Conception physique des bases de données à base ontologique : le cas des vues matérialisées

Bery Leouro Mbaiossoum

Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Laboratoire d'Informatique et d'Automatique pour les Systèmes

THESE

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AEROTECHNIQUE

(Diplôme National — Arrêté du 7 août 2006)

Ecole Doctorale : Sciences et Ingénierie pour l'Information, Mathématiques
Secteur de Recherche : INFORMATIQUE & APPLICATIONS

Présentée par :

MBAIOSSOUM BERY LEOURO

****

Conception physique des bases de données à base ontologique : le cas des vues matérialisées

****

Directeur de thèse : **Ladjel BELLATRECHE**
Co-encadrant : **Stéphane JEAN**

****

Soutenue le 12 décembre 2014
devant la Commission d'Examen

****

JURY

Rapporteurs :	**Zohra BELLAHSENE**	Professeur, Université Montpellier II
	Nadine CULLOT	Professeur, Université de Bourgogne
Examinateurs :	**Olivier TESTE**	Professeur, Université de Toulouse 2 Jean Jaurès
	Patrick MARCEL	Maître de Conférences/HDR, Université de Tours
	Ladjel BELLATRECHE	Professeur, ISAE-ENSMA, Poitiers
	Stéphane JEAN	Maître de Conférences, Université de Poitiers

Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Laboratoire d'Informatique et d'Automatique pour les Systèmes

THESE

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AEROTECHNIQUE

(Diplôme National — Arrêté du 7 août 2006)

Ecole Doctorale : Sciences et Ingénierie pour l'Information, Mathématiques
Secteur de Recherche : INFORMATIQUE & APPLICATIONS

Présentée par :

MBAIOSSOUM BERY LEOURO

****

Conception physique des bases de données à base ontologique : le cas des vues matérialisées

****

Directeurs de Thèse : **Ladjel BELLATRECHE & Stéphane JEAN**

****

Soutenue le 12 décembre 2014
devant la Commission d'Examen

****

JURY

Rapporteurs :	**Zohra BELLAHSENE**	Professeur, Université Montpellier II
	Nadine CULLOT	Professeur, Université de Bourgogne
Examinateurs :	**Olivier TESTE**	Professeur, Université de Toulouse 2 Jean Jaurès
	Patrick MARCEL	Maître de Conférences/HDR, Université de Tours
	Ladjel BELLATRECHE	Professeur, ISAE-ENSMA, Poitiers
	Stéphane JEAN	Maître de Conférences, Université de Poitiers

Remerciements

Au terme de cette recherche heuristique, je tiens à rendre grâce à Dieu tout Puissant de ce qui a permis à ce que j'atteigne ce très haut niveau d'études. Je tiens à exprimer toute ma profonde gratitude à tous ceux dont sans la participation active, cette recherche n'aurait pas vu le jour. Il s'agit de :

- Pr **Ladjel BELLATRECHE**, mon Directeur de thèse, pour sa disponibilité sans faille dans ses brillants conseils, orientations et échanges tout au long de cette thèse.
- Dr **Stéphane JEAN**, mon co-directeur, qui m'a marqué de sa disponibilité, sa confiance, son encadrement, sa promptitude à répondre à mes sollicitations durant tout le processus de la réalisation de ce travail. MM. BELLATRECHE et JEAN ont été très actifs dans la recherche de financement de cette thèse.
- Dr **ASLAOU Kobmbaye**, pour avoir bien accepté de m'accompagner dans mes activités dans mon pays le Tchad. Son soutien et ses conseils m'ont beaucoup servi.
- L'**Agence Universitaire de la Francophonie** dont la contribution se traduit par le financement de cette thèse,
- Pr **Emmanuel GROLLEAU**, Directeur du LIAS, pour m'avoir accueilli dans son laboratoire.
- Pr **Zohra BELLAHSENE** et **Nadine CULLOT** pour avoir accepté la lourde tâche d'être rapporteurs de cette thèse.
- Pr Olivier TESTE et Dr Patrick MARCEL pour avoir accepté de juger de la scientificité de cette recherche.
- Dr *Mickael BARON et Zoé FAGET* pour leurs contributions si efficaces dans les commentaires techniques lors des lectures de mes travaux.
- tous les membre du LIAS et à tous les doctorants présents pendant ces années de ma thèse (notamment *S. Khouri, B. Ahcene, K. Georges, B. Selma, S. Cheikh, L. Thomas, F. Geraud, B. Moustapha, B. Ilyes, B. Paule, B. Soumia, C. ChedLia, T. Valéry, B. Akli*) pour leurs amitiés et leurs collaborations sincères.
- *Hondjack Dehainsala* pour m'avoir encouragé à venir faire mes études à Poitiers.
- *NETOUA Sylvie*, mon épouse, pour la patience dont elle a fait montre tout au long de cette recherche.
- *Majorelle, Cyriac et Fortune* mes enfants qui, en dépit de mon éloignement de la maison nuptiale n'ont cessé d'exprimer leur amour à mon égard.
- *BERI Eléazar*, mon père et *BOUMBAYE Rodé*, ma mère pour m'avoir montré le chemin de l'école et leurs soutiens indéfectibles.
- tous les parents notamment *NEASKEM Jared, Namadji Leouro, Lalem Rachel, BEMBA Namadji Leouro, BENDIMAN Namadji, DJEMBER Philemon, NADWAI Philippe, DJOL DOG Urbain, MBAIBE Achime, MBAIBE Adolphe, MBAITELSEM Bery et DJEKOUNLA Malachie* pour leur marques de soutien multiforme.
- *NEKOULNANG Clarice, MONGLENGAR Serges, SADJINAN Mathieu, ALATAN, RA-*

HAMAT NOUBARANGAR, compatriotes doctorants avec qui, nous avons passé de bons et difficiles moments.
– toute la communauté tchadienne de Poitiers pour sa sympathie à mon endroit.
– tous les amis de Poitiers notamment *KABORE Samuel, RICHARD Marie-Christine, Stéphanie GIRAUD, Quentin* pour leur esprit d'amitié.

Je remercie enfin toutes les personnes qui se reconnaîtront par les divers soutiens dont j'ai fait l'objet.

A

Majorelle, Cyriac et Fortune, mes enfants

Table des matières

Partie II Contributions

Introduction générale

Ce chapitre préliminaire présente le contexte de notre thèse à savoir la diversité des bases de données définies pour stocker à la fois les ontologies et les données associées. Nous introduisons ensuite nos contributions qui portent sur la conception physique de telles bases de données en prenant en compte leur diversité.

1 Contexte et problématique

L'utilisation massive du Web et de l'Internet ont fait naître un besoin de partage, d'échange et d'analyse des objets et en particulier des données. Pour réaliser ces tâches, connaître le sens d'une donnée est devenu un *enjeu important* pour les entreprises, les gouvernements, les applications, les services, etc. Si nous nous penchons sur la technologie des bases de données dans sa première génération, les concepteurs construisaient des bases de données pour répondre à un besoin particulier, *mais sans se soucier si ces bases de données seraient intégrées et utilisées par des personnes en dehors de leur entreprise*. Lorsque cette dernière souhaite s'ouvrir et partager ses données, elle se heurte au problème d'hétérogénéité structurelle et sémantique des données. Afin de répondre à ce problème, les ontologies ont été largement utilisées pour expliciter le sens des concepts et des propriétés d'un domaine donné. Il n'y a pas que la technologie des bases de données qui a utilisé les ontologies, d'autres communautés l'ont également fait : *l'intelligence artificielle*, *le traitement de la langue naturelle*, etc.

Autour du concept d'ontologie, plusieurs travaux de recherche se sont concentrés sur : **(i)** la construction d'ontologies de domaine, **(ii)** la définition des formalismes de manipulation des ontologies, **(iii)** l'exploitation des ontologies et **(iv)** l'étude de son évolution. Ainsi, nous faisons les constats suivants :

1. Plusieurs ontologies de domaine ont été proposées et certaines normalisées, souvent dans les domaines où une *grande expertise existe*. On peut citer la médecine (le cas de l'ontologie Unified Medical Language System (UML)), l'ingénierie (IEC 61360-4 pour les composants électroniques, la norme ISO 23584 dans le domaine de l'optique et l'optro-

nique), l'environnement [1], les agences de voyage [1], les jeux vidéo [2], etc. De plus en plus de besoins de construction d'ontologie émergent.

2. La conceptualisation d'une ontologie nécessite des modèles formels pour la définir et pour offrir des mécanismes de raisonnement. Un nombre important de formalismes existent, que nous pouvons classer en deux types selon l'objectif visé : (a) des formalismes pour faciliter l'échange et le partage de données, où les concepts sont définis d'une manière unique (le cas de PLIB [2]), (b) des formalismes pour définir des correspondances entre vocabulaires et offrir des possibilités de déduction et d'inférence (le cas d'OWL [3]).

3. La maturité des ontologies a motivé plusieurs chercheurs et industriels à les utiliser pour répondre à plusieurs problématiques, comme l'indexation des documents, l'intégration et l'échange des données, le traitement de la langue naturelle, la conception des bases de données, etc. L'usage d'une ontologie apporte de nouveaux questionnements. Prenons le cas d'utilisation des ontologies pour concevoir des bases de données. Cette utilisation génère un gros volume de données qui doit être stocké, interrogé, optimisé, etc. Des solutions de persistance ont été proposées. Elles ont fait naître un nouveau type de bases de données, appelé *Base de Données à Base Ontologique* ($\mathcal{BDBO}$) ou bases de données sémantiques, qui stockent à la fois les instances et l'ontologie ou les ontologies décrivant leur sens. Plusieurs travaux ont été proposés pour offrir des modèles de stockage adéquats à ces données et à leur ontologie [4, 5, 6, 7, 8, 9]. Certains sont *hérités* du modèle de stockage traditionnel (table oriented storage), comme la représentation horizontale, où chaque classe de l'ontologie est stockée dans une table unique, d'autres sont *originaux* comme les représentations binaires et verticales (table de triplets). La présence d'une ontologie au sein d'un Système de Gestion de Base de Données (SGBD) a également contribué à l'évolution de l'architecture traditionnelle de ce dernier composée habituellement de la partie *contenu* et de la partie *méta-bas*e. Cette évolution vise à prendre en compte à la fois l'ontologie et son modèle appelé méta-schéma. Ce dernier a été proposé pour offrir un accès générique au modèle d'ontologie. Il joue, pour l'ontologie, le même rôle que celui joué par la méta-base pour les données. En ce qui concerne l'interrogation des données, plusieurs langages d'interrogation ont été proposés. On peut citer par exemple RDQL [10], SeRQL [11], RQL [12], Squish [13], SPARQL [14], OntoQL [15], etc. Le langage SPARQL a été retenu par le consortium W3C (World Wide Web Consortium) comme le langage standard pour les $\mathcal{BDBO}$. Ce langage ressemble au langage SQL et est souvent traduit dans ce dernier car la plupart des $\mathcal{BDBO}$ utilisent un SGBD Relationnel comme système sous-jacent.

4. Comme tout composant informatique, une ontologie évolue et nécessite donc une gestion de ses différentes versions. L'évolution d'une ontologie est normalement beaucoup plus problématique que les évolutions d'une base de données. Ceci est dû au caractère partageable de l'ontologie. Pour illustrer ce problème, citons Rogozan [16] qui s'interroge :

1. `https://bioportal.bioontology.org/ontologies/ENVO`
2. `http://www.mindmeister.com/fr/324669511/game-ontology-project`

"comment préserver l'accès et l'interprétation des objets au moyen de leur référencement à une ontologie évolutive" ?

Les constants précédents nous font remarquer que les ontologies sont devenues une réalité dans le paysage des bases de données et apportent une *forte diversité* qui a été suivie par les SGBD industriels. Si nous prenons l'exemple de Oracle (*Oracle Spatial and Graph*) et IBM (*SOR*), elles proposent des solutions différentes pour gérer les instances ontologiques. Cette différence concerne à la fois les modèles de stockage utilisés et l'architecture de leur SGBD cible. Cette diversité est au cœur des travaux présentés dans ce mémoire. Nous nous intéresserons en particulier à la conception physique de ce type de bases de données qui consiste à définir des structures d'optimisation prenant en compte cette diversité.

2 Motivations et Objectifs

La conception physique a eu un grand intérêt dans le contexte des bases de données décisionnelles, où une large panoplie de structures d'optimisation (les vues matérialisées, les index de jointure binaire, la compression, le partitionnement, le traitement parallèle, etc.) a été étudiée et supportée par les SGBD commerciaux et académiques. Durant la conception physique, une ou plusieurs structures d'optimisation sont sélectionnées. Le processus de sélection est souvent guidé par un modèle de coût mathématique estimant le coût des requêtes (souvent en termes d'entrées sorties) en présence de ces structures d'optimisation. La plupart des problèmes de sélection de structures d'optimisation est difficile, car l'espace de recherche lié à chaque structure peut être très large. Prenons le cas des vues matérialisées, toute requête ou partie d'une requête peut être candidate à la matérialisation. Étant donné que chaque structure d'optimisation a des contraintes comme l'espace de stockage et le coût de maintenance pour les vues matérialisées, on ne peut donc pas sélectionner toutes les vues candidates. En conséquence, nous faisons appel à des algorithmes d'exploration de l'espace de recherche, comme les algorithmes génétiques, le recuit simulé, les algorithmes glouton, etc. Pour quantifier la qualité de ces derniers, un modèle de coûts est utilisé. La question que nous nous posons dans nos travaux est comment transporter cette démarche traditionnelle sur les $\mathcal{BDBO}$?

Pour mener à bien notre réflexion, nous avons d'abord commencé par l'étude de la diversité des $\mathcal{BDBO}$ en analysant et comparant les approches existantes. Cette étude approfondie de différents types de $\mathcal{BDBO}$ nous a permis d'en proposer une formalisation unifiée des $\mathcal{BDBO}$. Cette formalisation prend en compte les différentes composantes engendrant la diversité : les modèles d'ontologie, les modèles de stockage, les architectures, etc. Cette formalisation nous sert comme cadre pour identifier les variables à utiliser dans un processus de sélection de structures d'optimisation.

La deuxième piste de recherche que nous avons explorée est la *définition d'un modèle de coût* pour évaluer la qualité de chaque type de $\mathcal{BDBO}$ en exécutant un ensemble de requêtes. Contrairement aux bases de données traditionnelles, où nous trouvons un nombre important de

modèles de coûts, peu existent dans le cas des $\mathcal{BDBO}$. Pour en définir un, nous avons d'abord suivi une démarche expérimentale afin d'extraire les paramètres pertinents de notre modèle de coûts. Nous avons alors considéré six exemples de $\mathcal{BDBO}$, trois industriels (Oracle, IBM SOR et DB2RDF) et trois académiques (Jena, Sesame et OntoDB - développée au sein de notre laboratoire). Ces systèmes sont divers, du fait que chacun a ses propres spécificités. Nous avons effectué une batterie de tests en utilisant le banc d'essai LUBM (*The Lehigh University Benchmark*) et ses requêtes. Par la suite un modèle de coût intégrant des paramètres issus de cette expérimentation a été développé pour évaluer toute $\mathcal{BDBO}$. Ce dernier est ensuite utilisé dans une approche que nous proposons pour sélectionner les vues matérialisées.

3 Contributions

Nos contributions à travers ce travail, sont quadruples.

Premièrement, nous avons établi un état de l'art sur les $\mathcal{BDBO}$. Une étude de leurs caractéristiques nous a permis de mettre en lumière la grande diversité de $\mathcal{BDBO}$ et de leurs usages. En effet il existe :

- différents usages d'ontologies (annotation, explicitation, intégration, échange de données, etc.), ce qui implique l'existence d'un nombre important de modèles d'ontologies (PLIB, RDFS, OWL, KIF, etc.);
- différents modèles de stockage (binaire, vertical, horizontal...) qui ont un impact sur la conception physique ;
- différents niveaux de modélisation (niveau données, niveau méta-données, niveau méta-ontologie, ...), ce qui conduit à des différentes architectures.

Deuxièmement, nous avons proposé un modèle unifiant les $\mathcal{BDBO}$ et un modèle de coûts générique qui intègrent les diversités remarquées.

Troisièmement, nous avons effectué une évaluation des performances de quelques $\mathcal{BDBO}$ selon deux critères : le temps de chargement des données et le temps de traitement de requêtes.

Enfin, nous avons exploité la présence de l'ontologie au sein de SGBD pour définir une méthodologie de sélection des vues matérialisées au niveau conceptuel. La présence de notre modèle de coûts nous a également aidé à définir une autre méthodologie pour sélectionner des vues prenant en compte la diversité des $\mathcal{BDBO}$.

Les approches que nous avons proposées, ont été validés théoriquement en utilisant le modèle de coûts et réellement sur un ensemble de $\mathcal{BDBO}$ comme Oracle, Sesame, Jena, IBM SOR, DB2RDF et OntoDB.

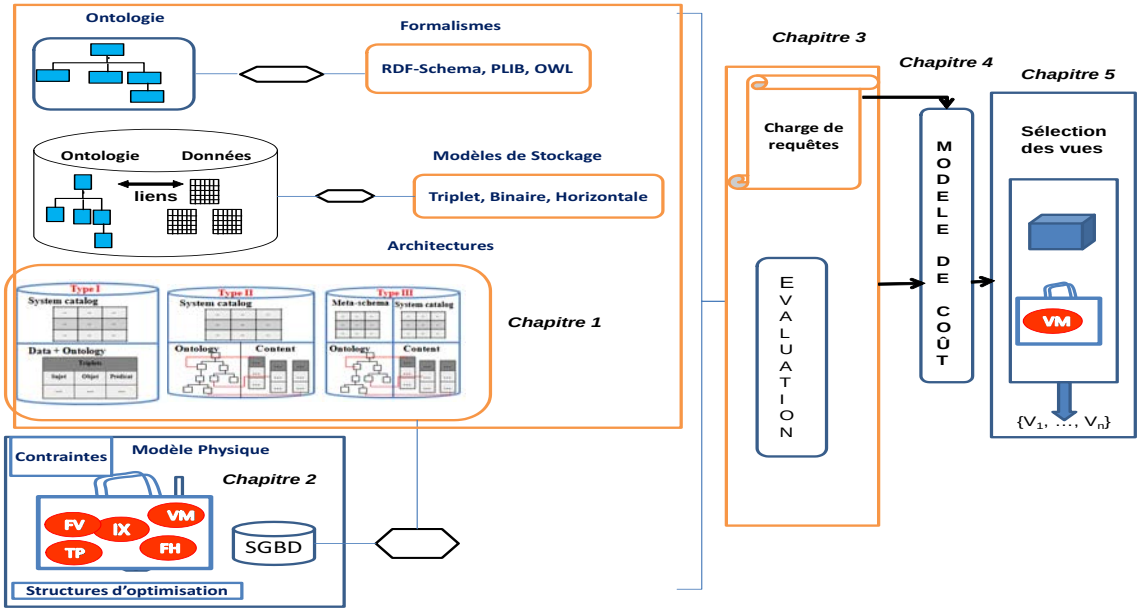


FIGURE 1 – Vue globale de la thèse

4 Structure du mémoire

Ce mémoire est organisé en trois parties et une annexe, comme illustré par la Figure 1.

La première partie comprend deux chapitres.

Le premier chapitre présente dans un premier temps un état des lieux sur les ontologies, leur modèles et introduit une classification des ontologies existantes. Une comparaison entre les modèles conceptuels et les ontologies est également donnée. Dans un deuxième temps, la notion de $\mathcal{BDBO}$ est définie avec l'ensemble de ses composantes : les modèles d'ontologies, les modèles de stockage et les architectures. Des exemples de $\mathcal{BDBO}$ existantes sont donnés pour illustrer leur diversité. Cette analyse nous a permis de donner une représentation multi-dimensionnelle des $\mathcal{BDBO}$, impliquant trois dimensions représentant: le modèle d'ontologie, le modèle de stockage et l'architecture. Enfin deux langages de requêtes sémantiques sont détaillés : le langage SPARQL et OntoQL qui a été développé au sein de notre laboratoire. Dans un souci de clarification et d'analyse, nous avons comparé les $\mathcal{BDBO}$ pour chaque niveau de diversité.

Le second chapitre décrit le problème de la conception physique d'une manière générale. Il commence par la définition formelle de ce problème. Un ensemble de structures d'optimisation est également présenté, à savoir les vues matérialisées et les index pour la catégorie redondante de ces structures et la fragmentation horizontale pour la catégorie non redondante. Pour chaque structure, une formalisation de son problème de sélection est donnée, suivie par les approches de sélection proposées dans la littérature. Une confrontation de la conception physique des bases de données traditionnelles avec les $\mathcal{BDBO}$ est également discutée, suivie par une démarche de

résolution qui intègre l'ensemble de étapes que nous considérons pertinentes pour revisiter la conception physique dans le contexte des $\mathcal{BDBO}$. Elle comprend quatre étapes principales: (i) la maîtrise des $\mathcal{BDBO}$ existantes, (ii) une étude expérimentale des $\mathcal{BDBO}$ afin de déterminer les paramètres pertinents pour les modèles de coûts qui sont au cœur de la conception physique, (iii) la proposition d'un modèle de coûts pour les $\mathcal{BDBO}$ et finalement, (iv) l'utilisation de ces modèles pour la sélection d'une structure d'optimisation, à savoir les vues matérialisées.

La deuxième partie de notre manuscrit présente l'ensemble de nos contributions. Elle contient les trois chapitres suivants.

Le chapitre 3 est consacré premièrement à la formalisation des ontologies et des $\mathcal{BDBO}$. La formalisation de l'ontologie présente la première dimension de diversité à savoir le modèle d'ontologie. Tandis que la deuxième dimension présente en détails la diversité concernant le modèle de stockage pour les instances et l'ontologie et l'architecture du SGBD cible. Ces formalisations nous donnent une vue globale sur l'ensemble des paramètres qui peuvent influencer la performance des requêtes et le chargement des données au sein des $\mathcal{BDBO}$. Six $\mathcal{BDBO}$ sont testées (Oracle, IBM Sor, Jena, Sesame, DB2RDF et OntoDB) sur les données du banc d'essai LUBM, selon les critères cités ci-dessus. Cette étude expérimentale nous permet d'identifier les paramètres importants pour établir notre modèle de coût.

Le chapitre 4 propose un modèle de coût pour estimer le coût des requêtes en termes d'entrées-sorties définies sur une $\mathcal{BDBO}$. Ce modèle prend en compte la diversité présentée dans le chapitre 1. Des formules mathématiques pour estimer le coût de la sélection, de la projection et de la jointure sont données. Ce modèle de coût est validé sur les données du banc d'essai LUBM.

Dans le chapitre 5 nous nous focalisons sur une structure d'optimisation populaire qui est les vues matérialisées. Rappelons que la sélection de ces dernières est difficile et basée sur un ensemble de requêtes connu à l'avance. Pour faciliter la présentation de nos propositions, nous commençons par introduire les concepts de base sur les requêtes, les arbres algébriques et le plan unifié regroupant l'ensemble des arbres algébriques des requêtes de départ. Trois approches de sélection de vues matérialisées sont décrites : une approche conceptuelle, dans laquelle les vues sont sélectionnées au niveau ontologique. Cette approche exploite la présence de l'ontologie dans la base de données. La deuxième approche dite imposée, qui suppose la connaissance de toutes les composantes d'une $\mathcal{BDBO}$. Cette approche est similaire aux approches de sélection des vues traditionnelles, et ne prend pas en compte la diversité. La troisième approche, dite simulée, prend en compte la diversité. Des algorithmes de type glouton et génétique sont proposés pour résoudre le problème de la sélection des vues. Une comparaison de nos propositions par rapport à l'état de l'art est donnée.

La troisième partie présente les conclusions générales de ce travail et esquisse diverses perspectives.

Les publications relatives à ces travaux sont :

- **Bery Mbaiossoum**, Selma Khouri, Ladjel Bellatreche, Stéphane Jean, Mickael Baron, Etude Comparative des Systèmes de Bases de Données à base Ontologiques, Actes du XXXème Congrès INFORSID, pp. 379-394, Montpellier, France, Mai 2012
- **Bery Mbaiossoum**, Ladjel Bellatreche, Stéphane Jean, Mickael Baron, Comparaison et Evaluation des Systèmes de Gestion de Base de Données Sémantiques, Ingénierie des Systèmes d'Information (ISI), 18(3): 39-63, 2013
- **Bery Mbaiossoum**, Ladjel Bellatreche, Stéphane Jean, Towards Performance Evaluation of Semantic Databases Management Systems, 29th British National Conference on Databases (BNCOD 2013), pp. 107-120, LNCS, Springer, Oxford, July, 2013
- **Bery Mbaiossoum**, Ladjel Bellatreche, Stéphane Jean, Materialized View Selection Considering the Diversity of Semantic Web Databases, Proceedings of the 18th Conference on Advances in Databases and Information Systems (ADBIS), pp. 163-176, LNCS, Springer, Ohrid, Republic of Macedonia September 2014,

Ce mémoire comporte une annexe qui présente les requêtes utilisées dans les expérimentations.

Première partie

Concepts fondamentaux

Chapitre 1

Ontologie et Bases de Données à Base Ontologique

Sommaire

1 Introduction

Les Bases de Données à Base Ontologique ($\mathcal{BDBO}$) sont au cœur de nos travaux et plus particulièrement leur conception physique. Contrairement à une base de données traditionnelle qui ne stocke que les instances d'une application, une $\mathcal{BDBO}$ stocke à la fois des instances et l'ontologie décrivant leur sens. La présence de cette nouvelle composante au sein d'un SGBD a poussé la communauté des bases de données à répondre aux quatre questions suivantes :

1. Qu'est-ce qu'une ontologie?
2. Comment définir formellement une ontologie?
3. Est-ce que les modèles de stockage traditionnels (orientés table relationnel) sont adaptés aux ontologies et aux instances ontologiques?
4. Est-ce que l'architecture de stockage traditionnelle d'un SGBD (constituée de deux parties: le contenu et la méta-base) peut être remise en question?

L'ensemble des réponses à ces questions peut influencer considérablement la définition d'un modèle de coût mathématique estimant le temps de réponse d'un ensemble de requêtes, car tout modèle de coût est sensible aux modèles de stockage utilisés par le SGBD cible et à la définition de structures d'optimisation au niveau conceptuel, vu les fortes similarités entre les ontologies et les modèles conceptuels.

Dans ce chapitre, nous commençons par présenter les concepts fondamentaux liés aux ontologies. Puis nous répondons aux questions que nous avons posées.

2 Qu'est-ce qu'une ontologie?

2.1 Origine de la notion d'ontologie

Le terme ontologie fut d'abord utilisé au XVIIeme siècle en philosophie aristotélicienne pour désigner la partie de la philosophie qui a pour objet l'étude des propriétés les plus générales de l'être, telles que l'existence, la possibilité, la durée ou le devenir (cf. CNRTL [3]). Le dictionnaire de la langue philosophique *FOULQ. St-JEAN 1962* [17] définit "l'ontologie comme une étude des êtres en eux-mêmes et non tels qu'ils nous apparaissent". D'après Meynard [18], la métaphysique et l'ontologie sont synonymes au sens strict du terme, c'est-à-dire l'étude de l'être dans ses propriétés générales et dans ce qu'il peut avoir d'absolu ; en d'autres termes, c'est l'étude de ce que sont les choses en elles-mêmes, dans leur nature intime et profonde, par opposition à la seule considération de leurs apparences ou de leurs attributs séparés.

Le terme est aussi utilisé en médecine pour désigner une doctrine qui prétend étudier l'être de la maladie, notamment des fièvres, comme si la maladie existait conformément à un type bien défini, à une essence.

3. Centre National de ressources Textuelles et Lexicales - http://www.cnrtl.fr/lexicographie/ontologie

2.2 Notion d'ontologie en informatique

En informatique et en science de l'information, une ontologie désigne un ensemble structuré des termes et concepts ayant une sémantique dans un domaine donné. L'ontologie constitue donc un modèle de données représentant un ensemble de concepts dans un domaine, ainsi que des relations entre ces concepts. Les concepts sont organisés en un graphe dont les relations peuvent être des relations sémantiques ou des relations de subsomption. Ce type de graphe est appelé le schéma d'ontologies.

En général, les ontologies utilisent les notions suivantes :

- *classe* : un ensemble, une collection ou un type d'objets. Une classe possède toujours un identifiant. Les classes sont toujours organisées en une hiérarchie à l'aide de relation de subsomption ;
- *propriété* : une caractéristique que des objets peuvent posséder ou partager. Cela peut aussi être une fonctionnalité des objets. Les propriétés permettent de distinguer les instances d'une classe. Comme les classes, les propriétés possèdent toujours un identifiant. Elles peuvent être typées, c'est-à-dire associées à un domaine et à un co-domaine de valeurs. Une propriété peut prendre ses valeurs dans des types simples ou dans des classes ;
- *type de valeurs* : un ensemble de valeurs dans lesquels les propriétés doivent prendre leurs valeurs. Il existe (1) des types simples (e.g. entier, réel, caractère, booléen, etc.), (2) des types classes et (3) des types collections (propriétés associées à des cardinalités minimum et maximum) ;
- *individu* : une instance de classes. Les individus sont caractérisés par leur appartenance à une (ou éventuellement plusieurs) classes(s) et par des valeurs de propriétés. Selon les formalismes, chaque individu peut ou non avoir un identifiant unique qui permet de le distinguer des autres ;
- *axiome* : un prédicat applicable à des classes, des relations ou des instances permettant d'assurer l'intégrité des données ou de faire des déductions de connaissances. Tous les formalismes permettent d'exprimer des axiomes de typage et de subsomption. Les autres axiomes dépendent de chaque modèle d'ontologie et des prédicats particuliers définis au niveau du langage. Par exemple, OWL permet d'exprimer des axiomes de cardinalité et PLIB des contraintes sur les domaines de valeurs.

Dans cette thèse, nous ne traitons que des ontologies en informatique.

2.3 Définition d'une ontologie

Une ontologie est une conceptualisation qui permet de représenter explicitement la sémantique d'un domaine par des "modèles objets" consensuels dont chaque concept (classe ou propriété) est associé à un identificateur universel permettant de référencer la sémantique qui lui correspond. Selon Gruber [19], une ontologie est une spécification explicite d'une conceptualisation. Cette définition a été enrichie par Pierra [20] en décrivant une ontologie comme une

représentation *formelle, explicite, référençable* et *consensuelle* de l'ensemble des concepts partagés d'un domaine en termes de classes d'appartenance et de propriétés caractéristiques. Cette définition a mis en avant quatre caractéristiques principales d'une ontologie à savoir : *formelle, explicite, référençable* et *consensuelle* que nous détaillons ci-dessous :

- *formelle* : exprimée dans un langage de syntaxe et de sémantique formalisées comme par exemple RDF schéma [21], OWL [3] ou PLIB [22] permettant ainsi des raisonnements automatiques ayant pour objet soit d'effectuer des vérifications de consistance, soit d'inférer de nouveaux faits ;
- *consensuelle* : admise par l'ensemble des membres (et des systèmes) d'une communauté. Ainsi, la conception d'une ontologie doit impliquer la participation des experts et des spécialistes du domaine. Tous les éléments ontologiques globaux partagés doivent être acceptés et donc être modélisés à l'identique par tous. Par exemple, les ontologies de produits conformes à la norme ISO 13584 (PLIB) font l'objet d'un consensus international résultant d'un processus rigoureux de standardisation et sont publiées comme des standards internationaux ISO ou IEC. Il convient de signaler que cette caractéristique est inhérente aux ontologies globales qui peuvent être importées et modifiées pour des usages locaux (ontologie locale) ;
- *référençable* : toute entité ou relation décrite dans l'ontologie peut être directement référencée par un symbole (« identifiant universel »), à partir de n'importe quel contexte, afin d'expliciter la sémantique de l'élément référençant ;
- *explicite* : les concepts du domaine et leurs différentes propriétés sont explicitement décrits et le domaine modélisé est décrit indépendamment d'un point de vue ou d'un contexte implicite. Des points de vue différents pourront donc partager les mêmes concepts.

2.4 Types d'ontologies

Suivant le degré d'abstraction visé, trois types d'ontologies ont été identifiés [23] :

- **les ontologies globales** (Top-Level Ontology) : ce sont des ontologies formelles qui présentent un haut niveau de généralité et d'abstraction. Elles sont le résultat d'un développement systématique, consensuel et rigoureux permettant le partage de connaissances et le transfert d'un contexte à un autre. Elles sont dédiées à une utilisation très générale. Elles servent de base pour le développement des ontologies locales spécifiques à des domaines donnés. On peut citer comme exemple le projet KRAFT [24] ou WordNet [4] [25] ;
- **Les ontologies de domaine** : ce sont des ontologies qui se limitent à une représentation des concepts de domaines données (géographie, médecine, etc.). Elles présentent un niveau d'abstraction moins élevé que celui des ontologies globales. En général, elles apportent une spécialisation des concepts généraux d'une ontologie globale. On peut citer comme exemple Uniprot [26], une ontologie décrivant les protéines ;
- **Les ontologies d'application** qui sont spécifiques à un champ précis d'application dans

4. dictionnaire de la langue anglaise, `http://www.cogsci.princeton.edu/wn`

un domaine donné. Elles présentent donc un haut degré de spécificité. Une ontologie d'application spécifie une ontologie de domaine ou une ontologie globale. Par exemple, l'ensemble des spécifications sur les cours d'eau de France est une ontologie d'application qui peut spécifier les concepts généraux d'une ontologie de domaine géographique. L'ontologie géopolitique [5] mise au point par la FAO (Organisation des Nations Unies pour l'Alimentation et l'Agriculture) pour faciliter l'échange et le partage de données de manière standardisée entre les systèmes de gestion d'informations relatives à des pays et/ou à des régions est aussi un exemple d'ontologie d'application.

Dans le cadre de notre recherche, nous nous intéressons aux ontologies de domaine. Nous présentons dans la section suivante une classification de ce type d'ontologies.

2.5 Taxonomie des ontologies de domaine

Les ontologies de domaine ne sont pas toutes identiques. Certaines se focalisent sur la définition des concepts d'un domaine d'étude. On les appelle les *ontologies conceptuelles (OC)*. Gruber [19] distingue deux types de concepts dans une ontologie : les concepts *primitifs* et les concepts *définis*. Les concepts primitifs ou canoniques représentent des concepts ne pouvant être définis par une définition axiomatique complète. Les concepts primitifs sont une base sur laquelle peuvent être définis d'autres concepts appelés concepts définis ou concepts non canoniques. Pierra [27] fait remarquer qu'à côté des ontologies basées sur les concepts, il existe des ontologies qui décrivent des mots, elles sont qualifiées d'*ontologies linguistiques (OL)*. Ainsi, trois catégories d'ontologies existent [27] :

- *Ontologies Conceptuelles Canoniques (OCC)* : ce sont des ontologies qui ne contiennent que des concepts primitifs ;
- *Ontologies Conceptuelles Non Canoniques (OCNC)* : ce sont des ontologies qui contiennent des concepts primitifs ainsi que des concepts définis ;
- *Ontologies Linguistique (OL)* : ce sont des ontologies qui définissent des termes éventuellement définis en plusieurs langues naturelles qui sont utilisés dans un domaine d'étude. Ces ontologies sont orientées vers les traitements linguistiques (par exemple TALN [6]).

Ces trois catégories d'ontologies peuvent être combinées en un modèle en oignon (Fig. 1.1). La couche de base de ce modèle est constituée par la couche canonique. La seconde couche non canonique est construite sur la couche canonique et la dernière couche est formée par les ontologies *OL* fournissant une représentation des concepts d'un domaine en langage naturel éventuellement dans plusieurs langues. Ce modèle en oignon définit donc une ontologie de domaine en couches complémentaires pour la résolution de différents problèmes.

Nous présentons ci-dessous brièvement chacune des catégories d'ontologie identifiées.

5. http://www.fao.org/countryprofiles/geoinfo.asp?lang=fr
6. Traitement Automatique de la Langue Naturelle

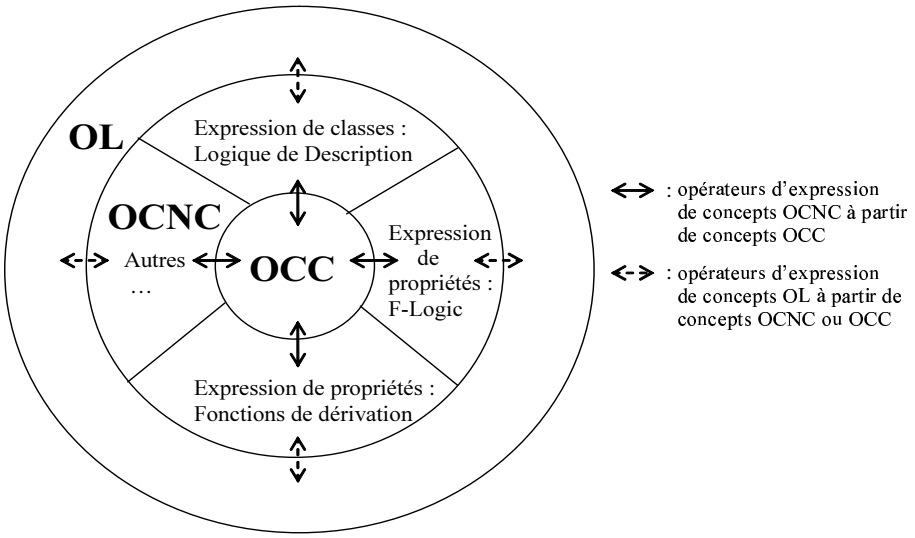


FIGURE 1.1 – Le modèle en oignon pour les ontologies de domaine (extraite de [28])

2.5.1 Ontologies Conceptuelles Canoniques (OCC)

Une ontologie canonique ne présente que des concepts primitifs (ou atomiques), c'est-à-dire des concepts qui ne peuvent pas être dérivés d'autres concepts. Les ontologies canoniques fournissent une base formelle pour une modélisation et un échange efficace de la connaissance d'un domaine. Ces ontologies sont importantes pour le domaine des bases de données (où la redondance est à éviter), pour le vocabulaire technique (où chaque notion est représentée par un mot *unique*) et pour les formats d'échange (où une seule représentation possible est prévue pour chaque information échangée). Dublincore [7] [29] est un exemple normalisé d'ontologies conceptuelles canoniques. Pour mieux illustrer nos propos, considérons un fragment de l'ontologie LUBM [30] présentée sur la figure 1.2. Elle est constituée de trois classes appelées *Personne* ayant comme propriété *nom* et *sexe*, *Etudiant* qui est une sous-classe de *Personne* et *Université*. Dans cette ontologie, tous les concepts sont canoniques. RDFS [21] et PLIB [31] sont des modèles d'ontologies communément utilisés pour définir des ontologies canoniques.

2.5.2 Ontologies Conceptuelles Non Canoniques (OCNC)

Ce sont des ontologies qui renferment dans leurs définitions des relations d'équivalence entre les concepts. Ce genre de relations permet de définir les concepts non canoniques. Les *OCNC* contiennent des concepts primitifs ainsi que des concepts définis. La définition de concepts se fait à l'aide des opérateurs portant sur les classes comme les opérateurs ensemblistes (union, intersection, différence) et les opérateurs portant sur les propriétés comme les restrictions sur les valeurs de propriétés ou sur la cardinalité. Elle peut se faire aussi à travers des relations algébriques ou logiques entre les propriétés. Les OCNC permettent de représenter à la fois les

7. http://dublincore.org/

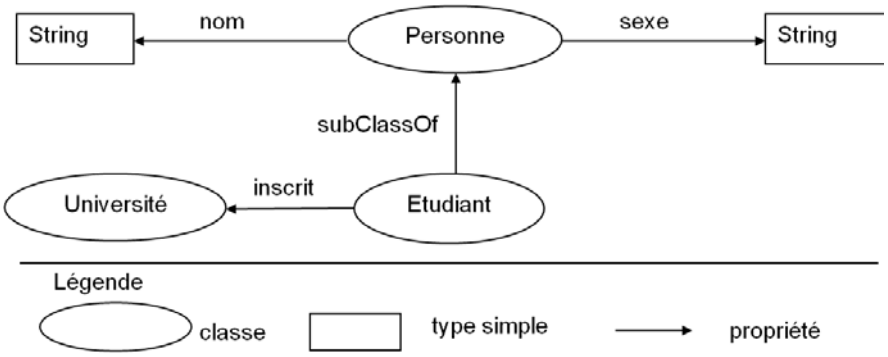


FIGURE 1.2 – Exemple d'une ontologie OCC

conceptualisations d'un domaine et les correspondances entre celles-ci. Dans notre exemple d'ontologie OCC précédent, on pourrait définir de manière informelle le concept canonique *Etudiant* comme un concept non canonique : *Etudiant* ≡ *Personne inscrite dans une université*. On peut ainsi incorporer dans notre OCC, des concepts non primitifs et obtenir donc une OCNC. La figure 1.3 présente une ontologie OCNC surcouche de l'ontologie OCC précédente avec une spécialisation du concept *Personne* par deux concepts définis *Homme* (Personne de sexe masculin), *Femme* (Personne de sexe féminin) et une définition informelle du concept *Etudiant*. Les constructeurs OCNC sont également très utiles pour définir des mappings entre différentes ontologies. Ils permettent également d'étendre les inférences que l'on peut réaliser sur une ontologie.

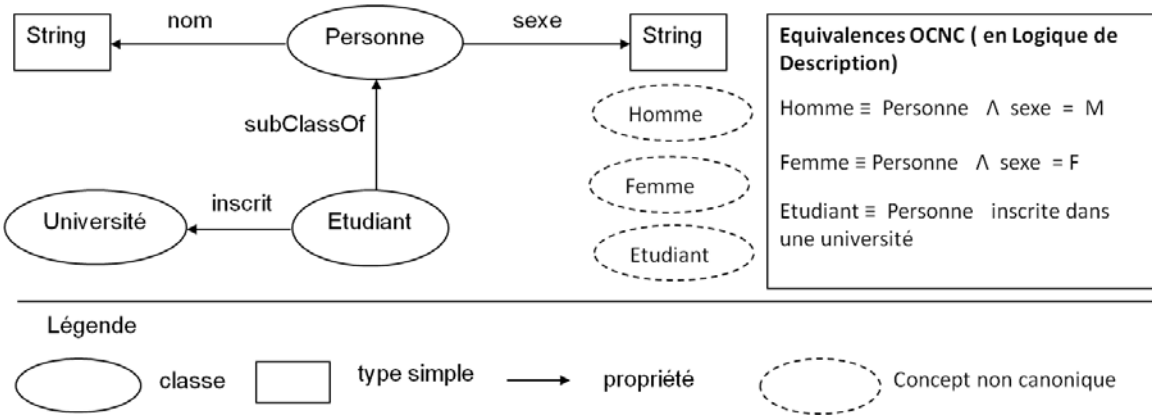


FIGURE 1.3 – Exemple d'une ontologie OCNC

2.5.3 Ontologie Linguistique (OL)

Ce sont des ontologies qui visent à représenter les mots utilisés dans un domaine particulier. Les ontologies linguistiques permettent de fournir une représentation en langage naturel des concepts d'un domaine. Les représentations peuvent être multilingues. Ces ontologies sont

utilisées pour la recherche de synonymes linguistiques et dans la recherche de documents pertinents par rapport à une requête qui s'exprime par un ensemble de mots. *WordNet* [25] est un exemple d'ontologie linguistique. On peut obtenir une ontologie OL à partir de notre exemple d'ontologie OCNC précédent par addition de commentaires ou de descriptions textuelles aux concepts définis ou primitifs. Par exemple, le concept *Homme* pourrait être associé aux termes *Man*, *Hombre*, *Mann*, etc. La construction de telles ontologies est souvent faite de façon semi-formelle par un processus d'extraction de termes dans un ensemble de documents du domaine et qui sont ensuite validés et structurés par des experts (humains) du domaine.

2.5.4 Modèle en oignon et conception des ontologies

On peut remarquer que la taxonomie des ontologies en couches présentée précédemment revêt une dimension méthodologique définissant deux méthodes de conception d'ontologies. La première méthode dite "des mots aux concepts" consiste à extraire la couche conceptuelle à partir de la couche linguistique. Elle nécessite l'extraction semi-automatique des termes du domaine (à partir de corpus textuels par exemple) et la définition des concepts à partir de ces termes. La deuxième méthode dite "des concepts aux mots" consiste à construire la couche linguistique à partir de la couche conceptuelle. Elle nécessite la définition des concepts canoniques d'un domaine obtenus par un consensus d'experts, un choix des opérateurs d'équivalence entre les concepts pour définir les concepts non canoniques ou dérivés, et une définition en langage naturel des concepts de l'ontologie.

2.5.5 Les ontologies dans le monde industriel

Les ontologies ont été largement utilisées dans le monde industriel, et en particulier, un nombre important d'ontologies dans le domaine de l'ingénierie ont été développées. On peut citer par exemple, l'ontologie normalisée IEC 61360-4 sur le domaine des composants électroniques, l'ontologie ISO 13399 sur le domaine des outils coupants, l'ontologie ISO 13584-501 sur le domaine des matériels de mesure, etc. Ces ontologies ont été acceptées comme des ontologies consensuelles et partagées par tous les participants impliqués dans le processus B2B (Business to Business), où chaque fournisseur décrit les classes de ses composants dans son catalogue en référençant le plus possible l'ontologie normalisée. Dans ce cas, l'ontologie joue le rôle d'un dictionnaire. Sa présence facilite l'intégration et l'échange des composants entre les différents fournisseurs. Notre laboratoire LIAS a participé au développement et à la normalisation d'un nombre important d'ontologies dans le domaine de l'ingénierie (avionique avec Airbus, énergie avec EDF, géologique avec l'Institut Français de Pétrole, etc.).

Dans cette section, nous avons défini et illustré par des exemples les notions fondamentales liées aux ontologies. Dans la section suivante, nous répondons à la deuxième question posée dans l'introduction en décrivant différents modèles d'ontologie.

3 Comment définir formellement une ontologie?

Un formalisme d'ontologies est un modèle permettant de représenter des ontologies. Généralement présenté sous la forme d'un modèle orienté objet, il est composé d'entités et d'attributs permettant de décrire les constructeurs d'une ontologie tels que les constructeurs de classes et de propriétés. Dans la littérature, on trouve beaucoup de modèle d'ontologies parmi lesquels PLIB [2], F-Logic [32], RDF/RDFS [21], DAML+OIL[33], etc. Ces modèles diffèrent selon l'objectif visé. Certains ont pour but de définir des ontologies pour la gestion et l'échange des données. Ils cherchent ainsi à définir la sémantique des concepts de manière unique et précise et se focalisent donc sur les *OCC*. Les modèles RDF-Schema [21] et PLIB [2] sont des exemples de ces formalismes. D'autres formalismes ont été proposés pour la description d'ontologies permettant de définir des correspondances entre vocabulaires et offrent ainsi des possibilités de déduction et d'inférence. Ces formalismes définissent ainsi des concepts primitifs et définis. OWL ou OWL Flight [3] en sont des exemples. Nous présentons ci-dessous les modèles d'ontologies qui nous intéressent dans nos travaux : les modèles RDF/RDFS, OWL et PLIB. Nous verrons qu'ils se basent sur un noyau commun de constructeurs sur lesquels nous avons basé nos travaux.

3.1 Formalisme RDF/RDFS

RDF (Resource Description Framework) est un langage utilisé pour la représentation des informations relatives aux ressources sur le Web. Il fut conçu pour annoter les documents du Web à l'aide de métadonnées (le titre, l'auteur et la date de modification d'une page web, etc.). Avec la généralisation du concept de « ressource Web », RDF est utilisé pour représenter des informations à propos de choses identifiables sur le Web. Beaucoup de sites de vente en ligne utilisent RDF pour la présentation de leurs articles. Les informations exprimées en RDF sont traitables par des applications au lieu d'être seulement affichées pour les humains. Ces informations peuvent ainsi être échangées entre les applications sans perte de sens. Cela veut dire que les informations peuvent être mises à la disposition d'autres applications que celles pour lesquelles elles ont été créées au départ.

Pour identifier les ressources, RDF utilise des URI (Uniform Resource Identifiers). Il décrit les ressources en fonction de propriétés et de valeurs de propriétés. Les instances RDF (ou déclaration RDF) sont présentées sous forme de triplets *(sujet, prédicat, valeur)*.

- *sujet* désigne la ressource à décrire. Le sujet est en général identifié par un URI mais peut aussi être un nœud anonyme. Une ressource peut être une page Web, une partie d'une page Web référencée par une ancre ou même un objet quelconque.
- *prédicat* représente une propriété qui peut être appliquée sur la ressource. Il est aussi représenté par un URI.
- *objet* est la valeur de la propriété. Il peut être une donnée ou une autre ressource. Il

est identifié par un URI mais peut aussi être un nœud anonyme ou un littéral (valeur éventuellement typé par un des types de données primitifs définis par XML Schéma) .

Un ensemble de données RDF peut être représenté par un graphe de nœuds et d'arcs représentant les ressources, leurs propriétés et valeurs. Par exemple, le triplet *(Etudiant, est-Inscrit, Cours)* est représenté graphiquement sur la figure 1.4.

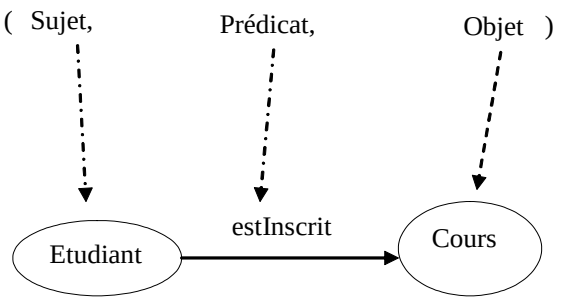


FIGURE 1.4 – Exemple d'un triplet RDF

La sémantique d'un document RDF est exprimée en s'appuyant sur la théorie des ensembles et la théorie du modèle [34]. Ce dernier vise à définir des structures mathématiques pour une interprétation des éléments du vocabulaire de RDF. Ces structures fournissent le moyen de vérifier formellement qu'une déclaration (triplet) est vraie dans un document RDF. En effet, tout triplet RDF peut être traduit en formule de logique du premier ordre, positive, conjonctive et existentielle :

$$(Sujet,\ Predicat,\ Objet) \Rightarrow \forall Object,\ \exists\ Sujet\ tel\ que\ Predicat(Sujet,\ Objet)$$

RDF introduit quelques prédicats prédéfinis, mais ne propose pas de constructeurs prédéfinis pour la conception d'ontologies. Il ne permet pas de formuler des contraintes sémantiques riches ou de faire des raisonnements. Ainsi, il a été rapidement étendu par un ensemble de constructeurs constituant le modèle RDF-Schema pour la définition d'ontologies.

RDF-Schéma (ou RDFS) est un modèle issu du Web Sémantique, étendant le modèle RDF par des constructeurs permettant la définition de classes et de propriétés. Il permet entre autres de bâtir des concepts, éventuellement définis à partir d'autres concepts, et ayant la particularité d'être partagés. Il offre un ensemble de constructeurs de modélisation orientés objets pour la définition de concepts. RDFS est un langage de représentation des connaissances avec une syntaxe et une sémantique. Il permet de réaliser des inférences [35]. Grâce à l'inférence, on peut déduire des nouveaux triplets à partir des triplets existants. Les expressions RDFS sont aussi écrites sous forme de triplets (triplets RDF) et restent des expressions RDF valides. La seule différence entre une expression RDF "normale" et une expression RDFS est l'accord fait sur la sémantique des termes définis en RDFS et par conséquent sur l'interprétation de certains triplets.

RDFS définit la classe *ressource* comme classe mère de toutes les classes. Il permet une description textuelle des classes et des propriétés au travers des attributs *rdfs:label, rdfs:comment, rdfs:seeAlso et rdfs:isDefinedBy*. RDFS ne dispose pas de primitives décrivant les équivalences conceptuelles. Les ontologies RDFS sont donc des ontologies canoniques (OCC). Nous présentons ci-dessous les principaux constructeurs de RDFS.

- **Constructeurs de classes.**
  Pour créer des classes, le constructeur *rdfs:class* est utilisé. Le constructeur *rdfs:subClassOf* permet de spécifier une organisation hiérarchique des classes.
- **Constructeurs de propriétés.**
  Le constructeur *rdfs:property* permet de spécifier des propriétés caractéristiques d'une classe. La hiérarchie des propriétés est créée à l'aide du constructeur *rdfs:subProperty*. Les constructeurs *rdfs:domain* et *rdfs:range* permettent de spécifier respectivement le domaine (une ou plusieurs classes) et le co-domaine de valeur d'une propriété (une classe ou un type de données).
- **Constructeurs d'instances.**
  L'appartenance d'une instance à une classe est définie grâce au constructeur *rdf:type*. Ce constructeur permet de "typer" une ressource (ou instance RDF) avec une propriété ou une classe définie en RDFS. Les triplets correspondant sont de la forme (*i*, *rdf:type*, *C*) indiquant que *i* est une instance de la classe *C*. Dans RDFS, il est possible qu'une instance appartienne à plusieurs classes même de différentes hiérarchies (subsomption). L'attribution d'une valeur *v* à une propriété *p* d'une instance *i* se fait à l'aide d'un triplet de la forme (*i*, *p*, *v*).
- **Les types de valeurs.**
  Les types de valeurs des objets manipulés dans RDFS peuvent être des instances de classes ou des littéraux. Les littéraux peuvent être typés par les types prédéfinis de XML Schéma (pour les chaînes de caractères, les numériques, les dates, etc.). RDFS fournit aussi un type collection (*rdfs:Container*) et utilise les types collection de RDF (*rdf:List*, *rdf:Bag*).
- **Les axiomes.**
  Hormis l'appartenance et la subsomption, RDFS ne permet pas de définir de nouveaux axiomes. Par contre, il permet la méta-modélisation. Ainsi, il permet suivant le rôle joué dans un contexte par une information, de la représenter comme une classe ou comme une instance. Dans la relation de subsomption, RDFS ne permet pas de référence circulaire, c'est-à-dire qu'une classe (ou une propriété) ne peut pas être sa propre sous-classe (ou sa propre sous-propriété).

RDFS étend ainsi RDF avec un ensemble de constructeurs qui fait de lui un formalisme d'ontologies. Dans la section suivante, nous présentons le formalisme OWL qui permet de définir des ontologies orientées inférence qui sont beaucoup utilisées dans le domaine du Web Sémantique.

3.2 Formalisme OWL

Le modèle OWL est actuellement reconnu par le W3C [8] comme étant le standard pour représenter des ontologies pour le Web Sémantique. Il permet une description très riche des ontologies et facilite la publication et le partage des ontologies sur le Web Sémantique. OWL est inspiré de DAML (projet américain) et d'OIL (projet Européen). Pour permettre la manipulation des ontologies par les humains et les machines, OWL étend le modèle *RDFS* par des opérateurs permettant la définition d'ontologies plus expressives et offrant des capacités de raisonnement supérieures. OWL étant construit sur RDF et RDFS, il utilise la syntaxe RDF/XML. Il permet de raisonner sur ces connaissances en s'appuyant sur la Logique de Description (axiomatique formelle).

