmes OBDA, en revanche, sont configurés sur des sources de données relationnelles existantes et exploitent leurs schémas spécifiques au domaine. En utilisant des ontologies et des mappings, ils exposent la base de données sous la forme d'un graphe RDF virtuel qui peut être interrogé à l'aide de SPARQL.

Certains systèmes OBDA ont des capacités de raisonnement. La stra-

tégie la plus courante pour les triplestores est le chaînage direct, qui consiste à étendre l'ensemble des triplets RDF au moyen d'inférences selon un ensemble de règles donné. Qui conviennent le mieux aux triplestores. Le chaînage direct présente certains inconvénients : les déductions peuvent être coûteuses en termes de temps et d'espace ; en outre, les mises à jour et les suppressions de triplets nécessitent une comptabilité supplémentaire pour le raisonnement incrémentiel. De plus, cette approche ne peut pas être adoptée sans sacrifier l'exhaustivité de la réponse aux requêtes lorsque le langage d'ontologie est capable d'inférer de nouveaux individus dans les données.

La réponse aux requêtes SPARQL en une seule requête sur des données relationnelles est le but de plusieurs approches et implémentations. Comme de nombreux triplestores natifs sont basés sur des bases de données relationnelles, il existe des recherches considérables sur le stockage efficace des données RDF dans un schéma relationnel. D'autre part, le mapping de bases de données relationnelles dans RDF est un moyen de tirer parti des données existantes sur le Web de données sémantiques. Nous pouvons différencier les outils RDB en RDF qui facilitent simplement la configuration d'une interface de données liées, et les outils qui exposent un SPARQL endpoint pour l'interrogation interactive des données, ce qui permet de servir une utilisation beaucoup plus sophistiquée.

Des études très récentes (Chaloupka & Nečaskỳ 2016) se concentrent sur le traitement de l'algèbre SPARQL avant toute transformation. Et proposent un algorithme pour traduire SPARQL-to-SQL basé sur l'arbre algébrique SPARQL, l'un des problèmes dans une telle approche est qu'il ne prend pas en charge de plus gros fragments de SPARQL (par exemple, l'agrégation, négation et requêtes de chemin). De plus, ne prend pas en compte les expressions complexes de la requête, comme l'optional graph pattern complexe.

Enfin, cette synthèse devrait non seulement servir comme étude comparative mais aussi les chercheurs impliqués dans la mapping RDB to RDF afin de soutenir l'évolution du Web des documents dans un «Web de données».
Face à la grande variété d'initiatives des approches de réécriture de requêtes, plusieurs études ont été menées pour comparer les approches et les techniques.Et pour cela nous avons identifié le besoin de générer des requêtes SQL qui peuvent être exécutées efficacement sur les moteurs relationnels. Par rapport aux approches existantes, nous ne appuyons pas uniquement sur les optimisations de la requête relationnelle, mais sur la requête SPARQL en premier par le traitement de son Algèbre SPARQL.

TABLE 4.2 – *Systèmes OBDA de réecriture SPARQL-to-SQL*

System	Mapping	Algorithme	Optimisations	Query Implementation
D2R server	D2RQ Mapping, R2RML	X	X	SPARQL $\rightarrow SQL \rightarrow RDB$
Mastro	R2RML	X	X	SPARQL $\rightarrow SQL \rightarrow RDB$
Ultrawrap	Native,R2RML	the existing algorithmic in SQL optimizers	Detection of unsatisfiable conditions, self-join elimination	SPARQL $\rightarrow SQL \rightarrow RDB$
Morph-RDB	R2RML	X	Self-join elimination	SPARQL$\rightarrow$ $SQL \rightarrow RDB$
SparqlMap	R2RML	BGPtoSQL, SPARQL to SQL (Chebotko)	Filter, Self-union, Self-join, Filter and Projection Pushing	SPARQL $\rightarrow SQL \rightarrow RDB$
Ontop	Ontop mapping, R2RML	Ontop Algorithm based on T-mapping	Sub query elimination, pushing join	SPARQL $\rightarrow SQL \rightarrow RDB$
EVI	R2RML	Algorithm based on SPARQL algebra	candidate selection,self-join optimization	SPARQL $\rightarrow SQL \rightarrow RDB$
EHOptional	R2RML	BGPtoSQL, SPARQL to SQL (Chebotko)	Compatibility Filter Reduction Left join Naturalization...	SPARQL $\rightarrow SQL \rightarrow RDB$

4.4 CONCLUSION

Le Web sémantique est une technologie relativement nouvelle, qui promet l'intégration et la réutilisation des données par des agents intelligents à l'échelle du Web grâce à l'utilisation d'un modèle de données RDF, d'un langage de requête SPARQL et d'ontologies. Cependant, Une condition critique pour l'évolution du Web est l'inclusion des grandes quantités de données stockées dans les bases de données relationnelles (RDB). Le mapping de ces quantités de données de RDB vers le RDF a fait l'objet d'un grand travail de recherche dans divers domaines et a conduit à l'implémentation d'outils de mapping génériques ainsi que des applications spé-

cifiques au domaine. En outre, le rôle de RDF en tant que plate-forme d'intégration pour les données provenant de différentes sources, principalement sous la forme de RDB, est l'une des principales motivations des efforts de recherche dans la mapping RDB à RDF.

Ce chapitre documente les approches actuelles du mapping RDB to RDF. Nous parlons des approches de matérialisation ainsi que certains outils OBDA pour la réécriture de requête. Et nous avons analysé les motivations, les points communs et les différences entre ces approches existantes. L'expressivité des langages de mapping existants n'est pas toujours suffisante pour produire des données sémantiquement riches et les rendre utilisables, interopérables et interconnectables.

Un des objectifs importants de l'état de l'art est d'analyser le «gain d'information» de mapping RDB to RDF à travers une modélisation explicite des relations entre les entités qui sont implicites ou inexistantes dans RDB et l'incorporation de «sémantique du domaine». Par l'utilisation d'une ontologie de domaine spécifique à l'application dans le mapping RDB to RDF est un aspect important pour tirer pleinement parti de l'expressivité des modèles RDF. Et examiner également l'utilisation des règles d'inférence sur le RDF menant à la découverte des nouvelles connaissances.

Suite à cette synthèse, nous présentons dans le chapitre suivant, une approche formelle de mapping RDB-to-RDF basée sur un ensemble d'algorithmes qui génère automatiquement un ensemble de mapping en combinant des techniques et règles bien connues. Cette formalisation qui peut être intégrés dans notre approche afin de prendre en charge des mappings complexes vers des bases de données, et même à automatiser partiellement la génération de mappings appropriés à partir des relations vers le modèle RDF.(Lehireche *et al.* 2017)

Une nouvelle approche d'intégration des bases de données relationnelles dans le web sémantique. Que nous proposons dans le cadre OBDA est décrite dans la deuxième partie du chapitre, pour la réécriture de requête SPARQL en SQL. Plus exactement la réécriture de requête de type Optional imbriqué qui est connu par sa complexité élevé. Notre solution (Lehireche & Malki 2019) est basée sur le traitement de l'algèbre SPARQL et la sémantique de la requête, cela a pour but d'améliorer la traduction en obtenant une requête SQL efficace et optimisé avec un meilleurs temps d'exécution.

5 Approche de réécriture de requête basée sur l'Algèbre SPARQL

5.1 Introduction

Les technologies du Web sémantique trouvent de plus en plus d'applications pour résoudre des problèmes difficiles de recherche, de découverte, de partage et d'intégration de données relationnelles et de ressources informatiques. La croissance rapide des ensembles de données sémantiques présente un nouveau défi : une gestion efficace des données RDF qui est cruciale pour prendre en charge de nouvelles applications sémantiques.

L'intégration des bases de données relationnelles est devenue un problème de recherche important pour le Web sémantique et les communautés de bases de données, d'une part. L'accès et la gestion des données basées sur l'ontologie dans le cadre d'Ontology-Based Data Access (OBDA) d'autre part(Rodriguez-Muro 2010), qui est un paradigme populaire d'organisation de l'accès à divers types de sources de données qui a été développé depuis le milieu des années 2000, ou les requêtes sont posées sur une couche conceptuelle, puis traduites en requêtes sur la couche de données. La couche conceptuelle se présente sous la forme d'une ontologie qui définit un vocabulaire partagé, et la couche de données se présente sous la forme d'une ou plusieurs sources de données existantes. Dans ce contexte, le modèle de données le plus répandu pour la couche conceptuelle et son langage de requête correspondant sont RDF et SPARQL. Aujourd'hui, la plupart des données d'entreprise (couche de données) sont stockées dans des bases de données relationnelles, il est donc crucial que les systèmes OBDA prennent en charge le mapping RDB à RDF. Le nouveau standard W3C pour le mapping RDB vers RDF, R2RML, a été créé dans ce but.(Rodriguez-Muro & Rezk 2015)

Les mappings R2RML sont utilisés pour exposer les bases de données relationnelles sous forme de graphes RDF virtuels. Ces graphes virtuels peuvent être matérialisés, générant des triplets RDF qui peuvent être utilisés avec des triplestores RDF, ou ils peuvent également être conservés virtuels et interrogés uniquement pendant l'exécution de la requête. L'approche virtuelle évite le coût de la matérialisation et permet de profiter de la maturité de plus de 40 ans des systèmes relationnels (par exemple, réponse aux requêtes efficace, sécurité, support de transaction robuste, etc.). L'une des approches les plus prometteuses pour répondre aux requêtes le graphe RDF est la réponse par réécriture de requêtes, c'est-à-dire la traduction de la requête SPARQL d'origine en une requête SQL équivalente. Cette requête SQL est ensuite déléguée au SGBD pour exécution.(Hazber *et al.* 2016)

Afin d'utiliser les avantages fournis par le SGBD, la technique de réécriture des requêtes doit produire des requêtes SQL «raisonnables»,

c'est-à-dire ni trop grandes ni trop complexes pour être efficacement optimisées par le moteur de base de données. Ainsi, la technique de réécriture de requêtes doit s'attaquer à deux problèmes différents : (i) un problème de traduction de requête qui implique des mappings RDB-à-RDF sur des schémas relationnels arbitraires, et (ii) un problème d'optimisation de requête. Il existe un certain nombre de systèmes et de techniques liés à ce problème, tels que ceux décrits dans le chapitre 4 (état de l'art). Cependant, chacune de ces approches a des limites qui affectent les aspects critiques de la réponse aux requêtes via le graphe RDF virtuel. Ces limitations incluent la génération de requêtes SQL inefficaces ou même incorrectes, le manque des implémentations. De plus, certains d'entre eux ne prennent pas en charge les schémas de base de données arbitraires car ils ne prennent pas en charge les langages de mapping RDB vers RDF, tels que R2RML. (Kontchakov *et al.* 2014)

Dans ce chapitre, nous présentons d'abord l'approche proposée dans le cadre du mapping de bases de données relationnelles en RDF dans la section 2, la section 3 est consacrée à l'approche de réécriture, en traitant le problème d'optimisation de requêtes, nous décrivons dans cette section notre approche d'optimisation basée sur l'algèbre SPARQL, nous définissons d'abord la sémantique de SPARQL et montrons par la suite son algèbre sur plusieurs exemples et opérations. Nous décrivons ensuite dans la section 4 une expérimentation de notre solution. Nous montrons enfin l'intérêt de notre approche à travers l'évaluation des résultats par rapport aux approches existantes présentées dans le chapitre précédent.

5.2 Mapping des Bases de Données Relationnelles vers RDF

Les mappings des bases de données relationnelles vers RDF sont appelés mappings RDB2RDF. Cette section définit la notion de mapping RDB2RDF en utilisant une ontologie, qui joue un rôle fondamental dans cette approche. Un algorithme efficace est défini pour faire les mappings pour les ontologies. Enfin, l'approche proposée pour faire ce mapping. (Lehireche *et al.* 2017)

5.2.1 Bases de Données Relationnelles aux Mapping RDF

Les chercheurs ont adopté une approche basée sur la matérialisation pour développer des systèmes OBDA : Dans cette approche de matérialisation, la base de données relationnelle d'entrée D, l'ontologie cible O et la

Mapping M (de D à O) sont utilisées pour dériver de nouveaux triplets qui sont stockés dans la base de données RDF Do, qui est considérée comme la matérialisation des données dans D étant donné M et O. La réponse à une requête SPARQL Q sur l'ontologie cible est calculée en posant directement Q sur Do. (Sequeda 2016)

Dans un système OBDA, les bases de données relationnelles sont mappées en graphes RDF, où chaque constante de D est transformée en IRI ou en littéral. Ce processus est généralement reporté à l'aide de certaines fonctions de transformation intégrées.

Un mapping d'une base de données relationnelle vers RDF est désigné comme un mapping RDB2RDF. Il existe deux normes connexes. Ici, une approche alternative est adoptée qui a été largement utilisée dans les domaines d'échange de données et d'intégration de données, et qui est basée sur le mapping R2RML. En substance, le standard R2RML décrit comment une base de données relationnelle peut être transformée en RDF en fonction de term maps et triple maps. La base de connaissances RDF résultante peut être matérialisée dans un triple store et ensuite interrogée à l'aide de SPARQL. Afin d'éviter une étape de matérialisation coûteuse, les implémentations R2RML peuvent mapper dynamiquement une requête SPARQL d'entrée dans une requête SQL correspondante, ce qui rend exactement les mêmes résultats que la requête SPARQL en cours d'exécution par rapport au graphe RDF matérialisé.

