Données de tests non fonctionnels de l'ombre à la lumière : une approche multidimensionnelle pour déployer une base de données

Lahcene Brahimi

▶ **To cite this version:**

Lahcene Brahimi. Données de tests non fonctionnels de l'ombre à la lumière : une approche multidimensionnelle pour déployer une base de données. Ordinateur et société [cs.CY]. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique - Poitiers, 2017. Français. NNT : 2017ESMA0009 . tel-01585828

HAL Id: tel-01585828

https://theses.hal.science/tel-01585828v1

Submitted on 12 Sep 2017

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

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

Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

THESE

pour l'obtention du Grade de

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

(Diplôme National — Arrêté du 25 mai 2016)

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

Présentée par :

Lahcène BRAHIMI

****

Données de tests non fonctionnels de l'ombre à la lumière : une approche multidimensionnelle pour déployer une base de données

****

Directeur de Thèse : **Ladjel BELLATRECHE**
Co-encadrant : **Yassine OUHAMMOU**

Soutenue le 03 juillet 2017
devant la Commission d'Examen

JURY

Rapporteurs :	**Djamel BENSLIMANE**	**Professeur, Université Lyon I, France**
	Yudith CARDINALE	**Professeur, Universidad Simòn Bolívar, Venezuela**
Membres du jury :	**Abdelkader HAMEURLAIN**	**Professeur, Université Toulouse III, France**
	Khalil DRIRA	**Directeur de Recherche CNRS , France**
	Ladjel BELLATRECHE	**Professeur, ISAE-ENSMA, Poitiers, France**
	Yassine OUHAMMOU	**Maître de conférences, ISAE-ENSMA, Poitiers, France**

Remerciements

Je tiens à adresser mes plus chaleureux remerciements à :

- *Mon directeur de thèse Ladjel BELLATRECHE, professeur des universités à l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique ENSMA de Poitiers et chef d'équipe d'Ingéniérie des Données et des Modèles, pour m'avoir accueilli au sein du Laboratoire d'Informatique et d'Automatique pour les Systèmes LIAS et pour m'avoir apporté ses précieux conseils personnels et professionnels tout au long de son encadrement de cette thèse.*
- *Mon co-encadrant Yassine OUHAMMOU, Maître de Conférences à l'ENSMA de Poitiers, pour toute sa coopération et toutes nos discussions très enrichissantes.*
- *Monsieur Abdelkader HAMEURLAIN d'avoir accepté de présider mon jury de thèse.*
- *Monsieur Djamel BENSLIMANE et Madame Yudith CARDINALE, pour m'avoir fait l'honneur d'être rapporteurs de cette thèse et d'avoir accepté cette lourde tâche.*
- *Monsieur Khalil DRIRA d'avoir accepté d'examiner mon travail et de participer à mon jury de thèse.*
- *Le directeur du laboratoire Emmanuel GROLLEAU pour sa gentillesse et son respect.*
- *Brice CHARDIN et mickael BARON d'avoir accepté de travailler avec moi pendant une période de cette thèse.*
- *Tout le personnel du laboratoire, permanents et doctorant(e)s et à mes ami(e)s et collègues sans exception, pour leur présence et leur respect.*
- *Mes chers frères et sœurs qui m'ont encouragé pendant ces trois années.*

Je voudrais conclure mes remerciements en exprimant ma plus profonde gratitude à mes chers parents, qui m'ont toujours soutenu dans les moments difficiles et m'ont poussé vers le mieux et à ma chère femme pour m'avoir soutenu tout au long de cette thèse et accompagné depuis notre mariage.
Enfin, Je suis super fier de mes quatre trésors (CH_4) Chahd, Chahine, Chihab et Chadha, mes sources de motivation qui ont vécu avec moi tous les bons et mauvais moments de cette thèse.

A Mes parents
Ma petite famille
Ma femme et mes CH_4 :
Chahd, Chahine, Chihab et Chadha

Table des matières

Chapitre 1 Introduction générale **1**

1 Contexte 3

1.1 Discussion et Problématique 8

1.2 Exploitation des données de tests localisées dans les articles scientifiques 14

2 Objectifs et contributions 16

3 Organisation de la thèse 18

4 Publications 20

Partie I État de l'art

Chapitre 2 Évolution du cycle de conception de bases de données **25**

1 Introduction 27

2 Les Causes de problème de sélection de plateforme : Évolution de cycle de vie 28

2.1 Évolution verticale du cycle de conception 29
2.1.1 Edgar Codd et le cycle de vie 30
2.1.2 Peter Chen et le cycle de vie 31
2.1.3 Omniprésence des besoins fonctionnels et non fonctionnels 33
2.1.4 Nouvelles applications et le cycle de vie 34
2.1.5 Émergence des SGBD et des plateformes 35
2.2 L'évolution interne du cycle de conception 36
2.2.1 Définition des besoins 37
2.2.2 La modélisation conceptuelle 38
2.2.3 La modélisation logique 39
2.2.4 La phase de déploiement 40
2.2.5 La modélisation physique 45
2.3 Évolution horizontale du cycle de conception 47
2.3.1 Les SGBD orientées objet 48
2.3.2 Diversification des modèles conceptuels 49
2.3.3 Évolution des besoins non fonctionnels 51
2.3.4 Apparition des SGBD Nosql et NewSQL 52
3 Bilan 53
4 Conclusion 54

Chapitre 3 Etudes des Approches de Selection des SGBD et Plateformes 55

1 Introduction 57
2 Test de bases de données 59
3 Le rôle des bancs d'essai dans notre démarche 63
3.1 YCSB - Yahoo Cloud Serving Benchmark 63
3.2 Banc d'essai du Transaction Processing Performance Council (TPC) 66
3.2.1 TPC-C 66
3.2.2 TPC-H 66

3.3 Star Schema Benchmark (SSB) 67
3.4 Bancs d'essai pour les bases de données objets 68
3.5 Banc d'essai pour des données spécifiques 70
3.5.1 Données génétiques et GenBase 70
3.5.2 BerlinMod 70
3.5.3 Linear Road Benchmark (LRB) 70
4 Les approches naïves de résolution de notre problème 71
4.1 Test matériel de toutes les plateformes 71
4.2 Des approches basées sur la simulation 71
4.2.1 Modèles de coût mathématique et simulation 72
4.3 Solutions basées sur des retours d'expérience 73
5 Vers la réduction de l'espace de recherche de notre problème 76
5.1 Techniques basées sur la classification des SGBD 76
5.2 Classification basée sur le théorème de CAP et les deux propriétés ACID et BASE 77
5.3 Classification basée sur le nombre d'utilisateurs et le volume de données 78
5.4 Solutions basées sur la décomposition des SGBD 79
6 Bilan et discussion 80
7 Conclusion 80

Partie II Contributions

Chapitre 4 Vers une explicitation des composantes de l'environnement de tests 85
1 Introduction 87

2 Description de l'environnement de tests 87
2.1 Environnement de test 88
2.2 Formalisation et structuration 89
3 Composantes de l'environnement de test 90
3.1 Métriques de performance 92
3.2 Modèle de coût et algorithme 92
3.3 Jeux de données et requêtes 93
3.4 SGBD et Plateformes 96
4 Ontologie pour les des systèmes de base de données (SBD) 98
4.1 Définition d'une ontologie 98
4.2 Différents types d'ontologies 100
4.3 Langages de représentation des ontologies 100
4.3.1 Formalisme RDFS 100
4.3.2 Formalisme DAML+OIL 101
4.3.3 Formalisme OWL 101
4.3.4 Formalisme PLIB 101
4.4 Étapes de conception des ontologies 101
4.5 SPARQL : Langage d'interrogation 102
4.6 Présentation de l'ontologie des SBD 103
4.6.1 Composantes du modèle des SBD 104
4.6.2 Classe des SGBD (DBMS) 105
4.6.3 Classe du matériel (Hardware) 108
4.6.4 Classe des systèmes d'exploitation (Operating_System) . 109
4.6.5 Exemple d'usage de SPARQL 110
5 Conclusion 111

Chapitre 5 Approche multidimensionnelle des données des environnements de tests 113

1 Introduction 115
2 Entrepôt de données 115
2.1 Définition d'un entrepôt de données 116
2.2 Magasin de données (datamart) 116
2.3 Architecture des entrepôts de données 117
2.4 Modèle de données 118
2.4.1 Schéma en étoile 118
2.4.2 Schéma en flocon 118
2.5 Types d'entrepôts de données 120
2.6 Traitement analytique en ligne 120
2.6.1 Cube de données 121
2.6.2 Opérations OLAP pour l'interrogation de l'entrepôt de données 122
3 Notre entrepôt de données de tests non fonctionnels 124
3.1 Développement de l'entrepôt de données 124
3.2 Interface d'insertion et de sélection de tests 127
3.2.1 Manifeste un moyen pour définir les exigences du déployeur 128
3.2.2 Interaction entre le manifeste et l'entrepôt 128
3.2.3 Insertion des tests/instances du dépôt 129
3.2.4 Recherche des tests 130
3.3 Exemples d'usage de notre Entrepôt 131
4 Conclusion 132

Chapitre 6 Système de recommandation pour le déploiement des bases de données **135**

1 Introduction 137
2 Définitions et concepts 137

2.1 Aide à la décision multicritère 137
2.2 Modèle de la somme pondérée 138
2.3 Apprentissage automatique et similarité 138
2.3.1 Similarité et distance 139
2.3.2 Similarité de requêtes 139
2.4 Système de recommandation 140
2.4.1 Filtrage collaboratif 141
2.4.2 Systèmes de recommandation basés sur le contenu 143
2.4.3 Filtrage basé sur la connaissance 143
3 Composition de notre système de recommandation 144
3.1 Phase 1 : Approche basée sur la décision multicritère 145
3.2 Phase 2 : Approche basée sur l'apprentissage automatique 146
4 Implémentation et expérimentation 146
4.1 Outils et environnement de développement 148
4.2 Évaluation et analyse des résultats de la première phase 148
4.3 Évaluation et analyse des résultats de la deuxième phase 151
4.3.1 Cas d'étude 1 : requêtes des bancs d'essai 151
4.3.2 Cas d'étude 2 : Requêtes des déployeurs 155
5 Conclusion 158

Chapitre 7 Conclusion générale et Perspectives **161**

1 Conclusion 163
1.1 Analyse des solutions existantes pour le problème de sélection des SGBD 163
1.2 Conception d'une ontologie des systèmes de base de données . . . 164
1.3 Développement d'un dépôt de données de tests 164
1.4 Constitution d'un système de recommandation 164
2 Perspectives 165

2.1 Validation de l'ontologie 165
2.2 Diversification de l'étude expérimentale 165
2.3 Intégration de la similarité catégorielle 166
2.4 Approfondissement des techniques de la décision multicritère . . . 166
2.5 Automatisation de la lecture des données de tests 166

Bibliographie **167**

Table des figures **185**

Liste des tableaux **189**

Glossaire **191**

Chapitre 1

Introduction générale

Sommaire

1	**Contexte**	**3**
2	**Objectifs et contributions**	**16**
3	**Organisation de la thèse**	**18**
4	**Publications**	**20**

1 Contexte

La technologie de bases de données ($\mathcal{BD}$) a connu quatre ères successives : **(i)** l'ère navigationnelle, **(ii)** l'ère relationnelle, **(iii)** l'ère multidimensionnelle, et **(iv)** l'ère non relationnelle. Dans l'ère navigationnelle (1965-1970), nous avons vu la naissance de deux modèles de données : le modèle *hiérarchique* et le modèle *réseau*. Le modèle *hiérarchique* consiste à stocker les données dans des enregistrements comportant un ensemble de champs ayant chacun un type de données. Les enregistrements sont organisés sous forme d'un arbre tel que chaque nœud (enregistrement) possède un seul parent. Chaque arbre comprend une racine et des branches qui permettent l'accès aux différents niveaux de données. Le niveau d'une donnée mesure sa distance à la racine. La racine est située au niveau supérieur. Les autres données se situent au niveau des nœuds de l'arbre (on parle de parents et d'enfants). Les relations entre les enregistrements d'un niveau et du niveau immédiatement inférieur sont de type *un à plusieurs* $(1, n)$. Les Systèmes de Gestion de Bases de Données (SGBD) hiérarchiques les plus utilisés sur le marché étaient ceux des systèmes IMS (Information Management System) et SYSTEM-2000.

Ce modèle souffrait de limites majeures [103] : (1) la redondance de l'information pouvant engendrer des inconsistances entre les données et (2) l'existence des données d'un enregistrement repose sur celle de son enregistrement parent. Le modèle de données *réseau* (ou Codasyl) a été proposé en 1969, suivi d'une autre version en 1973 complétant le modèle par une spécification d'un langage de manipulation des données. Il a servi de base pour la plupart des systèmes commerciaux au cours des années 1970 [90]. Il stocke les données dans une collection d'enregistrements organisés, non pas dans un arbre, mais dans un réseau (ou graphe) dont les nœuds sont les enregistrements. Ce modèle permet ainsi à une instance d'un enregistrement d'avoir plusieurs parents et non plus un parent unique. Le langage de manipulation des données Codasyl est un langage procédural permettant de récupérer et de manipuler un seul enregistrement à la fois. L'utilisateur accède à une donnée par un point d'entrée et navigue à travers le réseau à l'enregistrement contenant la donnée. Le fait qu'il représente une évolution logique du modèle hiérarchique, il offre une meilleure représentation des données et davantage de flexibilité que le modèle hiérarchique. Par rapport à ce dernier, le modèle réseau a permis l'élimination des redondances de données et la création de chemins d'accès multiples à une même donnée. Cependant, ce modèle ne permet pas la prise en compte réelle de phénomènes plus complexes et ne résout pas tous les problèmes d'indépendance des structures de données et des traitements.

Pour pallier ces limites, en début des années 1970, le chercheur *Edgar Frank « Ted » Codd* a inventé le *modèle relationnel* qui a révolutionné le monde *académique* et *industriel* (voir la figure 1.1). Cette invention a permis à la communauté de recherche de passer de l'ère navigationnelle à l'ère relationnelle. Ce modèle est fondé sur la *théorie des ensembles* et offre une indépendance des niveaux logique et physique. Les premières conférences internationales portant sur le thème de bases de données ont vu le jour dans les années 70. Citons comme exemple la conférence VLDB (Very Large Databases) lancée en septembre 1975, à Massachusetts, États-Unis et l'atelier SIGFIDET qui s'est déroulé pour la première fois en 1970 à Houston, Texas

et qui s'est fait appeler par la suite ACM SIGMOD (*Conference on Management of Data*) en 1975. Les efforts établis par les chercheurs ont contribué à l'émergence de trois faits importants, liés aux phases de conception, de déploiement et d'exploitation d'une application de $\mathcal{BD}$, à savoir : **(i)** la définition d'un cycle de vie de conception de $\mathcal{BD}$ composé de trois niveaux d'abstraction initialement proposés dans l'architecture ANSI/SPARC : *conceptuel*, *logique* et *physique*, **(ii)** une stabilité relative dans le temps avec une domination des SGBD relationnels et **(iii)** l'usage transactionnel en ligne qui a fait naître les applications de type OLTP (Online-Transaction Processing) en respectant les contraintes ACID (Atomicity, Consistency, Isolation, Durability). L'intérêt suscité par la communauté a poussé les chercheurs de l'université Berkeley, California a développé leur système Open Source *Ingrès*, qui a été utilisé entre 1970 et 1985.

Les industriels ne sont pas restés les bras croisés. L'entreprise IBM était la première à avoir développé le premier SGBD, appelé système **R** en 1976. La domination du modèle relationnel a également contribué au développement d'autres SGBD "Open Source" (par ex. PostgreSQL) et industriels (Oracle, DB2, SQL Serveur, Sybase, etc.). Notons que durant la période 1970-1985, *le nombre de SGBD développés autour du modèle relationnel restait raisonnable (environ 10).*

Information Retrieval — P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables

Figure 1.1 – Aperçu de l'article de Edgar Frank « Ted » Codd

La maturité du modèle relationnel a également convaincu les industriels comme *Oracle* à mener une démarche de standardisation des produits dérivés de ce modèle [comme son langage de requêtes SQL (*Structured Query Language*)] pour qu'ils soient acceptées dans le monde industriel. Ces efforts ont été concrétisés par l'acceptation de *l'Organisation Internationale de Normalisation* (ISO), affiliée à la *Commission Electrotechnique Internationale* (CEI) et *l'American National Standards Institute* (ANSI) comme langages standard pour les $\mathcal{BD}$ relationnelles.

Cette multiplication des SGBD relationnels sur le marché a engendré des problèmes en relation avec la *validation* et la *sélection* de la plateforme cible. En vu de pallier ce problème, en 1985, Jim Gray a publié un article intitulé : "A Measure of Transaction Processing Power" pour inciter la communauté à s'intéresser de plus en plus aux tests non fonctionnels à travers d'expérimentations réelle et/ou de simulation.

Cette vision du test est partagée par plusieurs communautés scientifiques. Par exemple, en physique le chercheur Lord Kelvin a dit : *"I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be"*.

En conséquence, des bancs d'essai ont été proposés offrant des schémas de $\mathcal{BD}$, des requêtes et des générateurs de données alimentant cette base de référence. Les premiers bancs d'essai ont été proposés par les universitaires. On peut citer le banc d'essai Wisconsin (développé au sein de l'Université de Wisconsin, États-Unis.), considéré comme le pionner dans la matière. Ensuite des organismes comme le TPC (Transaction Processing Council) créé en 1988 a proposé une série de bancs d'essai pour les applications transactionnelles. L'une des actions principales de l'organisme TPC est la publication de classement des SGBD selon des critères non fonctionnels via un dépôt des scénarii de tests non fonctionnels. Ce dépôt est souvent exploité par les chercheurs, les développeurs et les décideurs pour : **(i)** comparer leurs produits en se basant sur des données d'un banc d'essai et de fournir un indicateur fiable et global de qualité des produits (par exemple à des fins de publications scientifiques de résultats d'un nouveau algorithme de recherche/indexation, de maintenance des objets de bases de données avant d'acheter un SGBD/plateforme) et **(ii)** dimensionner leur système en fonction de ses besoins (avant l'achat d'une plateforme ou avant la migration d'une plateforme vers une autre).

Certes, d'autres paradigmes ont émergé comme les systèmes de $\mathcal{BD}$, orientés objet, mais ils n'ont pas pu détrôner le modèle relationnel, vu leur complexité et leur performance de traitement. Avec la demande continuelle des entreprises comme *Coca Cola* et *Walmart* pour les besoins d'analyse de la mine de données que possédaient pour augmenter leur pouvoir décisionnel, nous avons vu naître le modèle multidimensionnel avec la technologie d'entreposage des données. Edgar Frank « Ted » Codd a également apporté une plus-value à cette technologie, en introduisant douze règles définissant les fonctionnalités des systèmes décisionnels de type OLAP (*Online Analytical Processing*) [58]. Nous avons observé qu'à chaque fois qu'Ed-

gar Frank « Ted » Codd a effectué des propositions dans le domaine des $\mathcal{BD}$, la qualité de ces dernières a cru.

Parmi les exigences et les usages mis en avant par ces systèmes, nous pouvons citer particulièrement l'interactivité et la liberté offerte à l'analyste pour adapter la vue sur les données sans dégradation des performances. Pour cela, un entrepôt de données est organisé selon un modèle multidimensionnel de données. Celui-ci est défini par des axes d'analyse nommés « dimensions » et par des objets d'analyse, nommés « faits » et quantifiés par des « mesures ». Cette structure multidimensionnelle, sous forme d'un hypercube favorise l'interrogation par un outil OLAP et offre à l'utilisateur une navigation interactive orientée selon les dimensions définies au sein de l'entrepôt. Le modèle relationnel s'est invité pour stocker physiquement le modèle multidimensionnel, ce qui a donné lieu au modèle **R**OLAP (Relational OLAP). Dans ROLAP, nous trouvons la marque **R** du premier SGBD relationnel (System-R). Un autre stockage direct d'un hybercube a été proposé sous le nom "Multidimensional OLAP" (MOLAP) [187]. Il matérialise les agrégats dans des structures multidimensionnelles (tableaux multidimensionnels) à l'image de l'hypercube. Les solutions existantes sont basées sur des tableaux multidimensionnels hébergeant les cellules des hypercubes dans des structures spécialisées. Ces solutions sont parfois nommées SGBD multidimensionnels [6]. Les SGBD traditionnels comme Oracle ont proposé trois types de stockage du modèle multidimensionnel (ROLAP, MOLAP et Hybride). Les bancs d'essai ont également suivi cette évolution. L'organisme **TPC** a mis à la disposition des chercheurs et des entreprises trois bancs d'essai principaux à savoir TPC/D, TPC-H et TPC-R pour les applications décisionnelles.

Néanmoins, le succès du modèle relationnel n'échappe pas à la règle du *déclin* des civilisations, selon laquelle les civilisations suivraient un cycle de vie : *gestation, naissance, croissance, apogée, et déclin*. Ce déclin s'est vu vers la fin des années 2000, après 25 ans de règne du modèle relationnel et ses produits (SGBD, méthodes, outils, industrie, etc.) suite à l'apparition du paradigme NoSQL (Not only SQL). Ce dernier a été motivé par l'explosion des données variées du Web, la digitalisation des entreprises, l'ère numérique, les réseaux sociaux et les avancés technologiques en termes matériels (par exemple Cloud, machines parallèles, etc.), supports de stockage et des unités de traitements (par ex. GPU, FPGA, etc.) et les nouvelles exigences des entreprises en termes de scalabilité, disponibilité, élasticité, performance, etc. Le modèle relationnel était victime de la loi de marteau qui stipule : *"If the only tool you have is a relational database, everything looks like a table"*. Les modèles NoSQL ont transformé cette vision tabulaire des données, à une vision multiple (*One size doesn't fit all*). Ces modèles sont associés à des moteurs de stockage qui emploient une architecture distribuée capable de traiter des masses de données hétérogènes. Quatre classes principales caractérisent ces systèmes :

- Le modèle clé-valeur [74] qui consiste à modéliser les données de manière très déstructurée par une clé servant à identifier les données et une valeur. Les structures sous-jacentes de la valeur ne sont pas gérées par le système ; elles sont renvoyées à la charge de l'application cliente. Les applications bénéficiaires de ce modèle de stockage sont celles traitant

des données de capteurs, des logs de données, des profils des utilisateurs en exploitant des requêtes simples.

- Le modèle orienté document [56] repose sur une structuration horizontale, par imbrication de données, appelée document. Ce modèle est favorable à certains systèmes comme la gestion de contenu, le Web analytique et les catalogues de produits.

- Le modèle orienté colonne [174] structure les données verticalement, par regroupement de colonnes en familles de colonnes. Plusieurs entreprises utilisent ce modèle comme *Netflix* (pour le logging et l'analyse de sa clientèle) et *Ebay* (pour optimiser la recherche).

- Le modèle orienté graphe [104] est principalement destiné à la structuration de données en nœuds et relations formant un graphe. Ce modèle est basé sur la théorie des graphes. Neo4J est un des SGBD qui implémentent ce modèle qui est utilisé pour les réseaux sociaux, les données géo-spatiales et l'Internet des objets

Le modèle NoSQL a une caractéristique principale qui consiste à ne plus avoir un schéma commun à un ensemble de données au sein d'une même structure. Chaque donnée a son propre schéma, indépendamment des autres. Par conséquent, la notion de contraintes d'intégrité sur les structures de données n'est plus ou peu assurée par ces systèmes.

Contrairement à l'ère relationnelle, connu par le développement et la prospérité d'une poignée de SGBD, l'ère NoSQL, a vu une explosion de nombre de SGDB, car les besoins des entreprises ont réellement multiplié. La Figure 1.2 donne un aperçu des SGBD NoSQL. La plupart de ces systèmes relâchent les contraintes ACID et certains ne proposent pas de gestion de transactions.

A l'image des efforts déployés par le TPC pour le traitement traditionnelle et décisionnelle des données, le même organisme a proposé des bancs d'essai (par ex. TPCx-HS et TPCx-BB) dédiés aux applications issues des traitement des données massives.

La communauté de recherche s'est associée avec le TPC pour fédérer des chercheurs autour de la problématique de développement des bancs d'essai pour tester tout type d'application de $\mathcal{BD}$ et d'autres besoins non fonctionnels (comme l'énergie). Nous pouvons citer l'exemple de la conférence TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC) qui se déroule conjointement avec la conférence VLDB.

Suite à une recherche sur *google scholar* avec le mot clé : *database benchmark* et en choisissant la période 2005 et 2017, 131 000 résultats sont identifiés. Ce qui montre l'intérêt de la problématique liée au test et l'évaluation de la qualité des produits de bases de données. Le test concerne toutes les phases de cycle de vie de bases de données, mais les phases physique et de déploiement se taillent la part de lion des tests. Ces tests supposent que la $\mathcal{BD}$ est déjà déployée sur une plateforme donnée, ensuite l'évaluation de certains besoins non fonctionnels est établie. Ce sont des approches de validation **a posteriori**. Dans cette thèse, nous proposons une vali-

FIGURE 1.2 – Quelques SGBD NoSQL

dation **a priori**. Il s'agit de définir des modèles du système qui constituent des abstractions du SGBD et sa plateforme sur les quels des tests de validation sont réalisés.

1.1 Discussion et Problématique

En se basant sur cette discussion "historique" autour de l'évolution de la technologie des modèles de bases de données, nous concluons que toute application conçue pour gérer les données (que nous notons : A^{BD}) peut être schématisée par un 5-uplets :

$$A^{BD} :< C2V, SGBD, Plateforme, Usage, Sat >$$

où

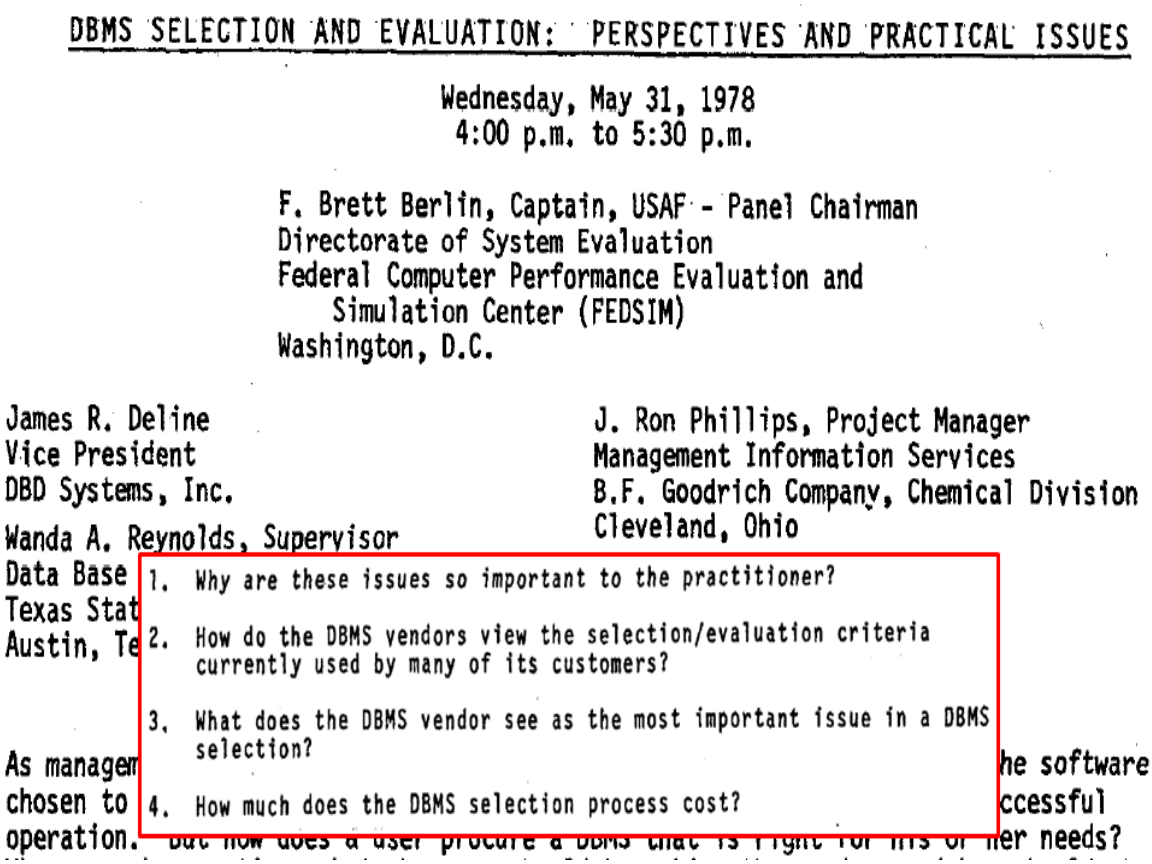
- $C2V$: représente le cycle de vie suivi pour concevoir cette application. Avec toutes les évolutions, un cycle de vie générique s'est dégagé pour concevoir tout type de $\mathcal{BD}$. Celui-ci comporte en effet un sous-ensemble des phases suivantes [111] : la collecte des besoins, la modélisation conceptuelle [180], la modélisation logique, ETL (*Extract, Transform, Load*) pour les applications décisionnelles, la phase de déploiement, la phase physique

[34] et la phase d'exploitation [88].

- $SGBD$: décrit le système de stockage utilisé pour persister les données de l'application à concevoir.
- $Plateforme$: représente la plateforme hébergeant ce SGBD. Plusieurs types de plateformes existent : centralisée, distribuée, parallèle, Cloud, etc. Toute plateforme est également associée à des supports de traitement (CPU, GPU, etc.).
- $Usage$: signifie le type de traitements effectués par le $SGBD$ (OLTP/OLAP/Mixte).
- Sat : représente la satisfaction des utilisateurs finaux de la configuration incluant le $SGBD$ et la $Plateforme$ utilisés. La satisfaction de l'utilisateur est souvent réalisée par des mesures réelles et/ou théoriques en utilisant des données réelles ou des données issues des bancs d'essai. Ces mesures sont liées à des besoins non fonctionnels comme la performance de traitements, la consommation énergétique des requêtes, etc.

La combinaison des différents critères identifiés engendre des nouvelles problématiques de test. Pour une $\mathcal{BD}$ déjà déployée seul le facteur Sat est traité ($A^{BD} :< C2V, SGBD, Plateforme, Usage, ? >$). L'étude des autres facteurs (seuls ou combinés) devra être menée par les chercheurs. Pour illustrer nos propos, le cas d'une $\mathcal{BD}$ en cours de développement son SGBD et sa plateforme sont souvent inconnus, ce qui oblige leur sélection ($A^{BD} :< C2V, ?, ?, Usage, ? >$). Cette situation a fait renaitre un ancien problème qui est la *sélection du couple plateforme et SGDB* qui a été identifié par F. Brett Berlin [30] dans SIGMOD 1978. L'auteur a soulevé la difficulté de choisir un SGBD qui convient aux besoins de l'utilisateur/les entreprises et les questions que l'utilisateur devrait demander aux vendeurs et comment évaluer les réponses de ces derniers.
Il a exhorté l'intérêt d'identifier les facteurs qui doivent être pris en compte dans le processus de sélection du SGBD, y compris les tests de présélection, les critères de sélection générale, l'évaluation des coûts, la transportabilité, la sécurité et la fiabilité. La figure 1.3 présente quelques questions soulevées dans [30] autour de la sélection et l'évaluation des SGBD, en particulier, l'importance de ce problème, l'identification des critères d'évaluation et le coût du processus de sélection des SGBD.

Sur la base de l'article de F. Brett Berlin [30] et le paysage que nous avons dessiné, nous souhaitons lancer un débat sur la nécessité de revisiter le problème de la sélection des SGBD [123] et surtout l'enrichir par la variabilité de plateforme [60]. Cela est motivé par le fait que dans les années 1970, certains chercheurs ont identifié le problème de sélection de SGBD alors que peu de SGBD existaient sur le marché. Dans notre époque, le nombre de SGDB commerciaux et académiques a fortement augmenté multiplié (voir la figure 1.2). Le même constat est aussi valable pour les plateformes de déploiement. Nous souhaitons donner quelques exemples dans ce sens. Dans [62], les auteurs ont proposé de sélectionner un SGBD pour le stockage des données d'agriculture au profit du département d'agriculture des États-Unis (le $SGBD$ de A^{BD} était inconnu). Récemment, avec l'explosion du nombre de plateformes avancées, plusieurs



DBMS SELECTION AND EVALUATION: PERSPECTIVES AND PRACTICAL ISSUES

Wednesday, May 31, 1978
4:00 p.m. to 5:30 p.m.

F. Brett Berlin, Captain, USAF - Panel Chairman
Directorate of System Evaluation
Federal Computer Performance Evaluation and
Simulation Center (FEDSIM)
Washington, D.C.

James R. Deline
Vice President
DBD Systems, Inc.

J. Ron Phillips, Project Manager
Management Information Services
B.F. Goodrich Company, Chemical Division
Cleveland, Ohio

Wanda A. Reynolds, Supervisor
Data Base
Texas Stat
Austin, Te

1. Why are these issues so important to the practitioner?
2. How do the DBMS vendors view the selection/evaluation criteria currently used by many of its customers?
3. What does the DBMS vendor see as the most important issue in a DBMS selection?
4. How much does the DBMS selection process cost?

As manage he software
chosen to ccessful
operation. er needs?

FIGURE 1.3 – Extrait de l'article montrant les questions relatives au problème de sélection des SGBD

études tentent d'évaluer un ensemble d'exigences non fonctionnelles pour les SGBD déployés sur une plateforme donnée et pour une activité spécifique. Dans [68], les auteurs évaluent la performance de *MongoDB* déployé sur une plateforme *Hadoop* pour l'analyse de données scientifiques. Cette situation est plus facile pour les entreprises, car elle suppose la connaissance préalable du SGBD et de la plateforme (les éléments de A^{BD} : $SGBD$ et $Plateforme$ sont connus et les restes des éléments sont inconnus).

Formellement, un *déployeur* (la personne qui s'occupe du processus de déploiement d'une application de $\mathcal{BD}$ A^{BD}) a le problème suivant : étant donné son manifeste décrivant les éléments suivants :

- une application de bases de données/entrepôt de données avec son schéma ;
- un ensemble de besoins fonctionnels ;
- un ensemble de besoins non fonctionnels (par exemple la performance de requêtes, la consommation énergétique, la maintenance de la $\mathcal{BD}$, etc.) ;

- un ensemble de contraintes.

Notre problème consiste à proposer une plateforme pour cette application satisfaisant l'ensemble des besoins non fonctionnels et respectant les contraintes (voir la figure 1.4).

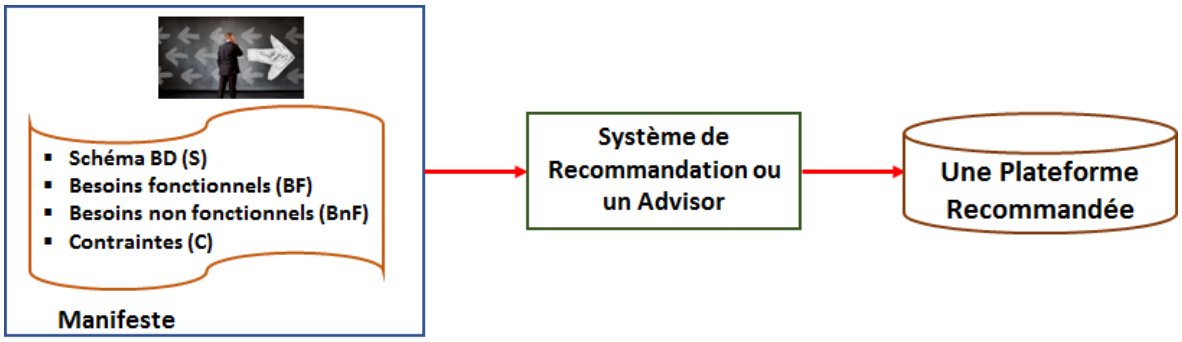


FIGURE 1.4 – Le *déployeur* et ses entrées

La complexité de notre problème est de $O(N \times M)$, où N et M représentent respectivement le nombre de SGBD et de plateformes existants.

Pour résoudre ce problème, nous avons identifié quatre approches que nous classifions dans deux catégories principales : **(a)** des approches dirigées par les tests non fonctionnels et **(b)** des approches à base de recommandations. Dans la première catégorie, nous distinguons deux solutions principales : **(1)** des solutions basées sur des tests non fonctionnels réels ($N \times M$ tests exhaustifs) qui sont souvent coûteuses [99], car elles demandent l'achat du SGBD et de sa plateforme. Elles sont souvent utilisées par les grandes entreprises telles que GAFA (Google, Apple, Facebook et Amazon) qui possèdent à la fois les données et les plateformes avancées. *Microsoft consacre par exemple 50 % de ses coûts de développement aux essais*. De même, le cycle de sortie des produits SAP dure 18 mois dont six mois sont consacrés uniquement à l'exécution des tests. **(2)** des solutions à base de simulation à l'aide des modèles de coût mathématiques estimant les métriques utilisées pour quantifier les besoins non fonctionnels. Le simulateur doit prendre en considération toutes les composantes d'un SGBD, la plateforme, la $\mathcal{BD}$, et les requêtes (la présence des sélections, des jointures, etc.) [146]. Cette maîtrise de ces paramètres est nécessaire pour définir des modèles de coût mathématiques.

Dans la deuxième catégorie, nous distinguons également deux sous-catégories : **(1)** des solutions dirigées par le retour d'expérience des utilisateurs/développeurs, comme ce qu'il propose *DB-Engine* [1]. *DB-Engine* publie un classement mensuel des SGBD depuis 2012, où environ 315 candidats étaient en lice pour le titre du SGBD le plus populaire. Dans le cadre de la thèse de Selma Bouarar [32], nous avons analysé l'ensemble des résultats publiés sur ce site. Nous avons distingué trois stades temporels : les deux premières décennies de "maturation"

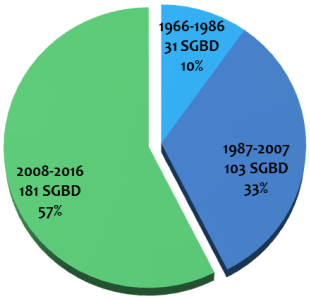
1. http://db-engines.com/en/ranking

de SGBD (1966-1986), les deux décennies de "maturité" (1987-2007) et les dernières années "dynamiques" (depuis 2008). Les résultats illustrés dans la figure 1.5 montrent que :

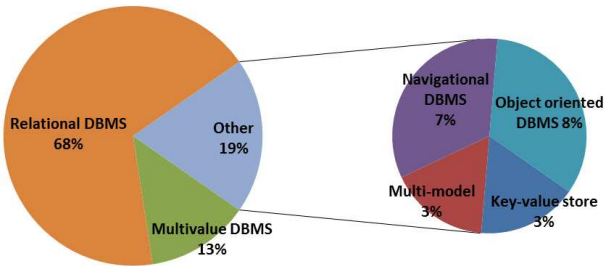
- Plus que la moitié des SGBD a été développé pendant le stade actuel, ce qui témoigne d'une croissance élevée des inventions et de la vulgarisation multidisciplinaire des $\mathcal{BD}$. De plus, cette croissance a tendance à augmenter dans les années à venir vu l'engouement de plus en plus d'organisations et d'individus pour la production, le stockage et l'analyse des données, et la divergence de leur exigences.

- Les types des modèles de données sous-tendant les SGBD se sont multipliés au fur et à mesure que les périodes se succèdent (de 6 à 14). En effet, le paysage des bases de données a connu, ces dernières années, un spectaculaire changement avec de nouveaux fournisseurs et beaucoup de nouvelles technologies qui se sont désormais imposées pour certaines applications, notamment dans le *Big Data*. Cette diversité a d'ailleurs été soulignée par le dernier numéro du fameux rapport de *Beckman* sur la recherche en *BD* [1].