OWL se décline en trois versions : *OWL-Lite*, *OWL-DL* et *OWL-Full*, qui sont des sous-modèles présentant un compromis entre leur pouvoir expressif et leur décidabilité de raisonnement. Le langage *OWL-Lite* décrit une hiérarchie de classifications et offre des mécanismes de contraintes simples. Par exemple, *OWL-Lite* gère des contraintes de cardinalité uniquement de valeurs 0 ou 1. Le langage *OWL-DL* offre une expressivité supérieure à OWL-Lite sans remettre en cause la complétude de calcul (inférences) et la décidabilité des systèmes de raisonnement. Il est fondé sur la logique descriptive. Le langage *OWL-DL* intègre toutes les structures de langage d'OWL avec des restrictions comme la séparation des types (une classe ne peut pas être en même temps un individu et une propriété, une propriété doit être associée à un individu ou une classe). Le langage *OWL-Full* est destiné aux utilisateurs souhaitant une expressivité supérieure à OWL-DL et une liberté syntaxique de RDF sans une garantie de décidabilité du raisonnement (inférence). Par exemple, dans *OWL-Full*, on peut simultanément traiter une classe comme une collection d'individus et comme un individu à part. Il existe une hiérarchie de dépendance entre ces trois sous langages : toute ontologie *OWL-Lite* valide est également une ontologie *OWL-DL* valide, et toute ontologie *OWL-DL* valide est également une ontologie *OWL-Full* valide. Le degré d'expressivité est croissant de OWL Lite à *OWL-Full*.

Nous présentons dans ce qui suit les constructeurs de OWL DL. Certaines limites de OWL Lite sont aussi évoquées.

3.2.1 Constructeurs de classe OWL

Il y a plusieurs manières de déclarer une classe OWL :

– à l'aide du constructeur *owl:Class* (classes nommées). Le concept d'héritage se traduit à l'aide de la propriété *subClassOf*. Il existe dans toute ontologie OWL une superclasse, nommée *Thing*, dont toutes les autres classes sont des sous-classes. Il existe également une classe nommée *noThing*, qui est sous-classe de toutes les classes OWL. Cette classe ne peut pas avoir d'instance ;

8. World Wide Web Consortium

- par énumération des instances de la classe à l'aide du constructeur "*owl:oneOf*" ;
- par restriction des propriétés (contrainte de valeur et cardinalité) : une contrainte de valeur porte sur la valeur de propriété d'une instance tandis qu'une contrainte de cardinalité porte sur le nombre de valeurs que peut prendre une propriété. Les restrictions sur des valeurs sont faites par les axiomes :
  - *allValuesFrom* qui exprime une restriction universelle, c'est-à-dire toutes les instances qui, pour une propriété donnée, prennent leur valeur dans une classe donnée ;
  - *someValuesFrom*, qui exprime une restriction existentielle, c'est-à-dire toutes les instances qui, pour une propriété donnée, ont au moins une valeur dans une classe donnée ;
  - *hasValue* qui permet de définir des classes selon l'existence de valeurs de propriétés particulières. Elle impose une valeur spécifique à toutes les instances de la classe en question.

  Les cardinalités sont définies à l'aide des axiomes *owl:minCardinality*, *owl:maxCardinality* et *owl:Cardinality* qui expriment les nombres d'éléments dans une relation ou son intervalle ;
- par intersection de deux ou plusieurs classes avec *owl:intersectionOf* ;
- par union de deux ou plusieurs classes grâce au constructeur *owl:unionOf* ;
- par complémentarité d'une autre classe c'est-à-dire une classe qui contient toutes les instances dans le domaine de discours n'appartenant pas à une autre classe. Cela se fait grâce au constructeur *owl:complementOf*.

3.2.2 Construction des propriétés OWL

On distingue deux types de propriété OWL : Les propriétés d'objets (*owl:ObjectProperty*), qui sont des relations entre les instances de deux classes et les propriétés de types de données (*owl:DatatypeProperty*), qui sont des relations entre des instances de classes, des littéraux RDF [36] et des types de donnée de schéma XML [37].

OWL offre des constructeurs pour définir et restreindre des propriétés (cf. définition des classes par restriction des propriétés). Nous pouvons définir un domaine et un co-domaine pour une propriété. Il est possible d'ajouter des caractéristiques de propriété qui fournissent un mécanisme puissant pour améliorer le raisonnement lié aux propriétés. Parmi les caractéristiques principales des propriétés, nous trouvons la transitivité, la symétrie, la fonctionnalité et l'inverse [38]. Une propriété peut être aussi définie comme une spécialisation (sous-propriété) d'une autre propriété (*rdfs:subProperty*).

3.2.3 Types de données

OWL utilise pour le type de données d'une propriété *owl:DatatypeProperty*, le type RDF-Schema *rdfs:Literal* ou un des types simples définis pour les XML Schéma. Pour le type de données d'une propriété *owl:ObjectProperty*, on utilise une classe en se servant du constructeur

rdfs:range pour la spécifier.

3.2.4 Instance d'une classe

On déclare l'appartenance d'un individu (instance) à une classe de la manière suivante :

⟨*NomClasse rdf:ID="individuID"*⟩ où *individuID* est l'identifiant de l'instance et *NomClasse* est le nom de la classe à laquelle on associe *individuID*. Exemple : ⟨*Université rdf:ID="UP"*⟩ déclare *UP* comme instance de la classe *Université*. La propriété *rdf:type* est aussi utilisée pour lier un individu à une classe dont il est membre.

Les instance sont identifiées par des URI et peuvent appartenir à plusieurs classes non liées par la relation de subsomption.

3.2.5 Constructeur d'axiomes

OWL offre la possibilité de préciser les cardinalités d'une propriété à l'aide des axiomes *owl:minCardinality*, *owl:cardinality* et *owl:maxCardinality*. L'axiome *owl:equivalentClass* indique que deux classes sont équivalentes, c'est-à-dire qu'elles possèdent les mêmes instances. L'axiome *owl:disjointWith* indique que deux classes données ne partagent aucune instance. L'axiome *owl:equivalentProperty* permet d'indiquer que deux propriétés sont équivalentes, c'est-à-dire qu'elles présentent les mêmes valeurs pour les mêmes instances. En OWL, l'hypothèse d'unicité des noms qui considère que les identifiants permettent d'identifier de manière unique les éléments dans une ontologie, n'est pas garantie. En effet, deux instances de différents identifiants peuvent être les mêmes. Pour gérer cela, OWL propose les axiomes *owl:sameAs*, *owl:differentFrom* et *owl:AllDifferent* permettant d'indiquer l'égalité ou l'inégalité des instances.

Avec ces constructeurs, les raisonnements associés à *OWL-DL* restent décidables. Le problème d'inférence en OWL DL est classé comme étant aussi difficile que ce problème dans les langages des logiques de description de la famille `SHOIN(D)`. Tobies [39] a montré que ces langages sont de complexité au pire des cas de temps exponentiel non déterministe (NExpTime). Dans la pratique, *OWL-Lite* est souvent utilisé pour obtenir de bonnes performances [40]. Celui-ci n'autorise pas la création de classes par les opérations booléennes *owl:unionOf* et *owl:complementOf*, par énumération de ses instances et par restriction de valeurs *owl:hasValue*. Les axiomes de cardinalité *owl:minCardinality*, *owl:cardinality* et *owl:maxCardinality* sont limités à prendre la valeur 0 ou 1 en *OWL-Lite*. Dans la section suivante, nous présentons le formalisme PLIB qui a été conçu pour définir des ontologies canoniques pour le domaine de l'ingénierie.

3.3 Formalisme PLIB

Le modèle PLIB (Parts LIBrary : série de normes ISO 13584) [2] est un modèle initialement défini pour la description des différentes catégories de composants industriels et de leurs instances. Son objectif est de permettre l'échange et la modélisation des différentes catégories de composants industriels et de leurs instances avec le plus de précision possible; c'est-à-dire que l'on doit savoir à quelle classe appartient un objet, à quels objets s'applique une propriété et avec quelle grandeur. Pour cela, une ontologie PLIB doit définir de façon la plus concise possible les classes et les propriétés qui caractérisent les objets d'un domaine du monde réel et les abstractions qu'une communauté peut en construire. Cela veut dire qu'une ontologie PLIB ne définit une nouvelle classe que quand celle-ci ne peut être complètement définie en termes d'autres classes et de valeurs de propriétés. PLIB offre des opérateurs de modularité permettant l'intégration dans un environnement homogène et cohérent d'ontologies hétérogènes provenant de différentes sources.

Dans un schéma d'ontologie PLIB, les propriétés jouent un rôle essentiel. Elles permettent d'associer à chaque concept une définition, une note, un nom, une image ou un document quelconque. Des classes sont définies seulement pour représenter des domaines de propriétés, et chaque propriété est définie dans le domaine d'une classe et n'a de sémantique que pour cette classe et ses sous-classes. Cette vision s'oppose à celle de OWL qui peut définir une propriété sans une classe comme domaine explicite. PLIB permet de construire des ontologies multilingues où le même identifiant de concept est lié à des descriptions dans plusieurs langues. La multi-représentation est aussi possible. En effet, une fois défini, un concept peut être associé à un nombre illimité de représentations. Le point de vue qui caractérise chaque représentation est également un concept représentable dans l'ontologie. PLIB ne fournit que des constructeurs canoniques et ainsi les ontologies PLIB sont des modèles canoniques d'échange. Nous présentons ci-dessous les principaux constructeurs de ce modèle.

3.3.1 Constructeurs de classes PLIB

PLIB propose des constructeurs de classes et des mécanismes pour les organiser en des hiérarchies de subsomption. Pour la définition des classes, PLIB fournit le constructeur *item_-class*. Les classes, comme les autres éléments d'une ontologie, sont identifiées par un *BSU* (Basic Semantic Unit). Le *BSU* est un code d'identification construit à partir de l'identifiant universel de l'organisation à l'origine de la construction de l'ontologie (*supplier*). PLIB propose aussi les constructeurs *functional_model_class* et *functional_view_class* pour la création des classes de représentation (classe ne contenant que des propriétés qui n'ont de sens que pour un point de vue "métier") et les classes de point de vue (classe définissant une perspective où les propriétés des classes de représentation sont définies).

Deux opérateurs de subsomption sont disponibles : *is_a* et *is_case_of*. L'opérateur *is_a* définit un héritage simple qui implique l'héritage des propriétés définies par les classes sub-

sumantes. L'opérateur *case_of* permet la subsomption multiple avec importation sélective et n'implique pas implicitement l'héritage de propriétés. Pour hériter d'une propriété d'une de ses classes subsumantes, une classe doit explicitement indiquer l'importation de cette propriété dans sa définition. On voit qu'une classe peut donc hériter d'une partie des propriétés des classes subsumantes. Cela permet une construction modulaire des ontologies de domaine qui n'importent d'autres ontologies de domaines que certaines classes et propriétés jugées utiles. Une classe PLIB peut avoir au plus une seule superclasse renseignée par l'attribut *its_superclass*.

3.3.2 Constructeurs de propriétés PLIB

Pour définir les propriétés, PLIB propose le constructeur *non_dependent_p_det* pour les propriétés autonomes, le constructeur *dependent_p_det* pour les propriétés dépendantes (c'est-à-dire, dont les valeurs dépendent de paramètres de contexte) et le constructeur *representation_det* pour les propriétés de représentation définies dans une classe de point de vue ou de représentation. PLIB propose un constructeur de paramètres de contexte (*condition_det*) qui décrit le contexte dans lequel une valeur de propriété est donnée. PLIB impose un *typage fort* des propriétés : chaque propriété doit obligatoirement préciser son domaine et son co-domaine. Le domaine et le co-domaine sont définis grâce aux constructeurs *name_scope* et *data_type*. Un co-domaine est un ensemble mathématique défini en extension ou en intention. Il peut être une classe (c*lass_instance_type*), une collection (*set_type, bag_type, list_type, array_type*) ou un autre type de données. Le champ d'application d'une propriété est défini par son domaine. Il arrive souvent que ce champ d'application soit composé de plusieurs classes issues de différentes branches de la hiérarchie. Pour résoudre ce problème, PLIB propose de qualifier les propriétés de la façon suivante :

- une propriété est définie sur une classe *C* au plus haut niveau de la hiérarchie où elle peut être définie sans ambiguïté. Sa portée s'étend sur toutes les sous-classes de *C* ;
- une propriété visible sur une classe *C* peut devenir *applicable* sur cette classe, c'est-à-dire que toute instance de *C* doit présenter une caractéristique ou une grandeur qui représente cette propriété.

3.3.3 Type de données PLIB

Le modèle PLIB fournit des types de données prédéfinis comme les entiers ou les réels. On peut associer à ces derniers des unités de mesure (*measure_type*) ou des unités monétaires (*currency_type*). PLIB permet d'utiliser une classe comme un type de données grâce au constructeur *class_instance_type*. PLIB fournit également le type *aggregate_type* pour la création de collections. Plusieurs types collections sont disponibles comme par exemple les listes (*list_type*), les tableaux (*array_type*), les sacs (*bag_type*) ou les ensembles (*set_type*). La cardinalité de ces collections peut être définie. Enfin, le modèle d'ontologies PLIB permet de créer des types de données utilisateurs comme par exemple des types énumérés (*non_quantita-*

tive_code_type, non_quantitative_int_type, ...).

3.3.4 Constructeurs des instances PLIB

Une instance représente un objet appartenant à une classe. Elle est définie par sa classe dite de base et l'ensemble de ses valeurs de propriétés. PLIB ne permet donc pas la multi-instanciation mais offre le mécanisme d'agrégation d'instance. Pour cela, PLIB distingue les propriétés essentielles d'une classe de celles qui dépendent d'un point de vue sur le concept représenté. Ainsi, une instance peut appartenir non seulement à sa classe de base mais aussi aux classes de représentation attachées à cette classe de base par la relation *is_view_of*. Cependant pour éviter la redondance au niveau des valeurs des propriétés des instances, les propriétés définies sur la classe de base ne sont pas reprises dans les classes de représentation.

3.3.5 Axiomes PLIB

Pour être valide, les ontologies PLIB doivent vérifier certains axiomes parmi lesquels on trouve :

- chaque concept ontologique est identifié par un *BSU* ;
- chaque instance doit appartenir à une seule classe de base (et ses superclasses) ;
- les instances ne sont décrites que par les propriétés applicables à leurs classes ;
- la valeur d'une propriété doit appartenir à son co-domaine.

PLIB n'autorise pas un héritage multiple et un cycle dans le graphe de subsomption (*is_a*) qui est une forêt.

Ces axiomes définissent des contraintes d'intégrité utilisées pour la validation des données. Si une instance est associée à une propriété non applicable à sa classe ou dont le type valeur n'est pas correcte, le système doit signaler cette erreur et cette incompatibilité entre l'ontologie et les données.

3.4 Synthèse

La présentation détaillée de ces trois formalismes d'ontologies (RDF/RDFS, PLIB et OWL) nous a permis d'identifier leurs caractéristiques communes et différentes. Ces dernières sont détaillées dans les sections suivantes.

3.4.1 Caractéristiques communes des formalismes d'ontologies

Tous les formalismes étudiés utilisent les concepts de classes et de propriétés pour la conceptualisation de domaine. La relation de subsomption est utilisée pour définir la hiérarchie entre les classes. Chaque concept est associé à un identifiant qui permet d'assurer son référencement

à partir de n'importe quel environnement. En RDFS et OWL, cet identifiant est un URI alors qu'en PLIB c'est un *BSU*. Ces formalismes utilisent des types de données simples et collections. Ils fournissent donc des constructeurs communs pour la définition des *OCC*. Le tableau 1.1 présente les constructeurs canoniques communs des formalismes RDFS, OWL et PLIB.

Constructeurs	RDFS	OWL	PLIB
Classes			
Subsumption de classes	multiple	multiple	simple/ modulaire
Propriétés			
typage fort	-	±(*Lite*)	+
dépendance entre propriétés	-	-	+
domaine/co-domaine multiples	+	±(*Lite*)	-
subsumption de propriétés	+	+	-
fonctionnelle	-	+	+
inverse fonctionnelle	-	+	-
cardinalité	-	±	+
Types de données			
simple	+	+	+
classe	+	+	+
collection	+	+	+
Individus			
identification universelle	+	+	-
multi-instanciation	+	+	-
point de vue	-	-	+
méta-modélisation	+	±(*Full*)	-
détection d'erreur de données	-	-	+

Table 1.1 – Constructeurs de la couche canonique des formalismes RDFS, OWL et PLIB.

Ces formalismes définissent aussi des méta-attributs qui décrivent textuellement des concepts et ce, dans différentes langues naturelles. En d'autres termes, chaque formalisme fournit des constructeurs spécifiques pour la couche OL comme on peut le voir sur le tableau 1.2 donnant les différents descripteurs des formalismes étudiés.

3.4.2 Différences entre les formalismes d'ontologies

Les formalismes d'ontologies diffèrent par leurs objectifs : certains sont orientés gestion et échanges des données (PLIB, RDFS) et d'autres sont orientés inférence (OWL). Les formalismes orientés gestion et échanges des données définissent un langage canonique et sont plus indiqués pour des bases de données. Les contraintes qu'ils définissent dans les ontologies sont considérées comme des contraintes d'intégrité dans les BD et permettent la détection d'er-

Constructeurs	RDFS	OWL	PLIB
Classe / Propriété			
commentaire	+	+	-
label	+	+	-
référence	-	+	-
source	+	+	-
définition	-	-	+
note	-	-	+
remarque	-	-	+
nom court	-	-	+
nom préféré	-	-	+
figure	-	-	+
document de définition	-	-	+
Individus			
commentaire	+	+	-
label	+	+	-
description multilingue	+	+	+

Table 1.2 – Descripteurs de la couche linguistique des formalismes RDFS, OWL et PLIB.

Constructeurs	RDFS	OWL	PLIB
Classes			
Intersection	-	+	±
Union	-	$\pm(DL/Full)$	-
Complement	-	$\pm(DL/Full)$	-
Énumération d'instances	-	$\pm(DL/Full)$	±
Restriction			
co-domaine	-	+	-
cardinalité	-	+	-
égalité/inégalité de valeur	-	$\pm(DL/Full)$	-
Caractéristiques des propriétés			
réflexivité, symétrie, transitivité	-	+	-
inverse	-	+	-
Déduction de faits	-	+	-
Vérification d'intégrité	-	-	+

Table 1.3 – Constructeurs de la couche non canonique des formalismes RDFS, OWL et PLIB.

reurs dans les données. Les formalismes orientés inférence se focalisent sur la définition des concepts définis (non primitifs). Ils proposent ainsi un ensemble de constructeurs d'équivalence conceptuelle (union, intersection, équivalence, ...) pouvant être combinés pour définir les nouvelles classes et pour raisonner sur ces différentes classes. Certains formalismes n'offrent pas de constructeur d'ontologies de la couche non canonique (OCNC). Nous pouvons citer le cas de RDFS qui n'offre pas constructeur de la couche non canoniques (Tableau 1.3).

4 Ontologies vs. Modèles Conceptuels

Les ontologies présentent des similitudes avec les modèles conceptuels classiques sur le principe de la modélisation car tous deux définissent une conceptualisation de l'univers du discours au moyen d'un ensemble de classes auxquelles sont associées des propriétés [41]. Les ontologies et les modèles conceptuels divergent cependant sur un point essentiel : l'objectif de modélisation qui est à l'origine de la contribution d'une ontologie dans une démarche de modélisation conceptuelle. Une ontologie est ainsi construite selon une approche descriptive, par opposition à un modèle conceptuel qui est construit selon une approche prescriptive. Une approche prescriptive a les implications suivantes [42]:

- seules les données pertinentes pour l'application cible sont décrites ;
- les données doivent respecter les définitions et contraintes définies dans le modèle conceptuel ;
- aucun fait n'est inconnu : c'est l'hypothèse du monde fermé ;
- la conceptualisation est faite selon le point de vue des concepteurs et avec leurs conventions ;
- le modèle conceptuel est optimisé pour l'application cible.

Ces caractéristiques, dues au fait qu'un modèle conceptuel dépend fortement du contexte dans lequel il a été conçu, sont à l'origine des hétérogénéités identifiées lors de l'intégration et de l'échange de données issues de sources hétérogènes [43]. Contrairement à un modèle conceptuel qui prescrit une base de données selon des besoins applicatifs (orientés application), une ontologie est développée selon une approche descriptive (orientée domaine). Elle permet de décrire les concepts et propriétés d'un domaine donné indépendamment de tout objectif applicatif et de tout contexte hormis le domaine sur lequel porte l'ontologie. En plus de la capacité de conceptualisation, les ontologies apportent d'autres dimensions que nous avons citées dans les sections précédentes : identification des concepts, la consensualité et le raisonnement.

- **Identification des concepts** : les concepts dans une ontologie possèdent des identificateurs universels (par exemple, un URI) leur permettant d'être référencés par des applications externes.
- **Consensualité** : l'aspect consensuel des ontologies facilite la tâche des concepteurs qui travaillent sur divers projets référençant les mêmes ontologies et qui souhaitent partager et échanger des données sur leurs modèles.

– **Raisonnement** : les ontologies, de par leur aspect formel, offrent des mécanismes de raisonnement permettant de vérifier la consistance des informations et d'inférer de nouvelles données.

A la lumière de ces différences, il est par conséquent intéressant de considérer une ontologie comme un premier niveau de spécification d'un modèle conceptuel. Nous citons comme exemple les trois travaux suivants proposés par Roldan-Garcia et al. [44], Sugumaran et al. [45] et Fankam et al. [42] qui permettent de définir le modèle conceptuel d'une base de données à partir d'une ontologie supposée préexistante.

Ayant présenté les concepts fondamentaux liés aux ontologies et à leurs modèles, dans la section suivante, nous introduisons la notion de base de données à base ontologique, leurs modèles de stockage et architectures afin de répondre aux deux dernières questions que nous avons posées dans l'introduction.

5 Base de Données à Base Ontologique

Dans de nombreux domaines, les ontologies sont utilisées pour expliciter la sémantique des données manipulées dans une application. La forte volumétrie des données décrites par des ontologies a entraîné le problème de passage à l'échelle. En conséquence, un nouveau type de base de données a été créé, appelé Bases de Données à Base Ontologique ($\mathcal{BDBO}$). Dans cette section, nous détaillons ce type de base de données et leur diversité.

Définition 1

On appelle base de données à base ontologique (appelée également base de données sémantique) une source de données qui contient des ontologies, un ensemble de données et des liens entre ces données et les éléments ontologiques qui en définissent le sens [46, 9]. Les données (instances) contenues dans une $\mathcal{BDBO}$ sont appelées des données (instances) à base ontologique.

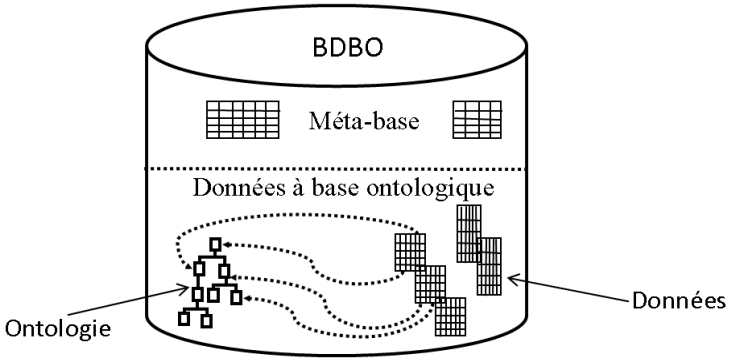


FIGURE 1.5 – Base de données à base ontologique

Une $\mathcal{BDBO}$ possède deux caractéristiques illustrées dans la figure 1.5 :

– les ontologies et les données sont toutes deux, représentées dans une unique base de données et peuvent faire l'objet des mêmes traitements (insertion, mise à jour, interrogation, etc.) ;
– toute donnée est associée à un élément ontologique qui en définit le sens et vice versa.

Plusieurs $\mathcal{BDBO}$ académiques et industrielles sont proposées dans la littérature parmi lesquelles on peut citer : RDFSuite [5], Jena [7, 47], Sesame [48], OntoDB [9], OntoMS [49], DLDB [8], RStar [50], KAON [51], 3Store [52] et PARKA [53], Oracle Semantic [54, 55], IBM Sor [56], etc.

En examinant ces $\mathcal{BDBO}$ nous avons identifié une grande diversité concernant les éléments suivants :
– le modèle d'ontologies supporté ;
– le schéma de base de données (modèle de stockage) utilisé pour stocker des ontologies ;
– le schéma de base de données utilisé pour représenter les données à base ontologique (instances) ;
– les architectures utilisées pour représenter l'ensemble des informations.

Nous détaillons dans ce qui suit les modèles de stockage et les architectures des $\mathcal{BDBO}$.

5.1 Modèle de stockage traditionnel ou autres modèles pour les $\mathcal{BDBO}$?

On peut remarquer dans la définition d'une $\mathcal{BDBO}$, que l'on a d'une part l'ontologie et d'autre part les données ontologiques (instances). Il est ainsi nécessaire de définir un schéma de tables pour le stockage de chaque partie et de définir un mécanisme liant les instances aux éléments de l'ontologie. Pour le stockage, le schéma d'ontologie et les instances ontologiques peuvent être stockés conjointement en utilisant le même modèle de stockage ou séparément en utilisant des modèles de stockage différents. Les différents modèles de stockage des ontologies et ceux de représentation des données à base ontologiques sont présentés ci-dessous.

5.1.1 Modèle de stockage des ontologies

Le problème de représentation des ontologies dans une base de données consiste à définir l'ensemble des tables nécessaires pour stocker les concepts ontologique et leurs relations. Dans la littérature, deux approches principales ont été proposées : la représentation générique et la représentation spécifique.

5.1.1.1 Représentation générique. Cette représentation appelée aussi *représentation verticale* utilise une unique table à trois colonnes *(sujet, prédicat, objet)* dite table de triplets pour stocker l'ontologie. Cette représentation est générique par rapport au modèle d'ontologie. En effet, l'ontologie est donc décomposée en un ensemble de triplets. Chaque élément E d'une ontologie est défini par des triplets de l'une des formes suivantes :

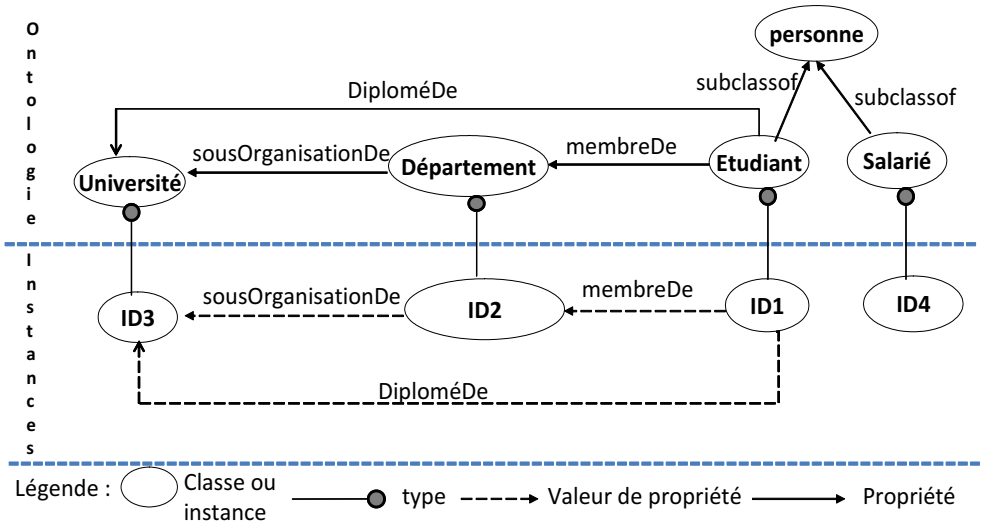


FIGURE 1.6 – Fragment de l'ontologie LUBM

– la forme *(E, rdf:type, entité)* : ces triplets permettent d'indiquer le type de l'élément. Ce type est une des entités définies en RDF-Schéma comme par exemple *rdfs:Class* ou *rdfs:Property* ;
– la forme *(E, attribut, valeur)* : ces triplets permettent de caractériser les éléments définis en leur assignant des valeurs d'attributs RDF-Schéma comme par exemple *rdf:label* ou *rdf:comment*.

Exemple 1

Soit le fragment de l'ontologie LUBM présenté sur la figure 1.6. Sa représentation générique est présentée sur la figure 1.7.

5.1.1.2 Représentation spécifique. Cette représentation dépend du modèle d'ontologies supporté. Elle consiste à représenter le modèle d'ontologies en utilisant le modèle relationnel ou relationnel-objet supporté par le SGBD sous-jacent à la $\mathcal{BDBO}$. Le schéma de tables est défini de façon ad hoc (non systématique) d'une implémentation à une autre. Il est fonction des détails des informations qu'on veut capturer et du lien établi avec la partie concernant les données. Chaque $\mathcal{BDBO}$ définit son propre schéma en fonction des concepts du modèle d'ontologies qu'elle supporte. Pour une ontologie RDFS, le schéma contient généralement les tables ; *class, subclass, domain, range, property, subproperty*.

Cette représentation est adoptée par les $\mathcal{BDBO}$ Sesame, RDFSuite, RSTAR, OntoDB, OntoMS, DLDB, PARKA et KAON. La représentation spécifique de notre fragment d'ontologie de la figure 1.6 est illustrée par la figure 1.8

Dans cet exemple, la représentation comporte les tables *Class, Property et Subclass*. La table *Class* stocke les classes des ontologies. Elle est composée d'une colonne *URI* permettant de sto-

Triplet		
sujet	**prédicat**	**objet**
http://lias.fr#Université	rdf:type	rdfs:Class
http://lias.fr#Département	rdf:type	rdfs:Class
http://lias.fr#Personne	rdf:type	rdfs:Class
http://lias.fr#Etudiant	rdf:type	rdfs:Class
http://lias.fr#Salarié	rdf:type	rdfs:Class
http://lias.fr#Etudiant	rdfs:suclassOf	http://lias.fr#Personne
http://lias.fr# Salarié	rdfs:suclassOf	http://lias.fr#Personne
http://lias.fr#estMembreDe	rdf:type	rdf:Property
http://lias.fr#estMembreDe	rdfs:domain	http://lias.fr#Etudiant
http://lias.fr#estMembreDe	rdfs:range	http://lias.fr#Département
http://lias.fr#sousOrganisation De	rdf:type	rdf:Property
http://lias.fr# Université	rdfs:comment	C'est une université de ...
…	…	…

FIGURE 1.7 – Représentation générique du fragment d'ontologie LUBM

Class			
Id	**URI**	**label**	**comment**
1	http://lias.fr#Université	Université	C'est une université de ...
2	http://lias.fr#Département	Département	
3	http://lias.fr#Personne	Personne	
4	http://lias.fr#Etudiant	Etudiant	
5	http://lias.fr#Salarié	Salarié	

SubClassOf	
sub	**sup**
4	3
5	3
…	…

Property					
id	**URI**	**comment**		**domain**	**range**
11	http://lias.fr#estMembreDe	appartenance	...	4	2
12	http://lias.fr#sousOrganisationDe	Sous-organe		2	1

FIGURE 1.8 – Représentation spécifique du fragment d'ontologie LUBM

cker l'identifiant externe d'une classe. Afin d'optimiser les traitements, un identifiant interne à la base de données (*id*) est également associé aux classes. Cette table permet également de stocker les noms (*label*) associés aux classes. Notons que si nous souhaitons associer plusieurs noms et/ou plusieurs commentaires à une même classe, la normalisation de cette table nécessite de définir deux nouvelles tables pour stocker ces informations. La table *Property* stocke les propriétés. Comme les classes, les propriétés sont associées à un identifiant interne (*id*) et externe (*URI*) et à des noms (*label*) et des commentaires (*comment*). Le domaine (*domain*) et co-domaine (*range*) des propriétés sont également spécifiés dans cette table par de références aux classes. La table *SubClassOf* permet de stocker la hiérarchie des classes en indiquant pour chaque classe ses superclasses.

5.1.2 Modèle de stockage des instances ontologiques

Trois principales approches sont utilisées pour la représentation des instances ontologiques au sein des $\mathcal{BDBO}$ [57] : *approche verticale, approche binaire* et *approche horizontale*. Ces trois approches sont présentées dans les paragraphes suivants.

5.1.2.1 Approche verticale. Cette approche consiste à représenter l'ontologie et ses instances par une table à trois colonnes. Ces colonnes représentent respectivement : (1) l'identifiant de la ressource ontologique (classe, propriété ou instance ontologique), (2) le nom de la ressource, et (3) la valeur de cette ressource. Dans cette représentation, chaque instance *i* d'une classe est définie par les triplets suivants :

- les triplets *(i, rdf:type, C)* qui permettent d'indiquer les classes auxquelles l'instance *i* appartient ;
- les triplets *(i, prédicat, valeur)* qui permettent de caractériser l'instance en lui assignant des valeurs de propriétés.

Cette représentation est simple et indépendante de l'ontologie utilisée. Elle facilite l'insertion de nouveaux triplets. La mise à jour d'une information peut nécessiter la modification de plusieurs triplets. L'interrogation est souvent complexe car elle peut nécessiter plusieurs opérations d'auto-jointure. La $\mathcal{BDBO}$ Sesame et celle d'Oracle utilise cette représentation pour la représentation des instances ontologiques.

Une variante de cette représentation appelée *approche verticale avec ID* consiste à utiliser des tables dictionnaires, c'est-à-dire des tables stockant les identifiants et leurs valeurs littérales. Une table dictionnaire est prévue pour des ressources et une autre pour les littéraux. La table de triplets utilise les identifiants des ressources et des littéraux référencés dans les tables *dictionnaire*. La représentation verticale des instances de notre fragment d'ontologie de la figure 1.6 est présentée sur la figure 1.9. Sa variante avec *ID* est présentée sur la figure 1.10

Triplet		
sujet	**prédicat**	**objet**
http://lias.fr#ID1	rdf:type	http://lias.fr# Etudiant
http://lias.fr#ID2	rdf:type	http://lias.fr#Département
http://lias.fr#ID3	rdf:type	http://lias.fr#Université
http://lias.fr#ID4	rdf:type	http://lias.fr#Salarié
http://lias.fr#ID1	http://lias.fr#estMembreDe	http://lias.fr#ID2
http://lias.fr#ID2	http://lias.fr#sousOrganisationDe	http://lias.fr#ID3
http://lias.fr# ID3	rdfs:comment	Université de Poitiers …
http://lias.fr# ID1	http://lias.fr#diplomédeDe	http://lias.fr# ID3
…	…	…

FIGURE 1.9 – Représentation verticale des instances du fragment d'ontologie LUBM

Ressource	
ID	URI
C1	http://lias.fr#Université
C2	http://lias.fr#Département
C3	http://lias.fr#Etudiant
C4	http://lias.fr#Salarié
P0	rdf:type
P1	http://lias.fr#estMembreDe
P2	http://lias.fr#estMembreDe
P3	http://lias.fr#diplomédeDe
P4	rdfs:comment
I1	http://lias.fr#ID1
I2	http://lias.fr#ID2
I3	http://lias.fr#ID3
I4	http://lias.fr#ID4
…	…

Litteral	
ID	URI
C1	http://lias.fr#Université
C2	http://lias.fr#Département
C3	http://lias.fr#Etudiant
P1	http://lias.fr#estMembreDe
P2	http://lias.fr#estMembreDe
L1	Université of Poitiers ....
..	…

Triplet		
sujet	**prédicat**	**objet**
I1	P0	C3
I	P0	C2
I3	P0	C1
I4	P0	C4
I1	P1	I2
I2	P2	I3
I3	P4	L1
I1	P3	I3
…	…	…

FIGURE 1.10 – Représentation verticale avec *ID* des instances du fragment d'ontologie LUBM

5.1.2.2 Approche binaire. Cette représentation consiste à décomposer les relations en deux catégories : les relations unaires (pour l'appartenance aux classes), et les relations binaires (pour les valeurs de propriétés). L'approche binaire se décline en trois variantes selon l'approche adoptée pour la représentation de l'héritage :

1. une table unique pour toutes les classes de l'ontologie. Cette table est constituée de deux colonnes *(Uri, ClassID)* : la colonne *Uri* désigne l'identifiant des instances et la colonne *classId* représente l'identifiant de la classe stockée dans l'ontologie ;
2. une table par classe avec héritage de table (si une base de données relationnelle-objet est utilisée). Cette représentation est appelée *ISA* [58]. Elle associe à chaque classe et propriétés stockées dans la partie ontologie une table spécifique pour leurs instances respectives. Les propriétés des superclasses sont reprises dans les sous-classes. Les tables de classes sont unaires tandis que les tables de propriétés sont binaires. Elles sont constituées de la colonne *id* pour les identifiants d'une instance et de la colonne *value* pour les valeurs de cette instance. Cette solution profite des fonctionnalités de SQL99 pour les requêtes hiérarchiques mais les mises à jour sont difficile à gérées. Par exemple, l'insertion d'une classe entre deux classes demande de supprimer des tables puis de les remettre en relation ;
3. une table par classe sans héritage de table. Cette variante, appelée *NOISA* [58], n'utilise pas l'héritage des tables offert par les SGBD relationnels objets. Les tables des propriétés et des classes sont définies séparément sans mises en relation. Les requêtes polymorphes nécessitent donc un accès à l'ontologie pour récupérer leurs classes et leurs propriétés. Cette variante entraîne de mauvaises performances pour les requêtes polymorphes car les calculs de sous-classes et des sous-propriétés sont récursifs et par conséquent coûteux [4].

La figure 1.11 illustre ces approches. La $\mathcal{BDBO}$ d'IBM SOR utilise cette approche binaire pour la représentation des ontologies et de leurs instances [59].

Cette approche a l'avantage d'être efficace en temps de réponse pour des requêtes simples. Cependant des requêtes complexes peuvent nécessiter plusieurs opérations de jointures [4]. Cette approche est utilisée par Sesame, RDF-Suite, DLDB et PARKA.

5.1.2.3 Approche horizontale. Cette approche est similaire à la représentation traditionnelle utilisée par les SGBD relationnels. Elle consiste à stocker toutes les instances des classes ainsi que leurs valeurs de propriétés dans une seule table relationnelle. L'identifiant d'instance, le nom de la classe et toutes ses propriétés sont stockés dans une entrée de cette table.

Cette approche est illustrée sur la figure 1.12. Cette approche est simple mais présente des inconvénients qui sont :
- le grand nombre de colonnes pour la table ;
- le nombre de valeurs de propriété limité à un ;
- le grand nombre de champs nuls ;

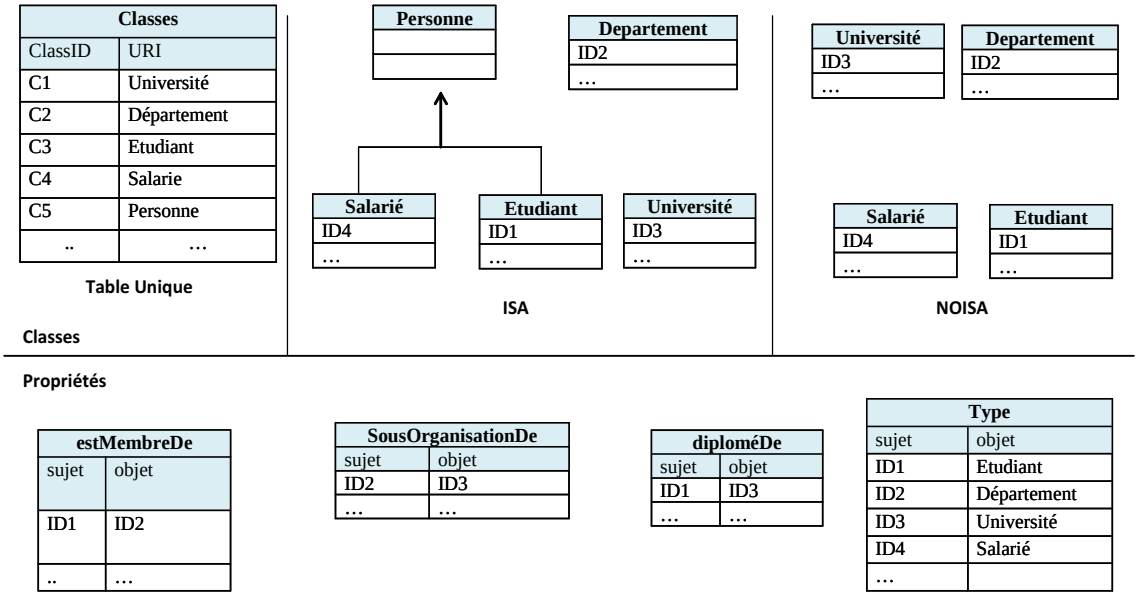


FIGURE 1.11 – Représentation binaire des instances du fragment d'ontologie LUBM

– la maintenance difficile : toute modification de l'ontologie entraine la restructuration de la table.

TableHorizontaleUnique					
ID	Type	estMembre	sousOrganisationDe	diplomeDe	…
http://lias.fr#ID1	Etudiant	ID2		ID3	
http://lias.fr#ID2	Departement		ID3		
http://lias.fr#ID3	Université				
http://lias.fr#ID4	Salarié				
....	... .	....			

FIGURE 1.12 – Table horizontale unique des instances du fragment d'ontologie LUBM

Une variante de cette approche consiste à utiliser une table pour chaque classe (Fig. 1.13). A chaque classe ontologique est associée une table ayant une colonne pour chaque propriété utilisée (une valeur associée) pour décrire au moins une instance de cette classe. Les $\mathcal{BDBO}$ OntoDB et OntoMS utilisent cette approche.

L'approche horizontale conserve la notion de schéma des données. Dans cette représentation, les tables utilisées représentent explicitement la structure des données. Cet aspect permet de faciliter l'intégration automatique des bases de données en utilisant cette $\mathcal{BDBO}$ [60]. Cette approche aide à résoudre des problèmes de conception de modèles conceptuels ou d'indexation sémantique de bases de données [61]. Cette variante est également efficace pour les requêtes portant sur les propriétés d'un individu ou d'un ensemble d'individus. Mais elle présente certains inconvénients de l'approche initiale comme les valeurs nulles car certains individus n'ont pas certaines propriétés.

Etudiant		
ID	estMembre	diploméDe
ID1	ID2	ID3

Université	
ID	comment
ID3	Université de Poitiers

Departement	
ID	sousOrganisationDe
ID2	ID

Salarié	
ID	…
ID4	

Personne	
ID	…

FIGURE 1.13 – Représentation horizontale des instances du fragment d'ontologie LUBM

NB : Il existe des $\mathcal{BDBO}$ qui combinent ces approches, dans ce cas on parle d'*approche hybride*.

Nous venons de voir les différentes approches de représentation des ontologies et des instances ontologiques dans des bases de données. Dans la section suivante, nous présentons les différentes architectures des $\mathcal{BDBO}$.

5.2 Architecture traditionnelle ou d'autres architectures pour le SGBD cible des $\mathcal{BDBO}$?

La présence d'une ontologie au sein du SGBD a permis de mettre en cause l'architecture initiale d'un SGBD contenant deux parties: le contenu et la méta-base. Deux tendances architecturales ont été identifiées: (i) garder l'architecture initiale d'un SGBD et (ii) faire évoluer cette dernière pour mieux prendre en compte la présence de l'ontologie. Dans les sections suivantes, nous détaillons ces deux tendances.

5.2.1 Garder l'existant

Les premières propositions de $\mathcal{BDBO}$ ont stocké les ontologies et leurs instances d'une façon similaire aux bases de données classiques en utilisant deux parties : les *données* et la *méta-base* aussi appelée *catalogue du système*. Nous appelons cette architecture, architecture de *type I* ou architecture "*deux quarts*". La partie "*données*" dans cette structure représente les instances ontologiques (instances de classes de l'ontologie et les valeurs de propriétés ontologiques) mais également le schéma de l'ontologie (classes, propriétés, etc.). La partie *"méta-base*" est la composante traditionnelle des bases de données classiques. Elle contient l'ensemble des tables systèmes permettant la gestion et le bon fonctionnement de l'ensemble des données contenues dans la base de données. Elle sert aussi de dictionnaire qui décrit les différents objets du système. Par exemple lorsqu'une table est créée, le *catalogue* enregistre sa structure et notamment ses colonnes avec leurs types de données, ses différentes contraintes, etc. Dans une $\mathcal{BDBO}$, toutes les tables et les attributs définis sont documentés dans la méta-base. Cette architecture, présentée sur la figure 1.14, impose ainsi le même modèle de stockage pour l'on-

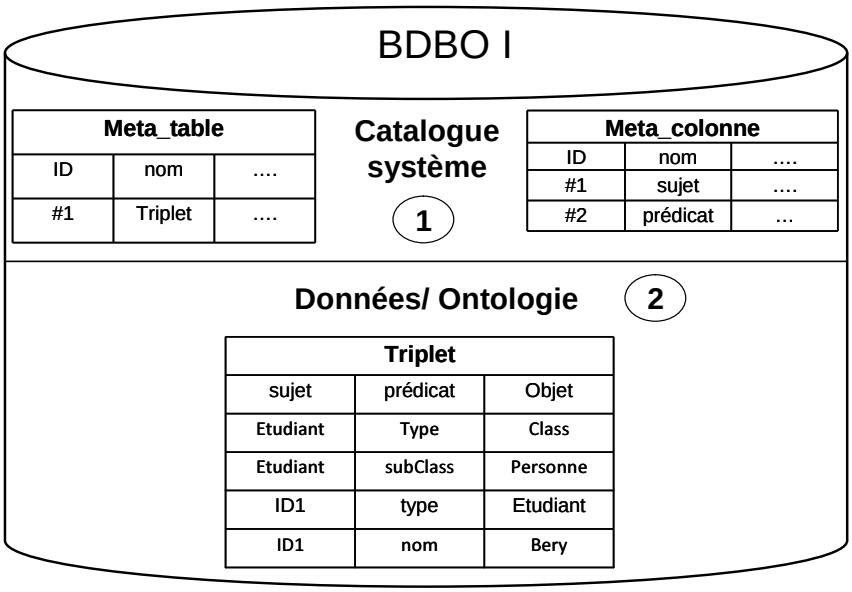


FIGURE 1.14 – Architecture *deux quarts*

tologie et pour les instances ontologiques. Jena et Oracle utilisent une architecture de ce type pour stocker les données (ontologie et instances) sous forme de triplets RDF (approche verticale). Cette architecture présente l'avantage d'être simple à mettre en œuvre. Les insertions des données sont faciles. Cependant elle présente l'inconvénient de tout stocker dans une seule table (modèle et instances) et ainsi les requêtes s'expriment en auto-jointures sur cette table volumineuse.

5.2.2 Faire évoluer l'architecture traditionnelle

Pour pallier cet inconvénient, une deuxième architecture dite architecture de *type II* ou architecture "*trois quarts*" (figure 1.15) a été proposée. Les instances ontologiques et l'ontologie sont stockées dans deux schémas différents. Cette nouvelle architecture scinde la base de données en trois parties : catalogue système, schéma du modèle ontologique, schéma des instances ontologiques. Les parties "*donnée*" et *"catalogue système"* jouent les mêmes rôles que précédemment. Le schéma du modèle ontologique est dédié au niveau ontologique. Il dépend du modèle d'ontologie utilisé (OWL, PLIB, RDFS, etc.). Il est composé de table(s) figée(s) pour le stockage des différents concepts ontologiques tels que les classes, les propriétés ou les relations de subsomption (sous-classes), etc. IBM SOR [59] est un exemple de $\mathcal{BDBO}$ suivant cette deuxième architecture.

Même si les deux schémas de stockage (modèle et instances) sont indépendants, cette architecture manque de flexibilité dans la mesure où elle impose un modèle d'ontologie figé, ce qui ne permet pas d'introduire de nouveaux concepts issus d'autres modèles d'ontologies. Une troisième architecture visant à combler cette insuffisance a été proposée. Cette troisième ar-

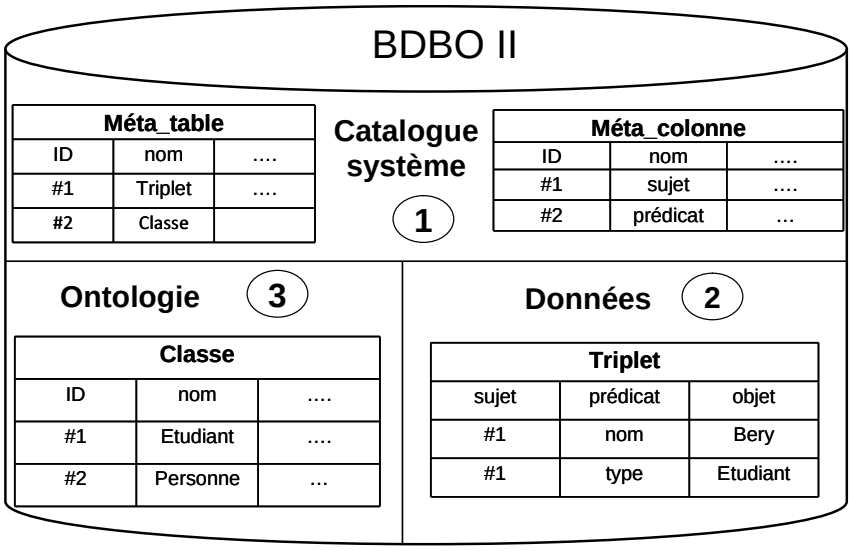


FIGURE 1.15 – Architecture *trois quarts*

chitecture qualifiée de *type III* (ou architecture "*quatre quarts*") a été proposée dans le cadre du projet *OntoDB* (OntoDB1 et OntoDB2) visant à répondre au besoin de flexibilité du modèle d'ontologie utilisé dans la $\mathcal{BDBO}$. Cette architecture propose, en plus des trois parties, une partie nommée *méta-schéma* qui joue, pour les ontologies, le même rôle que celui joué par le système catalogue pour les données. Le *méta-schéma* appelé méta-modèle représente, au sein d'un modèle réflexif, à la fois le modèle d'ontologie utilisé et le méta-schéma lui-même. Les modèles d'ontologies sont représentés comme des instances du méta-modèle. Cette partie *méta-schéma* permet un accès générique au modèle ontologique ainsi que l'introduction de nouveaux constructeurs issus de différents formalismes ontologiques. La figure 1.16 présente un schéma de données de la partie *méta-schéma* et de la partie *ontologie* de notre fragment d'ontologie LUBM. Cette architecture a été conçue de manière à ce qu'elle soit extensible afin de pouvoir y représenter d'autres modèles d'ontologies et ce grâce à sa partie méta-schéma. Cela donne la possibilité d'étendre le schéma de l'ontologie pour représenter les nouveaux concepts non supportés par les modèles d'ontologies existants. Le schéma de données de cette dernière étant susceptible d'évoluer ou d'être modifié ; l'auto-représentation du méta-modèle du modèle d'ontologie permet de rendre certains traitements génériques et/ou indépendamment du modèle d'ontologie utilisé.

A titre d'illustration, Belaid et al. [62] ont proposé d'étendre le modèle OntoDB pour la représentation de services informatiques et des concepts d'ontologies de services qui en définissent le sens. Khouri et al. [63] ont opté pour une telle architecture pour la représentation de la structure d'entrepôt de données intégrant le modèle des besoins. Cette vision qui permet de voir l'évolution des $\mathcal{BDBO}$ en termes de schémas de stockage et d'architectures est illustrée dans la figure 1.17.

Nous pouvons constater que cette architecture ressemble à l'architecture méta-données du

Entité		
ID	Nom	SuperEntiité
E#1	Ressource	
E#2	Class	E#1
E#3	Property	E#1
E#4	ObjectProperty	E#3
E#5	Data_property	E#1

Type	
ID	Nom
T#1	String
T#2	E#2
T#3	E#3

Attribut			
ID	Nom	Domaine	Co-domaine
A#1	name	E#1	T#1
A#2	Domain	E#1	T#1
A#3	Range	T#1	E#1
…	….	…	….

Méta-schéma

Ontologie

Class		
ID	Nom	Propriétés
C#1	Université	{D#1}
C#2	Departement	{O#2}
C#3	Etudiant	{O#1, O#3, D#1, D#2, ..}
C#4	Salarié	{O#1, D#1, D#2, …}
E#5	Personne	{D#1, D#2}

Object_Property			
ID	Nom	Domain	range
O#1	estMembreDe	C#3	C#2
O#2	sousOrganisationDe	C#2	C#1
O#3	diplomé De	C#2	C#1
…	…	…	

Data_Property			
ID	Nom	Range	….
D#1	nom	Strin g	
D #2	Age	Int	
…	…	…	

FIGURE 1.16 – Méta-schéma et ontologie

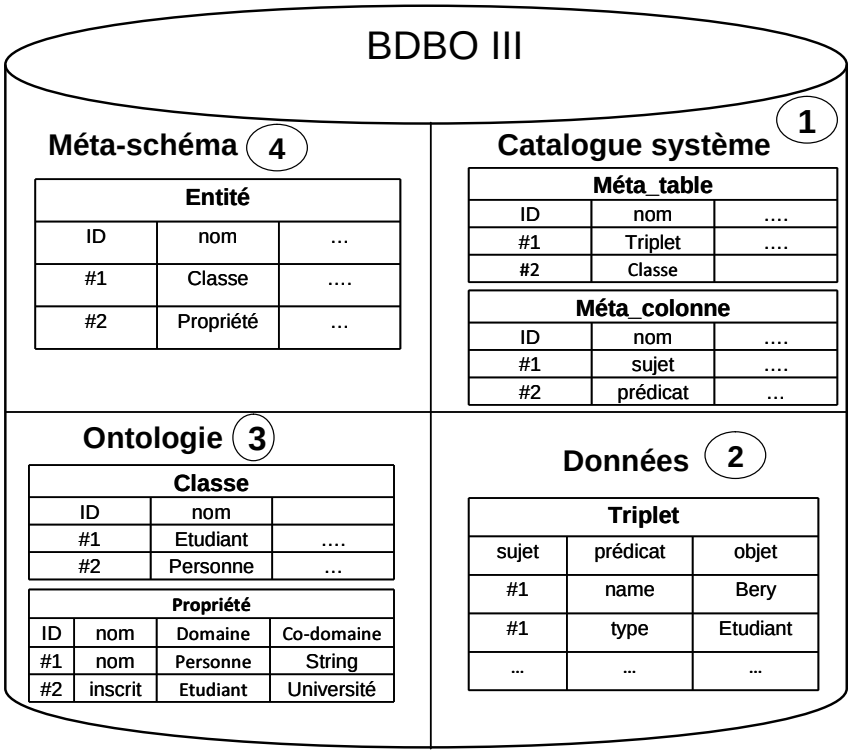


FIGURE 1.17 – Architecture *quatre quarts*.

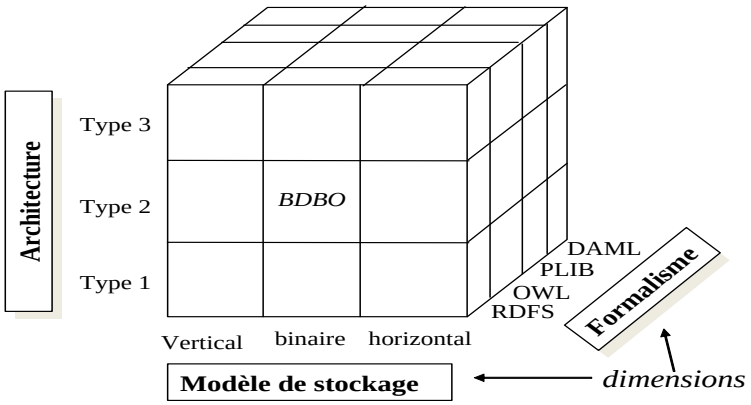


FIGURE 1.18 – Représentation multidimensionnelle des $\mathcal{BDBO}$.