5.2.2 R2RML (Relational Database to RDF Mapping Language)

R2RML est un langage de mapping qui permet de spécifier comment créer des triplets RDF à partir de données dans une base de données relationnelle. Les mappings R2RML sont exprimés sous forme de graphes RDF (en utilisant la syntaxe Turtle). Un mapping bien formé se compose d'un ou de plusieurs arbres avec une structure comme le montre la figure 5.1. Chaque arbre a un nœud racine, appelé nœud triple map, qui est connecté à exactement un nœud de table logique, un nœud de subject map et un ou plusieurs nœuds predicate object map. Intuitivement, le triple map indique comment construire un ensemble de triplets (sujet, prédicat, objet) en utilisant i) les données de la table logique (qui peut être une table, une vue ou une requête), ii) le sujet spécifié par le nœud subject map, et iii) les prédicats et objets spécifiés par chacun des nœuds predicate object map.

Exemple 5.1 : Soit DB une base de données composée par la table student avec des colonnes [id ; name] (les clés primaires sont soulignées). Voici un mapping R2RML valide pour cette base de données :

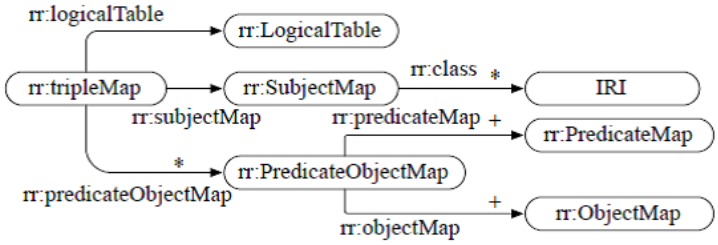

FIGURE 5.1 – *Un nœud de mapping R2RML bien formé*
(Rodriguez-Muro & Rezk 2015)

_ :m1 a rr :TripleMap ; rr :logicalTable [rr :tableName "student"] ;
rr :subjectMap [rr :template " :stud/ {id}" ; rr :class :Student] ;
rr :predicateObjectMap [rr :predicate :name ; rr :objectMap
[*rr : column"name"*]] .

Ce mapping contient un seul nœud de triple map. Le nœud de table logique fait référence à la table student. Le mapping génère des triplets dont le sujet est construit à l'aide d'un modèle qui fait référence à une colonne de la table, c'est-à-dire stud / id. Il indique également que ces sujets sont de type : Student (par la propriété rr : class) et ont une propriété associée nommée : name dont la valeur est obtenue à partir du nom de la colonne. En présence des lignes student = f (20 ; "Reda") (21 ; "Rym") g, un processeur R2RML générerait les triplets suivants :

:stud/20 rdf :type :Student ; :name "Reda" .
:stud/21 rdf :type :Student ; :name "Rym" .

5.2.3 Problèmes de Mapping RDB-to-RDF

Plusieurs approches (automatique ou manuelle) ont été proposées dans l'intégration RDB et web sémantique, concernant principalement la création et la maintenance de mappings entre le modèle relationnelle et le modèle RDF. Le mapping RDB-to-RDF est un domaine dans lequel de nombreux travaux ont été proposés au cours des dernières années. Généralement, l'objectif est d'exprimer le contenu RDB en utilisant l'ontologie en graphe RDF d'une manière qui permet de répondre aux requêtes soumises aux modèle RDF avec des données stockées dans la base de données relationnelles.

Pour intégrer les bases de données relationnelles (RDB) dans les applications Web sémantiques, le groupe de travail W3C RDB2RDF

a recommandé deux approches, Direct Mapping (DM)[1] et R2RML (Hazber *et al.* 2016). Le DM est une recommandation candidate du W3C. C'est la méthode par défaut pour traduire automatiquement une base de données relationnelle (schéma et données) aux instances d'ontologie (triplets OWL / RDF (S) et RDF) via un mapping direct sans interaction de l'utilisateur. L'ontologie représentée au format OWL / RDF (S). Le RDF reflétera le modèle de données exacte des données relationnelles, plutôt que le domaine des données. Un mapping directe fonctionne généralement en transformant chaque table en classe, colonne en propriété et relation avec une propriété d'objet.

Chaque ligne du tableau sera transformée en un individu qui sera membre de la classe du tableau. La clé étrangère transformée avec une propriété qui relie un individu à un autre. Néanmoins, la méthode DM peut ne pas être constamment suffisante ou optimale, en particulier lors du mapping d'une base de données relationnelle à une ontologie existante, tandis que le R2RML permet aux utilisateurs de définir manuellement les mapping en fonction de l'ontologie ciblée existante.

De nombreuses techniques de mapping RDB-to-RDF et outils correspondants ont été proposés ces dernières années. Malgré la convergence observée de plusieurs d'entre eux vers R2RML, les fournisseurs de données désireux de publier leurs données dans un format lisible par machine peuvent avoir du mal à faire un choix. Premièrement, différentes techniques véhiculent différentes approches philosophiques (par exemple, se concentrer sur Ontology learning, mapping language design, query engine design...) qui ont des implications sur la façon dont les données relationnelles sont exposées. Deuxièmement, le choix de la recommandation R2RML ne répond pas à toutes les questions : comme tout autre langage, R2RML a certaines limites en ce qui concerne les types de mappings qui peuvent être exprimés. De plus, en tant que langage de mapping indépendant de l'implémentation, R2RML ne répond pas à certaines questions courantes qui se produisent lors de la traduction de données relationnelles en RDF, telles que l'implémentation du processus de traduction. Ajoutons à ça, un problème majeur pour utiliser R2RML est l'effort de création manuelle de documents de mapping R2RML. Cela peut entraîner l'apparition de nombreuses erreurs dans les documents R2RML et nécessite des experts du domaine.

Ainsi que L'expressivité des langages de mapping existants n'est pas toujours suffisante pour produire des données sémantiquement riches et les rendre utilisables, interopérables et liées.

1. https ://www.w3.org/TR/rdb-direct-mapping/

5.2.4 Approche de Mapping RDB-to-RDF

Différentes approches de mapping RDB-to-RDF ont récemment été proposées, un langage de mapping est utilisé pour exposer les bases de données relationnelles sous forme de graphes RDF virtuels. En revanche, ces approches fournissent souvent de très brèves descriptions des outils développés, ce qui les rend difficiles à comparer. Les questions opérationnelles telles que le choix entre la traduction des données relationnelles en RDF et l'utilisation en temps réel des bases de données relationnelles, le choix des moyens d'accès (accès par requête ou données liées), et d'interrogation des données exposées, la durabilité des outils, etc., sont à peine abordées.

Afin de saisir la diversité des approches RDB-to-RDF, nous avons identifié trois besoins communs souvent ciblés : accéder aux données à partir du deep Web, données liées et intégration de plusieurs sources de données hétérogènes. Les outils donc basé sur le R2RML uniquement comme un langage de description de mapping, ils peuvent adopter des stratégies complètement différentes.

Les caractéristiques obligatoires précisent que R2RML doit (i) prendre en charge à la fois le mapping directe et le mapping basé sur la sémantique du domaine ; (ii) fournir suffisamment d'informations pour qu'un processeur prenne en charge à la fois on-demand mapping (réécriture des requêtes SPARQL en SQL) et la matérialisation des données. Les autres fonctionnalités obligatoires sont : la génération d'identifiants uniques, la prise en charge des conversions de types de données, le changement de nom de colonne, la relation plusieurs-à-plusieurs avec des triplets simples, 1 table à n classes.

Néanmoins, certaines situations suggèrent que R2RML ne peut pas être considéré comme un langage de mapping. Premièrement, les approches basées sur le mapping directe ne suggèrent pas forcément un langage de mapping : les règles de ce dernier peuvent facilement être encodées dans un programme sans avoir besoin d'interpréter un langage de mapping. Deuxièmement, certaines approches (comme l'ontology learning) traitent des modèles complexes qui ne peuvent pas être exprimés en R2RML. Enfin, R2RML ne fournit pas de fonctions de manipulation de données mais s'appuie plutôt sur les capacités du backend relationnel. Pour résoudre ces situations, une autre version de R2RML pourrait, par exemple, fournir des points d'extension de language pour permettre l'invocation de fonctions spécifiques au domaine fourni sous forme de plug-in logiciel.

Pour rendre viable l'intégration de SPARQL et des bases de données relationnelles, nous devons résoudre deux problèmes différents : (i) un

problème de traduction de requête qui implique des mappings SQL-RDF complexes, et (ii) un problème d'optimisation de requête. L'approche présentée dans cette thèse (Lehireche *et al.* 2017) traite les deux aspects. Tout d'abord, dans cette section nous proposons une approche formelle basée sur un ensemble d'algorithmes qui génère automatiquement un mapping adaptant et en combinant des techniques et règles bien connues. Cette formalisation qui peut être intégrés dans notre technique afin de prendre en charge des mappings complexes vers des bases de données, et même à automatiser partiellement la génération de mappings appropriés à partir des relations vers les vocabulaires RDF. Cependant, la croissance des représentations sémantiques sur le Web manque encore d'élan. De notre point de vue, une des principales raisons du manque de déploiement de ces outils et approches réside dans la complexité de générer manuellement des mappings. Notre formalisation est dynamique. Il ne prend en compte que la requête SPARQL actuelle. Il ne concerne que les données qui couvrent les triplets spécifiés dans la requête SPARQL. Le SQL est utilisé pour obtenir les données RDB qui couvrent les triplets spécifiés dans la requête SPARQL.

Nous montrons que la combinaison du mapping et de l'optimisation permet à notre approche de surpasser les systèmes de réponse aux requêtes SPARQL vers SQL bien connus.

Algorithmes de l'approche de mapping RDB-to-RDF

Cette section définit les algorithmes sur lesquels le mapping de la base de données relationnelle vers RDF est appliqué. Selon les idées d'algorithmes proposées dans la recommandation R2RML, et DM du W3C, nous avons conçu un ensemble d'algorithmes de mapping, ON_demand_ETL_RDB_To_RDF_Mapping () (Algorithme 1), Générer_triplets_couvrant_r(R) (Algorithme 2), Générer_requête_sql_couvrant (R) (Algorithme 3), appliquer_R_sur(t) (Algorithme 4), Générer_URI (triplet, LNR) (Algorithme 5), pour obtenir le fichier de mapping RDB–to–RDF. En résumé, ON_demand_ETL_RDB_To_RDF_Mapping () est l'algorithme principal invoquant les autres algorithmes pour implémenter les règles du mapping.

Algorithme 1 ON_demand_ETL_RDB_To_RDF_Mapping ()

Entrées : R // R est une requête SPARQL
: TE // TE est RDF triplestore initialement vide
Donnée_locale : T // T est l'ensemble de triplets générées pour R
Sorties : DS // DS, ensemble de données, est la réponse donnée par l'exécution de R
Début
1. DS ⟵ Appliquer_R_sur (TE)
2. Si DS = Ø Alors //R est non satisfaisant
3. T ⟵ Génrer_triplets_couvrant_r(R); // T est l'ensemble de triplets possibles qui contient une réponse à R
4. TE ⟵ TE ⊕ T; // exclusive union de TE et T
5. DS ⟵ Appliquer_R_sur(T);
6. Fin si
Fin

Description : Cet algorithme est l'algorithme principal pour générer des mappings, montre comment obtenir des données RDF sous la forme de triplets à partir des données relationnelles, en commençant par interroger le triple store avec une requête SPARQL (R). Nous avons le triple store (TE) de l'entreprise initialement vide, et nous générons le triplet qui contient la réponse à la requête (R), enfin, nous stockons le triplet généré dans le TE et répétons la demande.

Algorithme 2 Générer_triplets_couvrant_r(R)

Entrées : R // R est une requête SPARQL
Sorties : t // t est un ensemble de triplets
Donnée_locale : triplet // un triplet RDF (sujet, prédicat, objet)
: RSQL // ensemble de requêtes SQL qui couvrent les données de R
: TS // TS l'ensemble de table contenant le résultat de la requête SQL
: LNR // liste de nom de table de TS
Début
1. RSQL ⟵ Générer_requête_sql_couvrant_r(R) ;
2. TS ⟵ intérroger_BBDR(RSQL) ;
3. LNR ⟵ extraire_noms_relations(TS) ;
4. Pour chaque table de (TS) faire
5. Pour chaque ligne de la table faire
6. Triplet ⟶ sujet = valeur (ID associé à la ligne) ;
// valeur de la clé pour une instance de la relation
//i.e la première colonne pour la ligne de la table
7. Pour chaque colonne de la ligne faire // à partir de la deuxième colonne
8. Triplet ⟶ prédicat = nom (attribut associé à la colonne) ;
9. Triplet ⟶ objet = valeur (valeur associé à la cellule) ;
10. Triplet ⟵ Générer_URI (triplet, LNR) ;
11. t ⟵ t + triplet // stocker le triplet généré dans t
12. Fin pour
13. Fin pour
14. Fin pour
15. Fin si
16. Retourner (t) ;
Fin

Description : Le deuxième algorithme formalise le mapping comme un ensemble de table mappée en triplets, et illustre comment les données stockées dans cette table peuvent être transformées en RDF avec un sujet, un prédicat, un objet. Les composants d'un triplet sont générés à l'aide d'un terme du mapping défini. Chaque terme peut être constant ou variable. Les constantes pointent vers un URI ou un littéral RDF pour la valeur du composant tandis que la variable se réfère à une colonne du schéma relationnel où la valeur du composant peut être récupérée. Cet algorithme crée est utilisé pour traduire chaque ligne de la table logique

en un nombre de triplets RDF.