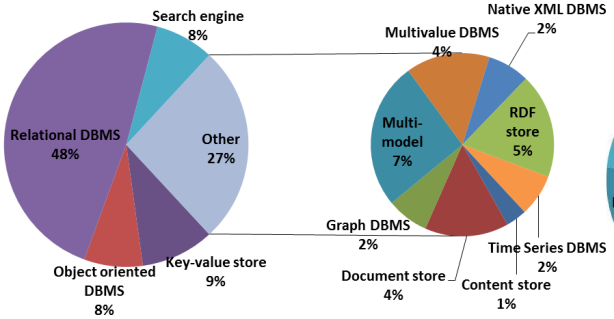
- Le modèle relationnel sous-tendait environ 70% des SGBD existants lors du stade initial des SGBD, contre une proportion de 50% lors du stade mature, jusqu'à atteindre seulement 25% pendant cette dernière décennie (depuis 2008). Malgré cela, le relationnel reste le leader en matière d'usage, grâce aux quatre éditeurs historiques : *Oracle*, *Microsoft*,



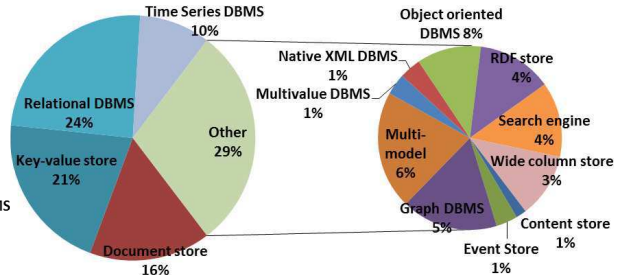
(a) Répartition du nombre de SGBD sur les 3 stades



(b) Types de SGBD inventés pendant le stade de maturation



(c) Types de SGBD inventés pendant le stade de maturité



(d) Types de SGBD inventés pendant le stade actuel

FIGURE 1.5 – Évolution en nombre et en type des SGBD

IBM et *SAP*. En parallèle, les technologies NoSQL sont portées par une belle dynamique, à commencer par *mongoDB* qui s'est même hissée à la 4ème place au classement des leaders devant *PostgreSQL*. *Cassandra* et *Redis* à la 7 et 9ème place respectivement, alors que trois ans de cela, aucun SGBD non-relationnel ne figurait dans le Top 10.

- Ce site ne prend pas compte les SGBD dédiés aux $\mathcal{BD}$ sémantiques comme IBM SOR [81].

(2) Des solutions basées sur les résultats des organismes spécialisés dans les bancs d'essai et les tests, comme le TPC. Ce dernier publie régulièrement les résultats de performance des SGBD sur des problématiques générales et des plateformes en fonction de leur banc d'essai. La figure 1.6 résume l'ensemble des solutions de choix de SGBD et de plateformes. Malheureusement, ces solutions concernent un certain type de SGBD, souvent dédiés aux grandes entreprises.

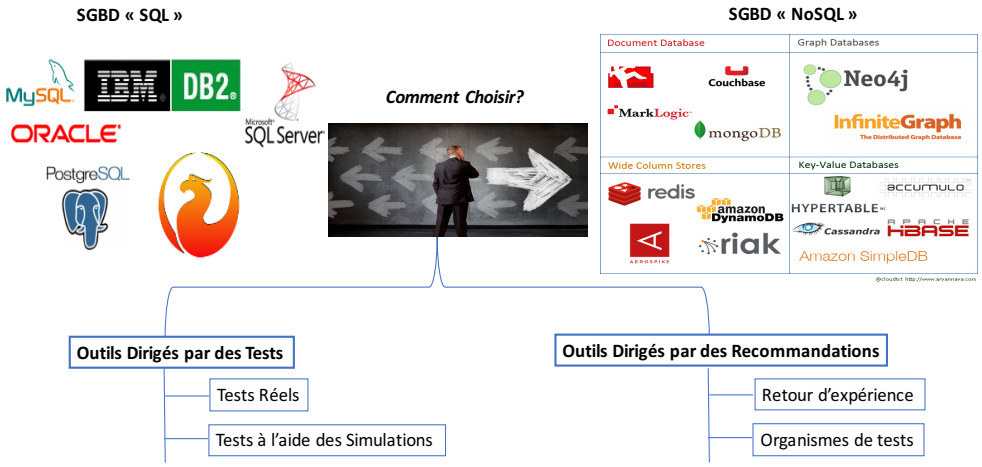


FIGURE 1.6 – Catégories de choix de SGBD et de plateformes

Discussion. La revue des approches de résolution de notre problème a mis en avant leur complexité en termes de coût d'exploration de l'espace de recherche et la partialité des résultats fournis par les Sites Web.

Cela nous a motivés de réduire cette complexité et incorporer d'autres ressources externes dans la prise de décision. Pour ce dernier point, il est intéressant de rappeler que les chercheurs passent leur grande partie de leur temps consacré à la recherche à tester les propositions. Ces tests sont souvent effectués sur des SGBD et des plateformes. Les résultats numériques de ces tests sont publiés dans des conférences et journaux spécialisés. Si nous considérons seulement les conférences et les revues des systèmes de $\mathcal{BD}$ et d'information, chaque année, plus de 80% des articles scientifiques fournissent des expériences intensives pour évaluer et comparer leurs solutions. Si nous prenons l'exemple de la conférence VLDB, édition 2016, 100 articles ont été publiés. Si ces données de tests sont stockées dans un vrai entrepôt de données avec un schéma

bien défini, le choix de SGBD et sa plateforme peut se faire d'une manière a priori et d'une manière efficace. Cet entrepôt de données stockera un sous-ensemble de l'espace de recherche global ($N \times M$).

Des expériences similaires dites a priori dans le contexte de $\mathcal{BD}$ existent. Prenons par exemple le cas du site DBLP qui recense environ 3 754 344 articles. Vu l'intérêt de la communauté scientifique sur l'utilisation de ce site et le nombre important d'articles scientifiques, un banc d'essai a été défini portant le nom de ce site. Il a été largement utilisé dans le contexte des $\mathcal{BD}$ XML, sémantiques [71] et récemment les réseaux sociaux [46]. Le schéma logique de ce banc a été extrait à partir des publications stockées sur le serveur DBLP (dont un fragment de son schéma est décrit dans la figure 1.7). Ce modèle contient des méta données associées aux articles. Toutefois, le contenu de ces derniers, en termes des résultats des expérimentations et ainsi que leurs environnements de tests, reste difficile à extraire. Cependant, la démarche de DBLP reste intéressante et elle pourrait être reproduite pour notre cas d'étude, puisqu'elle représente l'exemple par excellence de l'intérêt des approches d'exploitation des données scientifiques *d'une manière a priori*. Si nous focalisons sur les efforts établis par les chercheurs pour concevoir ce banc d'essai (DBLP), nous réclamons qu'ils ont utilisé les étapes suivantes : **(1)** la définition d'un schéma conceptuel ou logique de leur banc d'essai (voir la figure 1.7), **(2)** son peuplement manuel ou automatique et **(3)** son exploitation (par les chercheurs). Nous nous sommes inspirés de cette expérience pour concevoir notre démarche de résolution de problème de sélection de SGBD et sa plateforme.

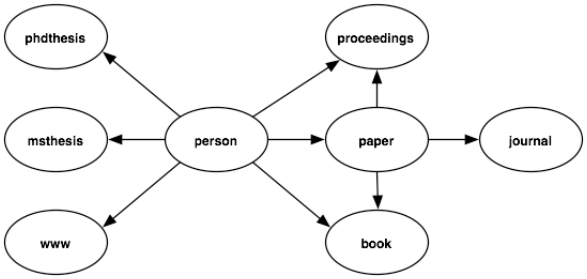


FIGURE 1.7 – Le Schéma de la base DBLP

1.2 Exploitation des données de tests localisées dans les articles scientifiques

Comme nous l'avons déjà mentionné la communauté de recherche de $\mathcal{BD}$ consacre beaucoup d'efforts, d'énergie, et d'argent pour tester ses produits (couvrant l'ensemble des phases de cycle de vie de conception) pour évaluer leur qualité. Si nous nous concentrons sur les phases de déploiement et physique qui doivent satisfaire des besoins non fonctionnels, les résultats de tests publiés par les chercheurs et par les organismes de bancs d'essai prennent en compte des dimensions suivantes : (a) les données utilisées pour les tests, (b) les algorithmes utilisés, (c)

la plateforme matérielle, (d) les besoins non fonctionnels, (e) les traitements (requêtes), (f) les structures d'optimisation utilisées, etc. Les résultats publiés sont exploités et analysés par les chercheurs pour positionner et comparer leurs travaux. Mais en les analysant, nous identifions qu'elles fournissent d'autres connaissances cachées. Vu leurs caractéristiques, les données de tests représentent un bon exemple de données obscures (*Dark Data*). Selon IDC (*International Data Corporation*) [2] et EMC [3], le monde comportera 40 zettabytes de données d'ici 2020. D'après le sondage Veritas Global Databerg Survey 2016, 85% de ces données seront des données obscures. D'après Gartner, les données obscures sont les ensembles d'informations que les organisations collectent, traitent et stockent pendant leurs activités régulières, mais n'arrivent pas à réutiliser.

Comme nous l'avons déjà mentionné, dans cette thèse nous proposons une démarche d'explicitation et de persistance de toutes les dimensions des environnements de tests ainsi que les résultats de tests afin de les sortir de l'obscurité. Une fois persistées, ces données accompagnées de leurs environnements peuvent être exploitées par des outils d'OLAP et un système de recommandation assistant les entreprises à choisir leur SGBD et leur plateforme (voir la figure 1.8). Pour exploiter cet environnement de tests non fonctionnels persisté, le *déployeur* se présente avec son manifeste. Si une similarité entre ses entrées (son schéma, ses besoins fonctionnels et non fonctionnels) et celles de l'entrepôt, notre environnement lui retourne des recommandations. Dans le cas contraire, un processus de raffinement de son manifeste est nécessaire (voir la figure 1.9).

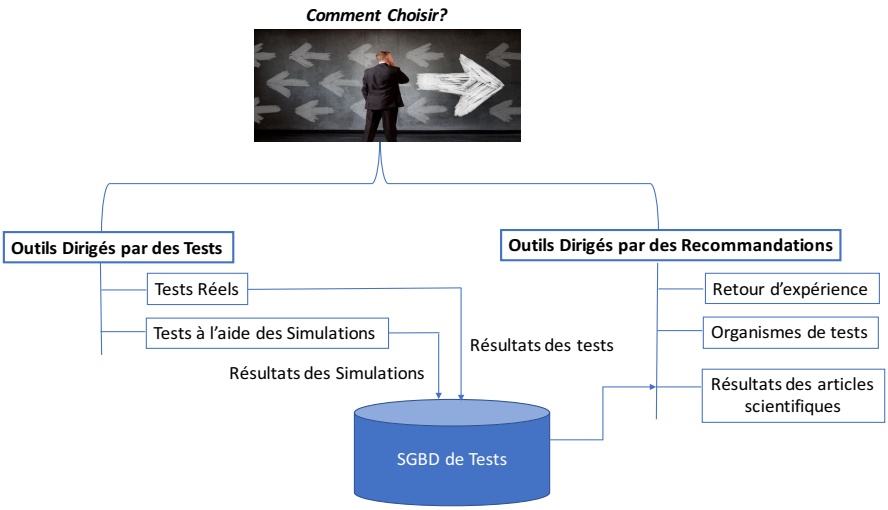


FIGURE 1.8 – Recommandations à partir des articles scientifiques

2. Une entreprise américaine de recherche spécialisée dans les technologies de l'information
3. Richard Edgar Roger Marino Corporation : est une entreprise américaine de logiciels et de systèmes de stockage fondée en 1979

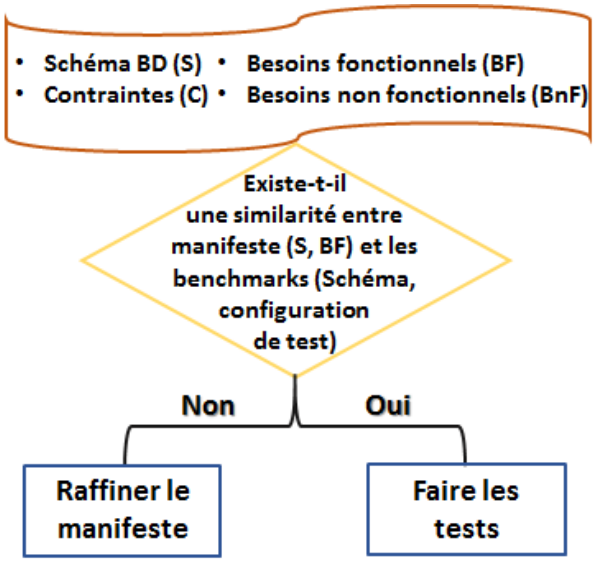


FIGURE 1.9 – Une approche de résolution de notre problème

2 Objectifs et contributions

Dans cette thèse, nous proposons une démarche de capitalisation des données de tests non fonctionnels obtenues par les chercheurs et les industriels pour assister les entreprises et les utilisateurs pour choisir leurs SGBD et leurs plateformes.

Pour ce faire, nous proposons la démarche illustrée dans la figure 1.10 qui peut être instanciée pour tout domaine générateur de données de test non fonctionnels publiées dans des articles scientifiques et/ou sources. Elle est composée de trois étapes principales : (1) l'analyse des articles, (2) la persistance des données de test dans un dépôt, et (3) l'exploitation de ce dépôt.

Vu les compétences que nous avons au sein du Laboratoire d'Informatique et d'Automatique pour les Systèmes (LIAS) du l'ISAE-ENSMA et l'Université de Poitiers sur les phases du cycle de vie de la base de données [111], nous proposons d'instancier cette démarche sur deux phases fortement dépendantes à savoir la phase de déploiement et la phase physique.

Pour ce faire, nous nous sommes fixés les objectifs illustrés par la figure 1.10 :

1. Rendre explicite l'ensemble des dimensions et les éléments impliqués par le processus de tests concernant les deux phases de déploiement et physique. La figure 1.11 donne un aperçu d'un environnement de test de l'article scientifique, de Sanjay Agrawal, Surajit Chaudhuri, Vivek R. Narasayya, intitulé : *Automated Selection of Materialized Views and Indexes in SQL Databases* [7], publié dans VLDB 2000. A partir de cet article, nous identifions clairement certaines dimensions comme la $\mathcal{BD}$ et la charge de requêtes. Dans le domaine de $\mathcal{BD}$, les articles scientifiques suivent souvent la même structuration (cf.

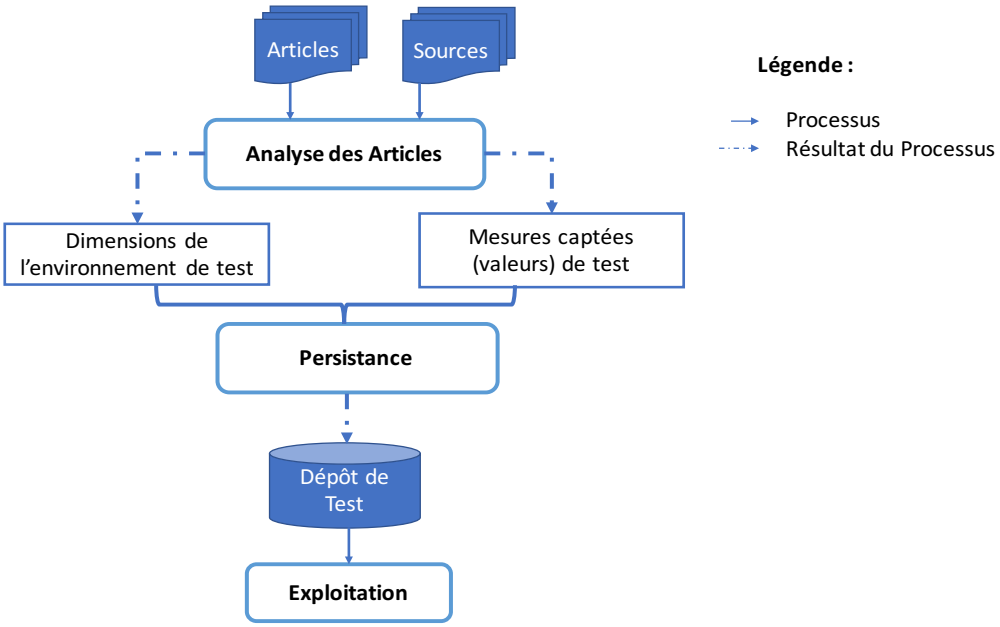


FIGURE 1.10 – Notre démarche

la base de DBLP) pour présenter leurs résultats scientifiques. Cette structuration nous a permis d'identifier le schéma de notre entrepôt de données.

Cette explicitation permettra aux usagers de bien cibler leur recherche et surtout avoir les sens sans ambiguïté des objets manipulés dans ce processus.

2. Extraire les données de tests à partir des articles scientifiques et le mettre dans un dispositif de stockage unique qui permet aux utilisateurs de l'exploiter. Cette extraction est suivie par une factorisation des données selon leurs dimensions respectives.

3. Offrir un outil décisionnel sur les données de tests afin de dégager les tendances sur l'utilisation des différentes dimensions.

4. Recommander, sur la base des données de tests stockées, les SGBD et leurs plateformes.

Nos contributions répondent aux objectifs ci-dessous cités comme suit (voir la figure 1.12) :

- Un état de l'art sur l'ensemble des travaux développés couvrant les phases de déploiement et physique de cycle de vie de conception de $\mathcal{BD}$. Cette étude nous permet d'identifier des dimensions pertinentes utilisées dans tout environnement de test. Elles couvrent le schéma de la base données, les requêtes, la plateforme, le SGDB, les unités de traitements, les besoins non fonctionnels, les hypothèses, les algorithmes utilisés, etc. La particularité de ces dimensions est qu'elles couvrent l'ensemble de phases de cycle de vie, en général, et les éléments de notre schéma, en particulier ($A^{BD} :< C2V, SGBD, Plateforme, Usage, Sat >$).

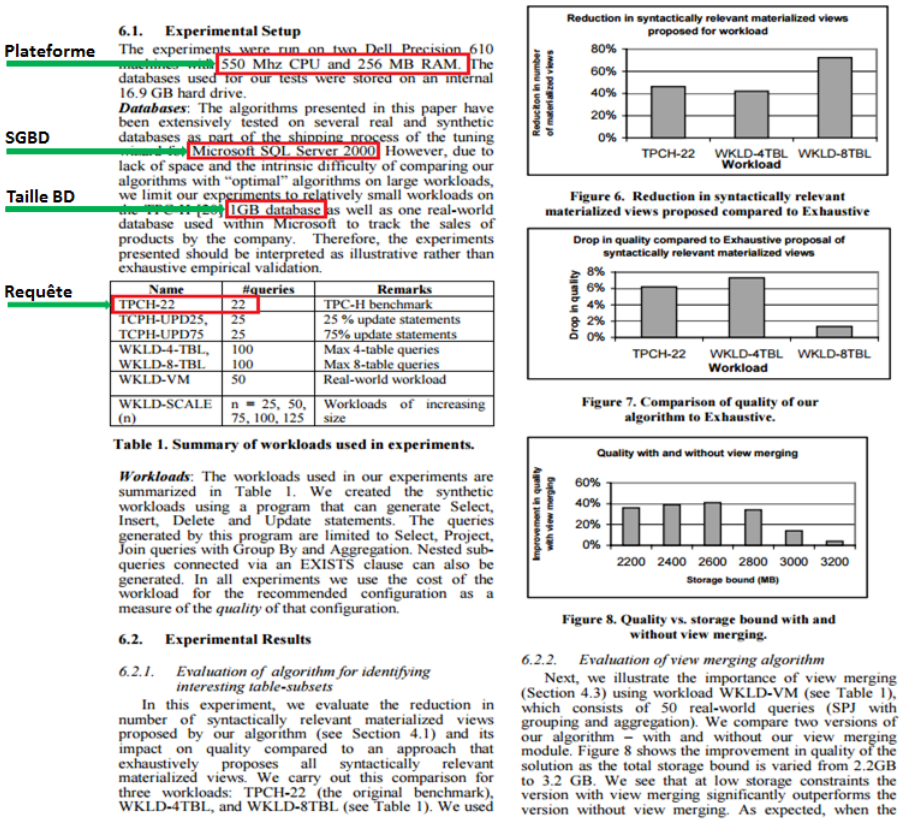


FIGURE 1.11 – Quelques dimensions de tests extraites de [7]

- A l'aide des ontologies et des méta-modèles, nous explicitons l'ensemble de dimensions.
- Proposition d'un entrepôt de données multidimensionnel associé à des outils OLAP afin d'explorer et naviguer sur les résultats de tests.
- Conception d'un système de recommandation des SGBD les plus appropriés à une demande en basant sur les modèles proposés auparavant et les exigences exprimées par les entreprises. Ce modèle est basé sur des techniques d'apprentissage exploitant les données de tests stockées dans notre entrepôt.

3 Organisation de la thèse

Cette thèse est organisée en deux parties. La première partie présente un état de l'art sur les différentes évolutions technologiques qui ont contribué à l'apparition des nouvelles méthodologies de conception de bases de données. Ces dernières sont la cause principale du problème de sélection des plateformes pour le déploiement d'une base de données. Une description détaillée sur les phases physique et déploiement de conception de bases de données seront présentées. La

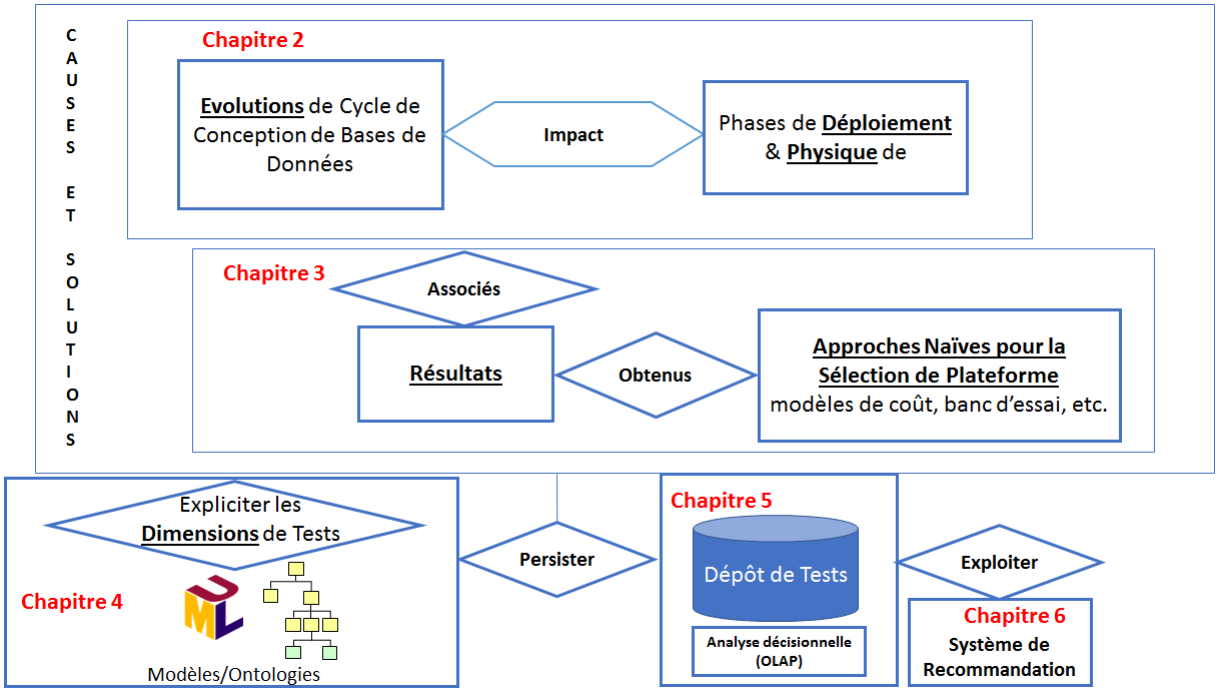


FIGURE 1.12 – Aperçu globale des contributions de la thèse

conception physique a toujours été le premier centre d'intérêt de la recherche en bases de données [32]. Cela peut s'expliquer par l'aspect d'entonnoir que prend cette phase en ayant pour place la dernière, et pour entrées les résultats des différentes autres phases (modèle logique et de déploiement) à traduire vers la sémantique du SGBD cible. La complexité et la diversité des tâches sont aussi pour beaucoup. D'ailleurs, la majorité des besoins non fonctionnels sont évalués. Cette description va nous permettre de dégager l'ensemble de dimensions impliquées dans le processus de tests.

Dans la deuxième partie, nous développons nos contributions sur la sélection des systèmes de gestion de base de données et de plateformes en exposant un dépôt de tests, une ontologie des systèmes de bases de données et un système de recommandation.

- **Première partie**
  - **Chapitre 2** permet de présenter l'évolution de la technologie de bases de données et son impact sur le cycle de vie. Ce chapitre consiste également à identifier trois évolutions principales : verticale, interne et horizontale et à détailler l'impact de cette évolution sur les phases de déploiement et physique et surtout leur dépendance.
  - **Chapitre 3** présente les approches utilisées dans le processus de sélection des plateformes, à savoir les techniques de tests réels (bancs d'essai), la simulation (modèles de coût), le retour d'expérience et d'usage et des solutions d'élagage (classification des SGBD).

- **Deuxième partie**

  - **Chapitre 4** décrit notre première contribution, à savoir, l'identification et l'explicitation de l'ensemble de dimensions des environnements de tests. Dans ce chapitre nous souhaitons partager avec le lecteur notre démarche d'explicitation. D'abord, nous avons commencé par la description de chaque dimension à l'aide des techniques issues de l'ingénierie dirigée par les modèles. Une fois construit, nous avons identifié la maturité et surtout la consensualité sur l'ensemble de concepts caractérisant certaines dimensions, comme la dimension représentant les SGDB. Cela nous a motivés de proposer des ontologies pour ces dimensions.
  - **Chapitre 5** Les modèles proposés dans le chapitre 4 sont les briques principales de notre entrepôt de données de tests. Chaque dimension est traduite par une table de dimension dans le jargon d'entreposage de données, et la table des faits contient les valeurs associées aux tests impliquant les différentes dimensions. Il est important de noter que le peuplement de notre entrepôt se fait manuellement à partir des articles scientifiques. La présence de cet entrepôt nous permet de l'associer à l'outil OLAP pour permettre aux usagers d'effectuer des analyses de données de tests et leurs environnements.
  - **Chapitre 6** présente notre système de recommandation et ses deux niveaux : sélection fonctionnelle et section non fonctionnelle. Ce système est construit à partir de l'entrepôt de données de tests et de l'ontologie des SGBD pour conseiller les entreprises à choisir le SGBD le plus approprié en fonction de leurs besoins. Ces niveaux sont basés sur les techniques d'apprentissage automatique et la décision multi-critères. Ainsi, nous exposons notre outil d'aide à la recommandation des SGBD et ses composantes en détaillant son architecture. Nous présentons aussi quelques cas d'utilisation et de scénarios de validation.

Enfin, nous concluons cette thèse en rappelant les différents travaux présentés et les résultats obtenus. Des perspectives seront également envisagées.

4 Publications

La liste suivante présente l'ensemble de productions réalisées dans le cadre de cette thèse, qui peuvent ajouter une pierre à l'édifice. Elles s'affectent aux chapitres de la manière suivante :

- Chapitre 4 [39, 38]
- Chapitre 5 [36, 39]
- Chapitre 6 [37, 38]

Publications internationales

- Lahcene BRAHIMI, Yassine OUHAMMOU, Ladjel BELLATRECHE, Abdelkader OUARED, More Transparency in Testing Results : Towards an Open Collective Knowledge Base, 10th IEEE International Conference on Research Challenges in Information Science (RCIS 2016), edited by IEEE, 2016, pp. 315-320 [39].
- Lahcene BRAHIMI, Ladjel BELLATRECHE, Yassine OUHAMMOU, A Recommender System for DBMS Selection Based on a Test Data Repository, 20th East-European Conference on Advances in Databases and Information Systems (ADBIS 2016), LNCS, edited by Springer, 2016, pp. 15 [37].
- Lahcene BRAHIMI, Ladjel BELLATRECHE, Yassine OUHAMMOU, Coupling Multi-Criteria Decision Making and Ontologies for Recommending DBMS, 22nd International Conference on Management of Data (COMAD 2017) [38].

Publications nationales

- Lahcene BRAHIMI, Vers une Suite Décisionnelle dédiée aux Données de Tests, Actes du 8 ièmes Forum Jeunes Chercheurs, 34 ièmes Congrés INFORSID 2016 (FJC-INFORSID 2016), 2016, pp. 61 [36].

Première partie

État de l’art

Chapitre 2

Évolution du cycle de conception de bases de données

Sommaire

1 Introduction 27

2 Les Causes de problème de sélection de plateforme : Évolution de cycle de vie 28

2.1 Évolution verticale du cycle de conception 29

2.1.1 Edgar Codd et le cycle de vie 30

2.1.2 Peter Chen et le cycle de vie 31

2.1.3 Omniprésence des besoins fonctionnels et non fonctionnels 33

2.1.4 Nouvelles applications et le cycle de vie 34

2.1.5 Émergence des SGBD et des plateformes 35

2.2 L'évolution interne du cycle de conception 36

2.2.1 Définition des besoins 37

2.2.2 La modélisation conceptuelle 38

2.2.3 La modélisation logique 39

2.2.4 La phase de déploiement 40

2.2.5 La modélisation physique 45

2.3 Évolution horizontale du cycle de conception 47

2.3.1 Les SGBD orientées objet 48

2.3.2 Diversification des modèles conceptuels 49

2.3.3 Évolution des besoins non fonctionnels 51

2.3.4 Apparition des SGBD Nosql et NewSQL 52

3 Bilan 53

4 Conclusion 54

Résumé. Ce chapitre présente en détail les causes du problème de sélection de plateforme pour une application de $\mathcal{BD}$. L'identification de ces causes nous permettra de le cerner et de le résoudre efficacement en prenant en compte ses difficultés et ses contraintes. Après une analyse approfondie des travaux existants, nous avons observé que ces causes sont dues principalement aux différentes évolutions qu'a subies la technologie de $\mathcal{BD}$ et les avancées logicielles et technologiques. Ces évolutions n'ont pas seulement impacté que le cycle de vie de conception de $\mathcal{BD}$, mais elles ont également contribué à la naissance des nouvelles méthodologies de conception et de *nouveaux métiers*. Dans la première génération de $\mathcal{BD}$, un seul métier existait, à savoir le concepteur. Après l'explosion de la volumétrie des données et les architectures de plateformes de déploiement (comme le client/serveur), le métier d'administrateur a vu le jour. La technologie des entrepôts de données et l'analyse en ligne ont fait naitre deux métiers : le data analyst et le data architect. Nous revendiquons à travers cette thèse la création d'un nouveau métier à savoir le déployeur (celui qui choisit une plateforme pour son application), vu l'explosion des SGBD et les plateformes. Le développement des outils assistant les entreprises à choisir leur plateforme est devenu un enjeu stratégique et économique important. La particularité de ces métiers est qu'ils se sont créés suite à l'évolution du cycle de vie de conception de $\mathcal{BD}$. Afin de montrer le lien entre ces métiers et les phases de cycle de vie, nous décrivons d'une manière synthétique les différentes évolutions de cycle de vie de conception de $\mathcal{BD}$. À la fin de ce chapitre, nous aurons les notions et concepts fondamentaux pour formaliser notre problème et surtout le résoudre.

1 Introduction

Depuis peu, nous assistons à une explosion exponentielle de données au sein des entreprises, sur Web, l'Internet des objets (IoT), les réseaux sociaux, etc. Cette volumétrie ne va pas cesser d'augmenter. D'après les experts de la donnée, en 2020, nous atteindrons 35 zettaoctet de données exploitables [4]. Cette situation a fait naitre un nouveau besoin important auprès des entreprises et des organisations qui consiste à approprier de cette masse de données, de la stocker et de l'exploiter efficacement, tout en prenant en compte leurs besoins non fonctionnels comme la performance des traitements, la minimisation de la consommation énergétique du matériel informatique, le prix, etc. Dans le contexte actuel, l'exploitation de cette masse de données est passé par le traitement et l'utilisation transactionnelle (de type OLTP), au traitement analytique avec des outils de types OLAP et de fouille de données, voire les deux en même temps (usage mixte de type OLTP/OLAP).

Les données sont devenues le nerf de la guerre. Par conséquent, les gouvernements ont déployé des efforts considérables concrétisés par des grands programmes (Big Data@ NSF [5] et les masses de données pour les sciences de l'Agence Nationale de la Recherche, France) pour répondre aux besoins que nous avons cités dans plusieurs domaines d'application sont demandeurs comme la science, la société, l'environnement, l'énergie, la médecine, la cybercriminalité, le terrorisme, le transport intelligent, etc.

L'explosion des données et le besoin de les analyser ont donné du grain à moudre aux quatre communautés de recherche, à savoir : la $\mathcal{BD}$ (pour acquérir, nettoyer, stocker, exploiter efficacement cette masse de données), *l'apprentissage statistique* (pour la prédiction et la fouille dans cette masse), le Web Sémantique (pour comprendre les données non structurées du Web) et le hardware (pour offrir des dispositifs de stockage et de traitement puissants et économiques).

En ce qui concerne la communauté de $\mathcal{BD}$, nous constatons une transformation sans précédente de son paysage. Nous sommes passés des SGBD traditionnels aux NewSQL en passant par les SGBD NoSQL. Ces SGBD sont accompagnés par des langages de requêtes divers et variés (par exemple SQL, CQL, SPARQL, XQuery, etc.), des nouveaux paradigmes de traitement des données distribuées (par exemple MapReduce, Hadoop, etc.) et structures d'optimisation (par exemple les index, le partitionnement, la réplication, les vues matérialisées, data compression, factorisation, etc.), des bancs d'essai (par exemple TPC-H, Ycsb, Ycsb++, etc.) ont été crées pour évaluer des besoins non fonctionnels des SGDB.

La communauté d'apprentissage statistique a déployé des efforts considérables pour proposer des modèles prédictifs (l'apprentissage symbolique, les réseaux bayésiens, les réseaux de neurones, etc.) qui apprennent à partir de données existantes afin de prévoir les tendances, les résultats et les comportements futurs. Ces efforts ont été motivés par l'explosion des ordinateurs avec des puissances de calcul importantes.

4. http://www.usinenouvelle.com/article/big-data-fait-parler-vos-donnees.N168880text
5. https://www.nsf.gov/cise/bigdata/

La communauté de Web Sémantique a mis à disposition aux chercheurs et entreprises des ontologies permettant d'une manière automatique d'annoter les documents, d'intégrer des sources de données, indexer les ressources informatiques, expliciter le sens des documents et des sources. Nous pouvons citer l'exemple des ontologies Yago [175] et DBpédia [119].

Finalement, la communauté hardware a fait un progrès considérable en mettant sur le marché des plateformes de déploiement (centralisées, parallèles, distribués, etc.) avec des supports de traitements dédiés à optimiser les traitements, comme le GPU.

Comme il a été déjà indiqué par les sociologues (comme Georges Friedmann [6]), tout progrès technologique a des conséquences importantes sur l'homme, la société, et les institutions. Une des conséquences qui nous intéresse dans le cadre de cette thèse concerne la vie du *déployeur*. *Cette personne doit choisir un SGBD et sa plateforme qui satisfait ses besoins fonctionnels.* Ce choix est souvent compliqué à faire, vu une large panoplie de plateformes.

Dans ce chapitre, nous présentons les causes qui ont rendu ce choix complexe. Ces causes proviennent principalement de l'évolution du cycle de vie de conception de $\mathcal{BD}$ et les avancées technologiques. L'analyse de ces causes nous permet de dégager les éléments pertinents pour résoudre notre problème.

2 Les Causes de problème de sélection de plateforme : Évolution de cycle de vie

Dans l'introduction générale, nous avons évoqué brièvement l'impact de l'évolution de la technologie (logicielle et matérielle) des $\mathcal{BD}$ sur leur cycle de vie. Dans cette section, nous présentons les différentes évolutions de ce dernier. Dans la thèse de Selma Khouri, effectuée au laboratoire LIAS de l'ISAE-ENSMA, nous avons identifie trois principales évolutions (voir la figure 2.1) [111] :

- une évolution *verticale* qui concerne l'ajout de nouvelles phases de cycle de vie. Cet ajout est motivé par les exigences de nouvelles applications (comme le cas ETL pour les entrepôts de données) et l'émergence des plateformes de déploiement ;
- une évolution *interne ou intra-phase* propre à chaque phase. Avec l'évolution globale des $\mathcal{BD}$, chaque phase est devenue de plus en plus complexe et à son tour, est associée également à son propre cycle de vie ;
- une troisième évolution *horizontale* qui a permis d'enrichir chaque phase par l'apparition de nouveaux modèles de données, structures d'optimisation, plateformes, supports de traitement et techniques d'exploitation de données.

6. http://unesdoc.unesco.org/images/0005/000594/059467fo.pdf, page 251

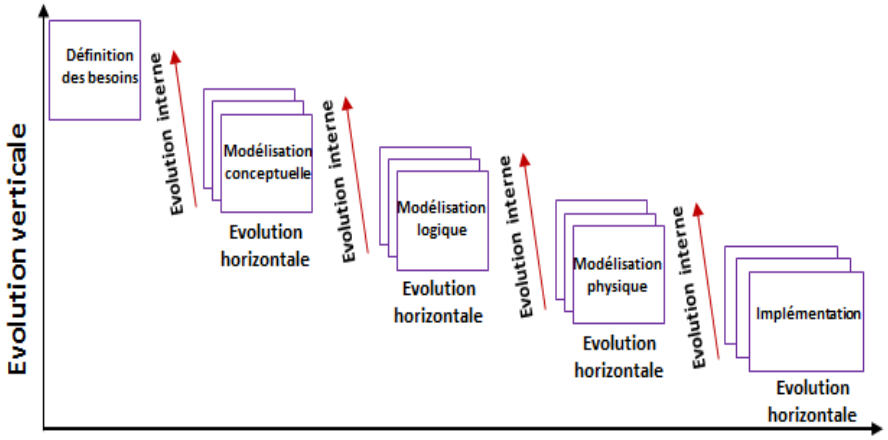


FIGURE 2.1 – Types d'évolution du cycle de conception des BD : horizontale, verticale, interne

Ces évolutions sont détaillées dans les sections suivantes.

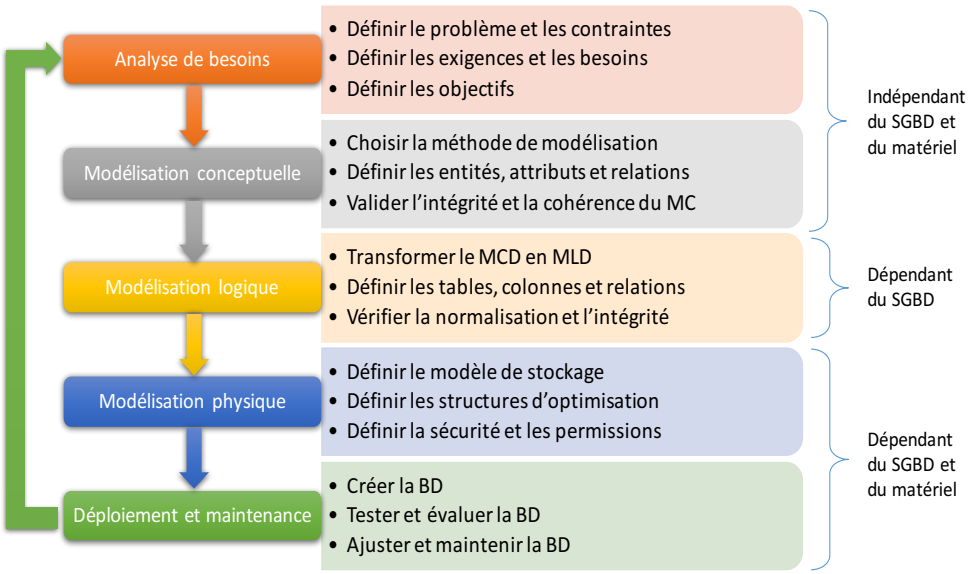


FIGURE 2.2 – Les phases du cycle de vie et leurs rôles

2.1 Évolution verticale du cycle de conception

Avec l'apparition des $\mathcal{BD}$, leur cycle de vie pourrait être vu comme un embryon qui a évolué au fil de temps, jusqu'à où il est devenu le cycle de développement de $\mathcal{BD}$ par excellence. Dans la première génération de $\mathcal{BD}$, le cycle de vie concernait que la *phase physique*, qui avait une forte relation avec les modèles de données existants. A cette époque, ces derniers couvraient deux principaux modèles qui ont connu un certain succès et ont été utilisé dans des systèmes industriels : le modèle *hiérarchique* (voir la figure 2.3) et le modèle *réseau* (voir la figure 2.4).