MOF (Meta Object Facility) [64]. En effet, cette architecture est constituée des mêmes quatre couches superposées comme celles du MOF. La couche modèle *M1* de l'architecture MOF correspond au modèle conceptuel, un sous-ensemble de l'ontologie. Cette couche contient la couche *M0* représentant les instances (les données dans l'architecture de type III). La couche méta-modèle *M2* correspond au (méta-) modèle d'ontologie, la couche méta-méta-modèle *M3* (MOF model) correspond au méta-modèle, lui-même réflexif, du langage de définition du modèle d'ontologie.

5.2.3 Cube de $\mathcal{BDBO}$

En considérant les trois principaux points de la diversité des $\mathcal{BDBO}$ identifiée précédemment, nous pouvons représenter l'espace des $\mathcal{BDBO}$ par un cube dont les axes sont respectivement les modèles de stockage, les architectures et les formalismes d'ontologies. Les points dans cet espace c'est-à-dire les cellules du cube, sont les types [9] de $\mathcal{BDBO}$. La figure 1.18 donne une représentation de cet espace. Toute $\mathcal{BDBO}$ peut être classée dans une cellule de ce cube. Par exemple, la $\mathcal{BDBO}$ Oracle est classée dans la cellule de coordonnées *(vertical, type 1, RDFS)* [10] et la $\mathcal{BDBO}$ OntoDB dans la cellule *(horizontale, type 3, PLIB)* comme on peut le voir sur la figure 1.19.

5.2.4 Synthèse

L'étude faite dans ce chapitre montre que contrairement aux bases de données relationnelles qui sont toutes similaires en termes d'architecture et de modèle de stockage, les $\mathcal{BDBO}$ présentent une grande diversité. Le tableau 1.4, propose une classification dirigée par les ar-

9. A ne pas confondre avec le type d'architecture
10. Oracle est également classé dans d'autres cellules car il supporte plusieurs modèles d'ontologies

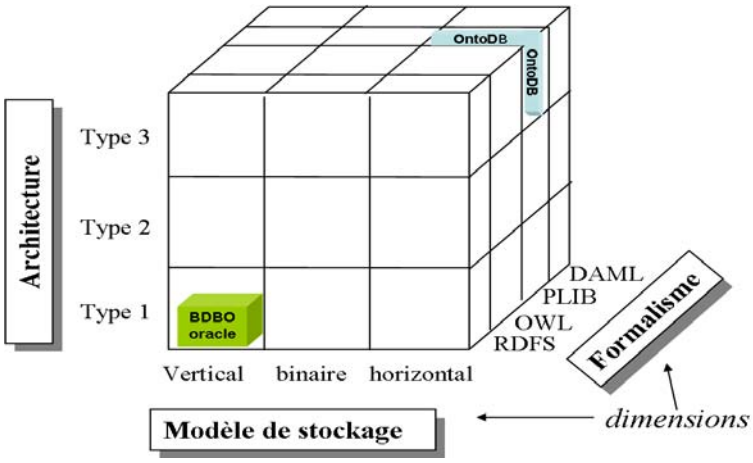


FIGURE 1.19 – Exemple de $\mathcal{BDBO}$ dans l'espace de $\mathcal{BDBO}$.

chitectures pour des exemples de $\mathcal{BDBO}$ couramment utilisées. On voit que la majorité de ces $\mathcal{BDBO}$ utilisent l'architecture *deux quarts* ou *trois quarts* (architecture de type1 ou de type2). OntoDB est la seule $\mathcal{BDBO}$ qui utilise l'architecture *quatre quarts* et propose à ses utilisateurs une possibilité d'extension du modèle d'ontologie.

$\mathcal{BDBO}$	$Type1$	$Type2$	$Type3$
Oracle	√		
Jena	√	√	
OntoDB			√
OntoMS		√	
RDFSuite		√	
Sesame	√	√	
SOR		√	
3Store		√	
RDFDB	√		
DLDB		√	
DB2RDF	√		

TABLE 1.4 – Classification des $\mathcal{BDBO}$ selon leur architecture

6 Langage de requêtes des $\mathcal{BDBO}$

Les données à base ontologique stockées dans une base de données sont susceptibles d'être sujettes à des consultations, des mises à jour ou des suppressions. Ces actions nécessitent un langage de manipulation (langage de requêtes). Chaque $\mathcal{BDBO}$ prévoit au moins un langage de

ce type. Comme dans les bases de données classiques, les utilisateurs qui veulent interroger des données ne doivent pas se préoccuper du modèle de stockage et des détails de mise en œuvre.

6.1 Principaux langages proposés

Plusieurs langages d'interrogation ont été proposés pour les données ontologiques. Bailey et al. [65] ont établi un état de l'art où ils ont classé ces langages en sept catégories qui se distinguent par le modèle de donnée supporté, leur expressivité et le support du schéma d'information: famille de SPARQL, famille de RQL, langages inspirés de XPath, XSLT ou XQuery, langages en anglais contrôlé, langages avec règles réactives, langages déductifs et autres langages.

La famille de SPARQL regroupe les langages qui traitent des triplets. Ces langages permettent d'interroger des données RDF sans tenir compte de la sémantique associée aux éléments des triplets. On retrouve dans cette catégories les langages SPARQL [14], RDQL [10], SquishQL [13], TriQL [66], etc. Le langage SPARQL, bien adapté à l'interrogation de données RDF et adopté par plusieurs $\mathcal{BDBO}$, a été retenu comme le standard pour les requêtes sémantiques par le W3C. Il est très utilisé dans de nombreux travaux dans le domaine du Web Sémantique. Nous le présenterons un peu plus en détail dans la section 6.2.

La famille RQL regroupe les langages qui prennent en compte les données et le schéma d'ontologie dans des interrogations. Le modèle de données RDF qu'ils supportent impose deux contraintes majeures : (1) les cycles sont interdits dans la relation de subsomption, et (2) pour chaque propriété, le domaine et le co-domaine doivent être définis. Cette famille regroupe les langages RQL[12], eRQL[67], SeRQL [11], OntoQL [15].

La famille des langages inspirés de XPath, XSLT ou XQuery regroupe des langages qui étendent un langage de requête XML. Certains sont mis en œuvre en ajoutant des fonctions supplémentaires ou en normalisant les données avant l'interrogation. On peut citer XQuery for RDF [68], XSLT for RDF [69], Versa [70], Path-Based Access to RDF (RDF Path, RPath [71], RxPath, RxSLT, et RxUpdate), etc.

La famille de langages en anglais contrôlé fait référence à Metalog [72], un système pour l'interrogation et le raisonnement dans le Web Sémantique. Metalog diffère notablement des autres langages de requêtes RDF par deux points: (1) Metalog combine l'interrogation et le raisonnement, et (2) la syntaxe du langage est en langage naturel (anglais).

Dans la famille de langages avec règles réactives, on trouve Algae [73], iTQL et WQL. Ces langages offrent des possibilités de définir dans des requêtes des règles de vérification.

La famille des langages déductifs renferme des langages basés sur un langage de règles de déduction et de transformation pour RDF. On trouve dans cette famille N3QL [74], R-DEVICE [75], TRIPLE [76], etc.

La famille "autres langages RDF" regroupe les langages de requêtes RDF qui ne font partie

d'aucune des familles précitées. On y rencontre RDF-QBE [77], RDFQL [78], etc.

Dans ce paragraphe, nous avons présenté les principales familles de langages de requêtes des données RDF. Dans les paragraphes suivants, nous présentons le langage SPARQL et le langage OntoQL que nous avons utilisés dans nos travaux.

6.2 Présentation du langage SPARQL

SPARQL [14] est un langage de requêtes pour des données représentées en RDF. Les requêtes SPARQL sont énoncées dans des « patrons de graphes élémentaires » qui servent de motifs de recherche des triplets de forme (*sujet*, *prédicat*, *objet*). SPARQL est donc capable de rechercher des patrons de graphe (*graph patterns*) obligatoires et optionnels ainsi que leurs conjonctions et leurs disjonctions. Les résultats des interrogations SPARQL peuvent être des ensembles de résultats ou des graphes RDF.

SPARQL propose quatre types de requêtes :

- les requêtes `SELECT` permettent d'extraire des informations d'une $\mathcal{BDBO}$ ou d'une source de données RDF interrogée ;
- les requêtes `CONSTRUCT` permettent de créer de nouveaux triplets à partir du résultat d'une requête ;
- les requêtes `DESCRIBE` permettent d'obtenir la description d'une ressource donnée. Les spécifications de SPARQL ne précisent pas la nature de la description d'une ressource. Elles imposent seulement que cette description soit un sous-ensemble du graphe interrogé ;
- les requêtes `ASK` retournent un booléen indiquant si la requête à une solution ou n'en a pas.

Les requêtes de type `UPDATE` sont réalisées grâce au langage SPARQL UPDATE (SPARUL [79]). Ce dernier utilise une syntaxe dérivée de SPARQL. Les opérations de mise à jour portent sur de collections de graphes dans un ensemble de données RDF. Ces opérations permettent de mettre à jour, de créer et de supprimer des sous-graphes RDF.

Nous présentons ci-dessous les requêtes de type SELECT qui nous intéressent dans la suite de ces travaux.

6.2.1 Requête SELECT.

La structure d'une requête SPARQL de type `SELECT` est similaire à celle du langage SQL. Elle se présente sous la forme générale [11] suivante :

11. Nous n'indiquons ici que les principales clauses d'une requête SPARQL.

PREFIX namespacesList
SELECT
liste_variables_exportées
[FROM
URL_sources_Données]
WHERE {

$$\left.\begin{array}{l} triple\_pattern_1 \\ triple\_pattern_2 \\ \ldots \\ triple\_pattern_n \end{array}\right\} = \text{patron de requête}$$

}
[Filter FilterExpression]
[modificateurs de solutions]

La clause `PREFIX` permet d'indiquer des alias sur les espaces de noms utilisés dans la requête. Un espace de nom (*namespace*) indique l'URI de l'ontologie utilisée. La clause `FROM` permet d'indiquer la ou les sources RDF interrogées. Cette clause est optionnelle. Si elle n'est pas spécifiée, l'environnement dans lequel la requête est exécutée est chargé de fournir la source de triplets. Par exemple, lorsqu'une requête SPARQL est exécutée sur une $\mathcal{BDBO}$, les données RDF interrogées sont celles contenues dans la $\mathcal{BDBO}$. La clause `WHERE` est constituée d'un ensemble de triplets pouvant contenir des variables (préfixées par `?`). Un interpréteur de requêtes SPARQL recherche les valeurs de ces variables pour lesquelles les triplets de la clause `WHERE` sont inclus dans le graphe RDF interrogé. Il retourne le sous-ensemble de ces valeurs correspondant aux variables spécifiées (*liste_variables_exportées*) dans la clause `SELECT`. SPARQL permet d'indiquer des conditions sur les variables utilisées dans la requête. Ces conditions sont définies grâce à l'opérateur `FILTER`. Cet opérateur prend en paramètre une expression booléenne. Ce résultat peut être modifié en spécifiant des modificateurs de séquence de solution (order by, distinct, reduced, offset, limit, ...) .

Exemple 2

La requête suivante retourne la valeur des propriétés `name` *et* `age` *pour les ressources de type* `Etudiant` *décrites par ces propriétés. Les résultats seront triés par ordre croissant sur les noms.*

```
  PREFIX lias: <http://www.ensma.fr/lias#>
  SELECT ?name ?age
   WHERE {         ?x rdf:type lias:Etudiant.
        ?x lias:name ?name .
         ?x lias:age ?age  }
ORDER BY ASC(?name)
```

Dans cette requête, la variable `?x` est introduite dans deux triplets de la clause `WHERE`. Cette variable représente les ressources RDF présentant une valeur pour les propriétés `name` et `email` définies sur l'ontologie dont l'espace de nom est `http://www.ensma.fr/lias`. La clause `SELECT` indique que cette requête retourne les noms (`?name`) et age (`?age`) de triplets obtenus comme résultats de la requête et la clause `ORDER BY` indique que les résultats seront donnés par ordre alphabétique croissant sur les noms (`ASC(?name)`).

SPARQL dispose également de l'opérateur `OPTIONAL` qui permet d'indiquer que des triplets sont optionnels dans la clause `WHERE`. Par conséquence, des résultats ne satisfaisant pas les triplets optionnels seront quand même retournés. Cependant, lorsqu'une variable utilisée uniquement dans des triplets optionnels n'a pas d'affectation correspondant aux données interrogées, la valeur `UNBOUND` est retournée. Par exemple, si nous modifions la requête de notre exemple 2 en ajoutant le patron de triplet optionnel `(?x lias:email ?email)`, (c'est-à-dire `OPTIONAL { ?x lias:email ?email }`), les ressources ne présentant pas les valeurs pour la propriété `email` seront quand même retournées avec la valeur `UNBOUND` pour la variable `?email`.

SPARQL offre également l'opérateur `UNION` pour joindre deux ensembles de triplets dans la clause `WHERE`. Pour une requête où deux ensembles de triplets sont liés par cet opérateur, un résultat est retourné dès lors qu'il permet de satisfaire l'un des deux ensembles de triplets.

6.2.2 Traitement d'une requête SPARQL

L'exécution d'une requête SPARQL passe par une série d'étapes parmi lesquelles on a :

1. le parsing de la requête SPARQL en un arbre de syntaxe abstrait (AST Abstract Syntax Tree)
2. la conversion de l'arbre de syntaxe abstrait en une requête abstraite (c'est-à-dire, une expression algébrique)
3. l'évaluation de la requête abstraite sur un ensemble de données RDF.

Le calcul d'une requête SPARQL de type SELECT se fait par appariement (graph pattern mapping) de graphe. Cela revient à chercher les sous-graphes qui apparient le patron de requête. L'évaluation porte sur l'expression algébrique de la requête. Or une requête peut avoir plusieurs expressions algébriques, il est important de choisir celle dont l'évaluation est bénéfique c'est-à-dire rapide et moins gourmande en ressources (CPU, mémoire). Il revient à l'optimiseur de requêtes de faire ce travail.

6.3 Présentation du langage OntoQL

Le langage OntoQL [15], développé au LIAS (Laboratoire d'Informatique, d'Automatique pour les Systèmes) est un langage d'exploitation des bases de donnés à base ontologique. Il est implémenté sur la $\mathcal{BDBO}$ OntoDB. OntoQL permet de définir, modifier et interroger les ontologies et les données à base ontologique suivant les différentes couches du modèle en oignon

présenté précédemment. En outre, OntoQL permet d'interroger simultanément les ontologies et les données d'une $\mathcal{BDBO}$ et propose les opérateurs traditionnels des bases de données. La syntaxe du langage OntoQL est proche de celle de SQL dans les différentes couches d'accès.

6.3.1 Requête SELECT.

La forme générale d'une requête OntoQL de type SELECT ressemble à celle de SQL. Sa synthèse est la suivante :

SELECT
liste_sélection
[FROM
Liste_de_Reférence_Données]
[WHERE {
Conditionde_recherche }
[modificateurs de solutions]
Using NAMESPACE<*Liste_namespace_definition*>
Using LANGUAGE <*LanguageID*>

Hormis les clauses *Using NAMESPACE* et *Using LANGUAGE*, toutes les clauses de cette syntaxe sont similaires à celles de SQL. La clause `SELECT` permet de préciser les attributs du résultat d'une requête, d'appeler des fonctions ou des expressions arithmétiques comme dans SQL. La clause `WHERE` est optionnelle, et permet de définir les conditions de restrictions. La clause `FROM` référence les objets (par exemple, les classes) sur lesquels porte la requête. La clause *USING NAMESPACE* indique que la requête doit être exécutée sur des éléments ontologiques appartenant à l'ontologie indiquée en paramètre. La clause *Using LANGUAGE* permet d'indiquer la langue naturelle utilisée.

Exemple 3

La requête de l'exemple 2 est exprimée en OntoQL par :

```
SELECT  E.name, E.age
FROM Etudiant E
Using NAMESPACE 'http://www.these.edu/Ontologies/LumbOntoDB.owl'
ORDER BY ASC(E.name)
```

La requête SELECT de OntoQL permet d'accéder aux différentes parties d'une $\mathcal{BDBO}$. Par exemple, la requête *SELECT #namespace FROM #Ontology* accède à la *partie Ontologie* et de rechercher les espaces de noms des ontologies stockées dans cette $\mathcal{BDBO}$.

6.3.2 Traitement d'une requête OntoQL

Le traitement d'une requête OntoQL passe par sept étapes comme indiqué sur la figure 1.20. Ces étapes sont :

1. génération de l'expression algébrique OntoAlgebra [12] correspondant à la requête OntoQL ;
2. optimisation de l'expression algébrique OntoAlgebra ;
3. transformation de l'arbre algébrique OntoAlgebra en un arbre algébrique utilisant des opérateurs de l'algèbre relationnelle étendue avec les opérateurs disponibles dans les SGBD relationnels-objets ;
4. optimisation de l'expression de l'arbre algébrique relationnel. Cette étape permet de faire des optimisations qui ne sont pas réalisées par le SGBD comme par exemple simplifier une requête imbriquée ;
5. transformation de l'expression de l'algèbre relationnelle en une requête SQL conforme au SGBD sur lequel la $\mathcal{BDBO}$ OntoDB est implantée (PostgreSQL) ;
6. exécution de la requête SQL sur OntoDB en utilisant le JBDC. Le résultat retourné est ainsi un ResultSet ;
7. transformation du ResultSet pour retourner le résultat de la requête OntoQL.

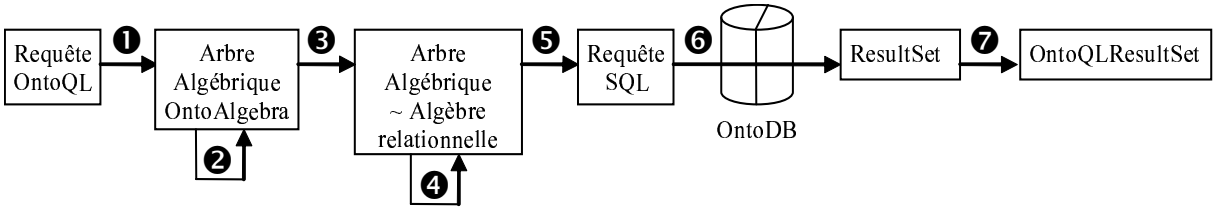


FIGURE 1.20 – Etapes du traitement d'une requête OntoQL (extraite de [28])

7 Conclusion

Dans ce chapitre, nous avons présenté la notion d'ontologie ainsi que ses caractéristiques principales. Plusieurs définitions sont proposées dans la littérature mais celle qui est communément citée est celle de Gruber : « an explicit specification of a conceptualization ». Plusieurs extensions de cette définition ont été proposées. Nous retenons celle de Pierra [20] qui décrit une ontologie comme une représentation formelle, explicite, référençable et consensuelle de l'ensemble des concepts partagés d'un domaine en termes de classes d'appartenance et de

12. Algèbre *Encore* adaptée aux modèles de données d'une $\mathcal{BDBO}$

propriétés caractéristiques. Suivant la nature des concepts utilisés dans l'ontologie, trois types d'ontologies existent : (i) les ontologies conceptuelles canoniques qui ne contiennent que des concepts primitifs, (ii) les ontologies conceptuelles non canoniques qui en plus des concepts primitifs, renferment dans leurs définitions des concepts définis et (iii) les ontologies linguistiques qui visent à représenter les mots utilisés dans un domaine donné. Elles fournissent une représentation en langage naturel des concepts d'un domaine. Les représentations peuvent être multilingues. Pour construire des ontologies, on fait appel à des formalismes comme RDFS et PLIB souvent utilisés dans le domaine de l'Ingénierie (ils sont orientés vers la gestion et l'échange des données), DAMLONT, DAML+OIL, FLogic et OWL dans le domaine de l'Intelligence Artificielle et du Web (ils sont orientés vers l'inférence des données). Sur la base de ces ontologies, la notion de bases de données à base ontologiques a été détaillée. Dans le monde des ontologies et de leur utilisation, la diversité est bien présente. Elle concerne à la fois leur usage par les différentes communautés de recherche (IA, BD, TALN ...) et la conséquence de leur mise en œuvre dans les bases de données. Dans ce contexte la diversité concerne les modèles d'ontologies, les modèles de stockage et les architectures du SGBD cible. Nous nous sommes également intéressés aux langages d'exploitation des bases de données sémantiques. Nous avons étudié deux langages principaux à savoir SPARQL et OntoQL. SPARQL est adopté par le consortium W3C comme le langage standard pour les $\mathcal{BDBO}$. Il est utilisé sur la plupart de $\mathcal{BDBO}$. OntoQL est un langage développé dans notre laboratoire pour exploiter les ontologies et les instances ontologiques.

Dans le chapitre suivant, nous étudions le problème de la conception physique dans les bases de données traditionnelles, en général, et dans les $\mathcal{BDBO}$ en particulier. Nous étudions l'impact de la diversité des $\mathcal{BDBO}$ sur la conception physique, une phase importante dans le cycle de vie de conception des bases de données avancées. L'objectif de ce chapitre est d'étudier en détail la conception physique traditionnelle, et ensuite de la généraliser pour l'instancier dans le cadre des $\mathcal{BDBO}$.

Chapitre 2

Conception Physique de Bases de Données

Sommaire

1 Introduction

L'optimisation de requêtes est un enjeu important dans le contexte de bases de données. Elle a toujours eu une place importante à travers les différentes générations de bases de données: les bases de données traditionnelles, les bases données XML, les bases de données décisionnelles, les bases de données statistiques et scientifiques, les bases de données sémantiques, et le big data. Puisque les applications de bases de données sont toujours à la recherche de temps de traitement de requêtes plus performant, plusieurs travaux ont été menés pour rendre les optimiseurs de requêtes plus efficaces. Pour atteindre cet objectif, les optimiseurs de requêtes ont pris en compte progressivement des paramètres issus des phases du cycle de vie de la conception de base de données (à savoir, la modélisation conceptuelle, la modélisation logique ETL, la modélisation physique et le déploiement (Figure 2.1)) et le langage de requêtes utilisé par le SGBD cible.

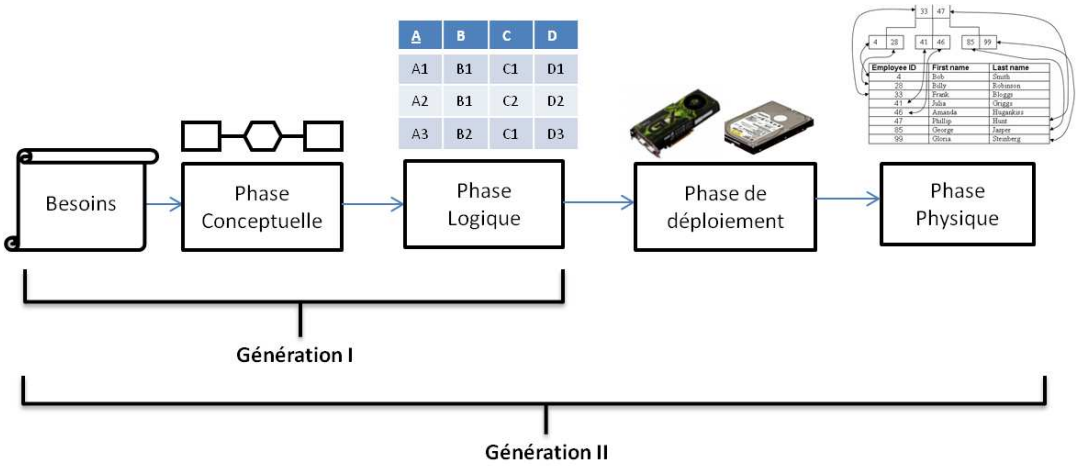


FIGURE 2.1 – Problème de conception physique

Deux générations principales d'optimiseurs de requêtes existent : *des optimisations dirigées par certaines phases de cycle de vie* ($\mathcal{ODCP}$) et d'autres dirigées par toutes les phases ($\mathcal{ODTP}$). La première génération des optimiseurs a été basée principalement sur l'étude des propriétés algébriques du langage de requêtes défini sur *le modèle logique* de la base de données à concevoir. Le *Rule-based Approach (RBA)* est un exemple de ce type d'optimisation. Plus précisément, le *RBA* utilise un ensemble de règles intuitives supposé optimiser les requêtes. Nous pouvons citer par exemple, la règle consistant à descendre les opérations de la sélection et de projection dans un arbre algébrique. Cette règle permet de réduire la taille des relations et des résultats intermédiaires manipulés. Ce type d'optimisation est simple à implémenter, pour cette raison, elle a été considérée dans l'ensemble des SGBD commerciaux et académiques. La principale limite de cette optimisation est qu'elle ignore les paramètres physiques liés à la base de données et la plateforme sur laquelle le SGBD est déployé. Les optimisations de type $\mathcal{ODTP}$ sont apparues pour pallier aux limites de la première génération d'optimisations. Comme son nom l'indique,

une optimisation de type $\mathcal{ODTP}$ prend en compte des paramètres conceptuels, physiques et de déploiement. L'approche basée sur des modèles de coût mathématiques, *Cost-Based Approach (CBA)* est un exemple de cette génération. Cette approche consiste à d'abord calculer le coût des différentes stratégies possibles correspondant aux plans d'exécution d'une requête en fonction des caractéristiques des fichiers sur lesquels sont implantées les relations, pour ensuite retenir le plan ayant le coût minimal. Pour illustrer le fonctionnement de cette optimisation, considérons l'exemple suivant:

Exemple 1

Considérons une requête impliquant une opération de jointure entre deux tables relationnelles R et S stockées sur un disque. Si nous souhaitons calculer le coût de cette jointure en utilisant une implémentation par hachage sachant qu'il existe plusieurs algorithmes pour implémenter une opération de jointure (boucles imbriquées, le tri, etc.), nous aurons besoin des paramètres issus de l'ensemble de phases du cycle de vie:

- ***la couche conceptuelle:*** *la longueur de chaque attribut de chaque table;*
- ***la couche logique:*** *la taille (en termes d'instances) de chaque table;*
- ***la couche physique:*** *le modèle de stockage utilisé pour stocker les tables (row store, column store)*
- ***la couche de déploiement:*** *les caractéristiques de disque comme la taille d'une page disque.*

Le coût de cette jointure, dénoté par JHash, est donné par la formule suivante [80]:

$$JHash = 3 \times (|R| + |S|) \tag{1}$$

avec $|R|$ et $|S|$ sont égales respectivement à $\lceil \frac{||R|| \times LG^R}{PS} \rceil$ et $\lceil \frac{||S|| \times LG^S}{PS} \rceil$, tels que :

- *: LG^R et LG^S désignent respectivement la taille d'une instance de R et de S (obtenue à partir du dictionnaire de données de la couche conceptuelle) ;*
- *$||R||$ et $||S||$ représentent respectivement le nombre d'instances de R et S (la couche logique) ;*
- *PS décrit la taille d'une page disque (la couche de déploiement).*

Cet exemple nous montre que l'optimisation de requêtes est sensible à l'ensemble de phases du cycle de vie de conception de bases de données. Si le langage de requêtes, le type de la base de données, le modèle de stockage, ou la plateforme de déploiement changent alors le processus d'optimisation doit être revisité. L'ensemble des optimisations est souvent traité dans la phase physique du cycle de vie. Les bases de données autres que orientées objet ne prennent qu'un seul paramètre de la phase conceptuelle, à savoir le dictionnaire de données. Rappelons que le modèle conceptuel exprime à la fois les besoins applicatifs et la connaissance du domaine sous une forme intelligible pour un utilisateur ultérieur. Malheureusement, c'est le modèle logique qui est exploité; et celui-ci, résultant de la normalisation (dans le cas de bases de données relationnelles) et de l'adaptation au système support, est en général très différent du modèle conceptuel. Dans les bases de données orientées objets, les optimiseurs prennent en compte

les caractéristiques du modèle objet dans l'optimisation: les expressions de chemins, les groupages d'objets, les parcours de pointeurs, les index de chemins, et aussi les méthodes utilisateur [81]. La présence du modèle d'ontologie au sein d'un SGBD doit être prise en compte pour la conception physique. La Figure 2.2 illustre bien les phases de cycle de vie prises en compte par les deux types de bases de données: traditionnelles et sémantiques.

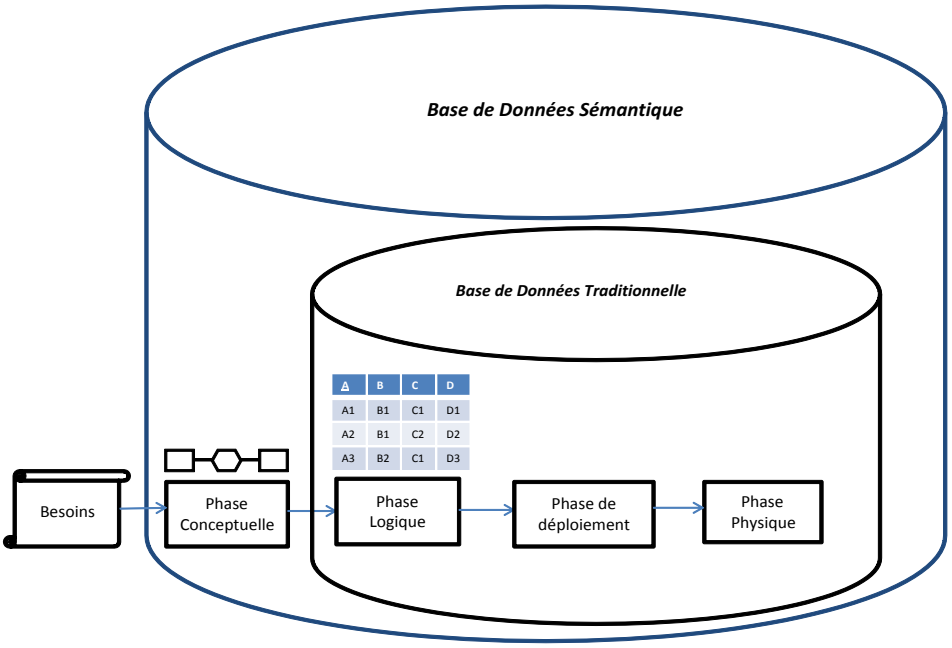


FIGURE 2.2 – Problème de conception physique dans les phases de cycle de vie

A partir de cette discussion, deux constats importants émergent: **(i)** la présence de l'ontologie dans une base de données sémantique peut être exploitée pour offrir des optimisations et **(ii)** tout changement au niveau des phases du cycle de vie entraine automatiquement une nouvelle visite de la conception physique.

Dans ce chapitre, nous détaillons le processus de la conception physique à la fois dans le contexte des bases de données traditionnelles et sémantiques. Nous y exposons la problématique, des exemples de structures d'optimisation ainsi que notre démarche de résolution dans le cas des $\mathcal{BDBO}$.

2 Conception physique

Un enjeu important pour la communauté de bases de données est la conception physique des bases de données avancées manipulant un gros volume de données tels que les entrepôts de données [82, 83]. Dans son papier *Self-Tuning Database Systems: A Decade of Progress* (prix de 10 Year Best Paper Award à la conférence prestigieuse Very Large Databases, édition 2007), Surajit Chaudhuri indique : "The first generation of relational execution engines were relatively

simple, targeted at OLTP, making index selection less of a problem. The importance of physical design was amplified as query optimizers became sophisticated to cope with complex decision support queries" [84]. Cette amplification est due aux caractéristiques suivantes liées aux bases de données avancées : (1) la complexité du schéma de la base de données, où l'ensemble des tables n'ont pas les mêmes caractéristiques. Prenons l'exemple d'un schéma d'un entrepôt de données relationnel, nous constatons deux types de tables: la table des faits normalisée contenant un nombre important d'instances, et des tables de dimension de taille moins importante, souvent dé-normalisées afin de minimiser le nombre de jointures nécessaires pour évaluer une requête, (2) le volume de données, (3) la complexité des requêtes qui nécessitent de plus en plus des opérations coûteuses comme la jointure et l'agrégation, (4) les exigences des décideurs sur le temps de réponse de requêtes et (5) la diversité des structures d'optimisation. Deux grandes classes de structures d'optimisation ont été distinguées [85]: des structures redondantes et structures non redondantes. Les structures redondantes sont celles qui nécessitent un espace mémoire pour leur stockage et ayant un coût de maintenance. Comme exemples de ces structures, on peut citer les index [86], les vues matérialisées [87], la réplication et la fragmentation verticale [88]. Tandis que les structures non redondantes ne demandent pas d'espace de stockage. Ce sont par exemple, la fragmentation horizontale [89] et le traitement parallèle sans réplication. Une autre difficulté associée à cette diversité est l'usage de ces structures. Certaines sont appliquées pendant la création de la base de données comme la fragmentation horizontale et d'autres pendant l'exploitation de la base de données comme les vues matérialisées. Cela rend le processus de sélection et d'utilisation de certaines structures sensibles. Dans la section suivante, nous décrivons en détails certaines des structures d'optimisation les plus utilisées par les SGBD commerciaux et académiques.

3 Quelques exemples de structures d'optimisation

Dans cette section, nous présentons deux exemples de structures d'optimisation redondantes, à savoir les index et les vues matérialisées et une structure d'optimisation non redondante, fragmentation horizontale.

3.1 Les index

Les techniques d'indexation ont été largement étudiées et constituent une option très importante pour la phase de conception physique des bases de données traditionnelles et avancées. Les index sont utilisés pour améliorer le temps d'accès aux données. La définition des index se fait souvent pendant l'exploitation de la base de données. Un nombre important d'index a été proposé et supporté par les SGBD commerciaux et académiques. Dans le contexte des bases de données traditionnelles, les index de type B-tree, les index de hachage, l'index de jointure, etc. ont été proposés. Certaines techniques d'indexation sont apparues dans le contexte d'entre-

pôts de données, vu la nature de requêtes OLAP (Online Analytical Processing). Ce sont par exemple les index binaires, les index de jointures binaires, les index de jointure en étoile, etc. Pour plus de détails sur les caractéristiques de ces index, nous recommandons la consultation de la thèse de Kamel Boukhalfa [85].

Sélectionner un ensemble d'index pour optimiser une charge de requêtes donnée est un problème difficile [90]. Pour répondre à ce problème, les premiers travaux ont proposé des solutions suivant l'approche $\mathcal{ODCP}$, où certaines règles comme l'usage des attributs candidats à l'indexation et la fréquence des requêtes ont été proposées [91]. Ces règles ne sont pas suffisantes pour offrir une sélection performante. Pour pallier à ce type de sélection, certains travaux ont formalisé le problème de sélection d'index (PSI) comme un problème d'optimisation suivant l'approche $\mathcal{ODTP}$.

Etant donnés :

- $Q = \{q_1, q_2, ..., q_n\}$, une charge de requêtes où chaque requête q_i possède une fréquence d'accès f_i, $(1 \leq i \leq n)$;
- un schéma de BD ou d'un entrepôt de données déployé sur une plateforme donnée.
- un espace S de stockage des index.

Le PSI consiste à trouver une meilleure configuration d'index, CI, permettant d'optimiser Q, c'est-à-dire réduire le coût total d'exécution des requêtes tout en respectant la contrainte de stockage ($Taille(CI) \leq S$).

Ce problème est NP-complet [90]. Plusieurs approches de résolution de ce problème ont été développées. Elles commencent par énumérer l'ensemble des attributs candidats à l'indexation. Ces attributs sont choisis parmi ceux présents dans les clauses WHERE, GROUP BY et ORDER BY des requêtes. Des algorithmes de sélection des attributs que l'on peut indexer ont été proposés ; ils sont guidés par un modèle de coût mathématique quantifiant la qualité de la solution obtenue. On peut citer des algorithmes génétiques, le recuit simulé, ou la programmation linéaire entière [92]. L'une de ces approches est utilisée dans le module *What-if* de SGBD SQL Server [93].

3.2 Les vues matérialisées

Une vue est une requête nommée. Elle est dite matérialisée si son résultat est stocké physiquement. Les vues améliorent l'exécution des requêtes en pré-calculant les opérations les plus coûteuses comme la jointure et l'agrégation, et en stockant leurs résultats dans la base. Dans cette situation, certaines requêtes nécessitent seulement l'accès aux vues matérialisées et par conséquent sont exécutées plus rapidement.

Les vues matérialisées peuvent être utilisées pour répondre à plusieurs objectifs, comme l'amélioration de la performance des requêtes ou la fourniture des données dupliquées (cache). Elles sont associées à trois problèmes majeurs, à savoir (i) le problème de leur sélection, (ii) le problème de leur maintenance et (iii) la réécriture des requêtes en fonction des vues.

3.2.1 La sélection des vues matérialisées

Etant donné que nous ne pouvons pas matérialiser toutes les vues pour des raisons de stockage, de maintenance et de réécriture, la sélection des vues matérialisées consiste à choisir un sous-ensemble de vues candidates permettant de réduire le coût d'exécution d'une charge de requêtes. La sélection des vues peut être effectuée sous certaines contraintes, généralement un quota d'espace et/ou un seuil de temps de maintenance à ne pas dépasser. Les principaux travaux sur la sélection des vues matérialisées ont été inspirés de l'approche $\mathcal{ODTP}$. Le problème de sélection des vues matérialisées (PSV) peut donc être formalisé comme suit [94, 95] :
Etant donnés :

- $Q = \{q_1, q_2, ..., q_n\}$ une charge de requêtes où chaque requête q_i possède une fréquence d'accès f_i, $(1 \leq i \leq n)$;
- un schéma de BD ou d'un entrepôt de données déployée sur une plateforme donnée;
- un ensemble de contraintes C.

Le problème de sélection de vues (PSV) consiste à trouver un ensemble de vues, VM, qui permet de réduire le coût total d'exécution des requêtes de Q et qui respecte les contraintes de C. Les contraintes de C peuvent être des contraintes de stockage, de maintenance, de temps d'exécution, etc.

Comme le problème de sélection d'index, le *PSV* est NP-complet [87]. Le PSV dans les entrepôts de données a été largement étudié tant pour l'approche MOLAP (multidimensionnal OLAP) que pour l'approche ROLAP (relational OLAP). Dans l'approche MOLAP, le cube de données est considéré comme la structure primordiale pour sélectionner les vues matérialisées. Chaque cellule du cube est considérée comme une vue potentielle. Dans l'approche ROLAP chaque requête est représentée par un arbre algébrique [96]. Chaque nœud (non feuille) est considéré comme une vue potentielle.

Deux types de PSV ont été considérés : le PSV statique et le PSV dynamique. Le PSV statique consiste à sélectionner un ensemble de vues à matérialiser afin de minimiser le coût total d'évaluation de ces requêtes, le coût de maintenance ou les deux, et ce, sous la contrainte de la ressource. Le problème suppose donc que l'ensemble des requêtes n'évolue pas. Si des évolutions sont enregistrées dans les requêtes alors il est nécessaire de reconsidérer totalement le problème (en reconstruisant les vues à matérialiser). Le PSV dynamique considère que l'ensemble des requêtes évolue dans le temps [97, 98, 99, 100]. Dans ce cas, l'algorithme dynamique peut être amené à modifier l'ensemble des vues matérialisées en fonction de nouvelles requêtes. En effet, il peut insérer de nouvelles vues et en supprimer d'autres dans le cas où l'espace maximum autorisé serait atteint. Pour combler les lacunes du PSV statique, Kotidis et al. [97] ont proposé un système appelé *DynaMat*, qui matérialise les vues d'une manière dynamique. DynaMat combine en fait les problèmes de la sélection et de la maintenance des vues.

Plusieurs travaux ont étudié le problème de sélection des vues matérialisées. Ces travaux peuvent être classés en deux grandes catégories [101] : (1) travaux dirigés par le temps d'interrogation et (2) travaux dirigés par le temps de maintenance. La première catégorie privilégie

l'optimisation des performances de l'interrogation de l'entrepôt et fonctionne sous contrainte d'espace de stockage des vues matérialisées. La seconde vise à réduire le temps de maintenance des vues matérialisées [94]. Elle est employée dans le cas où les rafraîchissements de l'entrepôt sont conséquents ou leur fréquence élevée. Notons que certaines approches ont été proposées dans le souci de satisfaire simultanément les deux besoins [102].

Comme les travaux effectués sur la sélection des index, ceux proposés pour la sélection des vues matérialisées utilisent souvent des heuristiques pour trouver une solution quasi-optimale. Les algorithmes proposés procèdent deux principales étapes : (1) génération des vues candidates et (2) sélection d'un sous-ensemble de ces vues. Dans la première étape, l'ensemble de vues matérialisées candidates V est construit à partir d'une charge de requêtes Q les plus fréquentes. Cet ensemble est structuré de manière à prendre en compte les relations susceptibles d'exister entre les vues candidates. Plusieurs structures ont été proposées pour représenter les relations entre les vues. Nous pouvons citer les structures suivantes : les treillis [103, 104, 105, 106, 107], les graphes [102, 108, 109, 110], les plans d'exécution des requêtes [111, 112, 113], etc.

Dans la deuxième étape, les algorithmes sélectionnent un sous-ensemble des vues candidates en fonction de la fonction *objectif* utilisée ainsi que des contraintes du problème de sélection. Plusieurs types d'algorithmes ont été utilisés pour effectuer cette sélection. Nous pouvons citer les algorithmes gloutons [103, 104, 114, 105, 106, 107, 112, 110], les méthodes issues de la recherche opérationnelle (sac à dos [113]) ou les algorithmes génétiques [115, 116].

3.2.2 La maintenance des vues matérialisées

Les tables de base changent et évoluent au rythme des mises à jour. Cependant, si ces changements ne sont pas reportés dans les vues matérialisées, leurs contenus deviendront obsolètes et leurs objets ne représenteront plus la réalité. La maintenance des vues matérialisées consiste à reporter les modifications survenues sur les tables de base au niveau des vues. Cela peut se faire selon trois approches : périodiques, immédiates et différée. Dans la première approche [117, 118], les vues sont mises à jour continuellement à des périodes précises ; dans ce cas ces vues peuvent être considérées comme des photographies (*snapshots*). Dans la seconde [119, 120], les vues sont mises à jour immédiatement à la fin de chaque transaction. Dans la dernière [121], les modifications sont propagées d'une manière différée. Dans ce cas, une vue est mise à jour uniquement au moment où elle est utilisée par une requête d'un utilisateur.

La maintenance des vues peut être effectuée en recalculant ces vues à partir des tables de base. Cependant, cette approche est complètement inefficace (très coûteuse). En effet, une bonne maintenance des vues est réalisée lorsque les changements (insertions, suppressions, modifications) effectués dans les tables sources peuvent être propagés aux vues sans être dans l'obligation de recalculer intégralement leur contenu. Pour résoudre ce problème, trois types de maintenance ont été proposées : incrémentale, autonome et en batch. La maintenance incrémentale consiste à identifier le nouvel ensemble de n-uplets à ajouter à la vue dans le cas d'une insertion

ou le sous-ensemble de n-uplets à retirer de la vue dans le cas d'une suppression, sans réévaluer intégralement la vue. La maintenance autonome assure que la maintenance d'une vue V peut être calculée uniquement à partir de V et des changements survenus sur les tables de bases sur lesquelles elle est définie. La maintenance en batch est effectuée en utilisant des transactions de mise à jour. Une transaction de maintenance est relativement longue et peut donc interrompre l'usage de l'entrepôt. Par conséquent, elle est exécutée souvent durant les périodes d'activité creuse (la nuit par exemple). Cette maintenance n'est pas souhaitable, car avec l'émergence d'internet, un entrepôt doit être opérationnel continuellement (24h/24).

3.2.3 La Réécriture de requêtes en fonction des vues matérialisées

Après le processus de sélection des vues, toutes les requêtes définies sur l'entrepôt doivent être réécrites en fonction des vues. Sélectionner la meilleure réécriture pour une requête est une tâche difficile [122, 123]. Le processus de réécriture de requêtes est supporté par la plupart des SGBD multidimensionnels (ex. Oracle). Le processus de réécriture des requêtes a attiré l'attention de nombreux chercheurs car elle est en relation avec plusieurs problèmes de gestion de données: l'optimisation de requêtes, l'intégration des données, la conception des entrepôts de données, etc. Le processus de réécriture des requêtes a été utilisé comme une technique d'optimisation pour réduire le coût d'évaluation d'une requête.

Plus formellement; ce processus peut se définir ainsi: soit une requête Q définie sur un schéma d'une base de données et un ensemble de vues $\{V_1, V_2, ..., V_i\}$ sur le même schéma. Est-il possible de répondre à la requête Q en utilisant seulement les vues? Alternativement, quel est le plan d'exécution le moins cher pour Q en supposant qu'en plus des tables de la base de données, on a aussi un ensemble de vues? La figure 2.3 donne une illustration de ce processus.

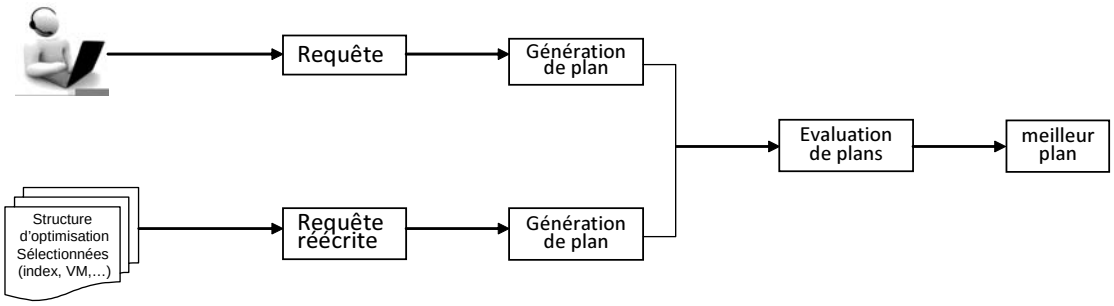


FIGURE 2.3 – Réécriture dans le processus d'optimisation

3.3 La Fragmentation horizontale

La fragmentation horizontale est une structure d'optimisation qui a été étudiée massivement dans les différentes générations de bases de données : les bases de données relationnelles, les

bases de données orientées objet, les bases de données XML, les entrepôts de données, etc. Initialement proposée comme une technique de conception logique des bases de données réparties dans les années 80 [124, 125], la fragmentation horizontale consiste à partitionner une table en fonction de ses n-uplets de façon à réduire le nombre des accès non nécessaires pour le traitement de certaines requêtes. Deux types de fragmentation horizontale existent : primaire [126] et dérivée [85]. La fragmentation horizontale primaire d'une table se base sur des attributs définis sur cette table. La fragmentation horizontale dérivée consiste à propager la fragmentation d'une table sur une autre table. Cette propagation n'est possible que si un lien père-fils existe entre les deux tables. Par conséquent, la fragmentation horizontale dérivée d'une table se base sur les attributs définis sur une ou plusieurs autres tables [85]. Contrairement aux autres structures d'optimisation (comme les index et les vues matérialisées), où la sélection des schémas d'optimisation se fait généralement lorsque la base de données est opérationnelle (ou créée), la sélection d'un schéma de fragmentation d'une base de données (ou un entrepôt de données) doit se décider avant sa création [127]. Cette situation rend sa sélection plus sensible que les autres structures. De plus, elle peut également être combinée avec d'autres structures d'optimisation comme les index [128], les vues matérialisées [89, 129] et le traitement parallèle [130].

Un nombre important de travaux sur la fragmentation horizontale a été développé dans le cadre des bases de données traditionnelles et avancées. Ces travaux ont suivi les deux approches que nous avons citées: $\mathcal{ODCP}$ et $\mathcal{ODTP}$. Les premiers travaux ont utilisé des critères comme les affinités entre les prédicats de sélection utilisés dans une charge de requêtes. Rappelons que la fragmentation horizontale est efficace si elle est définie sur les attributs figurant dans les prédicats de sélection. L'affinité entre deux prédicats représente la somme des fréquences d'accès des requêtes utilisant simultanément les deux prédicats. Cette approche est donc moins complexe que celle basée sur les prédicats dont la complexité est O(2n). Mais, elle ne prend en considération que la fréquence d'accès comme critère de regroupement. Or, pour fragmenter une base de données, d'autres paramètres doivent être pris en compte, comme les facteurs de sélectivité des prédicats, la taille de la table, la taille de page disque, le nombre de pages occupées par cette table, etc. Les approches basées sur les prédicats et sur les affinités ne fournissent aucune métrique permettant d'évaluer la qualité du schéma de fragmentation obtenu. Pour pallier ce problème, Bellatreche [131] a proposé dans le cadre de ses travaux de thèse une nouvelle approche de fragmentation basée sur un modèle de coût. Grâce à ce dernier, il est possible d'évaluer la qualité de la solution fournie.

Afin d'offrir aux concepteurs la possibilité de quantifier la qualité de leurs schémas de fragmentation, des algorithmes suivant l'approche $\mathcal{ODTP}$ ont été proposés dans le cadre des bases de données orientées objet et les entrepôts de données [124, 125, 132, 130, 128]. Les composantes principales de ces algorithmes sont: (i) un générateur de schémas de fragmentation, (ii) un modèle de coût et (iii) une sélection du schéma optimal.

1. Générateur de schémas de fragmentation : Cette étape consiste à générer tous les schémas de fragmentation possibles, d'une classe d'un schéma d'une base de données orientée

objet. Cette génération est guidée par l'ensemble de prédicats de sélection défini dans la charge de requêtes.

2. modèle de coût: Il constitue le noyau de ce type d'algorithmes. Il peut être vu comme une fonction d'une seule variable représentant un schéma de fragmentation S_i et attribuant le coût d'exécution $C(S_i)$ d'un ensemble de requêtes. Il prend en considération la taille des tables, le nombre de pages, les facteurs de sélectivité des prédicats, fan-out, etc. L'optimalité et la qualité des solutions obtenues par ces algorithmes sont liées à ce modèle de coût.

3. Sélection du schéma de fragmentation optimal: Il détermine le schéma de fragmentation S_{min} garantissant la meilleure performance parmi tous les schémas possibles.

4 Synthèse

D'après cette étude, nous concluons que les optimisations actuelles se basent sur l'ensemble de phases de cycle de vie de conception d'une base de données. En conséquence, le problème de la conception physique peut être présenté comme un problème d'optimisation. Etant donnés :

- Un schéma d'une base de données déployée sur une plateforme donnée;
- Une charge de requêtes $Q = \{Q_1, Q_2, ..., Q_m\}$, où chaque requête Q_j possède une fréquence d'accès f_j;
- Un ensemble de structures d'optimisation $SO = \{SO_1, SO_2, ..., SO_l\}$ supportées par le SGBD hôte utilisé pour répondre aux requêtes ;
- Un ensemble de contraintes liées à SO : $Cont = \{Cont_1, Cont_2, ..., Cont_l\}$, où chaque contrainte $Cont_i (1 \leq i \leq l)$ est associée à une structure d'optimisation SO_i.

Le problème de la conception physique consiste à sélectionner des schémas de structures d'optimisation afin de réduire le coût d'exécution de la charge de requêtes Q en présence de ces derniers tout en respectant les contraintes définies.

Pour résoudre ce problème, il faut considérer l'espace de recherche qui intègre tous les sous-espaces correspondants aux structures d'optimisation. Si Ins_i est le nombre d'instances de la structure d'optimisation SO_i, l'espace de recherche global est $2^{\sum_{i=1}^{t} Ins_i}$. Cela est dû au fait que les différentes instances peuvent interagir [133].

Dans la pratique, l'administrateur de base de données (DBA) doit effectuer les choix suivants: (a) choix de structures d'optimisation parmi SO supposé(es) pertinente(s) (b) choix du mode de sélection des schémas d'optimisation de ces dernières dans le cas où il décide d'utiliser plusieurs structures et (c) choix des algorithmes pour les sélections. Enfin le DBA passe à la validation des configurations recommandées par une mise en œuvre.

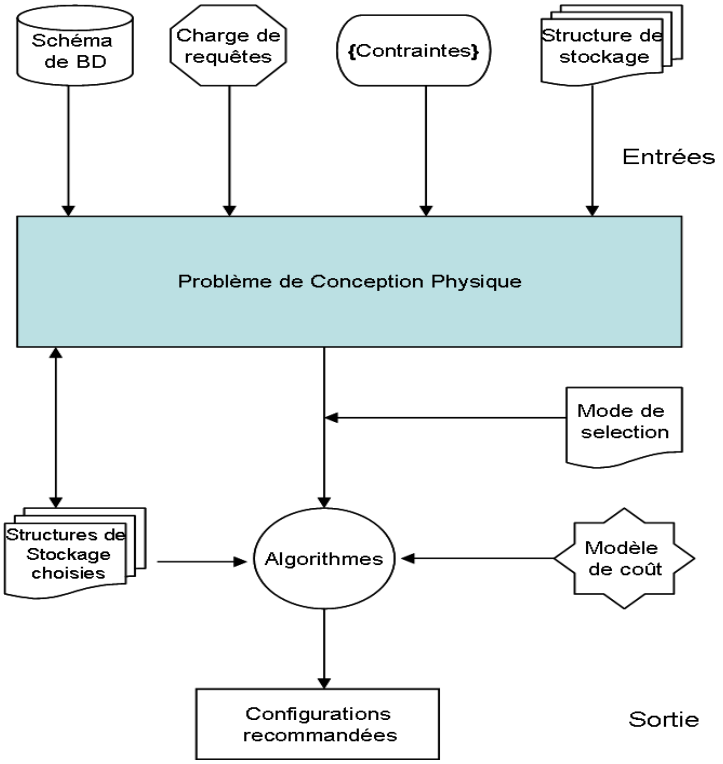


FIGURE 2.4 – Problème de conception physique

4.1 Choix structures d'optimisation

Nous avons vu qu'il existe plusieurs types de structures d'optimisation pour la conception physique. Le choix des structures pertinentes et adéquates pour optimiser une charge de requêtes est un problème complexe. Il nécessite une certaine expertise. Ce choix peut se faire manuellement en se basant sur l'expérience du DBA et les caractéristiques des requêtes [134]. Mais on peut aussi se servir des outils proposés par les SGBD commerciaux et non commerciaux comme DBDSGN [135], SQL Access Advisor [136], DB2 Design Advisor [133], et Parinda de PostgreSQL [137].

4.2 Choix de la méthode de sélection

Deux modes de sélection existent [85]: la sélection isolée [138] lorsque le DBA choisit une seule structure d'optimisation et la sélection multiple [130, 139] lorsque plusieurs structures sont choisies. Dans le cas d'une sélection multiple, plusieurs scénarii sont possibles pour combiner ces structures d'optimisation [140]. Le choix de l'ensemble des structures à utiliser et la manière de les combiner sont déterminants pour la performance du système [85].

4.3 Choix d'algorithme de sélection

Plusieurs classes d'algorithmes ont été proposées dans la littérature pour sélectionner des structures d'optimisation allant des algorithmes simples comme les algorithmes gloutons aux avancés comme les algorithmes génétiques [115, 116] et le recuit simulé [141].

4.4 Mise en œuvre des solutions d'optimisation

Une fois les structures d'optimisation recommandées par un algorithme de sélection, il est nécessaire de les valider par un déploiement dans environnement réel afin d'évaluer leur performance effective. On peut les valider à travers des outils commerciaux proposés comme Oracle SQL Access Advisor [136] et DB2 Design Advisor [133]. Cependant, ces outils présentent des limites liées aux structures d'optimisation et au mode de sélection fourni.

5 Vers une conception physique des $\mathcal{BDBO}$

La conception physique des $\mathcal{BDBO}$ pourrait être similaire à celle des bases de données traditionnelles. Mais en examinant les caractéristiques de ces dernières marquées par la diversité impliquant la présence de l'ontologie dans le SGBD, les modèle des stockage de données, l'architecture des $\mathcal{BDBO}$ et le langage de requêtes utilisées. Nous optons pour la revisiter pour que l'ensemble des caractéristiques soit inclus dans la formalisation.