Algorithme 3 Générer_requête_sql_couvrant (R)

Entrées : R // R est une requête SPARQL
Sorties : RSQL // ensemble de requêtes sql qui couvrent les données nécessaire par R
Donnée_locale : TinR //ensemble de triplets dans la clause « where » de R
Donnée_locale : LSP // ensemble de (sujet, prédicat) dans la clause « where » de R
Début
1. TinR ⟵ extraire_triplet_à_partir_de _where(R) ;
2. LSP ⟵ extraire_sujet_prédicat_àpartir_de(TinR) ;
3. RSQL ⟵ Générer_requête_sql_couvrant(LSP) ;
4. Retourne RSQL ;
Fin

Description : l'algorithme extrait les informations générales importantes sur la base de données et invoque la fonction extraire_triplet_à_partir_de_where pour représenter les triplets pour interroger la base de données et obtenir du résultat.

Algorithme 4 appliquer_R_sur(t)

Entrée : R // R est la requête SPARQL
Entrée : t // t est l'ensemble de triplets
Sorties : DS // DS, ensemble de données, est la réponse donnée par l'application de R sur l'ensemble de triplets
Début 1. Réponse ⟵ interpréter SPARQL (R,t) ; // instancie le triplestore avec t.
// appliquer R sur la requête SPARQL.
2. Si Réponse = null alors
3. DS ⟵ Ø
4. Sinon // obtenir les données et construire le résultat sur DS.
5. Fin si
6. Retourne DS ;
Fin

Description : l'algorithme peut être utilisé pour insérer des triplets dans le triplestore, en créant les données en fonction des informations du mapping obtenu précédemment. Après avoir obtenu le triplet, nous

devons appliquer le SPARQL sur cet ensemble de triplet pour donner une réponse.

Algorithme 5 Générer_URI (triplet, LNR)

Entrée : triplet // triplet extrait de la table
Donnée_locale : LNR // liste des noms des tables TS
Donnée_locale : URI
Sorties : triplet // triplet avec URIs
Début
1. URI = modèle_URI (triplet $\longrightarrow$ sujet)
2. Triplet $\longrightarrow$ sujet = URI
3. Si triplet $\longrightarrow$ objet $\in$ LNR alors // nom de relation
4. URI= modèle_URI (triplet $\longrightarrow$ sujet)
5. Fin si
6. Retourne triplet ;

Fin

Description : C'est l'un des algorithmes importants. Il génère l'URI unique utilisé comme sujet pour tous les triplets RDF générés à partir de la ligne de la table. Cet algorithme appelle la fonction modèle_URI à la forme d'identification de la clé primaire de chaque triple généré à partir de la ligne de table.

Implémentation

Les algorithmes décrits dans cette approche de mapping ont été implémentés sur la plateforme Eclipse IDE (J2SE, JDK 1.7). Ainsi, une connexion SQL à la base de données relationnelle et un IRI de base. Pendant ce temps, les sorties sont produites automatiquement, un document de mapping R2RML et l'ensemble de données RDF résultant. Ces sorties sont affichées à l'écran, et le document de mapping R2RML et les triplets RDF peuvent être enregistrés dans un fichier RDF dans différents formats de syntaxe. De plus, le fichier de mapping RDF peut être utilisé avec n'importe quel analyseur R2RML pour convertir des données relationnelles en triplets RDF. Le prototype de système actuel prend en charge les connexions SQL à MySQL.
Eclipse[2] propose un framework de développement logiciel fournissant des briques logicielles pour développer ces outils. Eclipse est à la fois considéré comme un EDI, un framework ou une plateforme, selon que

2. https ://www.eclipse.org/downloads/

l'on considère le projet, ses composants. Les EDI résultant de leur assemblage : En effet le projet Eclipse propose également des « packages » en téléchargement : il peut s'agir : d'applications « prêtes à l'emploi » facilitant la diffusion d'Eclipse, en intégrant chacune un ensemble cohérent de plugins autour de l'Eclipse Platform pour répondre à différents besoins spécifiques : Il s'agit essentiellement d'IDE spécialisés, tels que Eclipse Classic, Eclipse IDE for Java EE Developers, Eclipse IDE for C/C++ Developers, Eclipse for Mobile Developers, mais également d'AGL comme Eclipse Modeling Tools.

Apache Jena[3] est un Framework Java gratuit et open source pour la construction des applications du web sémantique et de l'annotation sémantique (Linked data). Il est composé de différentes APIs interagissant ensemble pour traiter les données RDF, Il fournit de nombreuses bibliothèques Java pour aider les développeurs à développer du code qui gère RDF, RDFS, RDFa, OWL et SPARQL conformément aux recommandations publiées du W3C. Jena comprend un moteur d'inférence basé sur des règles pour effectuer un raisonnement basé sur des ontologies OWL et RDFS, et une variété de stratégies de stockage pour stocker les triplets RDF en mémoire ou sur disque.

MySQL[4] est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multithread et multi-utilisateur.

C'est un logiciel libre, open source, développé sous double licence selon qu'il est distribué avec un produit libre ou avec un produit propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde4, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, PostgreSQL et Microsoft SQL Server.

Résultats expérimentaux et discussion

Nous avons réalisé des expériences en utilisant MYSQL, le langage de programmation Java, Eclipse IDE et l'outil Apache Jena. Des tests expérimentaux sur l'efficacité et la validité de nos algorithmes de mapping RDB2RDF ont été menés avec les tailles de schéma de la base de données créé avec MYSQL et testés dans nos expériences.

3. https ://jena.apache.org/
4. https ://dev.mysql.com/downloads/mysql/5.7.html

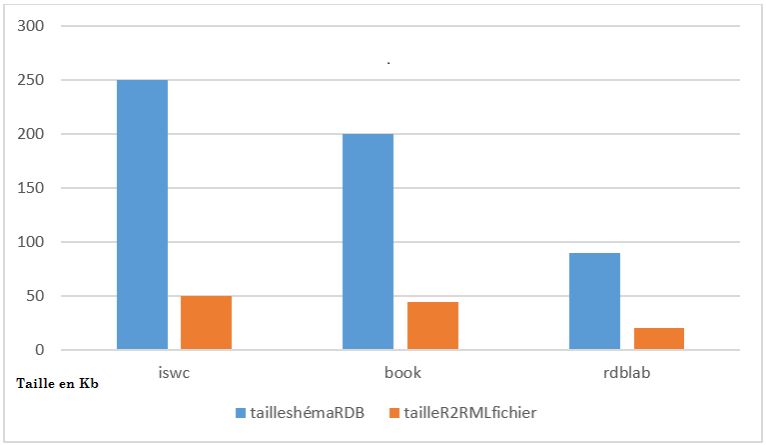

FIGURE 5.2 – *Tailles de schéma dans RDB et les fichiers de mapping R2RML*

Tout d'abord, on a fait une comparaison entre trois bases de données avec trois différentes tailles de données. Cette comparaison est réalisée sur des concepts essentiels dans notre approche. Ces concepts sont les facteurs importants pour la création de fichiers de mapping R2RML et affectant les performances de l'algorithme pour la génération du fichier de mapping R2RML et la conversion des données de RDB en RDF ensembles de données, les résultats (dans la figure 5.2) montrent les tailles du schéma RDB par rapport aux fichiers de mapping R2RML qui ont été extraits. A partir de ces deux champs. Nous pouvons conclure quels sont les facteurs les plus importants pour extraire le document de mapping R2RML de RDB et affecte le temps de la génération des données RDF. Bien que la taille de la base de données de iswc soit supérieure à la taille de la base de données de book et rdblab, la taille du fichier R2RML est petit, car d'autres facteurs ont influencé le temps d'exécution pour générer le fichier de mapping R2RML avec la taille de la base de données. Dans les travaux futurs, nous ajouterons une étude très détaillé de la performance et le temps d'exécution des Algorithmes de notre approche.

5.3 Optimisations de la réécriture SPARQL-–to–SQL

5.3.1 Syntaxe et Sémantique de SPARQL

Le problème naturel de l'interrogation des données RDF a été soulevé. Depuis lors, plusieurs conceptions et implémentations de langages de requête RDF ont été proposées. En 2004, le RDF Data Access Working Group (qui fait partie de "The Semantic Web Activity) a publié une première version de travail d'un langage de requête pour RDF, appelé SPARQL.(Prud'hommeaux 2008) Actuellement SPARQL est une recom-

mandation du W3C. Essentiellement, SPARQL est un langage de requête de matching de graphes. Étant donné une source de données D, une requête consiste en un motif (pattern) qui est comparé à D, et les valeurs obtenues à partir de ce matching sont traitées pour donner la réponse. La source de données D à interroger peut être composée de plusieurs sources. Une requête SPARQL se compose de trois parties. La partie de pattern matching, qui comprend plusieurs caractéristiques intéressantes de pattern matching de graphes, comme les parties Optionals, l'union de patterns, l'imbrication (nesting), le filtrage (ou la restriction) des valeurs de matching possibles, et la possibilité de choisir la source de données à associer par un pattern. Les modificateurs de solution, qui une fois la sortie du pattern calculée (sous forme de tableau de valeurs de variables), permettent de modifier ces valeurs en appliquant des opérateurs classiques comme projection, distinct, ordre, limite et offset. Enfin, la sortie d'une requête SPARQL peut être de différents types : requêtes oui / non, sélections de valeurs des variables qui correspondent aux patterns, construction de nouveaux triplets à partir de ces valeurs, et descriptions des ressources. (Arenas & Pérez 2011) Bien que, les fonctionnalités de SPARQL sont simples à décrire et à comprendre, il s'avère que leur combinaison fait de SPARQL un langage complexe, dont la sémantique est loin d'être comprise. En fait, la sémantique de SPARQL actuellement donnée dans le document, ne couvre pas toutes les complexités apportées par les constructions impliquées dans SPARQL, et inclut des ambiguïtés, des lacunes et des fonctionnalités difficiles à comprendre.

La formalisation de la sémantique de SPARQL est nécessaire, pour plusieurs raisons, notamment pour servir d'outil pour identifier et dériver des relations entre les constructeurs, identifier les notions redondantes et contradictoires, et pour étudier la complexité, l'expressivité et d'autres questions de comme la réécriture et l'optimisation. Il existe des propositions concernant la sémantique de certains fragments de SPARQL. Ainsi que sur les problèmes formels de la sémantique des langages de requête pour RDF qui peuvent être utiles pour SPARQL. En fait, SPARQL partage plusieurs constructions avec d'autres propositions de langages de requête pour RDF.

Dans cette section, nous donnons une formalisation algébrique du fragment de base de SPARQL sur RDF simple, c'est–à–dire RDF sans vocabulaire RDFS et règles littérales. Cela nous permet d'analyser les composants de base du langage et d'identifier certaines de ses propriétés fondamentales.

Syntaxe des expressions du SPARQL Graph pattern

Définition 1. Soit RDF–I l'ensemble de tous les IRI, RDF–L l'ensemble de tous les littéraux et RDF–B l'ensemble de tous les nœuds vides. L'ensemble des termes RDF est défini comme RDF–T = RDF–I ∪ RDF–L ∪ RDF–B.(Harris *et al.* 2013)
V désigne l'ensemble de toutes les variables, qui est infini et disjoint de RDF–T. Une variable est n'importe quel v ∈ V. Un mapping de solution μ est une fonction partielle, μ : V ⟶ RDF–T. Le domaine de μ est noté dom (μ). Une séquence de solutions M est une liste de mappings de solutions.
Un triple pattern est un tuple de (RDF–T ∪ V) × (RDF–I ∪ V) × (RDF–T ∪ V). Une requête SPARQL est un tuple (R, F, P) où R est la forme de requête du résultat, F est un ensemble de clauses d'ensemble de données (éventuellement vide) et P est un graph pattern. La forme de requête du résultat est l'expression SELECT W où W ⊂ V est un ensemble de variables. Une clause d'ensemble de données se présente sous la forme FROM g ou FROM NAME g où g ∈ RDF–I.
Pour définir l'algèbre SPARQL, nous devons d'abord définir la sélection, qui est ensuite utilisée comme paramètre pour les filter patterns.

