Comparaison Théorique et Empirique de Systèmes de Bases de Données Sémantiques

Bery Mbaiossoum* — **Ladjel Bellatreche** * — **Stéphane Jean** * — **Mickael Baron***

**LIAS/ISAE-ENSMA - Université de Poitiers 86960 Futuroscope Cedex France*
{mbaiossb, bellatreche, jean, baron}@ensma.fr

RÉSUMÉ. *L'utilisation d'ontologies est de plus en plus fréquente dans les systèmes d'information. Lorsque la quantité de données décrites par des ontologies dépasse les capacités de gestion en mémoire centrale, les concepteurs d'un système d'information doivent recourir à des bases de données particulières, appelées bases de données sémantiques ($\mathcal{BDS}$). Ce choix s'avère aujourd'hui délicat car les $\mathcal{BDS}$ sont diverses que ce soit en termes d'architecture, de modèles de stockage utilisés ou de formalismes d'ontologies supportés. Cet article vise à aider les concepteurs des applications manipulant un gros volume de données sémantiques à choisir une $\mathcal{BDS}$ adaptée. Nous présentons d'abord un état des lieux sur les ontologies, leurs formalismes, les modèles de stockage et les architectures cibles des Systèmes de Gestion de Base de Données (SGBD) associés. Puis nous proposons un modèle pour les $\mathcal{BDS}$ qui capture leur diversité. Enfin, nous présentons une comparaison à la fois théorique, à l'aide d'un modèle de coût mathématique, et empirique des performances de six $\mathcal{BDS}$ issues du milieu industriel et académique.*

ABSTRACT. *Ontologies are more and more used in information systems. When the size of data described by ontologies can not be manage in main memory, information system designers have to use specialized databases called semantic databases. As these databases use different architectures, storage layouts and ontology languages, this task is complex. In this paper, we propose a comparison and evaluation of these systems in order to help information system designers identify a relevant semantic database. We first present a state of the art on ontologies, their languages and storage layouts and architectures of associated databases. Then we propose a model for semantic databases that capture their diversity. Finally we provide a cost model and performance evaluation of six industrial and academic semantic databases.*

MOTS-CLÉS : *Ontologie, Base de données, Modèle de coût, Evaluation de performances*

KEYWORDS: *Ontology, Database, Cost model, Performance evaluation*

1. Introduction

Depuis de nombreuses années, la notion d'ontologie est fortement utilisée dans les systèmes d'information ($\mathcal{SI}$) pour contribuer à la résolution de problèmes variés tels que l'intégration de données ou la recherche d'information. Dans le domaine académique, cette utilisation se remarque par la présence chaque année de sessions et tutoriels dans des conférences comme CAISE (International Conference on Advanced Information Systems Engineering) ou ER (International Conference on Conceptual Modeling). Dans le domaine industriel, de nombreuses ontologies sont utilisées très largement (par exemple, Uniprot (`http://www.uniprot.org/`) dans le domaine de la protéomique) et sont même standardisées (par exemple, la norme IEC 61360-4 pour les composants électroniques ou la norme ISO 23584 dans le domaine de l'optique et l'optronique). Cet intérêt se remarque également par le support d'ontologies dans des outils commerciaux tels que les SGBD d'Oracle ou d'IBM. Avec l'utilisation de plus en plus importante d'ontologies dans les $\mathcal{SI}$, que ce soit dans le milieu académique ou industriel, il est donc important de s'intéresser aux outils permettant la persistance et l'exploitation d'ontologies. La taille des données manipulées dans un $\mathcal{SI}$ dépassant généralement les capacités de gestion en mémoire centrale, nous nous intéressons dans cet article à l'étude de systèmes de gestion de bases de données particulières, appelées *bases de données sémantiques* ($\mathcal{BDS}$), qui permettent le stockage à la fois des données et des ontologies qui en décrivent le sens. Cette étude peut contribuer à l'évaluation d'un $\mathcal{SI}$ puisque la qualité de celui-ci dépend en partie de la qualité des outils utilisés.

De nombreuses $\mathcal{BDS}$ ont été proposées dans le milieu académique telles que OntoDB (Dehainsala *et al.*, 2007), Sesame (Broekstra *et al.*, 2002) ou Jena (Carroll *et al.*, 2004) et apparaissent actuellement dans le milieu industriel comme par exemple Oracle Semantic Technology (Wu *et al.*, 2008) ou IBM SOR (Scalable Ontology Repository) (Lu *et al.*, 2007). Comparés au développement des bases de données traditionnelles, ces systèmes ont été développés et proposés dans un espace de temps très court. Le développement de nombreuses $\mathcal{BDS}$ résulte principalement de (1) *la diversité des formalismes d'ontologie* : chaque $\mathcal{BDS}$ utilise un formalisme particulier pour définir ses ontologies (par exemple, OWL (Bechhofer *et al.*, 2004), RDFS (Brickley *et al.*, 2002) ou PLIB (Pierra, 2008)) (2) *la diversité des modèles de stockage* : contrairement aux bases de données traditionnelles, où le modèle logique est stocké selon une approche relationnelle, les $\mathcal{BDS}$ utilisent une variété de modèles de stockage (représentation horizontale, binaire, etc.) pour stocker deux niveaux de modélisation : le niveau ontologie et le niveau des instances ontologiques et (3) *la diversité des architectures cibles* utilisées par le SGBD : une $\mathcal{BDS}$ peut utiliser un seul ou plusieurs schémas de base de données pour stocker l'ensemble des données.

La diversité des $\mathcal{BDS}$ proposées rend le choix des concepteurs des $\mathcal{SI}$ nécessitant des $\mathcal{BDS}$ difficile. Actuellement, la plupart des comparaisons effectuées s'intéresse uniquement aux performances de ces $\mathcal{BDS}$ sans prendre en compte ni leur possibilité de modélisation ni l'impact de leur architecture sur les performances offertes. Une autre dimension non traitée par la communauté, est l'absence de comparaison basée sur des modèles de coût. Définir un modèle de coût n'est pas une chose aisée (Gruser,

1996), car il n'existe pas un modèle unique mais une multitude de modèles adaptés à des systèmes très particuliers. Pourtant, un modèle de coût représente la composante principale d'un optimiseur physique d'une base de données (Gruser, 1996). Ils peuvent en effet avoir plusieurs rôles :

– *outil pour choisir le meilleur plan d'exécution pour une requête donnée.* Le choix du plan d'exécution optimal parmi les nombreux plans possibles pour exécuter une requête s'appuie sur un modèle de coût ;

– *outil pour l'optimisation dynamique (adaptative) des requêtes.* Dès qu'il y a des changements au niveau des paramètres sur lesquels se base l'optimiseur pour sélectionner les plans optimaux d'une requête, le modèle de coût peut être utilisé pour trouver de nouveaux plans ;

– *outil pour guider les algorithmes de recherche et de sélection.* Dans les bases de données ou les entrepôts de données, plusieurs problèmes d'optimisation ou de sélection sont difficiles à résoudre. Deux bons exemples sont les problèmes de sélection des vues et des index. Pour les résoudre, des heuristiques ont été définies en se basant sur des modèles de coût.

Comme nous le constatons, les modèles de coût jouent un rôle essentiel dans l'optimisation des requêtes et ont été peu étudiés dans le contexte des $\mathcal{BDS}$.

Pour compléter l'étude théorique sur les $\mathcal{BDS}$, nous proposons tout d'abord une formalisation de cette notion afin de fournir un cadre formel pour leur comparaison ainsi que pour clarifier la diversité des $\mathcal{BDS}$ existantes. Nous montrons que cette formalisation permet de représenter une variété de $\mathcal{BDS}$ en comparant six $\mathcal{BDS}$ dont trois issues du monde industriel (Oracle, DB2RDF et IBM SOR) et trois du monde académique (OntoDB, Jena et Sesame). La formalisation proposée permet de comparer l'architecture et les modèles de stockage de ces $\mathcal{BDS}$. Pour compléter cette proposition nous proposons une comparaison des performances de ces $\mathcal{BDS}$. Cette comparaison est d'abord faite de manière théorique par la définition d'un modèle de coût qui, à notre connaissance, est le premier proposé pour les $\mathcal{BDS}$. Puis nous présentons une évaluation des performances des six $\mathcal{BDS}$ de manière empirique par la mesure du temps de chargement et du temps de traitement des requêtes du banc d'essai LUBM (Lehigh University Benchmark). Cet article est une extension de celui présenté à la session *Evaluation des Systèmes d'Information* de la conférence INFORSID 2012 (Mbaiossoum *et al.*, 2012). Comparé à la proposition initiale, cet article détaille le modèle de $\mathcal{BDS}$ proposé, définit un modèle de coût pour les $\mathcal{BDS}$ et inclut la comparaison détaillée de nouvelles $\mathcal{BDS}$ (DB2RDF, Sesame et Jena).