Etant donnés:

- Une $\mathcal{BDBO}$ implémentée sur un SGBD donné avec ses spécificités concernant le stockage et l'architecture, et déployée sur une plateforme donnée;
- Une charge de requêtes en Sparql $Q = \{Q_1, Q_2, ..., Q_m\}$, où chaque requête Q_j possède une fréquence d'accès f_j;
- Un ensemble de structures d'optimisation $SO = \{SO_1, SO_2, ..., SO_l\}$ supportées par le SGBD hôte utilisé pour répondre aux requêtes ;
- Un ensemble de contraintes liées à SO : $Cont = \{Cont_1, Cont_2, ..., Cont_l\}$, où chaque contrainte $Cont_i (1 \leq i \leq l)$ est associée à une structure d'optimisation SO_i.

Le problème de la conception physique consiste à sélectionner des schémas de structures d'optimisation afin de réduire le coût d'exécution de la charge de requêtes Q en présence de ces derniers tout en respectant les contraintes définies.

La figure 2.5 décrit le problème de conception physique dans les bases de données à base ontologique.

En examinant les travaux existants, nous avons identifié que la conception physique dans le contexte des bases de données sémantique utilise la même démarche que celle des bases de données traditionnelles. Souvent, cette conception suppose un seul modèle de stockage et une

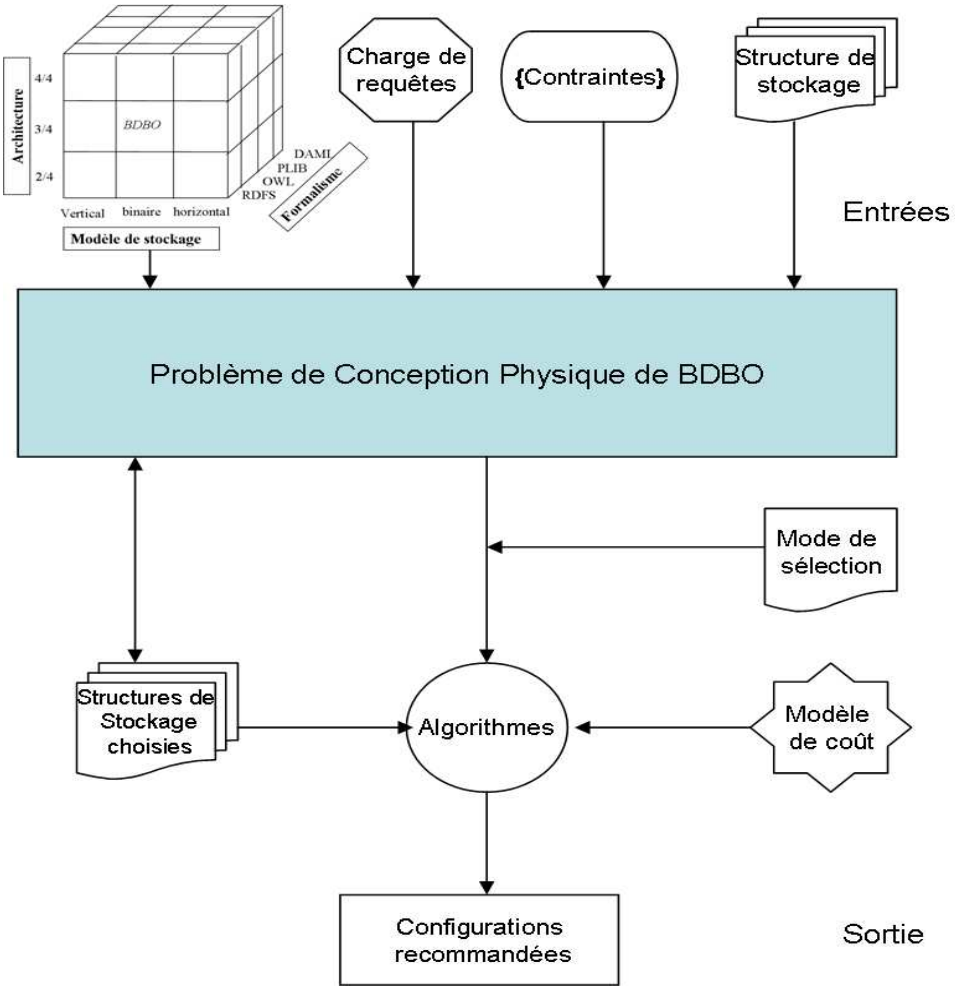


FIGURE 2.5 – Problème de conception physique dans les $\mathcal{BDBO}$

seule architecture comme dans le cas de [142] et [143]. Dans cette thèse, nous proposons une solution plus globale où l'ensemble de scénarii de stockage et d'architecturaux sont pris en compte.

Vu la difficulté du problème et la complexité des bases de données sémantiques, nous avons suivi la démarche suivante présentée sur la figure 2.6 :

1. Etude expérimentale des $\mathcal{BDBO}$. Cette étude nous a permis de comprendre les différents facteurs qui peuvent influencer la conception physique et surtout les caractéristiques de chaque type de base de données. Pour ce faire, nous avons utilisé six bases de données sémantiques, trois académiques (Jena, Sesame et OntoDB, développée au sein de notre laboratoire) et trois issues de milieu industriel, à savoir Oracle, DB2RDF et IBM SOR.
2. Développement d'un modèle de coût pour chaque type de $\mathcal{BDBO}$.
3. Considération d'une structure d'optimisation à savoir les vues matérialisées. Nous avons choisi cette structure pour trois raisons principales: (i) elle a été largement étudiée dans le cas des entrepôts de données, (ii) une vue peut être indexée et partitionnée, et (iii) elle a été étudiée récemment dans le cas des bases de données sémantiques.

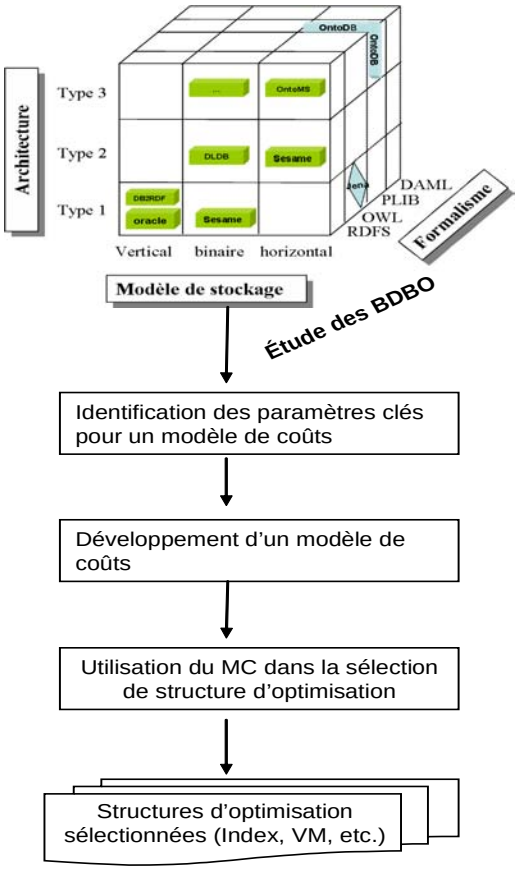


FIGURE 2.6 – Notre démarche. (MC:Modèle de coût)

6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la conception physique et avons mis en évidence les difficultés que l'on rencontre dans la conception physique des bases de données. Nous avons montré que la diversité de modèles de stockage des $\mathcal{BDBO}$, la variété de leurs architectures et la méconnaissance des tables sur lesquelles portent les requêtes amplifient davantage la complexité du problème de conception physique qui est déjà NP-complet dans les BD traditionnelles.

Ce constat nous motive à porter notre attention sur ce problème dans les bases de données à base ontologique.

Notre démarche est de proposer une formalisation unifiée des $\mathcal{BDBO}$ qui prend en compte leur diversité. Une fois cette formalisation établie, le développement des modèles de coût pour chaque spécificité devient un enjeu important. La qualité de ces derniers impacte celle de la conception physique. Pour instancier notre démarche de conception physique, nous prenons le cas des vues matérialisées. Ces points seront détaillés dans les chapitres suivants.

Deuxième partie

Contributions

Chapitre 3

Etude Empirique des Bases de Données à Base Ontologique

Sommaire

1 Introduction

Dans la conclusion du chapitre précédent, nous avons mentionné qu'étudier le problème de la conception physique dans le contexte des $\mathcal{BDBO}$ est une tâche difficile. Cela est dû à la forte diversité liée aux modèles de stockage et aux architectures du SGBD candidats pour stocker ce type de base. Une conception physique de bonne qualité nécessite la présence d'un bon modèle de coût. Ce dernier doit prendre en compte des paramètres influents de l'ensemble des phases du cycle de vie de conception des $\mathcal{BDBO}$ (Figure 2.1). En examinant la littérature, nous avons remarqué la présence d'un nombre limité de modèles de coût dans le contexte de base de données sémantiques. Pour identifier les paramètres de notre modèle de coût, nous avons suivi une approche expérimentale qui a permis d'évaluer trois types de $\mathcal{BDBO}$, chacune ayant un modèle de stockage et une architecture différente. Pour avoir une vue claire sur les $\mathcal{BDBO}$ utilisées, nous avons d'abord procédé à la généralisation un ensemble des $\mathcal{BDBO}$ en proposant un modèle unifié. Cette formalisation nous a permis d'identifier l'ensemble des composantes d'une $\mathcal{BDBO}$.

Dans ce chapitre, nous proposons d'abord une formalisation générique des $\mathcal{BDBO}$ puis une étude empirique de six exemples de $\mathcal{BDBO}$, à savoir OntoDB, Oracle, IBM SOR, DB2RDF, Sesame et Jena.

2 Les composantes d'une $\mathcal{BDBO}$

Afin de pouvoir traiter le problème de conception physique dans les $\mathcal{BDBO}$, nous proposons une formalisation des $\mathcal{BDBO}$ (modèle unifié des $\mathcal{BDBO}$), qui prend en compte la diversité des $\mathcal{BDBO}$ que nous avons identifiée au chapitre 1. Ce modèle unifié est basé sur les critères fondamentaux des $\mathcal{BDBO}$. Une $\mathcal{BDBO}$ stockant une ou plusieurs ontologies, nous présenterons d'abord une formalisation de cette notion. Mais avant tout, nous introduisons les notions sur les logiques de description qui sont utilisées dans nos formalisations.

2.1 Logiques de Description

Les logiques des descriptions (LD) sont utilisées pour représenter les connaissances et pour faire du raisonnement. Elles sont basées sur les notions de concepts et de rôles. Un concept correspond à un ensemble d'individus. Un concept peut être primitif (atomique) ou défini. Un rôle est une relation binaire entre deux individus. Les concepts et rôles peuvent être organisés en une hiérarchie grâce à la relation de subsomption. Une base de connaissances est composée d'une *TBOX* (Terminological Box) qui décrit les connaissances intentionnelles du domaine sous forme d'axiome et d'une *ABOX* (Assertion Box) qui décrit les connaissances extensionnelles du domaine (les instances). Les opérations de base du raisonnement terminologique sont la *classification* et l'*instanciation*. La *classification* s'applique aux concepts et rôles et permet de

déterminer la position d'un concept ou rôle dans leurs hiérarchies respectives. L'*instanciation* permet de retrouver les concepts dont un individu est susceptible d'être une instance.

Plusieurs familles de langages des logiques de description existent parmi lesquels $\mathcal{AL}$, $\mathcal{FL}$, $\mathcal{SHOQ}$[144], $\mathcal{SHIQ}$[145, 146], $\mathcal{SHOIN}$[147], etc. Le langage d'ontologie OWL est basé sur la famille ALC. OWL-DL est basé sur le fragment SHOIND(D) et OWL-Lite est basé sur le fragment SHIF(D). Chaque famille utilise un sous-ensemble de constructeurs. Dans ce qui suit, nous présenterons quelques constructeurs de LD et leurs correspondants OWL dont nous avons besoin pour nos formalisations (tableau 3.1) :

Syntaxe DL	OWL	**Définition**
A⊓B	intersectionOf	Conjonction de concepts
A⊔B	unionOf	Disjonction de concepts
¬ A	complementOf	Négation
$\{o_1, \dot{,} o_n\}$	oneOf	Enumération
$\forall\ r.C$	allValuesFrom	Quantificateur universel
$\exists\ r.C$	someValuesFrom	Quantificateur existentiel
$\exists P.o_1$	hasValue	Egalité de valeur
$\geq$ P.C_1	minCardinality	Cardinalité minimale
$\leq P.C_1$	maxCardinality	Cardinalité maximale
$= P.C_1$	cardinality	Cardinalité exacte
≡D	equivalentClass, sameAs	Equivalence entre concepts

TABLE 3.1 – Constructeurs de la description logique. *A, B* et *P* sont des noms de concepts et o_i et C_1 des noms d'instances

.

- *Intersection* (⊓, *owl:intersectionOf*) permet de construire une nouvelle classe comme une intersection de deux ou plusieurs classes. Par exemple, la classe *Mère* peut être définie comme l'intersection des classes *Parent* et *Femme* ($Mre = Parent \sqcap Femmes$).
- *Union* (⊔, *owl:UnionOf*) permet de déclarer une nouvelle classe comme étant le regroupement de deux ou plusieurs classes. Par exemple, on pourrait déclarer que la classe *Personne* est l'union des classes *Homme* et *Femme* ($Personne \equiv Homme \sqcup Femme$).
- *Complément* (¬, *owl:complementOf*) permet de définir une classe comme complémentaire d'une autre classe. Par exemple, on pourrait déclarer la classe *Homme* comme complémentaire de *Femme* dans la classe *Personne* ($Homme \equiv Personne - Femme$).
- *Énumération* (*owl:oneOf*) permet de définir une classe par extension : on liste les instances dans un ensemble. Par exemple *Doctorant = {Bery, Selma, ...,}*.
- *Restriction* (*owl:Restriction*) permet de déclarer une nouvelle classe à partir d'une autre en définissant des restrictions (contrainte de valeur et cardinalité) sur les instances de la classe. Il s'agit de :
  - *quantification Universelle* ($\forall\ p.C1$, *owl:allValuesFrom*) permet de déclarer une nou-

velle classe dont toutes les instances prennent leurs valeurs dans la classe C_1 pour la propriété p ;

- *quantification existentielle* ($\exists$ $p.C1$, *owl:someValuesFrom*) permet de déclarer une nouvelle classe dont les instances prennent au moins une valeur dans C_1 pour la propriété p ;
- *égalité de valeur* ($=$ $P.o_1$, *owl:hasValue*) permet de déclarer une nouvelle classe en imposant la valeur spécifiée (o_1) à toutes les instances de la classe en question. Par exemple, la classe *Homme* peut être déclarée comme la classe *Personne* avec la propriété *sexe* égale '*Masculin*' ($Homme \equiv Personne\ p \wedge p.sexe.Masculin$) ;
- *cardinalité minimale* ($\geq$ P.C_1, *owl:minCardinality*) permet de déclarer une nouvelle classe en fixant une borne inférieure au nombre d'instances dans une propriété ;
- *cardinalité maximale* ($\leq$ $P.C_1$, *owl:maxCardinality*) permet de déclarer une nouvelle classe en fixant une borne supérieure au nombre d'instances dans une propriété ;
- *cardinalité exacte* ($=$ $P.C_1$, *owl:cardinality*) permet de déclarer une nouvelle classe en indiquant exactement le nombre des instances dans une propriété.

2.2 Formalisation d'une ontologie

La formalisation d'une ontologie que nous proposons consiste en le 5-uplet suivant : *MO= <C, P, Applic, Ref, Formalisme>*.

- C représente les classes de l'ontologie. L'ensemble des classes peut être décomposé en deux sous-ensembles C_c et C_{nc} ($C = C_c \cup C_{nc}$) ; où C_c représente l'ensemble des classes canoniques, et C_{nc} l'ensemble des classes définies ou non-canoniques. Par exemple, nous pouvons imaginer une ontologie où $C_c = \{Homme, Femme\}$ et $C_{nc} = \{Personne\}$ car $Personne = \{Homme \cup Femme\}$. Comme autre exemple, si *Personne* est une classe canonique associée à la propriété *genre*, $C_c = \{Personne\}$ et $C_{nc} = \{Homme, Femme\}$ car $Homme \equiv Personne(genre = masculin)$ et $Femme \equiv Personne(genre = fminin)$.
- P représente les propriétés de l'ontologie. C'est à dire les propriétés utilisées pour décrire les instances de l'ensemble des classes C par des valeurs appartenant soit à des types simples, soit à d'autres classes. Comme pour l'ensemble des classes, l'ensemble des propriétés peut être aussi décomposé en deux sous-ensembles P_c les propriétés canoniques et P_{nc} les propriétés non canoniques. Par exemple si *nom* et *prénom* sont des propriétés de l'ontologie, $P_c = \{nom, prénom\}$ et $P_{nc} = \{nomComplet\}$ avec $nomComplet = nom + prénom$.
- $Applic : C \rightarrow 2^P$ est une fonction qui permet de lier chaque classe aux propriétés qui lui sont attachées, c'est-à-dire, celles qui peuvent être utilisées pour en décrire ses instances. Seules les propriétés ayant pour domaine une classe, ou l'une de ses super-classes, sont applicables à une classe. Si le domaine d'une propriété n'est pas spécifié, la racine de l'ontologie est implicitement considérée.
- $Ref : C \rightarrow (operateur,\ Exp(C))$ est une fonction qui associe à chaque classe un *opé-*

rateur d'inclusion ou d'équivalence et une expression sur des classes ou des propriétés. Les expressions utilisées pour définir des ontologies OWL qui sont basées sur les logiques de description présentent, à notre point de vue, un ensemble d'opérateurs complet couvrant plusieurs formalismes ontologiques. Ces expressions utilisent les opérateurs suivants : opérateurs ensemblistes (*intersectionOf* ($\cap$), *unionOf* ($\cup$), *complementOf* ($\neg$)), restrictions de propriétés (*AllValuesFrom* $\forall p.C$, *SomeValuesFrom* $\exists p.C$, *HasValue* $\ni p.C$) et les opérateurs de cardinalité ($\geq nR.C$, $\leq nR.C$). Les opérateurs d'inclusion ($\subseteq$) et d'équivalence jouent des rôles très importants : l'opérateur d'inclusion permet de construire des hiérarchies de classes et l'opérateur d'égalité intervient dans la définition des concepts non canoniques. *Exp* est une expression sur les classes $c \in C$ et les propriétés $p \in P$ utilisant aussi les opérateurs de la logique de description et les opérateurs ensemblistes. Par exemple, *Ref(Homme)=(*$\equiv$*, Exp(Personne))* où $Exp(Personne) = Personne.sexe.Masculin$. Notons que *Exp* peut être la fonction d'identité qui associe à une classe la même classe. Comme exemple, Ref(*Homme*)=($\sqsubseteq$, Exp(*Personne*)) avec $Exp(Personne) = Personne$.

– *Formalisme* est comme son nom l'indique le formalisme du modèle ontologique adopté. Cela peut être RDFS, OWL, PLIB, OIL, DAML, etc.

Vu le fait qu'une ontologie peut être exprimée dans différents modèles d'ontologies, il est possible de faire une spécialisation de cette formalisation pour un modèle d'ontologies donné. L'exemple suivant présente une formalisation des ontologies du modèle d'ontologies PLIB.

Exemple 1

OntologiePLIB = <C, P, Applic, Ref(C), PLIB> où Ref(c) =(OntoSub, Exp(c)), avec $c \in C$*. L'opérateur OntoSub de Plib est un opérateur permettant de définir un héritage partiel, où une classe référence une autre classe en héritant de tout ou d'une partie de ses propriétés. C'est une relation de subsomption*[13]*.*

2.3 Modèle unifié des $\mathcal{BDBO}$

Pour représenter la diversité des structures de $\mathcal{BDBO}$, nous avons trouvé nécessaire de proposer une structure générique permettant de faire ressortir le modèle ontologique, le schéma de stockage du schéma de l'ontologie, le schéma de stockage des instances ontologiques et l'architecture de $\mathcal{BDBO}$. Notre modèle unifié est le 7-uplet défini de la manière suivante :
$BDBO :< MO, I, Sch, Pop, SM_{MO}, SM_{Inst}, Ar >$, où

– *MO* représente les ontologies stockées dans la $\mathcal{BDBO}$ formalisée sous la forme d'un 5-uplet *<C, P, Applic, Ref, Formalisme>* comme défini dans la section précédente ;
– I : représente l'ensemble des instances ontologiques ;
– $Sch : C \rightarrow 2^P$ est une fonction qui associe à chaque classe l'ensemble des propriétés pour lesquelles les instances de cette classe ont des valeurs. On a la contrainte $\forall c \in C,\ Sch(c) \subseteq Applic(c)$;

- $Pop : E \rightarrow 2^I$, est une fonction qui associe à chaque classe ses instances de l'ensemble I ;
- *Modèle stockage (SM_{MO})*: le modèle de stockage de l'ontologie (vertical, binaire et horizontal) ;
- *Modèle stockage (SM_{Inst})*: le modèle de stockage des instances ontologiques ;
- *Modèle d'architecture (Ar)*: le type d'architecture de la base de données (Type I, II ou III).

Ce modèle sera illustré sur chaque $\mathcal{BDBO}$ présentée dans la suite de ce chapitre (§ 3.1).

Le modèle présenté dans cette section permet de représenter la diversité des $\mathcal{BDBO}$ au niveau macroscopique en termes de formalisme d'ontologie, de modèles de stockage et d'architecture. Dans la section suivante, nous chercherons à comparer plus précisément les $\mathcal{BDBO}$. Pour cela, nous allons nous intéresser à quelques $\mathcal{BDBO}$ particulières.

3 Etude des $\mathcal{BDBO}$

La conception physique des $\mathcal{BDBO}$ nécessite une connaissance très précise des $\mathcal{BDBO}$, nous avons choisi de comparer des $\mathcal{BDBO}$ fortement utilisées et venant de différentes communautés. Ainsi, nous nous intéressons à six $\mathcal{BDBO}$ dont trois sont issues du monde de la recherche (OntoDB, Sesame et Jena) et trois autres sont issues du monde industriel (Oracle, DB2RDF et IBM SOR d'IBM). Nous nous appliquerons à mettre en avant les caractéristiques identifiées au chapitre 1 qui montre la diversité des $\mathcal{BDBO}$.

3.1 Présentation des $\mathcal{BDBO}$

3.1.1 Base de Données à Base Ontologique OntoDB

OntoDB [9] est une architecture de $\mathcal{BDBO}$ conçue par le Laboratoire d'Informatique et d'Automatique pour les Systèmes (LIAS). Elle est conçue pour supporter l'évolution des ontologies et pour offrir un accès aux données au niveau ontologique et au niveau données. Une première monture est implantée sur le SGBD/RO PostGreSQL. Elle offre une nouvelle façon de concevoir des applications de bases de données en stockant dans la base de données explicitement les données, le modèle conceptuel qui définit la structure des données et l'ontologie qui définit le sens de ces données. Il est doté du langage OntoQL qui a été présenté au chapitre 1. Celui-ci permet à la fois d'interroger les ontologies, les données mais aussi les deux simultanément. Il prévoit également la possibilité d'utiliser le langage SPARQL. OntoDB se fonde sur les hypothèses de typage fort des propriétés [14] et des instances [15], la complétude de définition et la mono-instanciation. La $\mathcal{BDBO}$ OntoDB a été largement utilisée dans des projets industriels et

14. Toute propriété doit avoir un domaine et un co-domaine uniques;
15. toute instance du domaine ne peut être décrite que par des propriétés applicables à sa classe de base.

ANR (Projet ANR DaFOE4App: Differential And Formal Ontologies Editor For Applications et ANR E-wok hub: http://www-sop.inria.fr/edelweiss/projects/ewok/, etc.). Cette utilisation a été souvent basée sur les données techniques qui sont représentées par le formalisme PLIB.

3.1.1.1 Formalisme d'ontologies. Les ontologies gérées jusque-là sous OntoDB, sont conformes au modèle d'ontologies PLIB. Cependant, OntoDB est prévue pour supporter n'importe quel type d'ontologie. En effet, l'architecture d'OntoDB dispose d'un méta-schéma susceptible de prendre en compte tout schéma d'ontologie. Cela offre une structure de données pour manipuler aussi les ontologies RDFS, OWL et autres formalismes d'ontologies.

3.1.1.2 Architecture et Modèle de stockage. OntoDB utilise une architecture "*quatre quarts*" illustrée sur la figure 3.1. Elle se compose des quatre parties :

- partie *ontologie* qui stocke les ontologies c'est-à-dire les concepts permettant de représenter la signification des données (partie *donnée*);
- partie *méta-schéma* qui permet de représenter le modèle d'ontologies et le méta-schéma lui-même. Elle permet de rendre générique le traitement sur les ontologies ;
- partie *données* qui contient les instances c'est-à-dire les données proprement dites ;
- partie *méta-base*, appelée également "catalogue système", est la partie classique des SGBD qui contient la description des objets (tables, vues, index,...) de la base de données.

OntoDB utilise un modèle de stockage horizontal pour représenter les instances : une table est créée pour chaque classe de l'ontologie, ses colonnes correspondent au sous-ensemble de propriétés applicables de la classe, autrement celles qui sont utilisées par au moins une instance de la classe.

3.1.1.3 Formalisation. Une $\mathcal{BDBO}$ OntoDB dans notre modèle unifié est exprimée par : $BDBO_{OntoDB}$ *:< MO :<C, P, Applic, Ref(C), PLIB>, I, Sch, Pop :classe –>table, Spécifique, Horizontal, Type III>*. où *<C, P, Applic, Ref(C), PLIB>* représente une ontologie PLIB (cf; exemple 1), *Pop* associe à chaque classe la table correspondante.

3.1.2 Base de Données à Base Ontologique Oracle

La compagnie Oracle a récemment ajouté à son SGBD le support des langages RDF et OWL pour permettre à ses clients de bénéficier d'une plate-forme de gestion de données sémantiques. Cette fonctionnalité a été implantée pour la première fois dans le SGBD Spatial Oracle 10g et est désormais en option dans les bases de données Oracle sous le nom de *Oracle Spatial and Graph*.

Oracle propose les fonctionnalités suivantes pour les $\mathcal{BDBO}$:

- le stockage, le chargement et l'accès aux données et aux ontologies à travers le langage de manipulation de données ;

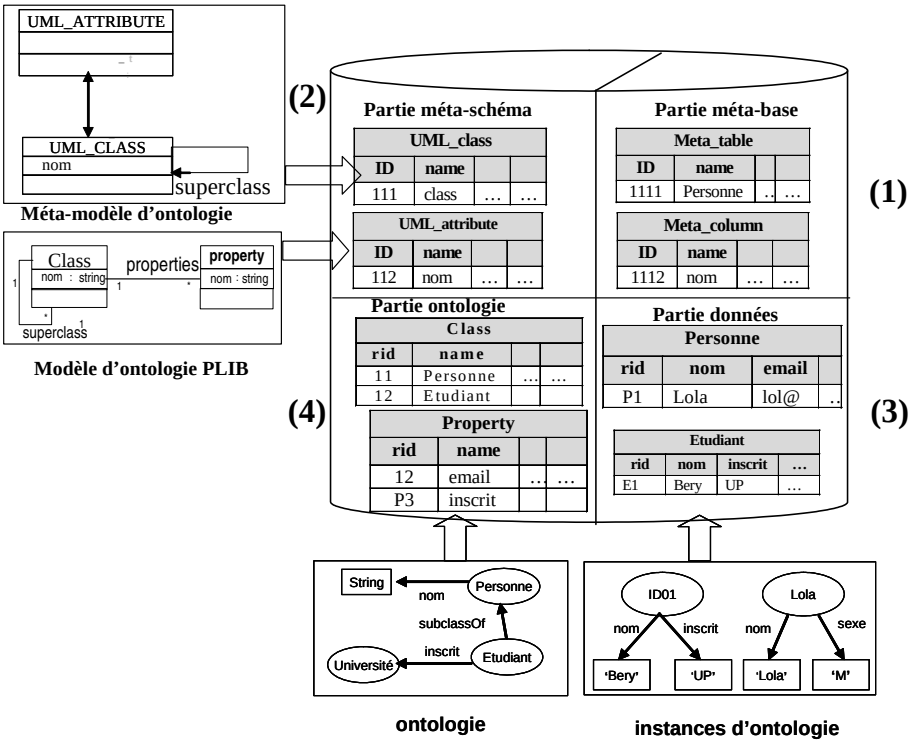


FIGURE 3.1 – Architecture OntoDB

- la déduction de connaissances en utilisant les règles sémantiques RDF/OWL et celles définies par l'utilisateur ;
- l'interrogation de données RDF/OWL et des ontologies en utilisant les patrons de graphe similaires à SPARQL inclus dans SQL (SEM_MATCH) ;
- l'interrogation de données relationnelles assistée par l'ontologie.

Oracle en collaboration avec les développeurs de Jena, ont mis en place une API appelée *Oracle Jena Adaptor*. Cette API implante et étend l'API Jena [16] qui est bien connue des développeurs des applications utilisant les ontologies. Les interfaces *Graph* et *model* de Jena ont été particulièrement enrichies. *Oracle Jena adaptor* étend donc les capacités de gestion de données sémantiques d'Oracle (Oracle 10gR2 RDF et Oracle 11g RDF/OWL) avec un ensemble de méthodes Java faciles à utiliser pour la plupart des fonctionnalités offertes. Un travail semblable fait avec les développeurs de Sesame a donné lieu à une API appelée *Oracle Sesame Adaptator*.

3.1.2.1 Formalisme d'ontologies. La $\mathcal{BDBO}$ Oracle prend en compte les ontologies RDF et OWL présentées sous le format N-Triples. Pour OWL, Oracle a défini deux sous-ensembles d'OWL-DL qu'elle traite dans les processus d'inférence [148] : OWLPrime et OWLSIF.

16. A Semantic Web Framework for Java - `http://jena.sourceforge.net`

3.1.2.2 Architecture et Modèle de stockage. Oracle utilise une architecture de type I ("*deux quarts*") et un modèle de stockage vertical (table de triplets). La table de triplets utilisée a été décomposée pour éviter de manipuler les longues chaînes de caractères (les URIs) [148]. Les valeurs lexicales des *sujets*, *prédicats* et *objets* sont mappées en identificateur (*ID*) entiers générés par le système. Une table de correspondance appelée *RDF_Value* fait le lien entre les URI, les littéraux et les identifiants générés. Etant donné un triplet RDF, ses trois composantes (URI ou littéraux) sont ainsi mappées aux identificateurs correspondants dans la table *RDF_Value*. Si aucune correspondance n'est trouvée pour un URI, un identifiant (*ValueId*) est généré et la correspondance est insérée dans la table *RDF_Value*. Un tuple comprenant l'identifiant du modèle (*Model_Id*) et les trois identifiants des URI est stocké dans la table des triplets appelée *RDF_LINK$* . La table *RDF_LINK$* permet de stocker tous les triplets de tous les modèles sémantiques de la base de données. La table *RDF_LINK$* est fragmentée (horizontalement) sur l'attribut *MODEL_ID* qui a une valeur unique pour chaque modèle (sémantique ou inféré). Ainsi, chaque partition correspond à un modèle sémantique ou à un modèle de données inféré.

3.1.2.3 Formalisation. Dans notre formalisation, une $\mathcal{BDBO}$ Oracle est représentée par:
$BDBO_{Oracle}$ *:< MO :<C, P, Ref(C), (RDFS, OWLSIF ou OWLPrime)>, I, Sch, Pop, Vertical, Vertical, Type I>*,
où *Ref(C)* : Application d'opérateurs des langages RDFS, OWLSIF et OWLPrime sur les classes et propriétés; *I* : les instances (tables *RDF_link* et *RDF_values*); *Applic* retourne pour chaque classe *c*, retourne toutes ses propriétés.

3.1.3 Base de Données à Base Ontologique SOR d'IBM

La société IBM s'est investie dans plusieurs travaux sur les technologies d'intégration et de gestion des données sémantiques et surtout celles relatives au Web Sémantique. Elle s'est impliquée dans le développement d'outils d'exploitation des ontologies qui sont un maillon important du Web Sémantique [149]. Parmi les outils basés sur les ontologies mis au point par IBM, nous trouvons *Integrated Ontology Development Toolkit (IODT)*: un outil dédié au stockage, à la manipulation, à l'interrogation et à l'inférence des ontologies et leurs instances. Dans le cadre de notre travail, nous nous sommes intéressés à SOR [59] qui est un référentiel des ontologies OWL. SOR est incorporé dans IODT. Il comprend un raisonneur et un outil de recherche. Il utilise pour le stockage une base de données relationnelle notamment DB2.

3.1.3.1 Formalisme d'ontologies. Les outils de SOR sont destinés à manipuler les ontologies du Web Sémantique notamment le formalisme OWL. Cependant, l'implantation actuelle manipule les ontologies OWL-Lite et OWL-DL.

3.1.3.2 Architecture et Modèle de stockage. SOR utilise une architecture type II ("*trois quarts*") pour le stockage des ontologies et de leurs instances. Il sépare les ontologies des instances et prévoit un mécanisme de raisonnement dans son modèle de stockage. Ainsi, il met en œuvre un modèle de stockage spécifique. En effet, SOR propose un schéma qui prend en compte des constructeurs complexes. Dans ce schéma, tous les prédicats des règles d'inférence *ABox* [17] ont des tables correspondantes dans la base de données. Ainsi, ces règles sont facilement traduisibles en une séquence d'opérations de l'algèbre relationnelle. Quatre catégories de tables sont définies dans le schéma de SOR : tables atomiques, tables des axiomes *TBox*, tables des faits *ABox* et tables des constructeurs de classes.

Les tables atomiques. Les tables atomiques sont les tables : *Ontology, PrimitiveClass, Property, Datatype, Individual, Literal et Resource*. Ces tables encodent l'URI avec un identifiant entier qui réduit la longueur des URI. Elles disposent d'une colonne *hashcode* qui est utilisée pour accélérer la recherche sur les URI et d'une colonne *ontologyID* qui indique de quelle ontologie une URL provient. La table *Property* stocke les caractéristiques des propriétés (symétrique, transitive, etc.). Pour profiter des opérations de comparaison internes aux bases de données, les littéraux booléens, dates et numériques sont représentés séparément en utilisant les types de données correspondants fournis par les bases de données.

Les tables des faits ABox. Il y a trois sortes d'assertions ABox développés dans le raisonnement : les assertions *TypeOf, Datatypeproperty* et *Objectproperty*. Les triplets de ces assertions sont stockés dans trois tables différentes : *TypeOf, RelationshipInd et RelationshipLit*. Une vue nommée *Relationship* est construite comme un point d'entrée aux triplets *objectproperty* et aux triplets *datatypeproperty*. Les triplets non pertinents au raisonnement tels que ceux dont le prédicat est *RDFS:comment*, sont stockés dans la table *Utility*.

Les tables d'axiomes TBox. Ce sont des tables utilisées pour garder les axiomes *TBox*. Il s'agit des tables *SubClassOf, SubPropertyOf, Domain, Range, DisjointClass, InversePropertyOf*.

Les tables des constructeurs de classes. Les tables des constructeurs de classes sont utilisées pour stocker les expressions de classe. SOR décompose une description complexe de classe en instances de concepts de classe OWL, assigne un nouvel *ID* à chaque instance et la stocke dans la table de la classe correspondante.

17. Une base de connaissances en Logique de Description est composée de *TBOX* (Terminological Box) décrivant les connaissances intentionnelles du domaine sous forme d'axiomes, et de ABOX (Assertion Box) décrivant les connaissances extensionnelles du domaine (les instances).

Une telle conception est motivée par l'explicitation de la sémantique de description des classes complexes. De cette façon, tous les nœuds dans un arbre de subsomption OWL sont matérialisés dans les tables de la base de données et les règles d'inférence peuvent ainsi être facilement implantées et rapidement exécutées via les instructions SQL. Le module de stockage est destiné à conserver les triplets originaux et les assertions inférées par le raisonneur DL et le moteur d'inférence. Mais les assertions inférées sont identifiées grâce à un flag spécifique. Ce schéma a l'avantage que certaines jointures se font entre les tables de petites dimensions contrairement à d'autres représentations dont les jointures se font sur la table centrale. En plus, il est indépendant de l'ontologie utilisée.

3.1.3.3 Formalisation. La représentation des $\mathcal{BDBO}$ SOR dans notre modèle unifié est : $BDBO_{SORIBM} :< MO :< C, P, Applic, Ref, OWL >, I, Sch(c), Spécifique, Binaire, Type\ II >$. Sch(c) renvoie les propriétés qui ont de valeurs de la classe *c*; où *Ref* traduit une application d'opérateurs de la logique de description (LD) sur les classes et propriétés; *I* : les instances des tables.

3.1.4 Base de Données à Base Ontologique DB2RDF d'IBM

Parmi les travaux récents de IBM sur l'exploitation de la sémantique des données, on trouve DB2RDF [150, 151] qui est un système de gestion de bases de données sémantiques (Base de données RDF). IBM l'a intégré dans la nouvelle version de son SGBD (DB2 10.1) mais on peut l'utiliser aussi sur DB2 9.6. IBM fournit un certain nombre de fonctions pour la gestion des données ontologiques dans un environnement Java (basé sur les paquetages de jena). DB2RDF supporte le langage SPARQL et utilise comme format de données le N-Triple [152] comme oracle. DB2RDF ne fait pas encore des déductions de données.

3.1.4.1 Formalisme d'ontologies. La $\mathcal{BDBO}$ DB2RDF utilise dans sa version actuelle le formalisme RDF/RDFS. Pour pouvoir être chargées, les instances ontologiques doivent être exprimées au format N-Triple.

3.1.4.2 Architecture et Modèle de stockage. la $\mathcal{BDBO}$ DB2RDF utilise une architecture de type *deux quarts*. Elle utilise un modèle de stockage vertical pour stocker ses données (schéma et instances) mais ce modèle est enrichi par l'utilisation de plusieurs tables [153] dont les quatre principales sont :

- la table *Direct primary* qui stocke les triplets avec un index sur le sujet. Les prédicats et les objets propres à un sujet sont stockés dans des colonnes paires de cette table.
- La table *Direct Secondary* est utilisée pour stocker les objets RDF qui partagent les mêmes sujets et prédicats.

– La table *Reverse primary* qui stocke les triplets indexés par leur objets. Les prédicats et les sujets d'un objet sont stockés dans des colonnes paires de cette table.
– La table *Reverse Secondary* qui stocke les sujets des déclarations RDF qui partagent les mêmes objets et prédicats.

Comme oracle, DB2RDF utilise des identifiants pour éviter de manipuler des longues chaînes d'URI.

3.1.4.3 Formalisation. Une $\mathcal{BDBO}$ DB2RDF dans notre modèle unifié est définie par :
$BDBO_{DB2RDF}$ *:< MO :<C, P, Applic, Ref, RDFS>, I, Sch, Pop, Vertical, Vertical, Type I>,*
où *Applic* retourne pour chaque classe *c*, toutes ses propriétés.

3.1.5 Base de Données à Base Ontologique Sesame

La $\mathcal{BDBO}$ sésame est un framework de gestion des données sémantiques développé par Aduna, un éditeur néerlandais de logiciels. Sésame permet le stockage et l'interrogation des ontologies et de leurs instances. C'est un outil facile à utiliser et fournissant une API permettant une connexion à tout système de stockage de données sémantiques. Elle offre la possibilité de faire des inférences de données. Sesame est flexible en ce sens qu'il peut être déployé sur des systèmes de stockage variés : mémoire centrale, systèmes de fichiers, bases de données relationnelles, etc. Elle dispose de son propre langage d'interrogation (SeRQL) mais supporte aussi le langage de requêtes SPARQL (le standard du W3C [18]). Sesame offre un accès transparent à des référentiels RDF distants en utilisant exactement la même API que pour l'accès local.

3.1.5.1 Formalisme d'ontologies. La $\mathcal{BDBO}$ Sésame a pour vocation de gérer les données du Web Sémantique. Elle supporte les modèles d'ontologies RDF/RDFS et OWL.

3.1.5.2 Architecture et Modèle de stockage. Sésame utilise une architecture de type I comme dans une base de données relationnelles. Pour le modèle de stockage, elle utilise la représentation verticale et la représentation binaire. Nous avons considéré les deux représentations dans les évaluations des performances (cf. section 4).

3.1.5.3 Formalisation. Une $\mathcal{BDBO}$ Sesame dans notre modèle unifié est définie par :
$BDBO_{Sesame}$ *:< MO :<C, P, Applic, Ref, RDFS ou OWL>, I, Sch, Pop, Vertical ou Spécifique, Vertical ou Binaire, Type I>,*
où *Applic* et *Pop* sont identiques à celles de la $\mathcal{BDBO}$ Oracle. *I* représente les instances de la table des triplets ou des tables de propriétés suivant la représentation.

18. World Wide Web Consortium, un organisme de normalisation chargé de promouvoir la compatibilité des technologies du World Wide Web

3.1.6 Base de Données à Base Ontologique Jena

Jena est une architecture pour stocker et gérer les données ontologiques. C'est aussi un framework open-source pour développer des applications pour le Web sémantique. Elle fournit un environnement de programmation pour les modèles d'ontologies RDF Schema et OWL. Jena utilise comme langage de requête SPARQL. Il propose de moteurs d'inférence à base de règles (prédéfinies et utilisateur).

3.1.6.1 Formalisme d'ontologies. Comme Sesame la $\mathcal{BDBO}$ Jena est orientée vers la gestion des données du Web Sémantique. Elle prend en compte les modèles d'ontologies RDF/RDFS et OWL.

3.1.6.2 Architecture et Modèle de stockage. Jena propose plusieurs façons de rendre persistantes les données que l'on regroupe sous deux vocables :

1. *TDB* (Tuple DataBase), un stockage fondé sur le système de gestion des fichiers ou la mémoire,
2. *SDB* (SPARQL DataBase), un système de stockage utilisant une base de données standard.

C'est ce dernier qui fait l'objet de notre travail car nous parlons des bases de données à base ontologique. Il existe deux versions de Jena : la première version appelée *jena1* utilise la représentation verticale (table des triples) [7] et la deuxième version *jena2* utilise une représentation spécifique (property-table et class-property-table) [6]. Dans les deux cas, l'architecture est de type "*deux quarts*". Nous utiliserons *jena2* dans notre travail.

3.1.6.3 Formalisation. Une $\mathcal{BDBO}$ Jena dans notre modèle unifié est définie par :
$BDBO_{Jena}$:< *MO* :<*C, P, Applic, Ref, RDFS*>, *I, Sch, Pop, Vertical, Vertical, Type I*>, pour le cas où la représentation verticale est utilisée et par
$BDBO_{Jena}$:< *MO* :<*C, P, Applic, Ref, RDFS*>, *I, Sch, Pop, Vertical, Binaire, Type I*> lorsque l'on utilise le modèle binaire. *I* représente les instances de la table des triplets ou des tables de propriétés suivant la représentation.

3.2 Critères de comparaison des $\mathcal{BDBO}$

Pour détailler notre comparaison des $\mathcal{BDBO}$, nous avons identifié quelques critères de comparaison essentiels des $\mathcal{BDBO}$. Le tableau 3.2 récapitule ces critères pour les $\mathcal{BDBO}$ considérées (Oracle, IBM SOR, DB2RDF, OntoDB, Jena et Sesame).

Nous présentons brièvement chacune de ces caractéristiques hormis, les modèles de stockage, l'architecture et le formalisme qui ont été déjà présentés et le SGBD qui sont les supports standards de persistance.

3.2.1 Échange de données

Cette caractéristique indique si la $\mathcal{BDBO}$ facilite le transfert de ses données sur une autre $\mathcal{BDBO}$. En effet, une $\mathcal{BDBO}$ est basée sur une ontologie qui est une conceptualisation consensuelle d'un domaine, dont chaque élément est référençable. Les données des $\mathcal{BDBO}$ utilisant la même ontologie peuvent être échangées car cette ontologie peut être considérée comme un format d'échange de données sur ce domaine. Cette fonctionnalité est importante quand on est dans un environnement nécessitant de l'intégration des données.

3.2.2 Support linguistique

Cette caractéristique indique si une $\mathcal{BDBO}$ est capable de prendre en compte les concepts d'une ontologie définis dans plusieurs langues naturelles. En effet, comme nous l'avons vu au chapitre 1, dans certains modèles d'ontologies (RDFS, OWL, PLIB) le multilinguisme est utilisé pour décrire les ontologies et les données.

3.2.3 Langage de requêtes

A fin de permettre l'interrogation de ses ontologies et ses instances ontologiques, une $\mathcal{BDBO}$ doit supporter au moins un langage de requêtes. Celui-ci doit permettre d'ajouter, modifier et supprimer les instances d'une $\mathcal{BDBO}$ ainsi que de définir les classes qu'elles instancient et les propriétés qui les caractérisent. Pour chaque $\mathcal{BDBO}$, nous listons les langages de requêtes qu'elle supporte dans le tableau 3.2.

3.2.4 Extraction d'ontologies

Cette caractéristique indique si on peut extraire d'une $\mathcal{BDBO}$ un sous-ensemble des concepts de l'ontologie. Les concepts extraits de la base de données doivent être cohérents. Dans la plupart des $\mathcal{BDBO}$, une requête SPARQL portant sur le schéma d'ontologie peut permettre une extraction.

3.2.5 Sécurité des données

Cette caractéristique est fondamentale dans les bases de données classiques. Les données doivent être protégées des accès non autorisés ou mal intentionnés [154]. Une $\mathcal{BDBO}$ est sensée disposer des mécanismes permettant l'autorisation, le refus ou la révocation de privilège d'accès aux données et à une ontologie. La plupart des $\mathcal{BDBO}$ profite de mécanismes de sécurité offerts par les SGBD relationnel ou relationnel-objet sous-jacents.

3.2.6 Gestion des versions et évolutions des ontologies

Les ontologies sont susceptibles d'évoluer. En effet, les ontologies peuvent être modifiées quand elles ne satisfont plus les besoins d'une communauté. Les causes des modifications peuvent être :

- l'intégration de nouveaux concepts et/ou de nouvelles propriétés ;
- la suppression des concepts ;
- le changement de système de classification ;
- la modification de contraintes ;
- la modification des concepts (fusion ou séparation d'entités) ;
- la modification du domaine ou du co-domaine des propriétés ;
- etc.

Une évolution entraîne la définition de nouvelles versions de ces ontologies. Le principe de *versioning* [155] énonce que la gestion de l'évolution de l'ontologie exige de pouvoir conserver et accéder aux différentes versions de l'ontologie générées au cours de l'évolution, ainsi que de permettre de spécifier par un modèle les relations qui rendent compte des modifications faites entre les versions. Cette fonctionnalité est nécessaire pour préserver les relations sémantiques existantes entre les différents concepts des ontologies lors de l'intégration dans la $\mathcal{BDBO}$ de nouvelles versions. Cette caractéristique n'est pas prise en compte dans la plupart des $\mathcal{BDBO}$ utilisées ; seule la $\mathcal{BDBO}$ OntoDB est prédisposée à cette gestion et à la gestion des cycles de vie des instances décrit ci-dessous.

3.2.7 Cycle de vie des instances

C'est une fonctionnalité qui permet à une $\mathcal{BDBO}$ de gérer l'évolution des instances. De la même manière que les ontologies, les données ontologiques évoluent. Une évolution d'une ontologie peut modifier la structure des instances comme par exemple lors de l'ajout d'une propriété ou d'un concept. Même les opérations sur les instances (ajout, suppression ou mise à jour) entraînent l'évolution des instances. Une $\mathcal{BDBO}$ peut offrir un mécanisme pour pouvoir accéder à toutes les instances des concepts existants dans une version donnée et des mécanismes pour une gestion du cycle de vie des instances des classes.

3.2.8 Moteur d'inférence

Un moteur d'inférence est un outil qui permet de déduire de nouvelles connaissances ou données à partir de celles qui sont disponibles. Certaines $\mathcal{BDBO}$ disposent de leurs propres moteurs d'inférence pour la déduction de nouvelle données à partir de celles qui sont disponibles (Oracle, SOR). D'autres font usage des moteurs d'inférence ontologiques existant comme Racer, Pellet, Fact, Fact++.

3.2.9 Base de règles utilisateur

Parmi les $\mathcal{BDBO}$ qui disposent de moteurs d'inférence, certaines offrent à l'utilisateur la possibilité de créer sa propre base des règles. C'est le cas d'Oracle, de Jena ou de Sesame. Les règles sont en général de la forme : *clause* $\rightarrow$ *consequence*. La déduction peut se faire lors du chargement de données (pour les $\mathcal{BDBO}$ saturées) ou à l'initiative de l'utilisateur après le chargement de données.

3.2.10 Exportation des données

C'est une fonctionnalité qui permet d'extraire d'une $\mathcal{BDBO}$ un sous-ensemble des instances associées à certains concepts. Les instances extraites de la base de données doivent être consistantes et cohérentes. Cette fonctionnalité permet le partage et l'intégration des données.

3.2.11 Vérification de la consistance

Cette fonctionnalité permet à une $\mathcal{BDBO}$ de s'assurer qu'une ontologie est consistante c'est-à-dire qu'elle est conforme à son modèle. La vérification peut porter aussi sur les instances. La vérification de la consistance d'une instance se fait en utilisant les axiomes définis dans l'ontologie. La plupart des $\mathcal{BDBO}$ ont des modules qui effectuent cette vérification soit automatiquement (SOR, OntoDB) soit à la demande de l'utilisateur (Oracle, Jena, Sesame).

4 Etude de performances des $\mathcal{BDBO}$

Pour aller plus loin dans notre comparaison des $\mathcal{BDBO}$, nous présentons dans cette section une batterie d'expérimentation pour évaluer les six $\mathcal{BDBO}$ étudiées selon deux critères: le temps de chargement des instances ontologiques et le temps d'exécution de requêtes. Nous présenterons d'abord l'ensemble des données sur lesquels les expérimentations sont effectuées, puis les résultats des tests et leurs interprétations.

4.1 Ensembles de données et environnement de tests

Pour disposer des données ontologiques nécessaires pour nos tests de performance, nous avons utilisé le banc d'essai LUBM [30] qui utilise le formalisme OWL. Il crée une ontologie sur le domaine universitaire. Chaque université est constituée de 15 à 20 départements. Dans chaque département, nous retrouvons des *enseignants* de différentes *catégories*, *des étudiants*, *des cours*, etc. Le schéma de cette ontologie est présenté sur la figure 3.2.

Nous avons généré trois ensembles d'ontologies ayant respectivement 1, 5 et 10 universités (Lubm01, Lubm05 et Lubm10). Le nombre d'instances ontologiques dans chaque ensemble

Critères	**Oracle**	**SOR**	**OntoDB**	**Jena**	**Sesame**	**Db2rdf**
Formalisme	RDF, OWL	OWL	PLIB	RDF, OWL	RDF, OWL	RDF
Échange de données	oui	oui	Oui	oui	oui	oui
Support linguistique	oui	oui	oui,	non	non	non
Langage de requêtes	sql, sparl	sparql	ontoql, sparql	rql, Rdql, sparql	serql, sparql	sql, sparql
Extraction d'ontologies	oui	oui	oui	oui	oui	oui
Sécurité des données	oui	oui	oui	oui	oui	oui
Gestion des versions et évolutions des ontologies	non	non	oui	non	non	non
Cycle de vie des instances	non	non	oui	non	non	non
Moteur d'inférence	oui	oui	non	oui	oui	non
Base de règles utilisateur	oui	non	non	oui	oui	non
Exportation des données	oui	oui	oui	oui	oui	oui
Vérification de la consistance	oui	oui	oui	oui	oui	oui
Modèle de stockage d'ontologies	Vertic.	Spécif.	Spécif.	Vertic., specif.	Vertic., specif.	Vertic.
Modèle de stockage des instances	Verticale	Binaire	Horiz.	Vertical, specif.	vertic., binaire	vertic.
Architecture	2/4	3/4	4/4	2/4	2/4	2/4
SGBD actuellement utilisés	Oracle	DB2, Derby	Postgres	Oracle, Postgres, mysql	Oracle, Postgres, mysql	DB2

TABLE 3.2 – tableau comparatif des $\mathcal{BDBO}$ étudiées (vertic : verticale, horiz: horizontale, specif : spécifique)

ainsi que leurs nombres triplets sont indiqués dans le tableau 3.3. Notons que les outils proposés par Oracle et DB2RDF ne chargent que des données au format N-TRIPLE (*sujet, prédicat, objet*). En conséquence, une conversion des données de notre banc d'essai défini en OWL vers le format N-TRIPLE [152] fut nécessaire. Pour ce faire, nous avons utilisé l'API Jena appelée *rdfcat*.

Nos évaluations ont été réalisées sur un ordinateur (DELL) doté d'un processeur Intel(R) Xeon(R) CPU E31225 à 3.10 GHZ et d'une mémoire vive de 4GO et d'un disque dur de 500Go. Les systèmes de gestion bases de données (SGBD) utilisés sont : Oracle 11g pour la $\mathcal{BDBO}$ Oracle, IBM DB2 9.6 pour SOR et DB2RDF et PosgresSQL 8.2 pour OntoDB, Jena et Sesame.

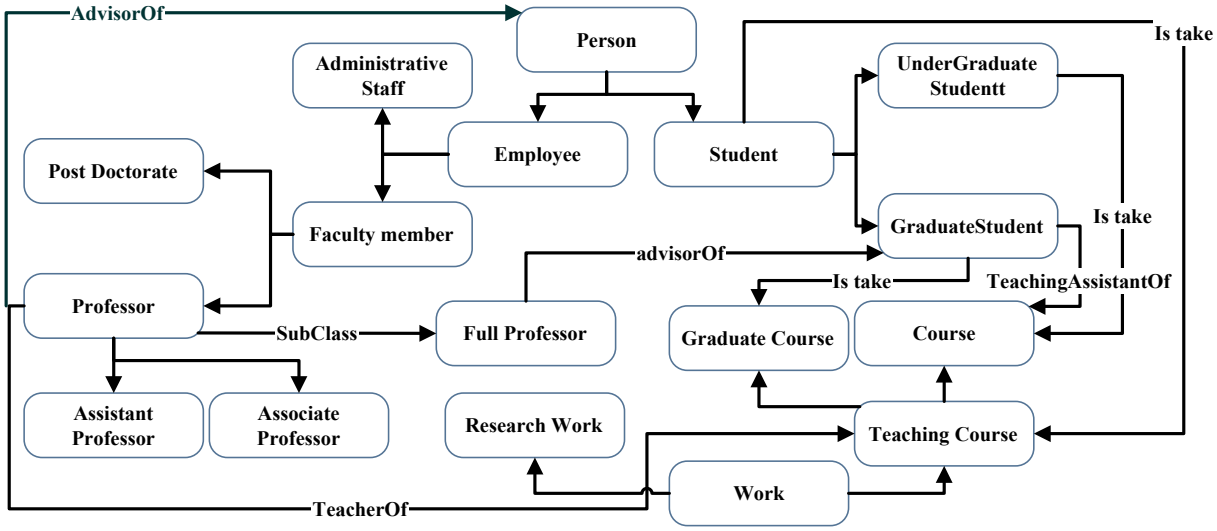


FIGURE 3.2 – Schéma de l'ontologie LUBM.

Jeux de Données	Nombre d'instances	Nombre de triplets
Lubm01	82415	100.851
Lubm05	516116	625.103
Lubm10	1052895	1.273.108

TABLE 3.3 – Ensembles de données générées.