Définition 2. Une sélection est définie récursivement comme suit : Une expression SPARQL est une sélection. Si P est un graph pattern, alors EXISTS (P) est une sélection. Si $\sigma 1$ et $\sigma 2$ sont des sélections, alors $\sigma 1 \bigwedge \sigma 2$, $\sigma 1 \bigvee \sigma 2$ et $\neg\ \sigma 1$ sont des sélections. Maintenant, tout est préparé pour définir l'algèbre SPARQL. Nous définissons le graph pattern, qui représente la clause WHERE complète dans la requête SPARQL.(Chaloupka & Nečaskỳ 2016)

Définition 3. Un graph pattern est défini récursivement comme suit :
- Un ∅ est un graph pattern appelé pattern vide.
- Un TP est un graph pattern appelé triple pattern.
- Si P1 et P2 sont des graphs patterns, alors P1 ⋈ P2, P1 ⋃ P2 et P1 ∖ P2 sont des graphs patterns appelés respectivement join, union et minus.
- Si P est un graph pattern et σ est une sélection, alors σ (P) est un graph pattern appelé filter.
- Si P1 et P2 sont des graphs patterns et σ est une sélection, alors P1 ⟕ P2 est un graph pattern appelé (left join) jointure gauche.
- Si P est un graph pattern et u ∈ RDF-I ⋃ V alors Γ u (P) est un graph pattern appelé graph.

En d'autres termes, nous pouvons transformer chaque requête SPARQL en un arbre algébrique, où les feuilles sont des patterns vides, des triples

patterns et des valeurs. De plus, les nœuds non–feuilles sont des jointures, des unions, des filtres, des jointures gauches, des graphs et des extensions.(Chaloupka & Nečaskỳ 2016)

Sémantique des expressions du SPARQL Graph pattern

Objectif de la définition de la sémantique de SPARQL est pour détecter des problèmes sémantiques, définir des techniques d'optimisation et évaluer la complexité du langage. La sémantique est définie par une algèbre sur des ensembles de mappings de variables :
Pour définir cette sémantique des expressions du SPARQL graph pattern, nous devons introduire une terminologie de (Pérez *et al.* 2009).
Un mapping μ est une fonction partielle $\mu : V \longrightarrow (U \cup L)$. En abusant de la notation, pour un triple pattern t tel que var (t) $\subseteq$ dom (μ), on note μ (t) le triple obtenu en remplaçant les variables en t selon μ. Le domaine de μ , noté dom (μ), est le sous–ensemble de V où μ est défini.

Deux mapping $\mu 1$ et $\mu 2$ sont compatibles lorsque pour tous ? X $\in$ dom ($\mu 1$) $\cap$ dom ($\mu 2$), il est le cas que $\mu 1$ (? X) = $\mu 2$ (? X), c'est–à–dire lorsque $\mu 1 \cup \mu 2$ est aussi un mapping. Intuitivement, $\mu 1$ et $\mu 2$ sont compatibles si $\mu 1$ peut être étendu avec $\mu 2$ pour obtenir un nouveau mapping, et vice versa. Notez que deux mappings avec des domaines disjoints sont toujours compatibles et que le mapping vide $\cup \emptyset$ (c'est–à–dire le mapping avec un domaine vide) est compatible avec tout autre mapping.

Soit $\Omega 1$ et $\Omega 2$ des ensembles de mappings. Nous définissons la jointure de, l'union de et la différence entre $\Omega 1$ et $\Omega 2$ comme :
$\Omega 1 \bowtie \Omega 2 = \{ \mu 1 \cup \mu 2 \mid \mu 1 \in \Omega 1, \mu 2 \in \Omega 2$ et $\mu 1, \mu 2$ sont des mappings compatibles$\}$,
$\Omega 1 \cup \Omega 2 = \{\mu \mid \mu \in \Omega 1$ ou $\mu \in \Omega 2 \}$,
$\Omega 1 \setminus \Omega 2 = \{ \mu \in \Omega 1 \mid$ pour tous $\mu' \in \Omega 2$, μ et μ' ne sont pas compatibles$\}$.
De plus, nous définissons la jointure externe gauche comme : $\Omega 1 \mathbin{⟕} \Omega 2 = (\Omega 1 \bowtie \Omega 2) \cup (\Omega 1 \setminus \Omega 2)$.

Intuitivement, $\Omega 1 \bowtie \Omega 2$ est l'ensemble des mappings résultant de l'extension des mappings en $\Omega 1$ avec leurs mappings compatibles en $\Omega 2$, et $\Omega 1 \setminus \Omega 2$ est l'ensemble des mapping en $\Omega 1$ qui ne peut être étendu avec aucun mapping en $\Omega 2$.

L'opération $\Omega 1 \cup \Omega 2$ est l'union théorique habituelle. Un mapping μ est dans $\Omega 1 \mathbin{⟕} \Omega 2$ s'il s'agit de l'extension d'un mapping de $\Omega 1$ avec un mapping compatible de $\Omega 2$, ou s'il appartient à $\Omega 1$ et ne peut être étendu

avec aucun mapping de $\Omega 2$. Ces opérations ressemblent à des opérations d'algèbre relationnelle sur des ensembles de mappings (fonctions partielles).
Nous sommes maintenant prêts à définir la sémantique des expressions de graph pattern comme une fonction $[\![.]\!]_G$ qui prend une expression d'un graph pattern et renvoie un ensemble de mappings.

Définition 4. L'évaluation d'un graph pattern P sur un graphe RDF G, noté par $[\![P]\!]_G$ est définie récursivement comme suit : (Rodriguez-Muro & Rezk 2015)
1. Si P un triple pattern t, alors $[\![P]\!]_G = \{\mu \text{ dom } (\mu) = \text{var(t) et } \mu(t) \in G\}$.
2. Si P est (P1 AND P2), alors $[\![P]\!]_G = [\![P1]\!]_G \bowtie [\![P2]\!]_G$.
3. Si P est (P1 OPT P2), alors $[\![P]\!]_G = [\![P1]\!]_G \rtimes [\![P2]\!]_G$.
4. Si P est (P1 UNION P2), alors $[\![P]\!]_G = [\![P1]\!]_G \bigcup [\![P2]\!]_G$.

L'idée derrière l'opérateur OPTIONAL est de permettre un matching facultatif des graphs patterns. Considérons l'expression du graph pattern (P1 OPT P2) et laissez $\mu 1$ être un mapping dans $[\![P1]\!]_G$.
S'il existe un mapping $\mu 2 \in [\![P2]\!]_G$ telle que $\mu 1$ et $\mu 2$ sont compatibles, alors $\mu 1 \cup \mu 2$ appartient à $[\![P1 \text{ OPT } P2]\!]_G$. Mais s'il n'existe pas un tel mapping $\mu 2$, alors $\mu 1$ appartient à $[\![P1 \text{ OPT } P2]\!]_G$.(Xiao *et al.* 2018)
Ainsi, l'opérateur OPT permet d'ajouter des informations à un mapping μ si les informations sont disponibles, au lieu de simplement rejeter μ chaque fois qu'une partie du pattern ne correspond pas. Cette fonctionnalité de matching facultatif est cruciale dans les applications Web sémantiques, et plus particulièrement dans la gestion des données RDF, où l'on suppose que chaque application n'a qu'une connaissance partielle des ressources gérées.

5.3.2 Traduction en Algèbre SPARQL

L'algèbre SPARQL [5] définit la sémantique d'une exécution de requête SPARQL. L'expression algébrique est formée à partir de la chaîne de requête en analysant d'abord, puis en appliquant des transformations comme décrit ci-dessous. Le résultat d'une requête peut alors être calculé par les règles d'évaluation appliquées à l'expression algébrique. Cela donne les résultats corrects de la requête – cela n'implique que l'exécution réelle d'une requête qui doit être effectuée de cette manière.

La syntaxe de requête est convertie en une syntaxe abstraite décrite à l'aide de cette algèbre SPARQL, qui représente les structures de données à l'aide de listes entre parenthèses. L'algèbre permet l'évaluation de re-

5. https ://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra.html

quêtes SPARQL, par la traduction de l'expression SPARQL, optimisation logique, optimisation physique et à la fin une évaluation de cette expression.

Exemple 5.1 : Pattern simple = Triplet

Turtle bibliosem

```
@prefix : < http : // example.org/ns#>.
 : book1 : title "SPARQL Tutorial" ; : price 42 ; : editor : jena .
 : book2 : title "The Semantic Web" ; : price 23 .
 : book3 : title "RDF Framework" ; : prix 53 .
 : book4 : title "SPARQL pour les Nuls " ; : editor : pour les nuls .
 : pour les nuls : address "Par is ".
```

Requête ex1

```
PREFIX : < http ://asws.upmc.fr/exemples#>
SELECT ? title
FROM <ex1 . ttl >
WHERE { : book1 : title ? title . }
```

Expression algébrique de « ex1 »

```
( prefix ( ( : < http ://asws.upmc.fr/exemples# >) )
    ( project ( ? title )
        ( bgp ( triple : book1 : title ? title ) ) ) )
```

triple : pattern (motif) de triplet
bgp : basic graph pattern (motif de graphe de base)

Exemple 5.2 : Union = Union

Requête Construct

```
PREFIX dc : < http :// purl.org/dc/elements/1.1/ >
PREFIX ns : < http :// example.org/ns#>
PREFIX ab : < http :// asws.org/ns#>
CONSTRUCT  ?x ab : prix    ?price ;    ab : titre   ? title .
FROM <ex1.ttl >
WHERE { {?x   ns : price   ?price . }
UNION
{?x    dc : title   ? title . } }
```

Expression algébrique de « construct »

```
( prefix ( ( dc : < http ://purl.org/dc/elements/1.1 / > )
( ns : < http ://example.org/ns# >)
( ab : < http ://asws.org/ns# >) )
( union
    ( bgp ( triple ?x ns : price ?price ) )
        ( bgp ( triple ?x dc : title ? title ) ) ) )
```

Exemple 5.3 : OPTIONAL = leftJoin

Requête optional

```
PREFIX : < http ://example.org/ns#>
SELECT  ?x  ? title  ?y  ?address
FROM <bibliosem . ttl >
WHERE { ?x  : title  ? title .
OPTIONAL { ?x  : editor  : notexists
    OPTIONAL { ?y  : address  ?address } } }
```

Expression algébrique de « optional »

```
( prefix ( ( : < http ://example.org/ns# >) )
    ( project ( ?x ? title ?y ?address )
    ( leftjoin
        ( bgp ( triple ?x : title ? title ) )
            ( leftjoin
            ( bgp ( triple ?x : editor : notexists ) )
            ( bgp ( triple ?y : address ?address ) ) ) ) ) )
```

5.3.3 Description du problème

Dans le cadre OBDA, les requêtes sont posées sur une couche conceptuelle, puis traduites en requêtes sur la couche de données. La couche conceptuelle se présente sous la forme d'une ontologie qui définit un vocabulaire partagé, et la couche de données se présente sous la forme d'une ou plusieurs sources de données existantes. Dans ce contexte, le modèle de données le plus répandu pour la couche conceptuelle et son langage de requête correspondant sont RDF et SPARQL. Aujourd'hui, la plupart des données d'entreprise sont stockées dans des bases de données relationnelles, il est donc crucial que l'approche OBDA prennent en charge les mappings RDB-to-RDF. Le nouveau standard W3C pour les mappings RDB vers RDF, le R2RML, a été créé dans ce but.

Les mappings R2RML sont utilisés pour exposer les bases de données relationnelles sous forme de graphes RDF virtuels. Ces graphes virtuels peuvent être matérialisés, générant des triplets RDF qui peuvent être utilisés avec des triplestore RDF, ou ils peuvent également être conservés virtuels et interrogés uniquement pendant l'exécution de la requête. L'approche virtuelle évite les coûts de matérialisation et permet de profiter de la maturité de plus de 40 ans des systèmes relationnels (par exemple, réponse aux requêtes efficace, sécurité, . . .).

L'une des approches les plus prometteuses pour répondre aux requêtes via un graph RDF virtuel est la réponse aux requêtes par réécriture de requêtes, c'est-à-dire la traduction de la requête SPARQL d'origine en une requête SQL équivalente. Cette requête SQL est ensuite déléguée au SGBD pour exécution. Afin d'utiliser les avantages fournis par le SGBD, la technique de réécriture des requêtes doit produire des requêtes SQL «raisonnables», c'est-à-dire ni trop grandes ni trop complexes pour être efficacement optimisées par le moteur de base de données. Ainsi, la technique de réécriture de requêtes doit s'attaquer à deux problèmes différents : (i) un problème de traduction de requête qui implique des mappings RDB à RDF sur des schémas relationnels arbitraires, et (ii) un problème d'optimisation de requête. Il existe un certain nombre de systèmes et de techniques liés à ce problème. Cependant, chacune de ces approches a des limites qui affectent les aspects critiques de la réponse aux requêtes via le graph RDF virtuel. Ces limitations incluent la génération de requêtes SQL inefficaces ou même incorrectes, le manque de fond formel et des implémentations médiocres. De plus, certains d'entre eux ne prennent pas en charge les schémas de base de données arbitraires car ils ne prennent pas en charge les langages de mapping RDB vers RDF, tels que R2RML. (Rodriguez-Muro & Rezk 2015)

En théorie, Pour répondre à une requête SPARQL, le système OBDA la reformule en une requête SQL, à évaluer par le SGBD. Une telle requête SQL peut être obtenue en (1) traduisant la requête SPARQL en une expression d'algèbre relationnelle sur le triplet de la relation du graphe RDF, puis (2) remplaçant les occurrences de triplet par les définitions correspondantes dans le mapping, Nous notons que, en général, l'étape (1) comprend également la réécriture de la requête de l'utilisateur par rapport à l'ontologie donnée. Incluant l'algorithme de traduction.