Notre travail peut être classé aux côtés de ceux de (Beckett, 2002; Sakr *et al.*, 2009; Lee, 2004; Faye *et al.*, 2012; Hertel *et al.*, 2009; Stegmaier *et al.*, 2009). Dans (Beckett, 2002), une étude est faite sur les $\mathcal{BDS}$ libres (open source) en illustrant leurs fonctionnalités et leur maturité. Dans (Sakr *et al.*, 2009), les modèles relationnels pour le stockage de données RDF sont présentés. Dans (Magkanaraki *et al.*, 2002; Hertel *et al.*, 2009), quelques $\mathcal{BDS}$ sont étudiées en mettant l'accent sur quelques fonctionnalités comme le stockage, le raisonnement et les langages d'interrogation.

Dans (Faye *et al.*, 2012) les différents modèles de stockage de données RDF sont caractérisés. Par contre, ces travaux ne proposent pas de comparaison expérimentale des performances des $\mathcal{BDS}$ étudiées. Dans (Lee, 2004) une étude comparative est effectuée en terme de temps de chargement de données. Cet article n'a par contre pas proposé d'évaluation en terme de performance des requêtes ce qui est fait dans (Stegmaier *et al.*, 2009; Liu *et al.*, 2005). A la différence de ces différents travaux, nous proposons une étude qui compare les différents modèles de stockage et architectures des $\mathcal{BDS}$ à la fois de manière théorique, par la proposition d'un modèle de coût, et de manière empirique par l'expérimentation de $\mathcal{BDS}$ académiques et industrielles.

Cet article est composé de sept sections. Pour commencer, la section 2 définit les notions principales liées aux ontologies et à leurs formalismes. La diversité des $\mathcal{BDS}$ est ensuite montrée dans la section 3 en décrivant les modèles de stockage (pour les instances et pour les ontologies) et architectures utilisés. A partir de l'étude faite dans les sections précédentes, la section 4 introduit une formalisation des $\mathcal{BDS}$ qui est illustrée sur une $\mathcal{BDS}$. Les sections 5 et 6 sont consacrées à l'évaluation des performances des $\mathcal{BDS}$ d'un point de vue théorique, par la définition d'un modèle de coût, puis d'un point de vue empirique. Enfin, la section 7 conclut cet article.

2. La notion d'ontologie et les formalismes associés

Les ontologies sont des conceptualisations qui permettent 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 (Gruber, 1993), une ontologie est une spécification explicite d'une conceptualisation. Cette définition a été enrichie par Jean et al. (Jean *et al.*, 2007) 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 met en avant trois caractéristiques qui distinguent une ontologie de domaine des autres modèles informatiques tels que les modèles conceptuels et les modèles de connaissance. Une ontologie est une représentation :

– *formelle* : exprimée dans un langage formalisé comme par exemple OWL (Bechhofer *et al.*, 2004), RDFS (Brickley *et al.*, 2002) ou PLIB (Pierra, 2008) 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 d'une communauté ;

– *référençable* : toute entité ou relation décrite dans l'ontologie peut être directement référencée par un symbole (« identifiant »), à partir de n'importe quel contexte, afin d'expliciter la sémantique de l'élément référençant.

Gruber (Gruber, 1993) 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.

Plusieurs formalismes ont été proposés pour concevoir des ontologies qui diffèrent selon l'objectif poursuivi. Certains formalismes ont pour but la définition d'ontologies visant à faciliter la gestion et l'échange de données. Ces formalismes cherchent à définir la sémantique des concepts de manière unique et précise et se focalisent donc sur les concepts primitifs. Les modèles RDF-Schema (Brickley *et al.*, 2002) et PLIB (Pierra, 2008) sont des exemples de formalismes permettant de décrire de telles ontologies. RDF-Schéma est un modèle issu du Web Sémantique, étendant le modèle RDF par des constructeurs permettant la définition de classes et de propriétés. Le modèle PLIB (Parts LIBrary) (Pierra, 2008) est un modèle défini pour la description des différentes catégories de composants industriels et de leurs instances. D'autres formalismes, comme par exemple OWL (Bechhofer *et al.*, 2004), ont été proposés pour pouvoir définir des ontologies permettant d'établir des correspondances entre vocabulaires et offrant ainsi des possibilités de déduction et d'inférences. Ces formalismes définissent ainsi des concepts primitifs et définis.

3. Stockage des ontologies dans des bases de données

Comme nous l'avons indiqué précédemment, la nécessité de stocker les ontologies et leurs instances au sein d'une base de données résulte du volume de données ontologiques devenant de plus en plus important. Plusieurs $\mathcal{BDS}$ ont été proposées pour permettre le stockage des ontologies et de leurs instances. Nous détaillons dans ce qui suit les différents modèles de stockage et architectures des $\mathcal{BDS}$.

3.1. *Les modèles de stockage des* $\mathcal{BDS}$

Trois principales approches sont utilisées pour la représentation des ontologies et de leurs instances au sein des bases de données. Ces trois approches sont illustrées sur la figure 2 en montrant le stockage du sous-ensemble de l'ontologie LUBM (Y. Guo *et al.*, 2005) présentée figure 1.

– *Approche verticale :* elle 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. Cette représentation facilite l'insertion de nouveaux triplets. Son interrogation est complexe car elle peut nécessiter plusieurs opérations d'auto-jointure. La $\mathcal{BDS}$ Sesame et celle d'oracle utilisent cette représentation pour le stockage des instances ontologiques.

– *Approche binaire* : elle 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 la manière dont est représenté l'héritage : (1) une table unique pour toutes les classes

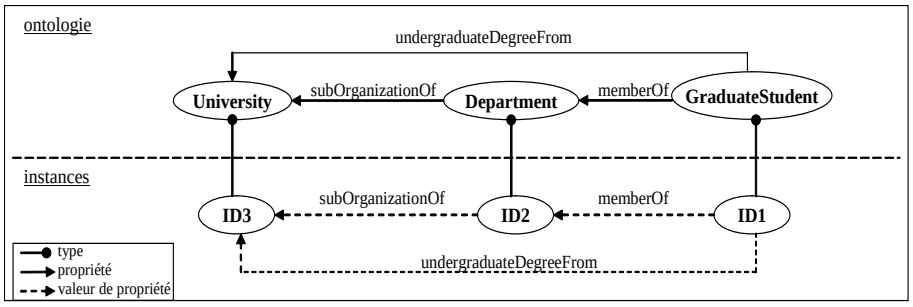
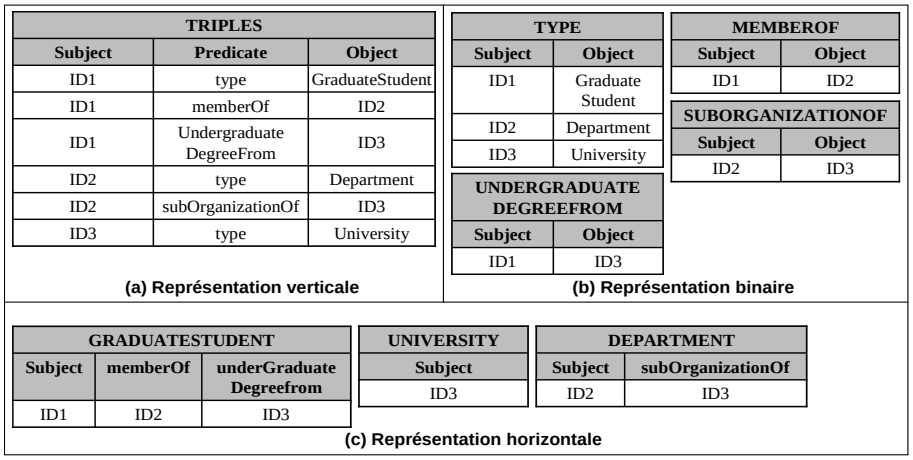


Figure 1. *Sous-ensemble de l'ontologie LUBM*



TRIPLES

Subject	Predicate	Object
ID1	type	GraduateStudent
ID1	memberOf	ID2
ID1	Undergraduate DegreeFrom	ID3
ID2	type	Department
ID2	subOrganizationOf	ID3
ID3	type	University

(a) Représentation verticale

TYPE

Subject	Object
ID1	Graduate Student
ID2	Department
ID3	University

MEMBEROF

Subject	Object
ID1	ID2

SUBORGANIZATIONOF

Subject	Object
ID2	ID3

UNDERGRADUATE DEGREEFROM

Subject	Object
ID1	ID3

(b) Représentation binaire

GRADUATESTUDENT

Subject	memberOf	underGraduate Degreefrom
ID1	ID2	ID3

UNIVERSITY

Subject
ID3

DEPARTMENT

Subject	subOrganizationOf
ID2	ID3

(c) Représentation horizontale

Figure 2. *Les trois principaux modèles de stockage utilisés par les $\mathcal{BDS}$*

de l'ontologie, (2) une table par classe avec héritage de table (si une base de données relationnelle-objet est utilisée) et (3) une table par classe sans héritage de table. La $\mathcal{BDS}$ SOR utilise cette approche pour le stockage des ontologies et de leurs instances.