4.2 Performances en termes de chargement des données (ontologie et instances)

Cette partie concerne les chargements de l'ontologie et de ses instances dans nos $\mathcal{BDBO}$. Notre mode opératoire se résume en quatre points : (1) la génération de données OWL ; (2) la conversion et concaténation de ces données dans un fichier N-Triple pour Oracle et DB2RDF ; (3) le calcul du temps de chargement (en sec.) des instances dans chaque $\mathcal{BDBO}$; et (4) l'interprétation des résultats.

Pour avoir des résultats homogènes, nous avons effectué quatre fois le chargement de chaque ensemble de données et nous avons retenu la moyenne obtenue. Les résultats sont consignés dans le tableau 3.4.

4.2.1 Chargement de données dans $\mathcal{BDBO}$ Oracle

Oracle offre trois moyens pour charger les ontologies et les données à base ontologique dans une base de données :

– Bulk load (*chargement global*);

– Batch load (*chargement par lot*);
– Load into table (*chargement par l'instruction insert into table*).

4.2.1.1 Bulk load (*chargement global.*) C'est la méthode la plus rapide pour charger les données à bases ontologiques dans une $\mathcal{BDBO}$ Oracle. Elle utilise l'utilitaire *SQLLoader*. Pour être chargées, les données doivent être mises dans des fichiers ou des tubes au format N-Triple. Elles sont d'abord chargées dans une table de relai avant d'être envoyées dans la base de données grâce à la procédure *SEM_APIS.BULK_LOADER_FROM_STAGING_TABLE*. Cette méthode de chargement a le défaut de ne pas prendre en compte les littéraux de plus de 4ko. Le processus de chargement est schématisé sur la figure 3.3.

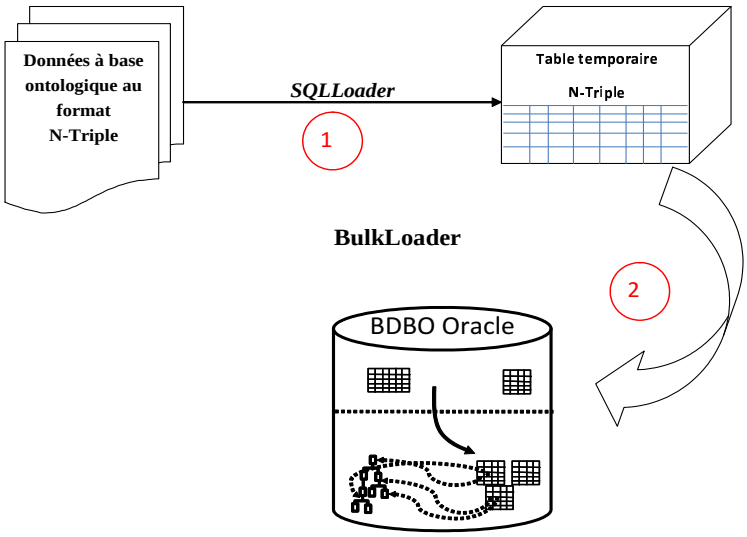


FIGURE 3.3 – Processus de *chargement global* d'Oracle.

4.2.1.2 Batch load (*chargement par lot*). Cette méthode a l'avantage de prendre en compte les littéraux de plus de 4ko. Il utilise une classe java (la classe BatchLoader) pour charger les données au format N-Triple d'un fichier.

4.2.1.3 Load into table (*chargement par l'instruction insert into table*) C'est la méthode classique d'insertion de données dans une base de données. Elle utilise les instructions du LMD [19] (Insert, update) sur les tables de données sémantiques. Elle est recommandée pour le chargement de faibles quantités de données. Cette méthode nécessite l'utilisation du constructeur *SDO_RDF_TRIPLE_S* (voir exemple 2).

Pour nos tests, nous avons fait le *chargement global* (*Bulk Load*) et le *chargement par lot* (*Batch Load*). Pour le premier, nous avons mesuré le temps de chargements des données dans la

19. langage de manipulation des données

table de relais (appelée *Staging Table*) et le temps de transfert dans le modèle sémantique. Pour réaliser cette évaluation, nous avons utilisé l'utilitaire d'Oracle *Sqlloader* qui renvoie le temps qu'il met pour charger les données. Pour le temps de transfert, nous avons utilisé le chronomètre d'Oracle. Le temps de chargement est alors égal à la somme de ces deux temps.

Pour le *chargement par lot*, la mesure du temps a été facilitée par l'API utilisée. En effet, à la fin du chargement, un résumé du déroulement du processus est renvoyé et le temps mis est affiché. Dans les deux cas, après chaque chargement, nous supprimons et recréons le modèle sémantique, avant d'effectuer un autre chargement.

Exemple 2

Pour insérer le triple (Bery, travaille, Oracle Techno), dans le modèle MdLias, dont la table sémantique est tab_lias, on exécute l'instruction : INSERT INTO tab_lias VALUES (1000, sdo_rdf_triple_s('MdLias', 'Bery','travaille', 'Oracle Techno'));

4.2.2 Chargement de données dans la $\mathcal{BDBO}$ SOR d'IBM

SOR dispose d'un module d'importation permettant le chargement des données ontologiques dans une base de données relationnelle. Ce module est formé d'un analyseur (parser) OWL et de deux traducteurs (*DB Translator* et *TBox Translator*). Le parseur examine et charge les fichiers OWL dans un modèle *EODM*[20]. Le *DB Translator* charge les assertions *ABox* dans la base de données. Le *TBox Translator* a deux tâches. La première consiste à charger les axiomes de la *TBox* dans le raisonneur DL et la seconde consiste à faire des déductions de données et de les insérer dans la base de données. Pour faire une évaluation de performance de SOR, nous nous sommes servis de l'*IODT*[21]. Nous avons utilisé comme SGBD, IBM DB2 Express 9. Nous avons chargé les différents ensembles de données LUBM générées dans SOR. A la différence des chargements sous Oracle et DB2RDF, les données n'ont pas subi de transformation de format, elles sont restées au format OWL. Pour chaque ensemble de données, nous avons mesuré le temps mis pour son chargement.

4.2.3 Chargement de données dans la $\mathcal{BDBO}$ DB2RDF

A l'image d'Oracle, DB2RDF accepte les données au format N-triple. Pour cela, nous avons utilisées les données converties en N-Triple. Puis nous avons utilisé les fonctions pour le chargement fournies dans un environnement Eclipse, pour charger les données. Le temps de chargement est mesuré pour chaque ensemble de données.

20. Eclipse Modeling Framework (EMF) Ontology Definition Metamodel
21. Integrated Ontology Development Toolkit - http://www.alphaworks.ibm.com

4.2.4 Chargement de données dans la $\mathcal{BDBO}$ OntoDB

Bien qu'OntoDB soit prédisposé à prendre en compte tout formalisme d'ontologies ; il n'est actuellement équipé que d'un module de chargement d'ontologie PLIB. Vu le fait que l'ontologie que nous utilisons pour les expérimentations est représentée en OWL, nous devons proposer un moyen pour charger cette ontologie dans OntoDB. Une solution possible serait de faire un programme permettant d'obtenir une ontologie PLIB à partir d'une ontologie OWL et de la charger dans OntoDB. Mais, vu la différence et la diversité des constructeurs et des opérateurs des deux formalismes, l'ontologie obtenue de cette manière n'est ni congruente ni équivalente à l'ontologie de départ. Nous avons donc préféré développer un module de chargement d'OWL dans OntoDB.

Ainsi, nous avons mis en place un module d'importation des ontologies OWL et leurs instances dans OntoDB. Ce module prend en paramètre une ontologie OWL et ses instances et, après une analyse, il extrait les concepts (classes et propriétés) et leurs instances, puis il crée les tables correspondantes dans la BD et les remplit par les instances appropriées. Nous nous sommes intéressés à la partie *canonique* de l'ontologie OWL (les concepts primitifs).

De manière formelle, notre module est une application d'appariement des concepts du formalisme d'ontologie OWL en concepts PLIB disponibles dans OntoDB. Si on note T, cette application, T est définie par : $T :< O_{OWL}, E_{owl} > \rightarrow < O_{OntoDB}, E_{OntoDB} >$ où O_{OWL} et E_{OWL} sont respectivement une ontologie et un concept de l'ontologie OWL, et où O_{OntoDB} et E_{OntoDB} sont respectivement ontologie et l'objet(table, attribut ou valeur) dans OntoDB (au formalisme d'ontologie PLIB). Soit C, P l'ensemble de classes et des propriétés canoniques de l'ontologie O en OWL. T se base sur deux considérations principales :

- $\forall c \in C$, $T(O_{OWL}, c)$ est une classe de PLIB définie dans OntoDB, par une table ;
- $\forall p \in P$, $T(O_{OWL}, p)$ est une propriété de PLIB attachée à Domaine(p) c'est-à-dire, toutes les classes qui sont domaines de la propriété p.

Il faut noter que cette correspondance n'est pas parfaite et elle est irréversible du fait que les concepts OWL non canoniques ne sont pas pris en compte et qu'il n'est pas possible de retrouver l'ontologie initiale par une application inverse. Notons également que notre utilitaire ne travaille que sur des ontologies *acycliques*. Cela est une exigence de modèle d'ontologie PLIB qui est au cœur du système OntoDB. Les ontologies PLIB ne comporte pas de cycles c'est-à-dire que l'organisation des classes et des propriétés est telle qu'on ne peut pas avoir un cycle. Or l'ontologie LUBM renferme des cycles. Nous avons rompu les cycles en transformant les propriétés de type *objectproperty* à l'origine des cycles en *datatypeproperty*.

4.2.5 Chargement de données dans les $\mathcal{BDBO}$ Jena et Sesame

Pour les chargements de données dans les $\mathcal{BDBO}$ Jena et sésame, nous avons utilisé un environnement Eclipse identique à celui utilisé pour SOR et DB2RDF. Pour chaque $\mathcal{BDBO}$, nous avons mis en œuvre les modèles et fonctions nécessaires pour le chargement. Etant donné que

Sesame donne la possibilité d'avoir des $\mathcal{BDBO}$ de *type I* et de *type II*, nous avons expérimenté ces deux types. Dans ce qui suit, nous appelons *SesameBdsI*, le système Sesame utilisant la représentation verticale ($\mathcal{BDBO}$ de type I) et *SesameBds2*, le système Sesame utilisé comme $\mathcal{BDBO}$ de type II. Les temps de chargement moyens sont consignés dans le tableau 3.4.

4.2.6 Temps de chargement

Les temps moyens de chargement des différents jeux de données dans les $\mathcal{BDBO}$ étudiées sont consignés dans le tableau 3.4. La figure 3.4 présente un histogramme de ces résultats. Les temps de chargement du système OntoDB n'y sont pas représentés du fait qu'ils sont d'une échelle de grandeur supérieure à celles des autres $\mathcal{BDBO}$.

Dataset	Lubm01	Lubm05	Lubm10
Oracle Batch	42	222	302
Oracle Bulk	12	55	116
IBM SOR	90	590	1147
DB2RDF	11	53	109
Jena2	25	188	558
SesameBdsI	16	158	391
SesameBdsII	27	231	522
OntoDB	15975	72699	146023

TABLE 3.4 – Récapitulatif des temps de chargement (sec).

4.2.7 Interprétation des résultats

En termes de chargement, comme on le constate sur le tableau 3.4 et la figure 3.4, DB2RDF et Oracle (Bulk) fournissent de très bons résultats, suivi dans l'ordre de SesameBdsI, Jena, SesameBdsII, Oracle Bulk et SOR. OntoDB se montre plus lente que les autres $\mathcal{BDBO}$ étudiées. Pour DB2RDF et Oracle (Bulk), cette performance peut s'expliquer par le fait que peu d'opérations sont effectuées sur des données à l'insertion dans ces systèmes mais aussi par la puissance des SGBD utilisés (oracle et DB2). La $\mathcal{BDBO}$ Sésame utilisant la table de triplets se relève plus performant que Jena [22] par contre, Sesame utilisant une représentation binaire donne de moins bonnes performances que Jena. En effet, Jena utilisant les tables *property-table* et *class-property-table* prend du temps pour dispatcher les données dans les tables indiquées. La $\mathcal{BDBO}$ SOR prend assez de temps pour les transpositions du schéma d'ontologies (T-Box) et des instances ontologies (A-Box) en des tuples pour les différentes tables. Elle met aussi un temps important pour faire les inférences des données au niveau de l'ontologie et des instances.

22. Jena version 2

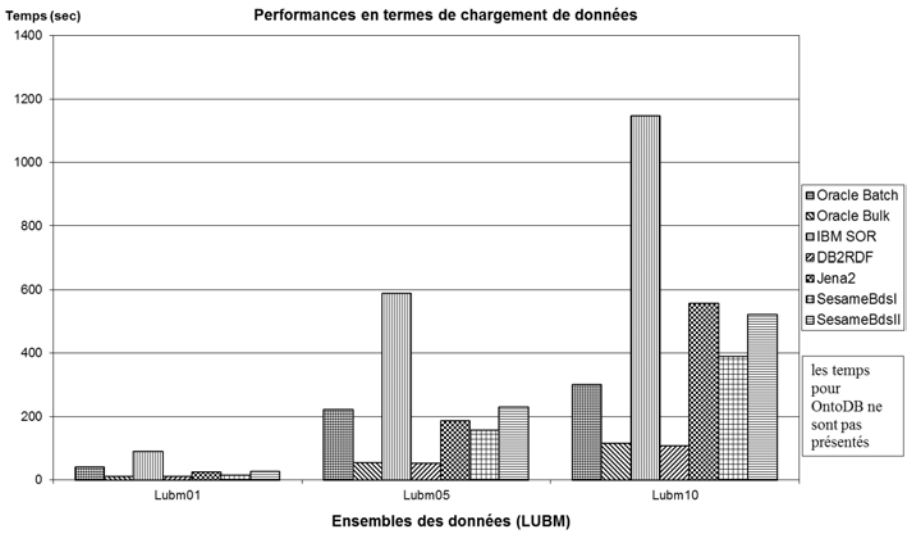


FIGURE 3.4 – Temps de chargement des données.

Cela justifie le fait que le temps de chargement est moins bon que ceux des certaines $\mathcal{BDBO}$ mises en évidence.

La lenteur que l'on observe sur OntoDB s'explique par les multiples sous-requêtes présentes dans la plupart des requêtes d'insertion des instances. En effet, OntoDB utilise les identifiants d'objet (*Oid*) pour traduire les références aux objets(les clés étrangères) surtout qu'il stocke les identifiants (*oid*) pour chaque attribut référencé. Lors du chargement, le système exécute des sous-requêtes pour chercher les *oid* des instances de ces attributs. A titre d'exemple, l'insertion d'une instance de *GraduateStudent* comporte au moins quatre sous-requêtes pour extraire les *oid* des cours suivis, du département d'appartenance, du *conseiller* (*advisor*) et de l'université d'où il était diplômé (*undergraduateDegreeFrom*). Pour éviter des erreurs lors de l'insertion, OntoDB maintient un ordre qui impose la création des superclasses d'une classe avant celle de la classe et de même pour les insertions des instances. Tous ces mécanismes contribuent à une cohésion de données mais allongent le temps de chargement et ne facilitent pas le passage à l'échelle.

On peut ainsi constater qu'en termes de chargement de données, les $\mathcal{BDBO}$ de type I sont plus rapides que les $\mathcal{BDBO}$ de type II et III.

4.3 Performances en termes de temps de réponse de requêtes

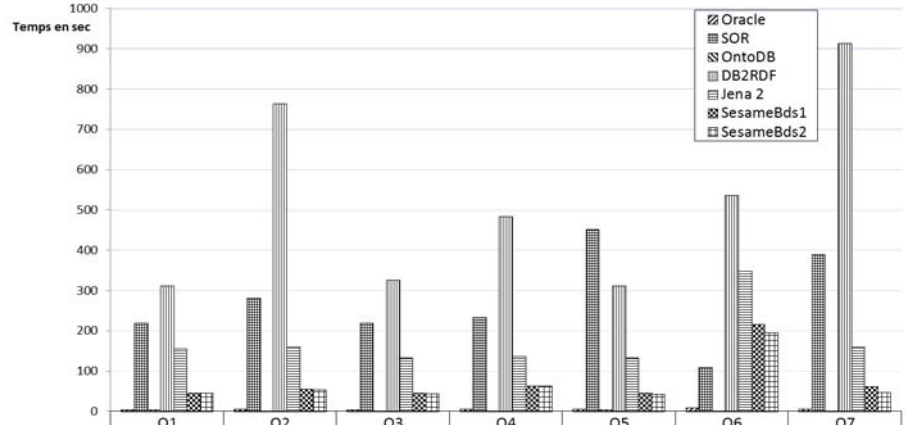
Nous avons mesuré les temps d'exécution des requêtes (en sec.) sur les six $\mathcal{BDBO}$. Comme pour la mesure des temps de chargement, pour chaque requête, nous avons effectué quatre fois les mesures et le temps moyen est retenu. Les requêtes utilisées sont celles proposées dans le banc d'essai LUBM. Nous les avons traduites pour qu'elles soient acceptées par nos $\mathcal{BDBO}$. Nous ne présentons ici que les résultats avec le jeu de données Lubm01, les autres résultats

étant similaires. Les résultats ont donné lieu aux histogrammes des figures 3.5 et 3.6. Nous avons utilisé la base de règles *owlprime* pour Oracle.

4.3.1 Interprétation des résultats

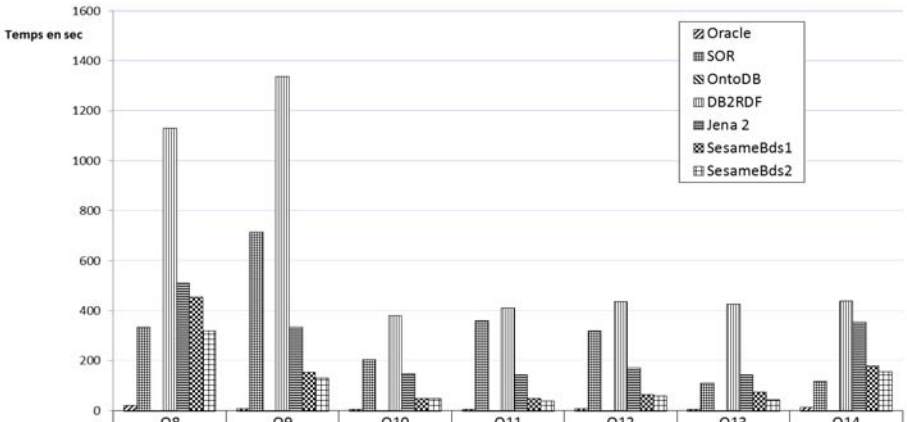
Pour ce qui est des performances en termes de temps de réponses aux requêtes, OntoDB réagit mieux que les autres $\mathcal{BDBO}$ étudiées. En effet, OntoDB, ayant stocké des *oid* des champs référencés, se sert de l'opérateur de référencement (*DEREF*) qui s'appuie sur ces *oid*, évitant ainsi certains calculs. En plus, OntoDB avec sa représentation horizontale des instances, effectue moins d'opérations de jointure que les autres systèmes parce que toutes les requêtes portant sur les propriétés d'une même classe se ramènent à des opérations de sélection sur la ou les tables relatives à cette classe ; ce qui n'est pas le cas dans les autres $\mathcal{BDBO}$. La requête 4 de LUBM en est une bonne illustration. Cette requête est composée de cinq triplets ayant tous un même domaine (*Professor*). Cette requête ne nécessite aucune jointure dans OntoDB mais demande au moins quatre jointures dans les autres systèmes. Nous pouvons aussi ajouter le fait que OntoDB scanne moins de données que les autres systèmes. Effectivement, une simple requête portant sur la sélection d'un seul attribut (comme la requête 6) nécessite un parcours de toute la table de triplets (c'est le cas dans Oracle, SesameDbsI et DB2RDF) alors que dans OntoDB, on ne parcourt que la ou les table(s) de l'attribut en question (les tables *UndergraduateStudent* et *GraduateStudent* pour la requête 6 de LUBM). D'où le bon temps de réponse.

Oracle vient en second lieu suivi de Sesame. Nous remarquons que Sesame dans la représentation binaire (sesameBdsII) réagit un peu mieux que la représentation verticale (sesameBdsI). Cette différence est bien marquée à la requête 14 où SesameBds1 parcourt toute la table de triplets alors que SesameBds2 ne parcourt que la table *typeof*. SOR se relève lent car il réalise plusieurs jointures pour toutes les requêtes. SOR donne des résultats plus efficaces que DB2RDF. Pourtant les deux utilisent le même SGBD (DB2 9.6). La mauvaise performance de DB2RDF peut trouver sa réponse dans la multitude des tables qui accompagne la table de triplets et qui impliquent plusieurs jointures pour répondre à des requêtes. En dépit de ses résultats peu performants, SOR réalise un bon travail de déduction. Par exemple à la requête 12 du LUBM, les autres $\mathcal{BDBO}$ ne retournent pas de résultat, car il y a pas de déclaration explicite d'instance de *Chair*. Cependant, SOR a réussi à renvoyer des résultats en se basant sur la propriété *headOf*, puisqu'un individu instance de *Professor* est instance de *Chair* s'il est à la tête d'un *Département*. Si on ajoute une règle traduisant cette assertion à des $\mathcal{BDBO}$ prenant en compte "les règles utilisateur" comme Oracle, elles fourniront aussi ces réponses. Mais ce n'est pas une règle prédéfinie dans ce système. Nous notons que Jena et Sesame utilisent aussi des moteurs d'inférence pour la déduction de données mais en utilisant des fichiers standard. Cette option est encore difficile à mettre en œuvre dans les bases de données. Mais, il est possible de faire appel à un moteur d'inférence lors de chargement de données et de stocker dans la base de données toutes les données y comprises celles déduites mais cela serait au détriment des performances de chargement. Il n'est pas exclu d'envisager l'utilisation d'un moteur d'inférence lors



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Oracle	4,68	5,56	3,74	5,82	5,88	9,51	6,18
SOR	220	281	218,5	234	452	109	390,33
OntoDB	4,53	2	0,75	1,834	4,1	0,24	2,15
DB2RDF	312	764,33	325,33	483,33	312	535,66	913
Jena 2	156	160	133	137	134	348	161
SesameBds1	46,5	55	45	63	46,5	216	62
SesameBds2	46,5	54	44	64	43	195	47

FIGURE 3.5 – Temps d'exécution de requêtes LUBM de 1-7.



	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Oracle	18,31	6,45	4,52	4,29	6,15	5	10,72
SOR	332,66	713,66	203,33	358,66	318,5	109	117,33
OntoDB	1,31	2,5	1,54	0,81	2,4	0,38	0,218
DB2RDF	1128,33	1336,66	379,66	410	436	426	438
Jena 2	509,33	334	146	141	171	141	353,6
SesameBds1	452,66	153,33	47	46,5	63	73	176,66
SesameBds2	317	130	47	36,66	57	42	156

FIGURE 3.6 – Temps d'exécution de requêtes LUBM de 8-14.

de l'exécution des requêtes mais cela allongera considérablement le temps de traitement des requêtes. Pour ce qui concerne OntoDB en matière de moteur d'inférence, elle n'en dispose pas mais lors d'insertion de données, elle sature la base de données de nouveaux faits déductibles des relations de subsomption. Cela peut aussi expliquer sa mauvaise performance en temps de chargement.

5 Paramètres influant identifiés à partir de cette étude

Dans cette étude empirique, nous avons constaté que les résultats de nos expérimentations sont divers dans les $\mathcal{BDBO}$ étudiées. Cette hétérogénéité est due à un certain nombre de paramètres parmi lesquels nous pouvons identifier : le SGBD, le format des instances, le modèles de stockage, l'architecture et le type de requêtes. En effet, les performances des SGBD ont influencé nos résultats, nous pouvons le constater dans les chargements de données où les SGBD commerciaux aguerris dans les chargements de données (Oracle, DB2) ont fourni de meilleurs temps de chargement. L'analyse et le formatage des données peuvent aussi constituer de facteurs influençant les systèmes. C'est le cas de SOR IBM et d'OntoDB qui prennent les instances OWL, les traitent et les repartissent dans les différentes tables appropriées. Le format de fichiers des instances peut également être mentionné : certaines $\mathcal{BDBO}$ exigent un format spécifique, cela entraine une conversion qui peut influencer les performances des systèmes si la conversion est faite lors des chargements des données.

L'architecture a certainement joué un rôle influant. Pour une architecture classique ("*deux quarts*"), nous avons remarqué que les chargements des données y sont effectués rapidement mais les temps de requêtes dépendent des types de requêtes et du modèle de stockage utilisé. Les architectures "*trois quarts*" et "*quatre quarts*" sont propices pour des requêtes portant sur le schéma d'ontologie car en séparant les ontologies des données, elles répondent rapidement aux questions sur les ontologies. L'influence du type de requête se dégage à travers les requêtes de type *mono-triplet* qui donne de bon résultat sur les $\mathcal{BDBO}$ binaires et horizontales sur tout SGBD mais pour les requêtes de type jointure et de type sélection *pluri-trplets* les résultats sont mitigés.

L'influence du modèle de stockage est traduite par la portée des requêtes. En effet, une requête SPARQL peut, suivant le modèle de stockage, porter une ou plusieurs tables, et entraîner des opérations de jointure ou d'union qui sont coûteuses dans le traitement des requêtes. Le modèle de stockage vertical est bon en matière de chargement de données mais il est coûteux pour les requêtes (auto-jointure sur la table de triplets). Le modèle de stockage binaire est bien indiquée pour les requêtes de sélection *nomo-triplet* et le modèle de stockage horizontale se comporte bien pour les requêtes de sélections (*mono-triplet* et *pluri-triplets*) mais il peut entrainer les opérations d'unions voire de la redondance selon la gestion de l'héritage choisie.

Pour le SGBD, dans la suite de notre travail, nous utiliserons le même pour toutes nos $\mathcal{BDBO}$. Nous proposerons un modèle de coût qui doit prendre en compte les modèles de sto-

ckage, l'architecture et le type de requêtes.

6 Conclusion

Dans ce chapitre, nous avons proposé un modèle unifié pour les $\mathcal{BDBO}$ et des critères de comparaison pour pouvoir comparer les $\mathcal{BDBO}$ de manière globale et également de manière plus détaillée. Nous avons comparé les caractéristiques de quelques $\mathcal{BDBO}$ issues des milieux académiques et industriels. Etant donné que les $\mathcal{BDBO}$ sont conçues pour le passage à l'échelle en termes de volumétrie des ontologies et des instances, il est également important d'étudier les performances des $\mathcal{BDBO}$ actuelles. Tout comme cela a été fait pour les bases de données relationnelles, nous avons effectué une comparaison expérimentales des performances des $\mathcal{BDBO}$ en termes de temps de chargement des instances ontologiques et de temps d'exécution des requêtes. En termes de temps de chargement, cette étude nous a permis de voir que les $\mathcal{BDBO}$ issues du milieu industriel, aguerries dans les chargements de données (Oracle, DB2RDF) fournissent de meilleurs temps de chargement. Cette bonne performance s'explique aussi par le fait que ces systèmes utilisent pour les instances et pour les ontologies le modèle de stockage vertical et l'architecture de type I. Ils ont donc peu de transformations à effectuer lors des chargements de données. Nous avons constaté que les $\mathcal{BDBO}$ horizontales sont lentes en chargement, cela est dû aux multiples opérations pour extraire les données des différentes tables et aussi pour établir des liens sémantiques entre les données et les concepts ontologiques.

En termes de temps d'exécution des requêtes, les résultats sont divers : pour certaines requêtes, certaines $\mathcal{BDBO}$ fournissent de meilleurs temps d'exécution que d'autres $\mathcal{BDBO}$ et pour d'autres requêtes, leurs temps d'exécution sont moins bons. Tout dépend de la structure de la requête (*mono-triplet, pluri-triplets,* etc.) Certaines $\mathcal{BDBO}$ comme IBM SOR sont saturées c'est-à-dire qu'elles contiennent les données initiales et les données inférées. D'autres ne contenant que des instances fournies sont dites non saturées. Ces dernières peuvent faire usage d'un moteur d'inférence pour faire de déduction de données. Il existe des $\mathcal{BDBO}$ qui donnent à l'utilisateur la possibilité de définir sa base de règles de déduction. Les résultats de certaines requêtes dépend du raisonnement offert par la $\mathcal{BDBO}$.

Ayant identifiés quelques facteurs influençant les performances des $\mathcal{BDBO}$, au prochain chapitre, nous proposerons un modèle de coût (Chapitre 4) qui prendra en compte les diversités de $\mathcal{BDBO}$ mais aussi les types de requêtes.

Chapitre 4

Modèles de Coût pour les Bases de Données à base Ontologiques

Sommaire

1 Introduction

Les modèles de coût représentent une composante importante des bases de données. Ils ont été utilisés par les différentes générations de bases de données pour répondre à des questions cruciales. La première utilisation des modèles de coût concernait l'optimisation de l'exécution de requêtes. Historiquement, les optimiseurs de requêtes des SGBD utilisaient une approche dirigée par des règles (*Rule-based Approach*). Cette approche utilise un ensemble de règles bien définies comme exécuter les opérations de sélections avant les opérations binaires comme la jointure. Avec l'apparition des schémas de bases de données impliquant un nombre important de tables (relations), cette approche a été substituée par des optimiseurs dits *dirigés par des modèles de coût* (*Cost-based Approach*) [80, 156]. Cette approche est utilisée actuellement dans la majorité des SGBD commerciaux. La deuxième utilisation des modèles de coût concerne le problème de la conception physique. Comme nous l'avons déjà évoqué au chapitre 3, durant cette phase un ensemble de structures d'optimisation (comme les vues matérialisées, les index, le partitionnement, etc.) est sélectionné. Vue la taille importante de l'espace de recherche de chaque problème lié à la sélection d'une ou plusieurs structures d'optimisation, des méta-heuristiques (par exemple, les algorithmes génétiques [116] ou le recuit simulé [157]) dirigées par des modèles de coût ont été proposées. La troisième utilisation des modèles de coût concerne les étapes de la phase de déploiement des bases de données sur des plateformes distribuées et parallèles [158].

Un modèle de coût est généralement développé en fonction des composantes principales d'un SGBD [159, 160], d'où sa décomposition en trois parties : (1) *le coût des entrées-sorties (IO)* pour lire et écrire entre la mémoire et le support de stockage comme les disque, (2) *le coût CPU* et (3) *le coût COM de communication sur le réseau* (si les données sont réparties sur le réseau). Ce dernier est généralement exprimé en fonction de la quantité totale des données transmises [161, 162]. Il dépend de la nature de la plateforme de déploiement.

Un modèle de coût peut être vu comme une fonction ayant des entrées et une sortie. Les entrées représentent (i) le schéma d'une base de données (entrepôt de données) gérée par un SGBD utilisant un modèle de stockage et une architecture donnés, (ii) une charge de requêtes, (iii) la plateforme de déploiement (machine centralisée, distribuée, parallèle, cloud, etc.). Pour chaque entrée, un ensemble de paramètres lui sont associés. Nous distinguons deux catégories de paramètres : *les paramètres calculés* (ex. les facteurs de sélectivité des prédicats) et les *paramètres non calculés* comme la taille des champs, la taille du support de stockage, etc. L'obtention des paramètres calculés se fait à l'aide des formules mathématiques et d'hypothèses comme par exemple les chemins d'accès aux données. Pour la partie requête, le calcul de ses paramètres dépend de la politique de traitement des requêtes sur la plateforme de déploiement. Ces entrées représentent clairement les entrées d'un problème de conception physique que nous avons détaillées dans le chapitre 2. En sortie, nous aurons un coût final d'exécution d'une charge de requêtes. Cette valeur de coût sera exploitée par les algorithmes selon le besoin.

Une large panoplie de modèles de coût a été proposée dans les différentes générations de SGBD selon la finalité (l'optimisation de requêtes, la sélection des structures d'optimisation, le problème de déploiement, etc.). En examinant la littérature sur les modèles de coût, nous avons remarqué deux phénomènes intéressants.

1. Le développement de modèle de coût devient un enjeu important à chaque apparition d'une génération de SGBD ayant son propre modèle de stockage. Ce fut par exemple, le cas des bases de données orientées objet, où un nombre important de travaux de recherche ont été effectués. On peut citer les travaux de Georges Gardarin [163, 164] et de Elisa Bertino [165, 166].
2. Lorsqu'une nouvelle génération propose des modèles différents de ceux utilisés dans les précédentes générations, les chercheurs proposent des modèles de stockage pour stocker ces nouveaux modèles dans les anciens modèles et des modèles des coûts pour leur évaluation. C'est le cas des bases de données XML, lorsqu'on a proposé des solutions de stockage relationnel [167, 168].

Les $\mathcal{BDBO}$ ont entraîné les deux phénomènes précédents, ce qui nous conduit à proposer un modèle de coût qui prend en compte la diversité des $\mathcal{BDBO}$.

Ce chapitre est consacré à la définition de l'ensemble de paramètres des entrées de notre modèle de coût. Une fois définis, nous proposons notre modèle de coût en prenant en compte les spécificités des $\mathcal{BDBO}$. Finalement, nous utilisons notre modèle de coût pour évaluer théoriquement quelques exemples de $\mathcal{BDBO}$ et comparons les résultats à ceux obtenus expérimentalement dans le chapitre précédent.

2 Revue des modèles de coût dans le contexte des $\mathcal{BDBO}$

Nous avons identifié plusieurs modèles de coût qui ont été proposés pour les $\mathcal{BDBO}$ [169, 170, 171, 172, 173]. La particularité de ces derniers est qu'ils se focalisent sur l'estimation de quelques paramètres liés aux $\mathcal{BDBO}$ et aux requêtes associées. Nous détaillons ces travaux dans la suite en notant qu'ils ne traitent pas de la diversité des $\mathcal{BDBO}$.

Maduko et al. [169] proposent des estimations de la cardinalité des patrons de graphes RDF à l'aide de statistiques sémantiques et structurelles. Ces derniers portent sur tous les sous-graphes d'une certaine taille contenus dans un ensemble de données RDF. Pour effectuer ce calcul, les auteurs font appel au schéma de l'ontologie. Tous les sous-graphes de l'ensemble de données peuvent être déduits à partir du graphe du schéma de l'ontologie et annotés avec leurs cardinalités exactes dans la base de données.

Shironoshita et al. [170], de leur côté, ont proposé une stratégie d'estimation de cardinalité construit sur des statistiques basées sur des prédicats. Les auteurs utilisent une fonction de probabilité pour estimer la cardinalité du résultat d'une requête sur un ensemble de données d'une ontologie. Comme dans les travaux de Maduko et al. [169], cette stratégie nécessite un

schéma d'ontologie, et est conçue dans un contexte de systèmes distribués.

Stocker et al. [171] ont proposé un modèle de coût pour optimiser les patrons de graphe RDF. Ce modèle utilise la sélectivité des patrons de triplets et porte sur l'optimisation des requêtes statiques. Les auteurs proposent de réorganiser les patrons de triplets en fonction de leur sélectivité. La sélectivité d'un patron de triplets est estimée en multipliant la sélectivité de chacun de ses composants (*sujet, prédicat et objet*). Les hypothèses d'indépendance et d'uniformité des données ont aussi été faites. Le calcul des statistiques est fonction de la taille de l'ensemble de données. Dans des ensembles de données de taille importante, générer les statistiques des données peut provoquer des calculs qui influenceraient les performances du système. Ainsi, l'évolutivité peut poser un problème à cette approche. Cependant, nous trouvons que cette approche peut-être utile si les statistiques sont calculées d'avance et gardées dans des tables appropriées.

Neumann et Weikum [172] ont élaboré une stratégie pour construire des statistiques pour tous les sous-graphes possible d'un ensemble de données RDF sans un schéma d'ontologie. Ils se sont intéressés aux sous-graphes de formes arbitraires (graphes en forme de chaîne, d'étoiles, etc.). Ils utilisent la technique d'indexation afin d'améliorer le traitement des requêtes. Leur approche repose en grande partie sur les capacités d'indexation natives. Ainsi, leur stratégie et sa mise en œuvre peuvent difficilement être adaptées aux différents types de $\mathcal{BDBO}$.

Kaoudi et al. [173] ont repris et adapté le calcul de la sélectivité de patrons de triplets de Stocker et al. [171] pour un environnement distribué (DHT : Distributed Hash Table). Neumann et Moerkotte [174] se sont intéressés à la sélectivité de patron de triplet et à l'estimation des cardinalités qu'ils utilisent dans l'ordonnancement des opérations de jointure.

3 Les paramètres de notre modèle de coût

Rappelons qu'une $\mathcal{BDBO}$ est formalisée par un 7-uplet :

$$BDBO :< MO, I, Sch, Pop, SM_{MO}, SM_{Inst}, Ar >$$

Établir un modèle de coût doit prendre en considération l'ensemble de ces composantes.

Comme dans les bases de données classiques, les principaux paramètres à prendre en compte dans la fonction de coût sont : les coûts d'accès disque, les coûts de stockage des résultats intermédiaires, les coûts de calcul, etc. Dans notre travail, nous avons établi certaines hypothèses: (i) la non prise en compte du coût de raisonnement, (ii) le coût des calculs est négligeable au regard des coûts d'accès, et (iii) l'existence de statistiques sur les données (à partir de l'ontologie et des catalogues) plus précisément, pour chaque relation nous considérons que les statistiques suivantes sont disponibles : le nombre de tuples stockés, de pages sur lesquelles sont stockées les tuples, de pages de la mémoire centrale et la taille des tuples. Nous admettons également les hypothèses faites dans le système R [175] stipulant que la distribution des valeurs est uniforme et que les attributs sont indépendants les uns des autres.

Les hypothèses précédentes sont habituellement faites lors de la construction du modèle de coût. Concernant l'hypothèse sur le raisonnement, nous distinguons deux types de $\mathcal{BDBO}$, à savoir les $\mathcal{BDBO}$ saturées et celles non saturées. Une $\mathcal{BDBO}$ est dite saturée si elle contient en plus des instances ontologiques initiales, des instances déduites, sinon elle est dite non saturée. Certaines $\mathcal{BDBO}$ sont automatiquement saturées lors du chargement de données. C'est par exemple le cas de SOR d'IBM. D'autres $\mathcal{BDBO}$ comme Jena et Sesame peuvent être saturées à la demande lors du chargement en utilisant un moteur d'inférence. On trouve aussi des $\mathcal{BDBO}$ qui ne sont pas saturées lors du chargement mais qui peuvent être saturées à tout moment à la demande de l'utilisateur. Oracle en est un exemple. Compte tenu de notre hypothèse sur le raisonnement, notre fonction de coût se base sur les $\mathcal{BDBO}$ saturées. Pour l'appliquer à des $\mathcal{BDBO}$ non saturées faisant des déductions, il faut ajouter le coût de déduction au coût d'exécution de la requête.

3.1 Paramètres de la fonction du coût

Soit q une requête et bds une $\mathcal{BDBO}$ sur laquelle q sera exécutée et $cout$ la fonction coût. En ce qui concerne l'architecture d'une $\mathcal{BDBO}$, nous pouvons remarquer que par rapport aux modèles de coût classiques, les modèles de coût dans les $\mathcal{BDBO}$ connaissent une " *augmentation* " traduite par les coûts d'accès au différentes parties de l'architecture :

– Dans une architecture *type I* :

$$Cout(q, bds) = cout\_acces(métabase) + cout\_acces(données)$$

– Dans une architecture *type II* :

$$Cout(q, bds) = cout\_acces(métabase) + cout\_acces(ontologie) + cout\_acces(données)$$

– Dans une architecture *type III* :
$Cout(q, bds) = cout\_acces(métabase) + cout\_acces(métaschéma) + cout\_acces(ontologie) + cout\_acces(données)$.

Le coût d'accès à la méta-base *(cout_acces(métabase))* faisait partie du modèle de coût dans les bases de données classiques est jugé négligeable du fait que la méta-base peut être gardée en mémoire (buffer). Ainsi, le coût d'une requête est fonction de l'architecture et du modèle de stockage de la $\mathcal{BDBO}$. Notre formalisation des $\mathcal{BDBO}$ fournit toutes ces informations. La signature de la fonction de coût est donc : $cout :< q, BDBO > \rightarrow \mathbb{R}$. Nous supposons que le *méta-schéma*, et l'*ontologie* sont de petites tailles et qu'ils peuvent être placés en mémoire centrale comme la *méta-base*. Cette hypothèse est réaliste pour les ontologies standardisées que l'on retrouve dans le domaine de l'ingénierie. Ainsi, dans toutes les architectures, le coût peut se résumer au coût d'accès aux données notamment en termes de nombre d'entrées/sorties c'est-à-dire $Cout(q, bdbo) = cout\_acces\ (données)$.

Le coût d'exécution d'une requête est lourdement influencé par les opérations exécutées dans la requête notamment la projection, la sélection et la jointure. Pour ce faire, notre fonction de coût s'intéresse à ces trois opérations. Les paramètres utilisés sont présentés sur le tableau 4.1.

Paramètre	**Signification**
|R|	nombre de pages nécessaires pour stocker la table R
||R||	nombre de n-uplets de la table R
M	taille de la mémoire en nombre de pages
PS	taille de la page système
Nbp(R)	nombre de tuples de la table R par page
sel(.)	facteur de sélectivité d'index ou d'un triplet
TSR	taille d'un n-uplet de la table R
Taux(R)	taux de remplissage de la table R
Nbv(a,R)	nombre de valeurs distinctes de l'attribut a dans la table R
P(I)	coût de parcours d'index I en E/S

TABLE 4.1 – Les paramètres du modèle de coût.

3.2 Template de requêtes

Nous considérons que les requêtes exprimées sur une $\mathcal{BDBO}$ respectant le *template* ci-dessous. Soit *C1*, . . ., *Cn* des classes et *p11*, . . . , *pnn* des propriétés d'une ontologie, les requêtes considérées sont de la forme suivante :
(?id1; type; C1) (?id1; p11; ?val11) $\cdots$ *(?id1; pn1; ?valn1) [FILTER()]*
(?id2; type; C2) (?id2; p12; ?val12) $\cdots$ *(?id2; pn2; ?valn2) [FILTER()]*
. . .
(?idn; type; Cn) (?idn; p1n; ?val1n) $\cdots$ *(?idn; pnn; ?valnn) [FILTER()]*

Exemple 3

La requête permettant d'avoir les étudiants inscrits dans des universités françaises peut être exprimée dans ce temple comme suit:
(?x type Etudiant) (?x estInscrit ?y)
(?y type Université)(?y pays France)

Chaque ligne porte sur une classe et ses propriétés. Ce *template* a été défini pour que les requêtes puissent être exprimées dans la plupart de langages d'interrogation des ontologies comme SPARQL, SeRQL, OntoQl, RQL, etc. Ainsi notre modèle de coût n'est pas seulement applicable à un seul langage.

3.3 Estimation des facteurs de sélectivité

La sélectivité de triplets et la sélectivité des prédicats (de sélection et de jointure) sont essentielles pour estimer la taille des résultats intermédiaires. Elle sont donc des paramètres primordiales pour estimer les coûts des requêtes [131]. Notre démarche consiste à nous appuyer sur les travaux qui se sont intéressés à l'estimation du facteur de sélectivité de patron de triplet [171, 173] et à les affiner pour prendre en compte l'implémentation relationnelle des données pour les $\mathcal{BDBO}$ de *type I* et sur les travaux qui se sont intéressés à l'estimation du facteur de sélectivité de prédicat [175, 131] pour les $\mathcal{BDBO}$ de *type II* et *type III*. Dans ce qui suit, nous présentons la sélectivité de patron de triplet pour des $\mathcal{BDBO}$ de *type I* et la sélectivité de prédicat pour les $\mathcal{BDBO}$ de *type II* et *type III*. Dans notre *template*, tout patron de triplet peut être exprimé en termes de prédicat de sélection sur les $\mathcal{BDBO}$ de *type II* et *type III*.

3.3.1 Sélectivité des patrons de triplet

Définition 1

La sélectivité d'un patron de triplet t notée sel(t) est le nombre de triplets correspondant au patron de triplet divisé par le nombre total de triplets [176, 171].

D'après Stocker et al. [171], elle se calcule suivant la formule :

$$sel(t) = sel(s) * sel(t) * sel(o)$$

où *sel(s)*, *sel(p)* et *sel(o)* sont respectivement la sélectivité du *sujet*, du *prédicat* et de l'*objet* du triplet. Ces sélectivités sont définies comme suit :

- Pour tout composant x qui est variable : *sel(?x)* =1;
- Pour le composant sujet s :

$$sel(s) = \frac{1}{R} \text{ où } R \text{ est le nombre total de ressources dans la table de triplets}$$

- Pour le composant propriété p :

$$sel(p) = \frac{T_p}{T}$$

  où T_p est le nombre total de triplets ayant p comme propriété et T est le nombre total des tuples dans la table de triplets.
- Pour le composant objet o : on utilise un histogramme de type *equal-width* qui présente la distribution des valeurs par propriété (son co-domaine). Le co-domaine de la propriété p est divisé en B classes (p, o_c).

$$sel(o) = \begin{cases} sel(p, o_c) & \text{si } p \text{ est définie} \\ \sum_{p \in P} sel(p_i, o_c) & \text{sinon.} \end{cases}$$

$$sel(p, o_c) = \frac{h_c(p, o_c)}{T_p}$$

où le couple (p, o_c) représente la classe de l'histogramme de la propriété p où se retrouve l'objet o. $h_c(p, o_c)$ est la fréquence de la classe (p, o_c) et T_p est le nombre de triplets ayant p comme propriété.

Si la taille de la $\mathcal{BDBO}$ le permet, on peut garder dans les statistiques de la $\mathcal{BDBO}$ les nombres d'occurrences de "*po*[23]". On peut alors calculer $sel(o)$ comme le nombre de triplets ayant p comme propriété et o comme valeur *(nbrTuple(?s, p, o)*) normalisé par le nombre de triplets ayant p comme propriété (T_p) : $sel(o) = \frac{nbrTuple(?s,p,o)}{T_p}$

La sélectivité du composant o est donc liée au composant p. Neumann et Moerkotte [174] l'ont qualifié de sélectivité conditionnelle.

3.3.2 Sélectivité de jointure de patrons de triplet

Kaoudi et al. [173] ont défini la sélectivité de jointure de deux patrons de triplet conjonctifs *TP1* et *TP2* comme étant le nombre de triplets résultant de la jointure de *TP1* et *TP2* (*joinCard(TP1,TP2)*) normalisé par le nombre total des triplets (T) au carré : $jsel = \frac{joinCard(TP1,TP2)}{T^2}$.

Cette formule n'a pas été adaptée à une implémentation relationnelle des triplets. Elle implique le calcul de la jointure de *TP1* et *TP2*. Etant donné que le résultat d'une exécution d'un patron de triplet est une table, nous choisissons utiliser l'approche relationnelle pour ce calcul [154]. Supposons que *R1* et *R2* soient des relations correspondant aux résultats des patrons de triplet *TP1* et *TP2*. Soit $card(R1 \bowtie R2)$ la taille de la jointure de *R1* et *R2*.

$$card(R1 \bowtie R2) = \frac{||R1|| * ||R2||}{max(V(R1, ?x), V(R2, ?x))}$$

où $||R1||$ et $||R2||$ sont le nombre d'instances de *R1* et de *R2* respectivement, *?x* est une variable partagée par *TP1* et *TP2*, et *V(Ri,?x)* est la taille du domaine des attributs *?x* de la relation *Ri*. En d'autres termes, $||Ri||$ est le nombre de triplets répondant au patron de triplet *TPi* ($||Ri|| = sel(TP) * T$, où T est le nombre total de triplets), et *V(Ri,?x)* est le nombre des valeurs distinctes que la variable *?x* peut prendre dans la solution de *TPi*. Pour le calcul de (*V(Ri, ?x)*), on peut distinguer deux cas. Si *TPi* a une variable, alors *V(Ri,?x)* est égal au nombre de solutions de la variable *?x*, c'est-à-dire $V(Ri, ?x) = ||Ri||$. Pour le second cas où *TPi* a deux variables, pour connaître la taille du domaine des variables apparaissant dans la requête, on utilise un histogramme qui, pour chaque patron de triplet, donne le nombre des valeurs distinctes de chaque composant ainsi que le nombre des valeurs distinctes des combinaisons de deux composants [173].

Comme l'estimation de la sélectivité de patron de triplet pour des $\mathcal{BDBO}$ de *type II et III* se ramène à l'estimation de sélectivité prédicat comme dans les BD relationnelles, nous présentons à la section suivante la sélectivité des prédicats.

23. triplet de la forme(?s, p, o)

3.3.3 Sélectivité des prédicats

Plusieurs travaux de recherche se sont intéressés à l'estimation de la sélectivité des prédicats [131]. La plupart d'entre eux admet les deux hypothèses suivantes :

1. uniformité de la distribution des données
2. indépendance entre les attributs de chaque relation.

Définition 2

La sélectivité d'un prédicat est le cœfficient représentant le nombre d'objet sélectionnés conformément à ce prédicat rapporté au nombre d'objets total d'une table. Si la sélection vaut 1, tous les objets sont sélectionnés. Si elle vaut 0, aucun objet n'est sélectionné.

L'estimation de la sélectivité des prédicats de sélection se fait comme suit : soient A_i et A_j deux attributs d'une table R. La sélectivité se fait selon le prédicat par une des formules suivantes :

$$sel(A_i = valeur) = \frac{1}{card(\pi_{A_i}(R))}$$

$$sel(A_i > valeur) = \frac{max(A_i) - valeur}{max(A_i) - min(A_i)}$$

$$sel(A_i < valeur) = \frac{valeur - max(A_i)}{max(A_i) - min(A_i)}$$

$$sel(p(A_i) \wedge p(A_j)) = sel(p(A_i)) * sel(p(A_j))$$

$$sel(p(A_i) \vee p(A_j)) = sel(p(A_i)) + sel(p(A_j)) - sel(p(A_i)) * sel(p(A_j))$$

$$sel(A_i \in \{valeurs\}) = sel(A_i = valeur) * card(\{valeurs\})$$

L'estimation de la sélectivité des prédicats de jointure de *R1* et *R2* (*jsel*) est donnée par la formule [154] :

$$jsel = \frac{||R1 \bowtie R2||}{||R1|| * ||R2||}$$

Certains travaux ne font aucune hypothèse de la distribution des données [131]. Dans ce cas une approche basée sur les histogrammes est suggérée.

3.3.4 Sélectivité des prédicats complexes

Un prédicat complexe est un prédicat composé de plusieurs prédicats reliés par des opérateurs de conjonction (ET) et de disjonction (OU). Grâce à la forme normale, la sélectivité de prédicats complexes se calcule de proche en proche. Pour les prédicats conjonctifs, elle se calcule selon la formule :

$$sel(pred_1\ ET\ pred_2) = sel(pred_1) * sel(pred_2|pred_1)$$

où $pred_1$ et $pred_2$ sont de prédicats et $sel(pred_2|pred_1)$ est la sélectivité de $pred_2$ sachant $pred_1$. Si les deux prédicats sont indépendants, $sel(pred_2|pred_1) = sel(pred_2)$. C'est toujours l'hypothèse envisagée. Aucun système n'est en effet capable de prendre en compte la corrélation entre les prédicats.

Et pour les prédicats disjonctifs, on utilise la formule :

$$sel(pred_1\ OU\ pred_2) = sel(pred_1) + sel(pred_2) - sel(pred_1\ ET\ pred_2).$$

Si $pred_1$ et $pred_2$ sont mutuellement exclusifs, $sel(pred_1\ OU\ pred_2) = 0$.

3.4 Coût d'une sélection et d'une projection

Une sélection dans notre *template* de requêtes est une requête qui porte sur une seule classe. Elle est de la forme :

(?id; type; C) (?id; p1; ?val1) ... (?id; pn; ?valn) [FILTER()].

Nous distinguons des sélections *mono-triplet* qui sont des requêtes constituées d'un seul motif de triplet des sélections *pluri-triplets* constituées de plus d'un motif de triplets portant tous sur une seule classe. Seules les sélections *mono-triplet* sont interprétées comme des sélections sur les représentations verticale et binaire, les sélections *pluri-triplets* impliquent des jointures.

Exemple 4

Soit le fragment d'ontologie de la figure 4.1.

1. ***Exemple de requête de sélection mono-triplet** : q = select ?x where (?x inscrit UP). Son équivalent SQL à considérer dans notre modèle de coût est :*
   - *sur un **modèle de stockage vertical**, q = select sujet from TableTriplets where objet=UP and predicat=inscrit;*
   - *sur un **modèle de stockage binaire**, q = select sujet from Inscrit where objet=UP ;*
   - *sur un **modèle de stockage horizontal**, q = select id from Etudiant where inscrit=UP*
2. ***Exemple de requête de sélection pluri-triplets** : q = select ?x, ?y where (?x inscrit UP)(?x nom ?y). Son équivalent SQL à considérer dans notre modèle de coût est :*
   - *Sur un **modèle de stockage vertical**,*
     q = select T1.sujet, T2.objet from TableTriplets T1, TableTriplets T2
     where T1.predicat=inscrit and T1.objet=UP and T1.sujet=T2.sujet;
   - *Sur un **modèle de stockage binaire**, q = select I.sujet, N.objet from Inscrit I, Nom N where I.objet=UP and I.sujet=N.sujet ;*
   - *Sur un **modèle de stockage horizontal**, q = select id,nom from Etudiant where inscrit=UP*
3. *Exemple de requête de jointure : q = select ?x, ?y, ?a where (?x inscrit ?y)(?y adresse ?a) Son équivalent SQL à considérer dans notre modèle de coût est :*
   - *Sur un **modèle de stockage vertical**, q = select T1.sujet, T1.objet, T2.objet from*

TableTriplets T1, TableTriplets T2 where T1.sujet=T2.sujet and T1.predicat=inscrit T2.predicat=adresse;

- *Sur un* ***modèle de stockage binaire****, q = select I.sujet, I.objet, A.objet from Inscrit I, Adresse A where I.sujet=A.sujet ;*
- *Sur un* ***modèle de stockage horizontal****, q = select E.id, E.inscrit, U.adresse from Etudiant E, Université U where E.inscrit=U.id*

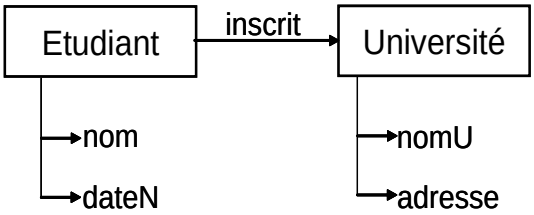


FIGURE 4.1 – Exemple de schéma d'une ontologie.

Nous définissons la fonction de coût comme celle des bases de données relationnelles. Pour la sélection *mono-triplet*, notre fonction est égale au nombre de pages de la table impliquée dans la requête :

- dans l'approche verticale, on a : $Cout(q, bdbo) = |T|$ où T est la table de triplets. Si des index sont définis sur les triplets (nous pensons aux six index jugés meilleurs sur la table de triplets [172], [177] : $cout(q, bdbo) = P(index) + sel(t) * |T|$, où *P(index)* est le coût de parcours d'index et *sel(t)* est la sélectivité du triplet *t*.
- dans l'approche binaire, la sélection porte sur la table de propriété du triplet : $Cout(q, bdbo) = |Tp|$ où *Tp* est la table de propriété du triplet de la requête. Avec un index défini sur le prédicat de sélection, $cout(q, bdbo) = P(index) + sel * |Tp|$, *sel* est la sélectivité de l'index.
- dans l'approche horizontale, la sélection porte sur la ou les tables relatives à des classes qui sont *domaine* de la propriété du triplet de la requête :

$$Cout(q, bdbo) = \sum_{Tcp \in domaine(p)} (|Tcp|)$$

où *Tcp* sont des tables correspondantes aux classes de la propriété *p* du triplet de la requête. S'il y a un index défini sur le prédicat de sélection,

$$cout(q, bdbo) = \sum_{Tcp \in domaine(p)} (P(index) + sel * |Tcp|)$$

où *sel* est la sélectivité de l'index.