L'un des problèmes les plus difficiles dans une telle approche est la traduction des requêtes SPARQL en algèbre relationnelle et SQL. Les premières tentatives (Chebotko *et al.* 2009) de traduction SPARQL-to-SQL, bien que réussies, ont révélé de sérieuses difficultés liées à l'exactitude et

à l'efficacité d'une telle traduction.

Nous sommes intéressés par les requêtes SPARQL contenant l'opérateur OPTIONAL imbriqué introduit pour traiter les informations manquantes, servant ainsi un objectif similaire à l'opérateur LEFT (OUTER) JOIN dans les bases de données relationnelles. Le graph pattern P1 OPTIONAL P2 (renvoie les réponses à P1 étendu (si possible) par les réponses à P2 ; lorsqu'une réponse à P1 n'a pas de correspondance dans P2 (en raison d'affectations de variables incompatibles), les variables qui se produisent uniquement dans P2 restent non liées (LEFT JOIN étend un tuple sans correspondance avec NULL). L'objectif de ce travail est le traitement efficace des requêtes avec OPTIONAL imbriqué dans le paramètre OBDA. Ce problème est important dans la pratique car (a) OPTIONAL est très fréquent dans les requêtes SPARQL réelles, (b) c'est une source de complexité de calcul : l'évaluation des requêtes est difficile pour le fragment avec OPTIONAL seul, (c) contrairement aux requêtes SQL écrites par des experts, les traductions SQL des requêtes SPARQL ont tendance à avoir plus de LEFT JOIN avec une structure plus complexe, que les SGBDs peuvent ne pas optimiser correctement. Nous illustrons maintenant la différence de structure avec un exemple.

Exemple 5.4 : Requête SPARQL avec OPTIONAL imbriqué

```
01 SELECT   ?stu   ?adv   ?coadv
02 WHERE {
03      ?stu   rdf :type   :GradStudent .
04      OPTIONAL {
05      ?stu   :hasAdvisor   ?adv .
06          OPTIONAL {
07              ?stu   :hasCoadvisor   ?coadv .
} } }
```

La Requête SPARQL renvoie : (1) chaque étudiant diplômé, (2) le conseiller (Advisor) de l'étudiant si cette information est disponible ; et (3) le co-conseiller (Coadvisor) de l'étudiant si cette information est disponible et si le conseiller de l'étudiant a été récupéré avec succès à l'étape précédente. En d'autres termes, la requête renvoie les étudiants et autant de conseillers que possible ; il ne sert à rien de retourner un co-conseiller s'il n'y a même pas de conseiller pour un étudiant.

La requête à trois variables : ?stu pour l'étudiant, ?adv pour le conseiller et ?coadv pour le co-conseiller. Il existe deux clauses Optional, la plus interne étant la clause Optional imbriquée.

La clause WHERE de cet exemple contient une partie non–optional et deux parties optional. La partie non-optional (sur la ligne 3) est le graph pattern de base défini par un triple pattern. Ce dernier cherche les étu-

diants diplômés. La partie Optional comprend deux clauses Optional et n'a pas besoin de correspondre pour que la requête réussisse. La première clause Optional de la ligne 04 cherche le conseiller de l'étudiant (si cette information existe. La deuxième clause Optional de la ligne 06 cherche le co–conseiller, ssi cette information est disponible et si le conseiller de l'étudiant a été récupéré avec succès.

L'exemple ci–dessus illustre plusieurs défis du traitement des requêtes SPARQL avec la présence de modèles Optional avec la complexité de la sémantique des variables partagées, et des clauses Optional imbriquées.

— **Sémantique de base des patterns Optional :** L'évaluation d'une clause Optional n'est pas obligée de réussir, et en cas d'échec, aucune valeur ne sera retournée pour ces variables non liées dans la clause Select.

— **Sémantique des variables partagées dans les patterns Optional :** En général, les variables partagées doivent être liées aux mêmes valeurs. Les variables peuvent être partagées entre les sujets, les prédicats, les objets et entre eux.

— **Sémantique des patterns Optional imbriqué :** Avant d'évaluer une clause Optional imbriquée, toutes les clauses contenant Optional doivent avoir réussi.

5.3.4 Nouvelle Approche de réécriture SPARQL–to–SQL

La conclusion naturelle est que des travaux sur la formalisation de la sémantique de SPARQL sont nécessaires. Une approche formelle de ce sujet est bénéfique pour plusieurs raisons, notamment pour servir d'outil pour identifier les notions redondantes et contradictoires, et pour étudier la complexité, l'expressivité et d'autres comme la réécriture et optimisation. À notre connaissance, il n'y a pas de travail aujourd'hui pour aborder systématiquement cette formalisation. Il existe des propositions concernant des aspects de la sémantique de certains fragments de SPARQL. Il existe également des travaux sur les problèmes formels de la sémantique des langages de requête pour RDF qui peuvent être utiles pour SPARQL. En fait, le traitement SPARQL du point de vue de la complexité syntaxique, sémantique, algorithmique et computationnelle, fait l'objet de ce travail.

Nous proposons une nouvelle approche de réécriture basé sur le traitement de l'Algèbre SPARQL (Lehireche & Malki 2019), la solution traite toutes les questions susmentionnées. Tout d'abord, Nous présentons un nouvel algorithme de traduction pour traduire les requêtes SPARQL de type OPTIONAL imbriqué en requêtes SQL. Il utilise des fichiers

R2RML pour définir le mapping. L'Algorithme vise à générer des requêtes optimisées. Nous utilisons des optimisations comme la sélection des candidats et l'optimisation de l'auto-jointure (comme le système -–ontop (Calvanese *et al.* 2017)). La principale différence est la façon dont nous traitons l'algèbre SPARQL. Nous utilisons une algèbre modifiée qui contient les informations de mapping. Par conséquent, nous pouvons effectuer certaines optimisations avant qu'une requête ne soit transformée en une forme relationnelle. Grâce à cela, la requête SPARQL est sous une forme mieux adaptée à la transformation. Et les optimisations sur le modèle relationnel sont plus efficaces. Deuxièmement, la requête est traduite dans la forme relationnelle et d'autres optimisations sont effectuées. Une fois l'expression relationnelle optimisée, l'étape finale consiste à l'exécuter sur la base de données relationnelle.

Notre transformation fonctionne avec l'algèbre SPARQL basée sur l'algèbre officielle du W3C (Harris *et al.* 2013), nous traitons le modèle optionnel différemment. Ainsi, nous définissons précisément la sémantique de SPARQL en détectant les problèmes sémantiques de la requête SPARQL avec l'opérateur Optional imbriqué. Puis en cherchant à définir des techniques d'optimisation sur l'algèbre SPARQL.

Notre objectif est de traiter le pattern optionnel imbriqué à l'aide de la clause LEFT JOIN traditionnelle de l'algèbre relationnelle.

La raison pour laquelle il existe plusieurs approches du pattern Optional imbriqué est le fait que l'évaluation de ce type de requête peut dépendre d'une variable définie ailleurs, plus précisément, nommée variables partagées. En général, les variables partagées doivent être liées aux mêmes valeurs. Les variables peuvent être partagées entre les sujets, les prédicats, les objets et entre eux. (Exemple 5.5)

Exemple 5.5 : Problème de partage de variables dans l'optional imbriqué (Lehireche & Malki 2019)

```
SELECT  ?book  ?title  ?edit  ?address
   WHERE {
   ?book  : title  ?title.
      OPTIONAL {?book  : editor  ?edit.
         OPTIONAL {?book  : address  ?title . }
}
}
```

: title « apprendre à programmer »
: editor « éditions eyrolles »
: address « 61, bd Saint–Germain 75240 Paris Cedex 05 »

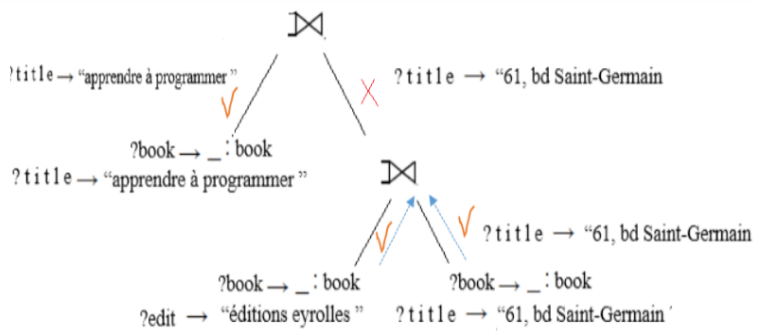

FIGURE 5.3 – *OPTIONAL imbriqués avec traitement relationnel (la jointure supérieure échoue).*

Le problème illustré sur l'exemple 5.5 se produit dans un cas où deux Optional sont imbriqués et une variable est utilisée à la fois à l'extérieur des Optional et à l'intérieur de la variable interne, mais pas dans celle externe. La question est : si ?title ne peut être lié qu'à deux valeurs différentes, laquelle des jointures de gauche échouera ? La question est importante car ?edit sera lié aux "éditions eyrolles" si la jointure inférieure échoue, et non lié si la jointure supérieure échoue.

Selon le modèle relationnel présenté ici, celui du haut échouera. Lors de l'évaluation de la jointure inférieure, il n'y a pas de conflit et le titre sera lié à «61, bd Saint-Germain 75240 Paris Cedex 05». À la jointure supérieure, il y a un conflit entre les deux valeurs différentes liées à ? title, et donc la jointure échouera et le résultat sera le tuple de la relation de gauche (figure 5.3).

Selon la sémantique SPARQL actuelle, celle du bas échouera. L'OPTIONAL interne échouera parce que ?title a «déjà été lié plus tôt» à «apprendre à programmer» et ne peut plus être lié à quelque chose de différent. L'OPTIONAL externe ne provoque pas de conflit alors et se liera ?edit (figure 5.4).

Ce problème complique la transformation de la requête SPARQL. Notre objectif est de mettre la requête sous une forme où l'algèbre peut être évaluée de bas en haut et chaque variable est dans la portée lors du traitement récursif de l'opérateur. Comme nous l'avons déjà montré. Le problème de la requête optional imbriquée avec les variables partagées. Il reste maintenant à définir l'algèbre sans le problème mentionné, ce qu'on appelle une requête sécurisée, et nous montrerons ensuite comment chaque requête peut être transformée en une requête sûre équivalente.

Par souci de simplicité, les termes ont été remplacés comme suit :
?a : ?book, ?b : ?title, ?c : ? edit.
P1 : title, P2 : edit, P3 : adress.

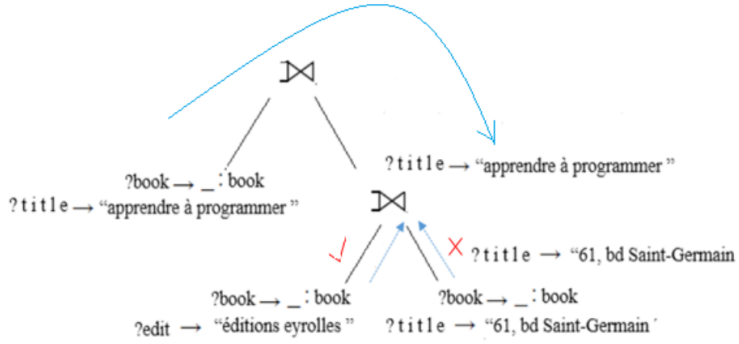

FIGURE 5.4 – *OPTIONAL imbriqués avec traitement SPARQL.*

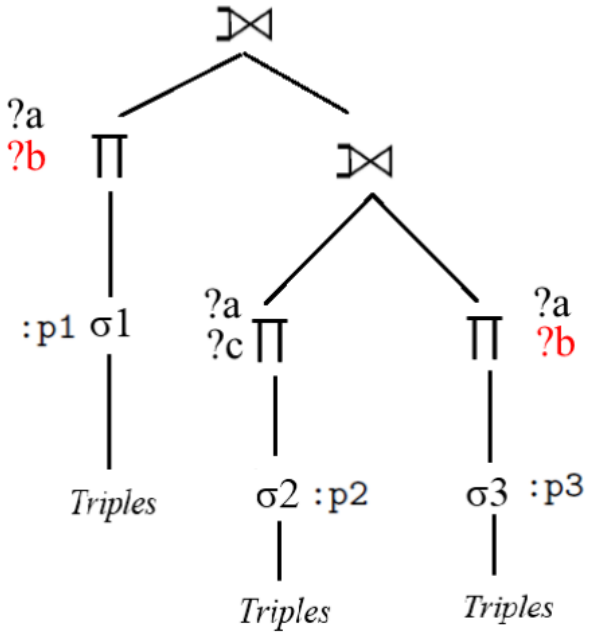

FIGURE 5.5 – *Algèbre SPARQL simplifiée pour trouver la formule de requête sécurisée.*

La requête de l'exemple 5.5 est traduite en arbre algébrique (dans la Figure 5.5)

La requête est dans une forme sécurisée lorsqu'aucune instance de ce problème n'est présente dans la requête. Formellement, la requête est dans une formule sécurisée lorsqu'il n'y a pas d'instance de la formule suivante dans l'arbre algébrique. $P_1 \rtimes \sigma 1 (P_3 \rtimes \sigma 2\ P_2)$

Ensuite, nous fusionnons cette solution dans l'algorithme du système Ontop (Calvanese *et al.* 2017). Cet algorithme présenté en dessous est une version simplifiée de la traduction. Il parcourt les feuilles de l'arbre d'algèbre SPARQL de façon ascendante ; et il passe par la liste S de nœuds dans l'arbre de requête Q dans l'ordre ascendant. De plus, l'algorithme commence par remplacer chaque feuille de l'arbre, qui est un triple pattern de la forme (s ; p ; o), par l'union des requêtes SQL définissant son

prédicat dans le mapping (lignes 7-–8). Dans cette étape, une fois qu'il a fini de traiter les feuilles, il continue vers les niveaux supérieurs de l'arbre (lignes 10–23), il continue jusqu'aux niveaux supérieurs de l'arborescence (lignes 10 à 23), où les opérateurs SPARQL (JOIN, OPTIONAL, UNION, FILTER et PROJECT) sont traduits en opérateurs SQL correspondants (InnerJoin, LeftJoin, Union, Filter et Project, respectivement). Une fois la racine traduite, le processus est terminé et l'expression SQL résultante est renvoyée.