– *Approche horizontale* : cette approche est similaire à la représentation traditionnelle utilisée par les SGBD relationnels. Elle consiste à créer une table par classe de l'ontologie. Cette table possède une colonne pour chaque propriété qui est valuée par au moins une instance de la classe correspondante. La $\mathcal{BDS}$ OntoDB utilise l'approche horizontale pour le stockage de son modèle ontologique et également de ces instances.

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

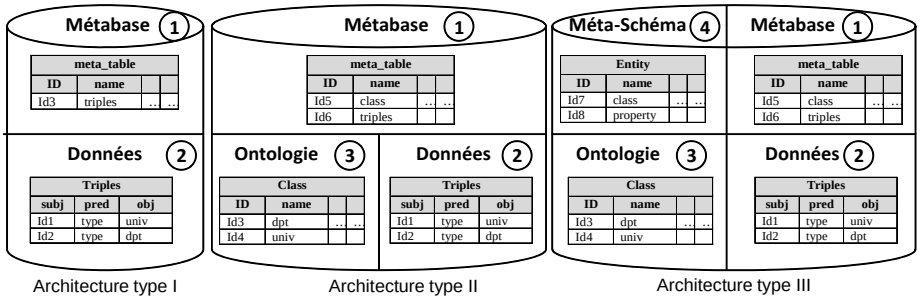


Figure 3. *Les différentes architectures des $\mathcal{BDS}$*

3.2. *Les architectures des $\mathcal{BDS}$*

Trois principales architectures sont utilisées par les $\mathcal{BDS}$ (illustrées sur la figure 3) :

– *architecture de type I (ou deux quarts)* : comme les bases de données classiques, cette architecture utilise deux parties : la partie données et la métabase. La partie *données* stocke les instances ontologiques mais également le schéma de l'ontologie (classes, propriétés, etc.). Cette architecture impose ainsi l'utilisation du même modèle de stockage pour les ontologies et pour les instances ontologiques. Jena et Oracle utilisent cette architecture en stockant l'ontologie (modèle et instances) sous forme de triplets RDF (approche verticale). Cette architecture présente l'inconvénient de stocker l'ensemble des informations dans une seule table et ainsi les requêtes s'expriment en termes d'auto-jointures sur cette table volumineuse ;

– *architecture de type II (ou trois quarts)* : dans cette architecture, les instances ontologiques et les ontologies sont stockées dans deux schémas différents. Cette architecture scinde ainsi la base de données en trois parties : métabase, schéma de stockage des ontologies et schéma de stockage des instances ontologiques. IBM SOR (Lu *et al.*, 2007) est un exemple de $\mathcal{BDS}$ qui utilise cette architecture. Elle présente l'inconvénient d'utiliser un schéma de stockage des ontologies figé ce qui ne permet pas d'utiliser des constructeurs issus de formalismes d'ontologies autres que celui supporté initialement par la $\mathcal{BDS}$;

– *architecture de type IIII (ou quatre quarts)* : cette architecture a été proposée dans le cadre du projet OntoDB (Dehainsala *et al.*, 2007) pour répondre au besoin de flexibilité du modèle d'ontologie utilisé dans la $\mathcal{BDS}$. Cette architecture propose pour cela une partie nommée méta-schéma qui joue, pour les ontologies, le même rôle que celui joué par la métabase pour les données. Cette partie permet un accès générique au modèle ontologique ainsi que l'introduction de nouveaux constructeurs issus de différents formalismes ontologiques.

4. Formalisation et critères de comparaison des $\mathcal{BDS}$

4.1. *Un modèle pour les* $\mathcal{BDS}$

Pour représenter la diversité des structures de $\mathcal{BDS}$, il est nécessaire de proposer une structure générique permettant de représenter cette diversité en termes de : modèle ontologique, instances ontologiques, schéma de stockage du modèle ontologique, schéma de stockage des instances ontologiques et architecture de $\mathcal{BDS}$. Nous proposons la formalisation $< MO, I, Sch, Pop, SM_{MO}, SM_{Inst}, Ar >$, où :

– MO représente un modèle ontologique générique. Ce modèle est formalisé par le 5-uplet suivant : $< C, P, Applic, Ref, Formalisme >$

- C représente les classes du modèle ontologique.

- P représente les propriétés du modèle ontologique.

- $Applic : C \rightarrow 2^P$ lie chaque classe aux propriétés dont elle est domaine.

- $Ref : C \rightarrow (operateur, Exp(C))$ est une fonction qui associe à chaque classe un opérateur (d'inclusion ou d'équivalence) et une expression sur d'autres classes. Les expressions définies pour les ontologies OWL basées sur les logiques de description présentent, de 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 cardinalités ($\geq nR.C$, $\leq nR.C$). *Exp* peut être la fonction d'identité qui associe à une classe la même classe (la classe se définit par elle-même comme la plus grande classe de la hiérarchie "Thing").

- $Formalisme$: le nom du formalisme du modèle ontologique adopté.

Par exemple, une ontologie PLIB sera définie comme une instance du 5-uplets suivant : $< C, P, Applic, < OntoSub, Is\_A >, PLIB >$. 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.

– I : représente l'ensemble des instances ontologiques

– $Sch : C \rightarrow 2^P$ associe à chaque classe l'ensemble des propriétés pour lesquelles les instances de cette classe sont valuées.

– $Pop : E \rightarrow 2^I$ associe à chaque classe ses instances de l'ensemble I.

– SM_{MO} : le modèle de stockage du modèle ontologique.

– SM_{Inst} : le modèle de stockage des instances ontologiques.

– Ar : le type d'architecture de la base (Type I, II ou III).

Pour illustrer cette formalisation, la $\mathcal{BDS}$ Oracle est représentée par : $< MO :< C, P, Applic, Oprateurs(RDFS,\ OWLSIF\ et\ OWLPrime), RDFS, OWLSIF\ ou\ OWLPrime >, InstancesRDF, Sch, TablesRDF\_link\ et\ RDF\_values, Vertical, Vertical, TypeI >$, où $OWLSIF$ et $OWLPrime$ sont des sous-ensembles d'OWL définis par Oracle pour sa technologie sémantique,

RDF_link est la table utilisée pour stocker les triplets et RDF_values est celle utilisée pour stocker les identifiants et leur valeurs.

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

4.2. *Comparaison détaillée des $\mathcal{BDS}$*

Pour que notre étude soit la plus complète possible, nous avons choisi des $\mathcal{BDS}$ fortement utilisées et venant des différentes communautés. Ainsi, nous nous intéressons à six $\mathcal{BDS}$ dont trois issues du monde de la recherche (OntoDB, Sesame et Jena) et trois autres issues du monde industriel (oracle, DB2RDF et SOR d'IBM).

4.2.1. *Présentation des $\mathcal{BDS}$*

4.2.1.1. Base de données sémantique OntoDB

OntoDB (Dehainsala *et al.*, 2007) a été conçue par le Laboratoire d'Informatique et d'Automatique des Systèmes (LIAS). Une première monture est implantée sur le SGBD PostgreSQL. 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 étant de type III, elle dispose d'un méta-schéma susceptible de prendre en compte tout schéma d'ontologie. Cela permet de manipuler aussi les ontologies RDF(S), OWL et autres. OntoDB utilise un modèle de stockage horizontal : une table est créée pour chaque classe de l'ontologie, ses colonnes correspondent à un sous-ensemble des propriétés applicables de la classe. Il est doté d'un langage de requête nommé OntoQL (Jean *et al.*, 2006). OntoDB a été utilisée dans des projets industriels et ANR (par exemple le projet ANR DaFOE4App : Differential And Formal Ontologies Editor For Applications ou E-wok hub). Cette utilisation a été souvent faite pour manipuler des données techniques qui sont représentées par le formalisme PLIB.