Pour la sélection *pluri-triplets*, les requêtes sont traduites en des jointures dans les approches verticales et binaires. Dans l'approche horizontale, la fonction de coût est la même que celle de la sélection *mono-triplet* puisque cela n'implique pas de jointure.

Une projection est une sélection sans filtre et ayant une liste d'attributs de taille inférieure ou égale au nombre de propriétés de la classe sur laquelle elle est définie. Son coût est le même que celui de la sélection.

3.5 Coût de la jointure

Les jointures sont des requêtes constituées d'au moins deux motifs de triplets. Cependant, dans l'approche horizontale, il faut que les triplets portent sur au moins deux classes de différentes hiérarchies (sinon c'est une *sélection pluri-triplets*). D'abord, pour l'approche verticale, nous remarquons que l'auto-jointure de la table des triplets peut se faire de deux manières :

1. comme jointure naïve de deux tables relationnelles classiques c'est-à-dire, un produit cartésien suivi d'une sélection. Nous appelons cela, " *jointure à la BDR* ".
2. en sens inverse de la " *jointure à la BDR* " c'est-à-dire des sélections suivies d'une jointure classique. Nous appelons cela, " *jointure retardée* " Nous esquissons un algorithme de cette deuxième approche ci-après.

Algorithm 1 Algorithme de l'auto-jointure de la table de triplets

```
1: // Algorithme de Jointure retardée
2: Entrée : T table de triplet, t1 et t2 deux motifs de triplets
3: Sortie : rel(t3) relation résultat
4: Début
5:     Faire une sélection suivant t1 sur T et mettre le résultat dans rel(t1)
6:     Faire une sélection suivant t2 sur T et mettre le résultat dans rel(t2)
7:     Faire une jointure classique de rel(t1) et rel(t2) et mettre le résultat dans rel(t3)
8: return rel(t3)
9: Fin
```

Le coût de la jointure de *rel(t1)* et *rel(t2)* dépend de l'algorithme de jointure utilisé (boucle imbriquée, merge-join, Hash-join). On peut améliorer cet algorithme dans un but d'optimisation de requêtes, en considérant la sélectivité de triplets comme dans [176] ou la "'*restrictivité*"' de triplets (nombre de variables dans les patrons de triplet) comme l'ont fait Groppe et al. [178].

Nous considérons distinctement ces deux approches de l'auto-jointure de la table de triplets.

Notre fonction de coût dans les autres approches est celle utilisée dans le BDR. Les coûts de jointure par boucle imbriquée (nested loop) et par hachage sont présentés dans le tableau 4.2. On rappelle que le coût de jointure par hachage de R_1 et R_2 est de $3 \times (|R_1| + |R_2|)$ et le coût de leur jointure par boucle imbriquée est $|R_1| + |R_2| * |R_1|/M$ où M est la taille de la mémoire en nombre de pages.

4 Validation du modèle de coût

Pour valider notre fonction de coût, nous avons considéré les requêtes de différents types. Nous présentons ici un résultat portant sur trois requêtes du benchmark LUBM [30] (requêtes 2, 4 et 6) représentant les types de requêtes présentés. La requête 2 est une requête de jointure,

Algorithme Nested loop sans index	*jointure retardée*	*jointure à la BDR* [154, 179]
Approche verticale join(T,T), T est la table de triplets	$2*\|T\|+2\|t1\|+\|t2\|*(1+\|t1\|/M)$	$\|T\|+\|T\|*\|T\|/M$
Approche binaire join(Tp1,Tp2), Tpi Table de propriété pi		$\|Tp1\|+\|Tp1\|/M*\|Tp2\|$
Approche horizontale join(Tcp1,Tcp2), Tcpi tables des classes		$\sum_{Tcp\in dom(p)}(\|Tcp1\|+\|Tcp1\|/M*\|Tcp2\|)$
Algorithme Nested loop avec index	*jointure retardée*	*jointure à la BDR*
Approche verticale join(T,T), T est la table de triplets	$\|T\|(sel(t1)+sel(t2))+P(I1)+P(I2)+\|t1\|+\|t1\|*\|t2\|/M$, *sel(t)* est la sélectivité des triplets	|T| +|T|*|T|/M
Approche binaire join(Tp1,Tp2), Tpi Table de propriété pi		$\|Tp1\|+\|\|Tp1\|\|*(P(I)+\|\|Tp2\|\|*sel)$, P(I) coût de parcours d'index I et *sel* la sélectivité de la valeur de recherche dans Tp2
Approche horizontale join(Tcp1,Tcp2), Tcpi tables des classes		$\sum_{Tcp\in dom(p)}(\|\|Tcp1\|\|+\|\|Tcp1\|\|*(P(I)+\|\|Tcp2\|\|*sel))$
Algorithme Nested loop sans index	*jointure retardée*	*jointure à la BDR*
Approche verticale join(T,T), T est la table de triplets	$2*\|T\|+4*(\|t1\|+\|t2\|)$	$6*\|T\|$
Approche binaire join(Tp1,Tp2), Tpi Table de propriété pi		$3*(\|Tp1\|+\|Tp2\|)$
Approche horizontale join(Tcp1,Tcp2), Tcpi tables des classes		$\sum_{Tcp\in dom(p)}3(\|Tcp1\|+\|Tcp2\|)$

TABLE 4.2 – Coûts de jointures. (dom(p) : domaine de p)

la requête 4 est une requête sélection *pluri-triplets* et la requête 6 est un exemple de requêtes de sélection *mono-triplet*. Ces requêtes sont traduites en SQL dans les différentes approches et une évaluation de leur coût est faite. Les statistiques utilisées sont présentées dans le tableau 4.3. Pour des calculs, nous avons supposé le taux de remplissage des tables de 80%, la taille des pages de 8Ko, la taille de champs (*sujet, propriété* et *objet*) de 200 octets chacune et la mémoire centrale de 50.000 pages. Chaque requête est considérée sur les différentes approches. Les coûts calculés de ces requêtes sont consignés dans le tableau 4.4.

Concepts	*Cardinalité*
|| triplets||	100838
||University||	979
||GraduateStudent||	1874
||underGraduateStudent||	5916
||Student ||	0
||Department||	15
|| memberOf||	7790
||subject (distincts) ||	17270
||property (distincts) ||	31
||objects (distincts) ||	14072
||type||	18213
||undergraduateDegreeFrom||	2414
||subOrganizationOf||	239
||Professor||	0
||AssociateProfessor||	176
||AssistantProfessor||	146
||Chair||	0
||VisitingProfessor||	0
||FullProfessor||	125
||Dean||	0
||name||	15972
||worksFor||	540
|| emailAddress||	8330
||telephone||	8330

TABLE 4.3 – Statistiques des instances ontologiques utilisées pour l'application du modèle de coût.

Comme attendu, les résultats montrent que l'exécution d'une requête d'auto-jointure de la table de triplets avec une *jointure à la BDR* nécessite plus d'entrées-sorties que son exécution effectuant d'abord les sélections suivant les motifs de triplets suivies de jointures (*jointure retardée*).

Requêtes	**Type de requête**	*A.V jointure retardée*	*A.V jointure à la BDR*	**Approche binaire**	**Approche horizontale**
Requête 6 Lubm	Sélection Nomo-triplet	9168	9168	1138	10289
Requête 4 Lubm	Sélection *pluri-triplets*	21308	137520	60864	121
Requête 2 Lubm	Jointure (hash join)	23188 1	65024	195246	18813

TABLE 4.4 – Coûts en termes d'E/S selon le modèle de coût. (A.V: Approche verticale)

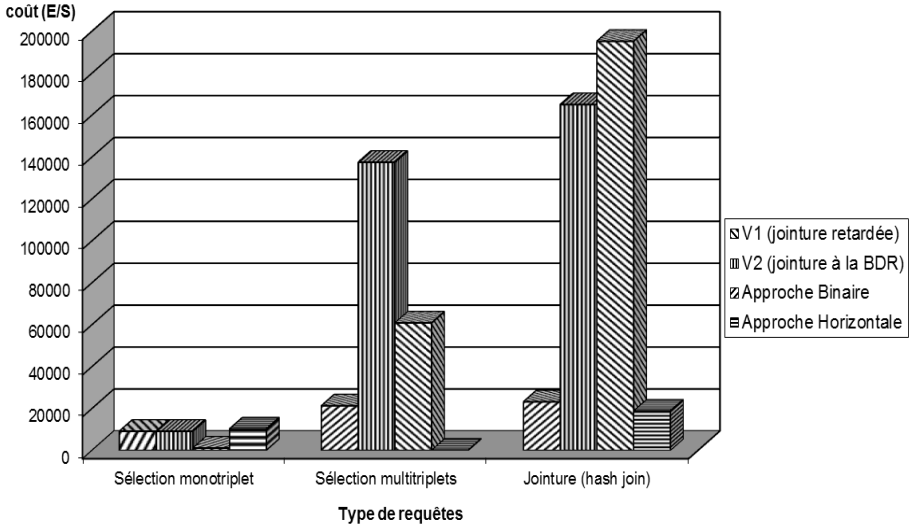


FIGURE 4.2 – Coûts de requêtes fournis par le modèle de coût.

Nous observons sur la figure 4.2 que pour une sélection *mono-triplet*, l'approche binaire réalise moins d'E/S et donc est susceptible d'être plus efficace que les autres approches. Dans notre domaine d'application, l'ingénierie, de telles requêtes sont rares. Dans d'autres types de requêtes (jointure et sélections *pluri-triplets*), l'approche horizontale se montre meilleure suivie de l'approche verticale. Cela confirme partiellement nos résultats expérimentaux (chapitre 3, section 4). Ces résultats sont confirmés par OntoDB qui représente l'approche horizontale et donne de bons temps de réponse, suivie de Oracle ayant une approche verticale et de Sesame utilisant une approche binaire. Par contre, pour certains systèmes expérimentés comme Jena, SOR et DB2RDF, les résultats des évaluations ne confirment pas très clairement ces résultats théoriques. Nous croyons que cela est dû à la procédure de jointure qui peut être différente de celle appliquée dans notre modèle de coût. Les techniques d'optimisation intégrées dans certaines $\mathcal{BDBO}$ notamment les vues matérialisées peuvent en être une raison à ces différences puisque notre modèle de coût ne les intègre pas.

5 Conclusion

Nous nous sommes intéressés dans ce chapitre aux modèles de coût dans les bases de données à base ontologique. Un modèle de coût permet d'estimer le coût d'exécution d'une requête sur une base de données. C'est un composant très utile pour un optimiseur de requêtes qui est un maillon très important pour tout SGBD [180]. Il permet à ce dernier d'évaluer les différents plans d'exécution d'une requête et de choisir le plan de moindre coût. En effet, un SGBD peut évaluer une requête de plusieurs manières (plans d'exécution). Chaque plan d'exécution est constitué d'une suite d'opérations à exécuter sur les tables et sur les résultats intermédiaires pour répondre à la requête. L'optimiseur de requêtes se charge de choisir parmi les plans d'exécution possibles d'une requête, celui dont le coût est le moindre. Le coût de l'exécution des différents plans d'exécution d'une requête peut considérablement varier. Ainsi, il est très important d'avoir un optimiseur de requêtes efficace qui peut identifier le plan optimal - ou le plan proche de l'optimal - et éviter les plans réellement mauvais. Ceci n'est pas une tâche triviale parce que le coût d'un plan peut varier en fonction de facteurs tels que la distribution de données sous-jacente, l'organisation physique des données sur le disque, la taille du buffer (tampon), l'ordre de tri des données, etc. Le meilleur plan pour un cas peut être un mauvais plan pour un autre cas. Pour choisir le meilleur plan pour une requête, un optimiseur de requête énumère un certain nombre de plans candidats selon une stratégie d'exploration de l'espace des plans, en utilisant des techniques algorithmiques telles que la programmation dynamique [181]. La qualité du plan choisi dépend clairement de la précision du modèle de coût. Un modèle de coût inexact peut entraîner le choix d'un mauvais plan pour une requête car l'optimiseur peut incorrectement estimer le coût de ce plan. Estimer le coût des opérateurs d'une requête nécessite une estimation de la sélectivité de ces opérateurs. Pour estimer une sélectivité, il faut la connaissance de la taille des résultats et des tailles des entrées. La taille de la sortie d'un opérateur est déterminée par sa propre sélectivité tandis que les tailles des entrées d'un opérateur sont déterminées par les sélectivités des opérateurs situés en dessous de l'opérateur dans le plan d'exécution de la requête.

Nous avons proposé un modèle de coût pour les $\mathcal{BDBO}$ en prenant en considération la diversité montrée au chapitre 1. Ce modèle de coût est donc fonction de l'architecture et du modèle de stockage utilisé. Pour les architectures de *type II* et *type III*, nous notons une *augmentation* dans le modèle de coût par rapport à l'architecture de *type I* dont le modèle de coût est identique à celui de bases de données relationnelles.

Une mise en application de ce modèle de coût nous a permis de voir qu'il ne se dégage pas un type de $\mathcal{BDBO}$ qui est efficace dans le traitement de tous les types de requêtes. Cela confirme le constat fait au chapitre 3, sur l'étude des performances des $\mathcal{BDBO}$ en termes de temps d'exécution des requêtes.

L'intérêt de notre modèle de coût est de donner au concepteur et à l'administrateur la possibilité de quantifier ses requêtes et éventuellement de choisir le type de $\mathcal{BDBO}$ approprié pour

une charge de requêtes usuelles.

Aussi notre modèle de coût sera-t-il exploité dans des problèmes de conception physique de $\mathcal{BDBO}$ pour le choix de structures d'optimisation. Le prochain chapitre propose des approches de définition et de sélection de vues matérialisées dans de $\mathcal{BDBO}$. Notre modèle de coût aura son application dans l'une de ces approches.

Chapitre 5

La Sélection des Vues Matérialisées dans les $\mathcal{BDBO}$: un Mode d'Emploi

Sommaire

1 Introduction

Dans les chapitres précédents, nous avons étudié les $\mathcal{BDBO}$ et la conception physique d'une manière séparée. L'objectif de ce chapitre est de les confronter pour voir l'impact de chacun sur l'autre. Les $\mathcal{BDBO}$ ont apporté deux caractéristiques principales : la présence de l'ontologie dans le SGBD et la diversité concernant les modèles de stockage et l'architecture du SGBD cible. La conception physique est l'une des phases clés du cycle de vie de la conception de bases de données, durant laquelle des structures d'optimisation sont sélectionnées. Pour confronter les deux mondes, nous considérons une seule structure d'optimisation à savoir les vues matérialisées. Rappelons qu'elles sont des structures redondantes du fait qu'elles nécessitent un espace de stockage et un coût de maintenance. Le processus de sélection des vues matérialisées est souvent réalisé par des approches dirigées par des modèles de coût. Notre choix de cette structure est principalement motivé par le fait qu'une vue matérialisée peut être indexée [128] ou partitionnée [89, 129]. Par ailleurs, les vues matérialisées optimisent un large spectre d'opérations (sélection, projection, jointure et agrégation), contrairement aux autres structures qui ne sont efficaces que pour certaines catégories de requêtes.

Le problème de sélection des vues matérialisées dans les $\mathcal{BDBO}$ représente un plus grand défi que dans le cas général car plusieurs formalisations de ce problème sont possibles en fonction de la spécificité de chaque $\mathcal{BDBO}$. Pour illustrer cela, considérons les différentes variantes du problème. Soient:

- $\mathcal{A} = \{A_1, A_2, .., A_n\}$ l'ensemble de toutes les architectures possibles;
- $\mathcal{SM} = \{SM_1, SM_2, .., SM_m\}$ l'ensemble des modèles de stockage existants ;
- $Part = \{Part_1, Part_2, .., Part_n\}$ l'ensemble des parties qu'un SGBD peut avoir.

Ainsi, le nombre de possibilités de sélection des vues matérialisées $\mathcal{P}$ est donné par l'équation suivante:

$$\mathcal{P} = card(\mathcal{A}) \times (card(\mathcal{SM}))^{card(Part)} + 1 \tag{1}$$

Le **+ 1** correspond à la sélection sur la base en présence de l'ontologie.

Dans le chapitre 2, nous avons identifié trois principaux modèles de stockage (table de triplet (SM_1), représentation horizontale (SM_2) et représentation binaire (SM_3)) et trois architectures majeures des $\mathcal{BDBO}$ ($\mathcal{BDBO}$ à deux parts, $\mathcal{BDBO}$ à trois parts et $\mathcal{BDBO}$ à quatre parts). Rappelons que le processus de sélection de vues matérialisées nécessite la prise en compte de trois principales parties (card(Part)=3) : méta-schéma ($Part_1$), ontologie ($Part_2$) et données ($Part_3$). En se basant sur les architectures et les modèles de stockage identifiés ainsi que les quatre parties qui forment un SGBD, $(3 \times \left(3^3\right))$ variantes du problème doivent être traitées. Pour traiter le problème de sélection de vues matérialisées dans notre contexte, trois approches sont possibles: (i) *la sélection basée sur l'ontologie et les requêtes*. Cette approche révolutionne les habitudes des concepteurs et des administrateurs, car la tâche de la sélection est remontée vers la phase conceptuelle. Cette sélection ne prend pas en compte l'implémentation physique de la

base de données. (ii) une sélection imposée, qui suppose l'existence préalable d'une $\mathcal{BDBO}$. Cette sélection est similaire à celle faite dans le contexte des entrepôts de données. Cette approche a été largement suivie par les chercheurs pour sélectionner des vues matérialisées dans les $\mathcal{BDBO}$ [182, 142]. Cette approche suppose que la plateforme de déploiement préexiste, en conséquence, elle ne peut pas être utilisée dans le cadre de simulation. Par simulation nous entendons le cas où une entreprise cherche un SGBD pour ses applications et souhaite faire appel à un simulateur pour choisir sa plateforme de déploiement. Pour répondre à ce besoin, une troisième approche existe, appelée *approche de simulation*, dans laquelle le processus de sélection des vues matérialisées considère la diversité des $\mathcal{BDBO}$.

Dans ce chapitre, nous détaillons deux approches : approche conceptuelle et approche simulée. Pour chaque approche, une formalisation de son problème est donnée. Nous commençons par donner quelques définitions et concepts afin de faciliter la présentation de l'ensemble des contributions.

2 Préliminaires

Définition 3 (*Triplet RDF*)

Un triplet RDF est un 3-uplet (sujet, prédicat, objet) $\in (U \cup B) \times U \times (U \cup B \cup L)$ *où* U *est un ensemble de'URI,* B *est un ensemble de nœuds anonymes et* L *est un ensemble de littéraux (string, int, ...).*

En d'autres termes, un triplet est une combinaison de trois éléments *sujet*, *prédicat* et *objet* écrite sous la forme : (*sujet*, *prédicat*, *objet*). C'est une déclaration exprimant une connaissance dans une ontologie.

Définition 4 (*Patron de triplet*)

Un patron de triplet est un 3-uplet (sujet, prédicat, objet) $\in (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$ *où* U *est un ensemble d'URI,* B *est un ensemble de nœuds anonymes,* L *est un ensemble de littéraux (string, int, ...) et* V *un ensemble de variables.*

Autrement dit, un patron de triplet est un triplet dont certains éléments sont des variables. Les patrons de triplets permettent d'exprimer de requêtes sur des bases de données sémantiques.

Définition 5 (*Requête SPARQL*)

Une requête SPARQL est une suite de conjonctions des patrons de triplet. Elle est considérée comme une suite de jointures entre les relations résultant de l'exécution des patrons de triplets (si aucune variable n'est partagée par les patrons de triplets, on a des produits cartésiens).

Il faut noter qu'une requête telle que définie est appelée aussi *patrons de requête* ou encore *patron élémentaire de graphe* (*Basic graph patterns*). Nous utiliserons de manière indifférente

ces appellations. Dans ce qui suit, nous ne traiterons que des requêtes SPARQL conjonctives sans Filter et sans patron de triplet optionnel. Les patrons de triplets de nos requêtes ne présentent pas non plus de variable à la position du prédicat. Nous précisons que les requêtes SPARQL conjonctives sont celle dont les patrons de requêtes partagent au moins une variable. Nous utilisons le symbole "ˆ" pour désigner les jointures dans des requêtes quand les patrons de triplets sont identifiés par des noms comme t_1, tp_1, tp_2, ...

Définition 6 (*Mapping de variables ou solution mapping*)

Un mapping $v : V \rightarrow T$ d'un ensemble de variables V vers un ensemble $T = B \cup U \cup L$, où B l'ensemble des nœuds anonymes, U l'ensemble de URI et L l'ensemble de littéraux. Par extension, pour chaque patron de triplet t (triplet avec des variables), $v(t)$ est le triplet qu'on obtient en remplaçant chaque variable $v \in var(t)$ par son mapping ; var(t) est l'ensemble des variables du patron de triplet t.

Définition 7 (*Mapping de patron d'instance (Graph Pattern Matching)*)

Un mapping de patron d'instance P est une combinaison d'un mapping d'instances RDF s, et d'un mapping de variable v, tel que $P(x) = v(s(x))$.

Définition 8 (*Solution pour un patron élémentaire de graphe*)

Soit BGP un patron élémentaire de graphe et G un graphe RDF. v est une solution pour BGP sur G quand il existe un mapping de patron d'instance P tel que P(BGP) est un sous-graphe de G et v est une restriction de P aux variables de la requête dans BGP.

Il faut noter qu'un mapping contient des valeurs pour les variables et les nœuds vides alors qu'une solution ne retient que les valeurs des variables. Deux solutions sont dites compatibles si elles ont des valeurs identiques pour les mêmes variables. Les opérateurs de l'algèbre de SPARQL opèrent sur des multi-ensembles Ω (ou des séquences) des mappings de solutions. L'évaluation des *BGP* produit donc un multi-ensemble Ω de mappings de solution v où chaque v est une solution pour le *BGP*. Elle porte sur une expression algébrique de la requête.

2.1 Représentation algébrique d'une requête SPARQL

Comme dans les bases de données relationnelles, chaque requête SPARQL peut être représentée par un arbre d'expression algébrique correspondant, appelé arbre de requête [183, 184, 185]. Un arbre de requête est une structure de données arborescente qui permet de visualiser graphiquement la requête. La racine correspond à la requête (c'est-à-dire, au résultat de la requête si les opérations sont exécutées). Les feuilles de l'arbre correspondent aux tables utilisées dans la requête. Dans la représentation verticale, les feuilles de l'arbre représentent la table des triplets. Dans les représentations binaire et horizontale, elles correspondent aux tables des propriétés et aux tables de classes respectivement. Les nœuds intermédiaires correspondent aux opérations algébriques comme la sélection, la jointure ou le produit cartésien. En fait, cha-

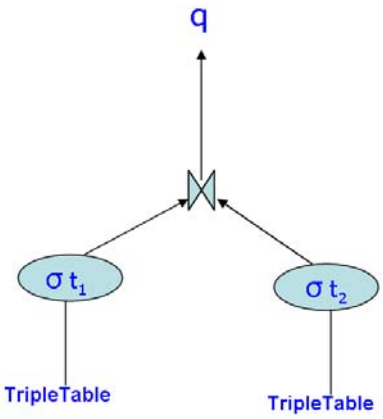


FIGURE 5.1 – Arbre de requête pour une $\mathcal{BDBO}$ Verticale

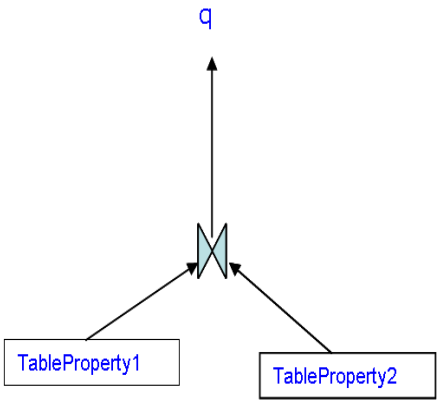


FIGURE 5.2 – Arbre de requête pour une $\mathcal{BDBO}$ binaire

cun de ces nœuds intermédiaires est considéré comme une relation résultant de l'exécution de l'opération sur ses opérandes.

Exemple 5

Soit q une requête constituée de deux triplets t1 et t2 c'est-à-dire, q = t1ˆt2. Les arbres de requêtes de q dans les représentations verticale et binaire sont représentés sur les figures 5.1 et 5.2.

Compte tenu du fait que le langage SPARQL est indépendant des différents types de $\mathcal{BDBO}$ et que les requêtes SPARQL s'appliquent sur n'importe quel type de $\mathcal{BDBO}$, pour présenter nos arbres de requêtes, nous utilisons directement les patrons de triplets. Ces derniers seront donc des feuilles de nos arbres. Cela nous permet de faire une abstraction des différents schémas de $\mathcal{BDBO}$. Ainsi, pour tous les types de $\mathcal{BDBO}$, l'arbre de la requête précédente q (Exemple 5) est donné sur la figure 5.3.

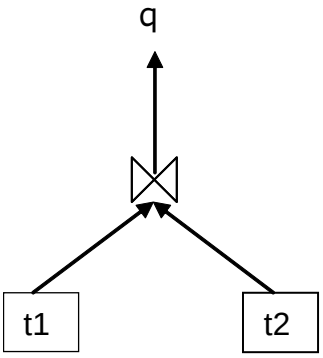


FIGURE 5.3 – Arbre de requête pour tout type $\mathcal{BDBO}$

2.2 Plan unifié générique de requêtes

Dans les entrepôts des données relationnels, la sélection des vues utilise une structure de données appelée *plan unifié de requêtes* (ou MVPP: Multiple Views Processing Plan [186]). Ce plan prend en compte l'interaction entre les requêtes [187]. Dans le monde des bases de données sémantiques, l'interaction est plus présente. Si nous prenons l'exemple d'une $\mathcal{BDBO}$ à base de triplet, toutes les requêtes passent par cette table, ce qui augmente considérablement l'interaction entre l'ensemble des requêtes. Le plan unifié est construit à partir des arbres algébriques individuels de requêtes. Il peut être vu comme un graphe orienté acyclique et étiqueté, dont la structure est définie par : (N, A), où N et A représentent les ensembles de nœuds et d'arcs du graphe. Un nœud est une opération algébrique ou une relation et un arc est un lien entre deux nœuds. La construction de ce plan est une tâche difficile [186]. Dans nos travaux nous utilisons les travaux de thèse d'Ahcène Boukorca, effectué au sein de notre laboratoire pour générer le meilleur plan [188].

Exemple 6

Soit trois requêtes $q0 = t1$, $q1 = t1 \hat{} t2 \hat{} t3$ et $q2 = t1 \hat{} t2 \hat{} t4$. La branche $t1 \hat{} t2$ est identique dans les requêtes q1 et q2. La mise en commun de deux plans passe par la fusion de cette branche commune. Le plan unifié de ces requêtes est présenté dans la figure 5.4.

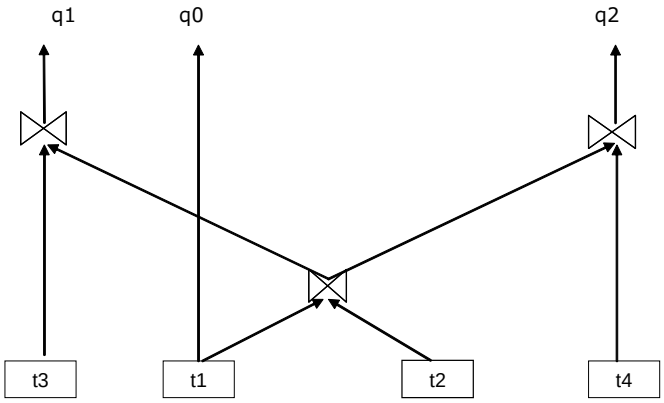


FIGURE 5.4 – Un plan unifié des requêtes SPARQL

Une fois construit, tout nœud intermédiaire peut être candidat à la matérialisation [186]. Les feuilles dans le plan unifié représentant les patrons de triplets résultent des opérations de sélection faites sur l'implémentation sous-jacente de la $\mathcal{BDBO}$. Les nœuds de sélection ($\sigma(.)$) qui expriment clairement la sémantique de l'exécution de patrons de triplets peuvent aussi être représentés sur le plan unifié (figure. 5.5). Pour un patron de triplet, un nœud de sélection est défini comme suit selon le modèle de stockage :

$$\sigma(t) = \begin{cases} \textit{select * from TT where condV} & \text{si } \mathcal{BDBO} \text{ verticale} \\ \textit{select * from TabProp(t) where condB} & \text{si } \mathcal{BDBO} \text{ binaire} \\ \textit{select * from TabClass(t) where condH} & \text{si } \mathcal{BDBO} \text{ horizontale} \end{cases} \quad (2)$$

où *TT*, *TabProp(t)* et *TabClass(t)* représentent respectivement la table de triplets, la table de propriétés et la table de classe relative au patron de triplet *t* ; *condV*, *condB* et *condH* représentent respectivement les prédicats de sélection correspondant au patron de triplet *t* dans le modèle de stockage vertical, binaire et horizontal. Les nœuds de projection ne sont pas représentés dans notre graphe de requêtes. Le plan unifié des requêtes de l'exemple 6 avec des nœuds de sélection est présenté sur la figure 5.5.

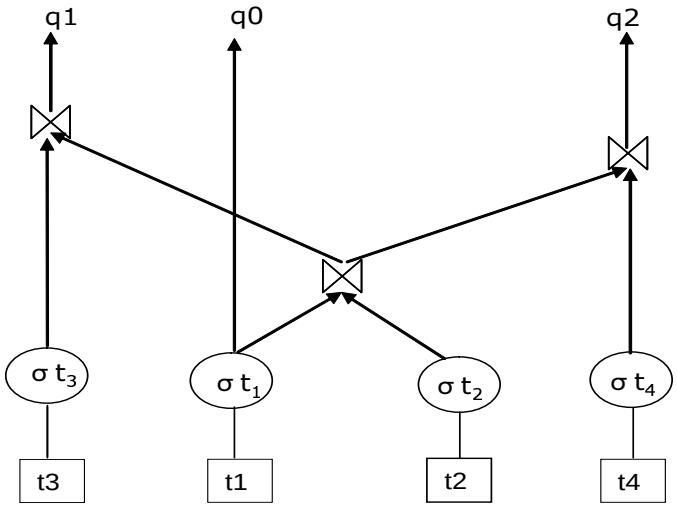


FIGURE 5.5 – Plan unifié des requêtes avec les nœuds de sélection.

Nous avons à présent défini les notions nécessaires pour présenter nos approches de sélection des vues matérialisées.

3 Sélection des vues matérialisées au niveau conceptuel

Formellement, le sélection des vues pour cette approche est définie comme suit. Etant donnés

– une ontologie de domaine utilisée par la $\mathcal{BDBO}$;
– une charge de requêtes SPARQL Q; où chaque requête a une fréquence d'accès;

Le problème de sélection des vues matérialisées consiste à sélectionner un ensemble de vues qui réduid le coût d'exécution de la charge de requêtes.

Cette approche se veut générique. Elle est définie au niveau conceptuel (ontologique) et exige donc le schéma d'ontologie. Elle se base sur les classes de l'ontologie et principalement celles qui sont impliquées dans la charge de requêtes. En plus des classes, elle prend en considération les propriétés impliquées dans les requêtes. Pour rappel, une requête SPARQL est une conjonction des patrons de triplet. Les requêtes SPARQL font souvent appel à plusieurs attributs qui sont des propriétés liées à une même entité, elles tendent donc à être (sinon à contenir)

des requêtes en étoiles (c'est-à-dire une requête dont le graphe est formé d'un nœud central et de ses voisins).

Il est vrai que les requêtes SPARQL ne font pas apparaître clairement les classes visées comme les requêtes SQL le font pour leurs tables, mais on peut identifier ces classes à travers les propriétés dans les patrons de triplets. En fait, tout triplet ou patron de triplet t_i porte sur au moins une classe : la classe du domaine de sa propriété.

L'idée fondamentale est que quand une même variable x apparaît comme *sujet* dans plusieurs triplets, (il existe des jointures entre les relations résultant de l'exécution des patrons de triplet à travers la variable x), en matérialisant la classe de x, ces patrons de triplet ayant x comme sujets, seront exécutés sur cette classe, et ainsi le nombre de ces jointures sera réduit. Cette approche visant donc à éliminer un certain nombre de jointures est susceptible d'accélérer l'exécution des requêtes. Chacune des classes découvertes est donc une vue à matérialiser. Nous présentons ci-dessous notre technique d'identification de ces classes.

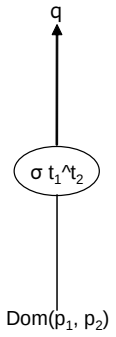


FIGURE 5.6 – t_1 et t_2 sur un seul domaine

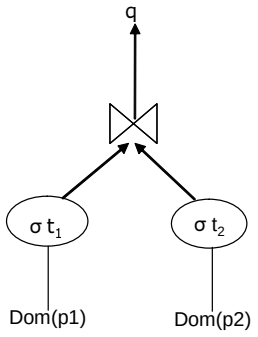


FIGURE 5.7 – t_1 et t_2 sur deux domaines

3.1 Identification des vues

Nous partons de l'arbre algébrique de chaque requête. Pour chaque triplet, nous considérons le domaine de sa propriété (la classe de son *sujet*). Il est évident que si cette classe est matérialisée, la branche de l'arbre construite sur ce patron de triplet aura comme feuille cette classe. Alors, en remplaçant chaque patron de triplet par la classe de sa propriété, et en fusionnant les classes identiques, on obtient un arbre de requêtes basé sur les classes. La fusion de classes se traduit par une union de leurs propriétés. Autrement dit, on regroupe tous les patrons de triplets ayant le même domaine. Soit t_i et t_j deux patrons de triplets et *C(p)*, une classe issue d'un triplet t, ayant p comme propriété, si t_i et t_j ont le même domaine. Le regroupement de t_i et t_j est traduit par la création d'une classe ayant p_i et p_j comme propriétés, $C(p_i, p_j)$. Les figures 5.6 et 5.7 présentent une illustration des arbres de requête basés sur les classes pour la requête $q = t1 \hat{} t2$. Le schéma d'ontologie est très important dans l'identification du domaine d'une propriété car il nous renseigne sur les caractéristiques des propriétés. Il faut noter que le

domaine d'une propriété peut comporter plusieurs classes. On note l'ensemble de ces classes pour une propriété *p*, *Dom(p)*. Dans le processus d'identification de classes, tous les patrons de triplets d'une requête doivent être considérés. Pour chaque variable, les classes retenues sont celles qui sont communes aux ensembles de classes obtenues pour chaque patron de triplet où se trouve la variable. Mais pour éviter d'avoir un grand nombre de ces classes, une prédominance est accordée au constructeur "*rdf:type*". Dans une requête, si une variable *x* est impliquée dans plusieurs triplets comme sujet, et que parmi les propriétés se trouve le constructeur "*rdf:type*", alors la classe de *x* retenue est celle spécifiée par "*rdf:type*".
Au fur et à mesure que l'on identifie les classes, on retient les propriétés définies sur ces classes qui sont présentes dans les requêtes. Nous rappelons que nous travaillons sur les configurations de patrons de triplets où le composant " *propriété* " est une constante ; donc " *propriété* " dévient un attribut des classes découvertes. Par souci de simplification, nous utilisons notre *template* de requêtes défini au chapitre 4 où pour chaque variable, le type est renseigné (c'est-à-dire, il y a un patron de triplet ayant comme propriété " *rdf:type* "). Le processus d'identification des classes est consolidé dans l'algorithme 3.1 qui prend une requête et renvoie ses classes et propriétés. Cet algorithme se sert des matrices d'usages présentées ci-dessous.

Algorithm 2 classIdentifier(q)

1: **Entrée** : q :une requête
2: **Sortie** : C(q) : ensemble de classes impliquées dans q avec leurs propriétés
3: **Début**
4: C(q) = vide
5: pour toute variable var de q faire
6: pour les triplets où var apparait faire
7: si var est sujet de triplet t_i et propriétés p_i définies faire
8: $C(q) = C(q) \cup (\cap_i Dom(p_i))$
9: Ajouter p_i comme propriété à chacune de classe de Dom(p_i)
10: si var est objet de triplet t_i dont s_i est défini faire
11: $C(q) = C(q) \cup classe(s)$ //classe(s) : les classes de s
12: Ajouter p_i comme propriété à chacune des classes de s
13: si var est propriété de triplet t_i et sujet variable faire
14: déterminer p_i
15: $C(q) = C(q) \cup (\cap Dom(p_i))$
16: Ajouter P_i a chacune de classe de Dom(p_i)
17: finPour
18: finPour
19: Retourner C(q)
20: **Fin**

3.2 Matrices d'usage

Partant du schéma d'ontologie, de la charge de requêtes et des résultats de l'identification, nous avons défini deux matrices d'usage : Matrice d'usage des classes (MUC) et matrice d'usage des propriétés (MUP). La matrice d'usage de classe prend en ligne des requêtes et en colonne des classes. Elle nous indique si une requête donnée utilise une classe donnée ou non. La matrice d'usage de propriété nous renseigne sur l'utilisation de chaque propriété par les requêtes. Ces deux matrices sont définies par les équations 3 et 4:

$$MUC\,[i,j] = \begin{cases} 1 & \text{si la requête } i \text{ utilise la classe } j \\ 0 & \text{sinon} \end{cases} \tag{3}$$

$$MUP\,[i,j] = \begin{cases} 1 & \text{si la requête } i \text{ utilise la propriété } j \\ 0 & \text{sinon} \end{cases} \tag{4}$$

Ces deux matrices permettent au travers des algorithmes suivants de définir toutes les vues pour une charge de requêtes donnée. L'algorithme *ViewDiscover* prend un schéma d'ontologie et une charge de requêtes et renvoie la liste des vues à matérialiser. Pour chaque requête, la méthode *ClassIdentifier* est appelée pour identifier les classes et les propriétés. A la fin de l'identification des classes et des propriétés, les matrices d'usage sont créées. Pour chaque classe de l'ontologie, s'il existe une requête qui utilise cette classe, celle-ci est prise comme une vue et on lui associe ses propriétés grâce à la méthode *findProperties*. Cette dernière considère toutes les propriétés liées à la classe traitée (notamment par la relation "*RDFS:Domain*"). Pour chaque classe, s'il y a une requête *i* qui utilise cette propriété *p* (c'est-à-dire, *MUP[i,q]=1*), alors cette propriété est ajoutée comme un attribut à la vue.

Algorithm 3 ViewDiscover

1: **Entrée** : Q : charge de requêtes
2: O : schéma d'ontologie
3: **Sortie** : V : liste de vues matérialisées
4: **Début**
5: $V = \emptyset$
6: Pour $q \in Q$ faire
7: classIdentifier(q)
8: finPour
9: Créer MUC et MUP
10: pour $c \in O$ // Classe ontologique
11: s'il existe $q \in Q$ tel $MUC[q, c] = 1$ alors
12: findProperties(c, MUP)
13: $V = V \cup c$
14: finsi
15: finpour
16: Retourner V
17: **Fin**

Algorithm 4 findProperties

1: **Entrée** : c : une classe
2: MUP : matrice d'usage des propriétés
3: **Sortie** : L : liste de propriétés de c
4: **Début**
5: $L = \emptyset$
6: Pour $p \in$ {propriété de c} faire
7: s'il existe $q \in Q$ tel $MUP[q, p] = 1$ alors
8: $L = L \cup \{c\}$
9: finsi
10: finpour
11: Retourner L
12: **Fin**

La figure 5.8 résume les différentes étapes de mise en oeuvre de l'approche conceptuelle. Cette dernière est une approche basée sur les règles (*rule-based approach*, approche $\mathcal{ODTP}$), cf. chapitre 2) et la règle utilisée est l'*usage des classes et des propriétés* par les requêtes définies au niveau conceptuel. Rappelons que dans les années 90, quelques efforts ont été établis pour définir des langage de requêtes au niveau conceptuel, on peut citer le cas du langage *ConQuer* [189].

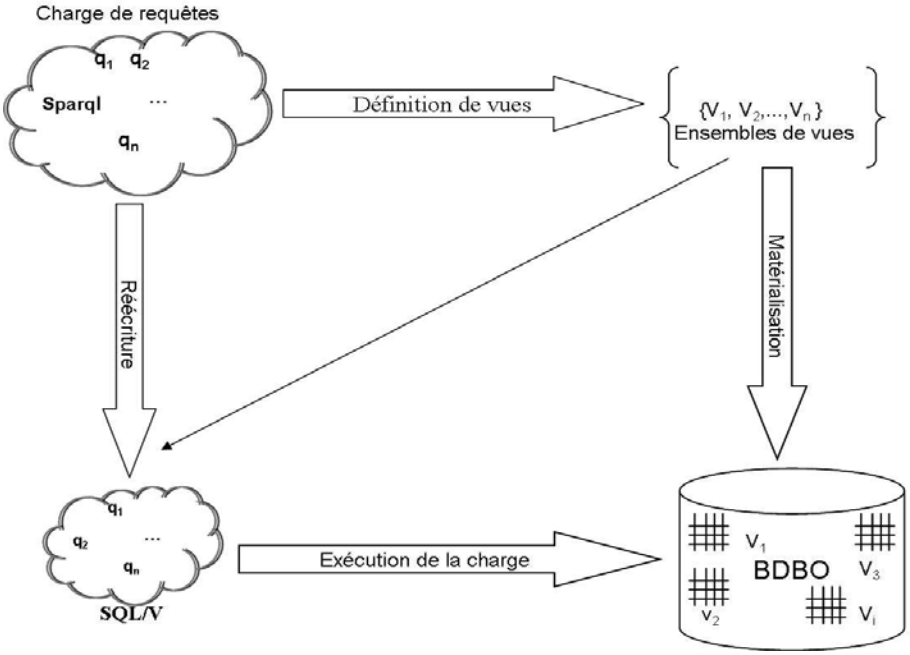


FIGURE 5.8 – Les étapes de mise en oeuvre de l'approche conceptuelle

Exemple 7

Soit Q une charge composée des deux requêtes suivantes :
Q1 : (?x rdf : type University)(?x name ? uname)
Q2 : (?y rdf:type Professor)(?y workFor ?x)(?x rdf : type University)(?x name ? Uname)

***Identification des classes et des propriétés** :*
*Le triplet (?x rdf : type University) est typé **University** alors la classe de la variable x est University. dom(name) ={University,Professor}. Mais pour la variable x, dom(name) = University car on prend l'intersection des ensembles de classes découverts.*
Le triplet (?y rdf:type Professor) ⇒ la classe de la variable y est Professor –> dom(workFor) = Professor. Pour la variable y, dom(name) = Professor.

***Matrice d'usage :** Les matrices d'usage sont présentées sur les tableaux suivants :*

Matrice d'usage des classe		
Requêtes	University	Professor
Q1	1	0
Q2	1	1

Matrice d'usage des Propriétés			
Requêtes	type	workFor	name
Q1	1	0	1
Q2	1	1	1

***Vues matérialisées** : on trouve deux vues matérialisées qui sont :*
- *University((rdfId, name) et ;*
- *Professor(RdfId, workFor, name).*

4 Approche imposée

Nous commençons par présenter la formalisation du problème. Etant donné :

- une $\mathcal{BDBO}$ ayant une architecture et un modèle de stockage donnés;
- une charge de requêtes SPARQL $\mathcal{Q}$, où chaque requête a une fréquence d'accès;
- une contrainte de stockage $\mathcal{S}$.

Le problème de sélection de vues matérialisées consiste à sélectionner un ensemble de vues qui minimise le coût d'exécution de la charge de requête et respecte la contrainte de stockage $\mathcal{S}$.

Le peu de travaux sur la sélection des vues matérialisées dans le contexte des $\mathcal{BDBO}$ utilisent cette approche. Goasdoué et al. [182] ont proposé une démarche de sélection de vues dans $\mathcal{BDBO}$ qui utilise la table des triplets comme modèle de stockage. Leur démarche est basée sur la notion d'état inspirée du travail de Theodoratos et Sellis [190] sur les vues matérialisées dans les entrepôts de données. Un état est une paire <*V*, *R*> où *V* est un ensemble de vues et *R* un ensemble de réécritures de requêtes sur ces vues. L'objectif est de trouver un état qui minimise le coût d'exécution de requêtes, l'espace de stockage et le coût de maintenance de vues mais sans contrainte de stockage. Ils utilisent une structure multi-graphe comme celle proposée par Ullman [191] pour représenter les requêtes et la technique de décomposition de graphe proposé par Wong et Youssefi [192] enrichie de nouveaux opérateurs. Dans le multi-graphe, chaque patron de triplet est représenté par un nœud. Un arc entre deux nœuds indique la jointure entre ces nœuds. Un arc réflexif c'est-à-dire, qui boucle sur un même nœud, symbolise une opération de sélection. A l'état initial, chaque requête est considérée comme une vue. Le multi-graphe de la charge de requêtes de l'exemple 7 est représenté sur la figure 5.9.

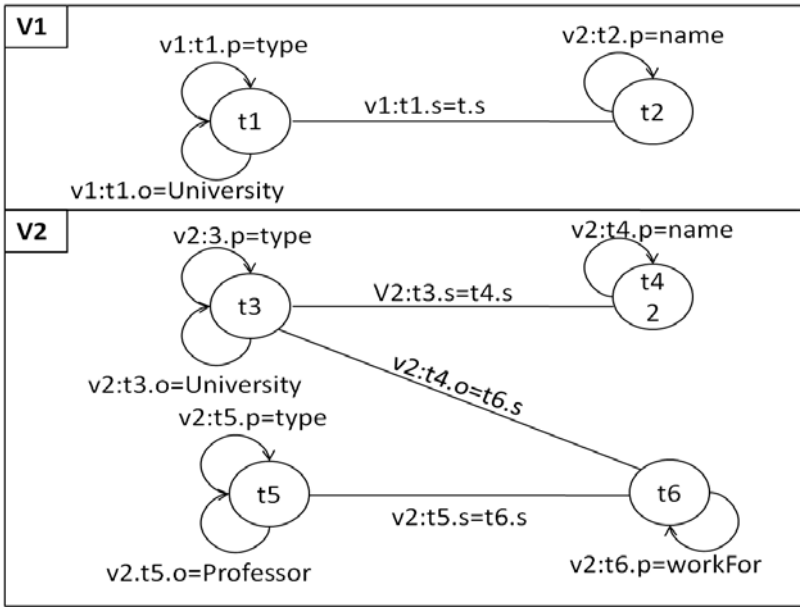


FIGURE 5.9 – Multi-graphe de requêtes

Les principales opérations mises au point pour définir les vues candidates (les états) sont *View Break (VB), Selection Cut (SC), Join Cut (JC)* et *View Fusion (VF).* Les trois premières effacent les prédicats des vues et peuvent diviser une vue en deux, augmentant le nombre de vues et la dernière fusionne des vues, réduisant ainsi le nombre de vues. Par exemple, une opération *Join Cut (JC)* sur la vue *V1* de la figure 5.9 produit les vues *V3* et *V4* présentées sur la figure 5.10.

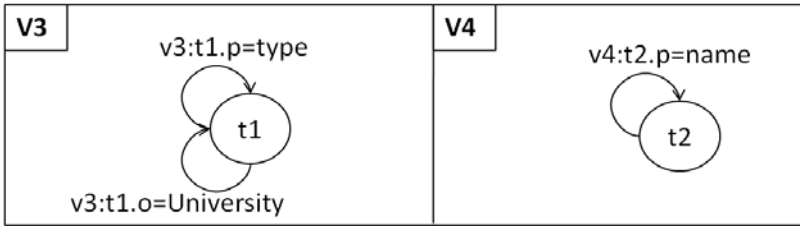


FIGURE 5.10 – Résult de Join Cut sur V1.

A chaque état est associé un coût prenant en compte l'espace mémoire pour toutes les vues, le coût de réécriture des requêtes et le coût de maintenance des vues matérialisées. Cette approche cherche à évaluer les requêtes avec des opérations d'assignation, de sélection, de projection et des boucles et à éviter les opérations de jointure. Les jointures sont effectuées plutôt à travers les boucles.

Castillo et Leser [142] ont proposé une approche pour la sélection des vues sur des données RDF pour une charge de requêtes SPARQL donnée. Cette approche suit la méthode de sélection d'index proposée par Chaudhuri et Narasayya [193] pour les bases de données classiques. Les données RDF utilisées sont représentées dans une table de triplets. L'idée est de découvrir les patrons de triplets partagés pour être utilisés comme index afin d'améliorer les temps de réponses. En considérant l'ensemble de requêtes comme un espace de recherche initial, l'approche de Castillo et Leser [142] étend cet espace en analysant chaque requête de la charge et en dégageant toutes les combinaisons des patrons de triplets connexes d'une certaine longueur (>=k) et en les ajoute à cet espace de recherche appelé espace des index candidats.

Cette approche divise le problème en trois étapes: la génération des ensembles d'index candidats, l'évaluation de ces ensembles et la sélection itérative des index. Les informations d'index sont extraites des patrons de triplet et les ensembles d'index sont générés. Une analyse de leur fréquence est effectuée. La stratégie de "*query containment*" est employée pour déterminer quels sont les patrons de triplet qui sont inclus dans d'autres. Cette approche utilise comme contrainte le nombre maximal de vues (index) que l'on peut créer mais ne prend pas en compte l'espace de stockage des index. Le modèle de coût repose sur un nombre d'occurrences potentielles des index candidats.

5 Approche simulée

La formalisation du problème dans cette approche est quasi similaire à la précédente, la différence étant qu'elle considère plusieurs variantes de $\mathcal{BDBO}$. Pour sélectionner des vues dans chaque variante, nous considérons les étapes suivantes :

1. identification des vues matérialisées candidates ;
2. annotation des candidats par des fonctions de coût qui dépendent des caractéristiques de la base de données cible ;
3. proposition des algorithmes de sélection.

Ces étapes sont détaillées dans les sections suivantes.

5.1 Identification des vues candidates

Bien que notre travail porte sur le problème de sélection des vues matérialisées, il entre aussi dans le cadre de l'optimisation multi-requêtes [24] (Multi-Query Optimisiation - MQO). En effet, les deux problèmes sont liés, surtout quand on utilise une approche consistant à réutiliser certains résultats intermédiaires. Cette partie traite de la construction de l'espace de vues matérialisées suivant une approche qui consiste à réutiliser certains résultats intermédiaires comme dans le cadre de l'optimisation multi-requêtes. Pour créer notre graphe unifié des requêtes, nous nous focalisons sur le partage des nœuds entre les requêtes. Cette démarche favorise la réutilisation des résultats intermédiaires et permet d'optimiser toute la charge de requêtes. Pour une requête donnée, il existe plusieurs plans d'exécution possibles. Cela est dû aux propriétés des opérations algébriques (appelées règles de transformation des arbres [191, 194]). Selon l'ordre donné aux opérations (sélection, jointure, projection) et aux opérandes, on obtient plusieurs plans. Le résultat d'une requête reste le même quel que soit le plan utilisé mais le coût d'une requête peut varier d'un plan à un autre. Le plan optimal d'une requête est celui dont le coût est minimal. Avant de construire notre graphe unifié, nous commençons d'abord par optimiser chaque requête individuellement. Optimiser une requête SPARQL revient à ordonner ses patrons de triplet. Plusieurs techniques sont proposées pour cette tâche [171, 173]. Dans notre travail, nous avons utilisé la sélectivité des patrons de triplet [171, 173]. Les triplets sont ordonnés suivant leurs sélectivités croissantes. Il est aussi intéressant d'ordonner les requêtes suivant un autre critère jugé pertinent (leurs coûts ou fréquences, ou leur nombre de patrons de triplet). Mais on doit garder à l'esprit que la construction des nœuds dépend de l'ordre d'arrivée des requêtes et des patrons de triplet. La construction du graphe unifié passe par les étapes suivantes :

– extraction des patrons de triplet : pour chaque requête de la charge de requêtes, on extrait tous les patrons de triplet qu'on garde dans un ensemble. Chaque triplet n'est stocké qu'une seule fois. Vu que les variables peuvent avoir des noms différents alors qu'elles

24. optimisation d'une charge de requêtes

font référence aux mêmes triplets, on harmonise les variables sur toute la charge de requêtes. Par exemple, *(?x undergraduateFrom ?u)* et *(?p undergraduateFrom ?z)* sont deux patrons de triplet référençant le même ensemble de triplets; on doit identifier que la variable ?*x* correspond à la variable *?p* et la variable *?u* à *?z*.

– extraction des nœuds de sélection : cette opération est la première opération sur les données. Elle représente l'exécution des patrons de triplet sur une $\mathcal{BDBO}$. Les nœuds correspondants sont en fait l'ensemble de triplets répondant à chacun des patrons de triplet.
– extraction de nœuds de jointure : pour chaque requête, nous dégageons les différents nœuds de jointure. Nous rappelons qu'une jointure est une conjonction des deux patrons de triplet partageant au moins une variable. Ces nœuds sont créés sur les nœuds de sélection des patrons de triplet impliqués. Nous avons opté pour les arbres de type *left-join* pour la première requête.
– définition des nœuds de projection que nous considérons comme des résultats de requêtes.

Notre approche de construction du graphe unifié suit celle qui a été proposée et expérimentée dans notre laboratoire [188]. Contrairement à cette dernière, nous ne considérons pas plusieurs liens entre deux nœuds donc nous n'utilisons pas un hypergraphe. Nous obtenons un simple graphe qui est notre plan unifié. Par exemple pour notre charge de requêtes de l'exemple 7, nous obtenons des graphes unifiés dont un exemple est donné sur la figure 5.11. Il est question

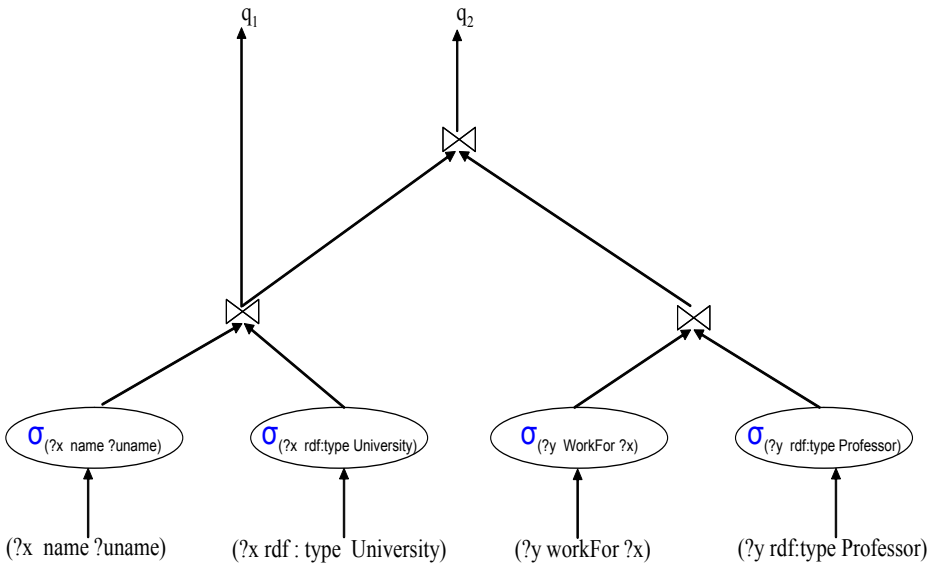


FIGURE 5.11 – Plan unifié des requêtes de l'exemple 7.

de savoir si ce graphe est optimal. Etant donné que notre graphe unifié est dépendant de l'ordre des requêtes, on ordonne les requêtes suivant un critère donné (leurs fréquences, leur coût, leur sélectivité minimale ou une combinaison donnée) et on crée les graphes unifiés en faisant une permutation circulaire jusqu'à ce que la première requête de la liste revienne en tête de liste. Le graphe retenu est celui qui a un coût minimum. Cette alternative ressemble à la méthode *"feasible"* de Yang et al. [195]. Elle est facile à mettre en œuvre mais peut entrainer un grand

nombre de calculs. En effet, pour une charge de N requêtes, il faut créer et comparer N graphes unifiés possibles.