Pour le traitement de requête optionnel imbriqué se trouve dans (lignes 14–16), où l'algorithme dans ce cas vérifie si la variable de type variable objet dans Optional externe se produit également dans OPTIONAL interne avec deux valeurs différentes, comme indiqué dans l'exemple 5.5 (la variable ici est ?title). Ce problème existe, nous remplaçons la formule de l'algèbre par celle définies pour avoir une requête sécurisée.

Algorithme de Traduction de requête SPARQL-to-SQL avec l'Optional imbriqué (Lehireche & Malki 2019)

Entrée : Q : requête SPARQL
M : mappings,
c : Optional Join Condition
v : Variable dans le motif du triplet
Sortie t : expression SQL
1 : S : liste des nœuds dans Q par ordre ascendant
2 : C : liste des conditions de jointure pour l'optional
3 : V : liste des variables dans le motif de triplet (nœud)
4 : sql empty map des nœuds aux expressions SQL
5 : pour nœud n à S faire
6 : pour chaque variable distincte v dans n faire
7 : si n is motif de triplet (triple pattern) alors traduction des feuilles
8 : sql[n] remplace–map–def(n,M)
9 : sinon /* traduction de nœuds non-feuilles */
10 : si n = JOIN(n1,n2) alors
11 : sql[n] InnerJoin(sql[n1] ; sql[n2])
12 : sinon si n = OPTIONAL(n1 ; n2 ; c) alors
13 : sql[n] LeftJoin(sql[n1] ; sql[n2] ; c)
14 : sinon si n = OPTIONAL(n1 ; OPTIONAL(n2 ; n3 ; c2) ; c1)
15 : et si v est une variable de prédicat dans n1 et se produit également dans n3 (interne) avec deux valeurs différentes, alors / * effectue la requête sous la forme sûre
16 : sql[n] LeftJoin(sql[n1] ; LeftJoin(sql[n2] ; sql[n3] ; c1))
17 : sinon si n = UNION(n1 ; n2) Alors
18 : sql[n] Union(sql[n1] ; sql[n2])
19 : sinon si n = FILTER(n1 ; e) alors
20 : sql[n] Filter(sql[n1] ; e)
21 : sinon si n = PROJECT(n1 ; p) alors
22 : sql[n] Project(sql[n1] ; p)
23 : fin si
24 : fin si
25 : fin pour
26 : fin pour
27 : retourne sql[S :last()]

5.3.5 Implémentations et expérimentations

Afin de vérifier l'efficacité de nos algorithmes de traduction, nous avons réalisé un ensemble d'expériences basé sur une par-

tie des requêtes SPARQL de type Optional imbriqué du Benchmark BSBM.(Bizer & Schultz 2009)

L'expérimentation nécessite les éléments suivants : une base de données relationnelle, une ontologie, un mapping de la base de données vers des ontologies et une liste de requête de type Nested Optional pattern. Et pour implémenter tout ça, nous utilisons les logiciels suivants :

MySQL[6] est un système de gestion de bases de données relationnelles (SGBDR). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, PostgreSQL et Microsoft SQL Server.

MySQL est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multi-thread et multi-utilisateur.

MySQL supporte deux langages informatique, le langage de requête SQL et le SQL/PSM (Persistent Stored Modules), une extension procédurale standardisée au SQL incluse dans la norme SQL : 2003. SQL/PSM.

IntelliJ IDEA[7] également appelé « IntelliJ », « IDEA » ou « IDJ » est un environnement de développement intégré (en anglais Integrated Development Environment – IDE) de technologie Java destiné au développement de logiciels informatiques. Il est développé par JetBrains, et disponible en deux versions, l'une communautaire, open source, sous licence Apache 2 et l'autre propriétaire, protégée par une licence commerciale. Tous deux supportent les langages de programmation Java, Kotlin, Groovy et Scala.

RDF4J[8] (OpenRDF Sesame) est un framework open source pour le stockage, l'interrogation et l'analyse des données RDF. Il a été créé par la société néerlandaise de logiciels Aduna dans le cadre de "On-To-Knowledge", un projet Web sémantique qui s'est déroulé de 1999 à 2002. Il contient les implémentations d'un triplestore en mémoire et d'un triplestore sur disque, ainsi que deux des packages de servlet distincts qui peuvent être utilisés pour gérer et fournir l'accès à ces triplestores, sur un serveur permanent. Le package RDF4J Rio (entrée / sortie RDF) contient une API simple pour les analyseurs et rédacteurs RDF basés sur Java. Les analyseurs et les écrivains pour les sérialisations RDF populaires sont distribués avec RDF4J, et les utilisateurs peuvent facilement étendre la liste

6. https ://www.mysql.com/
7. https ://blog.jetbrains.com/idea/tag/intellij-idea-2020-1/
8. https ://rdf4j.org/

en plaçant leurs analyseurs et écrivains sur le chemin de classe Java lors de l'exécution de leur application.

Ontop [9] est un système de graphe de connaissance virtuel. Il expose le contenu de bases de données relationnelles arbitraires sous forme de graphe de virtuel. Ce qui signifie que les données restent dans les sources de données au lieu d'être déplacées vers une autre base de données.

Ontop traduit les requêtes SPARQL exprimées sur les graphes de connaissances en requêtes SQL exécutées par les sources de données relationnelles. Il s'appuie sur les mappings R2RML et les ontologies. Le système prend en charge le SPARQL 1.0, des SGBD gratuits et commerciaux, il est Intégré à Protege 4.x, Compatible avec les outils de fédération de SGBD, un SPARQL endpoint.

Berlin SPARQL Benchmark (BSBM) [10] définit une suite de références pour comparer les performances des systèmes à travers leurs architectures. Le benchmark est construit autour d'un cas d'utilisation du commerce électronique dans lequel un ensemble de produits est proposé par différents fournisseurs et les consommateurs ont publié des avis sur les produits. Le mélange de requêtes du benchmark illustre le pattern de recherche et de navigation d'un consommateur à la recherche d'un produit.

5.3.6 Evaluation de résultats

Dans cette section, nous présentons l'évaluation des performances de l'algorithme proposés et des requêtes SQL résultante.

L'objectif des expériences est de vérifier l'efficacité de notre approche, à savoir les requêtes SQL résultantes, et l'exécution de la requête SPARQL sur la base de données relationnelle. Par conséquent, l'évaluation compare le temps d'exécution de notre algorithme avec le système Ontop.

Le temps d'exécution de la requête complète peut être divisé en trois parties :

1. Génération de la requête SQL
2. Exécution de la requête
3. Transformer les résultats SQL en termes RDF

Notre Algorithme produit des requêtes très similaires à celles produites par Ontop. Ainsi, la différence des performances se situe dans la première et la deuxième partie ou la génération de l'expression relationnelle est plus optimisée grâce au traitement préalable de l'algèbre SPARQL. La raison pour laquelle notre approche est plus performante est que la génération de requêtes est plus optimisée.

9. https ://ontop-vkg.org/
10. http ://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

TABLE 5.1 – *le temps global d'exécution des requêtes*

Requête	Système Ontop	Notre Approche
Q1 (simple)	1002 ms	1002 ms
Q2 (2 optional)	2014 ms	1070 ms
Q3 (3 optional)	4011 ms	3800 ms
Q4 (2 optional avec Filtre)	4122 ms	4030 ms

Les principaux résultats de nos expériences peuvent être résumés comme suit.

— Les temps d'exécution confirment que les optimisations sur l'algèbre SPARQL sont efficaces pour tous types de requêtes. Toutes les traductions optimisées affichent des performances comparables à celles du système Ontop. La plupart d'entre elles peuvent être évaluées en moins 5 secondes.

— Notre traduction optimisée est même légèrement plus efficace que les approches basé sur les optimisations sur le model relationnel.

5.4 Conclusion

La quantité croissante de données RDF sur le Web entraîne le besoin d'une gestion efficace et opérationnelle. Dans cette optique, de nombreux chercheurs ont proposé d'utiliser les SGBDRs pour stocker et interroger les données RDF en utilisant les langages de requête SQL et SPARQL. Les premières tentatives de traduction de SPARQL vers SQL ont révélé des défis pas très évidents liés à l'exactitude et à l'efficacité d'une telle traduction en présence de modèles de graphes optionnels imbriqués.

Dans ce travail, nous avons présenté une nouvelle approche de réécriture des requêtes pour produire des requêtes SQL «raisonnables», c'est–à–dire ni trop grandes ni trop complexes pour être efficacement optimisées par le moteur de base de données. Ainsi, cette technique s'attaque à deux problèmes différents : (i) un problème de mapping RDB-to-RDF sur des schémas relationnels, et (ii) un problème d'optimisation de requête. L'approche est essentielle pour les requêtes SQL obtenues en traduisant SPARQL dans le contexte d'OBDA avec des mappings et une ontologie.

Tout d'abord, nous avons présenté une approche de conception d'algorithmes pour générer automatiquement des documents de mapping qui reflètent le comportement de la spécification R2RML, et qui prend un schéma RDB comme entrée (contient toutes les informations sur le schéma

RDB), qui sera applicable comme support de base générant les triplets RDF à partir de données RDB. Deuxièmement, nous avons conçu et implémenté un algorithme pour l'optimisation de la réécriture des requêtes SPARQL avec des modèles de graphes optionnels imbriqués complexes en utilisant le traitement de son algèbre SPARQL afin de donner une définition optimale de la sémantique. Du point de vue Web Sémantique, nos optimisations exploitent les problèmes sémantiques du SPARQL, à savoir le problème de partage de variables dans la requête avec optional imbriqué. Nous pensons que de telles techniques n'ont pas été développées car les LEFT JOINs et / ou les conditions complexes comme les optional imbriqués ne sont pas introduites pour le traitement de l'algèbre SPARQL.

Les résultats expérimentaux montrent des facteurs importants pour la construction des mappings RDB-to-RDF et leurs influences sur le temps de génération du mapping et la taille des fichiers de mapping. Ces résultats reflètent ensemble l'efficacité de notre approche d'optimisation et sa mise en œuvre dans le contexte OBDA.

Nous travaillons sur la mise en œuvre de cette approche basée sur le traitement de l'algèbre SPARQL, pour prendre en charge des fragments plus importants de SPARQL (par exemple, agrégation, négation et path queries) et R2RML (par exemple, named graphs). Et étendre l'évaluation sur d'autre Benchmark.

Conclusion générale et perspectives

L'explosion continue des données RDF ouvre la porte à de nouvelles innovations en matière de big data, d'analyse des réseaux sociaux et d'initiatives Web sémantique, qui peuvent être partagées et réutilisées à travers différentes application, de l'entreprise et de la communauté. Le web sémantique est l'un des domaines de recherche les plus importants qui visent à construire un réseau de données basé sur le modèle de données RDF. Les bases de données relationnelles (RDB) sont les principales sources de données Web, la raison principale est que l'une des études a montré que les bases de données accessibles sur Internet contenaient jusqu'à 500 fois plus de données que le Web statique, et qu'environ 70% des sites Web sont soutenus par des RDB. Le groupe de travail RDB2RDF du W3C a récemment recommandé une spécification des langages pour mapper RDB (données et schémas) vers RDF et OWL.

Dans le cadre de l'Ontology-Based Data Access (OBDA), les requêtes sont posées sur une couche conceptuelle, puis traduites en requêtes sur la couche de données. La couche conceptuelle est donnée sous la forme d'une ontologie qui définit un vocabulaire partagé, et la couche de données est sous la forme d'une ou plusieurs sources de données existantes. Dans ce contexte, le modèle de données le plus répandu pour la couche conceptuelle et son langage de requête correspondant sont RDF et SPARQL. Aujourd'hui, la plupart des données d'entreprise sont stockées dans des bases de données relationnelles, il est donc crucial que les systèmes OBDA prennent en charge les mappings RDB-to-RDF. Le nouveau standard W3C pour les mappings RDB-to-RDF, R2RML, a été créé dans ce but.