4.2.1.2. La Base de données sémantique d'Oracle

Oracle a intégré un support pour les langages RDF et OWL dans son SGBD pour permettre à ses clients de bénéficier d'une plateforme 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. Oracle a défini deux sous-classes d'OWL DL qui sont exploitées dans les processus d'inférences (Wu *et al.*, 2008) : OWLPrime et OWLSIF. Oracle utilise une architecture de type I et un modèle de stockage en triplet. La table de triplet utilisée a été améliorée par une décomposition (normalisation).

4.2.1.3. Les Bases de données sémantique d'IBM : SOR et DB2RDF

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. 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 aux inférences sur les ontologies et leurs instances qui s'appuie sur le SGBD DB2. Dans le cadre de notre travail, nous nous sommes intéressés à l'outil SOR (Lu *et al.*, 2007) qui fait partie de IODT. Il utilise une architecture type II pour le stockage de l'ontologie et de ses instances. Il comprend un raisonneur et un outil de recherche.

Parmi les travaux récents de IBM sur l'exploitation de la sémantique des données, on trouve le système DB2RDF (IBM, 2012). IBM l'a intégré dans la nouvelle version de son SGBD (DB2 10.1). DB2RDF utilise la représentation verticale pour stocker ses données (schéma et instances). Ce modèle de stockage est enrichi avec plusieurs tables pour optimiser les traitements. En particulier, comme oracle, DB2RDF utilise des identifiants pour éviter de manipuler des longues chaînes d'URI. DB2RDF fournit également un certain nombre de fonctions pour la gestion des données ontologiques dans un environnement Java (fonctionnalités basées sur les paquetages de JENA). DB2RDF supporte le langage SPARQL et utilise comme format de données le format N-Triple. Son architecture est type I. Contrairement à SOR, DB2RDF ne dispose pas d'un moteur d'inférence.

4.2.1.4. La Base de données sémantique Sesame

La $\mathcal{BDS}$ Sesame est un framework de gestion des données sémantiques développé par Aduna, un éditeur néerlandais de logiciels. Sesame permet le stockage et l'interrogation des ontologies et de leurs instances définies avec les formalismes d'ontologies RDF, RDFS ou OWL. Ce système permet de faire des inférences sur les données et peut être déployé sur des systèmes de stockage variés : mémoire centrale, systèmes de fichiers, bases de données relationnelles en mémoire, etc. Il dispose de son propre langage d'interrogation (SeRQL) mais supporte aussi le langage de requêtes SPARQL. Pour le stockage de données, Sesame utilise la représentation verticale et la représentation binaire. Nous avons considéré ces deux représentations dans les évaluations des performances (cf. section 6).

4.2.1.5. La Base de données sémantique 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. Il fournit un environnement de programmation pour les formalismes d'ontologies RDF, RDF Schema (RDFS) et OWL. Jena utilise comme langage de requêtes SPARQL. Il propose un moteur d'inférences à base de règles. Comme la $\mathcal{BDS}$ Sesame, Jena propose deux façons de rendre persistantes les données : (1) *TDB (Tuple Data Base)*, un stockage fondé sur le système de gestion des fichiers ou la mémoire centrale et (2) SDB (SPARQL Data Base), un système de stockage utilisant une base

de données standard. Nous utilisons la version SDB par la suite puisque nous nous intéressons aux $\mathcal{BDS}$. Il existe également deux versions de Jena : la première version appelée Jena1 qui utilise la représentation verticale (table de triplets) (B.McBride, 2001) et la deuxième version, Jena2, qui utilise une représentation spécifique (*property-table* et *class-property-table*) (Wilkinson, 2006). Nous utiliserons Jena2 pour notre travail.

4.2.2. *Critères de comparaison détaillés des $\mathcal{BDS}$*

Pour détailler notre comparaison des $\mathcal{BDS}$, nous avons identifié quelques critères de comparaison essentiels (en plus de ceux évoqués précédemment tels que l'architecture ou le modèle de stockage). Ces critères récapitulés dans le tableau 1 sont les suivants.

– *Support linguistique :* cette caractéristique indique si une $\mathcal{BDS}$ est capable de prendre en compte le fait que les concepts d'une ontologie puissent être définis dans plusieurs langues naturelles.

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

– *Gestion des versions et évolutions des ontologies :* les ontologies étant susceptibles d'évoluer, une $\mathcal{BDS}$ doit permettre d'intégrer et de gérer différentes versions d'une même ontologie. Nous notons que cette caractéristique n'est pas prise en compte dans la plupart des $\mathcal{BDS}$ utilisées ; seule la $\mathcal{BDS}$ OntoDB est prédisposée au versionnement des ontologies.

– *Moteur d'inférences :* ils permettent de réaliser les déductions inhérentes à une ontologie. Certaines $\mathcal{BDS}$ disposent de leurs propres moteurs d'inférences pour la déduction de nouvelle données à partir de celles qui sont disponibles (Oracle, SOR). D'autres font usage de moteurs d'inférences ontologiques existant comme Racer, Pellet, Fact, Fact++ (par exemple, Jena ou Sesame).

– *Base de règles utilisateur :* parmi les $\mathcal{BDS}$ qui disposent de moteurs d'inférences, certaines offrent à l'utilisateur la possibilité de créer sa propre base des règles. C'est le cas d'Oracle, de Jena et de Sesame.

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

Dans cette section, nous avons proposé un modèle et des critères de comparaison pour pouvoir comparer les $\mathcal{BDS}$ de manière globale et également de manière plus détaillée. Les $\mathcal{BDS}$ ayant été conçues pour le passage à l'échelle en terme de volumétrie

des ontologies et des instances, il est également important d'étudier les performances des $\mathcal{BDS}$ actuelles. Tout comme cela a été fait pour les bases de données relationnelles, nous proposons une comparaison théorique à travers un modèle de coût (section 5), puis une comparaison empirique des six $\mathcal{BDS}$ considérées pour valider notre modèle de coût (section 6).

Caractéristiques	**Oracle**	**SOR**	**OntoDB**	**Jena**	**Sesame**	**Db2rdf**
Formalisme	RDF, OWL	OWL	PLIB	RDF, OWL	RDF, OWL	RDF
Support linguistique	oui	oui	oui,	non	non	non
Langage de requêtes	sql, sparql	sparql	ontoql, sparql	rql, rdql, sparql	serql, sparql	sql, sparql
Versionnement	non	non	oui	non	non	non
Moteur d'inférence	oui	oui	non	oui	oui	non
Règles utilisateur	oui	non	non	oui	oui	non
Consistance	oui	oui	oui	oui	oui	oui
SM_{MO}	V	S	S	V, S	V, S	V.
SM_{Isnst}	V	B	H	V, S	V, B	V
Architecture	Type I	Type II	Type III	Type I	Type I	Type I
SGBD	Oracle	DB2, Derby	Postgres	Oracle, Postgres, mysql	Oracle, Postgres, mysql	DB2

Tableau 1. *tableau comparatif des $\mathcal{BDS}$ etudiées (V : vertical, B : binaire, H : horizontal, S : spécifique)*

5. Comparaison théorique des performances des $\mathcal{BDS}$: modèle de coût

Nous proposons une fonction de coût qui permet de faire une estimation des performances des requêtes dans les $\mathcal{BDS}$. 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 fichiers intermédiaires et les coûts de calcul. Suivant les hypothèses classiques faites sur les modèles de coût, nous supposons que les coûts de calculs sont négligeables au regard des coûts d'accès, qu'il existe des statistiques sur les données (par exemple, le nombre d'instances par classe), que la distribution des valeurs est uniforme et que les attributs sont indépendants les uns des autres.

Avant de proposer notre fonction de coût, nous faisons remarquer qu'il existe des $\mathcal{BDS}$ saturées et des $\mathcal{BDS}$ non saturées. Une $\mathcal{BDS}$ est dite saturée si elle contient en plus des instances ontologiques initiales, des instances déduites, sinon elle est dite non saturée. Certaines $\mathcal{BDS}$ sont automatiquement saturées lors du chargement de données, c'est le cas de SOR d'IBM, d'autres comme Jena et Sesame peuvent être

saturées à la demande lors du chargement, en utilisant un moteur d'inférences. On trouve aussi des $\mathcal{BDS}$ 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. Notre fonction de coût se base sur les $\mathcal{BDS}$ saturées. Pour l'appliquer à des $\mathcal{BDS}$ non saturées faisant des déductions, il faut lui ajouter le coût de déduction lors de l'interrogation.