5.1.1 Annotation des candidats par des fonctions coût

Une fois le plan unifié obtenu, nous procédons au processus d'annotation des nœuds par des fonctions coût représentant le coût de construction et le coût d'accès.

5.1.1.1 coût de construction. Le coût de construction d'un nœud est le coût algorithmique du nœud tel que défini au chapitre 4. Ce coût est considéré aussi comme le coût de mise à jour du nœud ou encore le coût de maintenance (on suppose que les mises à jour sont complètes et faites de manière périodique, c'est-à-dire par récréation des nœuds). On note $C\_cout(v)$, le coût de construction du nœud v.

5.1.1.2 coût d'accès. Le coût d'accès d'un nœud est défini comme étant la taille du nœud. Par simplification, on le définit comme étant le nombre de tuples du nœud. On le note $A\_cout(v)$, pour un nœud v. Le calcul de ce coût dépend du type de nœud (nœud de sélection ou nœud de jointure)

Pour un nœud de sélection.

$$A\_cout = \begin{cases} sel(TP) * nbrTuple(TT) \text{ si la } \mathcal{BDBO} \text{ est verticale avec TT la table des triplets} \\ nbrTuple(TabProp(TP)) \text{ si la } \mathcal{BDBO} \text{ est binaire} \\ sel(TP) * nbrTuple(TabClasse(TP)) \text{ si la } \mathcal{BDBO} \text{ est horizontale} \end{cases}$$

où *nbrTuple(T)* est le nombre de tuples de la table *T*, *TabProp(TP)* et *TabClasse(TP)* sont la table de propriété et la table de classe relatives au patron de triplet *TP*, *sel(TP)* est la sélectivité du patron de triplet *TP* si on a une $\mathcal{BDBO}$ verticale et la sélectivité du prédicat correspondant à la traduction du patron de triplet *TP* en langage SQL, si on a une $\mathcal{BDBO}$ binaire et horizontale.

Pour un nœud de jointure.

$$A\_cout = jsel * nbrtuple(TP1) * nbrTuple(TP2) \tag{5}$$

où *nbrTuple(TP1)* et *nbrTuple(TP2)* représente le nombre de tuples dans chaque relation correspondant aux résultats des traitements des patrons de triplet *TP1* et *TP2*, *jsel* est leur sélectivité de jointure.

5.1.1.3 coût de requête. Le coût de requête est fonction des vues matérialisées. Si aucune vue n'est utilisée, le coût de la requête est le coût de l'algorithme utilisé pour calculer la requête. L'algorithme suivant permet de calculer le coût des requêtes en présence d'un ensemble de vues matérialisée (on utilise une jointure par hachage).

Algorithm 5 A_cout

```
1:  Entrée : q : requête
2:               M : ensemble de vues matérialisées
3:  Sortie : cout : coût de la requête q
4:  Début
5:      si (M est vide) retourner C_cout(q);
6:      sinon
7:          si q a des jointures alors
8:              si( dernierejointure de q ∈ M)
9:                  retourner A_cout(dernierejointure)
10:             sinon
11:                 retourner 3*(cout(partieG, M) + cout(partieDroite, M));
12:             finsi
13:         sinon
14:             si (dernièreSelection de q ∈ M)
15:                 retourner A_cout(derniereSelection)
16:             sinon
17:                 retourner C_cout(q)
18:             finsi
19:     finsi
20: Fin
```

5.1.1.4 Coût total de la charge. Le coût total de la charge de requêtes est la somme de coût de chaque requête multiplié par la fréquence de la requête.

$$cout\_total = \sum_{q \in Q} freq(q) * cout(q, M) \tag{6}$$

où *freq(q)* est la fréquence de la requête q et M l'ensemble de vues sélectionnées.

5.1.2 Algorithmes de sélection

Etant donné que chaque nœud est une vue potentielle et que nous disposons des caractéristiques de chaque nœud, il est question de sélectionner les vues qui minimisent le coût total de la charge des requêtes et qui respectent la contrainte de stockage.

Si la contrainte de stockage n'existait pas, l'algorithme de sélection de vues dans un MVPP de Yang et al. [195] nous suffirait pour avoir cet ensemble de vues optimal. Mais vu le fait que cet algorithme ne prend pas en compte la contrainte de stockage, et que le problème de sélection est un problème de sac à dos [196], nous avons utilisé un algorithme glouton et un algorithme génétique pour la sélection de vues.

5.1.2.1 Algorithme génétique. Les algorithmes génétiques [115] sont inspirés de l'évolution des espèces naturelles. Ils reproduisent le modèle d'évolution en vue de trouver une solution optimale à un problème. Ils visent à faire évoluer une population en vue d'améliorer ses individus. Ils considèrent une population et non un individu. Ainsi, ils donnent un ensemble de solution et non une solution. Ces solutions peuvent être différentes mais de qualité équivalente. Notre algorithme utilise la bibliothèque JeneGA développée en Java dédiée aux algorithmes génétiques et disponible sur `sourceforge.net/projects/jenes/` [197]. La particularité de cette dernière est qu'on lui donne le codage représentant un individu d'une population, la fonction fitness pour évaluer la qualité de l'individu, les critères de croisement et de mutation, et comme résultat, elle génère la solution demandée. La figure 5.12 illustre notre démarche (à base d'un algorithme génétique) de résolution de notre problème de sélection de vues matérialisées.

Nous considérons pour chaque vue, l'espace qu'elle occupe et le profit qu'on obtient si elle est la vue seule qui est matérialisée. L'espace occupé est égal au coût d'accès comme évoqué ci-dessus (5.1.1.2). Le profit d'une vue *v* est défini comme la différence entre le coût total de la charge sans vue et le coût total de la charge si la vue *v* est matérialisée.

$$profit(v) = \sum_{q \in Q} freq(q) * cout(q, \emptyset) - \sum_{q \in Q} freq(q) * cout(q, M) \quad (7)$$

où *freq(q)* est la fréquence de la requête *q* et *M={v}*.
Le problème revient à trouver un ensemble de vues dont la somme des espaces occupés soit inférieure ou égale à la contrainte de stockage *S* et qui maximisent les profits. Nous définissons notre chromosome comme un tableau de bits (0 ou 1) (Figure 5.12). Tous les nœuds intermédiaires de notre plan unifié de requêtes sont mappés dans le tableau de bits. Si le bit à une position est à 1, cela veut dire que le nœud correspondant est sélectionné pour la matérialisation. Une solution initiale est générée de manière aléatoire. Celle-ci sera améliorée pour devenir une bonne solution. Notre fonction *fitness* est basée sur le maximum de profit de chromosome. En effet, pour chaque chromosome (solution), la somme des bénéfices de ses nœuds et la somme des espaces occupés par ses nœuds sont calculées. Si la somme des espaces occupées est inférieure à la contrainte de stockage, le chromosome est déclaré valide et peut être amélioré pour devenir une solution, sinon le chromosome est non intéressant et abandonné. Nous avons fait usage des paramètres suivants souvent utilisés dans les algorithmes génétiques pour la sélection des vues [198] : probabilité de croisement (crossover) : 0.8, probabilité de mutation : 0.02, taille de la population : 1000 et le nombre maximum de génération : 200. Cet algorithme nous fournit des solutions assez intéressantes au regard des résultats de nos expérimentations. Hy-

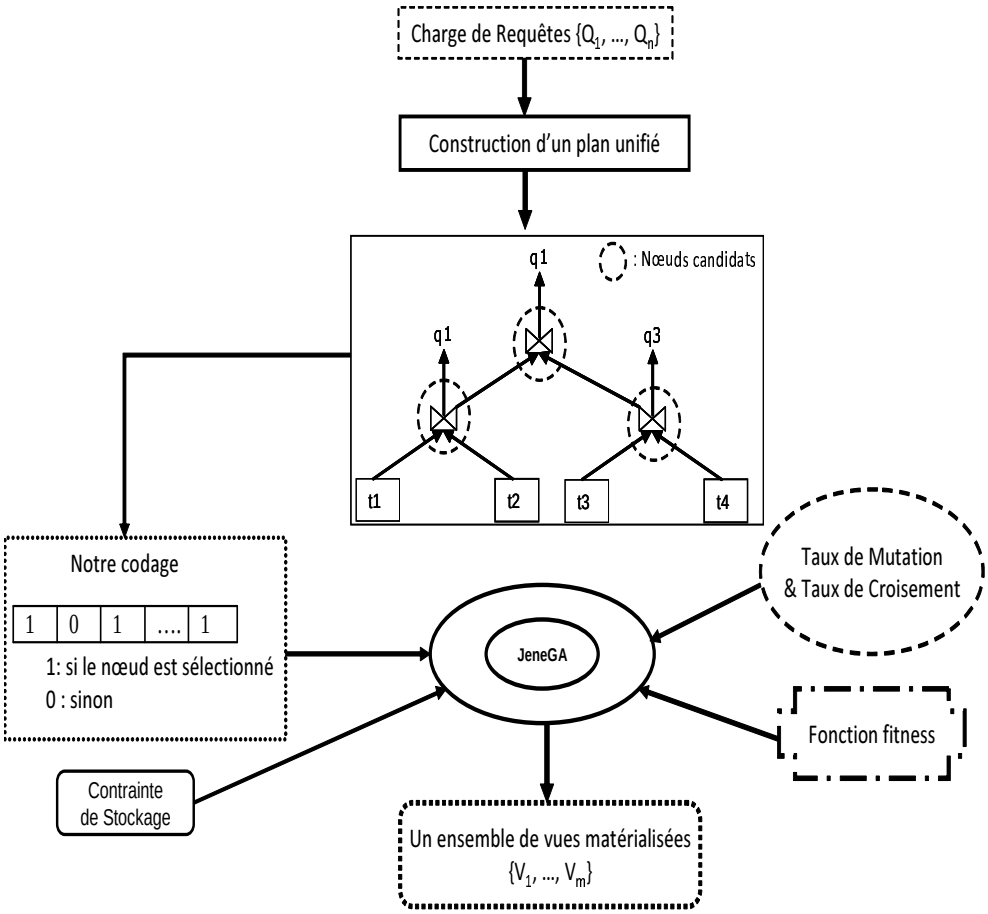


FIGURE 5.12 – Notre démarche de résolution à base d'un algorithme génétique

lock [198] a aussi montré que ces algorithmes sont très bons pour le problème de sélection des vues matérialisées.

5.1.2.2 Algorithme glouton. Notre algorithme glouton utilise la notion de bénéfice par unité d'espace (*BPS*) [199]. Le *BPS* est le rapport entre le bénéfice qu'apporte une vue et l'espace occupé par cette vue. La notion de bénéfice diffère de celle de profit définie ci-dessus. Le bénéfice prend en compte les apports des autres vues déjà retenues alors que le profit d'une vue ne considère que l'apport de cette vue seule.

$$benefice(v, M) = \sum_{q \in Q} freq(q) * cout(q, M) - \sum_{q \in Q} freq(q) * cout(q, M \cup v) \quad (8)$$

L'algorithme calcule de manière itérative les bénéfices de chaque vues et retient la vue ayant le plus grand bénéfice, jusqu'à ce que la contrainte de stockage ne soit plus respectée. Cet algorithme est donné ci-dessous (la fonction *S(v)* calcule l'espace occupé par *v*).

Algorithm 6 greedySelection

```
1:  Entrée : V : Plan unifié
2:  Sortie : M : ensemble des vues matérialisées
3:  Début
4:      M = ∅
5:      Tant que S(M) < S faire
6:          Pour chaque vue v faire
7:              calculer BPS(v,M)
8:          FinPour
9:          choisir v avec BPS maximum
10:         si (S(M) +S(v) <S) alors
11:             M = M ∪ v
12:             V = V − v
13:         Finsi
14:     finTantque
15:     Retourner M
16: Fin
```

NB : Les notions de l'arbre de requête et de plan unifié des requêtes basé sur les patrons de triplet présentées sont valables dans tous les types de $\mathcal{BDBO}$. Pour assurer le caractère générique de cette approche comme souhaité, une abstraction est faite sur les représentations réelles sous-jacentes utilisées dans les $\mathcal{BDBO}$ et les calculs commencent à partir des résultats individuels des patrons de triplet. Dans la mise en œuvre, on doit considérer les coûts de construction de nœuds de sélection suivant le type de $\mathcal{BDBO}$ dont on dispose. Car bien que les tailles des résultats de patrons de triplet individuels sont sensées être les mêmes quel que soit le type de $\mathcal{BDBO}$ utilisée; c'est-à-dire que les résultats d'une requête de sélection sur un même jeu de données doit être les mêmes quel que soit le type de $\mathcal{BDBO}$ utilisée, mais les coûts de calculs ne sont pas les mêmes. Les différentes étapes de cette approche sont illustrées sur la figure 5.13.

6 Expérimentation

6.1 Ensembles de données pour les tests

Nous avons effectué des tests de validation de nos approches sur un ordinateur (PC) de marque DELL doté d'un Processeur Intel(R) Xeon(R) CPU E31225 à 3.10 GHZ, d'une mémoire vive de 4GO et d'un disque dur de 500GO. Nous avons utilisé le benchmark LUBM [30]. Nous avons généré les instances pour des jeux de données contenant 10 et 100 universités notés *Lubm10* et *Lubm100* respectivement. Les nombres d'instances de classes, de propriétés, de

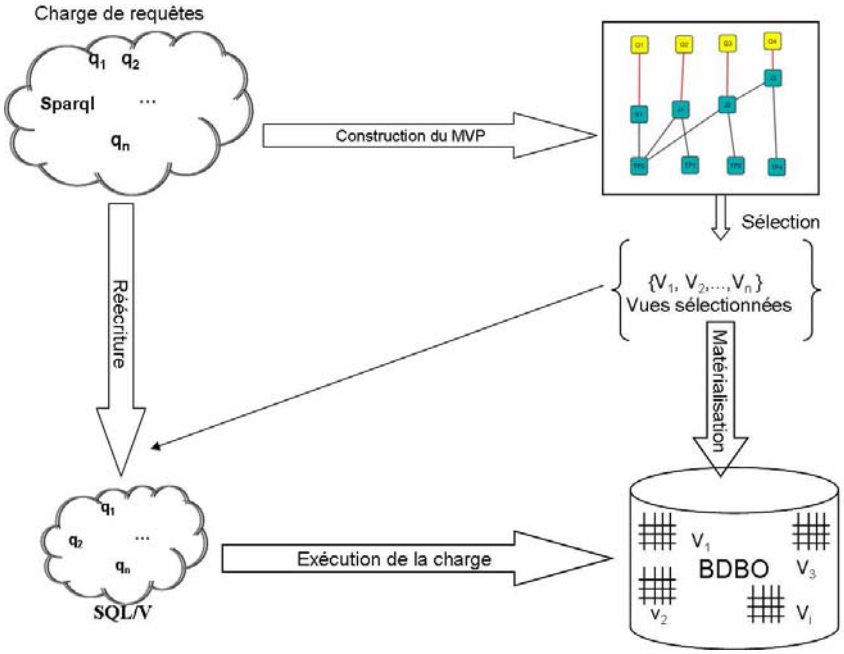


FIGURE 5.13 – Les étapes de mise en oeuvre de l'approche simulée

triplets et la taille des ensembles de données sont consignés dans le tableau 5.1. Nous avons utilisé les requêtes fournies par le benchmark. Pour chaque requête, nous avons fait quatre mesures de temps de traitement et nous avons retenu la moyenne. L'unité de mesure du temps est la milliseconde (ms). Nous avons fait cela pour les requêtes initiales sans les vues matérialisées et pour les requêtes avec les vues matérialisées dans les différentes approches. Nous avons utilisé le SGBD PostgreSQL 8.2. pour toutes les $\mathcal{BDBO}$ utilisées. Nous ne présentons que les résultats où la sélection est faite à l'aide de l'algorithme génétique car c'est celui qui a donné des meilleurs résultats.

Jeux de Données	Nombre d'instances de classe	Nombre d'instances de propriétés	Nombre de triplets
LumB10	1.052.895	1.273.108	1.272.870
LumB100	2779262	11096694	12.674.100

TABLE 5.1 – Ensembles de données générées.

6.2 Différents tests

Nous présentons dans cette partie les différents tests effectués, les résultats et leurs interprétations. Nous appelons *TBA* (Triple-Based Approach), notre approche basée sur les patrons de triplet et *CBA* (Class-Based Approach), notre approche basée sur les classes (approche conceptuelle). Nos expérimentations sont réalisées sur des $\mathcal{BDBO}$ existantes (Jena, Sesame et On-

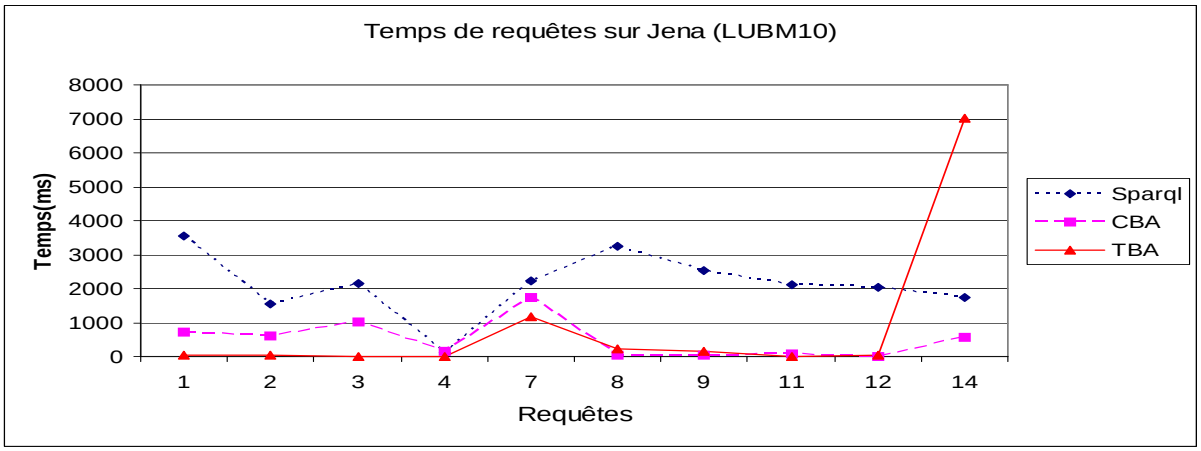


FIGURE 5.14 – Temps de traitement de requêtes sur LUBM10 sur la $\mathcal{BDBO}$ verticale

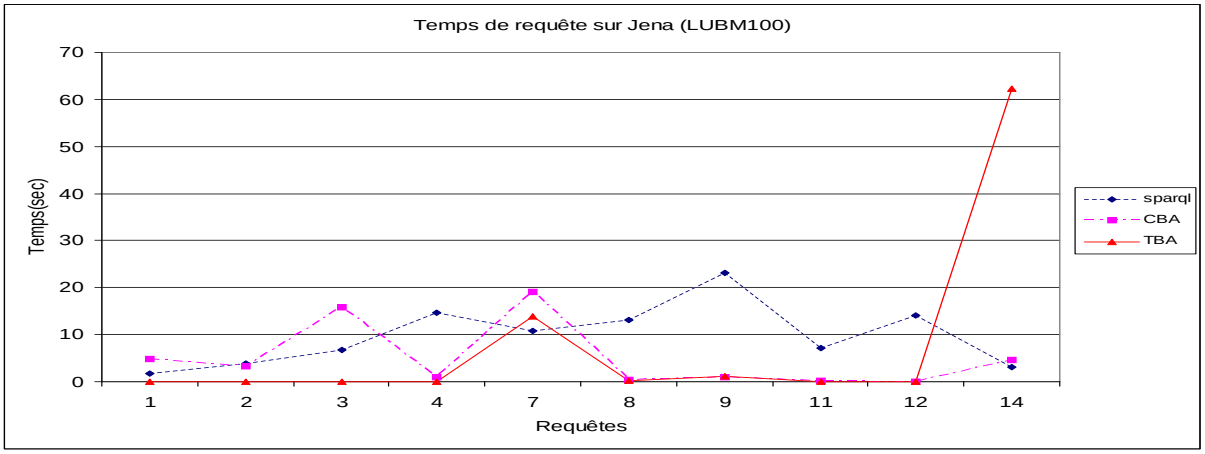


FIGURE 5.15 – Temps de traitement de requêtes sur LUBM100 sur la $\mathcal{BDBO}$ verticale

toDB) et sur une $\mathcal{BDBO}$ créée spécialement pour les tests que nous appelons $\mathcal{BDBO}$ native.

6.2.1 Expérimentations sur les $\mathcal{BDBO}$ existantes

Dans le but d'évaluer nos approches dans des $\mathcal{BDBO}$ existantes, nous avons créé et utilisé une $\mathcal{BDBO}$ verticale (Jena), une $\mathcal{BDBO}$ binaire (Sesame) et une $\mathcal{BDBO}$ horizontale (OntoDB). Considérant une charge de quatorze requêtes constituée des requêtes du benchmark LUBM, et grâce à nos approches, nous avons choisi un ensemble de vues que nous avons créées sur ces $\mathcal{BDBO}$. Nous avons exécuté la charge de requêtes sur les $\mathcal{BDBO}$ sans les vues matérialisées avec un moteur SPARQL (ce résultat est désigné sur les figures par *Sparql*) et avec les vues matérialisées à l'aide d'un moteur SQL. Les résultats sont présentés sur les figures 5.14 à 5.19.

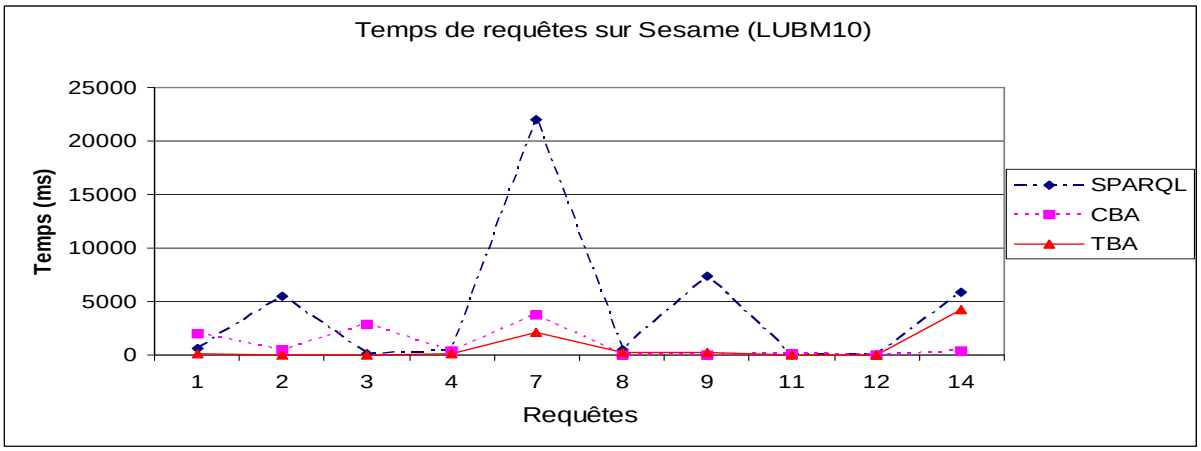


FIGURE 5.16 – Temps de traitement de requêtes sur LUBM10 sur la $\mathcal{BDBO}$ Binaire

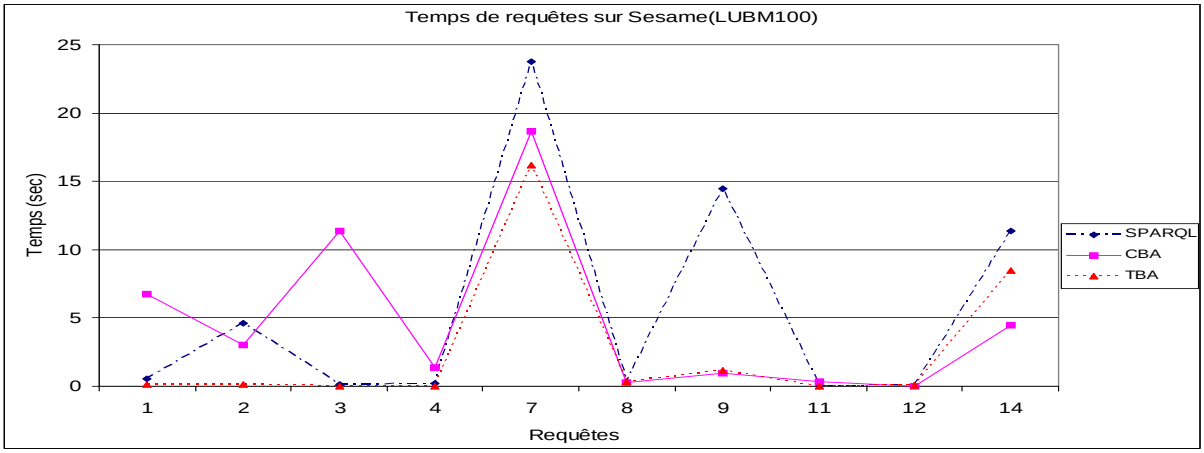


FIGURE 5.17 – Temps de traitement de requêtes sur LUBM100 sur la $\mathcal{BDBO}$ Binaire

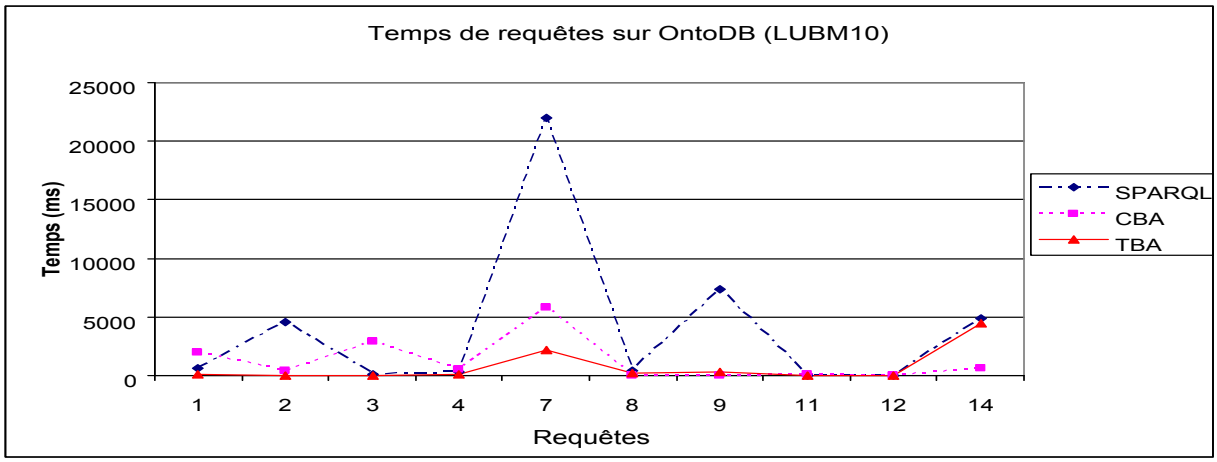


FIGURE 5.18 – Temps de traitement de requêtes sur LUBM10 sur la $\mathcal{BDBO}$ horizontale

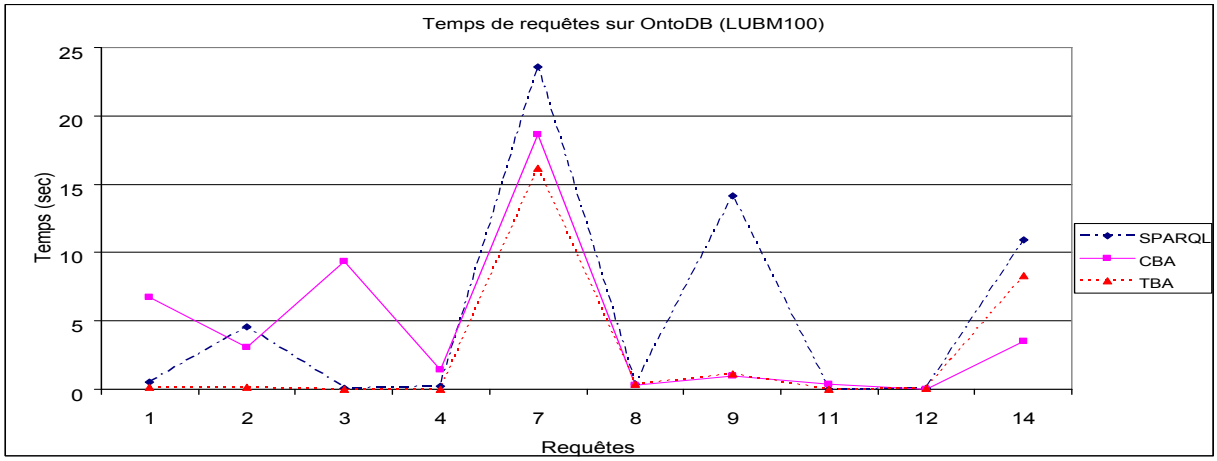


FIGURE 5.19 – Temps de traitement de requêtes sur LUBM100 sur la $\mathcal{BDBO}$ horizontale

6.2.1.1 Interprétation des résultats. TBA fournit des bons résultats pour toutes les requêtes qui utilisent les vues matérialisées. Les requêtes qui n'ont pas de vues appropriées sont exécutées sur les tables de la $\mathcal{BDBO}$. Leurs résultats sont moins bons que si elles sont exécutées avec un moteur SPARQL. En fait, les moteurs SPARQL de Jena et de Sesame utilisent des techniques d'optimisation et sont plus rapides que le moteur SQL *"plat"*. Castillo et Leser [142] ayant utilisé la réécriture SPARQL-SQL ont fait la même remarque. Cela veut dire que l'accès à la table de triplets de jena à travers ARQ [25], par exemple, est plus rapide que l'accès avec un moteur SQL. *CBA* traîne dernière *TBA* et produit des résultats moins intéressants que le moteur SPARQL pour certaines requêtes car certaines vues obtenues sont de tailles importantes. C'est le cas de requêtes 1 et 3 qui portent sur les vues *GraduateStudent* et *Publication* qui sont assez grandes. Cependant, la méthode *CBA* donne un temps cumulé acceptable malgré le fait que pour des requêtes individuelles, les résultats soient mitigés. Nous pouvons améliorer l'approche *TBA*, en utilisant un moteur SPARQL pour des requêtes n'ayant pas de vues appropriées. Cependant si la requête est partiellement couverte, il nous faut combiner les moteurs SQL et SPARQL.

Nous remarquons que l'approche CBA réagit bien sur des jeux de données de taille moyenne mais ses performances se dégradent quand la taille de données devient importante. On peut voir cela en comparant les figures 5.14 et 5.15. En réalité, quand plusieurs propriétés d'une classe sont impliquées dans la charge de requêtes, la vue correspondante est large (c'est-à-dire qu'elle dispose de plusieurs colonnes), cela peut alourdir le chargement de cette vue qui peut nécessiter plusieurs opérations d'entrées-sorties, surtout lorsque toute la vue est balayée.

25. moteur SPARQL implémenté dans Jena

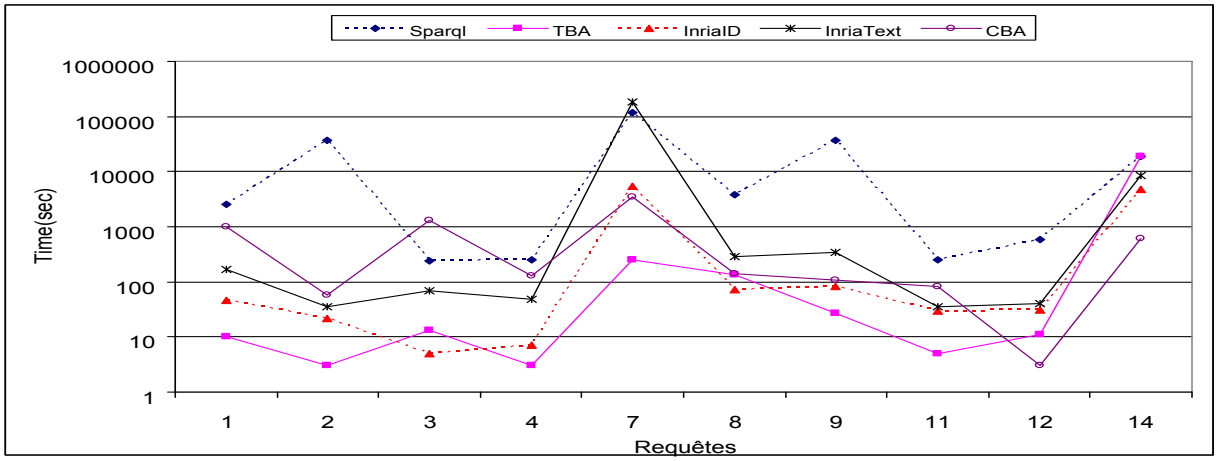


FIGURE 5.20 – Temps de traitement de requêtes sur LUBM10 sur la $\mathcal{BDBO}$ native

6.2.2 Expérimentations sur une $\mathcal{BDBO}$ native et résultats

Les expérimentations sur les $\mathcal{BDBO}$ existantes subissent l'influence des techniques d'optimisation mises en œuvre sous ces systèmes, par exemple l'existence de certains index ou clusters, l'utilisation des dictionnaires, etc. Pour éviter de fausser les résultats des expérimentations par ces influences et pour comparer au plus juste notre travail avec un travail similaire mené par l'INRIA [26][182] qui utilise aussi une $\mathcal{BDBO}$ native sous PostgreSQL, nous avons créé une $\mathcal{BDBO}$ verticale sous PostgreSQL. Sur cette $\mathcal{BDBO}$, nous avons appliqué nos deux approches mais aussi l'approche de l'INRIA [182]. Il faut noter que cette dernière utilise un dictionnaire et fournit des résultats sous forme d'identifiants. Nous appelons *Inria_ID* les résultats de cette approche avec les identifiants et *Inria_Texte* les résultats de la même approche mais sous forme de texte. L'exécution et la mesure des temps de traitement des requêtes dans les différentes approches donnent les résultats présentés sur les figures 5.20 et 5.21.

6.2.2.1 Interprétation des résultats. La plupart de nos requêtes utilisant des vues s'exécutent plus rapidement dans l'approche *Inria_ID* et *TBA*. Dans *TBA*, les requêtes (par exemple la requête 14) ne portant sur aucune vue sélectionnée sont exécutées directement sur la table de triplets. La différence entre les résultats des approches *TBA* et *Inria_ID* s'explique par le fait que cette dernière ne manipule que des valeurs entières et que les opérations y sont plus rapides que dans les autres approches qui manipulent les chaînes de caractères.

L'approche de l'INRIA [182] avec des résultats textuels (*Inria_Texte*) donne un temps moins bon que l'approche *TBA*. En effet, toutes les requêtes nécessitent une ou plusieurs jointures avec la table *dictionnaire* pour le renvoi des valeurs textuelles. Cela nécessite un temps important quand la liste des variables exportées est longue et quand il y a assez de tuples favo-

26. Institut National de Recherche en Informatique et en Automatique

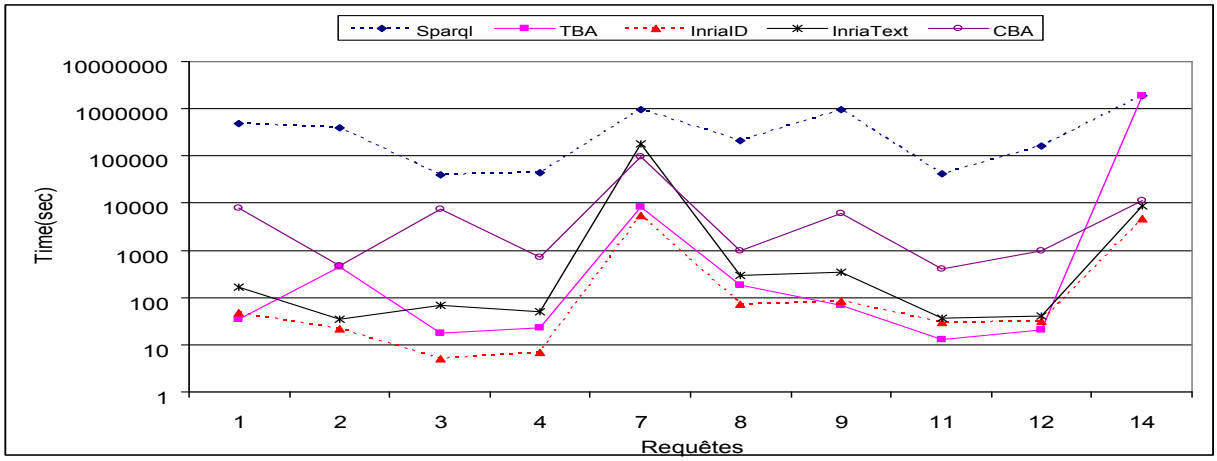


FIGURE 5.21 – Temps de traitement de requêtes sur LUBM100 sur la $\mathcal{BDBO}$ native

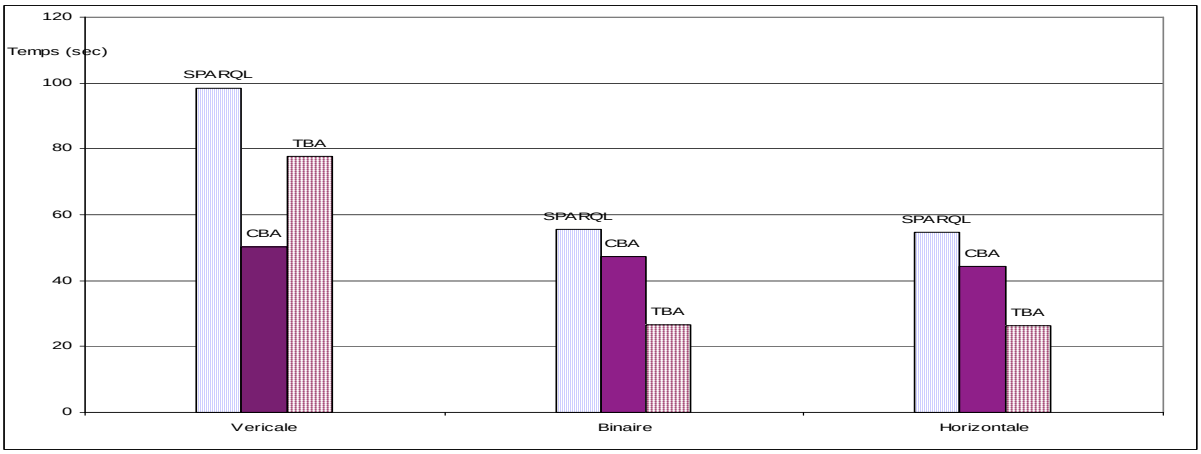


FIGURE 5.22 – Temps total de traitement de la charge de requêtes sur les $\mathcal{BDBO}$

rables (c'est le cas de la requête 7).

Par rapport à toute la charge de requêtes, l'*Inria_ID* fournit un bon temps global suivi de la méthode *CBA* pour les $\mathcal{BDBO}$ verticales (figures 5.23). Le temps global de *TBA* pour ce type de $\mathcal{BDBO}$ augmente en raison des requêtes ne bénéficiant pas des vues. En effet, ces requêtes sont traduites en SQL et exécutées sur la table des triplets, leurs temps d'exécution sont mauvais et influencent le temps cumulé. Mises à part ces requêtes, tous les temps de réponse de *TBA* sont bons. Il faut noter que toutes les approches présentent un gain substantiel pour les temps cumulés pour la charge de requêtes comme le montre les figures 5.22 et 5.23. L'approche *TBA* est meilleure pour toute la charge de requêtes dans les $\mathcal{BDBO}$ binaires et horizontales (figures 5.22).

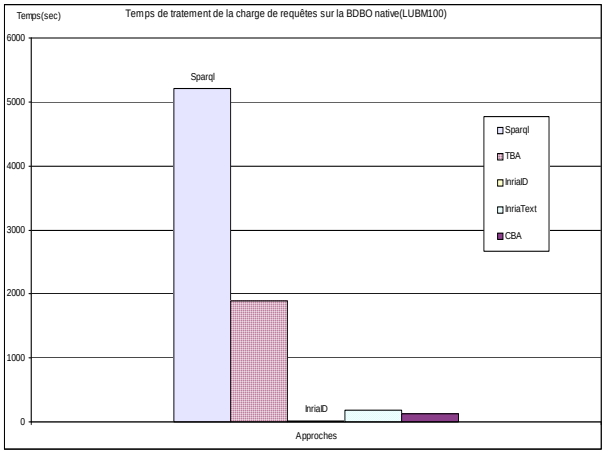


FIGURE 5.23 – Temps total de traitement de la charge de requêtes sur la $\mathcal{BDBO}$ native

6.2.3 Influences de la contrainte de stockage

Pour étudier l'influence de la taille de l'espace réservé pour les vues matérialisées sur le processus de sélection de vues, nous avons fixé la contrainte à 1%, 10% et 25% de la taille de toutes les vues candidates. Ensuite, nous avons fait des sélections des vues que nous avons créées pour chaque cas. Enfin, nous avons exécuté et mesuré le temps d'exécution des requêtes de notre charge de requêtes pour chaque cas. Nous avons utilisé l'approche *TBA*. Les résultats obtenus sont présentés sur la figure 5.24. Nous constatons que quand la contrainte de stockage est très forte c'est-à-dire que taille d'espace réservé aux vues matérialisées est petite (par exemple dans notre cas, 1% de l'espace nécessaire pour toutes les vues), la sélection est orientée vers les vues de sélection. Et les résultats sont aussi mitigés car ces vues n'apportent pas grand-chose à l'exécution de requêtes. A partir de 25% d'espace de toutes les vues, on obtient un résultat intéressant.

7 Conclusion

Dans ce chapitre nous nous sommes penchés sur le problème de conception physique des $\mathcal{BDBO}$. Vu la complexité de ce problème, nous avons choisi de traiter uniquement du problème de la sélection des vues matérialisées. La diversité des modèles de stockage et des architectures de ce type de BD complique davantage le problème de sélection. Une exploration de l'état de l'art sur la question nous a permis de constater que, dans les $\mathcal{BDBO}$, il n'a été traité que pour des $\mathcal{BDBO}$ verticale utilisant la table des triplets. Les approches proposées sont difficilement applicables à tous les types de $\mathcal{BDBO}$. Pour couvrir les différents types de $\mathcal{BDBO}$, nous avons présenté deux approches de résolution : une approche définie au niveau conceptuel (ontologique) basée sur les classes impliquées dans les requêtes et une approche définie au niveau logique basée sur les patrons de triplet utilisés dans la charge de requêtes. La première approche

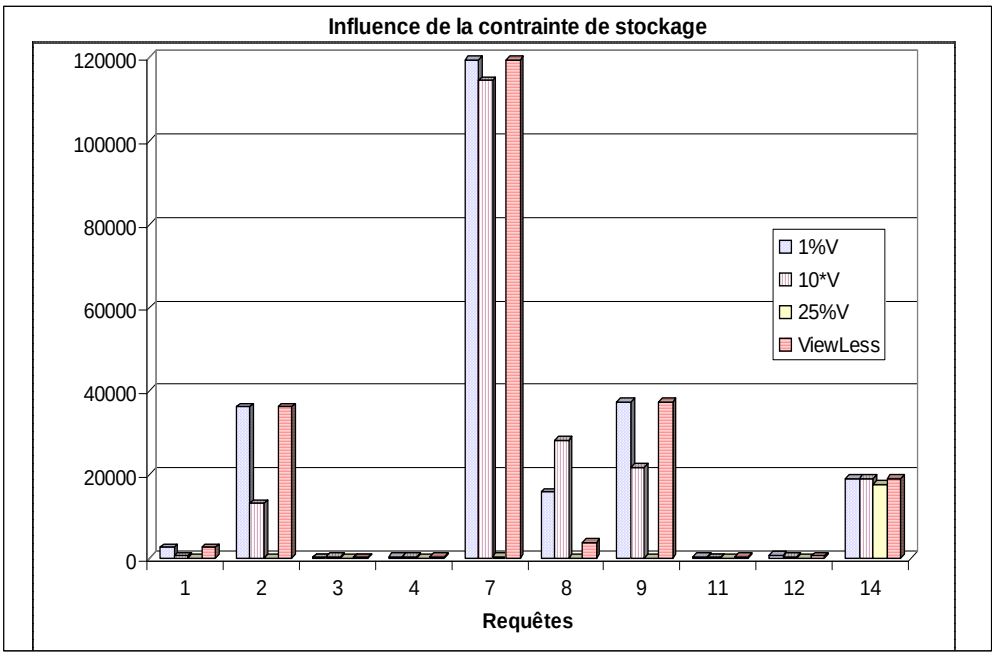


FIGURE 5.24 – Influence de la contrainte de stockage sur la sélection.

fait abstraction de l'implémentation tandis que la seconde prend en compte le modèle de stockage sous-jacent et utilise un modèle de coût adapté proposé dans le chapitre précédent. Ce modèle de coût offre une grande flexibilité et permet de quantifier la qualité des solutions lors de la phase de sélection. Ces approches se veulent génériques et prennent en compte la diversité des schémas de représentation des $\mathcal{BDBO}$, donc s'appliquent à tout type de $\mathcal{BDBO}$. Les expérimentations sur les différents types de $\mathcal{BDBO}$ montrent que ces approches sont pertinentes. Une comparaison avec les travaux antérieurs a été faite. Le tableau 5.2 révèle quelques points de comparaison de nos approches avec celles de Goasdoué et al. [182] et Castillo et Leser [142].

Les index générés par la méthode de Castillo et Leser [142] présentent une grande ressemblance avec nos vues. En effet, étant donné que ces index sont des combinaisons des patrons de triplet, certains d'entre eux se retrouveraient être nos nœuds intermédiaires. Mais comme, toutes les combinaisons possibles d'une certaine longueur k sont générées, certains de ces index ne se trouveront pas dans notre approche. Par contre, si $k=1$, toutes nos vues vont se retrouver parmi celles de Castillo et Leser [142]. A la différence de Castillo et Leser [142] qui génère les vues candidates par la combinaisons de tous les composants connexes d'une certaine longueur k, nous construisons un plan unifié d'exécution de toutes les requêtes et nous sélectionnons les nœuds les plus profitables pour le traitement de notre charge de requêtes. Nous savons que la sélection des vues est un problème NP-complet [200], et donc générer un grand nombre de vues compliquerait davantage l'obtention de la solution optimale. Dans l'approche de Castillo et Leser [142], pour une requête q ayant n patrons de triplet, on obtient $\sum_{i=k..n} C_i^n$ index. (c'est-à-dire, la somme de combinaison de i dans n). Pour une charge de requêtes Q, on peut avoir au pire des cas : $\sum_{q \in Q} \sum_{i=k..n} C_i^n$ index.

Le travail de Wilkinson [201] utilise une structure qui ressemble à notre approche basée sur les classes mais leurs propriétés regroupées peuvent provenir des différentes classes. Il faut noter que ce travail est plus relatif aux structures de stockage. Celui de Mengdong et Wu [202] sur la mise en cache de certains résultats des requêtes SPARQL en vue d'accélérer le traitement d'autres requêtes dans le cadre du Web Sémantique ressemble à notre approche basée sur les triplets (TBA) car les résultats mis en cache sont certains nœuds de notre plan unifié de requêtes. En revanche la sélection et matérialisation de ces nœuds n'étaient pas proposées.

Critères	CBA	TBA	Goasdoué et al.	Castillo et Leser
Type de $\mathcal{BDBO}$	I, II, III	I, II et III	I	I
Déploiement	centralisé	centralisé	centralisé	centralisé
Structure de données	graphe	graphe	multi-graphe	graphe
Schéma d'ontologie	utilisé	non	utilisé pour inférence	non
Fréquences	oui	oui	non	oui
Contrainte de stockage	non	oui	minimise le stockage mais pas de contrainte	oui
modèle de coût	coût de construction + coût accès	coût de construction + coût accès	coût cpu + esp de stockage +maintenance	coût de stockage

TABLE 5.2 – Comparaison des approches de *PSVM*.

Notre approche basée sur les classes présente quelques désavantages notamment la multiplicité de vues quand il y a une forte hiérarchie des classes (ce qui compliquerait la matérialisation) et la taille de vues qui a tendance à être énorme. L'inconvénient de cette seconde approche est également le grand nombre de calculs nécessaires pour la recherche du plan unifié optimal.

Conclusion générale et perspectives

Chapitre 6

Conclusion et Perspectives

6.1 Conclusion

Avec le développement de nombreuses ontologies dans différents domaines comme le Web Sémantique ou l'ingénierie, le besoin de stocker ces dernières dans des bases de données a conduit à la notion de $\mathcal{BDBO}$. Comme pour toutes les générations de bases de données, que ce soient les bases de données relationnelles, objet, XML ou décisionnelles, la conception physique d'une $\mathcal{BDBO}$ est un problème crucial pour garantir des temps de réponses acceptables aux requêtes des utilisateurs. Cette phase vise, en effet, à sélectionner un ensemble de structures d'optimisation sur une base de données pour optimiser une charge de requêtes. Cette phase repose donc sur la formalisation de la base de données considérée. En étudiant les travaux menés sur la conception physique des $\mathcal{BDBO}$, nous avons identifié que cette formalisation consiste à définir une $\mathcal{BDBO}$ comme une table de triplets qui est une représentation direct du langage RDF utilisé dans le contexte du Web Sémantique. Or, les ontologies étant utilisées dans de nombreux autres domaines et certaines étant définies avec d'autres modèles d'ontologies que ceux du Web Sémantique, toutes les $\mathcal{BDBO}$ ne se réduisent pas à une table de triplets. Nos travaux ont ainsi visé à prendre en compte la diversité des $\mathcal{BDBO}$ dans la phase de conception physique de ces dernières.

Les différentes contributions de ce travail sont les suivantes.

6.1.1 Etat de l'art sur les $\mathcal{BDBO}$

La première contribution de notre travail est un état de l'art détaillé des $\mathcal{BDBO}$ afin de définir les critères permettant de différencier les $\mathcal{BDBO}$ qui jouent un rôle important pour la phase de conception physique de ces bases de données. Les critères identifiés sont les suivants.

6.1.1.1 Modèle d'ontologie

Nous avons identifié que plusieurs modèles d'ontologies sont utilisés pour définir des ontologies. Ces modèles d'ontologies diffèrent selon les objectifs visés : certains se focalisent sur la gestion et l'échange de données, comme par exemple PLIB et RDFS, d'autres sont orientés vers l'inférence sur les données, comme par exemple DAML-ONT, DAML+OIL, F-Logic, et OWL. Chaque $\mathcal{BDBO}$ supporte un ou plusieurs modèles d'ontologies ce qui a un impact sur les performances associées puisque ces modèles diffèrent sur leur expressivité et sur les mécanismes de raisonnement proposés.

6.1.1.2 Modèle de stockage

Contrairement à un modèle conceptuel, une ontologie ne prescrit pas les propriétés qui sont utilisées pour décrire une instance, elle décrit uniquement toutes les propriétés qui peuvent être utilisées pour les caractériser. Ainsi, les instances ontologiques peuvent être plus ou moins structurées, c'est-à-dire que les instances peuvent n'avoir qu'un petit nombre de propriétés de l'ontologie (structuration faible) ou inversement un grand nombre de propriétés (structuration forte). La structuration des données est un facteur important lorsque l'on souhaite choisir un schéma de bases de données performant pour les stocker. La structuration des instances ontologiques étant variées et les $\mathcal{BDBO}$ stockant aussi l'ontologie associée, trois principales approches ont été proposées pour la persistance de l'ontologie associée et ses instances. (1) L'*approche verticale* où les ontologies et les données sont représentées dans un même schéma contenant une table principale appelée *table de triplets* constituée de trois colonnes *(sujet, prédicat, objet)*. Ce schéma rappelle la structure du RDF sur lequel se basent les modèles d'ontologies RDFS, DAML-OIL, OWL, etc. (2) L'*approche binaire* qui décompose les relations en deux catégories : les relations unaires (pour l'appartenance aux classes), et les relations binaires (pour les valeurs de propriétés). Cette approche a trois variantes suivant l'approche adoptée pour la représentation de l'héritage : (a) une table unique pour toutes les classes de l'ontologie, (b) une table par classe avec héritage de table (si une base de données relationnelle-objet est utilisée) et (c) une table par classe sans héritage de table. Toutes ces variantes utilisent une table binaire pour chaque propriété. (3) L'*approche horizontale* qui associe à chaque classe de l'ontologie une table ayant une colonne pour chaque propriété associée à une valeur pour au moins une instance de cette classe. Le choix d'un modèle de stockage particulier influence directement les performances des requêtes puisque celles-ci seront traduites par différentes opérations relationnelles sur le SGBD sous-jacent.

6.1.1.3 Architecture

Contrairement à une base de données traditionnelle, une $\mathcal{BDBO}$ doit stocker en plus des données l'ontologie qui en décrit le sens. Cette "*augmentation*" a vu naître trois grandes ar-

chitectures de $\mathcal{BDBO}$: (1) architecture de *type I* ou architecture "*deux quarts*" dans laquelle les ontologies et les données sont représentées dans un même schéma. Cette architecture est semblable à celle des bases de données classiques utilisant deux parties : *les données* et la *méta-base*, (2) architecture de *type II* ou architecture "*trois quarts*" où les ontologies et les données sont séparées et représentées dans deux schémas différents respectivement appelés *ontologie* et *données*. Le schéma *ontologie* est constitué d'un ensemble de tables (*class, property, subclass, subproperty, domain, range*, etc.) qui représente les constructeurs du modèle d'ontologies (classes, propriétés, etc.). Malgré cette séparation, cette architecture reste liée au modèle d'ontologies utilisé, (3) architecture de *type III* ou architecture "*quatre quarts*" qui ajoute à la structuration précédente une partie nommée *méta-schéma* jouant, pour les ontologies, le même rôle que celui joué par la *méta-base* pour *les données*. Cette partie stocke les constructeurs du modèle d'ontologies utilisé. Elle permet ainsi un accès générique au modèle ontologique ainsi que l'introduction de nouveaux constructeurs issus de différents modèles d'ontologies. Le choix d'une architecture particulière a un impact sur les performances puisque la taille des tables diminue lorsque l'on décompose le stockage des données dans plusieurs parties en fonction de leur niveau d'abstraction.

En considérant ces trois dimensions de la diversité des $\mathcal{BDBO}$, nous avons défini l'espace des $\mathcal{BDBO}$ comme étant un *cube* dont les axes sont respectivement les modèles de stockage, les architectures et les formalismes d'ontologies. Toute $\mathcal{BDBO}$ peut être classée dans une cellule de ce *cube*.