R2RML est un langage de mapping personnalisé, qui permet aux utilisateurs de définir les mappings manuellement. L'utilisateur expert exprime le schéma RDB en utilisant une ontologie cible existante afin de convertir les données relationnelles en ensembles de données RDF. Ces données RDF sont exposées sous forme de graphes RDF virtuels. Ces graphes virtuels peuvent être matérialisés, générant des triplets RDF qui peuvent être utilisés avec des triplestores RDF, ou ils peuvent également

être conservés virtuels et interrogés uniquement pendant l'exécution de la requête. L'approche virtuelle évite le coût de la matérialisation et (peut) permettre de profiter de la maturité de plus de 40 ans des systèmes relationnels (par exemple, réponse aux requêtes efficace, sécurité).

Le composant le plus complexe de cette approche est la réécriture de requêtes SPARQL vers SQL pour répondre aux requêtes sur un graphe RDF virtuel, qui signifie la traduction de la requête SPARQL d'origine en une requête SQL équivalente. Cette requête SQL est ensuite exécutée sur un SGBD. Afin d'utiliser les avantages fournis par le SGBD, la technique de réécriture des requêtes doit produire des requêtes SQL «raisonnables», c'est-à-dire ni excessivement grandes ni trop complexes pour être efficacement optimisées par le moteur de base de données. Ainsi, la technique de réécriture de requêtes doit s'attaquer à deux problèmes différents : (i) un problème de traduction de requête qui implique des mappings RDB-to-RDF sur des schémas relationnels, et (ii) un problème d'optimisation de requête.

Il existe un certain nombre de systèmes et de techniques liés à ces problèmes, tels que ceux décrits dans le chapitre 4. Cependant, chacune de ces approches a des limites qui affectent les aspects critiques de la réponse aux requêtes sur un graphe RDF virtuel. Ces limitations incluent la génération de requêtes SQL inefficaces ou même incorrectes, le manque de fond formel et des implémentations médiocres. De plus, certains d'entre eux ne prennent pas en charge les schémas de base de données arbitraires car ils ne prennent pas en charge les langages de mapping RDB-to-RDF, tels que R2RML.

Dans cette thèse, nous avons présenté une nouvelle approche de traduction de requête SPARQL-to-SQL. Pour répondre dans un premier temps à la problématique du mapping RDB-to-RDF, en proposant un ensemble d'algorithmes pour générer automatiquement des documents de mapping qui reflètent le comportement de la spécification R2RML, et qui prend un schéma RDB comme entrée (contient toutes les informations sur le schéma RDB), qui sera applicable comme support de base générant les triplets RDF à partir de données RDB.

Deuxièmement, pour répondre au problème d'optimisation de la réécriture, nous avons proposé une approche basée sur le traitement d'algèbre SPARQL qui définit la sémantique d'une exécution de requête SPARQL. L'expression algébrique est formée à partir de la chaîne de requête. Le résultat d'une requête peut alors être calculé par les règles d'évaluation appliquées à l'expression algébrique. Cela donne les résultats corrects de la requête. Nous nous sommes focalisés particulièrement sur la définition de la sémantique de SPARQL pour détecter des problèmes sé-

mantiques, définir des techniques d'optimisation et évaluer la complexité du langage. La sémantique est définie par une algèbre sur des ensembles de mappings de variables.

L'un des problèmes les plus difficiles dans une telle approche est la traduction des requêtes SPARQL de type OPTIONAL imbriqué, et introduit pour traiter les informations manquantes, Ce problème est courant car : (a) OPTIONAL est très fréquent dans les requêtes SPARQL réelles, (b) c'est une source de calcul de complexité. L'évaluation des requêtes est difficile pour le fragment avec OPTIONAL seul, (c) contrairement aux requêtes SQL écrites par des experts, les traductions SQL des requêtes SPARQL ont tendance à avoir plus de LEFT JOIN avec une structure plus complexe, que les SGBDs peuvent ne pas optimiser correctement.

Notre approche consiste à traduire la requête SSPARQL avec le modèle optionnel imbriqué à l'aide de la clause LEFT JOIN traditionnelle de l'algèbre relationnelle. La raison pour laquelle il existe plusieurs approches du pattern OPTIONNEL est le fait que l'évaluation du modèle optionnel imbriquée peut dépendre d'une variable définie ailleurs, plus précisément, nommée variables partagées. En général, les variables partagées doivent être liées aux mêmes valeurs. Ce problème complique la transformation de la requête SPARQL. Notre objectif est de mettre la requête sous une forme où l'algèbre peut être évaluée de bas en haut et chaque variable est dans la portée lors du traitement récursif de l'opérateur. Comme nous l'avons déjà montré le problème de la requête optionnelle imbriquée avec les variables partagées. Il reste maintenant à définir l'algèbre sans le problème mentionné, ce qu'on appelle une requête sécurisée. La requête est dans une forme sécurisée lorsqu'aucune instance de ce problème n'est présente dans la requête. Ensuite, nous fusionnons cette solution dans l'algorithme du système Ontop (Calvanese *et al.* 2017). Cet algorithme est une version simplifiée de la traduction. Il parcourt les feuilles de l'arbre d'algèbre SPARQL de façon ascendante ; et il passe par les nœuds dans l'arbre. En plus, l'algorithme commence par remplacer chaque feuille de l'arbre, qui est un triple pattern par l'union des requêtes SQL définissant son prédicat dans le mapping. Dans cette étape, une fois qu'il a fini de traiter les feuilles, il continue vers les niveaux supérieurs de l'arbre, il continue jusqu'aux niveaux supérieurs de l'arborescence, où les opérateurs SPARQL (JOIN, OPTIONAL, UNION, FILTER et PROJECT) sont traduits en opérateurs SQL correspondants (InnerJoin, LeftJoin, Union, Filter et Project, respectivement). Une fois la racine traduite, le processus est terminé et l'expression SQL résultante est renvoyée.

Nous avons également évalué les performances de notre approche de traduction des requêtes SPARQL, Afin de vérifier l'efficacité de notre algo-

rithme de traduction, nous avons réalisé un ensemble d'expériences basé sur une partie des requêtes SPARQL de type Optional imbriqué du Benchmark BSBM. Notre approche s'est avérée performante par rapport aux anciennes approches basés sur cette traduction répond aux deux objectifs définis pour notre travail :

— du point de vue de la génération de la requête SQL, ou la génération de l'expression relationnelle est plus optimisée grâce au traitement préalable de l'algèbre SPARQL.

— du point de vue du temps d'exécution du processus de traduction, confirment que les optimisations sur l'algèbre SPARQL sont efficaces pour tous types de requêtes. Toutes les traductions optimisées affichent de performances comparables à celles du système Ontop.

Perspectives

Dans cette section, nous présentons les possibilités d'étendre et d'approfondir les recherches réalisées dans ce travail en vue d'un travail futur. Ce travail peut être divisé en perspectives à court terme et à long terme.

Perspectives à court terme

— étudier la performance des algorithmes de génération des documents de mapping avec le temps d'exécution sur plusieurs tailles différentes de bases de données.

— Il serait intéressant d'évaluer notre approche de traduction sur plusieurs SGBDs open source (par exemple PostgreSQL, SQL Server, Oracle XE), et faire une comparaison globale sur les performances. Et étendre les expériences sur un plus grand nombre de dataset du BSBM benchmark et élaborer l'approche sur plusieurs scénarios.

Perspectives à long terme

Nous cherchons à étendre l'implémentation et faire des experimentations poussées par rapport aux approches existantes pour prendre en charge l'ensemble de l'algèbre SPARQL. Ansi, prendre en charge des fragments plus importants de SPARQL (par exemple, agrégation, négation et path queries) et R2RML (par exemple, named graphs). De plus, il existe encore quelques options pour améliorer le processus de transformation, de sorte que la requête sera générée plus rapidement et même légèrement plus optimisée.

Bibliographie

[Arenas & Pérez 2011] Marcelo Arenas et Jorge Pérez. *Querying semantic web data with SPARQL*. In Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, New York,United States, pages 305–316, 2011.

[Auer *et al.* 2009] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann et David Aumueller. *Triplify : light-weight linked data publication from relational databases*. In Proceedings of the 18th international conference on World wide web, Madrid, Spain, pages 621–630, 2009.

[Beckett & Grant 2003] Dave Beckett et Jan Grant. *Swad-europe deliverable 10.2 : Mapping semantic web data with rdbmses*. W3C Semantic Web Advanced Development for Europe (SWAD-Europe), 2003.

[Berners-Lee *et al.* 1998] Tim Berners-Lee*et al. Semantic web road map*, 1998.

[Berners-Lee *et al.* 2001] Tim Berners-Lee, James Hendler et Ora Lassila. *The semantic web*. Scientific american, vol. 284, no. 5, pages 34–43, 2001.

[Bizer & Cyganiak 2006] Christian Bizer et Richard Cyganiak. *D2r server-publishing relational databases on the semantic web*. In Poster at the 5th international semantic web conference, volume 175, 2006.

[Bizer & Schultz 2009] Christian Bizer et Andreas Schultz. *The berlin sparql benchmark*. International Journal on Semantic Web and Information Systems (IJSWIS), vol. 5, no. 2, pages 1–24, 2009.

[Calvanese *et al.* 2011] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi et Domenico Fabio Savo. *The MASTRO system for ontology-based data access*. Semantic Web, vol. 2, no. 1, pages 43–53, 2011.

[Calvanese *et al.* 2017] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano

Rodriguez-Muro et Guohui Xiao. *Ontop : Answering SPARQL queries over relational databases*. Semantic Web, vol. 8, no. 3, pages 471–487, 2017.

[Chaloupka & Nečaskỳ 2016] Miloš Chaloupka et Martin Nečaskỳ. *Efficient SPARQL to SQL translation with user defined mapping*. In Proceeding of the International Conference on Knowledge Engineering and the Semantic Web, Prague, Czech Republic, pages 215–229. Springer, 2016.

[Charlet & Kembellec 2016] Jean Charlet et Gérald Kembellec. *Du web sémantique au web des données, quels enjeux professionnels ?* I2D Information, donnees documents, vol. 53, no. 2, pages 54–55, 2016.

[Chebotko *et al.* 2009] Artem Chebotko, Shiyong Lu et Farshad Fotouhi. *Semantics preserving SPARQL-to-SQL translation*. Data & Knowledge Engineering, vol. 68, no. 10, pages 973–1000, 2009.

[Chen *et al.* 2013] YING Chen, XIAOMING Zhao et SHIQING Zhang. *Publishing rdf from relational database based on d2R improvement*. WSEAS Transactions on Information Science and Applications, vol. 10, no. 8, pages 241–248, 2013.

[Cyganiak *et al.* 2012] R Cyganiak, C Bizer, O Maresch et C Becker. *The D2RQ mapping language vo. 8*, 2012.

[Dehainsala 2007] Hondjack Dehainsala. *Explicitation de la sémantique dans les bases de données : Base de données á base ontologique et le modéle OntoDB*. PhD thesis, 2007.

[DuCharme 2013] Bob DuCharme. Learning sparql : querying and updating with sparql 1.1. " O'Reilly Media, Inc.", 2013.

[Eisenberg & Kanza 2012] Vadim Eisenberg et Yaron Kanza. *D2RQ/update : updating relational data via virtual RDF*. In Proceedings of the 21st International Conference on World Wide Web, New York, United States, pages 497–498, 2012.

[Erling & Mikhailov 2010] Orri Erling et Ivan Mikhailov. *Virtuoso : RDF support in a native RDBMS*. In Semantic Web Information Management, pages 501–519. Springer, 2010.

[Frischmuth *et al.* 2012] Philipp Frischmuth, Jakub Klímek, Sören Auer, Sebastian Tramp, Jörg Unbehauen, Kai Holzweissig et Carl-Martin Marquardt. *Linked data in enterprise information integration*. Semantic Web, pages 1–17, 2012.

[Giese *et al.* 2015] Martin Giese, Ahmet Soylu, Guillermo Vega-Gorgojo, Arild Waaler, Peter Haase, Ernesto Jiménez-Ruiz, Davide Lanti, Martín Rezk, Guohui Xiao, Özgür Özçep*et al.* *Optique : Zooming in on big data*. Computer, vol. 48, no. 3, pages 60–67, 2015.

[Gruber 1995] Tom Gruber. *What is an Ontology*. International Journal of Human-Computer Studies, vol. 43, no. 4-5, pages 907–928, 1995.

[Guarino *et al.* 2009] Nicola Guarino, Daniel Oberle et Steffen Staab. *What is an ontology ?* In Handbook on ontologies, pages 1–17. Springer, 2009.

[Harris *et al.* 2013] Steve Harris, Andy Seaborne et Eric Prud'hommeaux. *SPARQL 1.1 query language*. W3C recommendation, vol. 21, no. 10, page 778, 2013.