5.1. *Paramètres de la fonction de coût*

Soit q une requête et *bds* une base de données sémantiques sur laquelle q sera exécutée et *cout* la fonction de coût d'accès. En terme d'architecture, nous pouvons remarquer que, par rapport au modèle de coût classique, le modèle de coût des $\mathcal{BDS}$ doit prendre en compte les coûts d'accès aux différents niveaux de modélisation :

$$
\begin{aligned}
&Type\ I: \ \ cout(q, bds) = cout(métabase) + cout(données)\\
&Type\ II: \ cout(q, bds) = cout(métabase) + cout(ontologie) + cout(données)\\
&Type\ III: cout(q, bds) = cout(métabase) + cout(ontologie) + cout(données)\\
&\qquad\qquad\qquad\qquad + cout(métaschéma)
\end{aligned}
$$

Habituellement, le coût d'accès à la métabase *(cout(métabase))* est jugé négligeable du fait que la métabase peut être gardée en mémoire (buffer). De même, l'ontologie et le formalisme d'ontologie sont généralement de tailles négligeables comparées à la taille des instances. Par exemple, l'ontologie Uniprot est constituée de quelques classes et propriétés (stockage en kilo-octets) alors que la partie données est constituée de millions d'instances (centaines de giga-octets). Donc, dans toutes les architectures, le coût peut se résumer au coût d'accès aux données en nombre d'entrées/sorties c'est-à-dire $cout(q, bds) = cout(donnes)$. Ce coût est largement influencé par les opérations exécutées dans les requêtes et notamment la projection, la sélection et la jointure. Ainsi, notre fonction de coût se base sur ces trois opérations. Les paramètres et notations utilisés pour définir cette fonction sont présentés dans le tableau 2.

5.2. *Template de requêtes*

Soit $C_1, \ldots, C_n$ des classes de l'ontologie et $p_{11}, \ldots, p_{nn}$ des propriétés. Nous considérons que nos requêtes sont définies suivant le modèle ci-dessous qui peut être exprimé dans la plupart de langages d'interrogation des ontologies :

$$
\begin{aligned}
&(?id_1, type, C_1)\ (?id_1, p_{11}, ?val_{11})\ \cdots\ (?id_1, p_{n1}, ?val_{n1})\ [FILTER()]\\
&(?id_2, type, C_2)\ (?id_2, p_{12}, ?val_{12})\ \cdots\ (?id_2, p_{n2}, ?val_{n2})\ [FILTER()]\\
&\ldots\\
&(?id_n, type, C_n)\ (?id_n, p_{1n}, ?val_{1n})\ \cdots\ (?id_n, p_{nn}, ?val_{nn})\ [FILTER()]
\end{aligned}
$$

Paramètres	Signification
|R|	nombre de pages nécessaires pour stocker la table R
||R||	nombre de tuples 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'un index ou d'un triplet
TSR	taille d'un tuple 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 de l'index I en E/S

Tableau 2. *Les paramètres et notations du modèle de coût*

5.3. *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 : $(?id, type, C)(?id, p_1, ?val_1) \cdots (?id, p_n, ?val_n)[FILTER()]$. 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. Pour la sélection *mono-triplet*, notre fonction est égale au nombre de pages de la table impliquée dans la requête :

– dans l'approche verticale, $cout(q, bds) = |T|$ où *T* est la table de triplets. Si un index est défini sur la table de triplets, $cout(q, bds) = P(index) +$ *sel(t)* $* |T|$, où *P(index)* est le coût de parcours d'index et *sel(t)* est la sélectivité du triplet *t* comme définie dans (Stocker *et al.*, 2008).

– dans l'approche binaire, la sélection porte sur la table correspondant au prédicat du triplet, notée *Tp* : $cout(q, bds) = |Tp|$. Avec un index défini sur le prédicat de sélection, $cout(q, bds) = P(index) +$ *sel* $* |Tp|$ où *sel* est la sélectivité de l'index.

– dans l'approche horizontale, la sélection porte sur la ou les tables relatives aux classes qui forment le domaine de la propriété du triplet de la requête, notées *Tcp* : $Cout(q, bds) = \sum_{Tcp \in domaine(p)}(|Tcp|)$. Si un index est défini sur le prédicat de sélection, $cout(q, bds) = \sum_{Tcp \in domaine(p)}(P(index) +$ *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*. 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.

5.4. *Coût de la jointure*

Les jointures sont des requêtes constituées d'au moins deux motifs de triplets. Dans l'approche horizontale, ces triplets doivent porter sur au moins deux classes de différentes hiérarchies. Pour l'approche verticale, nous notons que l'auto-jointure de la table des triplets peut se faire de deux manières : (1) comme une jointure naïve de deux tables relationnelles classiques, c'est à dire par un produit cartésien suivi d'une sélection. Nous appelons *jointure à la BDR* cette approche et (2) en sens inverse de la *jointure à la BDR*, c'est-à-dire des sélections suivies d'une jointure classique. Nous appelons *jointure retardée* cette approche. Nous considérons successivement ces deux approches de l'auto-jointure de la table de triplets. Pour les autres approches, notre fonction de coût est la même que celle utilisée dans les bases de données relationnelles et dépend de l'algorithme de jointure utilisé. Les coûts de jointure par boucle imbriquée (*NL*) et par hachage (*Hash*) sont présentés dans le tableau 3 (les notations ont été présentées précédemment).

NL sans index	*jointure retardée*	*jointure à la BDR*
V : join(T,T)	$2*\|T\|+2\|t1\|+\|t2\|*(1+\|t1\|/M)$	$\|T\|+\|T\|*\|T\|/M$
B : join(Tp1,Tp2)	Non Applicable	$\|Tp1\|+\|Tp1\|/M*\|Tp2\|$
H : join(Tcp1,Tcp2)	Non Applicable	$\sum_{Tcp\in dom(p)}(\|Tcp1\|+\|Tcp1\|/M*\|Tcp2\|)$
NL avec index	*jointure retardée*	*jointure à la BDR*
V : join(T,T)	$\|T\|(sel(t1)+sel(t2))+P(I1)+P(I2)+\|t1\|+\|t1\|*\|t2\|/M$	$\|T\|+\|T\|*\|T\|/M$
B : join(Tp1,Tp2)	Non Applicable	$\|Tp1\|+\|\|Tp1\|\|*(P(I)+\|\|Tp2\|\|*Sel)$
H : join(Tcp1,Tcp2)	Non Applicable	$\sum_{Tcp\in dom(p)}(\|\|Tcp1\|+\|\|Tcp1\|\|*(P(I)+\|\|Tcp2\|\|*Sel))$
Hash avec index	*jointure retardée*	*jointure à la BDR*
V : join(T,T)	$2*\|T\|+4*(\|t1\|+\|t2\|)$	$6*\|T\|$
B : join(Tp1,Tp2)	Non Applicable	$3*(\|Tp1\|+\|Tp2\|)$
H : join(Tcp1,Tcp2)	Non Applicable	$\sum_{Tcp\in dom(p)}3(\|Tcp1\|+\|Tcp2\|)$

Tableau 3. *Coût de jointures (dom(p) : domaine de p).*

5.5. *Application du modèle de coût*

Pour mettre en oeuvre notre fonction de coût, nous avons considéré les requêtes 2, 4 et 6 du benchmark LUBM (Y. Guo *et al.*, 2005). Ces requêtes sont traduites en SQL

suivant les différents modèles de stockage et une évaluation de leur coût est faite en utilisant les statistiques du jeu de données LUBM01 (données dans la section 6). Les résultats obtenus sont consignés dans le tableau 4 (V1 représente l'approche verticale avec utilisation de la jointure retardée tandis que V2 correspond à l'utilisation de la jointure à la BDR) et illustrés sur la figure 4. Pour les calculs, nous avons fait les hypothèses suivantes : le taux de remplissage des tables est de 80%, la taille des pages est de 8Ko, la taille de champs (sujet, propriété et objet) est de 200 octets chacune et la mémoire centrale dispose de 50.000 pages.

Requête	**Type**	**V1**	**V2**	**B**	**H**
Q6	SMT	9168	9168	1138	10289
Q4	SPT	21308	137520	60864	121
Q2	Hash	23188 1	65024	195246	18813

Tableau 4. *Coût en terme d'E/S selon le modèle de coût (SMT/SPT : sélection mono/pluri-triplet(s))*

Ces 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'E/S que son exécution effectuant d'abord les sélections suivies de jointures. Nous observons également que pour une sélection *mono-triplet*, l'approche binaire réalise moins d'E/S, donc est susceptible d'être plus efficace que les autres approches. Dans les autres types de requêtes (jointure et sélections *pluri-triplets*), l'approche horizontale se montre meilleure suivie de l'approche verticale. Comme nous le verrons ultérieurement, ces résultats sont partiellement confirmés par nos résultats expérimentaux (section 6). En effet, nous verrons que OntoDB (approche horizontale) donne de bons temps de réponse, suivi de Oracle (approche verticale) et SOR (approche binaire). Par contre, pour certains systèmes expérimentés comme Jena, Sesame et DB2RDF, les résultats des évaluations ne confirment pas très clairement les résultats théoriques. Nous croyons que cela est dû aux techniques d'optimisation spécifiques qui peuvent être mises en place dans ces systèmes et qui ne sont pas prises en compte par notre modèle de coût.