6.1.2 Evaluation de performance de $\mathcal{BDBO}$

Pour pouvoir aborder le problème de la conception des $\mathcal{BDBO}$ au regard de la diversité identifiée précédemment, il est important d'avoir une maîtrise des caractéristiques de ces bases de données. Pour cela, nous avons étudié quelques $\mathcal{BDBO}$ existantes et nous avons dégagé leurs principales caractéristiques. Cette étude nous a permis de proposer un modèle unifié des $\mathcal{BDBO}$ qui intègre leur diversité. Ce modèle unifié a été utilisé pour comparer au niveau macroscopique six $\mathcal{BDBO}$ largement utilisées dans plusieurs communautés dont trois industrielles (Oracle, IBM SOR et DB2RDF) et trois académiques (Jena, Sesame et OntoDB développée au sein de notre laboratoire). Pour compléter cette étude, nous avons fait une évaluation des performances de ces six $\mathcal{BDBO}$ en termes de temps de chargement des données et en termes de temps de traitement de requêtes. Cela nous a permis d'identifier des paramètres pertinents influençant le traitement des requêtes à savoir l'architecture, le modèle de stockage, le type de requête et le SGBD sous-jacent.

6.1.3 Modèle de coût pour les $\mathcal{BDBO}$ prenant en compte leur diversité

En détaillant les travaux réalisés sur la conception physique des bases de données traditionnelles, nous avons identifié que les modèles de coût sont au cœur de ces travaux. Or, nous n'avons pas trouvé de modèle de coût dans la littérature qui prennent en compte la diversité des $\mathcal{BDBO}$. Nous avons donc proposé un nouveau modèle de coût basé sur le modèle unifié des $\mathcal{BDBO}$ introduit précédemment. Notre modèle de coût quantifie les entrées-sorties nécessaires pour le traitement d'une requête. Il intègre la diversité des $\mathcal{BDBO}$ et peut aider les concepteurs à choisir un meilleur type de $\mathcal{BDBO}$ selon un critère de performance donné et une charge de requêtes usuelles. Ce modèle de coût a été mis en application dans une étude que nous avons menée dont les résultats confirment partiellement ceux que nous avons obtenus dans l'étude expérimentale faite précédemment. Les résultats confirmés sont les suivants. Le modèle de stockage vertical est meilleur en matière de chargement des données mais il est très coûteux pour les requêtes (en raison des auto-jointures nécessaires sur la table de triplets). Le modèle de stockage binaire est approprié pour les requêtes de sélection *mono-triplet* (avec peu de propriétés) et le modèle de stockage horizontal pour les requêtes de sélections *mono-triplet* et *pluri-triplets* mais le modèle de stockage horizontal peut conduire à des opérations d'unions et à la redondance des données. Concernant les résultats théoriques non confirmés par nos expérimentations nous notons qu'ils concernent des $\mathcal{BDBO}$ spécifiques. Nous pensons que cette divergence vient d'optimisations particulières faites dans ces $\mathcal{BDBO}$ ou des structures d'optimisation utilisées que nous ne considérons pas dans notre modèle de coût. En perspective de ces travaux sur le modèle de coût, il serait donc intéressant de spécialiser notre modèle pour une $\mathcal{BDBO}$ particulière.

6.1.4 Approches de sélection des vues matérialisées pour les $\mathcal{BDBO}$

La définition du modèle de coût précédent, nous a permis d'aborder le problème de la conception physique des $\mathcal{BDBO}$. Ce problème étant très large, nous nous sommes restreint au problème de sélection des vues matérialisées dans les $\mathcal{BDBO}$. Nous avons fait ce choix car les vues matérialisées sont des structures d'optimisation qui permettent d'optimiser de nombreux types de requêtes et qui, par ailleurs, ont été peu traitées dans la littérature. En effet, les rares travaux effectués dans ce sens portent sur les $\mathcal{BDBO}$ verticales et ainsi ils ne prennent pas en compte la diversité des $\mathcal{BDBO}$. La formalisation du problème de la conception physique des $\mathcal{BDBO}$ s'apparente à celle des bases de données traditionnelles. Nous rappelons que ce problème est NP-complet [200] dans les BD traditionnelles. La diversité des modèles de stockage et des architectures des $\mathcal{BDBO}$ vient compliquer ce problème. Nous avons identifié trois approches pour traiter le problème de sélection de vues matérialisées dans le contexte des $\mathcal{BDBO}$: (i) la *sélection basée sur l'ontologie et les requêtes*. Cette approche que nous appelons aussi *sélection conceptuelle* est originale car elle remonte la tâche de sélection des vues matérialisée à la phase conceptuelle. Ainsi, elle ne prend pas en compte l'implémentation physique

de la base de données mais requiert uniquement le schéma de l'ontologie. (ii) Une *sélection imposée*, qui suppose l'existence préalable d'une $\mathcal{BDBO}$. Elle suppose donc la connaissance du modèle de stockage, du SGBD sous-jacent et des autres caractéristiques de la $\mathcal{BDBO}$. Elle est similaire à celle faite dans le contexte des bases de données traditionnelles. (iii) La dernière approche appelée *approche simulée* intègre dans le processus de sélection des vues matérialisées la diversité des $\mathcal{BDBO}$. Nous avons proposé une *approche conceptuelle* et une *approche simulée*. Notre *approche conceptuelle* est basée sur les classes de l'ontologie impliquées dans la charge des requêtes. Elle fait une abstraction de l'implémentation des $\mathcal{BDBO}$. Par contre, elle nécessite la présence de l'ontologie. Notre approche simulée prend en compte la phase logique, physique et de déploiement. Elle est basée sur les triplets. Elle est générique et applicable à tout type de $\mathcal{BDBO}$. En effet, cette approche construit un graphe de requêtes (plan unifié des requêtes) basé sur les patrons de triplet des requêtes de la charge. Elle prend en compte le modèle de stockage sous-jacent et utilise notre modèle de coût dans le processus de la sélection de vues matérialisées. Les expérimentations réalisées sur les différents types de $\mathcal{BDBO}$ ont montré que nos approches sont pertinentes et comparables à d'autres approches traitant de la même question pour un type de $\mathcal{BDBO}$ particulier [182].

6.2 Perspectives

A court terme, nous pensons qu'il est important de faire une validation plus approfondie de nos travaux. En effet, même si nos expérimentations ont donné des résultats intéressants, il est important de les réaliser sur une plus grande volumétrie de données avec d'autres benchmarks impliquant des requêtes plus variées et sur d'autres $\mathcal{BDBO}$ pour voir la robustesse de nos propositions. Côté outillage, il serait intéressant d'envisager la mise au point d'un simulateur intégrant notre modèle de coût et nos approches.

A plus long terme, nous notons que nos travaux ont été menés dans un contexte bien particulier où l'on considère une $\mathcal{BDBO}$ comme un système centralisé nécessitant l'optimisation d'un ensemble de requêtes connu. Même si nous pensons que ce contexte concerne de nombreux cas d'application, il serait intéressant d'élargir nos travaux en remettant en cause certaines hypothèses faites. Nous pensons notamment au traitement des données massives ("big data") qui apporte de nouvelles dimensions partiellement traitées dans nos travaux. Nous décrivons ces dimensions ci-dessous et évoquons des perspectives de recherche qu'elles impliquent pour nos travaux.

6.2.1 Volume

Une $\mathcal{BDBO}$ centralisée n'est pas en mesure de gérer des données massives. En effet, celles-ci sont caractérisées par l'ajout de téraoctets de données par jour. Dans ce contexte, il est nécessaire de considérer les $\mathcal{BDBO}$ dans un contexte distribué ce que nous n'avons pas fait dans

nos travaux. Le partitionnement jouant un rôle crucial dans cette problématique et le partitionnement pouvant reposer sur un modèle de coût, nous pensons qu'il serait intéressant d'étendre nos travaux en considérant d'autres structures d'optimisation comme le partitionnement ou les index succincts très utilisés dans le contexte des données massives. Il faudra pour cela adapter notre modèle de coût à ces structures d'optimisation et prendre en considération les coûts de transfert de données. Il faudra également revoir les algorithmes de sélection puisque dans ce cas, nous ne ferons plus une sélection de manière isolée sur les vues matérialisées mais une sélection en combinant un ensemble de structures d'optimisation qui peuvent avoir un impact l'une sur l'autre.

6.2.2 Variété

Dans nos travaux nous avons considéré le stockage d'ontologies et de leurs instances. Comme nous l'avons vu, ces données présentent une diversité qui ont été au cœur de nos travaux. Dans le cadre des données massives, cette variété est encore plus grande puisqu'il s'agit de traiter des données de diverses natures qui peuvent être ontologiques mais également semi-structurées, textuelles, multimédia, etc. La gestion de cette diversité a vu naître de nombreuses nouvelles solutions de stockage qualifiées de NoSQL ou NewSQL qui proposent des performances et des fonctionnalités très variées. Une des questions centrales actuellement est de déterminer la solution à utiliser lorsque l'on fait face à un problème de traitement de données massives. Nous pensons que la démarche que nous avons suivie dans nos travaux pourrait être intéressante pour ce problème que nous qualifions de "déploiement". Il s'agirait d'abord de bien étudier les solutions NoSQL et NewSQL proposées pour identifier les dimensions de leur diversité et de confirmer ces dimensions par des études expérimentales. Cette étude devrait permettre de déterminer si il est possible de construire un modèle unifié pour ses solutions et un modèle de coût associé qui permettrait alors de traiter le problème du déploiement.

6.2.3 Vélocité

Dans nos travaux, nous considérons un contexte statique où l'ontologie est stable et les requêtes importantes sont connues à l'avance. Ces hypothèses ne sont pas forcément vraie dans un contexte de traitement de données massives où l'on considère que les données doivent être capturées et analysées en temps réel. Dans une $\mathcal{BDBO}$, cela voudrait dire que l'ontologie et/ou ses instances pourraient être mises à jour fréquemment et que les requêtes pourraient ne pas être connues à l'avance. Il faudrait donc étudier les impacts de l'évolution de l'ontologie et des données sur les approches de sélection des vues matérialisées proposées dans nos travaux. De même, il faudrait redéfinir ce problème, lorsque la charge de requête est dynamique. Enfin, le problème de maintenance des vues matérialisées serait crucial dans un tel contexte puisque les données changeraient rapidement. Aussi, il serait nécessaire d'aborder la question de la maintenance incrémentale des vues matérialisées dans les $\mathcal{BDBO}$.

Bibliographie

[1] M. Rousset et C. Reynaud, « PICSEL and xyleme: Two illustrative information integration agents », *in Intelligent Information Agents - The AgentLink Perspective*, p. 50–78, 2003.

[2] G. Pierra, « Context representation in domain ontologies and its use for semantic integration of data », *Journal Of Data Semantics (JoDS)*, vol. 10, p. 174–211, 2008.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider et L. Stein, « Owl web ontology language reference », *W3C, http://www.w3.org/TR/owlref/*, 2004.

[4] D. J. Abadi, A. Marcus, S. R. Madden et K. Hollenbach, « Scalable Semantic Web Data Management Using Vertical Partitioning », *in Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, p. 411–422, 2007.

[5] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis et K. Tolle, « The ics-FORTH RDFSuite: Managing voluminous RDF description bases », *in Proceedings of the 2nd International Workshop on the Semantic Web*, p. 1–13, 2001.

[6] K. Wilkinson, « Jena Property Table Implementation », *in Proceedings of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS'06)*, p. 35–46, 2006.

[7] B.McBride, « Jena: Implementing the rdf model and syntax specification », *in Proceedings of the 2nd International Workshop on the Semantic Web*, 2001.

[8] Z. Pan et J. Heflin, « Dldb: Extending relational databases to support semantic web queries », *in Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, p. 109–113, 2003.

[9] H. Dehainsala, G. Pierra et L. Bellatreche, « OntoDB: An Ontology-Based Database for Data Intensive Applications », *in Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, p. 497–508, 2007.

[10] A. Seaborne, « Rdql – a query language for rdf », *W3C Member Submission 9 January*, 2004. `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`.

[11] J. Broeskstra et A. Kampman, « SeRQL: A Second Generation RDF Query Language », *in SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, November 2003.

[12] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis et M. Scholl, « RQL: a declarative query language for RDF », *in Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, p. 592–603, 2002.

[13] L. Miller, A. Seaborne et A. Reggiori, « Three Implementations of SquishQL, a Simple RDF Query Language », *in Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, p. 423–435, 2002.

[14] E. Prud'hommeaux et A. Seaborne, « Sparql query language for rdf », *W3C Candidate Recommendation 14 June*, 2007. `http://www.w3.org/TR/rdf-sparql-query/`.

[15] S. Jean, Y. Ait-Ameur et G. Pierra, « Querying Ontology Based Database Using OntoQL (an Ontology Query Language) », *in ODBASE'06*, p. 704–721, 2006.

[16] C. D. Rogozan, *Gestion de l'évolution des ontologies: méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions d'ontologie*. Thèse doctorat, Université du Québec à Montréal, 2009.

[17] P. Foulquié et R. Saint-Jean, *Dictionnaire de la langue philosophique*. Paris, Presses Universitaires de France, 1962.

[18] L. Meynard, *La Métaphysique*. Dans Foulq.-St-Jean 1962, 1959.

[19] T. R. Gruber, *Formal ontology in conceptual analysis and knowledge representation. Chapter: Towards principles for the design of ontologies used for knowledge sharing.* Kluwer Academic Publishers., 1993.

[20] G. Pierra, « Context representation in domain ontologies and its use for semantic integration of data », *Journal Of Data Semantics (JODS)*, vol. 4900, p. 173–210, 2008.

[21] D. Brickley et R. Guha, « Rdf vocabulary description language 1.0: Rdf schema », *W3C, http://www.w3.org/TR/rdfschema/*, 2002.

[22] ISO-13584-42, « Industrial Automation Systems and Integration Parts LIBrary Part 42 : Description methodology : Methodology for Structuring Parts families », rap. tech., ISO, 1998.

[23] N. Guarino, *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, vol. 46. IOS press, 1998.

[24] P. R. S. Visser, M. D. Beer, T. J. M. Bench-Capon, B. M. Diaz et M. J. R. Shave, « Resolving ontological heterogeneity in the kraft project », *in Proceedings of the 10th International Conference on Database and Expert Systems Applications*, DEXA '99, (London, UK), p. 668–677, Springer-Verlag, 1999.

[25] G. A. Miller, « Wordnet: a lexical database for english », *Communications of the ACM*, vol. 38, p. 39–41, 1995.

[26] R. Apweiler, A. Bairoch, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O. Donovan, N. Redaschi et L. S. Yeh, « Uniprot: the universal protein knowledgebase », *Nucleic Acids Research*, vol. 32, p. D115–D119, 2004.

[27] G. Pierra, « Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data », *Journal of Advanced Manufacturing Systems, World Scientific Publishing Company*, 2006.

[28] S. Jean, G. Pierra et Y. Ait-Ameur, *Domain Ontologies : a Database-Oriented Analysis*, vol. 1, p. 238–254. Springer, 2007.

[29] S. Weibel, « The dublin core: a simple content description model for electronic resources », *Bulletin of the American Society for Information Science and Technology*, vol. 24, p. 9–11, 1997.

[30] Z. P. Y. Guo et J. Heflin, « Lubm: A benchmark for owl knowledge base systems », *in Journal of Web Semantics*, vol. 3, p. 158–182, 2005.

[31] G. Pierra, « Context-explication in conceptual ontologies : The plib approach », *in Proc. of Concurrent Engineering (CE'2003)*, p. 243–254, July 2003.

[32] M. Kifer, G. Lausen et J. Wu, « Logical Foundations of Object-Oriented and Frame-Based Languages », *Journal of the ACM (JACM)*, vol. 42, p. 741–843, 1995.

[33] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider et L. A. Stein, *DAML+OIL Reference Description*. World Wide Web Consortium, december 2001. `http://www.w3.org/TR/daml+oil-reference`.

[34] M. Klaene, « Using ibatis sql maps for java data access », *Resources for Java server-side developers*, vol. 1, 2004.

[35] P. Hitzler, M. Krtzsch et S. Rudolph, « Foundations of semantic web technologies », *Chapman & Hall/CRC*, 2009.

[36] F. Manola, E. Miller, B. McBride *et al.*, « Rdf primer », *W3C recommendation*, vol. 10, p. 6, 2004.

[37] D. C. Fallside, « Xml schema part 0: primer second edition », *W3C recommendation*, 2004.

[38] M. Smith, C. Welty et D. L. McGuinness, « Owl web ontology language guide. w3c recommendation », *URL http://www. w3. org/TR/owl-guide*, 2009.

[39] S. Tobies, *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. Thèse doctorat, RWTH Aachen, Germany, 2001.

[40] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu et Y. Pan, « Minerva: A scalable owl ontology storage and inference system », *in The Semantic Web Conference*, p. 429–443, Springer, 2006.

[41] P. Spyns, R. Meersman et M. Jarrar, « Data modelling versus ontology engineering », *ACM SIGMod Record*, vol. 31, no. 4, p. 12–17, 2002.

[42] C. Fankam, L. Bellatreche, D. Hondjack, Y. A. Ameur et G. Pierra, « Sisro, conception de bases de données à partir d'ontologies de domaine. », *Technique et Science Informatiques*, vol. 28, no. 10, p. 1233–1261, 2009.

[43] M. Lenzerini, « Data integration : A theoretical perspective », *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, p. 233–246, June 2002.

[44] Roldan-Garcia, M. Navas-Delgado, Aldana-Montes et J. Aldana-Montes, « A design methodology for semantic web database-based systems », *Third International Conference on Information Technology and Applications(ICITA'05)*, vol. 1, p. 233–237, 2005.

[45] V. Sugumaran et V. C. Storey, « The role of domain ontologies in database design: An ontology management and conceptual modeling environment », *in ACM Trans. Database Syst.*, vol. 31, (New York, NY, USA), p. 1064–1094, ACM Press, 2006.

[46] G. Pierra, H. Dehainsala, Y. Ait-Ameur et L. Bellatreche, « Base de Données à Base Ontologique : principes et mise en œuvre », *Ingénierie des Systèmes d'Information*, vol. 10, p. 91–115, 2005.

[47] K. Wilkinson, C. Sayers, H. Kuno et D. Reynolds, « Efficient rdf storage and retrieval in jena2 », *HP Laboratories Technical Report HPL-2003-266*, p. 131–150, 2003.

[48] J. Broekstra, A. Kampman et F. van Harmelen, « Sesame: A generic architecture for storing and querying rdf and rdf schema », *in Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, p. 54–68, 2002.

[49] M. J. Park, J. H. Lee, C. H. Lee, J. Lin, O. Serres et C. W. Chung, « An Efficient and Scalable Management of Ontology », *in Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, p. 975–980, 2007.

[50] L. Ma, Z. Su, Y. Pan, L. Zhang et T. Liu, « RStar: an RDF Storage and Query System for Enterprise Resource Management », *in Proceedings of the 30th International Conference on Information and Knowledge Management (CIKM'04)*, p. 484–491, 2004.

[51] E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, G. Stumme, Y. Sure, J. Tane, R. Volz et V. Zacharias, « Kaon - towards a large scale semantic web », *in Proceedings of the 3rd International Conference on E-Commerce and Web Technologies (EC-WEB'02)*, (London, UK), p. 304–313, Springer-Verlag, 2002.

[52] S. Harris et N. Gibbins, « 3store: Efficient Bulk RDF Storage », *in Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, p. 1–15, 2003.

[53] K. Stoffel, M. Taylor et J. Hendler, « Efficient Management of Very Large Ontologies », *in Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference AAAI'97/IAAI'97*, p. 442–447, 1997.

[54] S. Das, « Rdf support in oracle rdbms », *Oracle Technical Presentation*, 2009.

[55] C. Murray, « Oracle spatial resource description framework (rdfi) », *Oracle Corporation*, 2005.

[56] IBM, « Thinking ontologies at ibm center for advanced studies », *IBM Center for Advanced Studies of Rome*, 2006.

[57] C. Fankam, « Ontodb2 : un système flexible et efficient de base de données à base ontologique pour le web sémantique et les données techniques », ph.d. thesis, Poitiers University, 2009.

[58] Y. Theoharis, V. Christophides et G. Karvounarakis, « Benchmarking Database Representations of RDF/S Stores », *in Proceedings of the 4th International Semantic Web Conference (ISWC'05)*, p. 685–701, 2005.

[59] J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan et Y. Yu, « Sor: a practical system for ontology storage, reasoning and search », *in Proceedings of the 33rd international conference on Very large data bases (VLDB'07)*, p. 1402–1405, 2007.

[60] D. Nguyen-Xuan, *Intégration de base de données hétérogènes par articulation apriori d'ontologies : application aux catalogues de composants industriels*. Thèse doctorat, LISI/ENSMA et Université de Poitiers, 2006.

[61] M. del Mar Roldán García, I. N. Delgado et J. F. A. Montes, « A Design Methodology for Semantic Web Database-Based Systems », *in Proceedings of the 3rd International Conference on Information Technology and Applications (ICITA'05)*, p. 233–237, IEEE Computer Society, 2005.

[62] N. Belaid, Y. Aït Ameur et J. F. Rainaud, « A semantic handling of geological modeling workflows », *in International ACM Conference on Management of Emergent Digital EcoSystems*, p. 83–90, 2009.

[63] S. Khouri et L. Bellatreche, « Osons tout persister dans un entrepôt: Données et modèles », *in 7èmes Journées Francophones sur les Entrepôts de Données et analyse en Ligne (EDA'11)*, 2011.

[64] Object Management Group, *Meta Object Facility (MOF), formal/02-04-03*, october 2002.

[65] J. Bailey, F. Bry, T. Furche et S. Schaffert, « Web and semantic web query languages: A survey », *in Proceedings of the First international conference on Reasoning Web*, p. 35–133, Springer-Verlag, 2005.

[66] J. J. Carroll, C. Bizer, P. Hayes et P. Stickler, « Named Graphs, Provenance and Trust », *in Proceedings of the 14th international conference on World Wide Web (WWW'05)*, (New York, NY, USA), p. 613–622, ACM Press, 2005.

[67] K. Tolle et F. Wleklinski, *easy RDF Query Language (eRQL)*, 2004. `http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/eRQL`.

[68] J. Robie, L. M. Garshol, S. Newcomb, M. Fuchs, L. Miller, D. Brickley, V. Christophides et G. Karvounarakis, « The syntactic web: Syntax and semantics on the web », *Markup Languages: Theory and Practice*, vol. 3, p. 411–440, 2001.

[69] D. Steer, « Treehugger 1.0 introduction », *Online only*, 2003.

[70] U. Ogbuji, « Thinking xml: Basic xml and rdf techniques for knowledge management: Part 6: Rdf query using versa », *Online only, April*, 2002.

[71] K. Matsuyama, M. Kraus, K. Kitagawa et N. Saito, « A path-based rdf query language for cc/pp and uaprof », *in Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, p. 3–7, IEEE, 2004.

[72] M. Marchiori, A. Epifani et S. Trevisan, « Metalog v2. 0: Quick user guide », rap. tech., Technical report, W3C, 2004.

[73] E. Prud'hommeaux, « Algae rdf query language », *Online only*, 2004.

[74] T. Berners-Lee, « N3ql–rdf data query language », *Online only*, 2004.

[75] N. Bassiliades et I. P. Vlahavas, « Capturing rdf descriptive semantics in an object oriented knowledge base system. », *in WWW (Posters)*, 2003.

[76] M. Sintek et S. Decker, « Triple – a query, inference, and transformation language for the semantic web », *in The Semantic Web – ISWC 2002*, p. 364–378, Springer, 2002.

[77] D. Reynolds, « Rdf-qbe: a semantic web building block », rap. tech., Technical Report HPL-2002-327, HP Labs, 2002.

[78] S. Powers, *Practical rdf*. " O'Reilly Media, Inc.", 2003.

[79] A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, K. Kjernsmo *et al.*, « Sparql/update: A language for updating rdf graphs », *W3C Member Submission*, vol. 15, 2008.

[80] R. Ramakrishnan, *Database Management Systems*. WCB/McGraw Hill, 1998.

[81] E. Bertino, M. Negri, G. Pelagatti et L. Sbattella, « Object-oriented query languages: The notion and the issues », *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 3, p. 223–237, 1992.

[82] L. V. S. Lakshmanan, éd., *Design and Management of Data Warehouses, Proceedings of the 4th Intl. Workshop (DMDW'02), Toronto, Canada, May 27*, vol. 58*in CEUR Workshop Proceedings*, CEUR-WS.org, 2002.

[83] A. Datta, B. Moon, K. Ramamritham, H. Thomas et I. Viguier, « Have your data and index it, too: Efficient storage and indexing for datawarehouses », Techreport Technical Report 98-7, Department of Computer Science, The University of Arizona, 1998.

[84] S. Chaudhuri et V. Narasayya, « Self-tuning database systems: a decade of progress », *in Proceedings of the 33rd international conference on Very large data bases*, p. 3–14, VLDB Endowment, 2007.

[85] B. Kamel, *De la conception physique aux outils d'administration et de tuning des entrepôts de données*. Thèse doctorat, ENSMA et Université de Poitiers, juillet 2009.

[86] H. Gupta, V. Harinarayan, A. Rajaraman et J. D. Ullman, « Index selection for olap », *in Proceedings of 13th International Conference on Data Engineering*, p. 208–219, IEEE, 1997.

[87] H. Gupta, « Selection of views to materialize in a data warehouse », *in ICDT*, p. 98–112, 1997.

[88] A. A. B. Lima, C. Furtado, P. Valduriez et M. Mattoso, « Improving parallel olap query processing in database clusters with data replication », *Distributed and Parallel Database Journal*, vol. 25, no. 1-2, p. 97–123, 2009.

[89] S. Agrawal, V. Narasayya et B. Yang, « Integrating vertical and horizontal partitioning into automated physical database design », *in Proceedings of the ACM SIGMOD international conference on Management of data*, p. 359–370, ACM, 2004.

[90] D. Comer, « The difficulty of optimum index selection », *ACM Transactions on Database Systems (TODS)*, vol. 3, p. 440–445, 1978.

[91] N. Pasquier, Y. Bastide, R. Taouil et L. Lakhal, « Discovering frequent closed itemsets », *ICDT*, p. 398–416, 1999.

[92] M. Simonnard, *Programmation linéaire*. Dunod, 1962.

[93] S. Chaudhuri et V. Narasayya, « Autoadmin 'what-if' index analysis utility », *Proceedings of the ACM SIGMOD International Conference on Management of Data*, p. 367–378, June 1998.

[94] H. Gupta, « Selection and maintenance of views in a data warehouse », ph.d. thesis, Stanford University, September 1999.

[95] J. Ullman, « Efficient implementation of data cubes via materialized views », *in the Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, p. 386–388, 1996.

[96] P. A. Bernstein et D.-M. W. Chiu, « Using semi-joins to solve relational queries », *Journal of the ACM*, vol. 28, p. 25–40, January 1981.

[97] Y. Kotidis et N. Roussopoulos, « Dynamat: a dynamic view management system for data warehouses », *in ACM SIGMOD Record*, vol. 28, p. 371–382, ACM, 1999.

[98] Z. Bellahsene et P. Marot, « Materializing a set of views: Dynamic strategies and performance evaluation », *in Database Engineering and Applications Symposium, 2000 International*, p. 424–428, IEEE, 2000.

[99] Z. Bellahsene, M. Cart et N. Kadi, « A cooperative approach to view selection and placement in p2p systems », *in On the Move to Meaningful Internet Systems: OTM'10*, p. 515–522, Springer, 2010.

[100] J. Zhou, P.-A. Larson, J. Goldstein et L. Ding, « Dynamic materialized views », *in Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, p. 526–535, IEEE, 2007.

[101] I. Mami et Z. Bellahsene, « A survey of view selection methods », *SIGMOD Record*, vol. 41, no. 1, p. 20–29, 2012.

[102] M. de Souza et M. Sampaio, « Efficient materialization and use of views in data warehouses », *SIGMOD Record*, vol. 28, no. 1, p. 78–83, 1999.

[103] V. Harinarayan, A. Rajaraman et J. D. Ullman, « Implementing data cubes efficiently », *in ACM SIGMOD Record*, vol. 25, p. 205–216, ACM, 1996.

[104] E. Baralis, S. Paraboschi et E. Teniente, « Materialized views selection in a multidimensional database. », *in VLDB*, vol. 97, p. 156–165, 1997.

[105] K. R. H. Uchiyama et T. Teorey, « A progressive view materialization algorithm », *in 2nd ACM International Workshop on Data warehousing and OLAP (DOLAP 99), Kansas City, USA*, p. 36–41, 1999.

[106] P. D. A. Shukla et J. Naughton, « Materialized view selection for multi-cubedata models », *in 7 th International Conference on Extending DataBase Technology (EDBT 00), Konstanz, Germany*, p. 269–284, 2000.

[107] T. Nadeau et T. Teorey, « Achieving scalability in olap materialized view selection », *in 5th ACM International Workshop on Data Warehousing and OLAP (DOLAP 02), McLean, USA*, p. 28–34, 2004.

[108] M. Golfarelli et S. Rizzi, « View materialization for nested gpsj queries. », *in DMDW*, p. 10, 2000.

[109] D. Theodoratos et W. Xu, « Constructing search spaces for materialized view selection », *in Proceedings of the 7th ACM international workshop on Data warehousing and OLAP (DOLAP 04), Washington DC, USA*, p. 112–121, ACM, 2004.

[110] A. Gupta, I. S. Mumick *et al.*, « Maintenance of materialized views: Problems, techniques, and applications », *IEEE Data Eng. Bull.*, vol. 18, p. 3–18, 1995.

[111] X. Y. C. Zhang et J. Yang, « An evolutionary approach to materialized view selection in a data warehouse environment », *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 31, no. 3, p. 282–294, 2001.

[112] S. V. e. K. K. S.R. Valluri, « View relevance driven materialized view selection in data warehousing environment », *in 13th Australasian Database Technologies Conference (ADC'02), Melbourne, Australia*, p. 187–196, 2002.

[113] X. Baril et Z. Bellahsène, « Selection of materialized views: a cost-based approach », *in 15th International Conference on Advanced Information Systems Engineering (CAiSE'03), Klagenfurt, Austria*, p. 665–680, 2003.

[114] A. Sanjay, C. Surajit et V. R. Narasayya, « Automated selection of materialized views and indexes in microsoft sql server », *Proceedings of the International Conference on Very Large Databases*, p. 496–505, September 2000.

[115] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[116] J. Kratica, I. Ljubic et D. Tosic, « A genetic algorithm for the index selection problem », *in Applications of Evolutionary Computing, EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, Essex, UK, April 14-16, 2003, Proceedings*, p. 280–290, 2003.

[117] M. Adiba et B. G. Lindsay, « Database snapshots », *vldb*, p. 86–91, October 1980.

[118] B. G. Lindsay, L. M. Haas, C. Mohan, H. Pirahesh et P. F. Wilms, « A snapshot differential refresh algorithm », *in Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 28-30, 1986.*, p. 53–60, 1986.

[119] S. Ceri et J. Widom, « Deriving production rules for incremental view maintenance », *in Proceedings of 17th International Conference on Very Large Data Bases, Barcelona, Catalonia, Spain*, p. 577–589, September 1991.

[120] J. A. Blakeley, P.-A. Larson et F. W. Tompa, « Efficiently updating materialized views », *in ACM SIGMOD Record*, vol. 15, p. 61–71, ACM, 1986.

[121] A. Segev et J. Park, « Maintaining materialized views in distributed databases », *in Proceedings of fifth International Conference on Data Engineering*, p. 262–270, IEEE, 1989.

[122] D. Srivastava, S. Dar, H. Jagadish et A. Y. Levy, « Answering queries with aggregation using views », *in Proceedings of the 22th International Conference on Very Large Data Bases*, p. 318–329, Morgan Kaufmann Publishers Inc., 1996.

[123] J. Gryz, « Query rewriting using views in the presence of functional and inclusion dependencies », *Information Systems*, vol. 24, p. 597–612, 1999.

[124] S. Ceri, M. Negri et G. Pelagatti, « Horizontal data partitioning in database design », *in Proceedings of the ACM SIGMOD international conference on Management of data*, p. 128–136, ACM, 1982.

[125] D. DeWitt et J. Gray, « Parallel database systems: the future of high performance database systems », *Communications of the ACM*, p. 85–98, 1992.

[126] S. Chakravarthy, J. Muthuraj, R. Varadarajan et S. B. Navathe, « An objective function for vertically partitioning relations in distributed databases and its analysis », *in Distributed and Parallel Databases Journal*, vol. 2, p. 183–207, April 1994.

[127] L. Bellatreche et K. Y. Woameno, « Dimension table driven approach to referential partition relational data warehouses », *in to appear in ACM Twelfth International Workshop on Data Warehousing and OLAP (DOLAP09)*, 2009.

[128] L. Bellatreche, M. Schneider, H. Lorinquer et M. Mohania, « Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses », *in Data Warehousing and Knowledge Discovery*, p. 15–25, Springer, 2004.

[129] J. Rao, C. Zhang, N. Megiddo et G. Lohman, « Automating physical database design in a parallel database », *in Proceedings of the ACM SIGMOD international conference on Management of data*, p. 558–569, ACM, 2002.

[130] T. Stöhr, H. Märtens et E. Rahm, « Multi-dimensional database allocation for parallel data warehouses. », *in VLDB*, p. 273–284, 2000.

[131] L. Bellatreche, *Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données*. Thèse doctorat, Université Blaise Pascal de Clermont Ferrand, décembre 2000.

[132] M. T. Özsu et P. Valduriez, *Principles of distributed database systems*. Springer, 2011.

[133] D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano et S. Fadden, « Db2 design advisor: integrated automatic physical database design », *in Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, p. 1087–1097, VLDB Endowment, 2004.

[134] S. Rozen et D. Shasha, « A framework for automating physical database design », *in Proceedings of the 17th International Conference on Very Large Data Bases*, p. 401–411, Morgan Kaufmann Publishers Inc., 1991.

[135] S. Finkelstein, M. Schkolnick et P. Tiberio, « Physical database design for relational databases », *ACM Transactions on Database Systems*, vol. 13, p. 91–128, 1988.

[136] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait et M. Ziauddin, « Automatic sql tuning in oracle 10g », *in Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, p. 1098–1109, VLDB Endowment, 2004.

[137] C. Maier, D. Dash, I. Alagiannis, A. Ailamaki et T. Heinis, « Parinda: an interactive physical designer for postgresql », *in Proceedings of the 13th International Conference on Extending Database Technology*, p. 701–704, ACM, 2010.

[138] P. O'Neil et D. Quass, « Improved query performance with variant indexes », *sigmod*, p. 38–49, May 1997.

[139] L. Bellatreche, K. Boukhalfa et M. K. Mohania, « Pruning search space of physical database design », *in 18th International Conference On Database and Expert Systems Applications (DEXA'07)*, p. 479–488, 2007.

[140] N. Bruno et S. Chaudhuri, « To tune or not to tune?: a lightweight physical design alerter », *in Proceedings of the 32nd international conference on Very large data bases*, p. 499–510, VLDB Endowment, 2006.

[141] K. Boukhalfa et L. Bellatreche, « Combinaison des algorithmes génétique et de recuit simulé pour la conception physique des entrepôts de données. », *in INFORSID*, p. 673–686, 2006.

[142] R. Castillo et U. Leser, « U.: Selecting materialized views for rdf data », *in In: Semantic Web Information Management Workshop (SWIM*, Citeseer, 2010.

[143] F. Goasdoué, K. Karanasos, J. Leblay et I. Manolescu, « View selection in semantic web databases », *Proc. VLDB Endow.*, vol. 5, p. 97–108, oct. 2011.

[144] I.Horrocks et U.Sattler, « Ontology reasoning in the shoq(d) description logic », *In Proc. of the 17th Int. Joint Conf. on Artificial Intelligence*, p. 199–204, 2001.

[145] I. Horrocks, P. F. Patel-Schneider et F. van Harmelen, « From shiq and rdf to owl: the making of a web ontology language. », *J. Web Sem.*, vol. 1, no. 1, p. 7–26, 2003.

[146] I.Horrocks et U.Sattler, « Decidability of shiq with complex role inclusion axioms. », *Artificial Intelligence*, vol. 160, p. 79–104, 2004.

[147] I.Horrocks, U.Sattler et S.Tobies, « Practical reasoning for expressive description logics », *In Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, vol. 1705, p. 161–180, 1999.

[148] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai et J. Srinivasan, « Implementing an inference engine for rdfs/owl constructs and user-defined rules in oracle », *in ICDE'2008*, p. 1239–1248, April 2008.

[149] O. L. T. Berners-Lee, J. Handler, « The semantic web », *Scientific American*, May 2001.

[150] IBM, *RDF application development for IBM data servers*. IBM, 2012. http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.db2.-luw.wn.doc%2Fdoc%2Fc0060011.html.

[151] G. R. Mario Briggs, Priya Ranjan Sahoo et F. Anwar, « Resource description framework application development in db2 10 for linux, unix, and windows », *IBM*, OCT 2012.

[152] D. Beckett, *RDF 1.1 N-Triples*. W3C Recommendation, February 2014. `http://www.w3.org/TR/2014/REC-n-triples-20140225/`.

[153] M. Briggs, « Db2 nosql graph store what, why & overview », Mar. 2012. IBM : https://www.ibm.com/developerworks/mydeveloperworks/blogs/nlp/resource/DB2_NoSQLGraphStore.pdf?lang=en.

[154] G. Gardarin, *Bases de données, les systèmes et leurs langages*. Eyrolles, 1994.

[155] M. Klein et N. F. Noy, « A component-based framework for ontology evolution », *in Proceedings of the IJCAI*, vol. 3, Citeseer, 2003.

[156] L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi et D. Laurent, « A framework for combining rule-based and cost-based approaches to optimize olap queries », *in 1ème Journée Francophone sur les Entrepôts de données et l'Analyse en ligne (EDA'05), Revue des Nouvelles Technologies de l'Information, RNTI-B-1*, p. 177–196, 2005.

[157] L. Bellatreche, K. Boukhalfa et P. Richard, « Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms », *International Journal of Data Warehousing and Mining*, vol. 5, p. 1–23, March 2009.

[158] B. Soumia, *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données : application aux plateformes parallèles*. Thèse doctorat, ENSMA - Poitiers, juin 2014.

[159] M. G. Lohman, D. Daniels, L. M. Haas, R. Kistler et P. G. Selinger, « Optimization of nested queries in a distributed relational database », *Proceedings of the International Conference on Very Large Databases*, p. 403–415, August 1984.

[160] M. T. Özsu et P. Valduriez, *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[161] P. M. G. Apers, « Data allocation in distributed database systems », *ACM Transactions on database systems*, vol. 13, no. 3, p. 263–304, 1988.

[162] L. Bellatreche, K. Karlapalem et Q. Li, « Complex methods and class allocation in distributed object-oriented databases », *in the 5th International Conference on Object Oriented Information Systems (OOIS'98)*, p. 239–256, September 1998.

[163] G. Gardarin, J.-R. Gruser et Z.-H. Tang, « A cost model for clustered object-oriented databases », *VLDB*, p. 323–334, 1995.

[164] G. Gardarin, J.-R. Gruser et Z.-H. Tang, « A cost-based selection of path expression processing algorithms in object-oriented databases », *vldb*, p. 390–401, 1996.

[165] E. Bertino et P. Foscoli, « An analytical model of object-oriented query costs », *Proceedings of the Fifth International Workshop on Persistent Object Systems, San Miniato (Pisa)*, p. 241–261, September 1992.

[166] E. Bertino, « On modeling cost functions for object-oriented databases », *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, p. 500–508, May/June 1997.

[167] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita et C. Zhang, « Storing and querying ordered xml using a relational database system », *in SIGMOD '02: Proceedings of the ACM SIGMOD international conference on Management of data*, (New York, NY, USA), p. 204–215, ACM Press, 2002.

[168] D. Florescu et D. Kossmann, « A performance evaluation of alternative mapping schemes for storing XML data in a relational database », rap. tech., NRIA Rocquencourt - RODIN, 1999.

[169] A. Maduko, K. Anyanwu, A. P. Sheth et P. Schliekelman, « Estimating the cardinality of rdf graph patterns », *in WWW* (C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider et P. J. Shenoy, éds), p. 1233–1234, ACM, 2007.

[170] E. P. Shironoshita, M. T. Ryan et M. R. Kabuka, « Cardinality estimation for the optimization of queries on ontologies », *SIGMOD Record*, vol. 36, p. 13–18, 2007.

[171] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer et D. Reynolds, « Sparql basic graph pattern optimization using selectivity estimation », *in Proceedings of the 17th international conference on World Wide Web*, WWW '08, (New York, NY, USA), p. 595–604, ACM, 2008.

[172] G. Neumann, Thomas et Weikum, « Rdf-3x: a risc-style engine for rdf », *Proc. VLDB Endow.*, vol. 1, p. 647–659, août 2008.

[173] Z. Kaoudi, K. Kyzirakos et M. Koubarakis, « Sparql query optimization on top of dhts », *in International Semantic Web Conference (1)*, p. 418–435, 2010.

[174] T. Neumann et G. Moerkotte, « Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins », *in ICDE*, p. 984–994, 2011.

[175] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie et T. G. Price, « Access path selection in a relational database management system », *in Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, p. 23–34, ACM, 1979.

[176] A. Bernstein, C. Kiefer et M. Stocker, « Optarq: A sparql optimization approach based on triple pattern selectivity estimation », rap. tech., University of Zurich, Department of, 2007.

[177] C. Weiss, P. Karras et A. Bernstein, « Hexastore: sextuple indexing for semantic web data management », *Proceedings of the VLDB Endowment*, vol. 1, p. 1008–1019, Aug. 2008.

[178] J. Groppe, S. Groppe, S. Ebers et V. Linnemann, « Efficient processing of sparql joins in memory by dynamically restricting triple patterns », *in Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, (New York, NY, USA), p. 1231–1238, ACM, 2009.

[179] T. M. Connolly et C. E. Begg, *Database Systems: A Practical Approach to Design, Implementation and Management (4th Edition)*. Pearson Addison Wesley, 2004.

[180] M. Jarke et J. Koch, « Query optimization in database systems », *ACM Comput. Surv.*, vol. 16, p. 111–152, 1984.

[181] R. Bellman, « Dynamic programming and lagrange multipliers », *Proceedings of the National Academy of Sciences of the United States of America*, vol. 42, no. 10, p. 767, 1956.

[182] F. Goasdoué, K. Karanasos, J. Leblay et I. Manolescu, « View selection in semantic web databases », computer science, INRIA, 2011. Rapport de recherche no. 7738.

[183] J. Pérez, M. Arenas et C. Gutierrez, « Semantics and complexity of sparql », *in The Semantic Web-ISWC 2006*, p. 30–43, Springer, 2006.

[184] F. Frasincar, G.-J. Houben, R. Vdovjak et P. Barna, « Ral: An algebra for querying rdf », *World Wide Web*, vol. 7, p. 83–109, mars 2004.

[185] R. Cyganiak, « A relational algebra for SPARQL », rap. tech., Digital Media Systems Laboratory, HP Laboratories Bristol, 2005.

[186] J. Yang, K. Karlapalem et Q. Li, « Algorithms for materialized view design in data warehousing environment », *in VLDB*, p. 136–145, 1997.

[187] K. Amira, *L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels*. Thèse doctorat, ENSMA - Poitiers, Décembre 2013.

[188] A. Boukorca, L. Bellatreche et S.-A. B. Senouci, « Votre plan d'exécution de requêtes est un circuit intégré : Changer de métier », *in EDA*, p. 133–148, 2013.

[189] A. C. Bloesch et T. A. Halpin, « Conquer: a conceptual query language », *in Conceptual Modeling –ER'96*, p. 121–133, Springer, 1996.

[190] D. Theodoratos et T. K. Sellis, « Designing data warehouses », *Data Knowl. Eng.*, vol. 31, p. 279–301, 1999.

[191] J. D. Ullman, *Principles of Database Systems, 2nd Edition*. Computer Science Press, 1982.

[192] E. Wong et K. Youssefi, « Decomposition - a strategy for query processing », *ACM Trans. Database Syst.*, vol. 1, p. 223–241, 1976.

[193] S. Chaudhuri et V. R. Narasayya, « An efficient cost-driven index selection tool for microsoft sql server », *in Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, (San Francisco, CA, USA), p. 146–155, Morgan Kaufmann Publishers Inc., 1997.

[194] J. U. H. Garcia-Molina et J. Wisdom, *Database Systems: The Complete Book*. Pearson Education Inc, 2009.

[195] J. Yang, K. Karlapalem et Q. Li, « A framework for designing materialized views in data warehousing environment », *in ICDCS*, p. 458–465, 1997.

[196] P. Toth et S. Martello, « Knapsack problems: Algorithms and computer implementations », *Discrete Mathematics and Optimization. Wiley*, 1990.

[197] L. Troiano, D. De Pasquale et P. Marinaro, « Genetic algorithms in java », 2014. URL: http://jenes.intelligentia.it.

[198] R. Hylock, « A comparison of heuristic algorithms to solve the view selection problem », *Information Systems*, p. 27, 2005.

[199] A. Shukla, P. Deshpande, J. F. Naughton *et al.*, « Materialized view selection for multidimensional datasets », *in VLDB*, vol. 98, p. 488–499, 1998.

[200] H. Gupta et I. S. Mumick, « Selection of views to materialize under a maintenance cost constraint », *in Database Theory-ICDT'99*, p. 453–470, Springer, 1999.

[201] K. Wilkinson et K. Wilkinson, « Jena property table implementation », *in In SSWS*, 2006.

[202] Y. Mengdong et G. Wu, « Caching intermediate result of sparql queries », *in 20th International World Wide Web Conference (WWW2011)*, p. 159, March/April 2011.

Annexe A

Requêtes du benchmarch LUBM utilisées

Cette annexe présente les requêtes du benchmark LUBM [30] que nous avons utilisées pour les expérimentations.

1. **SELECT** ?x **WHERE** {?x rdf:type ub:GraduateStudent.
   ?x ub:takesCourse http://www.Department0.University0.edu/GraduateCourse0.} ;

2. **SELECT** ?x ?y ?z **WHERE** {
   ?x rdf:type ub:GraduateStudent.
   ?x ub:undergraduateDegreeFrom ?y. ?x ub:memberOf ?z.
   ?y rdf:type ub:University. ?z rdf:type ub:Department.
   ?z ub:subOrganizationOf ?y.}

3. **SELECT** * **WHERE** {
   ?x rdf:type ub:Publication.
   ?x ub:publicationAuthor http://www.Department0.University0.edu/AssistantProfessor0.}

4. **SELECT** * **WHERE** {
   ?x rdf:type ub:FullProfessor.
   ?x ub:worksFor http://www.Department0.University0.edu.
   ?x ub:name ?y1.
   ?x ub:emailAddress ?y2.
   ?x ub:telephone ?y3. } ;

5. SELECT ?x **WHERE** { ?x rdf:type ub:Person).}

6. **SELECT** ?x **WHERE** { ?x rdf:type ub:Student.}

7. **SELECT** ?x **WHERE** {?x rdf:type ub:UndergraduateStudent>.
   ?x ub:takesCourse ?y.
   ?y rdf:type ub:Course.
   ?z rdf:type ub:AssistantProfessor.
   ?z ub:teacherOf ?y. } ;
8. **SELECT** ?x ?y **WHERE** {
   ?x rdf:type ub:UndergraduateStudent.
   ?x ub:memberOf ?z.
   ?x ub:emailAddress ?email.
   ?z rdf:type ub:Department.
   ?z ub:subOrganizationOf http://www.University0.edu.} ;

9. **SELECT** ?x ?y **WHERE** {?x rdf:type ub:UndergraduateStudent.
   ?y rdf:type ub:FullProfessor.
   ?z rdf:type ub:Course.
   ?x ub:advisor ?y.
   ?x ub:takesCourse ?z.
   ?y ub:teacherOf ?z.} ;

10. **SELECT** ?x **WHERE** {
    ?x rdf:type ub:Student. ?x ub:takesCourse GraduateCourse0. } ;

11. **SELECT** ?z **WHERE** {?z rdf:type ub:ResearchGroup.
    ?z ub:subOrganizationOf http://www.Department0.University0.edu. };

12. **SELECT** * **WHERE** { ?x rdf:type ub:AssistantProfessor.
    ?x ub:worksFor ?z.
    ?z rdf:type ub:Department.
    ?z ub:subOrganizationOf http://www.University0.edu.} ;

13. **SELECT** ?x **WHERE** {
    ?x rdf:type ub:Person. ?x ub:hasAlumnus http://www.University0.edu. } ;

14. **SELECT** * **WHERE** {?x rdf:type ub:UndergraduateStudent.};

Table des figures

Liste des tableaux

Glossaire

API : Application Programming Interface
BD : Base de données
BDR : Base de données relationnelle
$\mathcal{BDBO}$: Base de données à base ontologique
BDT : Base de données traditionnelle
BDO : Base de données objet
BDS : Base de données Sémantique
CC : Classe canonique
CNC : Classe non canonique
ISAE : Institut Supérieur de l'Aéronautique et de l'Espace
LIAS : Laboratoire d'Informatique, d'Application et de Systèmes
LISI : Laboratoire d'Informatique Scientifique et Industrielle
LO : Ontologie Locale
OC : Ontologie Conceptuelle
OCC : Ontologie Conceptuelle Canonique
OCNC : Ontologie Conceptuelle Non Canonique
OL : Ontologie Linguistique
OWL : Web Ontology Language
PLIB : Parts Library - Norme ISO 13584
RDF : Ressource Description Framework
RDFS : Ressource Description Framework Schema
SDB : SPARQL DataBase
SQL : Structured Query Language
SPARQL : Simple Protocol And RDF Query Language
TDB : Tuple DataBase
UML : Unified Modeling Language
URI : Uniform Resource Identifier
URL : Uniform Resource Locator
W3C : World Wide Web Consortium

Résumé

La forte volumétrie des données décrites par des ontologies a conduit à la naissance des bases de données à base ontologique ($\mathcal{BDBO}$). Les communautés industrielles et académiques se sont intéressées de près à cette technologie, où plusieurs solutions ont été proposées pour représenter, stocker les données sémantiques au sein des SGBD et manipuler ses données via des langages de requêtes comme SPARQL et OntoQL.

Parallèlement, la conception physique est devenue une étape primordiale dans le cycle de vie de conception des bases de données avancées. Durant cette phase, des structures d'optimisation de requêtes sélectionnées. Si de nombreux travaux ont été menés sur la conception physique dans le contexte des bases de données traditionnelles, peu se sont intéressés à la conception physique dans les $\mathcal{BDBO}$. Après une analyse approfondie des principales $\mathcal{BDBO}$ existantes, nous affirmons que leur conception physique est plus complexe. Cela est dû à leur diversité qui porte sur : (1) des formalismes supportés : chaque $\mathcal{BDBO}$ utilise un formalisme particulier pour définir ses ontologies (OWL, PLIB ou FLIGHT) ; (2) des modèles de stockage : une variété de modèles de stockage (représentation horizontale, spécifique, verticale, etc.) sont utilisés pour les $\mathcal{BDS}$ et (3) des architectures des SGBD cibles supportant ces bases. Trois types d'architecture existent : (a) architecture "deux quarts" qui est celle des BD traditionnelles, où les ontologies et les données sont stockées ensemble. (b) Le second type d'architecture qualifié de "trois quarts" sépare les ontologies des instances ontologiques; (c) la dernière architecture dite "quatre quarts" ajoute à la précédente une partie appelée méta-schéma.

Notons que le modèle de coût est la composante principale de la phase de conception physique. Il permet de guider la sélection des structures d'optimisation et quantifier sa qualité. Pour cette raison, nous avons développé des modèles de coûts mathématiques pour estimer le coût de chargement des données et le temps de réponse des requêtes en termes de nombre d'Entrées-Sorties entre le disque et la mémoire pour chaque type de $\mathcal{BDBO}$. Les résultats théoriques sont confrontés avec les résultats pratiques obtenus à partir de six $\mathcal{BDBO}$ dont trois industrielles (Oracle et IBM SOR, DB2RDF) et trois académiques (Jena, Sesame et OntoDB, développé au laboratoire LIAS de l'ISAE-ENSMA). Ces modèles de coûts ont été utilisés dans le processus de sélection des vues matérialisées, des structures d'optimisation populaires dans le contexte des bases de données. Nous avons proposé deux approches de matérialisation : une approche conceptuelle et une approche physique. Dans la première approche, la sélection des vues matérialisées est faite sur les classes et les propriétés utilisées par les requêtes d'interrogation (SPARQL). Dans l'approche physique, la sélection prend en compte l'architecture et les modèles de stockage du SGBD cible. Des expérimentations ont été conduites pour évaluer la qualité de nos approches en les confrontant avec les principaux travaux existants.

Conception physique de bases de données à base ontologique : le cas des Vues Matérialisées

Présentée par :

MBAIOSSOUM BERY LEOURO

Directeurs de Thèse :

Ladjel Bellatreche et Stéphane Jean

Résumé

La forte volumétrie des données décrites par des ontologies a conduit à la naissance des bases de données à base ontologique (BDBO). Plusieurs communautés se sont intéressées à cette technologie et ont proposé des solutions pour persister les données sémantiques dans des SGBD.

Parallèlement, la conception physique est devenue une étape primordiale dans le cycle de vie de conception des bases de données (BD). Durant cette phase, des structures d'optimisation sont sélectionnées. Si de nombreux travaux ont été menés sur la conception physique dans le contexte des BD traditionnelles, peu se sont intéressés à la conception physique dans les BDBO qui est plus complexe. Cette complexité est due à la diversité des BDBO qui porte sur des formalismes supportés, des modèles de stockage et des architectures utilisés.

Pour guider la sélection des structures d'optimisation et mesurer sa qualité, nous avons développé un modèle de coût pour estimer le coût des requêtes dans les BDBO. Les résultats théoriques sont confrontés avec les résultats pratiques obtenus à partir de six BDBO dont trois industrielles (Oracle et IBM SOR, DB2RDF) et trois académiques (Jena, Sesame et OntoDB du LIAS de l'ISAE-ENSMA). Ce modèle de coût a été utilisé dans le processus de sélection des vues matérialisées. Nous avons proposé deux approches de matérialisation : une approche conceptuelle où la sélection des vues matérialisées est faite sur les classes et les propriétés utilisées par les requêtes et une approche simulée où la sélection prend en compte la diversité des BDBO. Des expérimentations ont été conduites pour évaluer la qualité de nos approches en les confrontant avec les principaux travaux existants.

Mots-clés : Ontologies (informatique), Bases de données--Conception, Bases de données--Interrogation, Architecture—Informatique, Diversité des bases de données à base ontologique, Optimisation des requêtes, Conception physique, Vues matérialisées

Abstract

The high volume of data described by ontologies led to the creation of Ontology-Based Database (OBDB). Many communities are interested in this technology and have proposed solutions to persist semantic data in DBMS.

Meanwhile, the physical design has become an essential step in the life cycle of database design, in which optimization structures are selected. While many studies have been conducted on the physical design in the context of traditional databases, few have focused on the physical design in OBDB which is more complex. This complexity is due to the diversity of OBDB which focuses on formalisms supported, storage models and architectures used.

To guide the selection of optimization structures, we have developed a cost model to estimate the cost of queries in OBDB. The theoretical results are compared with the practical results obtained from six OBDB including three industrial (Oracle, IBM SOR and DB2RDF) and three academic (Jena, Sesame and OntoDB of the LIAS Lab of ISAE-ENSMA). This cost model was used in the materialized views selection process. We proposed two approaches of materialized views selection: a conceptual approach where the selection of materialized views is made on the classes and properties used by queries and a simulated approach where the selection takes into account the diversity of OBDB. Experiments were conducted to evaluate the quality of our approaches and compare them with the main existing work.

Mots-clés : Ontologies (Information retrieval), Database design, Querying (Computer science), Architecture--Data processing, Ontology-based database diversity, Query optimisation, Physical design, Materialized views.