Ces deux modèles souffraient une limite majeure matérialisée par la *faible indépendance physique et logique des données*. Ceci nécessitait de prévoir toutes les données et opérations nécessaires avant la conception de la $\mathcal{BD}$, afin d'éviter de re-coder d'éventuels changements ou mises à jour. Les modèles proposés étaient certes efficaces pour les requêtes et les opérations initiales pour lesquelles la $\mathcal{BD}$ a été conçue. Par contre, ils ne fournissaient pas suffisamment de souplesse pour accéder aux enregistrements efficacement lorsque de nouvelles requêtes et transactions étaient identifiées. Ces transactions étaient souvent coûteuses en termes de temps de traitement et en argent. Cela est dû à la réécriture et aux tests des programmes [77].

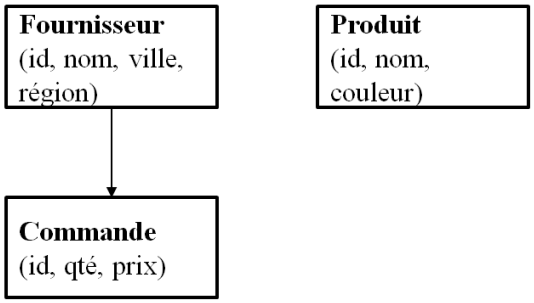


FIGURE 2.3 – Exemple d'une représentation hiérarchique

2.1.1 Edgar Codd et le cycle de vie

Le monde des $\mathcal{BD}$ a connu un bouleversement important lorsqu'en 1970 *Edgar Codd* a proposé le modèle *relationnel* en remplacement des traditionnels modèles navigationnels [57]. L'importance de cet apport a d'ailleurs valu à *Edgar Codd* d'obtenir le prix Turing en 1981 [59].

Le modèle relationnel répond à la nécessité d'assurer l'indépendance physique des données. En particulier, il permet une séparation entre les deux niveaux d'abstraction des données, que sont les niveaux logique et physique, en utilisant une abstraction mathématique du contenu implémenté dans la $\mathcal{BD}$ [103]. Dans ce modèle, les données stockées sont organisées selon des structures simples et flexibles (des tables) qui peuvent s'adapter à tout type d'information. Ces données sont ensuite gérées en utilisant leur valeur, et non par leur position (comme c'est le

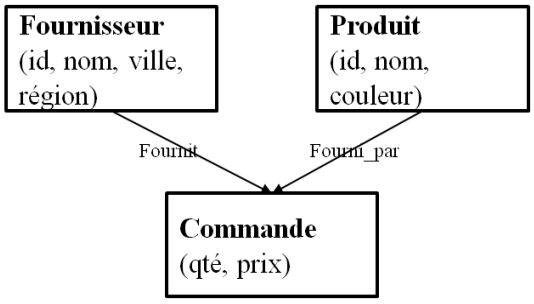


FIGURE 2.4 – Exemple d'une représentation en réseau

cas dans le modèle réseau), car cette dernière est souvent difficile à maintenir. Quant à l'accès aux données, il s'effectue par le biais d'un langage de manipulation qui exploite les concepts de la théorie des ensembles afin de permettre aux utilisateurs de la $\mathcal{BD}$ d'exprimer des opérations s'appliquant simultanément sur plusieurs blocs ou ensembles d'informations. Enfin, le modèle relationnel se base sur l'algèbre relationnelle pour représenter de façon formelle et rigoureuse l'organisation et la gestion de la $\mathcal{BD}$. Dans ses travaux, *Edgar Codd* définit le modèle relationnel en mettant en évidence trois composantes essentielles [59] :

- la *partie structurelle* représente la collection de types de structures de données, c'est-à-dire, pour le modèle relationnel, les domaines, les relations, les attributs, les tuples, les clés candidates et primaires.
- la *partie manipulation* correspond à la collection d'opérateurs et de règles d'inférence pouvant être appliqués sur des instances valides de la $\mathcal{BD}$ en vue d'obtenir leur restitution, leur modification ou leur gestion, c'est-à-dire pour le modèle relationnel, les opérateurs algébriques (select, project, join, etc.) qui transforment des relations en d'autres relations.
- la *partie intégrité* correspond à la collection de règles d'intégrité définissant implicitement ou explicitement l'ensemble des états consistants de la base, c'est-à-dire pour le modèle relationnel, les contraintes d'intégrité d'entités et d'intégrité référentielle.

Cette définition a été par la suite appliquée à tous les autres types de modèle de données. L'introduction du modèle relationnel a rendu nécessaire l'ajout de la phase logique dans le cycle de vie des $\mathcal{BD}$. Cette première évolution verticale du cycle de vie est illustrée en figure 2.6. Certes, *Edgar Codd* était le pionnier qui a fait évoluer le cycle de vie, mais d'autres chercheurs l'ont suivi comme Professeur Peter Chen, comme nous allons le voir dans la section suivante.

2.1.2 Peter Chen et le cycle de vie

Le modèle relationnel permet une séparation des modèles physiques et logiques. Cependant, il oblige les concepteurs à avoir une connaissance des modèles théoriques (en particulier le processus de normalisation et les dépendances fonctionnelles) afin de pouvoir concevoir leurs applications. Il est ainsi peu adapté au monde de l'entreprise où les utilisateurs désirent avoir des modèles de données pouvant exprimer toute la sémantique représentée par les règles de gestion de leur compagnie mais aussi une méthodologie unifiée pour la conception de leurs $\mathcal{BD}$ qui soit supportée par des outils [53]. L'utilisation croissante des $\mathcal{BD}$ dans les entreprises a donc obligé les concepteurs à chercher plus de conceptualisation et de sémantique. C'est dans ce contexte que le modèle *entités/associations* (E/A) a été introduit en 1976 par *Peter Chen* [54]. Par rapport au modèle relationnel, ce nouveau modèle a l'avantage d'être plus proche du réel et donc plus compréhensible par les concepteurs de $\mathcal{BD}$ ainsi qu'illustré en figure 2.5 : c'est un modèle simple, car utilisant peu de concepts (entités, propriétés, associations, cardinalités), et visuel permettant de concevoir des applications.

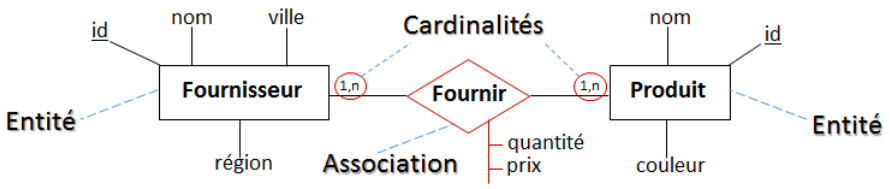


FIGURE 2.5 – Un exemple de diagramme entités/associations

Tout comme le modèle relationnel, ce modèle a un fondement mathématique et s'appuie plus précisément sur la théorie des ensembles et les relations mathématiques. Cependant, une différence majeure existe entre ces deux modèles. Dans le modèle relationnel, la structure de *valeurs de données* (table) est définie comme un produit cartésien des domaines de ses attributs. Au contraire, le modèle E/A utilise ce même constructeur de relation mathématique (à savoir le produit cartésien) pour décrire la structure *des entités* : une relation est le produit cartésien des entités [53]. En ne considérant pas seulement les valeurs des données, mais en traitant directement les entités du monde réel, le modèle E/A introduit un nouveau niveau d'abstraction.De plus, la sémantique est enrichie par rapport au modèle relationnel avec par exemple la représentation des cardinalités, mais aussi le lien entre les entités qui est explicitement représenté dans le modèle E/A (alors qu'il est implicite dans le modèle relationnel) [53].

Si, à l'époque, le modèle E/A a révolutionné le monde de $\mathcal{BD}$, il n'a pas été implémenté immédiatement dans un SGBD. Cette situation trouve probablement son origine dans le succès du modèle relationnel mais aussi dans l'absence d'un langage de requêtes [103]. C'est pourquoi des tentatives ont alors été faites pour proposer un langage de requêtes sur le modèle conceptuel, comme le langage ConQuer publié dans le cadre de la conférence ER'1996 [31]. Du côté de la communauté de conception des $\mathcal{BD}$, le modèle E/A a rapidement rencontré un vif succès, succès qui s'explique en grande partie par la présence du niveau conceptuel, ce dernier étant un niveau d'abstraction plus élevé que les niveaux physique et logique. L'introduction par *Peter Chen* d'une méthodologie de construction d'un schéma initial E/A a permis de populariser ce modèle dans les outils de conception de $\mathcal{BD}$, comme par exemple *Silverrun*, *ERwin* et *ER/Studio*. Dans ces outils, la traduction du schéma E/A en une collection de tables en troisième forme normale a pu être automatisée. Cette popularité s'est aussi traduite au travers de l'organisation de plusieurs autour du thème de la modélisation conceptuelle (comme la *conférence ER*, dont la première édition s'est tenue en 1979). Cette dynamique a été renforcée par l'apparition d'autres modèles dits "sémantiques" se voulant plus expressifs et comportant davantage de sémantique par rapport au modèle relationnel (comme les modèles proposés par *Smith et al.* [171] et *Hammer et McLeod* [100]). De leur côté, les modèles objets utilisent les concepts du modèle E/A afin d'apporter des aspects dynamiques aux modèles statiques existants [53].

Les travaux menés par Peter Chen ont conduit à l'ajout d'une phase conceptuelle dans le cycle de vie des $\mathcal{BD}$. Cettedeuxième évolution verticale du cycle de vie, illustrée en figure 2.6, a été unifiée dans l'architecture ANSI/SPARC [183] qui est désormais universellement reconnue

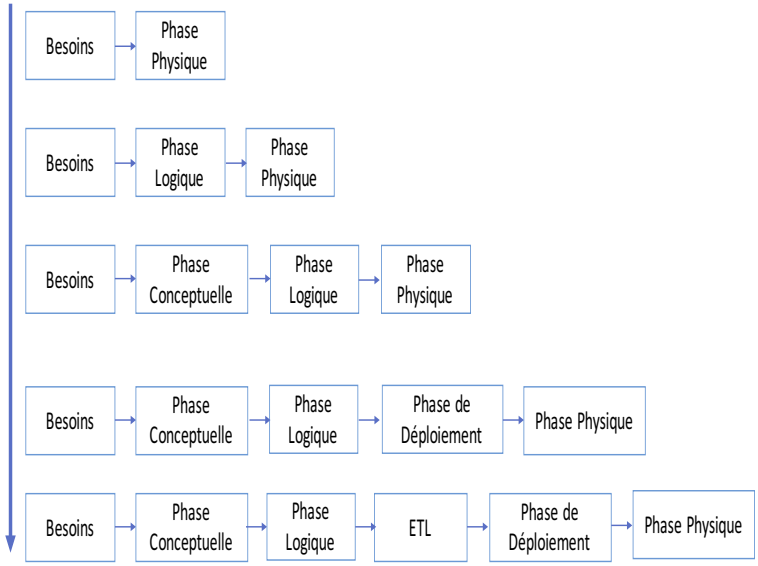


FIGURE 2.6 – Evolution verticale du cycle de conception des $\mathcal{BD}$

dans la communauté des $\mathcal{BD}$. Cette architecture distingue trois niveaux d'abstraction (ainsi qu'illustré en figure 2.7) :

- le niveau *conceptuel* pour lequel la connaissance du domaine est formalisée en utilisant un formalisme de modélisation conceptuel (E/A, modèle sémantique ou autre).
- au-dessus, les modèles *externes* dans lesquels les vues "utilisateurs" permettent d'adapter les données fournies aux besoins des différentes catégories d'utilisateurs.
- et enfin en-dessous, le modèle *interne* qui présente une spécification des données telle qu'elle sera implémentée dans le système de $\mathcal{BD}$.

L'architecture ANSI/SPARC permet d'assurer l'indépendance logique (la modification du schéma conceptuel de la $\mathcal{BD}$ n'impacte pas ses schémas externes ou ses programmes d'applications) et physique (modifier le schéma interne de la $\mathcal{BD}$ au travers par exemple de la création de nouvelles structures d'accès ou la réorganisation des fichiers de la $\mathcal{BD}$ ne modifie ni le schéma conceptuel ni les schémas externes) des données. Le modèle peut donc être modifié à un niveau de la $\mathcal{BD}$ sans que cela ait un impact sur le modèle au niveau supérieur [77].

2.1.3 Omniprésence des besoins fonctionnels et non fonctionnels

Les besoins des utilisateurs, qu'ils soient fonctionnels ou non-fonctionnels, jouent un rôle très important car ils sont en entrée de chacune des différentes phases de la conception d'un produit informatique. Leur définition relève du domaine de l'*ingénierie des besoins* (IB). Celle-ci est en effet "concernée par l'identification de buts assignés au système envisagé et l'opérationnalisation de ces buts en contraintes et exigences imposées au système et aux agents qui en

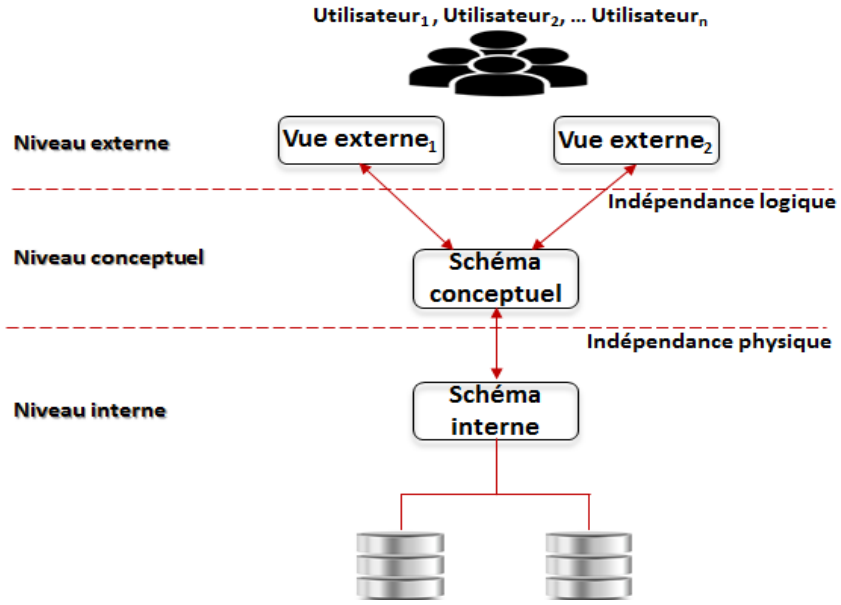


FIGURE 2.7 – L'architecture ANSI/SPARC

assureront le fonctionnement. L'IB peut être vue comme le processus qui permet de transformer une idée floue en spécification précise des besoins servant de support à la spécification du système et de ses interfaces avec l'environnement" [159]. L'ensemble des besoins et exigences des utilisateurs relatifs aux *données* et aux *traitements* doivent donc être définis de manière détaillée [77], de même que les exigences (fonctionnelles et non-fonctionnelles) précisant l'ensemble des opérations et traitements supportés par l'application de $\mathcal{BD}$. Afin de pouvoir réaliser cette phase de définition, il est nécessaire au préalable d'avoir identifié la portée de l'application de $\mathcal{BD}$ ainsi que de l'ensemble de ses utilisateurs. De même, il faut avoir réalisé l'étude de l'environnement du système incluant l'analyse des types de traitements et de leur fréquence, l'analyse des flux d'informations du système, des entrées et sorties des traitements, des caractéristiques géographiques des utilisateurs ainsi que l'étude et l'analyse de la documentation existante relative à l'application de la $\mathcal{BD}$ et l'établissement des questionnaires destinés aux utilisateurs.

2.1.4 Nouvelles applications et le cycle de vie

Aux applications transactionnelles de type (OLTP) ont succédé à la fin des années 80 et début des années 90 les applications d'entreposage des données, afin de répondre aux nouveaux besoins décisionnels émis par les entreprises et les organisations. Cela a eu un impact sur l'évolution verticale du cycle de vie (ainsi qu'illustré en figure 2.6), et en particulier sur le processus d'Extraction, de Transformation et de Chargement (ETL). ETL représente une phase importante du cycle de vie de conception d'un entrepôt de données [89], car il est en charge de l'intégration des données provenant de différentes sources hétérogènes, de donc de leur in-

tégrité et de leur exactitude (ce qui impacte directement les décisions prises par les décideurs des entreprises). Cela a conduit plusieurs compagnies à proposer des outils payants permettant de le mettre en œuvre (comme IBM Information Server, InfoSphere DataStage, SAS Data Integration Studio, Oracle Warehouse Builder (OWB), Sap BusinessObjects Data Integration et des produits Open Source comme Talend Open Studio, Pentaho Data Integration (ex Kettle) et Clover ETL). Dans le processus ETL, les données sont d'abord extraites à partir des sources de données hétérogènes ($\mathcal{BD}$, fichiers plats, ERP, Web, etc.). Elles sont ensuite stockées dans une zone temporaire, nommée Data Staging Area (DSA). Elles y sont transformées, nettoyées et homogénéisées. Enfin, ces données sont chargées dans l'entrepôt de données cible [29]. Il faut cependant noter que ce processus est coûteux en termes de temps et d'argent : il consomme plus de 70% des ressources [89].

2.1.5 Émergence des SGBD et des plateformes

Les recherches menées sur les aspects matériels ont conduit au développement de nouveaux SGBD et de nouvelles architectures de déploiement. Les $\mathcal{BD}$ réparties géographiquement ainsi que les possibilités d'accès en parallèle aux données ont ainsi été étudiées pendant les années 90, et ont mené à la définition de plusieurs prototypes. Le projet *Gamma*, par exemple, a été initié afin d'explorer la capacité des disques à traiter les données en parallèle et a abouti au développement d'une machine de $\mathcal{BD}$ parallèle également appelée *Gamma*. D'autres systèmes parallèles, comme ceux d'*IBM*, de *Tandem*, d'*Oracle*, d'*Informix*, de*Sybase*, ou encore d'*AT&T* se sont également appuyés sur les recherches conduites dans le cadre de ce projet [90]. C'est ainsi qu'aujourd'hui, la majorité des systèmes de $\mathcal{BD}$ (machines centralisées, machines parallèles, Cloud, etc.) offre la possibilité de distribuer et de répliquer les données entre les nœuds d'un réseau informatique [90].

Cependant, si des solutions commerciales s'appuyant sur des machines parallèles (comme celles proposées par Teradata et Netezza) ont été adoptées par des grandes entreprises françaises telles que Carrefour et Banque Populaire, elles demeurent toutefois coûteuses (en termes de droits de licence et coûts d'installation et de maintenance) pour des PME (Petites et Moyennes Entreprises). Ce constat a été ainsi à l'origine de l'émergence de solutions de type Cluster de $\mathcal{BD}$. Les performances d'un SGBD dépendant fortement de la plateforme sur lequel il s'exécute (ainsi qu'illustré en figure 2.8, le choix de celle-ci s'avère donc primordial.

Pour répondre à l'explosion du nombre de SGBD et de plateformes de déploiement, le cycle de vie de conception de $\mathcal{BD}$ a donc dû être enrichi par la phase de déploiement (illustré en figure 2.6). Cette phase était incluse dans la phase physique pour la première génération de $\mathcal{BD}$, le choix proposé aux concepteurs étant alors beaucoup plus réduit. En se basant sur cette discussion et ainsi qu'illustré en figure 2.2, nous proposons un cycle de vie générique de conception d'une $\mathcal{BD}$ comportant les six phases principales suivantes :

1. la collection des besoins fonctionnels et non fonctionnels,

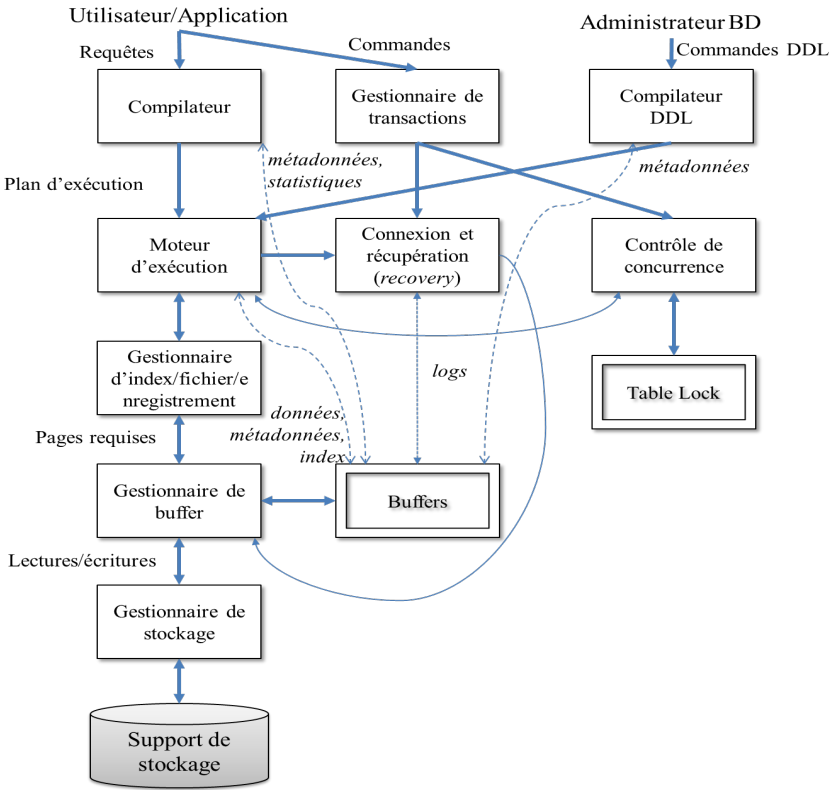


FIGURE 2.8 – Différentes composantes d'un SGBD

2. la modélisation conceptuelle,

3. la modélisation logique,

4. la phase ETL,

5. la modélisation de déploiement,

6. la modélisation physique.

2.2 L'évolution interne du cycle de conception

Un cycle de conception des $\mathcal{BD}$ avancées a été défini s'appuyant sur les niveaux de l'architecture ANSI/SPARC, comprenant six principales étapes [139] : la définition des besoins, la phase de modélisation conceptuelle, la phase de modélisation logique, ETL (cas des entrepôts de données), la phase de déploiement et la phase physique. Ces différentes étapes correspondent aux divers niveaux de l'architecture ANSI/SPARC de la façon suivante : le niveau externe, le niveau conceptuel et ETL expriment les phases de modélisation conceptuelle et logique et le niveau interne correspond aux phases de déploiement et physique [139, 77]. Une explication

détaillée des phases de ce cycle de conception par la spécification des étapes de chaque phase a été donné dans la figure 2.9.

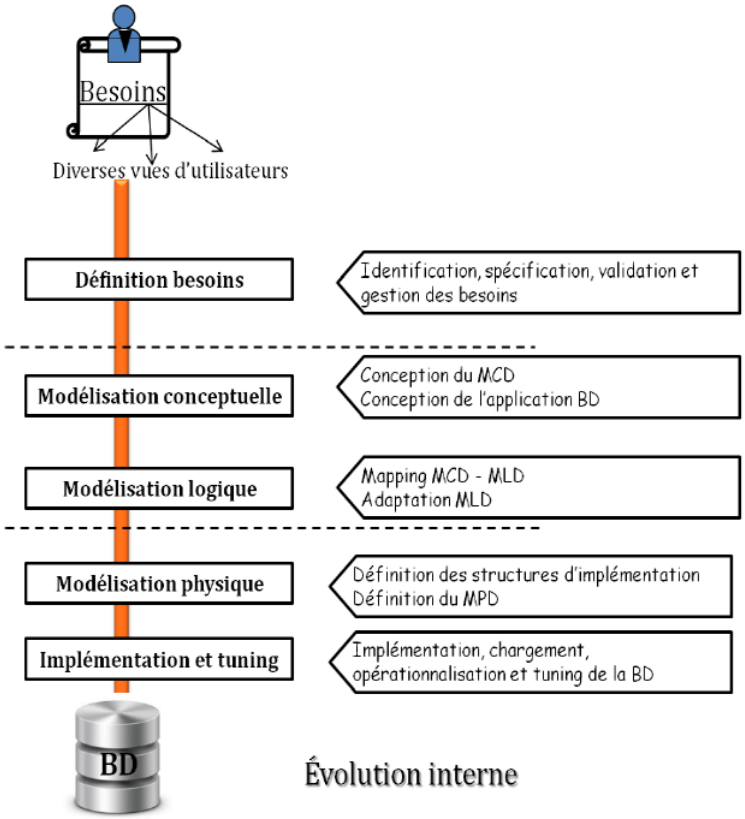


FIGURE 2.9 – Évolution interne du cycle de conception des BD

2.2.1 Définition des besoins

La phase de définition des besoins comprend les quatre étapes suivantes [172] :

- *La collecte des besoins* : les besoins sont collectés auprès des utilisateurs finaux ou des clients du système. Plusieurs techniques de collecte des besoins peuvent être utilisées comme les interviews, les réunions ou les workshops.

- *La spécification des besoins* : consiste à formaliser les besoins et à identifier leurs caractéristiques attendues. L'étape de collecte des besoins fournit généralement un premier ensemble de besoins informels, inconsistants ou incomplets. Trois principales techniques de spécification des besoins sont utilisées pour spécifier les besoins collectés d'une manière plus structurée [125] :

  1. *Les techniques informelles* : sont construites en langue naturelle avec ou sans règles de structuration. Leur usage introduit des risques d'ambiguïtés car ni leur syntaxe ni

leur sémantique ne sont parfaitement définies. Parmi ces techniques, nous citons : le *questionnaire* et le *cahier des charges*.

2. *Les techniques semi-formelles* : sont généralement basées sur des notations graphiques qui ont une syntaxe précise et permettent d'avoir une vision claire du système. Ces modèles sont de bons vecteurs de communication entre les concepteurs et les utilisateurs du système. Parmi ces techniques, nous citons les modèles de la méthode *Merise* et les modèles *UML* (comme les cas d'utilisations ou les diagrammes de séquence pour spécifier les besoins relatifs aux traitements).
3. *Les techniques formelles* : sont basées sur des notations mathématiques qui fournissent un cadre précis et non ambigu pour la modélisation des besoins. Nous pouvons citer la spécification des *méthodes B* et *EB3* (Entity-Based Black-Box).

- *La validation des besoins* : Durant cette phase, les experts du domaine et les utilisateurs valident le modèle obtenu lors de l'étape précédente, afin d'éviter la propagation de toutes inconsistances ou erreurs des besoins durant les étapes de conception et d'implémentation de la base, dont le coût de correction s'avère très important par la suite. Plusieurs outils et langages peuvent être utilisés pour faciliter les tâches de validation des besoins. Des langages formels comme *OCL* ou *Z* sont employés pour compléter la spécification des besoins afin de vérifier leur consistance [77].

- *La gestion des besoins* : Cette phase permet d'établir et de maintenir l'intégrité des besoins pendant que l'application de $\mathcal{BD}$ évolue; en appliquant les activités adéquates et les outils de vérification de la traçabilité des besoins dans le cas où ces derniers évoluent très fréquemment.

2.2.2 La modélisation conceptuelle

Cette phase prend comme entrée les spécifications de besoins collectés auprès des utilisateurs du système. Ses différentes tâches sont assurées par les analystes et les concepteurs de l'application A^{BD}. Elle comprend deux principales étapes menées en parallèle : la conception du modèle conceptuel et la conception de l'application et des transactions [77].

- *La conception du modèle conceptuel* : qui consiste à élaborer le Modèle Conceptuel des Données (MCD) décrivant les exigences des utilisateurs relatives aux *données*. Ce modèle comprend une description détaillée de la structure de la $\mathcal{BD}$, exprimée en utilisant les concepts fournis par un modèle de données d'un haut niveau d'abstraction décrit indépendamment de toute contrainte d'implémentation. Plusieurs formalismes permettent de définir ces modèles existent. Nous pouvons citer le modèle *E/A* [54], le *digramme de classes* d'UML et le modèle *Express*. L'établissement du modèle conceptuel repose généralement sur un dictionnaire regroupant les détails de l'ensemble des propriétés manipulées par le A^{BD}.

- *La conception de l'application de $\mathcal{BD}$* : les opérations et les transactions définies à partir des exigences des utilisateurs relatives aux traitements, sont utilisées pour spécifier les requêtes des utilisateurs de haut niveau. Cette étape permet de vérifier si le schéma conceptuel défini répond à toutes les exigences identifiées [77]. Les opérations peuvent être de trois différents types [77] : les opérations de restitution des données, les opérations de mise à jour et les opérations combinant la restitution et les mises à jour.
  Ces opérations doivent être spécifiées au niveau conceptuel, en identifiant en premier lieu les entrées/sorties de chacune des opérations ainsi que leur comportement fonctionnel [77]. Cette étape de conception peut inclure une tâche d'identification des processus qui consistent en un ensemble d'opérations complexes. Des outils et des notations sont utilisés pour spécifier les processus comme *BPwin*, les outils de modélisation de *workflow*, certains diagrammes *UML* (comme le diagramme d'activité et le diagramme de séquence), ou certains modèles de la méthode *Merise* [158] [7] qui permettent de représenter les traitements de l'application au niveau conceptuel comme le modèle conceptuel de traitements.

2.2.3 La modélisation logique

Cette phase prend en entrée le Modèle Conceptuel de Données (MCD), et fournit en sortie un Modèle Logique des Données (MLD). Ce dernier représente l'organisation des données dans un modèle de données prêt pour l'implémentation. Cette organisation est nommée aussi *déploiement logique* des données. Le modèle logique de données possède des constructeurs de modélisation faciles à comprendre par les utilisateurs; ces constructeurs ignorent les détails physiques d'implémentation. La traduction du modèle conceptuel en un modèle logique se fait de manière directe voire automatique, en suivant un ensemble de règles de traduction prédéfinies. La phase de modélisation logique se fait en deux principales étapes :

- *Mise en correspondance du modèle :* la première étape consiste à traduire le modèle conceptuel vers un modèle logique indépendamment des caractéristiques du SGBD.

- *Adaptation du modèle :* une deuxième étape consiste à adapter le schéma logique obtenu aux spécificités (constructeurs de modélisation et contraintes) du ou des SGBD sur lesquels sera implémenté le modèle logique. Le modèle logique est décrit à la fin de cette phase dans un *langage de définition des données* (LDD), il peut être complété lors de la phase de modélisation physique. Ils existent plusieurs outils de conception permettent de générer automatiquement le modèle logique décrit selon un LDD à partir d'un modèle conceptuel de données comme *ERwin*, *BPwin*, *Rational Rose* et *Visio*.

7. Merise est une méthode d'analyse, de conception et de gestion de projet informatique lancée par le gouvernement français dans les 1970 et 1980 pour l'informatisation massive des organisations.

2.2.4 La phase de déploiement

Le choix et la plateforme du SGBD sont identifiés durant cette phase. Dans la première génération de conception de $\mathcal{BD}$, cette phase n'était pas bien identifié. Certains chercheurs considéraient que le choix du SGBD doit précéder la phase logique [77]. Ce dernier peut dépendre de plusieurs facteurs stratégiques, techniques et économiques, il peut représenter une contrainte devant être respectée dès le début de la conception. L'application de $\mathcal{BD}$ peut aussi être conçue de manière générique dans l'objectif de la déployer sur plusieurs plateformes. Dans ce cas, le choix du SGBD et sa plateforme peut être effectué lors de la phase de déploiement, où le *déployeur* choisit la plateforme qui répond le mieux aux besoins de l'application. Ainsi l'arrivée des machines parallèles et distribués a rendu la phase de déploiement de plus en plus complexe.
Le cycle de cette phase implique quatre principales étapes [28] : **(1)** le choix de l'architecture matérielle, **(2)** la fragmentation, **(3)** l'allocation, **(4)** la réplication et **(5)** l'équilibrage de charges. Selon le périmètre de notre champ d'étude (sélection SGBD et plateforme), nous proposons de détailler la première étape.

2.2.4.1 Choix de l'architecture matérielle Il est entendu qu'une plateforme de $\mathcal{BD}$ est constituée d'un ou de plusieurs serveurs, d'un système d'exploitation, d'un SGBD et d'un mécanisme de stockage de données. Le choix d'une architecture matérielle destinée à supporter une $\mathcal{BD}$ volumineuse dépend essentiellement des exigences du meilleur rapport *prix/performances*, la capacité d'*extensibilité* et la *disponibilité des données* [101].
Actuellement, plusieurs modèles d'architectures parallèles sont disponibles tels que les Multiprocesseurs Symétriques (*Symmetric Multi Processor-SMP*), les Clusters, les Machines Massivement Parallèles(MMP). Ces différentes architectures sont classifiées selon le partage de la mémoire (*shared-memory*), partage des disques (*shared disks*), sans aucun partage (*shared nothing*) et partage partiel des ressources (*shared-something*).
Nous décrivons d'abord les trois architectures conventionnelles et leurs structures hybrides qui tentent de combiner les avantages de chaque architecture, en suite les architectures alternatives telles qu'elles sont présentées dans la littérature [143, 178]. Il y a trois architectures conventionnelles et des architectures hybrides qui tentent de combiner les avantages de chacune d'elles [73, 178]. Les trois architectures conventionnelles, avec leur architecture hybride sont illustrées dans la figure 2.10.

- **Les architectures à mémoire partagée** (*shared-memory*). Les processeurs et les disques ont accès à une mémoire commune, typiquement par un bus ou un réseau d'interconnexion. L'intérêt de l'utilisation de cette architecture est l'efficience de la communication entre les processeurs, où les données sont accessibles par n'importe quels processeurs. Chacun des processeurs peut envoyer rapidement un message aux autres en écrivant en mémoire au lieu d'utiliser les canaux de communication spécifiques. L'inconvénient majeur de cette architecture est sa scalabilité limitée (32 à 64 nœuds), qui est dû au fait

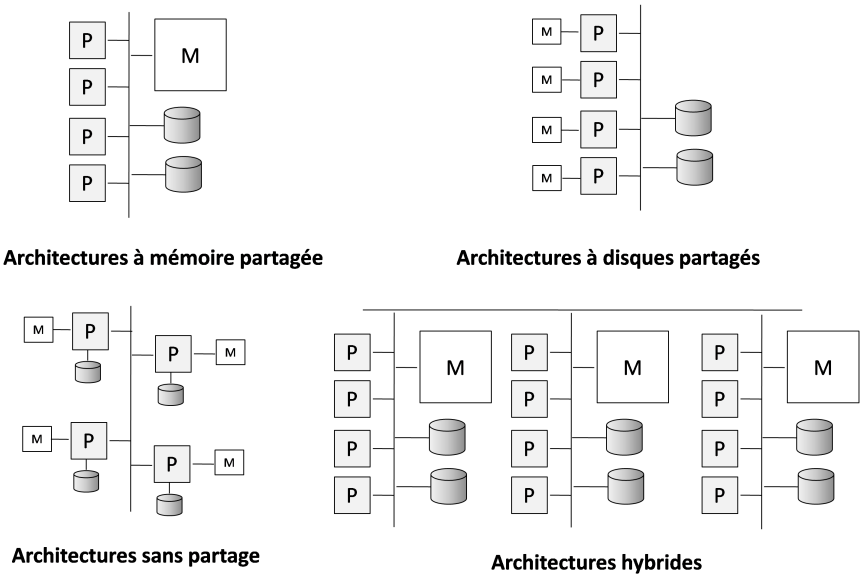


FIGURE 2.10 – *Architetcures matérielles usuelles*

que le bus (le réseau d'interconnexion) devient un goulot d'étranglement. L'ajout de nouveaux processeurs implique l'accroissement du temps d'attente nécessaire pour accéder à la mémoire principale partagée. Généralement, l'architecture à mémoire partagée dispose d'une large mémoire cache au niveau de chaque processeur de sorte, ce qui évite autant que possible le référencement de la mémoire partagée. En outre, les caches doivent être cohérents, si un processeur effectue une écriture dans un emplacement mémoire, les données de cet emplacement doivent être soit mises à jour ou supprimées de tous les autres processeur où les données sont mises en cache.

- **Les architectures à disques partagés** (*shared disks*). Chaque processeur détient sa propre mémoire centrale et le disque est partagé entre tous les processeurs. Cette architecture est similaire à l'architecture à mémoire dans le sens qu'une seule mémoire secondaire est partagée. Elle souffre de congestion dans le réseau d'interconnexion quand plusieurs processeurs essayent d'accéder au disque en même temps. En effet, le traitement de l'ensemble des sous-requêtes nécessite la récupération des données du disque partagé pour les stocker dans la mémoire locale du processeur.
  Ainsi, la hiérarchie de la mémoire qui est partagée représente la principale différence entre l'architecture à mémoire partagée et celle à disque partagé. Dans le contexte des nouvelles architectures logicielles, les architectures à disques partagés et à mémoire partagée sont considérés comme des machines multiprocesseurs symétriques. Une **machine SMP typique** est constituée de 2 à 16 CPU. Ce nombre important de processeurs n'est pas tolérable à cause des problèmes de passage à l'échelle. Chaque processeur maintient

son propre cache et une mémoire principale partagée entre tous les processeurs. La taille de la mémoire principale et du cache diffère d'une machine à l'autre. Plusieurs disques peuvent être attachés à une machine SMP et tous les CPU auront un accès similaire. Le système d'exploitation alloue normalement les tâches selon un ordonnanceur. Lorsqu'un processeur est inactif, une tâche dans sa file d'attente est immédiatement attribuée. De cette façon, l'équilibrage est relativement facile à réaliser.

- **Les architectures sans partage** (*shared-nothing*). Ce type d'architecture attribut à chaque nœud sa propre mémoire et son propre disque. Les processeurs communiquent entre eux via un réseau de communication à haut débit. Par ailleurs, les réseaux d'interconnexion des systèmes sans partage sont généralement conçus pour être évolutifs, dont la capacité de transmission augmente avec le nombre des nœuds ajoutés. En conséquence, l'architecture sans partage est plus scalable et peut aisément supporter un large nombre de processus. *Teradata, Grace* et le prototype de recherche Gamma sont de type *shared nothing*. Le problème de l'accès concurrent aux données partagées n'est pas posé pour cette architecture, mais l'équilibrage de charge est difficile à atteindre même pour les requêtes simples, car les données sont stockées localement au niveau de chaque disque. Aussi, le problème de la mauvaise distribution est l'un des challenges dans le traitement parallèle des requêtes sur une machine de ce type. Dans le contexte des nouvelles architectures logicielles, la machine sans partage est catégorisée comme machine massivement parallèle (*Massively Parallel Processing-MPP*). Cette architecture est caractérisée par son réseau d'interconnexion à haut débit.

- **Les architectures hybrides**. L'architecture hybride représente une combinaison des architectures "sans partage" et "à mémoires partagées", en combinant les avantages de chacune et compense leurs inconvénients. Elle équilibre la charge des architectures à mémoires partagées tout en permettant l'extensibilité des architectures sans partage. Cette architecture est nommée *Shared Something*, Elle augmente la flexibilité de la configuration (nombre et taille des nœuds) et diminue le coût de communication réseau en raison que le nombre de nœuds est réduit. Le parallélisme intra-requête peut être isolé à un seul multiprocesseur shared-memory, car il est beaucoup plus facile de paralléliser une requête dans une architecture à mémoire partagée que dans une autre sans partage. En outre, le degré de parallélisme sur un seul nœud à mémoire partagée peut être suffisant pour la plupart des applications. Autrement dit, le parallélisme intra-requête est réalisé par l'exécution parallèle sur les nœuds. Ainsi plusieurs variantes de cette architecture existent, mais fondamentalement chaque nœud représente une machine à mémoire partagée connectée au réseau d'interconnexion de l'architecture sans partage.

La baisse du coût des ordinateurs et l'avènement des réseaux à haut débit ont donné naissance à de nouvelles architectures matérielles distribuées pouvant être utilisées comme des plateformes logicielles pour le déploiement des systèmes parallèles.