6. Comparaison des performances des $\mathcal{BDS}$: expérimentations

Dans cette section, nous menons une batterie d'expérimentations pour évaluer les six $\mathcal{BDS}$ étudiées selon deux critères : le temps de chargement des instances ontologiques et le temps d'exécution des requêtes. Dans les sections suivantes, nous décrivons les ensembles de données sur lesquels les expérimentations sont effectuées, présentons les évaluations et les interprétations des résultats.

6.1. *Ensemble de données pour les tests*

Pour disposer des données ontologiques nécessaires pour nos tests de performance, nous avons utilisé le benchmark LUBM qui se base sur le formalisme OWL. Le géné-

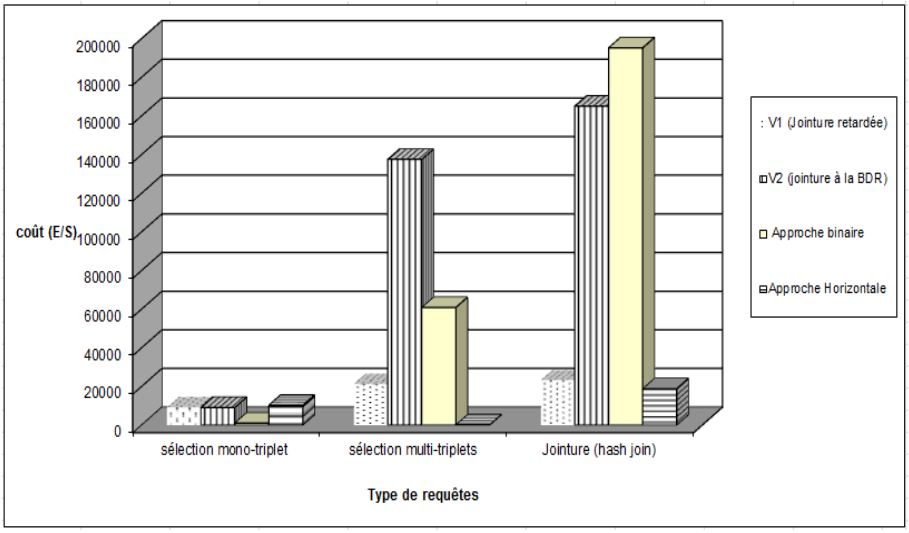


Figure 4. *Coûts de requêtes évalués théoriquement par notre modèle de coût*

rateur associé à ce benchmark crée une ontologie portant sur le domaine universitaire. Chaque université est constituée de 15 à 20 départements. Chaque département est constitués d'enseignants de différentes catégories, d'étudiants, de cours, etc. Nous avons généré trois ensembles de données ayant respectivement 1, 5 et 10 universités (Lubm01, Lubm05 et Lubm10). Le tableau 5 indique les statistiques de ces jeux de données en nombre d'instances et triplets. 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 benchmark, exprimées en OWL, vers le format N-Triple est nécessaire. Pour ce faire, nous avons utilisé l'API Jena appelée *rdfcat*.

Nos expérimentations 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 SGBD utilisés sont : Oracle 11g pour la base de données sémantiques Oracle, IBM DB2 9.6 pour SOR et DB2RDF et PosgreSQL 8.2 pour OntoDB, Jena et Sesame.

Jeux de données	Nombre d'instances	Nombre de triplets
Lubm01	82.415	100.851
Lubm05	516.116	625.103
Lubm10	1.052.895	1.273.108

Tableau 5. *Statistiques des jeux de données générés*

6.2. *Performances en termes de chargement des données (ontologie et instances)*

Cette partie concerne les chargements de l'ontologie et de ses instances dans nos $\mathcal{BDS}$. 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 $\mathcal{BDS}$; et (4) l'interprétation des résultats. Les résultats sont consignés dans le tableau 6. Pour avoir des résultats homogènes, nous avons effectué quatre fois le chargement de chaque ensemble de données et nous avons retenu la moyenne obtenue.

6.2.1. *Le chargement de données pour Oracle*

Pour Oracle, 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 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 mis 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.

6.2.2. *Le chargement de données pour IBM SOR*

Pour faire une évaluation de performance de SOR, nous avons utilisé le framework IODT (`http://www.alphaworks.ibm.com`) et IBM DB2 Express 9. A la différence de chargement sous Oracle, 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.

6.2.3. *Le chargement de données pour DB2RDF*

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

6.2.4. *Le chargement de données pour 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'ontologies PLIB. Vu que l'ontologie que nous souhaitons utiliser pour nos tests 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 pour pouvoir la charger dans OntoDB. Mais, vu la

différence et la diversité de constructeurs et des opérateurs des deux formalismes, l'ontologie obtenue de cette manière ne serait pas équivalente à l'ontologie de départ. Aussi, nous avons préféré développer un module de chargement des données OWL dans OntoDB sans passer par le formalisme PLIB. Ainsi, nous avons mis en place un module d'importation des ontologies OWL et leurs instances dans OntoDB. Ce module prend 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 base de données 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, $T :< O_{OWL}, E_{owl} >\rightarrow< O_{OntoDB}, E_{OntoDB} >$ où O_{OWL} et E_{OWL} sont respectivement une ontologie et un concept ou instance de cette ontologie dans le formalisme OWL, O_{OntoDB} et E_{OntoDB} sont respectivement une ontologie et un objet (table, attribut ou valeur) dans OntoDB (dans le formalisme d'ontologie PLIB). Soit C et P l'ensemble des 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 PLIB et est définie dans OntoDB (une table correspondante est créée pour ses instances).
- $\forall p \in P, T(O_{OWL}, p)$ est une propriété PLIB et est 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 est irréversible du fait que les concepts OWL non canoniques ne sont pas pris en compte et qu'ainsi, il n'est pas possible de retrouver l'ontologie initiale par une application inverse.

Notons également que cet utilitaire ne travaille que sur des ontologies *acycliques*. Cela est une exigence de modèle d'ontologie PLIB qui est au coeur du système OntoDB. Or nos ontologies LUBM renferment des cycles. Nous avons rompu les cycles en transformant les propriétés de type *objectproperty* à l'origine des cycles en *datatypeproperty*.

6.2.5. *Le Chargement de données pour Jena et Sesame*

Pour les chargements de données dans les $\mathcal{BDS}$ Jena et Sésame, nous avons utilisé un environnement Eclipse identique à celui utilisé pour SOR et DB2RDF. Pour chaque $\mathcal{BDS}$, nous avons mis en oeuvre les modèles et fonctions nécessaires pour le chargement. Étant donné que Sesame offre la possibilité d'avoir un $\mathcal{BDS}$ avec une approche verticale et $\mathcal{BDS}$ avec une approche binaire, nous avons expérimenté ces deux approches. Les temps de chargement moyens sont consignés dans le tableau 6. Dans ce qui suit, nous appelons SesameBdsI, le système Sesame utilisant une approche verticale et SesameBds2, le système Sesame utilisé avec une approche binaire.

La figure 5 présente une histogramme de ces résultats. Les temps de chargement du système OntoDB n'y est pas représenté du fait de la différence d'échelle par rapport aux autres $\mathcal{BDS}$.

Dataset	Lumb01	LumB05	LumB10
Oracle Batch	42	222	302
Oracle Bulk	12	55	116
IBM SOR	90	590	1147
DB2RDF	11	53	109
Jena	25	188	558
SesameBdsI	16	158	391
SesameBdsII	27	231	522
OntoDB	15975	72699	146023

Tableau 6. *Récapitulatif des temps de chargement (sec).*

En terme de chargement, comme on le constate sur le tableau 6 et la figure 5, DB2RDF et Oracle (Bulk) fournissent de très bons résultats, suivis dans l'ordre de SesameBdsI, Jena, SesameBdsII, Oracle Bulk et SOR. OntoDB se montre plus lente que les autres $\mathcal{BDS}$ étudiées. Pour DB2RDF et Oracle (Bulk), cette performance peut s'expliquer par le fait que peu d'opérations sont effectuées sur des données lors de l'insertion de données (pas de sous-requêtes afférentes) mais aussi par la puissance des SGBD utilisés (oracle et DB2) qui ont des outils de chargement spécialisés (par exemple, SQLLoader). Sesame utilisant la table de triplets se relève plus performant que Jena, mais lorsque celui-ci utilise une représentation binaire, les temps de chargement sont plus élevés qu'avec Jena. La $\mathcal{BDS}$ SOR prend assez de temps pour les transpositions du schéma d'ontologies (T-Box) et des instances ontologiques (A-Box) en des tuples pour les différentes tables. Elle met aussi un temps important pour faire les inférences des données aux niveaux ontologie et instance. Cela justifie le fait que les temps de chargement sont plus importants que ceux des la plupart des autres $\mathcal{BDS}$.