[Hazber *et al.* 2016] Mohamed AG Hazber, Ruixuan Li, Guandong Xu et Khaled M Alalayah. *An approach for automatically generating R2RML-based direct mapping from relational databases*. In Proceeding of the International Conference of Pioneering Computer Scientists, Engineers and Educators, Changsha, China, pages 151–169. Springer, 2016.

[Heath & Bizer 2011] Tom Heath et Christian Bizer. *Linked data : Evolving the web into a global data space*. Synthesis lectures on the semantic web : theory and technology, vol. 1, no. 1, pages 1–136, 2011.

[Hert *et al.* 2011] Matthias Hert, Gerald Reif et Harald C Gall. *A comparison of RDB-to-RDF mapping languages*. In Proceedings of the 7th International Conference on Semantic Systems, Graz Austria, pages 25–32, 2011.

[Klímek & Necaskỳ 2011] Jakub Klímek et Martin Necaskỳ. *Generating lowering and lifting schema mappings for semantic web services*. In Proceeding of IEEE Workshops of International Conference on Advanced Information Networking and Applications, Washington, United States, pages 29–34. IEEE, 2011.

[Kontchakov *et al.* 2014] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao et Michael Zakharyaschev. *Answering SPARQL queries over databases under OWL 2 QL entailment regime*. In Proceeding of the International Semantic Web Conference, Berlin, Heidelberg, pages 552–567. Springer, 2014.

[Lefrançois & Zimmermann 2017] Maxime Lefrançois et Antoine Zimmermann. *Support uniforme de types de données personnalisés dans RDF et SPARQL.* In Proceeding of EGC, pages 321–326, 2017.

[Lehireche & Malki 2019] Nesrine Lehireche et Mimoun Malki. *Efficient Processing of SPARQL Optional query in an RDBMS*. In Proceeding of the International conference Innovation and New Trends in Information Technology(INTIS), Tangier, Morocco, pages 149–155, 2019.

[Lehireche *et al.* 2017] Nesrine Lehireche, Mimoun Malki, Ahmed Lehireche et Reda Mohamed Hamou. *On Demand ETL of RDB to RDF Mapping for Linked Enterprise Data*. International Journal of Strategic Information Technology and Applications (IJSITA), vol. 8, no. 3, pages 91–100, 2017.

[McGuinness *et al.* 2004] Deborah L McGuinness, Frank Van Harmelen*et al. OWL web ontology language overview*. W3C recommendation, vol. 10, no. 10, page 2004, 2004.

[Mendes *et al.* 2011] Pablo N Mendes, Max Jakob, Andrés García-Silva et Christian Bizer. *DBpedia spotlight : shedding light on the web of documents*. In Proceedings of the 7th international conference on semantic systems, New York, United States, pages 1–8, 2011.

[Miles & Bechhofer 2009] Alistair Miles et Sean Bechhofer. *SKOS simple knowledge organization system reference*. W3C recommendation, vol. 18, page W3C, 2009.

[Motik *et al.* 2014] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks et Dan Olteanu. *Parallel materialisation of datalog programs in centralised, main-memory RDF systems*. In Proceeding of Twenty-Eighth AAAI Conference on Artificial Intelligence, Berlin, Heidelberg. Citeseer, 2014.

[Nečaskỳ *et al.* 2012] Martin Nečaskỳ, Irena Mlỳnková, Jakub Klimek et Jakub Malỳ. *When conceptual model meets grammar : A dual approach to XML data modeling*. Data & Knowledge Engineering, vol. 72, pages 1–30, 2012.

[Pérez *et al.* 2009] Jorge Pérez, Marcelo Arenas et Claudio Gutierrez. *Semantics and complexity of SPARQL*. ACM Transactions on Database Systems (TODS), vol. 34, no. 3, pages 1–45, 2009.

[Priyatna *et al.* 2014] Freddy Priyatna, Oscar Corcho et Juan Sequeda. *Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph*. In Proceedings of the 23rd international

conference on World wide web, New York, United States, pages 479–490, 2014.

[Priyatna 2015] Freddy Priyatna. *Methods and Techniques for the Generation and Eficient Exploitation of RDB2RDF Mappings*. ETS de Ingenieros Informáticos (UPM), 2015.

[Prud'hommeaux 2008] Eric Prud'hommeaux. *SPARQL query language for RDF, W3C recommendation*. http ://www. w3. org/TR/rdf-sparql-query/, 2008.

[Rodriguez-Muro & Rezk 2015] Mariano Rodriguez-Muro et Martin Rezk. *Efficient SPARQL-to-SQL with R2RML mappings*. Journal of Web Semantics, vol. 33, pages 141–169, 2015.

[Rodriguez-Muro 2010] M Rodriguez-Muro. *Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics (Ph. D. thesis)*. KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, 2010.

[Sahoo *et al.* 2009] Satya S Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda et Ahmed Ezzat. *A survey of current approaches for mapping of relational databases to RDF*. W3C RDB2RDF Incubator Group Report, vol. 1, pages 113–130, 2009.

[Sequeda & Miranker 2013] Juan F Sequeda et Daniel P Miranker. *Ultrawrap : SPARQL execution on relational data*. Journal of Web Semantics, vol. 22, pages 19–39, 2013.

[Sequeda *et al.* 2009] Juan F Sequeda, Rudy Depena et Daniel P Miranker. *Ultrawrap : Using sql views for rdb2rdf*. Proc. of ISWC2009, 2009.

[Sequeda *et al.* 2011] Juan F Sequeda, Syed Hamid Tirmizi, Oscar Corcho et Daniel P Miranker. *Survey of directly mapping sql databases to the semantic web*. Knowledge Engineering Review, vol. 26, no. 4, pages 445–486, 2011.

[Sequeda *et al.* 2014] Juan F Sequeda, Marcelo Arenas et Daniel P Miranker. *OBDA : query rewriting or materialization ? In practice, both !* In Proceeding of the International Semantic Web Conference, Berlin, Heidelberg, pages 535–551. Springer, 2014.

[Sequeda 2016] Juan Federico Sequeda. Integrating relational databases with the semantic web, volume 22. IOS Press, 2016.

[Sheth & Meersman 2002] Amit Sheth et Robert Meersman. *Amicalola report : database and information systems research challenges and opportunities in semantic web and enterprises*. ACM SIGMOD Record, vol. 31, no. 4, pages 98–106, 2002.

[Spanos *et al.* 2012] Dimitrios-Emmanuel Spanos, Periklis Stavrou et Nikolas Mitrou. *Bringing relational databases into the semantic web : A survey*. Semantic Web, vol. 3, no. 2, pages 169–209, 2012.

[Strandhaug 2014] Marius Strandhaug. An r2rml mapping management api in java : Making an api independent of its dependencies. Master's thesis, 2014.

[Taelman *et al.* 2019] Ruben Taelman, Miel Vander Sande et Ruben Verborgh. *Bridges between GraphQL and RDF*. In W3C Workshop on Web Standardization for Graph Data. W3C, 2019.

[Tirmizi *et al.* 2008] Syed Hamid Tirmizi, Juan Sequeda et Daniel Miranker. *Translating sql applications to the semantic web*. In Proceeding of the International Conference on Database and Expert Systems Applications, Turin, Italy, pages 450–464. Springer, 2008.

[Unbehauen & Martin 2017] Jörg Unbehauen et Michael Martin. *SPARQL Update queries over R2RML mapped data sources*. INFORMATIK 2017, 2017.

[Unbehauen *et al.* 2013] Jörg Unbehauen, Claus Stadler et Sören Auer. *Optimizing SPARQL-to-SQL rewriting*. In Proceedings of International Conference on Information Integration and Web-based Applications & Services, Vienna, Austria, pages 324–330, 2013.

[Xiao *et al.* 2018] Guohui Xiao, Roman Kontchakov, Benjamin Cogrel, Diego Calvanese et Elena Botoeva. *Efficient handling of SPARQL optional for OBDA*. In Proceeding if the International Semantic Web Conference, MONTEREY, CALIFORNIA, USA, pages 354–373. Springer, 2018.

[Xu *et al.* 2006] Zhuoming Xu, Shichao Zhang et Yisheng Dong. *Mapping between relational database schema and OWL ontology for deep annotation*. In 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)Hong Kong, China, pages 548–552. IEEE, 2006.

Annexe

Base de données relationnelle

Car

RegNum	CarModel
JP2323	Hyundai Creta
GF4554	Peugeot 3008

Person

ID	Name	OwnsCar
1	Ali	GF4554
2	Majid	Null

Exemple 3.1 : Le mapping directe de la base de données relationnelle

```
1 @prefix rdf : <http : / /www.w3 . org /1999/02/22-rdf-syntax-ns#> .
2 @base : <http : / /www. example . org/> .
3
4 <Person /ID=1> rdf : type <Person>;
5          <Person#ID> 1;
6          <Person#name> " Ali ";
7          <Person#OwnsCar> " GF4554";
8          <Person#ref-OwnsCar> <Car/ reg= GF4554> .

9 <Person /ID=2> rdf : type <Person>;
10           <Person#ID> 2;
11           <Person#name> "Majid" .
12
13 <Car/ reg= GF4554> rdf : type <Car>;
14      <Car#reg> " GF4554 ";
15      <Car#CarModel> " Peugeot 3008" .
16 <Car/ reg= JP2323> rdf : type <Car>;
17      <Car#reg> " JP2323";
18      <Car#CarModel> " Hyundai Creta "
```

Exemple 3.2 : Exemple de Mapping R2RML de la même base de données ci-dessus.

```
@prefix rdf : <http : / /www.w3 . org /1999/02/22-rdf-syntax-ns#> .
@prefix r r : <http : / /www.w3 . org /ns / r2rml#> .
@prefix f o a f : <t tp : / / xmlns . com/ f o a f /0.1/> .
@prefix ex : <http : / /www. example . org/> .

ex :CarMap
        a r r : TriplesMap;
        r r : logicalTable [ r r : tableName "Car " ];
        r r : subjectMap [
         r r : template " http : / /www. example . org / car /RegNum";
         r r : c l a s s ex : Car;
         ];
         r r : predicateObjectMap [
             r r : predicate ex : hasReg;
             r r : objectMap [ r r : tableName "RegNum" ];
         ];
         r r : predicateObjectMap [
         r r : predicate ex : carModel;
         r r : objectMap [ r r : column "CarModel " ];
         ] .

ex : PersonMap
         r r : TriplesMap;
         r r : logicalTable [ r r : tableName " Person " ];
         r r : subjectMap [
             r r : template " http : / /www. example . org / person /ID";
         r r : class foaf : Person;
         ];
         r r : predicateObjectMap [
             r r : predicate foaf : name;
             r r : objectMap [ r r : column "name " ];
];
         r r : predicateObjectMap [
         r r : predicate ex : ownsCar;
         r r : objectMap [
             r r : RefObjectMap;
             r r : parentTriplesMap ex :CarMap;
         r r : joinCondition [
```

```
39              r r : child "OwnsCar ";
40              r r : parent "RegNum";
41              ];
42              ];
43              ] .
```

Exemple 3.3 Le graphe RDF résultant donné en appliquant le mapping de l'exemple 3.2

```
1 @prefix rdf : <http ://www.w3.org /1999/02/22-rdf-syntax-ns#> .
2 @prefix foaf : <http : //xmlns.com/foaf /0.1/> .
3 @prefix ex : <http : / /www. example . org/> .
4
5 ex : car / JP2323
6           a ex : Car;
7           ex : hasReg " JP2323 ";
8           ex : carModel " Hyundai Creta " .
9
10 ex : car / GF4554
11           a ex : Car;
12           ex : hasReg " GF4554 ";
13           ex : carModel " Peugeot 3008" .
14
15 ex : person /1
16           a foaf : Person;
17           foaf : name " Ali ";
18           ex : ownsCar ex : car / GF4554 .
19
20 ex : person /2
21           a foaf : Person;
22           foaf : name "Majid" .
```

Exemple 3.4 : Un exemple de mapping D2RQ pouvant être utilisé sur la base de données dans l'exemple 3.1.

```
1 @prefix rdf :<http ://www.w3 .org /1999/02/22-rdf-syntax-ns#>.
2 @prefix f o a f : <http ://xmlns.com/foaf/0.1/>.
3 @prefix ex : <http ://www.example.org/>.
4
5 ex :CarMap a d2rq : ClassMap;
6          d2rq : uriPattern "http ://www.example.org/reg=@@Car.reg@@";
7          d2rq : class ex : Car;
8          # Specifies a database connection( not covered here )
9          d2rq : dataStorage ex : Database1 .
10
11 ex : PersonMap a d2rq : ClassMap;
12           d2rq : uriPattern "http ://www.example.org/ID=@@Person.ID@@";
13           d2rq : class foaf : Person;
14      2rq : dataStorage ex : Database1 .
15
16 ex : CarOwner a d2rq : PropertyBridge;
17           d2rq : belongsToClassMap ex : PersonMap;
18           d2rq : property ex : ownsCar;
19           d2rq : refersToClassMap ex :CarMap;
20           d2rq : join " Person .OwnsCar => Car .RegNum"
```