- **Les architectures grappes de machines** (cluster). Un cluster est un ensemble de nœuds interconnectés pour partager des ressources pour un seul et même système. Ces ressources peuvent être matérielles comme un disque ou logiciel comme un système de gestion de données. Les nœuds d'un cluster sont des machines simples comme un micro-ordinateurs (PC) ou plus puissantes comme les SMP. Notant que l'utilisation des composantes *off-the-shelf* est essentielle pour obtenir le meilleur rapport prix/performance, tout en exploitant le progrès continu des composantes matérielles. Dans sa forme la moins chère, l'interconnexion entre les nœuds peut être réalisée par un simple réseau. Cependant, il existe actuellement un standard pour l'interconnexion des nœuds d'un cluster nommé *Myrinet and Infiniband* qui permettent le déploiement de réseaux haut débit tels que *Gigabit Ethernet* (GbE) avec une faible latence pour le transfert des messages. Contrairement aux systèmes distribués où le cluster est cerné dans une zone géographique réduite, et il est généralement homogène. L'architecture d'un cluster peut être soit sans partage soit à disque partagé.

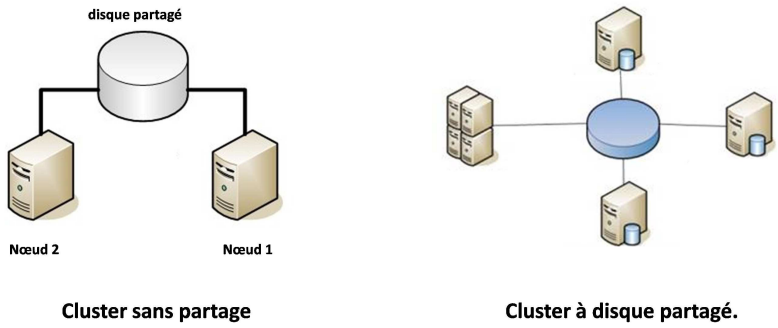


FIGURE 2.11 – *Exemple d'un cluster d'ordinateurs*

Les clusters sans partage ont été largement utilisés car ils fournissent le meilleur rapport *qualité/prix* et peuvent regrouper des milliers de nœuds (extensibles). *Les clusters à disques partagés* existent sous deux formats.

  - Un *NAS* (*Network Attached Storage*) est une plateforme qui partage le disque sur un réseau en utilisant un protocole de distribution des fichiers systèmes comme *Network File System*. Mieux adapté aux applications à faible débit telles que la sauvegarde et l'archivage de données à partir de disques durs des *PC*. Cependant, il est relativement lent, donc il ne convient pas pour la gestion des $\mathcal{BD}$ car il devient rapidement un goulot d'étranglement avec de nombreux nœuds,
  - Un *SAN* (*Storage Area Network*) fournit des fonctionnalités similaires au *NAS* mais avec des interfaces à niveau inférieur. Il utilise un protocole basé sur des blocs, ce qui facilite la gestion de la cohérence du cache. SAN fournit un haut débit de données et peut évoluer jusqu'à un grand nombre de nœuds. Son seul inconvénient à l'égard de *shared-nothing* est le coût d'acquisition élevé.

Les architectures clusters possèdent plusieurs avantages. Elles combinent la flexibilité et la performance des architectures à mémoires partagées au niveau de chaque nœud d'une part. D'autre part, elles assurent l'extensibilité et la disponibilité de l'architecture à disques partagés ou sans partage. En outre, l'utilisation des nœuds *off-the-shelf* à mémoire partagée avec une interconnexion standard de clusters fournit une alternative rentable pour approprier une plateforme à haute performance comme *NUMA* (*Non Uniform Memory Architecture*) ou *MPP* (*Massive Parallel Processing*).

- **Grille de calcul**. Elle résulte de la combinaison de ressources à partir de multiples domaines administratifs appliqués à une tâche commune, généralement à un problème scientifique, qui nécessite le traitement d'un grand volume de données. Cette architecture permet le partage, la sélection et l'agrégation de ressources autonomes réparties géographiquement de manière dynamique durant l'exécution des requêtes en fonction de leur disponibilité, capacité de stockage, puissance de calcul, coût et qualité de service [97]. Ce sont des clusters à grande échelle, dont un exemple est illustré par la figure 2.12.

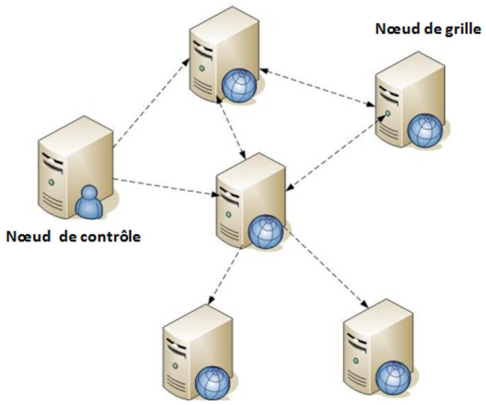


FIGURE 2.12 – *Exemple de l'infrastructure grille de calcul*

- **Le Cloud.** Un nuage (*Cloud*) est particulièrement avantageux pour les petites et moyennes entreprises qui souhaitent externaliser complètement leurs infrastructures de centres de données, ou les grandes entreprises qui souhaitent obtenir une haute capacité de calcul et de stockage sans engager de coût élevé de construction de centres de calcul importants en interne. Le *Cloud* est une extension de ce paradigme où les capacités des applications sont exposées comme des services sophistiqués qui peuvent être accessibles sur un réseau facturés selon la consommation.
  Un (*Cloud*) est un système parallèle composé d'un ensemble d'ordinateurs interconnectés, dynamiquement provisionnés et présentés comme une ou plusieurs ressources informatiques unifiées basées sur le service. Fondamentalement, un Cloud peut être : un *Cloud public* qui désigne une structure souple et ouverte dédié à la vente de services dont les informations peuvent être consultées à partir d'Internet.

Un *Cloud privé* est un réseau composé de ressources propriétaire ou un centre de calcul qui fournit des services hébergés à un nombre limité de personnes. Les informations et les applications qui s'exécutent sur le nuage privé peuvent être consultées à partir de l'intranet de l'entreprise en utilisant une connexion VPN entre le centre de calcul interne et l'infrastructure du sous-traitant. La figure 2.13 illustre un exemple de l'infrastructure Cloud de calcul.

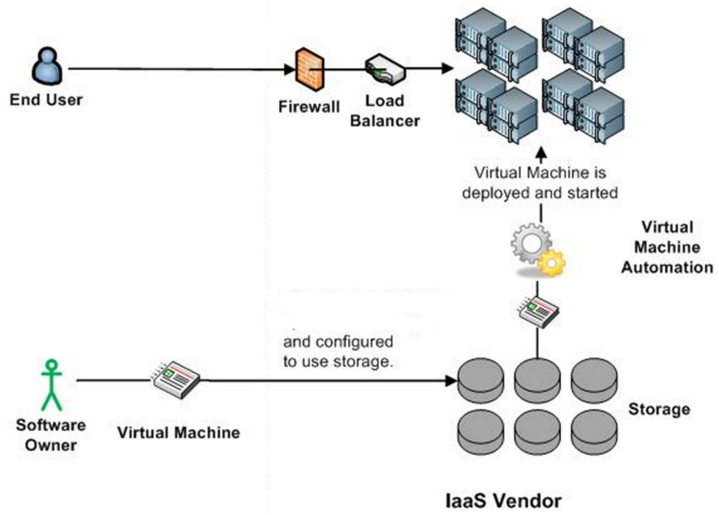


FIGURE 2.13 – *Exemple de l'infrastructure cloud de calcul*

2.2.5 La modélisation physique

Avec la naissance des applications décisionnelles s'est développée l'étape de la conception physique [51]. Celle-ci joue un rôle majeur dans le développement des bases et entrepôts de données. Elle permet en effet de proposer un large choix de solutions pour le stockage de données, que ce soit en termes de regroupement, de partitionnement, d'indexation, etc. [139], afin de satisfaire les besoins non fonctionnels identifiés lors de la phase de collecte des besoins (comme par exemple le temps de réponse aux requêtes, l'espace de stockage utilisé par les fichiers de la base ou le débit moyen des transactions, la consommation énergétique). L'objectif est de créer le meilleur modèle possible de stockage de données qui assure à la fois la performance adéquate et l'intégrité de la $\mathcal{BD}$. La conception physique prend toute son importance avec les entrepôts de données, en raison du volume de données, de la complexité des requêtes et de l'évolution des besoins non fonctionnels (des critères liés à la qualité de service et la consommation énergétique remplaçant des critères de performance [163]). La phase de conception physique se décompose en quatre tâches principales devant être réalisées par l'administrateur[20] :

1. le choix des structures d'optimisation. L'identification des structures pertinentes dépend de l'étude de la charge à optimiser et de la plateforme sur laquelle celle-ci s'exécute. En raison du grand nombre de structures d'optimisation existantes (comme la fragmentation, les vues matérialisées, les index, la compression, factorisation, réplication, etc.), celle-ci requiert un haut niveau d'expertise.

2. le choix du mode de sélection. L'optimisation peut être effectuée en utilisant une ou plusieurs structures d'optimisation. Dans le premier cas, il s'agit d'une sélection isolée. Dans le second cas, il s'agit d'une sélection multiple et plusieurs scenarii peuvent alors être employés pour combiner ces techniques [109]. Le choix de l'ensemble des structures à employer et du mode de combinaison est déterminant quant à la performance du système.

3. le développement des algorithmes de sélection. Pour chaque problème d'optimisation, il existe plusieurs algorithmes de niveau de complexité croissante. Les approches simples sont souvent faciles à mettre en œuvre, mais donnent une faible efficacité (comme c'est le cas pour les approches de la gestion du buffer [109]). De leur côté, les algorithmes lourds ont une meilleur efficacité mais au prix d'un temps d'optimisation élevé (comme c'est le cas pour les approches de la fragmentation horizontale [49]). Les compromis s'avèrent très rares au niveau de la conception physique.

4. la validation et le déploiement des solutions d'optimisation. Afin de pouvoir être déployées sur un environnement réel, les recommandations obtenues par un algorithme de résolution doivent être validées pour évaluer leur performance effective. Pour ce faire, plusieurs outils existent, qu'ils soient commerciaux comme *Oracle SQL Acces Advisor* [64] et *DB2 Design Advisor* [199] ou académiques comme Parinda [128]. Cependant, ces outils présentent des limitations liées au choix des structures d'optimisation et au mode de sélection fourni.

La phase de conception physique peut été formalisée comme le problème mathématique suivant [20] : Etant donnés :

- un schéma d'une $\mathcal{BD}$ déployé sur un SGBD et une plateforme,
- un ensemble de besoins non fonctionnels,
- une charge de requêtes $Q = \{Q_1, Q_2, \ldots, Q_L\}$ où chaque requête Q_l possède une fréquence d'accès f_l,
- l'ensemble des structures d'optimisation $\mathcal{SO} = \{SO_1, SO_2, \ldots, SO_T\}$ supportées par le SGBD,
- et enfin l'ensemble des contraintes liées à $\mathcal{SO}$: $C = \{C_1, C_2, \ldots, C_T\}$ où chaque contrainte C_t est associée à une structure d'optimisation SO_t.

Le problème consiste alors à sélectionner un ou plusieurs structures d'optimisation satisfaisant l'ensemble des besoins non fonctionnels de même que les contraintes définies dans C. Ce problème étant NP-complet [109] en raison de la taille de l'espace de recherche, plusieurs heuristiques ont été proposées. Afin de mesurer la qualité de ces différents algorithmes de résolution, des modèles de coût mathématique estimant les besoins non fonctionnels ont été définis [109]. Ceci considèrent les paramètres pertinents liés au contexte de la $\mathcal{BD}$, son SGBD ainsi que sa plateforme, la politique d'exécution de requêtes (centralisée/parallèle) [28], etc. Ils sont le plus souvent développés en fonction des composantes principales d'un SGBD [28] ,ce qui amène à considérer :

1. le coût des entrées-sorties (IO), nécessaire aux lectures et écritures entre la mémoire et le support de stockage (comme les disques),

2. le coût CPU,

3. et le coût de communication sur le réseau COM (si les données sont réparties sur le réseau). Ce dernier est généralement exprimé en fonction de la quantité totale des données transmises et dépend de la nature de la plateforme de déploiement.

L'évolution de la technologie a fortement impacté la phase physique, en faisant du SGBD et de l'architecture matérielle de la plateforme des entrées du problème de la conception physique.

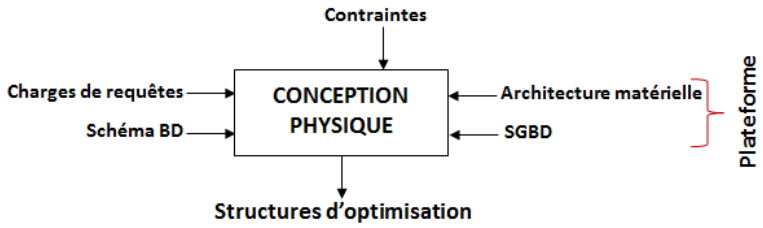


FIGURE 2.14 – *Phase physique dans le cycle de vie*

2.3 Évolution horizontale du cycle de conception

Le cycle de conception des $\mathcal{BD}$ établi a été marqué par un autre type d'évolution, que nous appelons évolution "horizontale". Cette évolution est représentée par la diversification des modèles conceptuels, logiques et physiques qui sont apparus pour répondre à des types d'applications spécifiques (voir la figure 2.15). Nous retraçons dans ce qui suit les modèles les plus marquants.

2.3.1 Les SGBD orientées objet

Le cycle de conception des $\mathcal{BD}$ a connu une évolution majeure suite l'introduction de plusieurs modèles sémantiques faisant apparaître la phase de modélisation conceptuelle comme une phase à part entière. Cependant, ces modèles sémantiques n'ont pas été implémentés directement au sein de la $\mathcal{BD}$. Afin de remédier à ce problème et permettre d'exporter au niveau logique la sémantique apportée par ces modèles, différentes propositions ont été formulées. Les SGBD orientés objet (OO) ont été ainsi introduits au milieu des années 80 et ont suscité un vif intérêt. Ces SGBD visaient en particulier à apporter une solution au problème de dysfonctionnement (en anglais "impedance mismatch") entre les langages de programmation orientés objet (comme par exemple le langage C++) et les $\mathcal{BD}$ relationnelles [103]. Les $\mathcal{BD}$ relationnelles utilisent en effet l'encapsulation de structures dont les attributs ont des types différents de ceux du langage de programmation. Cela nécessite l'utilisation de conversions de types ainsi que des contrôles de types, rendant la programmation d'autant plus complexe. Les SGBD objet, qui se basent sur un modèle de données supportant les concepts de la programmation orientée objet, ont donc été proposés. Ils permettent de fournir un stockage persistant pour les objets et les structures de données utilisés par les développeurs dans leurs programmes, qui étaient habituellement perdus une fois que le programme se termine [77]. Si tous les langages de programmation permettent de sauvegarder les objets dans des fichiers, cette forme de persistance demeure assez primitive et mène aux problèmes classiques de gestion des données. L'approche $\mathcal{BD}$ *OO* apporte de son côté une solution à la gestion transparente de la persistance des objets. Si les SGBD orientés objet ont suscité un fort engouement, ils n'ont cependant pas réussi à pénétrer de façon significative le marché des $\mathcal{BD}$. Ils ne représentent en effet qu'à peine 5% des cas d'utilisation [77]. Cela peut s'expliquer notamment par l'incompatibilité entre les différentes solutions de SGBD *OO* proposées sur le marché, ainsi que par le manque du support de ces SGBD vis-à-vis des transactions et requêtes (ce qui est au contraire la force des SGBD relationnels) [103]. Cet échec commercial des SGBD orientés objet n'a pas empêché l'évolution des SGBD existants, qui ont vu l'intégration de nouvelles fonctionnalités et de nouvelles structures et types de données aux SGBD relationnels classiques. Ces fonctionnalités ont été créées soit sous la forme de modules optionnels, spécifiquement destinés à un type d'application (applications de traitements d'images, applications spatiales) [77], soit sous pour répondre à un objectif plus général, comme l'incorporation de concepts orientés objet dans les systèmes relationnels. Ce fut notamment le cas dans le cadre du projet *Ingres*, mené, en 1972 à l'université de Berkeley, portant sur les systèmes d'informations géographiques et visant à modéliser les données géographiques dans une $\mathcal{BD}$ relationnelle [90]. Les systèmes relationnels ayant été conçus pour supporter des types de données classiques et non géographiques, l'expression de requêtes comportant des concepts géographiques (emplacement, dimension, etc) sous forme de requêtes SQL devient très compliquée. Les recherches menées dans le cadre du projet *Ingres* pour répondre à cette problématique ont été à l'origine de la naissance des SGBD relationnels objets (comme par exemple *PostgreSQL*). Ces derniers ont permis la définition de types de données, d'opérateurs, de fonctions et de méthodes définis par l'utilisateur [77, 169]. Depuis,

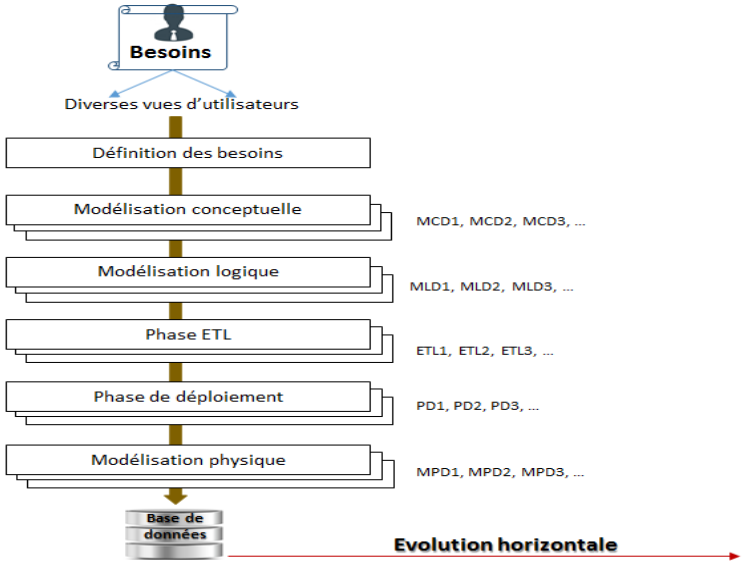


FIGURE 2.15 – Evolution horizontale du cycle de conception des $\mathcal{BD}$

de nombreux nouveaux types de SGBD spécifiques à des d'applications données (comme les SGBD orientés *graphe*, ou plus récemment les SGBD *NoSql*) ont été proposés. Ces SGBD, sur le modèle des SGBD objet, cherchent à offrir des modèles de données plus flexibles et plus adaptés aux besoins spécifiques des concepteurs, des programmeurs et des utilisateurs.

2.3.2 Diversification des modèles conceptuels

Avec l'émergence des applications Web au début des années 2000, il est devenu nécessaire de pouvoir représenter des données semi-structurées. Pour ce faire, de nouveaux modèles conceptuels ont été introduits, en particulier les modèles XML (eXtended Markup Language) qui combinent désormais les concepts des modèles employés dans les systèmes de documents avec ceux utilisés pour la modélisation de $\mathcal{BD}$ [77]. XML est aujourd'hui devenu la principale norme permettant l'échange de données entre les différents types de $\mathcal{BD}$ et les pages Web, en remplacement de plusieurs techniques pré-existantes. Par la suite on été développés les modèles *ontologiques* (une *ontologie* étant "une spécification formelle et explicite d'une conceptualisation partagée d'un domaine de connaissance" [91]) du Web sémantique qui s'appuient pour la plupart sur le modèle XML. *RDF* (bien que son créateur *Tim Berners Lee* la classe plutôt dans la famille des modèles E/A ou plus généralement des modèles conceptuels), *RDFS*, *OIL*, *DAML* et *OWL*) sont quelques exemples de modèles ontologiques. Récemment, ces modèles ontologiques ont été utilisés dans des applications réalisant aussi bien l'annotation de documents, que la gestion de catalogues de composants industriels, l'intégration de données ou encore la construction des $\mathcal{BD}$. Ontologies et modèles conceptuels conceptualisent l'univers du discours au moyen de classes hiérarchisées et de propriétés caractéristiques. Cette similarité a été exploitée pour concevoir des $\mathcal{BD}$ partir des ontologies [176] et définir des processus ETL

sur les ontologies afin d'éviter l'implémentation physique de chaque source de données [111]. Par contre, ontologies et modèles conceptuels diffèrent sur plusieurs points, le premier étant leur objectif de modélisation. Ainsi, pour répondre à un cahier des charges applicatif donné, les modèles conceptuels prescrivent les informations devant être représentées dans un système informatique particulier. De leur côté, les ontologies cherchent à décrire, en se basant sur un consensus, l'ensemble des informations permettant la conceptualisation des domaines d'applications indépendamment de toute application et de tout système dans lequel elles peuvent être utilisées. Une seconde différence porte sur la structure de modélisation et l'identification des concepts. Dans un modèle conceptuel de données, chaque concept a un sens dans le contexte du modèle dans lequel il est défini, tandis que dans une ontologie, chaque concept est identifié individuellement et constitue une unité élémentaire de connaissance [79]. Un modèle conceptuel peut donc être défini à partir d'une ontologie en extrayant uniquement les concepts pertinents répondant à un besoin applicatif donné. Modèles conceptuels et ontologies diffèrent également quant à leur gestion de la consensualité. Si les concepts définis dans un modèle conceptuel ne sont pas réutilisables à l'extérieur de ce modèle, les classes et propriétés d'une ontologie sont associées à des identifiants universels leur permettant d'être référencées de manière unique depuis n'importe quelle structure ce qui permet leur réutilisabilité. Cette caractéristique des ontologies permet de réaliser aisément l'intégration sémantique de tous les systèmes basés sur une même ontologie dès lors que les références à l'ontologie ont été explicitées. Une quatrième différence concerne la non canonicité des informations représentées. Dans un modèle conceptuel, les informations du domaine sont décrites par le biais d'un langage minimal et non redondant. Au contraire, les ontologies s'appuient sur des concepts atomiques ainsi que des concepts définis fournissant des accès alternatifs à la même information. Enfin, une dernière différence porte sur leur façon d'aborder le raisonnement. Au contraire des modèles conceptuels, il existe, dans les ontologies, des mécanismes d'inférence permettant de raisonner sur ces modèles afin d'aboutir à une classification des concepts utilisés, ainsi qu'à la vérification de la cohérence des spécifications. Les ontologies ont connu un vif succès, et leur utilisation intensive a conduit à l'introduction d'un nouveau type de données appelées *données sémantiques* ou *données ontologiques*. Elles correspondent aux données référençant des ontologies. Ces données peuvent être gérées soit en mémoire centrale, soit au sein d'une $\mathcal{BD}$. Si quelques systèmes, comme *OWLIM* [113]ou *Jena* [134] ont adopté la gestion en mémoire centrale pour des raisons de facilité de chargement et de mise à jour des données, cette approche se révèle totalement inadaptée pour la gestion de grands volumes de données ontologiques. La gestion en $\mathcal{BD}$, qui propose des mécanismes efficaces pour le stockage et l'interrogation des données, a donc été adoptée pour permettre le passage à l'échelle et a donné naissance aux *Bases de Données à Base Ontologique* (BDBO) [71]. La diversification des modèles de données au niveau logique et physique cherche à répondre au besoin d'exporter davantage de sémantique au sein de la $\mathcal{BD}$ et d'apporter davantage de flexibilité pour les utilisateurs. Dans le cas des BDBO, toute la sémantique des données de la base est représentée et persistée (stockée) au sein de la $\mathcal{BD}$ sous la forme d'une ontologie locale (voir la figure 2.16). Cette dernière peut soit être utilisé directement comme modèle

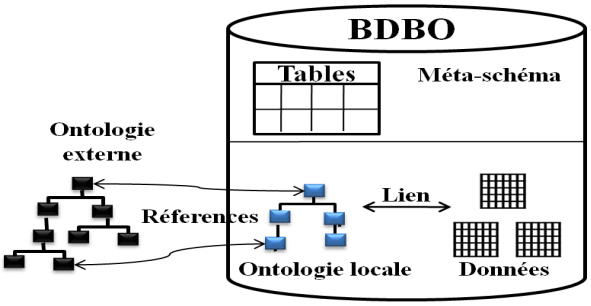


FIGURE 2.16 – Architecture d'une BDBO

conceptuel de la $\mathcal{BD}$, soit servir à extraire ce même modèle conceptuel. Désormais, en plus du modèle logique de données, son modèle conceptuel est également représenté au sein de la base.

L'introduction des BDBO a aussi contribué à l'apparition de nouveaux modèles de données physiques ainsi que de nouvelles architectures de $\mathcal{BD}$. Alors que dans une $\mathcal{BD}$ classique, le modèle logique est stocké selon une approche relationnelle, dans une BDBO différents modèles de stockage physique (représentation horizontale, spécifique, etc.) sont désormais utilisés pour stocker les deux niveaux de modélisation que sont le niveau ontologie et le niveau des instances ontologiques. De même, l'architecture des systèmes gérant les $\mathcal{BD}$ a évolué avec l'introduction de l'architecture mono-couche (où les données sont manipulées au niveau de la mémoire centrale : *main memory databases*), de l'architecture à deux couches (représentant la méta-base et le contenu de la base), l'architecture à trois couches et l'architecture à 4 couches (voir la figure 2.16). C'est l'architecture à deux couches qui est aujourd'hui utilisée pour la majorité des systèmes De même, l'émergence des $\mathcal{BD}$ sémantiques a mené à l'apparition de nouveaux SGBD comme RDFSuite, Jena, Sesame, OntoDB, OntoMS, DLDB, RStar, KAON, 3Store et PARKA, Oracle Semantic ou encore IBM Sor [133]. Ces SGBD sont absents des sites Web consacrés à la classification des BD (comme DB-Engine). Afin de pallier l'absence d'outils permettant de comparer ces systèmes sémantiques, quelques évaluations utilisant des bancs d'essai ont été conduites. En particulier, le banc d'essai OntoBench permet d'évaluer trois SGBD sémantiques, à savoir Oracle Sémantique [67], IBM Sor et OntoDB, en considérant les deux besoins non fonctionnels que sont le coût de chargement des données (ontologie et instances) et la performance en terme de temps de réponse des requêtes [121] (disponible sur la forge du LIAS (https://forge.lias-lab.fr/projects/ontodbench).

2.3.3 Évolution des besoins non fonctionnels

En 2000, Eric Brewer [40] a annoncé la naissance d'un nouveau théorème CAP à l'occasion de sa présentation lors du symposium d'ACM sur les principes de l'informatique distribuée. Avant l'arivée de ce théorème, les utilisateurs et les chercheurs exigeaient simplement le bon fonctionnement de leurs systèmes (ordinateurs, applications, $\mathcal{BD}$, internet, médias). Le théo-

rème de CAP a été révisé plus tard et modifié à travers plusieurs travaux [85]. Le théorème de CAP déclare qu'il existe trois exigences essentielles du système nécessaires à la conception, à la mise en œuvre et au déploiement des applications dans les systèmes informatiques répartis. Ces exigences sont la cohérence, la disponibilité et la tolérance au partitionnement [40].

2.3.3.1 Théorème de CAP Le théorème de CAP, également connu sous le nom de théorème de Brewer, annonce trois exigences. Il s'agit de (1) **la cohérence** (*Consistency*) : tous les nœuds du système voient exactement les mêmes données au même moment, (2) **la disponibilité** (*Availability*) : il faut garantir que toutes les requêtes en lecture ou écriture reçoivent une réponse et (3) **la tolérance au partitionnement** (*Partition-Tolerance*) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement.
A l'issue de ce théorème et les propriétés ACID sont apparues afin de garantir la cohérence .

2.3.3.2 Propriétés ACID (*Atomic, Consistent, Isolated, Durable*) Ce sont des propriétés qui garantissent la fiabilité d'exécution des transactions. Les propriétés ACID sont [4] : (1) **atomicité** : cette propriété assure qu'une transaction se termine complètement ou alors pas du tout, (2) **cohérence** : cette propriété assure que chaque transaction ramène le système à un état valide, (3) **isolation** : cette propriété assure que toute transaction s'exécute comme si elle était la seule sur le système et (4) **durabilité** : cette propriété assure que lorsqu'une transaction est confirmée, elle demeure enregistrée même à la suite d'une panne.
La perte de cohérence par les systèmes informatiques distribués a conduit à l'apparition des propriétés BASE, une alternative d'ACID, afin de maintenir la fiabilité de ces systèmes tels que Google BigTable, Dynamo d'Amazon et Cassandra de Facebook. En effet, des travaux proposent ce type de systèmes pour remplacer le concept ACID [48].

2.3.3.3 Propriétés BASE Les propriétés BASE sont considérées comme l'opposé des propriétés ACID [4]. Elles sont (1) **disponibilité fondamentale** : toutes les données sont distribuées et disponibles selon le théorème de CAP (haute disponibilité). Même en cas d'échec, le système continue à fonctionner et la réponse ne sera pas forcément juste à 100%, (2) **état souple** : pas de garantie de cohérence, l'état du système pourrait changer avec le temps même pendant les périodes de veille. En raison de la cohérence éventuelle, le système peut avoir des changements en cours et son état est toujours souple et (3) **cohérence éventuelle** : le système deviendra constant dans le temps et terminera par un état valide, même lorsque les données ne sont pas cohérentes à un instant t.

2.3.4 Apparition des SGBD Nosql et NewSQL

La faiblesse des SGBDR en termes de performance, d'évolutivité et de flexibilité des besoins de traitement à grande échelle des données complexes a conduit à l'apparition des nouvelle ten-

dance des SGBD à savoir les NoSQL. Ceux-ci offrent en effet de très bonnes performances pour traiter de gros volumes de données faiblement structurées. Ils ne permettent pas de réaliser des transactions atomiques, cohérentes, isolées et durables (ACID). Une $\mathcal{BD}$ NoSQL peut délaisser quelques une de ses propriétés pour des raisons de performances. Ensuite, dans le but de bénéficier à la fois des avantages du relationnel et de NoSQL, c-à-d de pouvoir gérer des données massives tout en gardant la notion relation, une nouvelle famille est apparue, ce sont les NewSQL. Cette famille, basée sur les systèmes relationnels, offre une performance évolutive semblable à celle des systèmes NoSQL, notamment le traitement transactionnel en ligne (OLTP) des charges de requêtes tout en maintenant les garanties ACID d'un système de $\mathcal{BD}$ traditionnelle [75, 92].

3 Bilan

Après cette analyse, nous souhaitons mettre en évidence deux points importants. Le premier point concerne la présence d'un déluge de données de test liées aux phases de déploiement et physique. Aucun travail scientifique ne peut être publié ou breveté sans donner tout l'environnement de tests et ses valeurs numériques. Les organismes comme TPC publient leur résultat de tests des SGBD et leurs plateformes d'une manière régulière, en considérant les données et les requêtes de leurs bancs d'essai.

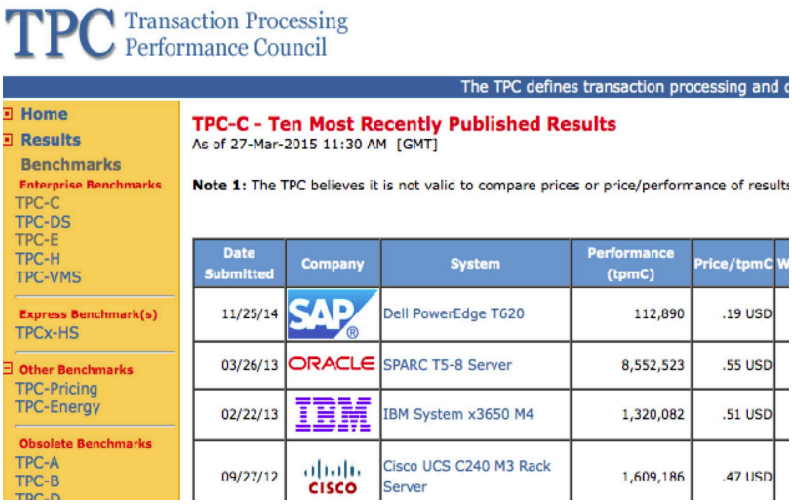


FIGURE 2.17 – *Cliché des résultats de TPC-C*

Si les données et les requêtes du *déployeur* sont similaires à celles de TPC-C (voir la figure 2.17), il peut utiliser les meilleurs SGBD et plateforme proposés par le TPC. Les résultats de ce type de site concernent les SGBD traditionnels et ne donnent pas des résultats concernant certaines spécificités d'un SGBD et plateforme, comme les structures d'optimisation utilisées par ce SGBD, les algorithmes de sélection de ces structures, etc.

Le deuxième point concerne la connaissance préalable du SGBD et la plateforme pour les phases de déploiement et physique. Pour illustrer ce point, considérons par exemple le problème de la sélection des index pour une application de $\mathcal{BD}$. Ce problème est une instance du problème de conception physique. Plusieurs algorithmes ont été proposés pour sélectionner la meilleure configuration d'index pour une $\mathcal{BD}$. Ces algorithmes sont testés en utilisant divers SGBD et plateformes. Pour illustrer ce propos, prenons l'exemple des travaux de Chaudhuri et al. dans son article *Microsoft Index Tuning Wizard for SQL Server 7.0* [52] qui a utilisé SQL Server pour évaluer ses algorithmes. Par contre une variante du même problème publié dans [26] utilise Oracle10g. Cette diversité de SGBD, si elle est bien maîtrisée pourrait être exploitée pour aider les *déployeurs* à choisir leur plateforme.

Pour conclure, nous revendiquons l'exploitation simultanée des résultats de tests des organismes comme TPC et ceux publiés dans les articles scientifiques.

4 Conclusion

Dans ce chapitre, nous avons présenté d'une manière synthétique l'évolution de la technologie de $\mathcal{BD}$ et son impact sur le cycle de vie. Trois évolutions principales ont été identifiées : verticale, interne et horizontale. Malgré la diversité de chaque évolution, nous avons pu dégager les points importants qui sont en relation avec notre domaine d'étude, à savoir les phases de déploiement et physique et surtout leur dépendance. Cette présentation a pu montrer les limites des sites Web spécialisés dans le classement des SGBD, car souvent ils ne prennent pas en compte tous les SGDB (le cas des SGBD sémantiques). Un autre élément concerne la présence des besoins non fonctionnels dans les phases de notre étude (déploiement et physique). Vu la complexité de ces deux phases, le recours aux modèles de coût mathématiques est souvent fortement recommandé. Une grande majorité des travaux de recherche de la communauté de$\mathcal{BD}$ est concentrée sur ces deux phases. En conséquence, une mine d'articles scientifiques est disponible qui présente les résultats des tests associés aux problématiques liées à ces deux phases.

Ce chapitre nous a donnés tous les ingrédients de formaliser, d'étudier la complexité de notre problème de sélection de SGBD et de sa plateforme. Une fois étudié, nous proposons une démarche de sa résolution. Toutes ses étapes seront étudiées dans le chapitre suivant.

Chapitre 3

Etudes des Approches de Selection des SGBD et Plateformes

Sommaire

1 Introduction 57

2 Test de bases de données 59

3 Le rôle des bancs d'essai dans notre démarche 63

3.1 YCSB - Yahoo Cloud Serving Benchmark 63

3.2 Banc d'essai du Transaction Processing Performance Council (TPC) 66

3.2.1 TPC-C 66

3.2.2 TPC-H 66

3.3 Star Schema Benchmark (SSB) 67

3.4 Bancs d'essai pour les bases de données objets 68

3.5 Banc d'essai pour des données spécifiques 70

3.5.1 Données génétiques et GenBase 70

3.5.2 BerlinMod 70

3.5.3 Linear Road Benchmark (LRB) 70

4 Les approches naïves de résolution de notre problème 71

4.1 Test matériel de toutes les plateformes 71

4.2 Des approches basées sur la simulation 71

4.2.1 Modèles de coût mathématique et simulation 72

4.3 Solutions basées sur des retours d'expérience 73

5 Vers la réduction de l'espace de recherche de notre problème 76

5.1 Techniques basées sur la classification des SGBD 76

5.2 Classification basée sur le théorème de CAP et les deux propriétés ACID et BASE 77
5.3 Classification basée sur le nombre d'utilisateurs et le volume de données 78
5.4 Solutions basées sur la décomposition des SGBD 79
6 Bilan et discussion 80
7 Conclusion 80

Résumé. Ce chapitre représente le cœur de nos discussions et nos réflexions sur la nécessité de considérer le problème de sélection de plateforme pour une application donnée. La résolution de ce problème assistera le *déployeur*. Vu la complexité du problème de déploiement d'une $\mathcal{BD}$, nous avons donné une instance de ce problème. Dans ce dernier, nous supposons la présence d'un *déployeur* avec son manifeste incluant les éléments suivants : (a) le schéma de $\mathcal{BD}$, (b) les besoins fonctionnels, (c) les besoins non fonctionnels et (d) ses contraintes. Pour résoudre ce problème, nous supposons que ses éléments sont similaires à ceux proposés par les bancs d'essai. Cette hypothèse nous a amenée à présenter les différents bancs d'essai utilisés par la communauté.

Nous avons présenté trois types d'approches naïves pour résoudre notre problème : les techniques basées sur les tests effectifs, les techniques basées sur la simulation et des techniques basées sur le retour d'expérience. Des limites de chaque type sont données. Des solutions d'élagage ont été également présentées pour réduire la complexité du problème qui peuvent être combinées avec les techniques précédemment décrites.

Cette discussion nous a permis d'identifier la nécessité d'en proposer une autre technique moins coûteuse et fiable, en exploitant les résultats de tests déjà publiés dans les articles scientifiques et les sites Web spécifiques.

Dans le chapitre suivant, nous présentons notre approche d'explicitation de tous les éléments de test (les bases de données, les requêtes, la plateforme de déploiement, les besoins non fonctionnels, etc.). Cette explicitation se fait via des modèles et des ontologies.

1 Introduction

Dans le chapitre précédent, nous avons détaillé les causes du problème de sélection de plateformes dans l'ère de la diversité technologique. Ces causes sont en lien fort avec l'évolution du cycle de vie de conception de $\mathcal{BD}$ et les avancées technologiques. Le problème de sélection est peu formalisé, mais en même temps les entreprises et les développeurs se posent des questions sur le choix de leur SGBD et de leur plateforme. En faisant une simple recherche sur google en tapant "*sgbd que choisir*", nous trouvons 84 200 réponses qui couvrent des retours d'utilisateurs sur l'utilisation des SGBD et plateformes. Ces résultats peuvent être exploités, analysés et bien présentés aux *déployeurs* pour effectuer leur choix. Les propriétaires du site Web *DB-Engine* ont suivi cette démarche, en proposant un classement mensuel des SGBD selon le critère de la popularité. Ce dernier est calculé à partir des statistiques de résultats de recherche de SGBD dans des moteurs de recherche comme Google et Bing, etc. Ce classement est subjectif et il ne prend pas en compte toutes les caractéristiques des plateformes. Une plateforme est composée de plusieurs composants matériels et logiciels, la substitution d'un composant par un autre impacte fortement la performance de la plateforme.

Pour illustrer nos propos, considérons un des composants importants d'un SGBD, à savoir l'optimiseur de requêtes. Son rôle consiste à sélectionner le meilleur plan d'exécution pour une requête données (le plan qui satisfait le ou les besoins non fonctionnels). Ce choix est difficile (NP-complet). Deux approches existent pour répondre à cet objectif. Les approches dirigées par des règles (*Rule-based Approach-RBA*), comme faire descendre aussitôt les sélections au plus bas du plan (*Push Down Selections*) afin de réduire la taille des opérations, puis enchaîner avec des opérations binaires comme la jointure. Ce type d'approche s'avérait insuffisant face à l'explosion des schémas de $\mathcal{BD}$ (en termes de tables). Un autre type d'approches a vu le jour pour venir compléter la RBA, dite dirigé par des modèles de coût (*Cost-based Approach-CBA*). Cette approche permet d'évaluer tous les plans possibles avant de choisir le meilleur. Le coût d'un plan d'exécution est évalué en cumulant le coût des opérations élémentaires, de proche en proche selon l'ordre défini par le plan d'exécution ainsi que l'algorithme, jusqu'à l'obtention du coût total. Les SGBD commerciaux offrent les deux modes de sélection (RBA et CBA). Les classements de SGBD existants ne prennent pas ce type de détails. Un autre exemple concerne la diversité des modèles de stockage utilisés par un SGBD donné (stockage orienté colonne et stockage orienté ligne) [3]. Ces stockages impactent fortement la performance des requêtes. Les systèmes basés sur le stockage colonne favorisent les applications décisionnelles, tandis que les systèmes basés sur le stockage en ligne favorisent les requêtes transactionnelles [126]. Le lien fort entre l'usage et la satisfaction des besoins non fonctionnels existent (A^{BD} :< $C2V, SGBD, Plateforme, Usage, Sat$ >).