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 référencer les objets. Les instances référençant une autre instance doivent donc aller récupérer les identifiants créés en utilisant des sous-requêtes. 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 l'étudiant était diplômé (*undergraduateDegreeFrom*). Pour éviter des erreurs lors d'insertion, OntoDB maintient un ordre qui impose la création de superclasses d'une classe avant celle de la classe et de même pour l'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.

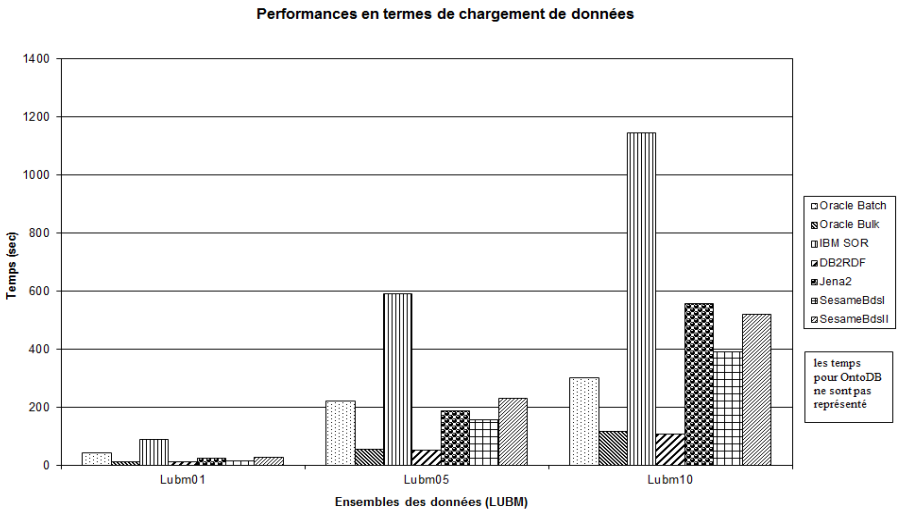
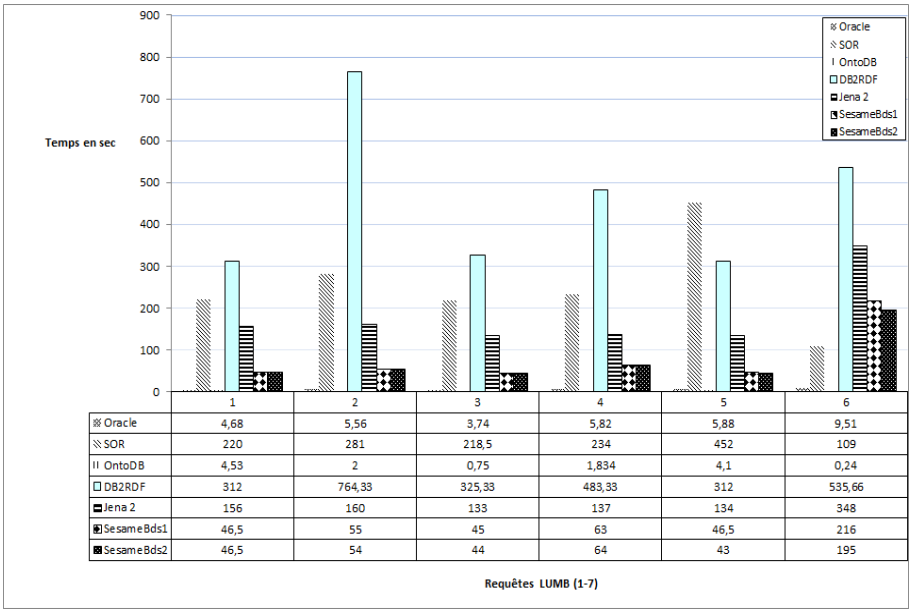


Figure 5. *Temps de chargement des données.*

6.3. *Performances en termes de temps de réponse des requêtes*

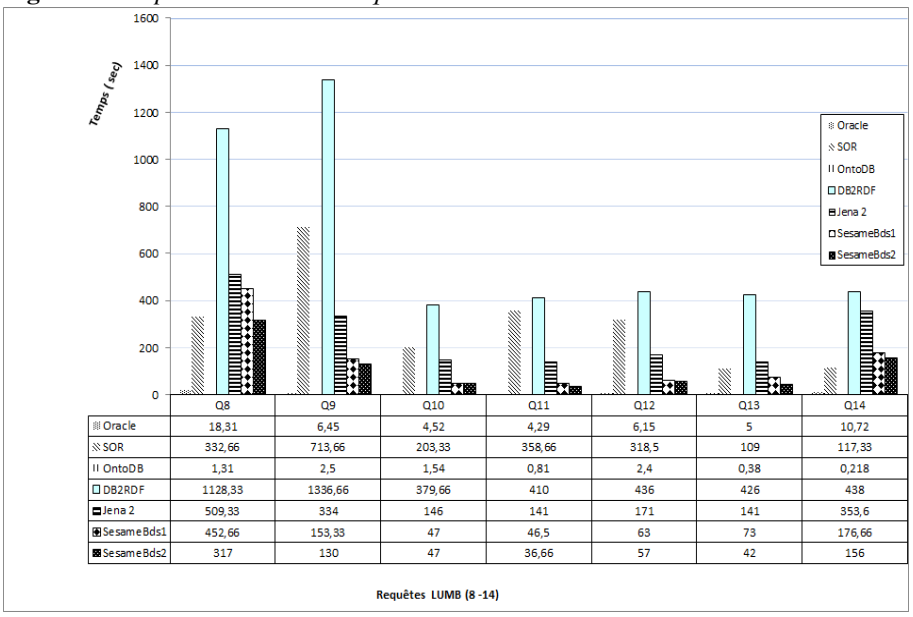
Nous avons mesuré les temps d'exécution des requêtes (en sec.) sur les six $\mathcal{BDS}$. Comme pour la mesure des temps de chargement, pour chaque requête, nous avons effectué quatre fois les mesures et le temps moyen est retenu. Les requêtes utilisées sont celles proposées dans le benchmark LUBM. Nous les avons traduites pour qu'elles soient acceptées par nos $\mathcal{BDS}$. Le jeu de données utilisé est Lubm01. Les résultats ont donné lieu aux histogrammes des figures 6 et 7. Nous avons utilisé la base de règles *owlprime* pour Oracle.

Nous observons tout d'abord que OntoDB obtient les meilleurs résultats. L'utilisation de la représentation horizontale est une des raisons de ces résultats. Dans les requêtes exécutées ce modèle de stockage permet de réaliser un nombre moindre d'opérations de jointure. En effet, toutes les requêtes portant tous 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 $\mathcal{BDS}$. La requête 4 de LUBM en est une bonne illustration. Cette requête est composée de cinq triplets portant tous le 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 qu'OntoDB parcourt moins de données que les autres système. 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 (Oracle, SesameDbsI et DB2RDF) alors que dans OntoDB, seules les tables de l'attribut en question (*UndergraduateStudent* et *GraduateStudent* pour la requête 6) sont parcourues. Oracle vient en second lieu, suivi de Sesame. Nous remarquons que la version de



	1	2	3	4	5	6
Oracle	4,68	5,56	3,74	5,82	5,88	9,51
SOR	220	281	218,5	234	452	109
OntoDB	4,53	2	0,75	1,834	4,1	0,24
DB2RDF	312	764,33	325,33	483,33	312	535,66
Jena 2	156	160	133	137	134	348
SesameBds1	46,5	55	45	63	46,5	216
SesameBds2	46,5	54	44	64	43	195

Figure 6. *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 7. *Temps d'exécution de requêtes LUBM de 8-14.*