Cette discussion fragilise les solutions de choix de SGBD et sa plateforme proposées par des sites Web. Parallèlement la solution exhaustive consiste à tester tous les SGBD et les plateformes est coûteuse. Cette complexité est due aux facteurs suivants.

- L'absence des données de la base. Lors de la conception d'une $\mathcal{BD}$, les données n'existent pas. Même si elles existent, les faire migrer d'une plateforme à une autre représente un processus coûteux en termes de temps et d'argent.

- L'absence des requêtes d'une manière explicite. Ces dernières peuvent être extraites à partir des besoins fonctionnels, mais elles doivent être adaptées aux plateformes de test.

- Fournir toutes les plateformes de tests auprès d'un *déployeur* est une solution imaginaire.

Pour les deux premiers points, le *déployeur* peut faire recours aux bancs d'essai et identifier celui qui est le plus proche à ses données et ses requêtes. Plusieurs bancs d'essai de comparaison ont été proposés par la communauté de $\mathcal{BD}$ à savoir : les bancs d'essai de la famille TPC [8] (TPC-H, TPC-R, TPC-C, etc.), *Yahoo Cloud Serving Benchmark* (YCSB) [60] et *distributed multi-phase Yahoo Cloud Serving* (YCSB++) [145], *Star Schema Benchmark* (SSB), *Linear Road Benchmark* (LRB) [14], *Wisconsin* [72], *BerlinMOD* [76] et *GenBASE* [177].

Pour contourner le dernier point, le *déployeur* peut faire appel à la simulation qui concerne les SGBD et leurs plateformes. Cette simulation exige au *déployeur* la connaissance des détails de l'ensemble de composants des plateformes afin de proposer des modèles mathématiques estimant le coût de chaque fonction du SGBD (par exemple l'optimiseur de requêtes, le gestionnaire de transactions, la réplication, etc.). Souvent un *déployeur* n'a pas toutes les compétences nécessaires pour proposer des modèles mathématiques. Une décomposition d'un SGBD en composants pourrait être une solution pour le *déployeur* afin de tester les composants dont il est expert. Des travaux existent dans la littérature permettant cette décomposition. Ils sont basés sur les modèles de lignes de produits logiciels (*Softawre Product Lines-SPL*) [116]. L'incorporation des SPL dans le contexte des $\mathcal{BD}$ reste un domaine ouvert. Pour plus de détails sur cette thématique, nous recommandons au lecteur la thèse de Selma Bouarar effectuée au sein du laboratoire LIAS [32].

Devant des problèmes complexes, des techniques d'élagage (*pruning*) peuvent être une solution pour réduire l'espace de recherche [24, 117] ensuite utiliser l'une des approches ci-dessous présentées (voir la figure 3.1). Dans notre contexte, le processus d'élagage peut se baser sur plusieurs critères. Parmi ces critères, on trouve : (a) l'usage de l'application (OLTP, OLAP ou OLTP/OLAP) et (b) La considération que le top 10 des SGBD proposés par DB-Engine. Pour chaque type d'usage certaines plateformes sont plus recommandées que d'autres, par exemple le SGBD MySQL pour OLTP et Infobright ICE pour OLAP. Une fois élagué, le processus de test traditionnel peut être effectué sur la base des plateformes restantes. Certes, l'élagage réduit la complexité de notre problème, mais les solutions proposées ont les mêmes limites que celles que nous avons évoquées précédemment.

Devant cette situation, nous proposons une nouvelle approche pour résoudre notre problème. Elle est inspirée des solutions que nous avons précédemment évoquées (les approches dirigées

8. http://www.tpc.org/

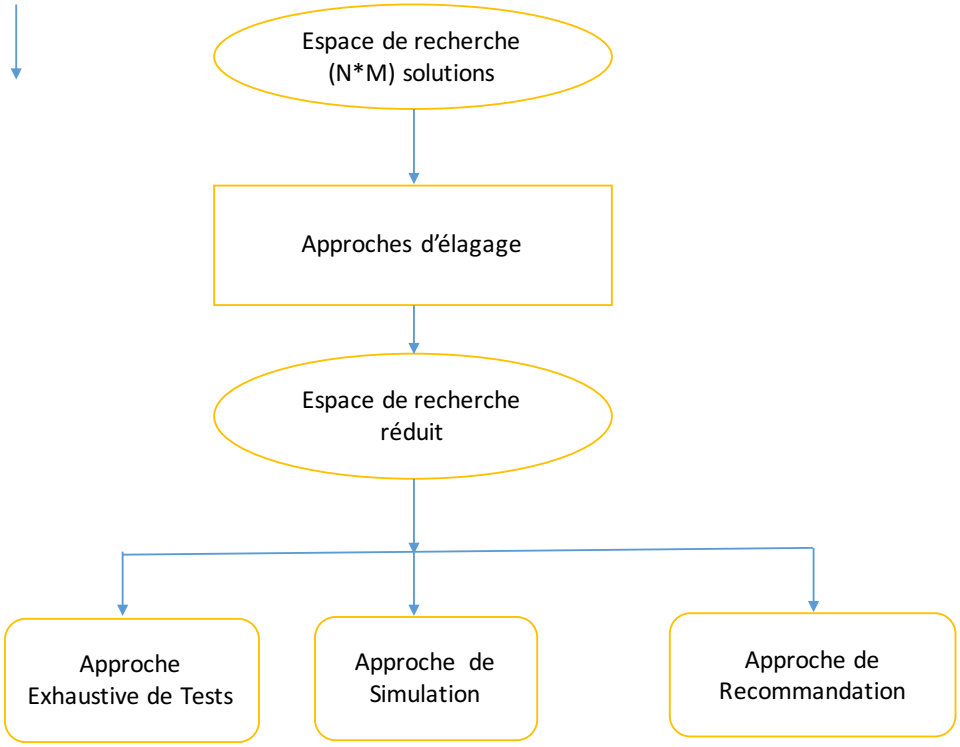


FIGURE 3.1 – Démarche d'élagage de résolution de notre problème

par le retour d'expérience et les approches de tests). toutefois elle diffère selon deux points importants : (i) les générateurs de données de tests et (ii) l'endroit de publication de ces données de test. Au lieu que le *déployeur* teste ses SGBD, il délègue ce processus aux chercheurs scientifiques, qui ont déjà établis certains tests sur des SGBD et des plateformes. Les résultats de ces tests sont publiés dans des articles scientifiques, des brevets et les sites Web spécialisés comme le TPC. Ces résultats sont précis et détaillés par rapport à ce que nous trouvons sur DB-Engine.

Dans ce contexte, ce chapitre présente en détails l'ensemble des approches naïves de résolution de notre problème ainsi que le processus de leur utilisation par un *déployeur*. Ce processus concerne l'identification de la similarité entre les données et les besoins fonctionnels du *déployeur* et ceux des bancs d'essai. Avant de détailler ces aspects, nous présentons la problématique de test dans les $\mathcal{BD}$.

2 Test de bases de données

Une $\mathcal{BD}$ peut être vue comme un produit informatique qui doit être testé. Etant donné qu'elle est associée à un cycle de vie, toutes ses phases doivent être testées et validées [88]. Trois familles de tests existent dans le cadre des $\mathcal{BD}$: (i) *test structurel de la* $\mathcal{BD}$, (ii) *test fonctionnel* et (iii) *test non fonctionnel*. Le premier consiste à traiter des tests concernant les tables,

les colonnes, le schéma, les procédures stockées, les vues et la vérification des déclencheurs, etc. Le second consiste à vérifier la fonctionnalité de la base de données du point de vue de l'utilisateur. Le type le plus courant de tests fonctionnels est la boîte blanche et la boîte noire [197]. Le dernier type de test consiste à vérifier les exigences non fonctionnelles et évaluer la performance du système. L'évaluation de performance devient une étape importante de test du système pour déterminer sa compétence. Les administrateurs de $\mathcal{BD}$ s'appuient souvent sur des outils techniques pour évaluer et valider chaque phase. La sécurité, la performance, le temps de réponse, la consommation d'énergie, etc. sont des contraintes très importantes qui jouent un rôle essentiel dans l'évaluation et la validation du cycle de vie de conception de $\mathcal{BD}$. Pour chaque phase de conception de $\mathcal{BD}$, le tableau 3.1) présente les tâches à vérifier, les critère et les outils d'évaluation nécessaires.

Phase	Spécification	Critère	Outils d'évaluation
Modèle conceptuel	- besoins, analyse de données, modélisation, charge de requêtes [88]	- sécurité, qualité, usage [33]	- Expert & utilisateurs finaux (Vérification de la syntaxe [88], mini $\mathcal{BD}$, retours d'utilisateurs [108], vérification des règles)
Modèle logique	- Structure de données, type de données, domaine d'attributs	- Normalisation [33]	- Vérification du prototype [88]
Modèle physique	- Matériel, structure de stockage, structure d'optimisation	- Performance[88], temps de réponse, consommation d'énergie	- Modèle de coût, bancs d'essai
Implémentation	- Stockage spécial, fichiers de données, chargement de données	- Performance, intégrité, Accès simultané, sécurité	- Tuning [88] (Outils intégrés dans les SGBD)
Exploitation	- Nouvelles données, Accès par les utilisateurs, nouvelles exigences commerciales	- Maintenance	- Audit, tuning

Tableau 3.1 – Test de cycle de vie des bases de données

La phase physique a eu un intérêt particulier pour les activités de test. Cette phase représente

la vitrine de toute application de $\mathcal{BD}$. En conséquence, elle doit répondre à l'ensemble des besoins non fonctionnels collectés durant la phase de besoins. D'après Gling [87], les exigences non fonctionnelles sont distinguées en deux catégories : les exigences de performance et les exigences spécifiques de qualité. Plusieurs types de ces exigences existent :

Les exigences de performance

- *Temps d'exécution :* il représente le temps nécessaire pour les ressources d'exécution d'un système afin de traiter une requête.
- *Temps de réponse ou temps de latence :* il représente le temps entre le lancement d'une requête et le temps d'arrivée de la première réponse. La meilleure valeur de temps de réponse d'une requête correspond à son temps d'exécution.
- *Débit du système ou bande passante :* c'est le nombre de requêtes effectuées dans un intervalle de temps.
- *Pourcentage d'utilisation des ressources :* c'est la valeur qui correspond à la proportion du temps pendant laquelle la ressource est utilisée sur une période donnée.
- *Taux de transmission :* c'est le nombre de tuples produits par unité de temps.

Les exigences de qualité

- Fiabilité : c'est l'étude des défaillances du système telles que les bogues, et les erreurs système, par exemple la détermination du temps entre les pannes et du temps de réparation.
- Facilité d'utilisation : spécification de la qualité du support en ligne, manuels et documentation permettant d'utiliser efficacement un système, par exemple la détermination du temps moyen de formation nécessaire pour s'adapter avec un nouveau système.
- Sécurité : c'est la détermination des protocoles d'authentification et de traçabilité d'accès aux systèmes, par exemple la capacité du système à résister à des tentatives d'usage non-autorisées et le pourcentage d'attaques réussies.
- Disponibilité : c'est le rapport entre le temps pendant laquelle le système fonctionne correctement et la durée totale de fonctionnement.
- Élasticité : capacité de passer à l'échelle et s'adapter au changement de manière dynamique, c'est à dire sans devoir interrompre le traitement des requêtes (par exemple lors de l'ajout ou la suppression d'un nœud).
- Passage à l'échelle : capacité du système de s'adapter face à une montée de charge, nous différencions le passage à l'échelle horizontal et vertical. Cette mesure peut être quantifiée en mesurant l'accroissement de la performance du système avec l'ajout de machines (horizontal), ou l'utilisation de machines individuellement plus performantes (vertical).

- Tolérance aux pannes : le déclenchement de fautes est souvent dû à une panne générale du système (crash), à un nombre énorme de requêtes en cours de traitement ou à une requête à longue durée de traitement. Quand de tels cas surviennent, le système doit être capable de les gérer sans réinitialiser les requêtes.

Dans cette thèse, nous nous concentrons seulement sur les besoins non fonctionnels liés à la performance.

L'intérêt de la phase de conception physique a motivé les communautés académique et industrielle à développer des outils (connues sous le nom *Advisors*) assistant les administrateurs de $\mathcal{BD}$ dans leurs tâches en leur recommandant des structures d'optimisation (vues matérialisées (VM), index (IX), fragmentation horizontale (FH), fragmentation verticale (FV), traitement parallèle (TP), clustering (CL)). Nous pouvons citer les advisors proposés par les éditeurs de SGBD traditionnels : Oracle Access Advisor [64], *Data Tuning Advisor* de Microsoft SQL Server [51], *Design Advisor* de DB2 d'IBM [199]. D'autres advisors académiques ont été proposés comme Parinda [128] et SimulPhD [23].

Ces outils sont basés sur des modèles de coût estimant la performance des requêtes. L'administrateur prpose un schéma de $\mathcal{BD}$ et des requêtes puis l'outil lui propose des recommandations. Ces outils peuvent contribuer partiellement à la résolution de notre problème, si l'objectif de *déployeur* se limite aux tâches d'administration. Il est important de signaler que ces outils sont payants, concernent un nombre limité de SGBD et surtout qu'ils se limitent aux structures d'optimisation. Même si nous retenons ces dernières, ils n'offrent pas un consensus sur l'ensemble de structures utilisées comme le montre le tableau 3.2.

Outil	SGBD	Techniques d'optimisation						Modèle de coût
		Ix	FH	FV	VM	TP	CL	
Oracle Access Advisor	Oracle	X	X		X			Optimiseur
Databse Tuning Advisor	SQL Server	X	X	X	X			Optimiseur
DB2 Advisor	DB2	X	X		X		X	Optimiseur
WarLock	-	X	X			X		Mathématique

Tableau 3.2 – Comparaison des principaux advisors[35]

La présence des advisors au niveau physique va dans le sens de l'ensemble de contributions de cette thèse à savoir la proposition des outils (advisors) pour la phase de déploiement.

Dans la section suivante, présentons les bancs d'essai, qui représentent une des composantes principales de notre approche. Une grande partie des données de tests non fonctionnels est basée sur les schémas et les données des bancs d'essai.

3 Le rôle des bancs d'essai dans notre démarche

Vu l'adoption des bancs d'essai par la communauté de $\mathcal{BD}$ et surtout leur diversité qui suit celle des $\mathcal{BD}$, notre hypothèse qui stipule que *les entrées du déployeur sont similaires à celles d'un banc d'essai est raisonnable et pratique*. Nous supposons également que ce lien est effectué d'une manière manuelle.

Pour illustrer l'intérêt des bancs d'essai, nous consacrons cette section pour les présenter et préciser leurs rôles.

Les bancs d'essai (benchmarks) offrent aux chercheurs, industriels et étudiants un moyen incontournable en termes de données et de requêtes pour évaluer leurs produits et les comparer à d'autres produits tout en considérant les mêmes données et requêtes.

L'un des objectifs principaux d'un banc d'essai est d'estimer les performances d'un SGBD ou de comparer les performances de plusieurs SGBD à travers une série de tests, en exprimant un ensemble de métriques tels que le temps de réponse, le débit, l'énergie consommée, le nombre d'entrées/sorties effectuées et l'espace mémoire ou disque occupé. Généralement les bancs d'essai disposent d'une $\mathcal{BD}$ (un schéma physique, et des outils de génération d'instances de chaque table), un générateur d'une charge de requêtes et des sondes placées au niveau du système cible pour remonter des mesures et de savoir comment réagissent individuellement des composantes du système (mémoire, processeur, disque et réseau). L'efficacité d'un banc d'essai est défini par : (i) son périmètre de couverture des exigences et besoins demandés par rapport au nombre d'utilisateurs, (ii) sa portabilité et sa compatibilité avec les différents systèmes cibles, (iii) sa simplicité de mettre en œuvre, (iv) sa scalabilité par rapport à la taille de la base de données à évaluer et sa variabilité concernant la configuration matérielle (CPU, disque, RAM, etc.) à tester et (iv) sa reproductibilité afin de fournir les mêmes résultats dans le même environnement pour différents tests.

Le tableau 3.3 présente quelques bancs d'essai. Pour chaque banc d'essai, il est mentionné son cas d'utilisation, ses métriques et quelques travaux scientifiques dans lesquels il a été utilisé.

3.1 YCSB - Yahoo Cloud Serving Benchmark

YCSB [48, 60] est une structure extensible pour mesurer la performance de multiples systèmes de gestion de $\mathcal{BD}$ (PNUTS, BigTable, HBase, Hypertable, Azure, Cassandra, CouchDB, Voldemort, MongoDB, OrientDB, Infinispan, Dynomite, Redis, GemFire, GigaSpaces XAP, DynamoDB et autres). Il est conçu pour les systèmes cloud. Il est destiné à déterminer les paramètres d'évaluation tels que l'élasticité, la disponibilité, la réplication, le débit et le temps de latence et il se compose de deux modules (voir la figure 3.2).

- Exécuteur de charges de requêtes, sert à charger les données de tests, générer les opérations qui seront spécialisées et émettre par un client YCSB à une $\mathcal{BD}$.

Banc d'essai	Cas d'utilisation	Certains métriques	Exemples
TPC-H	Base de données décisionnelles	QphH@size, Throughput@size	Évaluation de performance de MySQL Cluster [140]
TPC-C	OLTP	tpmC	Évaluation des ordinateurs utilisant le stockage SSD [196]
TPC-VMS	$\mathcal{BD}$ dans un environnement virtuel	VMStpmC, VMStpsE, VMSQphH	Évaluation de performance dans un environnement virtuel [70]
YCSB	$\mathcal{BD}$ NoSQL en mode cloud	Débit, latence	Benchmarking entre Cassandra, HBase, PNUTS et MySQL [60]
YCSB++	$\mathcal{BD}$ NoSQL en mode cloud	Statistiques internes (CPU, Network), chargement en bloc	Comparaison de performance entre HBase and Accumulo [145]
SSB	Entrepôts de données	Temps d'exécution, coût d'exécution (E/S)	Évaluation de performance des algorithmes des structures d'optimisation (VM, index, etc.) [110]
BerlinMOD	Systèmes temporels et spatiaux	Temps de réponse	Évaluation du système Secondo [76]
GenBase	Biologie et santé	Biclustering, covariance	Performance de PostgreSQL, système de régression R [177]
LUBM	Ontologie	Temps de chargement, temps de réponse	Évaluation de performance des systèmes de base de connaissances OWL [96]
OO7	$\mathcal{BD}$ orienté objet	Débit, efficacité des mises à jour	Comparaison entre un modèle de coût générique et un autre théorique [83]
XOO7	$\mathcal{BD}$ XML	Temps de conversion de données, temps de réponse	
BigDataBench	Moteurs de recherche, réseaux sociaux	Capacité de traitement et consommation d'énergie	Evaluation de trois plateformes : Xeon, Atom et Tilera [152]

Tableau 3.3 – Différents bancs d'essai

- Les charges de requêtes standards représentent une combinaison entre les opérations basiques, lecture, mise à jour, suppression et numérisation. Les opérations de lecture consistent à lire une seule ligne, la numérisation consiste à lire un ensemble de lignes consécutives et la mise à jour permet d'insérer une nouvelle ligne ou de modifier une ligne existante.

La distribution YCSB comprend six charges de requêtes :

- charge de requêtes A : 50% lecture et 50% écriture ;
- charge de requêtes B : 95% lecture et 5% écriture ;
- charge de requêtes C : lecture seule ;
- charge de requêtes D : les derniers enregistrements insérés sont les plus demandés ;
- charge de requêtes E : 95% Scan et 5% lecture ;
- charge de requêtes F : lecture-modification-écriture, le client lit un enregistrement, le modifiera et écrira les modifications.

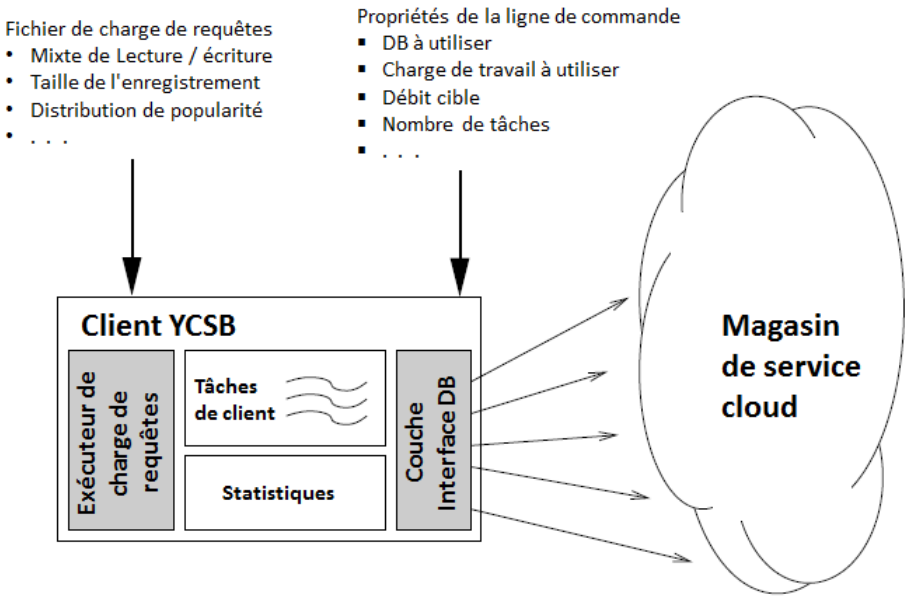


FIGURE 3.2 – Schéma du banc d'essai YCSB

3.2 Banc d'essai du Transaction Processing Performance Council (TPC)

Le TPC (Transaction Processing Performance Council) est un organisme à but non lucratif créé pour définir des bancs d'essais de $\mathcal{BD}$ et diffuser les résultats d'évaluation de performance. Les bancs d'essais proposés par le TPC sont répartis en quatre catégories : TPC-C et TPC-E sont des bancs d'essai pour le traitement des transactions en ligne, TPC-H, TPC-DS et TPC-DI sont destinés pour les systèmes d'aide à la décision, TPC-VMS et TPCx-V pour les $\mathcal{BD}$ virtualisées, TPCx-HS et TPCx-BB pour les données massives (*Big Data*) et TPC-Energy et TPC-Pricing pour des spécifications communes.

3.2.1 TPC-C

Le banc d'essai TPC-C [9] est un repère de traitement des transactions en ligne (OLTP). TPC-C est plus complexe que les repères OLTP précédents tels que TPC-A en raison de ses multiples types de transactions. Il implique un mélange de cinq transactions simultanées de types différents. Elles sont exécutées en ligne ou en file d'attente pour une exécution différée. TPC-C simule un environnement informatique complet où une population d'utilisateurs exécute des transactions sur une $\mathcal{BD}$ qui est composée de neuf types de tables (voir la figure 3.3) avec un large éventail d'enregistrements et de tailles de population. Il est centré sur les principales activités (transactions) d'un environnement d'entrée de commandes. Ces transactions comprennent la saisie et la livraison des commandes, l'enregistrement des paiements, la vérification des commandes et le suivi du niveau des stocks dans les entrepôts. Bien que le banc d'essai TPC-C est destiné à représenter une charge de requêtes générique des fournisseurs de gros (grossistes) et il ne se limite pas à l'activité d'un secteur particulier, mais représente plutôt toute l'industrie qui doit gérer, vendre ou distribuer un produit ou un service [120]. La métrique de performance rapportée par TPC-C est mesurée en transactions par minute (tpmC).

3.2.2 TPC-H

Le TPC-H [10] est un banc d'essai simulant un environnement de $\mathcal{BD}$ de systèmes décisionnelles. Il contient une suite de requêtes dédiées pour les entreprises et de modifications de données simultanées. Les requêtes et les données remplissant la $\mathcal{BD}$ ont été choisies pour avoir une grande pertinence pour l'ensemble du secteur. Ce banc d'essai illustre les systèmes des prises de décisions qui examinent des grands volumes de données. TPC-H exécute des requêtes avec un haut degré de complexité [15]. La métrique de performance rapportée par TPC-H est appelée *TPC-H Composite Query-per-Hour-QphH@size*. Cette métrique indique le nombre de requêtes exécutées pendant une heure par rapport à la taille de la $\mathcal{BD}$ et reflète plusieurs aspects de la capacité du système à traiter les requêtes. Ces aspects incluent la taille de la $\mathcal{BD}$ sélectionnée

9. http://www.tpc.org/tpcc/default.asp
10. http://www.tpc.org/tpch/default.asp

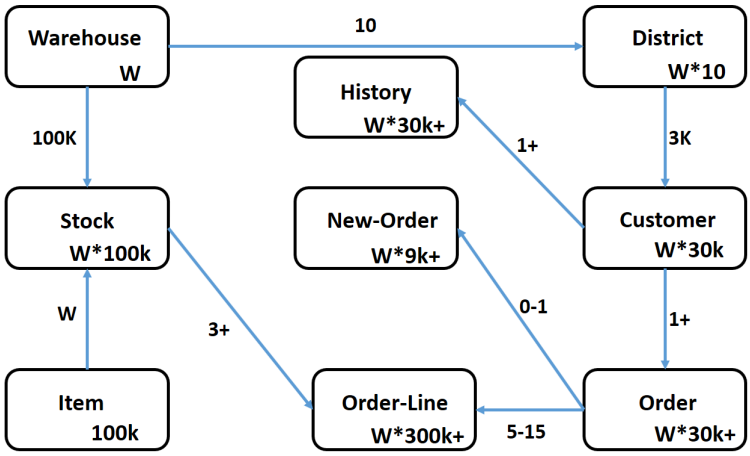


FIGURE 3.3 – Schéma du banc d'essai TPC-C

dans laquelle les requêtes sont exécutées, la puissance de traitement des requêtes lorsque les requêtes sont soumises par un flux unique (Power@size) et le débit de la requête lorsque les requêtes sont soumises par plusieurs utilisateurs simultanés (Throughput@size). Ainsi, la métrique Price/performance représentée par prix par QphH@Size permet de comparer les prix et les performances entre les systèmes [179]. TPC-H consiste à modéliser l'activité d'une entreprise de livraison de produits. A cette fin, il utilise un schéma de $\mathcal{BD}$ simple composé de huit tables de base (voir la figure 3.4). Les tables ont des tailles différentes qui changent proportionnellement à une constante connue sous le nom du facteur d'échelle. Les facteurs d'échelle disponibles sont : 1, 10, 30, 100, 300, 1000, 3000, 10000, 30000 et 100000. Le facteur d'échelle détermine la taille de la $\mathcal{BD}$ en GBytes. Le package TPC-H fournit un générateur de données (DBGEN) pour charger les tables de $\mathcal{BD}$ avec des quantités différentes de données synthétiques. La charge de requêtes contient 22 requêtes et deux procédures de mise à jour, représentant des rafraîchissements de données périodiques. Du point de vue technique, les requêtes incluent une large gamme d'opérateurs et de contraintes de sélectivité, accèdent à un grand pourcentage des données et de tables peuplées et génèrent une activité intensive de disque et de CPU. Le TPC-H définit la charge de requêtes en utilisant QGEN, une application fournie dans le paquage TPC-H.

3.3 Star Schema Benchmark (SSB)

Le SSB [3, 9] est un banc d'essai des entrepôts de données dérivé de TPC-H. Contrairement au TPC-H, il utilise un manuel de schéma en étoile des entrepôts de données. Il contient moins de requêtes que TPC-H et a des exigences moins strictes en terme de formes de calibrage (*tuning*) autorisées et interdites.
Des modifications de schéma ont été apportées au schéma TPC-H pour le transformer en une

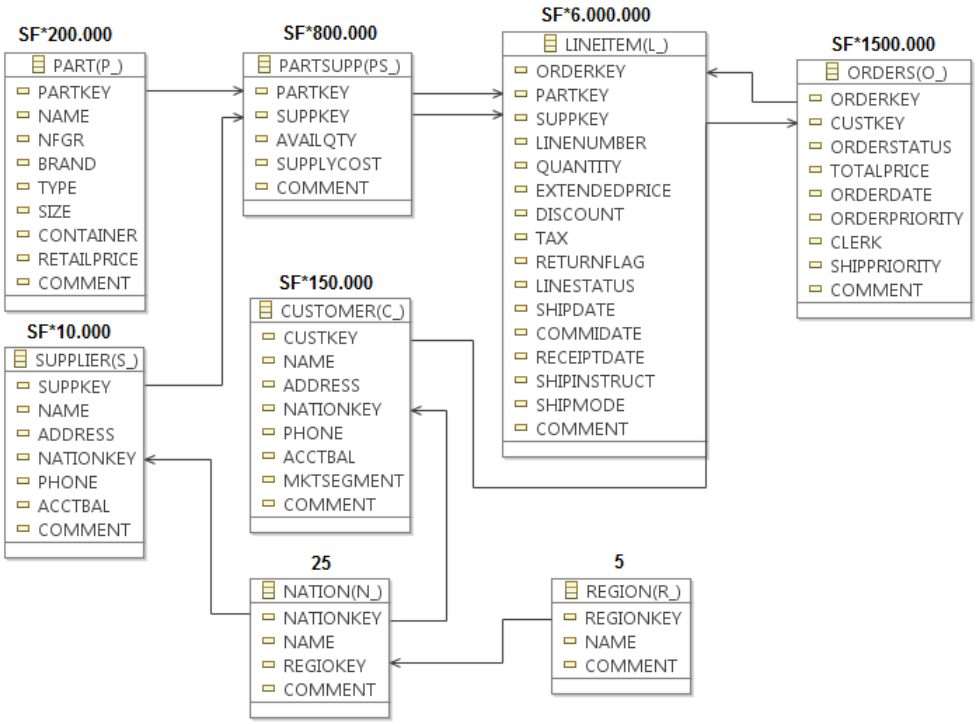


FIGURE 3.4 – Schéma du banc d'essai TPC-H

forme de schéma en étoile plus efficace, comme schématise la figure 3.5. Les tables *LINEITEM* et *ORDERS* sont regroupées en une seule table représentant la table de faits nommée *LINEORDER*. En revanche, les attributs de *LINEITEMS* et d'*ORDERS* sont également dispersées car un entrepôt de données ne stocke pas ces informations dans une table de faits. Ces attributs ne peuvent pas être agrégés car ils consomment un espace de stockage important. La table *PARTSUPP* est supprimée car elle appartient à un datamart différent des données *ORDERS* et *LINEITEM*. Une table de dimension appelée *DATE* est ajoutée au schéma.

3.4 Bancs d'essai pour les bases de données objets

Le banc d'essai BUCKY [47] est conçu comme une $\mathcal{BD}$ de l'université. Les liens Person à Student, Person à Employee, Student à TA, Employee à Staff, Employee à Instructor, Instructor à TA et Instructor au Professor représentent l'héritage entre les types (voir la figure 3.6). Les lignes restantes représentent les relations entre les instances de types et sont étiquetés à chaque extrémité avec le nom par lequel la relation est connue à cette fin. BUCKY est conçu pour être exécuté sur un système relationnel-objet, il peut également être exécuté sur un système relationnel, l'objectif est de tester les fonctionnalités clés qui différent les SGBD orientés aux SGBD Relationnel-objet. Cinq types de requêtes peuvent être testés par BUCKY : (1) les requêtes impliquant des types de lignes avec l'héritage, (2) les requêtes impliquant des références

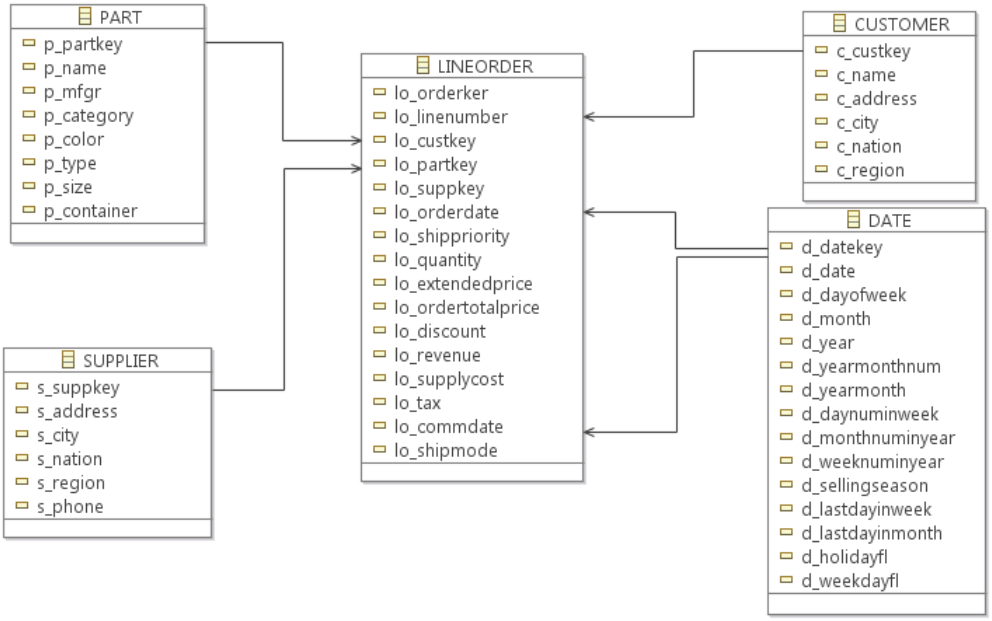


FIGURE 3.5 – Schéma du SSB

inter-objets, (3) les requêtes impliquant des attributs mise en valeur, (4) les requêtes impliquant des méthodes d'objets de ligne et (5) les requêtes impliquant des attributs de type de données abstraites et leurs méthodes.

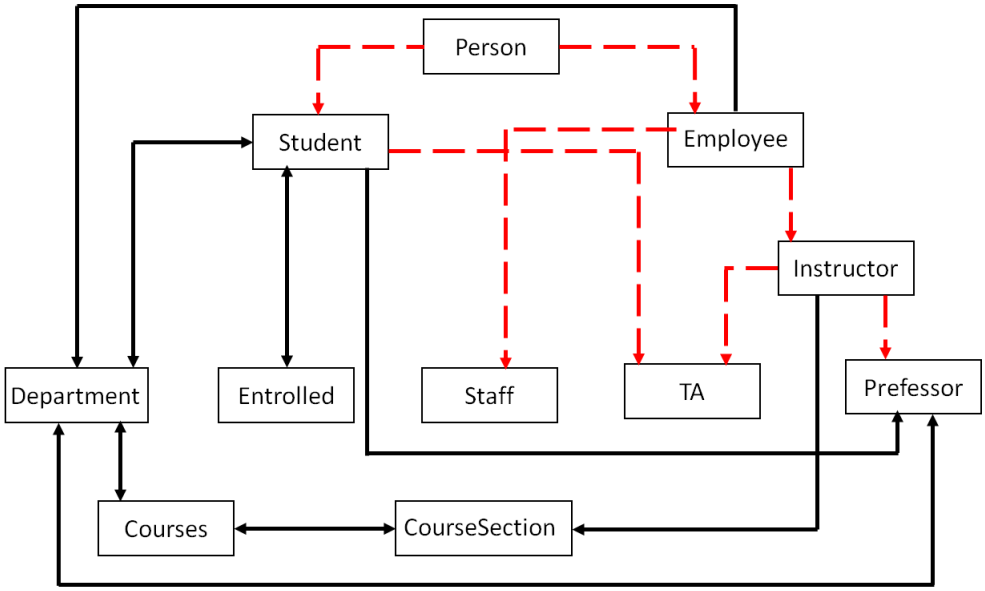


FIGURE 3.6 – Schéma du banc d'essai BUCKY

3.5 Banc d'essai pour des données spécifiques

3.5.1 Données génétiques et GenBase

Les données génétiques sont collectées à travers des échantillons biologiques (sang, salive, sperme, etc.) pour des fins scientifiques. Cette collection de données devient rapidement l'objet de nombreux chercheurs de *Big Data* en raison de son volume. La production rapide de ce genre de données brutes a conduit à l'inflation et la difficulté de la gestion et l'analyse statistiques des données (filtres, jointures, régression, etc.). Cela a attiré l'intention des biologistes et des bioinformaticiens du groupe pharmaceutique suisse *Novartis* et de l'institut de recherche américain *Broad* pour développer un nouveau banc d'essai appelé GenBase dédié aux données de puces à ADN [177]. Les puces à ADN sont utilisées pour mesurer les valeurs d'expression de plusieurs milliers de gènes en même temps et pour faire une variété d'analyse pour identifier la fonction de gènes et la voie de la maladie. Les requêtes représentent les opérations fréquemment exécutées sur les données de puces à ADN telles que la régression et les statistiques. Le but est de tester et d'analyser la performance des systèmes sur quatre jeux de données : les données de puces de gènes, les méta-données du patient, les méta-données du gène et les données d'ontologie génétique [11].

3.5.2 BerlinMod

BerlinMOD [76] est un outil de benchmarking pour les systèmes de gestion de $\mathcal{BD}$ spatio-temporelles. Il permet de comparer les différents détails et de mesurer la performance des requêtes visant des données de points mobiles des différents SGBD spatio-temporels. Avec BerlinMOD, on peut tester la mise en œuvre, comme le déplacement des types de données d'objets, les structures d'index et les opérateurs spatio-temporels.

3.5.3 Linear Road Benchmark (LRB)

Linear Road Benchmark (LRB) est conçu pour les systèmes de gestion de flux de données (*Data Stream Management System-DSMS*) [14] traitant les données en flux continu en exécutant des requêtes historiques continues tout en produisant les résultats de requêtes en temps réel [2]. LRB permet de comparer les caractéristiques de performance des systèmes de ce type et aussi par rapport aux autres systèmes comme les SGBD relationnels. Le but est de mesurer le temps de réponse et le chargement de requêtes. Ce dernier désigne la capacité de traitement d'entrées par un système tout en respectant les délais de réponse et l'exactitude de contraintes.

Cette section a présenté d'une manière détaillée les bancs d'essai les plus populaires. Dans la section suivante, nous présentons certaines approches naïves de résolution de notre problème en utilisant les bancs d'essai.

11. http://istc-bigdata.org/index.php/genbase-a-benchmark-for-the-genomics-era/

4 Les approches naïves de résolution de notre problème

Comme nous l'avons évoqué dans l'introduction de ce chapitre, notre problème peut être résolu naïvement en utilisant les approches classiques : (i) le test matériel proprement dit de toutes les plateformes, (ii) la simulation et (iii) le retour d'expérience, que nous détaillons dans les sections suivantes.

4.1 Test matériel de toutes les plateformes

Cette approche consiste à tester exhaustivement les entrées du *déployeur* sur toutes les plateformes qui peuvent exister. Cette solution est coûteuse pour sa mise en place. Dans la littérature, plusieurs résultats de test de plateformes considérant des bancs d'essai existent dans plusieurs contextes de $\mathcal{BD}$. Le tableau 3.4 recense certains travaux sur l'évaluation de performance des $\mathcal{BD}$ déployées sur un cloud en mentionnant les bancs d'essai utilisés ainsi que leur SGBD.

Travaux	Banc d'essai / Modèle	Systèmes
Shi et al., 2010 [168]	YCSB	HBase, Cassandra, Hive et HadoopDB
Cooper et al., 2010 [60]	YCSB	Cassandra, HBase, PNUTS[12]et MySQL fragmenté
Konstantinou et al., 2011 [114]	YCSB	HBase, Cassandra et Riak
K.-Y. Guo et al., 2016 [95]	Modèle d'évaluation de performance des systèmes élastiques cloud	
Villalpando et al., 2016 [188]	Framework de conception, de validation et de comparaison de performance	

Tableau 3.4 – Recensement des travaux de performance de $\mathcal{BD}$ en mode cloud

Le tableau 3.5 recense les travaux sur l'évaluation de performance des $\mathcal{BD}$ de type Big Data.

4.2 Des approches basées sur la simulation

Tester toutes les plateformes est une solution impraticable. Vu la similarité entre nos travaux et les systèmes complexes qui font appel régulièrement à la simulation, nous pouvons facilement

Travaux	Banc d'essai	Système/Plateforme
Quan et al., 2013 [152]	BigDataBench	Xeon, Atom et Tilera
Kos et al., 2015 [115]	nouveau banc d'essai proposé	Plateformes (centres de données)
Wei et al., 2016 [190]	nouveau banc d'essai proposé	Spark et GraphLab

Tableau 3.5 – Recensement des travaux de performance de $\mathcal{BD}$ *Big Data*

emprunter ce concept pour résoudre notre problème. La simulation dans les phases physique et déploiement de $\mathcal{BD}$ passe souvent par la définition des modèles de coût mathématiques [124].

4.2.1 Modèles de coût mathématique et simulation

Un modèle de coût est exprimé par une fonction ayant des entrées et une sortie. Les entrées représentent les paramètres liés (i) au schéma d'une $\mathcal{BD}$ (ou d'entrepôt de données) gérée par un SGBD utilisant un modèle de stockage et une architecture, (ii) à la charge de requêtes, (iii) à la plateforme de déploiement (machine centralisée, distribuée, parallèle, cloud, etc.). Pour chaque entrée, un ensemble de paramètres lui sont associés. Ces paramètres sont divisés en deux catégories de paramètres : les paramètres calculés comme les facteurs de sélectivité des prédicats et les paramètres non calculés comme la taille des champs, la taille des tables, et la taille du support de stockage. Les paramètres calculés sont souvent obtenus à travers des formules mathématiques et des hypothèses comme 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 celles d'un problème de conception physique. La sortie est représentée par le coût final d'exécution d'une charge de requêtes comme le coût des entrées-sorties (IO) (pour lire et écrire entre la mémoire et le support de stockage), le coût CPU et le coût COM de communication sur le réseau exprimant en fonction de la quantité totale des données transmises. Cette valeur du coût sera exploitée par des algorithmes selon le besoin.

La particularité des modèles de coût est qu'ils ont suivi l'évolution de $\mathcal{BD}$ et les avancées technologiques [25]. La figure 3.7 expose l'évolution du développement des modèles de coût sur quatre générations : la première génération [167] décrit les premiers modèles de coût s'appuyant sur une requête et un schéma de $\mathcal{BD}$ et estimant le coût des opérations simples. La deuxième génération [55] a pris en considération les techniques d'optimisation comme les index et les vues matérialisées. L'estimation du coût lié à l'architecture de déploiement de la $\mathcal{BD}$ (distribuée, parallèle, cluster de $\mathcal{BD}$, etc.) a été intégrée dans la troisième génération [12, 21]. Les modèles de coût de la quatrième génération [144] ont été enrichis par des formules intégrant les modèles de stockage (colonne, ligne) dédiés aux $\mathcal{BD}$.

Les modèles de coût ont plusieurs utilité. Ils peuvent être utilisés par des algorithmes de

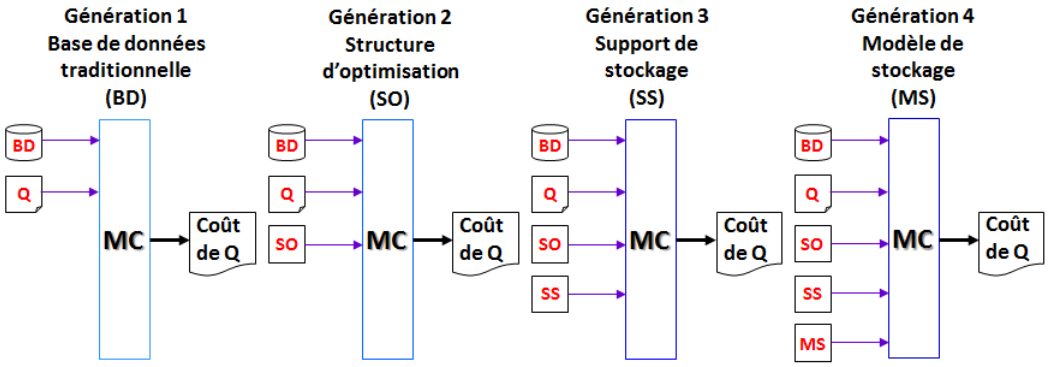


FIGURE 3.7 – Évolution des modèles de coût

sélection de structures d'optimisation pour quantifier la qualité de leur solution. Certains algorithmes d'exploration de l'espace de recherche de ces structures les utilisent comme des fonctions de *fitness* (le cas des algorithmes génétiques). Ils peuvent également être utilisés pour le calibrage de $\mathcal{BD}$.

Ces modèles de coût peuvent être associés à des opérations de base de requêtes comme la sélection, la jointure, l'union, etc. L'évaluation de ces opérations est fortement liée à la plateforme de déploiement. Le tableau 3.6 donne quelques exemples de modèles de coût ainsi que leur utilisation..

Avec les architectures de Cloud et l'émergence du principe *Pay As You Go*, d'autres modèles de coût économiques ont été proposés. Ces modèles prennent en compte les caractéristiques de la $\mathcal{BD}$ ainsi que le coût matériel de sa plateforme. Pour illustrer ces propos, considérons les travaux de Nguyen et al. [141] et Brighen et al. [43] qui ont proposé des modèles de coût mathématiques utilisés pour sélectionner des vues matérialisées pour une $\mathcal{BD}$ déployée sur Cloud.

La simulation est une solution intéressante pour les systèmes complexes, mais pour le cas d'un *déployeur*, elle reste complexe. Car le *déployeur* n'a pas les compétences dans toutes les spécificités du système. Par contre, cette simulation est recommandée pour les chercheurs ayant une certaine expertise en une composante particulière de notre problème. A titre d'exemple, notre laboratoire a proposé un outil (*advisor*) pour la partie physique, appelé SimulPhD [22] dont l'objectif est de recommander des schémas de partitionnement horizontal pour un entrepôt de données et une charge de requêtes définie sur ce dernier. Cet outil a été réalisé par un doctorant et un groupe de 4 étudiants, ce qui montre les moyens colossaux mis en place.

4.3 Solutions basées sur des retours d'expérience

Les solutions basées sur le retour d'expérience et d'usage portent sur la capitalisation des informations de recherches à travers des sites Web ou sur la base des sondages pilotes d'un

Travaux	Structure physique	Algorithmes / Méthodologie
Cornyn et al., 1977 [61]	Plateforme matérielle (IBM, DEC PDP-11[13]et Interdata	Méthodologie utilisée pour calculer les coûts du cycle de vie des systèmes (hardware et software)
Gardarin et al., 1996 [83]	$\mathcal{BD}$ orientée objet	Nouveau modèle de coût paramétré avec des opérateurs adéquats pour les systèmes d'objets
Yang et al. [194]	Vues matérialisées	Algorithmes de sélection des vues à matérialiser
Manegold et al., 2002 [130]	Hiérarchie de mémoire	Approche de combinaisons des modèles de coût basiques
obermeier et al., 2008 [142]	Environnement distribué	Modèle de coût pour les requêtes SPARQL
Zhang et al. [198]		Modèle de coût réel (hardware et exécution)
Nguyen et al. [141]	Vues matérialisées / cloud	Modèle de coût budgétaire
Bausch et al., 2012 [17]	Mémoire flash, Hachage, index, tri	Modèle de coût générique basé sur le modèle de coût du PostgreSQL
Rasin et al., 2013 [153]	Vue matérialisés et stockage en colonne avec compression	Algorithme de conception
Wentao et al., 2013 [193]	CPU et E/S	Calibrage des unités de coûts dans le modèle de coût de l'optimiseur
D. Basu et al., 2015 [16]	Tuning d'index	Approche d'apprentissage

Tableau 3.6 – Recensement des travaux des modèles de coût

échantillon d'utilisateurs. Elles sont fondées autour des paramètres liés aux recherches concernant les SGBD. Comme nous avons cité dans l'introduction le site australien *DB-Engines* [14] vise à recueillir et à présenter des informations sur les SGBD et ses trois grandes familles SGBDR, NoSQL et NewSQL. *DB-Engines* présente à chaque mois un classement des SGBD en fonction de leur popularité (voir la figure 3.8).

Cette popularité est mesurée selon certains paramètres décrits comme suit :

- Le nombre de mentions du système sur les sites Web. Cette statistique est mesurée par le

14. http://db-engines.com/

Rank Nov 2016	Rank Oct 2016	Rank Nov 2015	DBMS	Database Model	Score Nov 2016	Score Oct 2016	Score Nov 2015
1.	1.	1.	Oracle	Relational DBMS	1413.01	-4.09	-67.94
2.	2.	2.	MySQL	Relational DBMS	1373.56	+10.91	+86.71
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1213.80	-0.38	+91.48
4.	↑ 5.	↑ 5.	PostgreSQL	Relational DBMS	325.82	+7.12	+40.13
5.	↓ 4.	↓ 4.	MongoDB	Document store	325.48	+6.67	+20.87
6.	6.	6.	DB2	Relational DBMS	181.46	+0.90	-21.07
7.	7.	↑ 8.	Cassandra	Wide column store	133.97	-1.09	+1.05
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	125.97	+1.30	-14.99
9.	9.	↑ 10.	Redis	Key-value store	115.54	+6.00	+13.13
10.	10.	↓ 9.	SQLite	Relational DBMS	112.00	+3.43	+8.55

FIGURE 3.8 – Classification issue de *DB-Engines*

nombre de résultats de recherche dans les moteurs *Google* [15], *Bing* [16] et *Yandex* [17]. Afin de compter seulement les résultats pertinents, le site utilise le nom du système avec le terme *BD*.

- L'intérêt général pour le système. Pour cette mesure le site se base sur la fréquence des recherches dans le site *Google Trends* [18].
- La fréquence des discussions techniques concernant le système. Ce paramètre est mesuré par le nombre de questions connexes et le nombre d'utilisateurs intéressés sur les sites spécialisés dans le domaine des questions informatiques tels que *Stack Overflow* [19] et *DBA Stack Exchange* [20].
- Le nombre d'offres d'emploi sur les principaux moteurs de recherche d'emploi, tels que *Indeed* [21] et *Simply Hired* [22], dans lesquels le système est mentionné.
- Le nombre de profils dans les réseaux professionnels les plus populaires au niveau international, tels que *LinkedIn* [23] et *Upwork* [24], dans lesquels le système est mentionné.
- La pertinence dans les réseaux sociaux. Le site compte le nombre de tweets dans lesquels le système est mentionné.

DB-Engines calcule la valeur de popularité d'un système en normalisant et en faisant la moyenne des paramètres individuels. Le score de popularité est toujours une valeur relative, qui doit être

15. https://www.google.fr/
16. https://www.bing.com/
17. https://www.yandex.com/
18. https://www.google.fr/trends/
19. http://stackoverflow.com/
20. http://dba.stackexchange.com/
21. http://www.indeed.fr/
22. http://www.simplyhired.fr/
23. https://fr.linkedin.com/
24. https://www.upwork.com/

interprétée seulement en comparaison avec d'autres systèmes. Le classement de *DB-Engines* ne prend pas en compte le nombre d'installations des systèmes, ni leur utilisation dans les systèmes informatiques.

La deuxième référence de classification des SGBD à base d'usage est les sondages pilotes. Dans la plupart des cas, ils ciblent un échantillon d'utilisateurs et un ensemble de SGBD définis à priori. A titre d'exemple dans les $\mathcal{BD}$ de gestion des cours [19], les éducateurs sont confrontés à des options changeantes. Le but est de soutenir ce type de $\mathcal{BD}$, y compris les SGBD commerciaux (tels que Oracle, DB2 et Teradata), des alternatives open source (telles que MySQL et PostgreSQL), et d'autres produits de la gamme Microsoft tels que : Access et SQL Server. Les éducateurs ont basé sur une variété de critères : coût, accessibilité pour les étudiants, facilité d'installation et utilisation, soutien et objectifs pédagogiques. Ils ont cherché à fournir un cadre pour la sélection du système le plus approprié pour les $\mathcal{BD}$ de gestion des cours.

5 Vers la réduction de l'espace de recherche de notre problème

Vu la complexité des solutions basées sur les tests effectifs et la simulation, l'élagage de l'espace de recherche peut être utilisé pour réduire cette complexité. Plusieurs techniques d'élagage peuvent exister : (1) la classification basée sur les familles des SGBD, (2) la classification d'usage principal des SGBD, (3) la classification basée sur les utilisateurs et (4) la classification basée sur la décomposition d'un SGBD.

5.1 Techniques basées sur la classification des SGBD

Nous avons vu dans le chapitre précédent, que les SGBD ont connu plusieurs évolutions. Ces évolutions peuvent être vues comme des familles de SGBD. Nous distinguons alors trois familles : les SGBD Relationnel, les SGBD NoSQL et les SGBD NewSQL [75, 92]. Le tableau 3.7 présente une comparaison entre ces trois familles, en utilisant les critères suivants : le support du langage SQL, le modèle de données utilisé, le mode de stockage utilisé, le support des requêtes complexes, les jointures et le volume de données.

Cette comparaison montre que la grande différence entre ces trois familles concerne de manière générale l'évolutivité horizontale non supportée et les requêtes complexes faiblement gérées par les SGBD relationnels. Ainsi, la jointure est partiellement supportée par les NoSQL et les données sont faiblement sécurisées par les NoSQL et les NewSQL.

	Relationnel	NoSQL	NewSQL
Application	Transaction	Recherche	Analyse
Schéma	Tables	Clé-valeur, document, colonne, graphe	Les deux
Évolution horizontale	Non supporté	Supporté	Supporté
Langage SQL	Supporté	Non supporté	Supporté
Requête complexe	Faible	Fort	Très fort
Jointure	Supportée	Supportée partiellement	Supportée
OLTP	Supportée	Non supporté	Supportée
Transactions	ACID	BASE	ACID
Performance	Faible	Fort	Fort

Tableau 3.7 – Comparaison entre SGBD Relationnels, NoSQL et NewSQL

5.2 Classification basée sur le théorème de CAP et les deux propriétés ACID et BASE

Cette classification est liée directement aux contraintes du théorème de CAP et aux propriétés ACID et BASE (voir la figure 3.9), ces dernières étant issues du théorème de CAP [145].

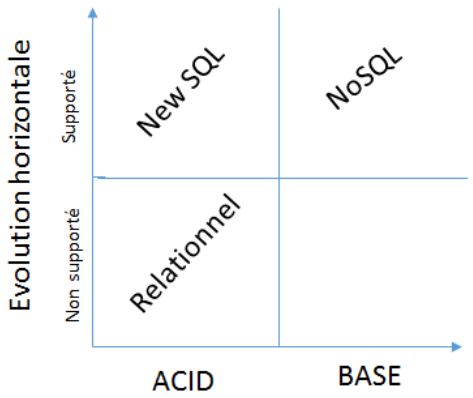


FIGURE 3.9 – Classification des SGBD suivant les propriétés ACID, CAP et BASE

D'après le théorème de CAP, un système ne peut garantir que deux contraintes en même temps. De plus, il n'est pas possible d'assurer dans des systèmes ayant une architecture distribuée des transactions respectant complètement les propriétés ACID. La lecture des contraintes

de CAP montre que BASE assure la tolérance au partitionnement et la disponibilité (AP), et ACID assure la cohérence et la disponibilité (CA).

Cette classification nous permet de distinguer trois classes illustrées par la figure 3.10 :

1. Les systèmes CA ou ACID décrivant les systèmes traditionnels et ceux de la nouvelle génération NewSQL [40, 48, 92].
2. Les systèmes AP ou BASE représentent les systèmes NoSQL comme CouchDB et Cassandra [40, 92].
3. Les systèmes CP représentent les systèmes NoSQL comme MongoDB et HBase [40].

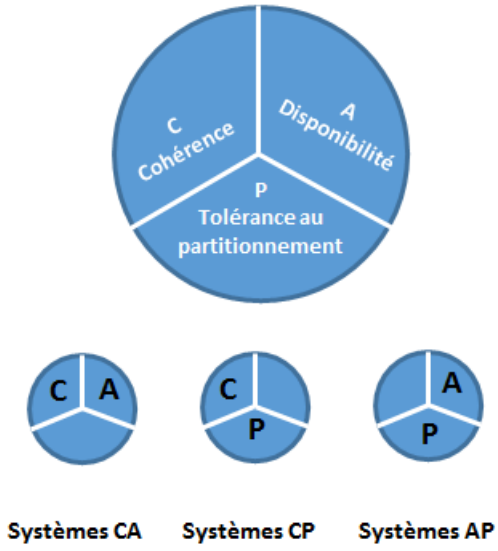


FIGURE 3.10 – Classification des systèmes selon le théorème de CAP

5.3 Classification basée sur le nombre d'utilisateurs et le volume de données

Cette classification nous offre une troisième solution pour choisir notre système cible, mais elle est un peu limitée en termes de fonctionnalités. Cette solution est en effet liée uniquement au nombre d'utilisateurs et à sa croissance ainsi qu'au volume de données à gérer et à sa progression [157].

Cette classification est définie par quatre classes (voir la figure 3.11).

1. les SGBD personnels : ce type de systèmes gère une taille de données de quelques méga-octets avec un nombre très réduit d'utilisateurs ne dépassant pas les 50 et généralement pas d'accès simultanés aux données.

2. Les SGBD de groupe : ce type de systèmes gère une taille assez petite de l'ordre de quelques dizaines de gigaoctet et à peu près 100 utilisateurs.

3. les SGBD d'entreprise : ce type gère une grande taille de données se mesurant en téraoctet et des milliers d'utilisateurs accédant simultanément aux données.

4. les SGBD Web (*Big Data*) : ce type représente la nouvelle technologie des SGBD permettant le traitement de données massives avec un nombre d'utilisateurs presque illimité.

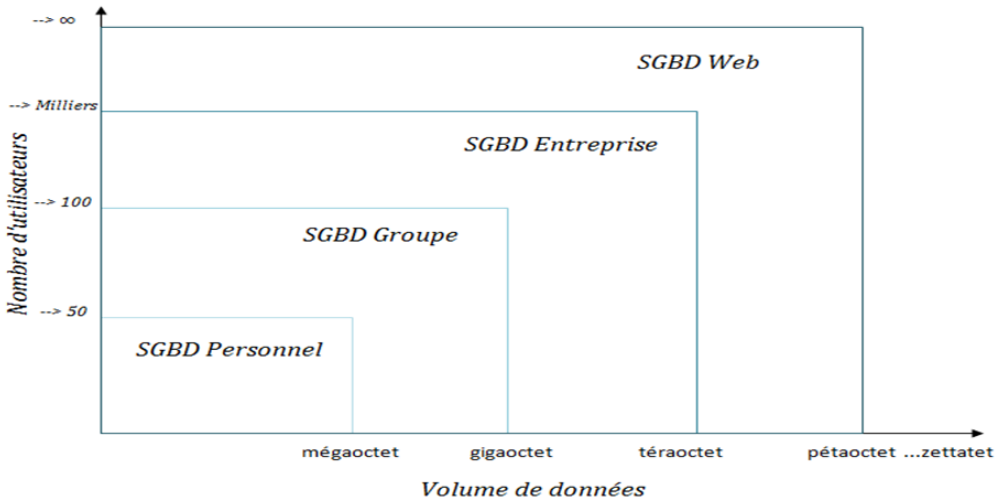


FIGURE 3.11 – Classification selon le nombre d'utilisateurs et le volume de données

5.4 Solutions basées sur la décomposition des SGBD

La décomposition des SGBD est une autre occasion de traiter le problème de sélection des meilleurs SGBD. Cette tentative a été motivée par le slogan *"one size fits all"* [162] en intégrant la variabilité dans la conception des SGBD. Ils ont adopté les modèles de lignes de produits logiciels (Softawre Product Lines). Cela permet de concevoir des systèmes spécialisés en faisant varier certains paramètres des SGBD d'origine pour respecter les nouvelles exigences telles que l'énergie. Pour ce but, des efforts ont été déployés pour le modèle conceptuel des fonctionnalités des SGBD. Rosenmuller et al. [162] ont présenté un travail sur *le projet de recherche FAME-DBMS*[25]. L'objectif est de développer, étendre et évaluer des techniques et des outils pour implémenter et personnaliser les SGBD. Ces techniques doivent tenir compte des exigences particulières des systèmes embarqués. Pour cela, ils ont utilisé les modèles de lignes de produits logiciels basés sur la composition statique des fonctionnalités.
La complexité et le caractère moins prévisible des SGBD (c-à-d la cohérence de la performance avec une augmentation des fonctionnalités et de la croissance des données n'est pas certaine)

25. https://www4.cs.fau.de/Research/FAME-DBMS/

ont conduit les chercheurs et les ingénieurs des $\mathcal{BD}$ à revisiter les architectures des SGBD pour répondre à la nouvelle tendance. À cette fin, les auteurs de [185] ont proposé un SGBD biologique appelé *SGBD cellulaire*. Cette architecture a été inspirée par le projet FAME-DBMS permettant le développement de SGBD hautement personnalisables et autonomes. La principale limitation de ces efforts est qu'ils ne considèrent que quelques concepts et propriétés (des *SGBD les plus populaires* et ignorent les autres SGBD pertinents [86].

6 Bilan et discussion

Les solutions basées sur les résultats de tests à savoir les approches basées sur les tests effectifs et la simulation dirigée par des modèles de coût restent coûteuses et nécessitent des efforts de mise en places considérables. Ce coût couvre le prix des plateformes (disque, CPU, mémoire et réseau, etc.) et logiciels (systèmes d'exploitation, SGBD, etc.), installation, configuration, maintenance, formation, etc.

L'efficacité des solutions basées sur le retour d'expérience repose sur la collaboration de façon globale du plus grand nombre d'acteurs potentiels. L'étape la plus difficile dans le retour d'expérience est l'analyse des informations recueillies selon la crédibilité des utilisateurs.

Les techniques d'élagage contribuent à la réduction de l'espace de recherche de notre problème, et sa résolution. Elles peuvent réduire la complexité du problème en termes de coût et d'efforts. Mais vu le nombre de SGBD par familles (ou classes), ces solutions restent également coûteuses.

7 Conclusion

Ce chapitre représente le cœur de nos discussions et nos réflexions sur la nécessité de considérer le problème de sélection du SGBD et de plateforme pour une application donnée. La résolution de ce problème assistera le *déployeur*. Vu la complexité du problème de déploiement d'une $\mathcal{BD}$, nous avons donné une instance de ce problème. Dans ce dernier, nous supposons la présence d'un *déployeur* avec son manifeste incluant les éléments suivants : (a) le schéma de $\mathcal{BD}$, (b) les besoins fonctionnels, (c) les besoins non fonctionnels et (d) ses contraintes. Pour résoudre ce problème, nous supposons que ses éléments sont similaires à ceux proposés par les bancs d'essai. Cette hypothèse nous a amenés à présenter les différents bancs d'essai utilisés par la communauté.

Nous avons présenté trois types d'approches naïves pour résoudre notre problème : les techniques basées sur les tests effectifs, les techniques basées sur la simulation et des techniques basées sur le retour d'expérience. Des limites de chaque type sont données. Des solutions d'élagage ont été également présentées pour réduire la complexité du problème qui peuvent être

combinées avec les techniques précédemment décrites.

Cette discussion nous a permis d'identifier la nécessité d'en proposer une autre technique moins coûteuse et fiable, en exploitant les résultats de tests déjà publiés dans les articles scientifiques et les sites Web spécifiques.

Dans le chapitre suivant, nous présentons notre approche d'explicitation de tous les éléments de test (les $\mathcal{BD}$, les requêtes, la plateforme de déploiement, les besoins non fonctionnels, etc.). Cette explicitation se fait via des modèles et des ontologies.

Deuxième partie

Contributions

Chapitre 4

Vers une explicitation des composantes de l'environnement de tests

Sommaire

1 Introduction 87

2 Description de l'environnement de tests 87

2.1 Environnement de test 88

2.2 Formalisation et structuration 89

3 Composantes de l'environnement de test 90

3.1 Métriques de performance 92

3.2 Modèle de coût et algorithme 92

3.3 Jeux de données et requêtes 93

3.4 SGBD et Plateformes 96

4 Ontologie pour les des systèmes de base de données (SBD) 98

4.1 Définition d'une ontologie 98

4.2 Différents types d'ontologies 100

4.3 Langages de représentation des ontologies 100

4.3.1 Formalisme RDFS 100

4.3.2 Formalisme DAML+OIL 101

4.3.3 Formalisme OWL 101

4.3.4 Formalisme PLIB 101

4.4 Étapes de conception des ontologies 101

4.5 SPARQL : Langage d'interrogation 102

4.6 Présentation de l'ontologie des SBD 103

4.6.1 Composantes du modèle des SBD 104

4.6.2 Classe des SGBD (DBMS) 105
4.6.3 Classe du matériel (Hardware) 108
4.6.4 Classe des systèmes d'exploitation (Operating_System) 109
4.6.5 Exemple d'usage de SPARQL 110
5 Conclusion 111

Résumé. Toute solution pour résoudre notre problème doit offrir une description détaillée de tout l'environnement de tests non fonctionnels. Cela aide les *déployeurs* à considérer certaines dimensions de cet environnement et connaitre leur spécification. Les organismes comme TPC sont en avance sur cette explicitation de leur environnement. Pour chaque banc d'essai, le TPC nous offre l'ensemble de spécification de chaque banc d'essai connu sous le nom *TPC-H specification* (si nous considérons le banc d'essai TPC-H). L'objectif de ce chapitre est de formaliser et de décrire l'environnement de test et le transformer en une entité mémorisable et machinable. En effet, il présente une description des environnements de tests afin d'expliciter leurs composantes essentielles. La description présentée est faite de manière progressive allant d'une expression semi-formelle jusqu'à l'obtention d'une ontologie du domaine des systèmes de $\mathcal{BD}$.

1 Introduction

Les deux derniers chapitres présentent l'univers de discours des SGBD et une étude comparative de certaines techniques de sélection de SGBD vis-à-vis des besoins du *déployeur*. Nous avons montré que les techniques présentées en plus de leur complexité ne répondent que partiellement au problème de sélection de SGBD et nous avons montré l'importance d'avoir une technique moins coûteuse tout en exploitant les travaux existants autour de tests non fonctionnels. Vu la complexité matérielle et financière de notre problème, nous avons proposé une approche de persistance de données de tests dans un entrepôt de données. Pour ce faire, le schéma de cet entrepôt de données doit être complètement construit à partir des environnements de tests publiés dans les articles scientifiques. Cette construction contribue à la description et l'explicitation de toutes les dimensions de l'environnement de tests, extrait à partir des articles scientifiques dans des conférences dédiées aux *Core Databases*. L'un des intérêts de la présence d'un tel entrepôt est la reproduction et la comparaison des résultats des chercheurs. Le professeur Jens Dittrich, un des chercheurs confirmés dans le domaine des $\mathcal{BD}$ [26], évoque dans une présentation consacrée à la gestion des données [27], la nécessité de reproduire les données de tests (à savoir les SGBD, les algorithmes utilisés, les jeux de données, les plateformes de déploiement, etc.), en particulier quand il s'agit d'une publication scientifique qui requiert des résultats améliorés par rapport aux travaux existants.

Ce chapitre décrit l'ensemble des dimensions de notre entrepôt de données. Pour faciliter cette description, nous proposons un exemple de motivation présentant un environnement de test qui sera utilisé tout au long de ce chapitre. Nous présentons par la suite la formalisation du processus d'un test non fonctionnel ainsi que la modélisation de toutes ses composantes. Nous utilisons des ontologies conceptuelles pour décrire chaque dimension. Le recours aux ontologies permet d'expliciter le sens de toutes les dimensions.

2 Description de l'environnement de tests

Souvent les composantes constituant les environnements de tests ainsi que les résultats de tests sont localisés dans des articles scientifiques publiés par les chercheurs ou dans des sites Web attachés à des organismes spécialisés comme le TPC (*Transaction Processing Council*). En ce qui concerne les articles scientifiques, l'environnement de test et les résultats se trouve souvent dans la section de l'étude expérimentale de validation et/ou d'évaluation. Ces résultats représentent une mine de données de tests qui mérite d'être exploitée pour comparer et tester les nouveaux produits.

26. https://infosys.uni-saarland.de/people/dittrich.php
27. https://www.youtube.com/watch?v=O7Qgo6RSzmE

2.1 Environnement de test

L'environnement de test désigne l'ensemble des composantes d'exécution (matérielles et logicielles), d'usage et de satisfaction dont dépendent les tests d'un système de base de données. En effet, un test représente une expérience scientifique articulée autour de trois catégories d'éléments : les données en entrée, la manière dont le test se déroule et les résultats trouvés par rapport aux objectifs du test. Comme mentionné auparavant, les chercheurs dans le domaine des bases de données trouvent des difficultés pour reproduire des tests qui sont déjà effectués. Cette difficulté est due au manque de traçabilité et de transparence des tests en l'occurrence leurs environnements.

Afin de pouvoir réutiliser les environnements pour reproduire des tests et avoir la même granularité de comparaison avec d'autres tests, nous devons alors identifier et mémoriser les éléments qui ont un impact sur les résultats des tests comme : les jeux de données utilisés et leurs structurations, la description des requêtes et leurs charges, la configuration matérielle, leurs systèmes d'exploitation et leurs caractéristiques de stockage et de calculs, l'architecture de déploiement en précisant les caractéristiques et l'infrastructure système, les structures d'optimisation des requêtes ainsi que les outils spécialisés tels que les modèles de coûts. Une fois l'environnement de tests serait une entité explicite et mémorisable, la reproduction des tests dans les mêmes environnements d'origine devient une tâche faisable.

Le tableau 4.1 que nous avons élaboré présente un extrait de l'ensemble d'informations qui existent dans une publication scientifique de notre équipe de recherche [164]. Nous avons opté pour ce choix car le contact direct avec les auteurs nous a aidés non seulement à extraire les informations noyées dans les discussions de l'article mais aussi à lever certaines ambiguïtés sémantiques sur d'autres éléments. L'objectif des tests présentés dans l'article était de minimiser la consommation d'énergie et le temps de réponse lors de l'exécution des requêtes.

Pour établir le tableau 4.1 et classifier les données, nous nous sommes posées trois questions essentielles :

1. Quel est l'objectif et le type de performance ciblé par le test ?
2. Quelle est la technique suivie pour effectuer le test ?
3. Quels sont les données dont la technique a besoin pour produire un résultat qui correspond à l'objectif et au type de performance fixés ?

Parmi les composantes de cet environnement de test on trouve que l'algorithme génétique de tri non dominé (NSGA-II) a été choisi comme un algorithme d'optimisation afin de résoudre le problème pour une charge de requêtes donnée. L'environnement de test est également décrit par d'autres paramètres liés à la base de données, aux requêtes et à la plateforme. Les paramètres de la base de données sont : le schéma, le jeu de données, la compression des données et la stratégie de traitement des requêtes. On s'intéresse aussi aux types de ces dernières. Les paramètres de la

plateforme sont l'architecture du déploiement, la mémoire principale en tant que périphérique de stockage principal et le disque dur en tant que périphérique de stockage secondaire. Le tableau 4.1 résume ces éléments.

Plateforme	Marque : Dell precision T1500 CPU : Intel Core i5 2.27GHz, Mémoire : 4GB of DDR3 Système d'exploitation : Ubuntu 14.04 LTS kernel 3.13 Déploiement : Centralisé Hypothèse : Sans cache
Ensemble de données	Star Schema Benchmark (SSB), Taille : 100 GB Requêtes de Star schema Benchmark (SSB) Structures d'optimisation : Vues matérialisés
SGBD	Oracle 11gR2
Algorithme	Algorithme génétique de tri non dominé NSGA II
Métriques	Temps de réponse, coût de la CPU, coût d'entrée-sortie et coût de consommation d'énergie
Matériel externe	Watts UP? Pro ES [28]
Type de matériel expérimental	Simulation et réel

Tableau 4.1 – Environnement de test

2.2 Formalisation et structuration

Nous présentons une formalisation de l'environnement de test afin de clarifier, structurer et classifier ses principales composantes. La classification est basée sur le rôle de chaque composante dans un environnement de test. Nous prenons l'illustration du tableau 4.1 comme un exemple de déroulement.

- **Test :** un test représente une expérience scientifique articulée autour de trois éléments : les entrées $Inputs_i$, la technique suivie pour effectuer le test $Means_i$ et les résultats escomptés (les sorties $Outputs_i$). $Test_i =< Inputs_i, Means_i, Outputs_i >$.

- **Entrées (Inputs) :** les entrées ($Inputs_i$) représentent les spécifications de M composantes impliquées dans un test réalisé ($Test_i$). $Inputs_i =< IN_1, IN_2, .., IN_M >$, chaque IN_j est défini par un ensemble de caractéristiques issues de l'un des M domaines suivants : SGBD, Plateformes Matérielles (PM), Plateformes Logicielles (PL), Schémas (S) et Requêtes (Q) , tel que $IN_j \in \{SGBD \oplus PM \oplus PL \oplus S \oplus Q\}$.

  Exemple : Dans l'exemple de déroulement, nous prenons les entrées ($Inputs_{exp}$) relatives aux quatre composantes : SGBD, Plateforme matérielle (PM), Plateforme logicielle (PL),

28. https://www.wattsupmeters.com/

Schéma (S) et Requête (Q). $Inputs_{exp} =< IN_1^{exp}, IN_2^{exp}, IN_3^{exp}, IN_4^{exp}, IN_5^{exp} >$
- $IN_1^{exp} = (SGBD_{Nom} : Oracle, SGBD_{Version} : 11gR12)$
- $IN_2^{exp} = (CPU_{Frequence} : 2.27GHz, CPU_{Marque} : Intel - i5, Memoire_{Capacite} : 4GBytes)$
- $IN_3^{exp} = (OS_{Nom} : Ubuntu, OS_{Version} : 14.04)$
- $IN_4^{exp} = (Schema_{Nom} : SSB, Donnees_{Taille} : 100GBytes)$
- $IN_5^{exp} = (Requete_{Nom} : SSB - Q1.1)$

- **Sorties (Outputs) :** le but d'un test est de trouver la configuration optimale qui minimise ou maximise une mesure pour répondre à une exigence non fonctionnelle. Les sorties $Outputs_i$ d'un test $Test_i$ représentent la métrique de performance visée, l'objectif, la valeur obtenue ainsi que son unité de mesure. Par conséquent, $Outputs_i =< Metrique, objectif, valeur, unit >$.

  Exemple : D'après le tableau 4.1, l'objectif des auteurs est de trouver une configuration pour minimiser le temps de réponse de l'exécution des requêtes SSB.
  $Output_{exemple} =<$ ($MetriqueType$: temps de réponse), ($Objectif$: minimisation de temps de réponse), ($Valeur$: temps de réponse de la requête SSB), ($Unite$: seconde) $>$.

- **Moyens (Means) :** Il existe deux techniques à suivre pour effectuer les tests : la simulation et l'expérimentation réelle. En ce qui concerne la simulation, les résultats de tests sont obtenus suite à l'utilisation des modèles de coûts mathématiques qui permettent d'avoir des valeurs estimés proches de la réalité mais avec une marge d'erreur. L'expérimentation, quant à elle, est le fait de tester sur des vraies plateformes ce qui permet d'obtenir des résultats mesurés. La deuxième technique est beaucoup plus utilisée par les grandes entreprises que par les chercheurs, malgré elle est coûteuse en termes d'architecture matérielle, de temps et d'argent.

Si nous prenons en compte ce formalisme, le tableau 4.1 devient comme suit (voir le tableau 4.2)

3 Composantes de l'environnement de test

A partir de la formalisation effectuée nous pouvons maintenant identifier les composantes d'un environnement de test. Ce dernier peut être vu comme un conteneur central qui regroupe des informations liées aux composantes. Nous avons modélisé l'environnement en utilisant le formalisme UML (*Unified Modeling Language*) [29] comme c'est illustré dans la figure 4.1. La classe `Test` est la racine du modèle. Elle est liée, à travers les relations de composition, aux autres composantes, à savoir (i) le schéma et le jeu de données (la classe `DataSet`), (ii) la charge de requête (la classe `Query`), (iii) la plateforme (les classes `HardwareArchitecture` et `OperatingSystem` qui héritent de la classe `Platform`), (iv) le SGBD (la classe `DBMS`), (v) la

29. http://www.omg.org/spec/UML/

Entrées	**Laboratoire**	LIAS/ENSMA
	Date	14/05/2015
	Plateforme	Marque : Dell precision T1500 CPU : Intel Core i5 2.27GHz CPU : Mémoire : 4GB of DDR3
	Ensemble de données	Star Schema Benchmark (SSB), Taille : 100 GB
	Système d'exploitation	Ubuntu 14.04 LTS kernel 3.13
	Charge de requêtes	Requêtes de Star schema Benchmark (SSB)
	Déploiement	Centralisé
	Structures d'optimisation	Vues matérialisés
	SGBD	Oracle 11gR2
	Algorithme	Algorithme génétique de tri non dominé NSGA II
Sorties	**Métriques**	Temps de réponse, coût de la CPU, coût d'entrée-sortie et coût de consommation d'énergie
Moyens	**Matériel externe**	Watts UP? Pro ES
	Type de matériel expérimental	Simulation et réel

Tableau 4.2 – Environnement de test selon le formalisme

technique utilisée (les classes `RealExperimentation` et `CostModel` qui héritent de la classe `TechniqueType`) et la métrique (la classe `Metric`).

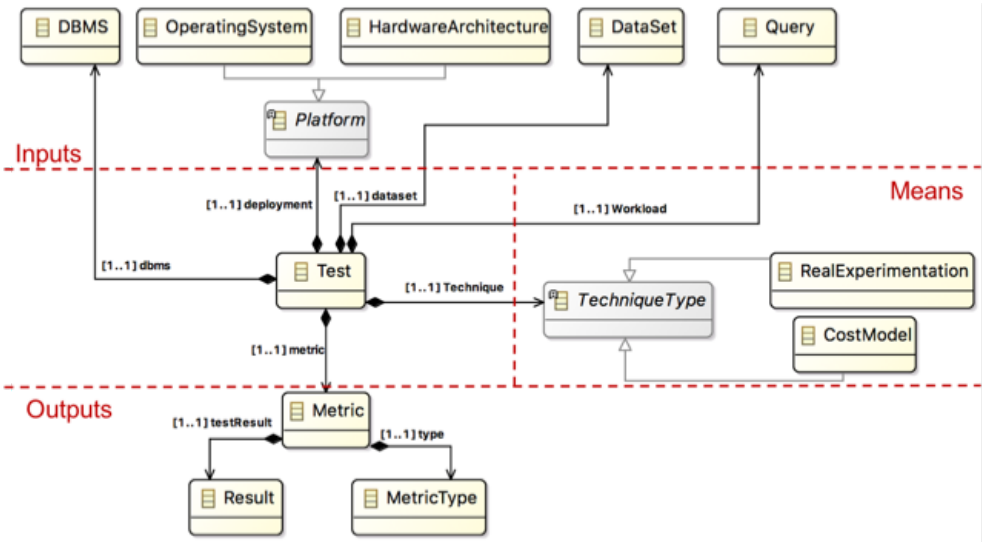


FIGURE 4.1 – Extrait du modèle de l'environnement de test mettant l'accent sur la classe `Test` et ses composantes

En ce qui concerne les composantes de l'environnement de test, la figure 4.1 ne contient que

la classe racine (c-à-d le point de départ en cas d'instanciation) de chaque composante. Nous détaillerons par la suite chacune des ces composantes.

3.1 Métriques de performance

Dans note contexte, une métrique de performance représente l'objectif du test pour répondre à une des exigences non fonctionnelles, également appelées attributs de qualité [161].