Sesame qui utilise la représentation binaire (sesameBdsII) donne de meilleurs résultats que la version qui utilise la représentation verticale (sesameBdsI). Cette différence se remarque à la requête 14 où SesameBds1 parcourt toute la table de triplets alors que SesameBds2 ne parcourt que la table *typeof*. En ce qui concerne les systèmes d'IBM, SOR donne de meilleurs résultats que DB2RDF. Pourtant, les deux utilisent le même SGBD (DB2 9.6). Les faibles performances de DB2RDF peuvent s'expliquer par la multitude de tables qui accompagnent la table de triplets et qui impliquent la réalisation de plusieurs jointures pour répondre aux requêtes. Nous notons également, que SOR est la $\mathcal{BDS}$ la plus complète au niveau des déductions. Par exemple, pour la requête 12, les autres $\mathcal{BDS}$ ne retournent pas de résultat car il y a pas de déclaration explicite d'instance de *Chair*. A l'inverse SOR renvoie des résultats en se basant sur la propriété *headOf* puisqu'un individu qui est une instance de *Professor* est une instance de *Chair* s'il est à la tête d'un *Département*. Si on ajoute une règle traduisant cette assertion à une $\mathcal{BDS}$ qui prend en compte une base de règles utilisateur (par exemple Oracle), elle fournira également des réponses. Mais ce n'est pas une règle prédéfinie dans ce système. Nous notons que Jena et Sesame utilisent des moteurs d'inférence pour la déduction de données en mémoire centrale ou en fichier standard mais pas sur des données stockées en bases de données. Il est également possible de faire appel à un moteur d'inférence lors du chargement des 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 et du passage à l'échelle. Il n'est pas exclu d'envisager l'utilisation d'un moteur d'inférence lors de l'exécution des requêtes mais cela allongera alors considérablement le temps de traitement des requêtes. Pour OntoDB, cette $\mathcal{BDS}$ ne dispose pas d'un moteur d'inférence mais réalise la saturation au chargement des données. Cela peut aussi expliquer les longs temps de chargement.

7. Conclusion

Durant ces dernières années les ontologies ont été de plus en plus utilisées dans une variété de domaine. En conséquence un fort besoin de gestion de ces ontologies en bases de données s'est fait ressentir. Aussi, des universitaires et des industriels ont proposé des solutions de persistance s'appuyant sur des SGBD existants pour permettre l'interrogation efficace de ces données tout en supportant une volumétrie importante. Contrairement aux SGBD traditionnels qui sont similaires en termes d'architecture et de modèle de stockage, les travaux menés ont conduit à la définition de base de données dites sémantiques qui présentent une grande diversité en termes de formalismes d'ontologies supportés, de modèles de stockage et d'architectures utilisés.

Pour faciliter la compréhension de cette diversité, nous avons présenté un état de l'art sur les ontologies, leurs formalismes, les modèles de stockage et les architectures utilisées par les bases de données sémantiques. Pour compléter cette étude nous avons présenté un modèle qui capture cette diversité ainsi qu'une étude plus détaillée de six bases de données sémantiques dont trois provenant du milieu industriel (Oracle, IBM SOR et DB2RDF) et trois du milieu académique (OntoDB, Sesame et Jena). Pour aller

plus loin dans cette étude, nous nous sommes intéressés aux performances des bases de données sémantiques qui est un critère important qui a motivé leur définition. Nous avons mené cette étude à la fois de manière théorique, par la définition d'un modèle de coût et de manière empirique par la mesure du temps de chargement et du temps de réponses de requêtes du benchmark LUBM. Les résultats obtenus montrent que notre modèle de coût permet de prédire assez fidèlement les performances obtenues. Nous avons cependant identifié la nécessité de l'affiner pour les bases de données sémantiques qui utilisent des optimisations particulières. Quand aux performances elles-mêmes, nous notons tout d'abord l'efficacité des $\mathcal{BDS}$ industrielles en termes de temps de chargement des données ontologiques. Elles s'appuient en effet sur des outils de chargement dédiés qui avaient déjà été optimisés pour les bases de données relationnelles (par exemple, SQLLoader d'Oracle). Pour les temps de réponse des requêtes, les résultats sont très divers. Certaines bases de données sémantiques comme SOR se focalisent sur le raisonnement, ce qui se traduit en une dégradation de performance même pour les requêtes n'en nécessitant pas. Pour les autres bases de données sémantiques, la représentation horizontale, utilisée en particulier par OntoDB, donne de très bons résultats pour la majorité des requêtes.

Parmi les perspectives de ce travail, nous prévoyons d'abord de réaliser d'autres expérimentations sur des volumétries plus importantes et sur différents benchmarks afin de valider notre modèle de coût. Une amélioration de la fonction de coût pour prendre en compte les particularités des $\mathcal{BDS}$ est aussi envisageable. Ensuite, nous prévoyons d'utiliser ce modèle de coût pour étudier le problème de sélection des vues matérialisées dans le contexte des $\mathcal{BDS}$.

8. Bibliographie

Bechhofer S., Harmelen F. V., Hendler J., Horrocks I., McGuinness D., Patel-Schneider P., Stein L., « OWL Web Ontology Language Reference », 2004.

Beckett D., *Scalability and Storage : Survey of Free Software / Open Source RDF storage systems*, Semantic Web Advanced Development for Europe (SWAD-Europe) for EU IST-2001-34732, 2002.

B.McBride, « Jena : Implementing the RDF model and syntax specification », *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb'01)*, 2001.

Brickley D., Guha R., « RDF Vocabulary Description Language 1.0 : RDF Schema », *W3C, http ://www.w3.org/TR/rdf-schema/*, 2002.

Broekstra J., Kampman A., van Harmelen F., « Sesame : A Generic Architecture for Storing and Querying RDF and RDF Schema », *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, p. 54-68, 2002.

Carroll J. J., Dickinson I., Dollin C., Reynolds D., Seaborne A., Wilkinson K., « Jena : implementing the semantic web recommendations », *Proceedings of the 13th international World Wide Web conference (WWW'04)*, p. 74-83, 2004.

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

Faye D., Curé O., Blin G., « A survey of RDF storage approaches », *ARIMA,*p. 11-35, 2012.

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

Gruser J. R., Modèle de coût pour l'optimisation de requête objet, Thèse de doctorat, Université de Paris VI, Décembre, 1996.

Hertel A., Broekstra J., Stuckenschmidt H., « RDF storage and retrieval systems », *Handbook on Ontologies*p. 489-508, 2009.

IBM, « RDF application development for IBM data servers », 2012.

Jean S., Ait-Ameur Y., Pierra G., « Querying Ontology Based Database Using OntoQL (an Ontology Query Language) », *Proceedings of the OTM Confederated International Conferences (ODBASE'06)*, p. 704-721, 2006.

Jean S., Pierra G., Ameur Y. A., « Domain Ontologies : A Database-Oriented Analysis », *Chapter 19 of Web Information Systems and Technologies, International Conferences, WEBIST 2005 and WEBIST 2006. Revised Selected Papers*, p. 238-254, 2007.

Lee R., Scalability report on triple store applications, Technical report, MIT, 2004.

Liu B., Hu B., « An Evaluation of RDF Storage Systems for Large Data Applications », *International Conference on Semantics, Knowledge and Grid (SKG'05)*, p. 59, 2005.

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

Magkanaraki A., Karvounarakis G., Anh T. T., Christophides V., Plexousakis D., « Ontology storage and querying », *ICS-FORTH Technical Report*, 2002.

Mbaiossoum B., Khouri S., Bellatreche L., Jean S., Baron M., « Etude comparative des systèmes de bases de données à base ontologiques », *30ème congrès Inforsid*, p. 379-394, 2012.

Pierra G., « Context Representation in Domain Ontologies and its Use for Semantic Integration of Data », *Journal Of Data Semantics (JoDS)*, vol. 10, p. 174-211, 2008.

Sakr S., Al-Naymat G., « Relational processing of RDF queries : a survey », *SIGMOD Record*, vol. 38, n° 4, p. 23-28, 2009.

Stegmaier F., Gröbner U., Döller M., Kosch H., Baese G., « Evaluation of current RDF database solutions », *Proceedings of the 10th International Workshop on Semantic Multimedia Database Technologies (SeMuDaTe'09))*, p. 39-55, 2009.

Stocker M., Seaborne A., Bernstein A., Kiefer C., Reynolds D., « SPARQL basic graph pattern optimization using selectivity estimation », *Proceedings of the 17th international conference on World Wide Web (WWW'08)*, p. 595-604, 2008.

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

Wu Z., Eadon G., Das S., Chong E. I., Kolovski V., Annamalai M., Srinivasan J., « Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle », *ICDE'2008*, p. 1239-1248, April, 2008.

Y. Guo Z. P., Heflin J., « LUBM : A Benchmark for OWL Knowledge Base Systems », *Journal of Web Semantics*, vol. 3, p. 158-182, 2005.