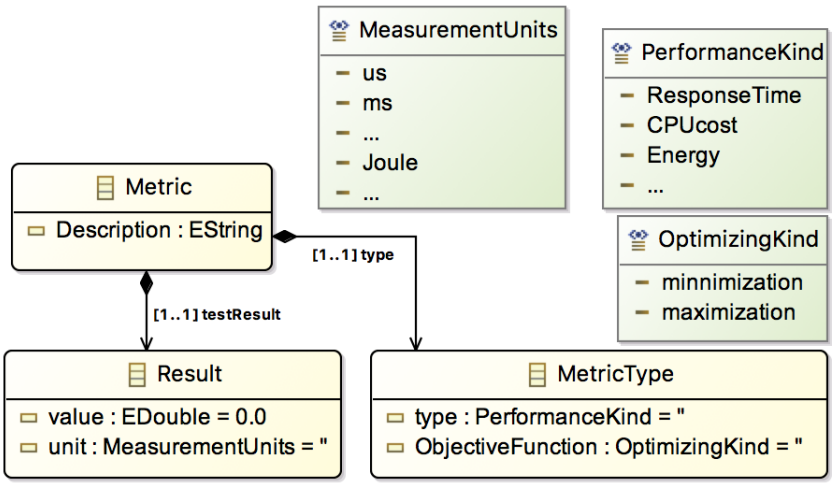


FIGURE 4.2 – Extrait du modèle de l'environnement de test mettant l'accent sur la composante `Metric`

Pour évaluer les exigences non fonctionnelles correspondant à la phase de déploiement, nous utilisons de nombreuses métriques qui doivent être maximisées ou minimisées. Nous pouvons citer certaines métriques classiques telles que le temps de réponse, le débit du système, consommation d'énergie, etc. La figure 4.2 montre les caractéristiques dont nous avons besoins pour décrire finement les éléments de sortie d'un environnement de test. Ces caractéristiques sont exprimées en digramme de classe UML. Chaque métrique d'un test effectué (instance de la classe `Metric`) est définie par un résultat (la classe `Result` et ses attributs), par le type de métrique de performance et par sa fonction objective (la classe `MetricType` et ses attributs).

3.2 Modèle de coût et algorithme

Le modèle de coût est un modèle mathématique dédié à l'estimation du coût de chaque plan d'exécution d'une requête donnée par rapport à une infrastructure de déploiement. Cette technique est actuellement utilisée par la majorité des SGBD. Elle représente même un des points essentiels de compétitivité pour comparer et différencier les SGBD. En effet, dans les SGBD commerciaux, les modèles de coût sont confidentiels car ils déterminent la qualité de l'optimiseur des requêtes.

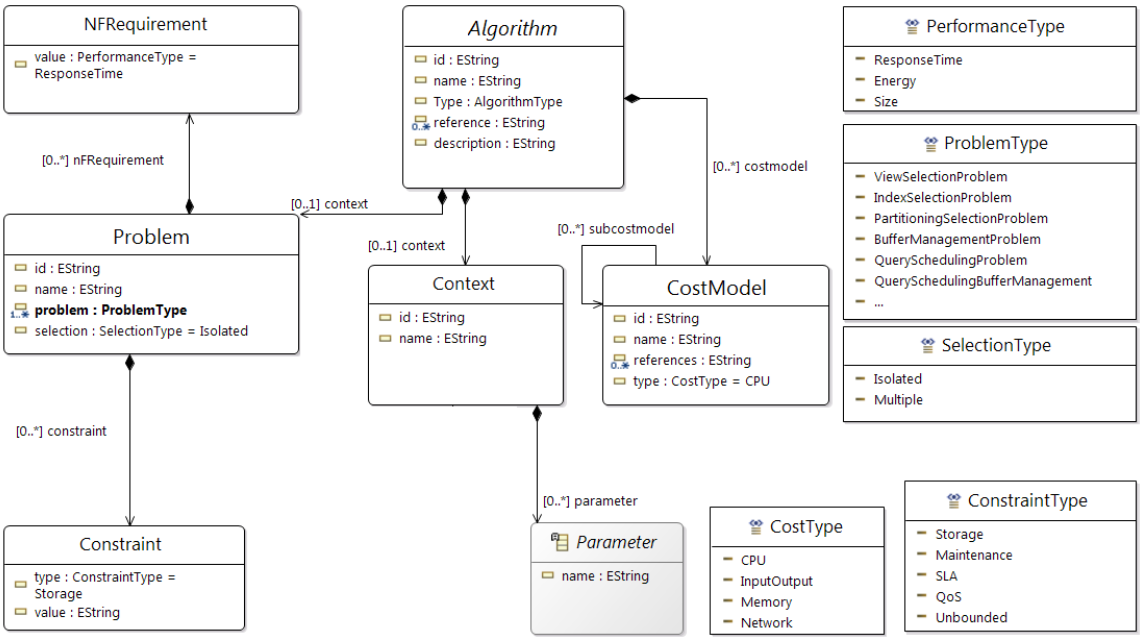


FIGURE 4.3 – Extrait du modèle de l'environnement de test mettant l'accent sur la composante `CostModel`

La figure 4.3 détaille les éléments dédiés à exprimer les modèles de coût (la classe `CostModel`). L'utilisation des modèles de coût permet d'étudier les coûts de toute les configurations possibles d'une manière exhaustive, ce qui mène vers un problème d'explosion combinatoire. C'est pour cela que les modèles de coût sont généralement accompagnés par l'utilisation des algorithmes d'optimisation afin de trouver un résultat optimal (la classe `Algorithm`). Notons que comme on s'intéresse à la réutilisation des travaux existants, les classes `Algorithm` et `CostModel` sont caractérisées par l'attribut `references` qui indique les articles scientifiques qui présentent en détail ces éléments. Nous précisons également qu'un modèle de coût ne délivre un résultat qu'on peut qualifier comme valide que quand on l'applique dans le contexte (la classe `Context`) qui le correspond. Ce contexte représente l'ensemble des paramètres (la classe `Parameter`) relatifs au système en cours de test. Autrement dit, les paramètres d'un modèle de coût sont issus des composantes d'entrées de l'environnement de test. De plus, le type de l'exigence non fonctionnelle qui caractérise un modèle de coût est le même que le type de la métrique de performance de l'environnement de test.

3.3 Jeux de données et requêtes

Dans le processus de création d'une base de données, l'établissement du schéma de la base de données représente l'étape fondamentale, y compris pour les bases de données NoSQL qui sont avantageuses par l'absence d'un schéma prédéfini. Cette élaboration permet de faciliter la construction des requêtes. En effet, nous avons élaboré deux modèles pour les bases de données

relationnelles et NoSQL, schématisés dans les figures 4.4 et 4.5.

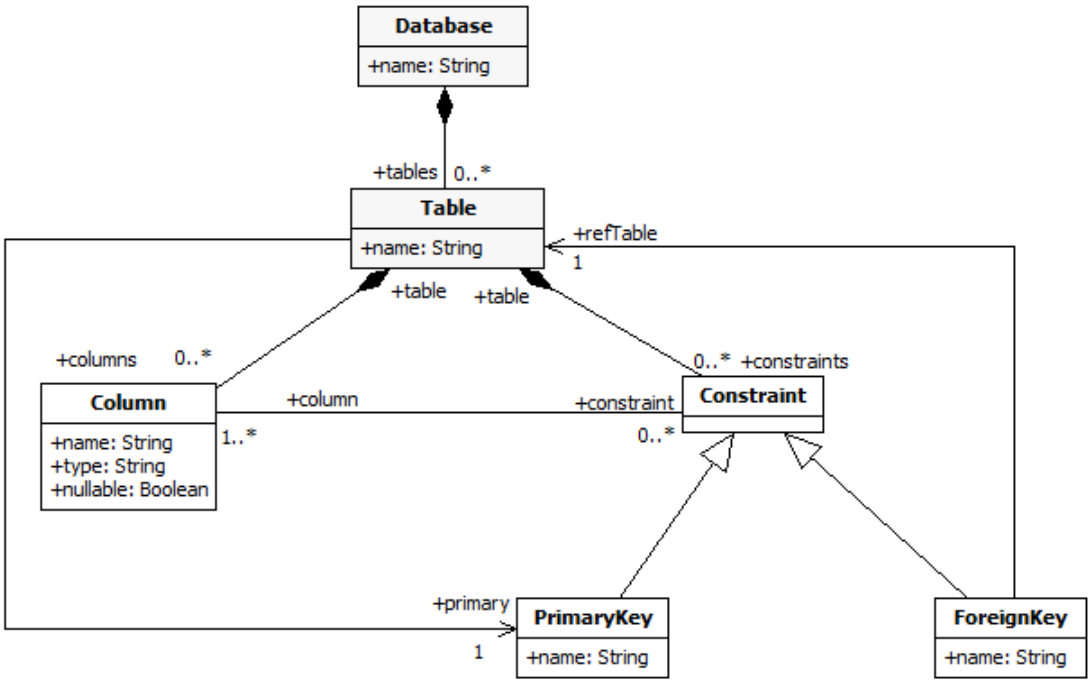


FIGURE 4.4 – Modèle structurel de base de données relationnelle

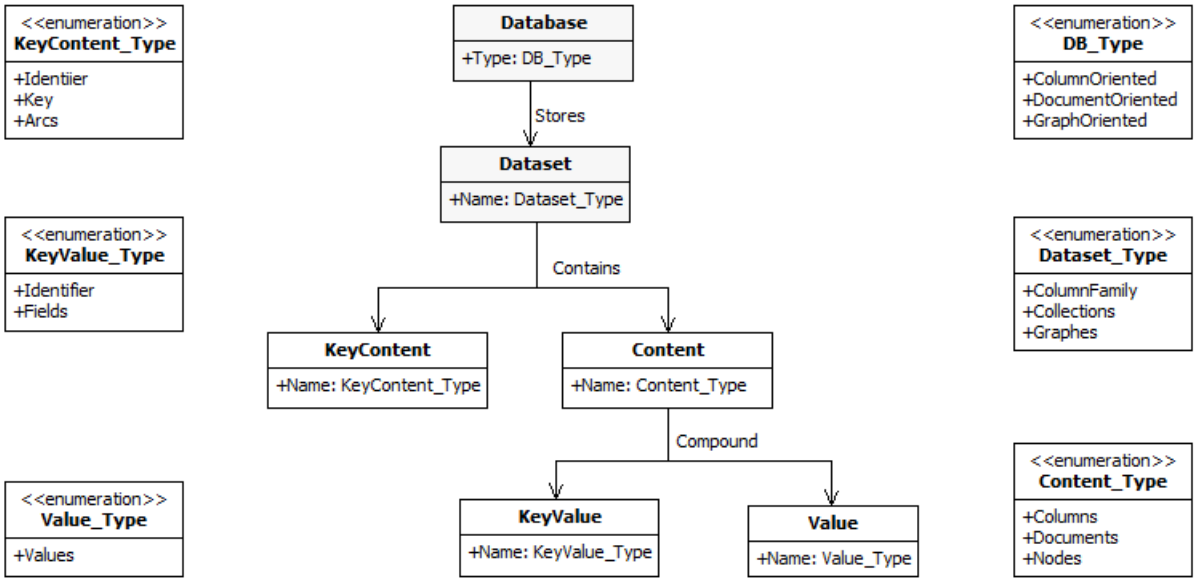


FIGURE 4.5 – Modèle structurel de base de données NoSQL

Les requêtes représentent l'élément indispensable dans le processus de test des bases de données. Notre objectif est comment extraire, d'un test effectué, la description des données utilisées et non pas les données elles-même. En conséquence, nous avons opté pour une modélisation descriptive au lieu de structurelle. Nous nous sommes inspiré des travaux du projet

PLASTIC [84] qui ont proposé une représentation des requêtes à l'aide d'un vecteur de fonctionnalités. Chaque vecteur contient des attributs structuraux et d'autres statistiques liés aux requêtes et aux tables. Les attributs structuraux représentent l'information liée aux requêtes et leurs schémas de méta-données associés.

- Degré de table (DT : Degree of Table) : c'est le nombre de jointures dans lesquelles une table est impliquée.
- Degré de séquence (DS : Degree Sequence of query) : il est dérivé de l'attribut Degré de Table (DT). Il contient tous les degrés des tables utilisées. Il est sous forme de vecteur non-croissant.
- Nombre de prédicats de jointures d'index (JIC : Join predicate Index Counts) : c'est le nombre de prédicats de jointures impliquant 0, 1 ou 2 index respectivement.
- Nombre de prédicats de table (PC : Predicate Counts of a table) : les prédicats peuvent être SARGables ou NON-SARGables. La différence est que les premiers peuvent être évalués à travers les index, contrairement aux seconds, c-à-d la possibilité d'utiliser ou non une recherche dans un index [167]. Cet attribut stocke le nombre de prédicats SARGables. La raison de son inclusion est qu'un index ne peut être utilisé que si le prédicat est SARGable.
- Index Flag d'une table (IF) : c'est un attribut booléen associé à chaque table. Il prend la valeur "vrai" si tous les prédicats de sélection et de projection sur la table en question peuvent être évalués en utilisant un index. Cet attribut est très significatif, car dans le cas où il est à "vrai", l'optimiseur lui associe un plan qui utilise uniquement l'index et non la table elle-même.

Les attributs statistiques sont les attributs associés à la base de données et qui sont nécessaires pour le modèle de coût. Ils sont disponibles dans le catalogue des systèmes.

- Taille de table (TS : Table Size) : c'est la taille de la table, qui n'est autre que le produit du nombre de tuples présents dans la table par leur taille.
- Taille effective de table (ETS : Effective Table Size) : c'est la taille effective de chaque table impliquée dans une opération de jointure. Cette taille représente l'impact des prédicats de sélection et de projection sur cette table.

En pratique les vecteurs de requêtes sont stockés dans la base de données de graphe de requêtes sous forme de plusieurs vecteurs, dont l'un contient des attributs globaux relatifs à la requête, et les autres contiennent des attributs locaux relatifs à chaque table impliquée dans la requête. Ces attributs sont résumés ci-dessous :

- Attributs de table :
  - DT_i : nombre de prédicats de jointure dans lesquels une table particulière est impliqué ;
  - $JIC[0..2]_i$: nombre de prédicats de jointure de la caractéristique d'index 0, 1 et 2 impliquant T_i ;
  - IF_i : indicateur booléen (accès avec index uniquement à la table T_i) ;
  - $PCSarg_i$: nombre de prédicats SARGables de la table T_i ;
  - $PCNSarg_i$: nombre de prédicats non-SARGable de la table T_i ;
  - TS_i : taille totale de table ;
  - ETS_i : taille effective de chaque table participant à une jointure.
- Attributs de requête :
  - NT (Number of Tables) : nombre de tables impliquées dans la requête ;
  - NJP (total Number of Join Predicates) : nombre de prédicats de jointure de la caractéristique de requête 0, 1 et 2 impliquant T_i ;
  - DS : vecteur de DT_i de toutes les tables impliquées dans la requête.

Nous proposons un modèle illustré dans la figure 4.6 : le modèle de description de schéma et de requête. Ce modèle est basé sur le vecteur des caractéristiques des requêtes. Avec ce genre de modèle on peut décrire les tables et les requêtes des bancs d'essai comme SSB.

Pour comprendre la notion de vecteur de fonctionnalités de requête, nous présentons dans la figure 4.7 un exemple de schéma S et deux requêtes Qa et Qb. En outre, nous montrons l'instanciation de certains attributs. Le tableau de la figure 4.7 montre que quatre tables du schéma S sont impliquées dans la requête Qa avec un nombre de jointure NJP égal à 3. Il montre également que le vecteur de jointure des tables (T1, T2, T3 ,T4) est égal à (2, 1, 2, 1) d'où le graphe (b) de la figure4.7 .

3.4 SGBD et Plateformes

Parmi les composantes de l'environnement de test que nous avons identifiées on trouve les composantes relatives à toute la structure d'exécution. Cette dernière consiste en la fusion des trois couches de paramètres relatifs au SGBD, à la plateforme logicielle (Operating system) et la plateforme matérielle (voir la figure 4.8).

En essayant de modéliser chacune des trois composantes par décrire leurs univers, nous nous sommes rendu compte de deux spécificités qui caractérisent ces trois composantes. La première spécificité c'est que les paramètres des trois composantes sont souvent statiques dans un groupe

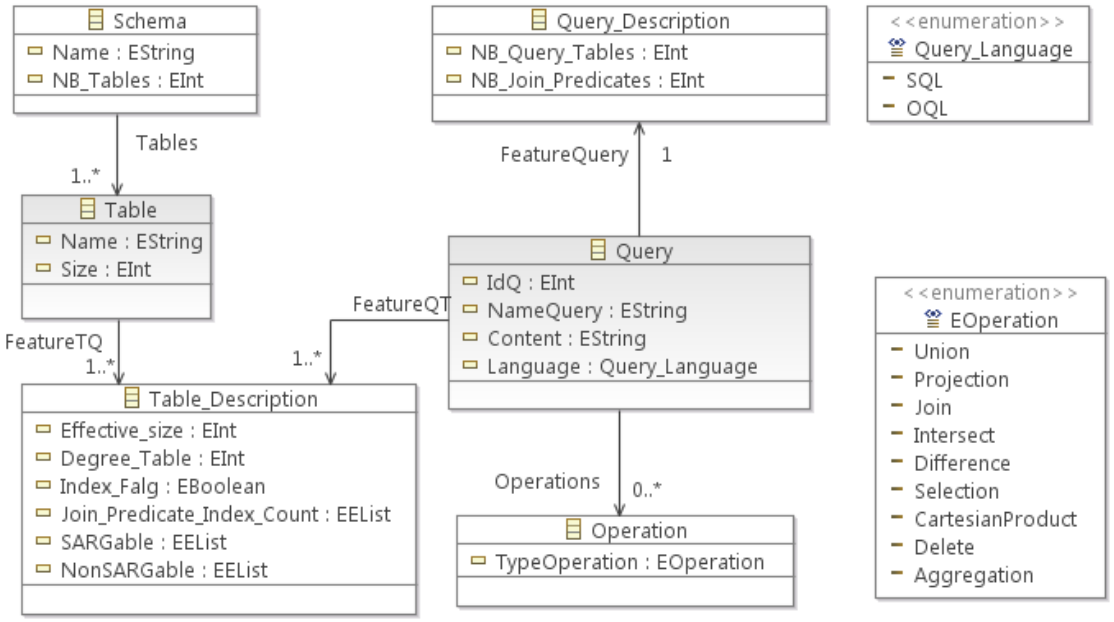


FIGURE 4.6 – Extrait du modèle de l'environnement de test mettant l'accent sur les composantes `DataSet` (schéma) et `Query`

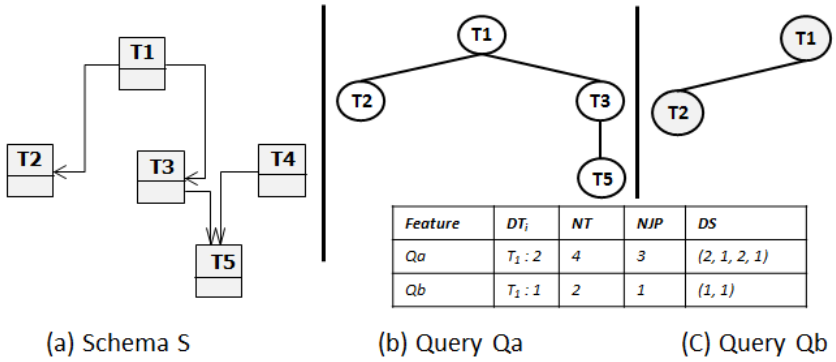


FIGURE 4.7 – Modèle de schéma S et graphe de requêtes Qa et Qb

de test. Autrement dit, en analysant des articles scientifiques présentant les tests, nous avons remarqué que d'un test à l'autre les composantes "SGBD et plateformes" sont moins variées par rapport aux autres composantes. La deuxième spécificité c'est que chaque configuration de mapping entre les trois composantes représentent un système de base de données. En outre, les paramètres des trois composantes sont interdépendants.

Pour toutes ces raisons, notre processus de conception au niveau de l'explicitation de composantes de l'environnement de test va se poursuivre en utilisant une modélisation à base ontologique. La conception d'une ontologie pour décrire en détails et de manière explicite les systèmes de base de données nous permettra de fournir un vocabulaire standard pour intégrer les données issues de différentes sources.

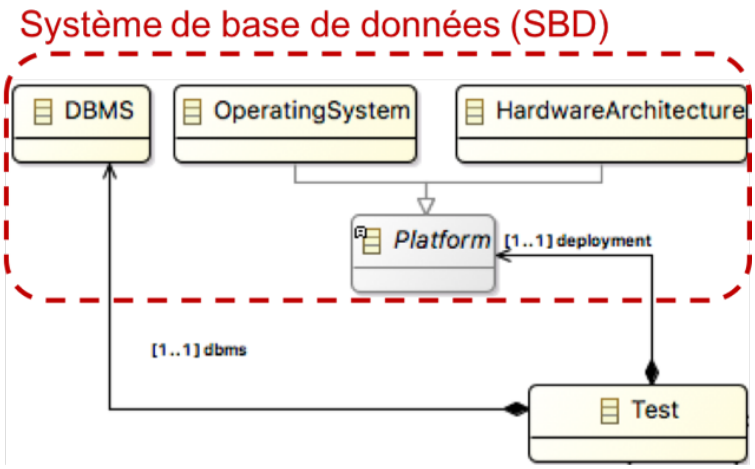


FIGURE 4.8 – Extrait du modèle de l'environnement de test mettant l'accent sur les trois composantes d'un SBD

4 Ontologie pour les des systèmes de base de données (SBD)

La nouvelle tendance des bases de données a conduit vers l'apparition d'une diversité des SGBD et de leurs fonctionnalités, et également à l'apparition des équipements informatiques et des plateformes. Compte tenu de cette diversité, le processus de sélection des meilleurs SGBD ou des plateformes est complexe. Face à cette complexité, nous proposons dans une première étape de concevoir une ontologie des systèmes de base de données (SBD). Cette ontologie couvre toutes les trois composantes des SBD en l'occurrence les SGBD, les systèmes d'exploitation et les plateformes matérielles.

A l'exception de l'ontologie des supports de stockage qui a été faite dans notre laboratoire par *Bellatreche et al.* [25], l'exploration des autres ontologies existantes ne nous a pas permis de trouver des modèles qui décrivent finement les concepts des SBD. Cette absence a motivé notre démarche de conception d'une ontologie pour intégrer tous les concepts des SBD.

4.1 Définition d'une ontologie

En 1993, Gruber [91] propose la définition suivante : une ontologie est une spécification explicite d'une conceptualisation d'un domaine. Dans [149], G. Pierra a définit l'ontologie comme une représentation formelle, explicite, référençable et consensuelle de l'ensemble des concepts partagés d'un domaine sous forme de classes, de propriétés et de relations qui les lient. Dans cette définition nous trouvons les termes clés des ontologies [107] :

- Formelle : une ontologie est une conceptualisation définie dans un langage traitable par machine et basée sur une théorie formelle qui permet de vérifier un certain niveau de cohérence.

- Explicite : l'ensemble des concepts et propriétés d'une ontologie sont spécifiés explicitement indépendamment d'un point de vue particulier ou d'un contexte implicite.

- Référençable : signifie que chaque concept de l'ontologie est référencé de manière unique à l'aide d'un identifiant afin d'expliciter la sémantique de l'élément référencé.

- Consensuelle : signifie que l'ontologie est admise et acceptée par l'ensemble des acteurs d'une communauté qui représentent un nombre plus important que celui des membres de conception d'une application particulière.

Malgré la diversité des modèles d'ontologies, ils comportent tous les mêmes notions :

□ Le concept est défini comme une entité composée de trois éléments distincts :

- (i) le terme exprimant le concept en langage naturel ;
- (ii) la notion ou l'intention du concept : la signification du concept ;
- (iii) les objets dénotés par le concept, appelés également réalisations ou extensions du concept. Les concepts d'une ontologie peuvent être classés en deux catégories [91] :
  - (a) Les concepts primitifs ou canoniques : ce sont les concepts de base à partir desquels d'autres concepts de l'ontologie peuvent être définis, c'est à dire qui ne peuvent pas être dérivés d'autres concepts.
  - (b) Les concepts définis : ce sont les concepts décrits par une définition complète à travers des conditions nécessaires et suffisantes exprimées en termes d'autres concepts.

□ Les classes décrivent les concepts d'un domaine. Une classe peut avoir des sous-classes qui représentent des concepts plus spécifiques que la super classe. Une classe peut avoir des instances représentant les extensions du concept.

□ Les attributs décrivent les propriétés des classes et des instances de l'ontologie.

□ Les relations désignent les associations définies entre les concepts de l'ontologie, comme la relation de subsomption *is-a* ou est-un (sous classe d'une classe). Elles permettent de lier deux concepts en spécifiant le concept de départ de la relation et le concept d'arrivée. Des propriétés peuvent être ajoutées à la relation, telles que la transitivité et la symétrie.

□ Les axiomes désignent les assertions acceptées comme vraies dans le domaine étudié. Les axiomes et les règles permettent de vérifier la cohérence d'une ontologie.

4.2 Différents types d'ontologies

Les applications des ontologies sont diverses dans le monde du développement (comme dans le Web sémantique). En effet, différents types d'ontologies existent.

- Ontologie globale : l'ontologie globale est dédiée à l'utilisation générale avec une représentation de plus haut niveau d'abstraction d'une manière formelle. Ce type contribue à organiser les notions des collaborateurs générales et de partager les connaissances concernant un projet. Ce type d'ontologie est beaucoup utilisé dans la conception des ontologies spécifiques telles que l'ontologie géographique GeoNames [195].

- Ontologie du domaine : l'ontologie du domaine est construite à partir des scénarios d'entreprises pour représenter un domaine spécifique tel que la santé, la finance, le social,etc. sous forme de base de connaissances pour lesquels elle sera utilisée. Il existe plusieurs exemple d'ontologies de domaine à savoir : TOVE, une ontologie pour l'intégration d'entreprise [82] et Ménélas, une ontologie des maladies coronariennes [30].

- Ontologie d'application : une ontologie d'application décrit une représentation de très haut niveau de spécification d'un sujet d'application pour un domaine quelconque. Ce type d'ontologie permet de définir les notions à appréhender, de les relier entre elles et d'indexer les ressources décrivant ces notions en tenant compte du contexte du champ d'étude considéré. A titre d'exemple, on peut citer l'ontologie d'application de l'indexation des ressources d'une formation en statistique [50].

4.3 Langages de représentation des ontologies

Pour définir une ontologie, plusieurs formalismes sont disponibles : (i) *Resource Description Framework (RDF)* [118], (ii) *RDF Schema (RDFS)* [41], (iii) *DARPA Agent Markup Language+Ontology Inference Layer or Ontology Interchange Language (DAML+OIL)* [105], (iv) *Ontology Web Language (OWL)* [191] et (v) *Parts LIBrary (PLIB)* [148].

4.3.1 Formalisme RDFS

Le Ressource Description Framework (RDF) est une recommandation du W3C pour la formulation de méta-données sur le World Wide Web [131]. Le schéma RDF (RDFS) [42] étend cette norme avec les moyens de spécifier le vocabulaire du domaine et les structures d'objets. Ces techniques permettront d'enrichir l'enchaînement du Web avec une sémantique transformable par machine, donnant ainsi naissance à ce qu'on a appelé le Web sémantique.

30. https://bioportal.bioontology.org/ontologies/TOP-MENELAS

4.3.2 Formalisme DAML+OIL

DAML+OIL est un langage fusionnant les caractéristiques des deux langages : DARPA Agent Markup Language (DAML) du ministère de la défense américain [135], et Ontologie Inférence Layer (OIL) développé par la communauté de recherche européenne [80]. DAML+OIL suit une approche orientée objet décrivant la structure en termes de classes et de propriétés. DAML+OIL s'appuie sur des standards antérieurs du W3C tels que RDF et RDFS, en étendant ces langages avec des primitives de modélisation plus riches.

4.3.3 Formalisme OWL

OWL est un langage pour la définition d'ontologies sur le Web. OWL décrit un domaine en termes de classes, de propriétés et d'individus et peut inclure de riches descriptions des caractéristiques de ces objets [18]. OWL est destiné à être utilisé lorsque les informations contenues dans les documents doivent être traitées par les applications, par opposition aux situations où le contenu doit seulement être présenté aux humains. OWL a plus de facilités pour exprimer la signification et la sémantique que XML, RDF et RDFS, et de capacité par rapport à ces formalismes pour représenter le contenu interprétable par machine sur le Web.

4.3.4 Formalisme PLIB

Le modèle d'ontologie PLIB a été conçu initialement pour l'échange et l'intégration automatique de composants de base de données et leurs instances [150]. PLIB permettent la description de classes, de propriétés, de domaines de valeurs, d'instances d'objets et de sources d'information. Une classe est une collection d'objets crée pour définir le domaine de certaines propriétés. Une propriété est une relation binaire entre deux classes ou une classe et un domaine de valeurs où elle n'a de sens que pour cette classe et ses différentes sous-classes. Un domaine de valeurs est un ensemble défini en extension ou en intention. Une instance représente un objet appartenant à une classe [8]. PLIB permet également de créer des ontologies multi-lingues où chaque entrée est associée à un identificateur unique.

4.4 Étapes de conception des ontologies

Diverses communautés développent des ontologies afin de spécifier les concepts et les relations de leurs domaines. Différentes approches sont apparues. Nous distinguons deux types d'approches : (i) le premier type est basé sur l'extraction des informations à partir de l'existant relatif aux documents et (ii) le deuxième type est basé sur la création d'ontologies à partir de rien en utilisant des informations implicites du champ d'étude. Les approches du premier type sont dédiées à la conception des ontologies globales et disposent d'un ensemble d'étapes : initialisation, spécification, conceptualisation, implémentation, test, exploitation. L'ingénierie

ontologique concernée par la conception des ontologies offre des outils pour assister le développeur durant le processus de conception d'ontologies.

- **Initialisation :** l'idée d'avoir une ontologie pour un domaine quelconque revient principalement aux besoins communs des usagers de ce domaine, par exemple créer une ontologie des équipements informatiques peut servir comme un modèle commun pour tous les producteurs.
- **Spécification :** cette étape consiste à définir le domaine de l'ontologie par rapport aux besoins exprimés par les collaborateurs, et à délimiter le périmètre du domaine en décrivant ses éléments. Dans cette étape, nous étudions les solutions et les ontologies déjà construites afin d'extraire tous les éléments, d'envisager de les ré-utiliser si nécessaire et de choisir le formalisme le plus adéquat aux besoins.
- **Conceptualisation :** cette étape consiste à décrire de façon abstraite chaque concept collaborant à répondre aux besoins et à couvrir tout le périmètre. Le but de cette étape est de conceptualiser tous les concepts ontologiques.
- **Implémentation :** cette étape consiste à implémenter tous les concepts sous le formalisme retenu lors de la phase de spécification. Dans cette étape, on peut faire appel à un éditeur d'ontologies, tel que Protégé.
- **Test :** cette phase consiste à mettre des scénarios de tests afin d'évaluer l'implémentation faite dans l'étape précédente.
- **Exploitation :** cette étape consiste à mettre en œuvre l'ontologie construite en respectant les étapes précédentes. L'exploitation d'une ontologie comprend aussi le travail de diffusion de l'ontologie pour amener de nouveaux acteurs du domaine à l'utiliser.

Le deuxième type est plus souple par rapport au premier type, il est porté sur les étapes de la méthode globale décrite précédemment. Le but est de créer des ontologies locales, spécifiques à des sources ou des usages. Toutefois ces ontologies locales partagent un même vocabulaire ce qui génèrent des concepts redondants dans les différentes ontologies. Cette redondance nous oblige à définir les correspondances entre les concepts nécessaires pour chaque ontologie.

4.5 SPARQL : Langage d'interrogation

Le langage SPARQL [31] est une recommandation W3C largement utilisée pour rechercher des données RDF. Il a pris sa place comme une technologie primordiale dans le Web sémantique. SPARQL a été influencé par la spécification du format RDF pour définir sa syntaxe et sa sémantique. C'est ainsi que les requêtes SPARQL, tout comme les données RDF, sont décrites

31. https://www.w3.org/TR/rdf-sparql-query/

par des triplets. SPARQL prend également en charge l'agrégation, les sous-requêtes, la négation, la création de valeurs par des expressions. Dans Le langage SPARQL, nous distinguons quatre formes de requête [32] : (i) ASK : elle permet de vérifier la présence ou non d'au moins d'une solution pour un ensemble de contraintes en renvoyant un booléen, (ii) DESCRIBE : elle consiste à retourner les informations décrivant les résultats et vérifiant les contraintes spécifiées sous forme d'un graphe, (iii) CONSTRUCT : elle permet de retourner les résultats des requêtes sous forme de graphe RDF défini à travers des templates, et (iv) SELECT : cette forme est similaire aux requêtes SQL et permet d'extraire les résultats des requêtes qui vérifient l'ensemble de contraintes.

4.6 Présentation de l'ontologie des SBD

Pour atteindre notre objectif, nous avons revu et recensé l'ensemble des concepts associés aux SBD. Cette phase nous permet dans un premier temps de classer les produits informatiques en trois groupes. Cette classification est fondée sur une première analyse des fonctionnalités des produits.

- commun à tous les produits : prix, popularité, commentaires, documentation, site web.
- commun à tous les produits logiciels : licence (propriétaire ou open source), les systèmes d'exploitation (Windows, Linux, Unix), etc.
- commun et spécifique aux SGBD : modèle de $\mathcal{BD}$ (relationnel, orientée objet, orientée valeur-clé, orienté document, orientée colonne, orienté graphe , XML, événement, RDF, etc. [33]), déploiement (centralisé, nuage, etc.), langages de programmation prises en charge (Java, C, C ++, Perl, etc.), langage de requête (SQL, XQuery, SPARQL, XPath, etc.) [154], les techniques de manipulation de données et méthodes d'accès [109].

Dans notre domaine d'étude, nous nous concentrons sur les fonctionnalités des SGBD et leurs multiplicités. Comme dit précédemment, l'indisponibilité d'une classification pour les SGBD et les fonctionnalités associées était la motivation pour faire un pas vers la conception du modèle proposé. L'intégration des fonctionnalités du système dans le même modèle facilite la recherche d'un système cible pour un besoin spécifique, et extraire une classification des systèmes selon les critères exprimés. Notre proposition est de concevoir un modèle des SBD, principalement basée sur des recherches antérieures sur les SGBD (FAME-DBMS) dans [162] et cellulaires SGBD dans [185], en particulier dans les caractéristiques de décomposition et de modélisation. Nous bénéficions également de l'ontologie de domaine grâce à sa facilité de représentation, d'interrogation et de raisonnement.

32. https://www.w3.org/TR/rdf-sparql-query/
33. http://db-engines.com/en/ranking

L'objectif de la modélisation des SGBD est de comprendre sa structure et son fonctionnement et de les représenter en des concepts clés, des attributs et des instances liées au domaine. En utilisant en particulier le standard OWL, on peut aller bien au-delà des relations hiérarchiques entre classes en représentant des relations dynamiques, difficiles à exprimer facilement en SQL ou UML.

Notre étude et analyse effectuées sur les SBD nous a permis de définir l'ensemble des concepts, des relations qui les relient et des instances. Ce recensement nous a conduits à construire une ontologie en respectant les étapes du premier type de conception des ontologies qui est basé sur l'existant relatif aux documents visant à extraire de l'ensemble des informations lié à notre champ d'étude. C'est évident que la structure de notre ontologie s'appuie sur les trois principales parties des systèmes de $\mathcal{BD}$: le SGBD, la plateforme matérielle et le système d'exploitation. Pour cela, notre ontologie se divise en trois classes : DBMS, Operating_System et Hardware.
Pour la visualisation des ontologies, nous avons le choix entre deux manières : la représentation par une arborescence (comme dans un navigateur de fichier), ou par un graphe. L'arborescence offre une représentation qui permet de voir facilement les relations hiérarchiques entre les termes tandis que la représentation sous forme de graphe est plus adaptée pour la visualisation des autres types de relations. La figure 4.9 schématise une vue en graphe des trois principales classes de notre ontologie.

4.6.1 Composantes du modèle des SBD

Dans le but de réaliser un modèle spécial pour tous les systèmes de gestion de $\mathcal{BD}$, nous avons développé un modèle commun contenant toutes les fonctionnalités (communes ou non communes) réparties dans des catégories en fonction de leurs rôles dans la gestion des données. Notre modèle proposé se divise en trois classes : *DBMS, Operating_system, Hardware*.

- La disponibilité des systèmes d'exploitation avec les caractéristiques inhérentes est un facteur très important pour les utilisateurs afin d'assurer une intégration facile de nouveaux SGBD avec l'ensemble du système d'information.

- Le système d'exploitation a une forte interaction avec la partie représentant le matériel (*Hardware*), la fréquence et la taille de la mémoire vive. La vitesse d'horloge et les noyaux de processeur, en plus de la capacité et la vitesse du disque dur, sont des caractéristiques importantes.

- *DBMS* est le troisième composant, il est considéré comme le noyau du système de base de données (SBD). Nous allons le présenter dans la suite de ce chapitre.

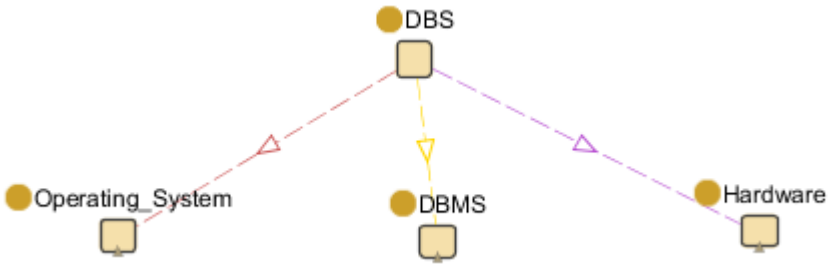


FIGURE 4.9 – Vue en graphe des trois classes de l'ontologie des systèmes de $\mathcal{BD}$ (SBD)

4.6.2 Classe des SGBD (DBMS)

Il ya quelques années, il y avait une douzaine de SGBD que les entreprises et les organisations éducatives partageaient pour développer des applications de $\mathcal{BD}$. Aujourd'hui, nous assistons à une prolifération de nouveaux SGBD développés pour répondre aux nouveaux besoins motivés par l'ère du Web et du Big Data.

De nombreux SGBD sont disponibles sur le marché : (i) des versions open source aux versions commerciales, (ii) de SGBD spécialement destinés aux professionnels aux SGBD pour les débutants et (iii) de SGBD spécifiques tels que les moteurs de recherche et l'indexation des données (comme Elastic search) aux SGBD à usage général (comme Oracle). Le point commun est que tous les systèmes disposent de multiples fonctionnalités et souvent le coût de fonctionnalités est très important. Cela revient parfois au prix des systèmes commerciaux et revient parfois au coût du traitement, du stockage, de la maintenance, de la formation, etc., en particulier pour les systèmes open source. Pour cela, nous nous focalisions sur le composant SGBD et ses fonctionnalités. Comme indiqué précédemment, l'indisponibilité d'une classification pour les modèles de SGBD et les caractéristiques associées était la motivation pour faire un pas vers la conception d'un modèle des SGBD en étudiant et en analysant leurs caractéristiques et leur fonctionnement notamment en ce qui concerne l'accès aux données, la manipulation, le traitement et le partage de données et le modèle de stockage (voir la figure 4.10). L'intégration des fonctions du système dans le même modèle facilite la recherche du système souhaité.

Le but est de concevoir un modèle commun dédié spécialement pour instancier tous les SGBD avec toutes leurs caractéristiques (communes ou non communes). En effet la classe DBMS est devisée en sous-classes selon leur rôle dans la gestion des données (voir la figure 4.10) :

- La première partie est liée aux techniques d'accès aux données décrivant les différentes structures d'optimisation à savoir les index, les vues matérialisées, la fragmentation verticale/horizontale, etc., et les méthodes de manipulation de données à savoir le langage de requêtes, les opérations de jointure, les opérations d'agrégation, etc.

- La deuxième partie est liée à la technologie logicielle qui décrit le partage des données,

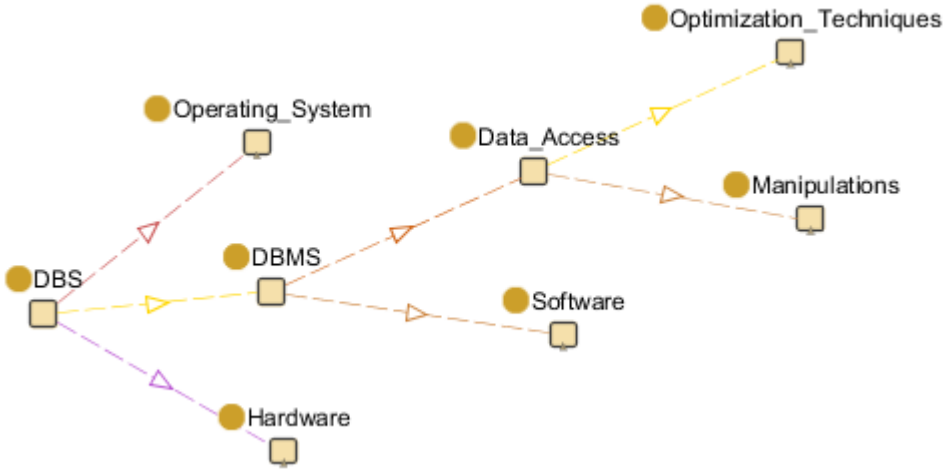


FIGURE 4.10 – Vue en graphe de la classe des SGBD

le cloud, la réplication et d'autres techniques.

4.6.2.1 Sous-classe accès aux données La sous-classe d'accès aux données a mis en évidence à la fois la manipulation des données et les techniques d'optimisation. La manipulation des données (voir la figure 4.11) présente la fonctionnalité des langages de requête prises en charge par le SGBD pour créer et manipuler des $\mathcal{BD}$, à savoir *Select-From-Where* ou l'équivalent dans d'autres langues. Le regroupement, la transaction et l'agrégation (sum, count, avg, min, max) représentent les besoins ordinaires de requêtes. Les procédures stockées, les déclencheurs, le traitement temporel et flux spatial représentent des besoins spécifiques de requêtes [160].

Dans l'autre côté, il existe de larges structures d'optimisation prises en charge par le SGBD qui peuvent améliorer la performance des requêtes. La sélection des structures d'optimisation appropriées pour une charge de requêtes donnée est un problème NP difficile [27]. Plusieurs algorithmes et approches ont été proposés pour traiter ce problème. Ils peuvent être divisés en deux catégories [27] en fonction de leurs ressources utilisées.

- La première catégorie représente les techniques redondantes où nous pouvons citer la fragmentation verticale, les vues matérialisées, les index et la gestion du *Buffer*. Dans toutes ces techniques, le coût de stockage supplémentaire est considéré pour mettre en œuvre les structures des données existantes, et le coût de maintenance supplémentaire pour mettre à jour les données.

- La deuxième catégorie représente les techniques non redondantes à savoir la fragmentation horizontale, le traitement parallèle et l'ordonnancement des requêtes. A la différence de la première classe, elles ne nécessitent pas de coûts de stockage supplémentaire, car

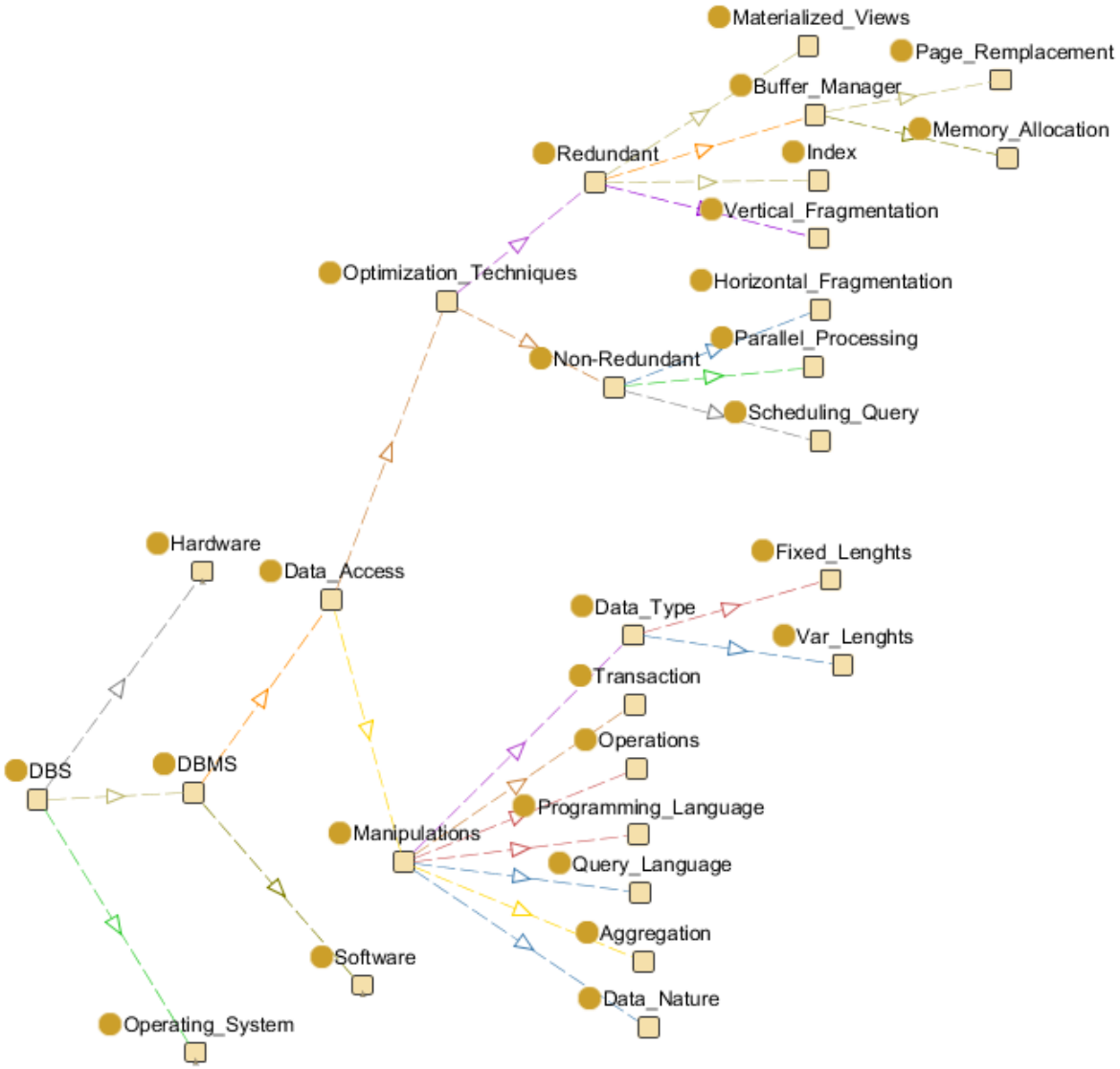


FIGURE 4.11 – Vue en graphe de la sous-classe Data Access

elles se rapportent à la distribution des données ou des requêtes.

4.6.2.2 Sous-classe software La sous-classe software représente le modèle de stockage (relationnel, clé-valeur, orienté document, orienté colonne, orienté graphe, etc.). Cela comprend la structure de la $\mathcal{BD}$, les tables dans le cas des SGBD relationnels, les collections dans le cas des SGBD orienté documents, etc., la définition des éléments d'information (champs) stockés, de leurs types de données, des règles de validation et des propriétés. Il comprend également la définition des clés primaires (le champ unique dans chaque $\mathcal{BD}$), les clés étrangères (qui seront utilisées pour relier les tables, les collections) et les clés secondaires (ces champs seront utilisés pour configurer des index de recherche supplémentaires pour accélérer la recherche). Elle présente ainsi la gestion des groupes et utilisateurs, droits d'accès, les techniques de sauvegarde automatique et de restauration de données. Elle permet aussi de fournir des détails concernant le stockage des données (système de fichiers, mémoire), les systèmes d'exploitation pris en charge (Windows, Unix, Linux), les languages de programmation supportés, les modes de réplication (synchrone asymétrique, synchrone symétrique, asynchrone asymétrique et asynchrone symétrique), les concepts de transaction (ACID, BASE), l'architecture (centralisée, distribuée, cloud, etc.) (voir la figure 4.12).

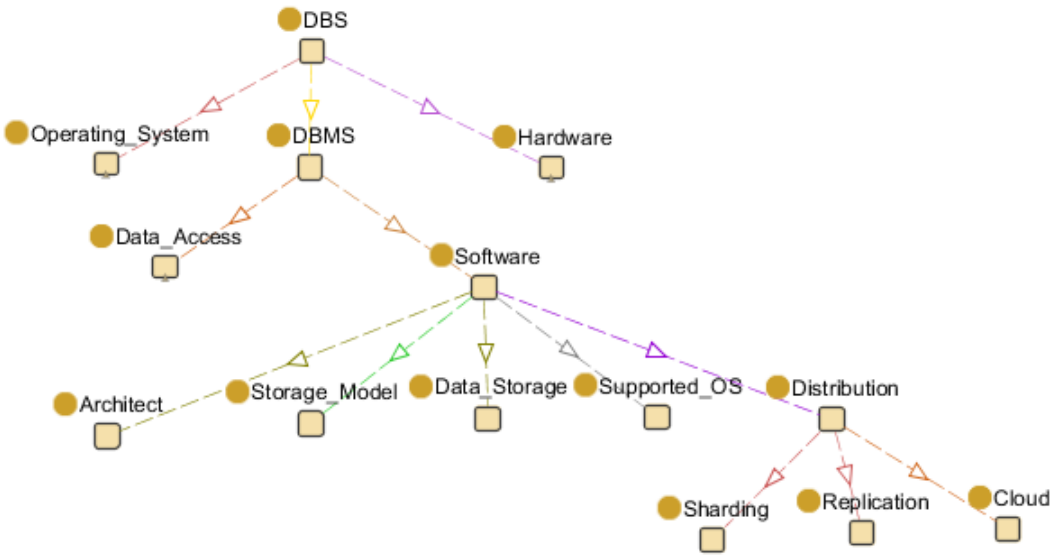


FIGURE 4.12 – Vue en graphe de la sous-classe Software

4.6.3 Classe du matériel (Hardware)

La composante matérielle représente une partie importante dans les systèmes de $\mathcal{BD}$. Dans cette classe, nous avons décrit la configuration matérielle à savoir les machines utilisées et leurs caractéristiques. Comme nous le savons, chaque machine informatique dispose des supports de stockage, des unités de traitement et des mémoires. Les supports de stockage servant à stocker

les données sont multiples comme le disque dur, SSD (mémoire flash), les clés USB, les cartes mémoires et les disques hybrides. De plus, chaque support de stockage est qualifié par un ensemble de caractéristiques. Nous citons entre autres la capacité de stockage, la vitesse d'accès, la vitesse de rotation des plateaux et le taux de transfert (en lecture et en écriture). Aussi, chaque unité de traitement (processeur) est caractérisée par son type (CPU, GPU, etc.), sa microarchitecture (nombre de pipelines, nombre de mémoires cache, etc.), son horloge (exprimée en MHz ou GHz), sa finesse de gravure exprimée en nanomètres (nm) et son nombre de cœrs de calcul. La dernière composante est la mémoire qui représente un élément essentiel dans les machines. Elle est caractérisée par : son type (DDR, DDR2, DDR3), sa taille, sa fréquence, son timing (nombre de cycles d'horloge pour effectuer une opération). La figure 4.13 résume les éléments principaux de la classe Hardware.

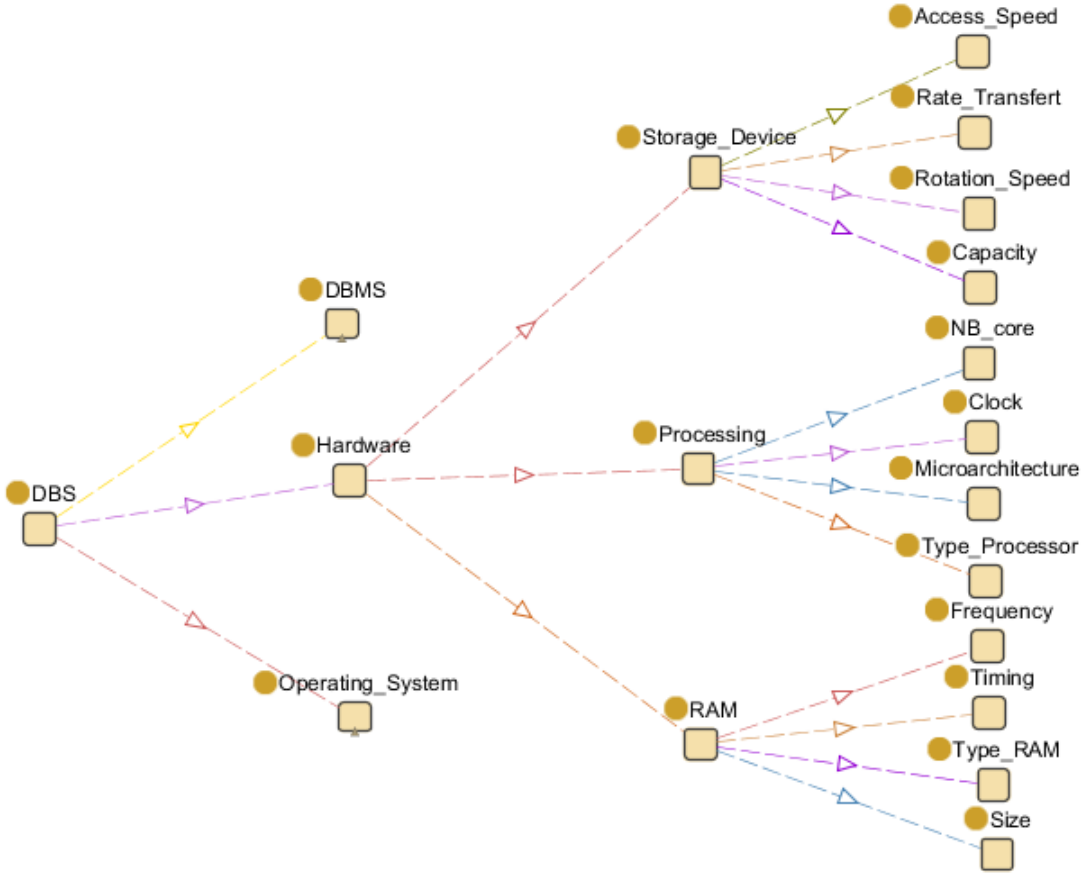


FIGURE 4.13 – Vue en graphe de la classe Hardware

4.6.4 Classe des systèmes d'exploitation (Operating_System)

Le système d'exploitation est un élément crucial dans les systèmes de base de données. Son rôle est de permettre aux utilisateurs d'accéder aux machines en exploitant ses ressources matérielles. En rapport avec les ressources, le système d'exploitation offre une suite de services

généraux facilitant la création et l'utilisation de logiciels applicatifs. A cet effet, les systèmes d'exploitation possèdent un ensemble d'applications : (i) le noyau offre des mécanismes pour la gestion de la mémoire, des processus, des fichiers, des entrées-sorties principales, et des fonctionnalités de communication, (ii) l'interpréteur de commande fournit un langage de commandes permettant la communication avec le système et (iii) le système de fichiers présente à l'utilisateur une vue abstraite sur ses données et permet de les localiser à partir d'un chemin d'accès (arborescence). En général, nous pouvons distinguer deux familles de systèmes d'exploitation : les Windows et les UNIX. La gamme des systèmes d'exploitation Windows domine le marché des PCs avec ses différentes distributions à savoir Windows NT (Windows 2000, Windows XP, vista, Windows 7, Windows 8, Windows 10), Windows CE dédiés aux systèmes embarqués et Windows mobiles, etc. La gamme UNIX est le plus utilisé notamment dans les appareils mobiles et les supercalculateurs à travers ses branches à savoir la variante BSD (FreeBSD, NetBSD et OpenBSD), Linux et ses dérivés, Android et MacOS, etc.

4.6.5 Exemple d'usage de SPARQL

Notre ontologie consiste à faciliter la recherche des informations concernant les SBD. L'exemple le plus simple est de trouver la liste des SGBD supportant un ensemble de fonctionnalités exprimées par l'utilisateur. Nous interrogeons notre ontologie à travers des requêtes SPARQL, le langage standard d'interrogation des ontologies. La figure 4.14 montre un exemple de l'usage simple de notre ontologie où l'utilisateur cherche la liste des SGBD qui supportent le langage de programmation "Matlab".

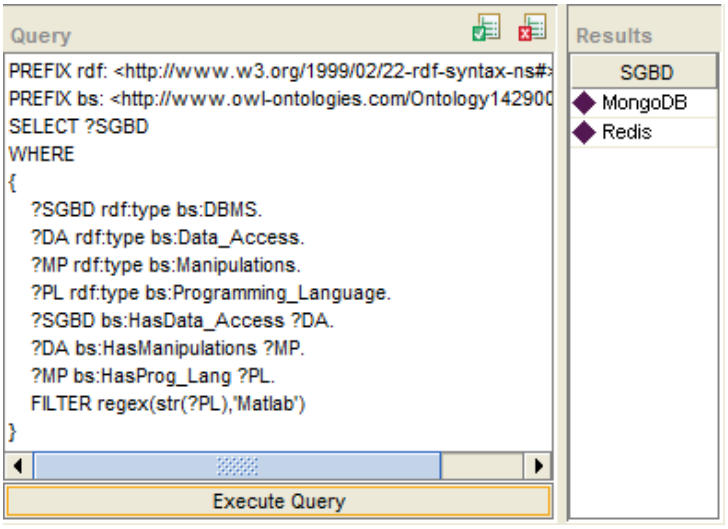


FIGURE 4.14 – Liste des SGBD supportant le langage de programmation "Matlab"

Dans le chapitre 6, nous allons expliquer comment construire cette requête en prenant en considération deux éléments. (i) le premier c'est que les utilisateurs ont souvent une connaissance partielle du contenu, de la structure, de la sémantique et des hypothèses sous-jacentes

d'une ontologie. (ii) Le deuxième élément c'est que les ontologies sont souvent incomplètes, malgré leur taille très importante, par exemple YAGO [175] décrit plus de 10 millions d'entités et 120 millions de faits sur ces entités.

5 Conclusion

Dans ce chapitre, nous avons abordé un problème très pertinent dans le domaine des $\mathcal{BD}$, il s'agit de l'obscurité des composantes qui caractérisent un environnement de test. Nous avons analysé les éléments qui ont un impact direct sur les résultat des tests afin de les expliciter. Ensuite nous avons fournit une représentation de l'environnement de test à l'aide d'une formalisation détaillée. Nous avons également détaillé chaque composante en utilisant deux formalismes : les modèles de conception et les ontologies. L'utilisation des ontologies a été motivée par la difficulté de séparer certaines composantes de l'environnement de test qui forment également les SBD. Nous avons donc présenté la conception de notre ontologie des SBD pour décrire de manière consensuelle tous les concepts et propriétés d'un système de base de données. Ensuite nous avons présenté la structure de notre ontologie en s'appuyant sur les trois principaux éléments des SBD : le SGBD, la plateforme matérielle et le système d'exploitation représentant respectivement les classes DBMS, Hardware, Operating_System. Nous avons également bien détaillé les classes et les sous classes de la composante "DBMS" décrivant les fonctionnalités des SGBD.

Nous avons aussi montré les fortes relations entre les différentes fonctionnalités des classes principales. Cette dépendance dégage d'autres perspectives liée à la capacité de notre ontologie en terme de raisonnement. Notez que les ontologies ont largement contribué à la définition des systèmes recommandés [136]. Le but est d'étudier la possibilité de remplacer un SGBD par d'autres offrant des fonctionnalités équivalentes dans des cas particuliers.

Dans le chapitre suivant, nous allons exploiter les résultats de ce chapitre pour présenter une approche multidimensionnelle. Cette dernière se base essentiellement sur un dépôt de tests qui sera également présenté.

Chapitre 5

Approche multidimensionnelle des données des environnements de tests

Sommaire

1 Introduction 115

2 Entrepôt de données 115

2.1 Définition d'un entrepôt de données 116

2.2 Magasin de données (datamart) 116

2.3 Architecture des entrepôts de données 117

2.4 Modèle de données 118

2.4.1 Schéma en étoile 118

2.4.2 Schéma en flocon 118

2.5 Types d'entrepôts de données 120

2.6 Traitement analytique en ligne 120

2.6.1 Cube de données 121

2.6.2 Opérations OLAP pour l'interrogation de l'entrepôt de données 122

3 Notre entrepôt de données de tests non fonctionnels 124

3.1 Développement de l'entrepôt de données 124

3.2 Interface d'insertion et de sélection de tests 127

3.2.1 Manifeste un moyen pour définir les exigences du déployeur 128

3.2.2 Interaction entre le manifeste et l'entrepôt 128

3.2.3 Insertion des tests/instances du dépôt 129

3.2.4 Recherche des tests 130

3.3 Exemples d'usage de notre Entrepôt 131

4 Conclusion 132

Résumé. L'explicitation de l'ensemble de dimensions de l'environnement de tests non fonctionnels nous a motivés de les traduire vers un véritable entrepôt de données dont ses dimensions représentent celles que nous avons explicitées dans le chapitre précédent. Par ailleurs sa table des faits décrit les données de tests. La présence d'un tel entrepôt permet aux *déployeurs*, chercheurs, entreprises, étudiants, etc. de l'explorer et de l'analyser pour extraire des tendances comme les bancs d'essai les plus utilisés, les laboratoires les plus importants en termes de génération de données de tests, etc. Dans ce chapitre, nous détaillons le processus de construction, de déploiement et d'exploitation de notre entrepôt de données. Des exemples d'usage de cet entrepôt sont également donnés.

1 Introduction

Dans le chapitre précédent, nous avons explicité les composantes de l'environnement de test pour pouvoir capitaliser les efforts en termes de travaux de tests non fonctionnels générés par la communauté de $\mathcal{BD}$. Ces données progressent à un rythme soutenu. Ces progrès couvrent toutes les phases de la conception d'applications de $\mathcal{BD}$. Cette mine d'information doit être stockée à des fins d'analyse. La technologie des entreposages de données a été proposée pour répondre à ces besoins. Cette technologie a fait le succès de plusieurs entreprises comme Coca Cola et Walmart. D'après le cabinet Gartner, le marché de l'informatique décisionnelle (business intelligence) augmente de 8,4% par an et atteindra un chiffre d'affaires mondial de 27 milliards de dollars en 2019 [66]. Cette technologie est associée à des outils de type OLAP qui repose sur une représentation des données proche des intuitions de l'analyste (dans notre cas le *déployeur*), en leur offrant des interfaces d'expression aisées de requêtes complexes. La construction d'un entrepôt de données passe par le cycle de vie que nous avons présenté dans le Chapitre 2. L'explicitation de l'ensemble d'entités que nous avons établie dans le chapitre précédent offre les dimensions de notre entrepôt de données et les résultats de tests représentent les mesures de l'entrepôt. Le chargement de notre entrepôt se fait via les données disponibles dans les articles scientifiques.

Ce chapitre débute par la présentation de l'ensemble de notions fondamentales liées aux entrepôts de données couvrant l'ensemble de ses phases (conceptuelle, logique, ETL, déploiement, physique et exploitation), une fois construit, des exemples d'usage de notre entrepôt sont présentés.

2 Entrepôt de données

La conception des $\mathcal{BD}$ opérationnelles vise à garantir des accès rapides et simultanés aux données tout en garantissant la cohérence. Elles sont basées sur le paradigme du traitement des transactions en ligne (*Online Transaction Processing-OLTP*). Ces $\mathcal{BD}$ sont conçues avec un degré élevé de normalisation en utilisant des dépendances fonctionnelles et des formes normales [78]. Les exigences analytiques imposées par les grandes entreprises [192] portent sur le besoin d'agréger un grand volume de données historiques provenant de nombreuses tables (à l'aide de jointures). Néanmoins, les $\mathcal{BD}$ opérationnelles ne supportent pas les requêtes analytiques car elles n'ont pas été conçues pour stocker les données historiques et les traiter de manière efficace. Ces limitations ont largement contribué à la naissance d'une nouvelle technologie prenant en charge le traitement analytique en ligne (*OnLine Analytical Processing-OLAP*) qui est l'entrepôt de données. Un entrepôt de données (*Data Warehouse-DW*) a pour but de soutenir la prise de décision. Il collecte des données à partir de diverses sources de données hétérogènes, autonomes, évolutives et distribuées [186], les transforme, les nettoie et les charge finalement dans de nouvelles structures de données conçues pour prendre en charge les requêtes OLAP. Ces

structures sont représentées par un hypercube, avec les dimensions correspondant à diverses perspectives d'affaires, et les cellules contenant les mesures à analyser. Ci-après, nous donnons un aperçu concernant un entrepôt de données. L'architecture classique, les langages d'interrogation, les étapes de conception et les opérations de manipulation d'un entrepôt de données sont aussi discutés.

2.1 Définition d'un entrepôt de données

Un entrepôt de données est un dépôt de données utilisé pour collecter, ordonner, journaliser et stocker des informations provenant de différentes sources à toutes fins utiles d'accès et d'analyse [13]. Le fondateur des entrepôt de données *William Inmon* a définit un entrepôt de données comme un ensemble de données orientées sur des sujets, intégrées, non volatiles, historisées, variables dans le temps, résumées et disponibles pour l'interrogation et la prise de décisions dans l'entreprise [106].

Les concepts de base de l'entrepôt de données sont :

- Orienté sur le sujet : cela signifie que l'entrepôt de données se concentre sur les besoins analytiques qui dépendent de la nature des activités réalisées par les entreprises (par exemple, gestion des stocks, recommandation sur les ventes de produits, etc.).
- Intégré : c-à-d que les données sont extraites de plusieurs sources organisationnelles et externes qui doivent être intégrées après les processus de nettoyage, l'intégration généralement effectuée par un processus d'extraction, de transformation et de chargement (*Extract Transform Load-ETL*).
- Non volatile : la durabilité des données est garantie en interdisant la modification et la suppression des données, élargissant ainsi la portée des données à une période plus longue que celle offerte habituellement par les systèmes d'exploitation.
- Variable dans le temps : la possibilité de conserver, pour la même information, différentes valeurs liées à ses changements (évolution).

2.2 Magasin de données (datamart)

Un magasin de données (*datamart*) représente un sous-ensemble d'un entrepôt de données fournissant des données souhaitées par les utilisateurs à travers une représentation verticale des données portant sur un métier particulier. D'un point de vue à court terme, un datamart peut remplacer un entrepôt de données sans investir beaucoup d'efforts mais dans une perspective à long terme, un datamart n'est jamais un substitut à un entrepôt de données [106]. Les structures de datamart sont généralement appelées structures multidimensionnelles et sont servies par la technologie OLAP.

2.3 Architecture des entrepôts de données

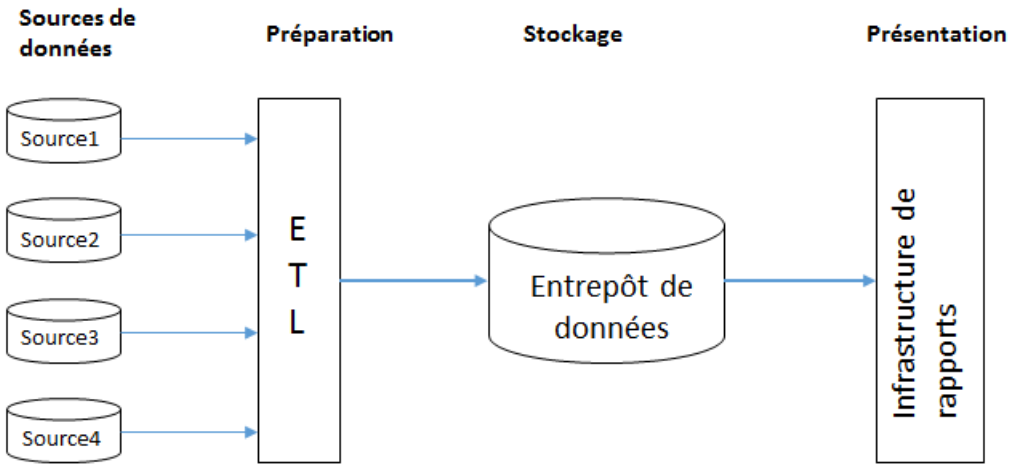


FIGURE 5.1 – Architecture des entrepôts de données

L'architecture d'un entrepôt de données s'appuie généralement sur quatre services (voir la figure 5.1) :

- **Sources de données** pour des besoins de gestion, les entreprises disposent de données stockées momentanément et qui représentent une masse d'information de grande valeur. Cette masse est enregistrée dans des $\mathcal{BD}$ opérationnelles, des fichiers XML, etc. Les sources de données peuvent être internes (produites à l'intérieur des entreprises) ou externes (sur le Web) ;

- **Service de préparation** en utilisant les outils ETL qui sont des logiciels responsables de l'extraction de données provenant de plusieurs sources d'information hétérogènes d'entreprise, de leur nettoyage, de leur personnalisation et de leur insertion dans un entrepôt de données cible [170]. Cette phase joue un rôle crucial dans le processus de la conception des entrepôts de données. La qualité de l'entrepôt dépend fortement de l'ETL. Plusieurs outils commerciaux et académiques d'ETL sont également disponibles : Informatica - Power Center, SAP - BusinessObjects Data Integrator, Microsoft - SQL Server Integration Services, Oracle Warehouse Builder (OWB), Pentaho Data Integration, Talend Open Studio, etc. [34] ;

- **Service de stockage** représente le dépôt de données dédié au stockage d'un entrepôt de données ou de plusieurs datamarts. Les entrepôts de données et les datamarts sont décrits par les méta-données qui définissent la sémantique des données, des règles organisationnelles, des politiques et des contraintes liées aux éléments de données ;

34. http://datawarehouse4u.info/ETL-tools.html

- **Service de présentation** permet aux utilisateurs de manipuler, explorer et afficher le contenu d'un entrepôt de données sous différentes formes (statistiques, tableaux, etc.) par le biais de nombreux outils. Il comprend l'exécution des requêtes OLAP, des outils de reporting et statistiques pour fournir des tableaux de bord selon les exigences de décision et des outils de data mining (exploration de données) pour découvrir des connaissances précieuses à partir de données actuellement stockées.

2.4 Modèle de données

Dans les modèles de $\mathcal{BD}$ classiques, nous trouvons les termes "tables et relations". Une table représente une entité et une relation représente un lien entre ces entités. A l'instar de ces termes, dans les entrepôts de données, nous trouvons "dimensions et faits". Une dimension décrit un axe selon lequel nous pouvons faire des analyses (par exemple la dimension PRODUIT, la dimension CLIENT). Les faits sont les tables qui contiennent les données opérationnelles de l'entreprise, dont dépendent toutes les dimensions et sur lesquelles les analyses sont basés. La table de fait contient des clés étrangères venant des tables de dimensions et de valeurs numériques appelées mesures [129] (par exemple la table de fait de quantités vendues).
Il existe deux modèles d'entrepôt de données : schéma en étoile et schéma en flocon.

2.4.1 Schéma en étoile

Le schéma en étoile est le modèle le plus simple et le plus utilisé pour développer des entrepôts et des magasins de données. Un schéma en étoile est composé d'une table de fait et de plusieurs tables de dimensions. Les dimensions ont un champ clé et un champ supplémentaire pour chaque attribut. La table de fait est une table centrale qui contient des informations transactionnelles et des clés étrangères pour les tables de dimensions, alors que les tables de dimensions ne contiennent que des données de base. La figure 5.2 montre un schéma en étoile avec une table de fait et quatre tables de dimension (CLIENT, PRODUIT, LOCALISATION et TEMPS). Les principaux avantages de la conception du schéma en étoile sont la facilité de compréhension et la réduction du nombre de jointures nécessaires pour récupérer les données [69].

2.4.2 Schéma en flocon

Le schéma en flocon est une variante du schéma en étoile qui se compose d'une table de fait connectée à plusieurs tables de dimensions. Chaque table de dimension peut être connectée à d'autre tables avec une relation de type *un à plusieurs*. Un schéma en flocon peut posséder un nombre illimité de dimensions et de niveaux pour chaque dimension. La figure 5.3 illustre un schéma en flocon avec quatre dimensions dont la dimension PRODUIT qui possède deux niveaux.

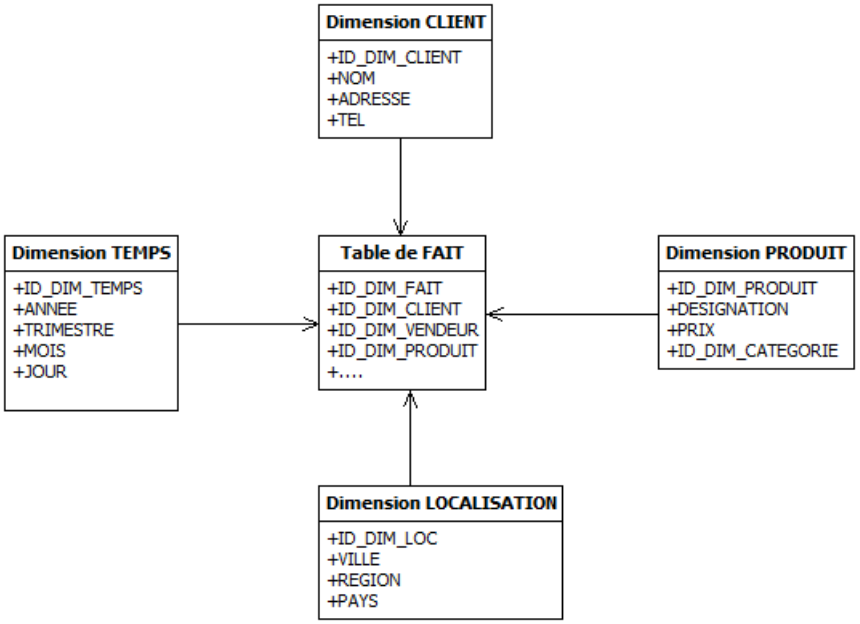


FIGURE 5.2 – Entrepôt de données : modèle en étoile

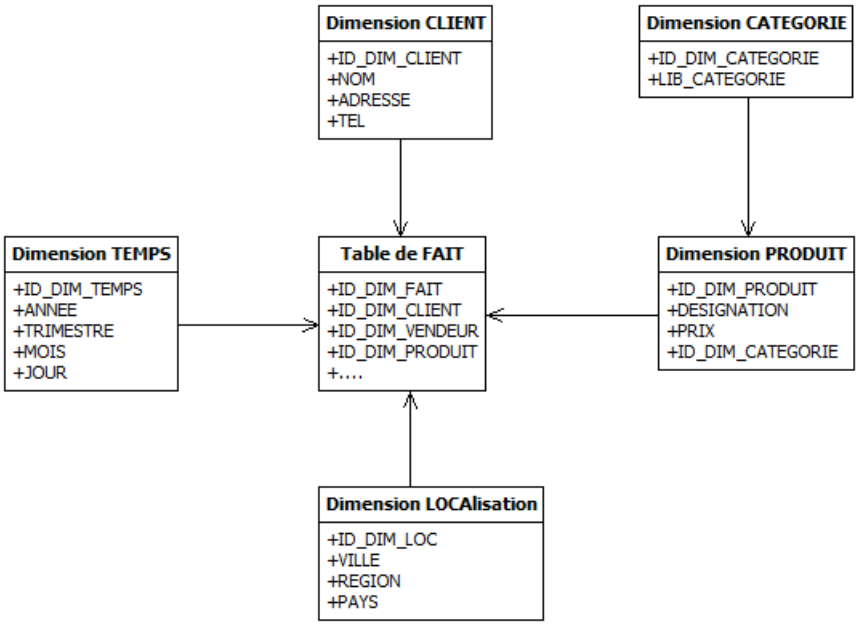


FIGURE 5.3 – Entrepôt de données : modèle en flocon

2.5 Types d'entrepôts de données

Les SGBD relationnels (SGBDR) sont certainement la catégorie la plus utilisée pour développer un entrepôt de données à cause de la ressemblance entre les SGBDR et les modèles des entrepôts de données. Cependant, de manière générale, nous pouvons développer les entrepôts de données avec n'importe quel SGBD. Nous illustrons quatre types d'entrepôt de données.

- Les SGBDR : représentent le type le plus utilisé dans la plupart des cas. Parmi les fournisseurs de ce type, nous pouvons citer IBM, Microsoft, Oracle et SAP.
- Les SGDB analytiques spécialisés : ce type offre des performances d'analyse pour supporter les charges de requêtes spécifiques des entrepôts de données. Il représente un système optimisé reposant sur un matériel, des logiciels et une technologie de $\mathcal{BD}$ en mémoire de premier plan. Il permet de procéder à des analyses ultra rapides et intelligentes en toute simplicité. Pour ce type, nous pouvons citer IBM pure data et InfoSphere [98].
- Appliance d'entrepôt de données (*Data Warehouse Appliances-DWA*) : ce type permet aux utilisateurs d'effectuer des opérations d'analyse de données sur des plateformes parallèles sans nécessiter une connaissance détaillée des techniques de traitement parallèles. Les DWA peuvent être traités comme des appareils fournissant un grand nombre de périphériques de stockage parallèles qui permettent une utilisation du disque à haute performance. Ces appareils désignent un ensemble pré-configuré et équilibré de matériels (serveurs, mémoire, capacité de stockage) et de logiciels (système d'exploitation, SGBD). Ils forment une unité avec une redondance intégrée pour une haute disponibilité et positionnée comme une plateforme pour l'utilisation des SGBD dans les entrepôts de données et le OLTP. Pour ce type, nous pouvons citer Microsoft SQL Server PDW, Oracle Exadata, Teradata, Netezza, XtremeData et Greenplum [184].
- Les entrepôts de données dans le cloud : ce type consiste à fournir un entrepôt de données come un service (*Data Warehousing as a Service-DWaaS*) c-à-d un entrepôt de données stocké sur le Cloud [93]. Le SGBD et les machines sont accessibles via Internet, tandis que le client fournit les données et paie pour le service d'infogérance. Les grands spécialistes de ce type sont Microsoft, Amazon et IBM.

2.6 Traitement analytique en ligne

Le traitement analytique en ligne (OLAP) caractérise le traitement analytique sur de grandes $\mathcal{BD}$ historiques (entrepôts de données) orientées vers la prise de décision [45]. OLAP désigne l'ensemble des technologies permettant la prise de décision stratégique rapide et fiable sur des données modélisées en multidimensionnel en donnant aux utilisateurs la possibilité de les analyser dynamiquement [129]. Les requêtes sont très complexes et impliquent des agrégations. Les applications OLAP sont aussi largement utilisées par les techniques de data mining.

2.6.1 Cube de données

La modélisation multidimensionnelle est une approche inventée par *R.Kimball* [112]. Le modèle multidimensionnel offre une organisation des données selon des axes. Ces axes décrivent les éléments principaux de l'activité d'une entreprise. Trois formes de représentation des données sont définies dans le processus décisionnel : l'entrepôt, le datamart et le cube (ou hypercube). Le cube est une méthode de stockage de données sous forme multidimensionnelle et correspond à une vue métier avec plusieurs dimensions où l'analyste choisit les mesures à observer selon certaines dimensions. Un cube est une collection de données agrégées et consolidées pour récapituler l'information et expliquer la pertinence d'une observation. Le cube de données est exploré à l'aide de nombreuses opérations permettant la flexibilité et la rapidité d'accès et de visualisation des données sous différentes perspectives et la capacité de manipuler les données en utilisant les différentes fonctions d'agrégation classiques : min, max, count, sum, avg, ainsi que d'autre opérations spécifiques.

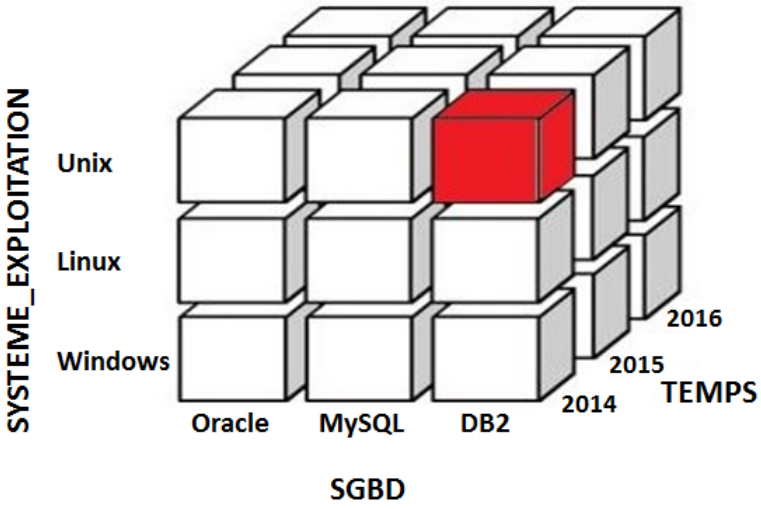


FIGURE 5.4 – Vue d'un cube OLAP à trois dimensions

La figure 5.4 schématise un cube OLAP à trois dimensions : SGBD, SYSTEME_EXPLOITATION et TEMPS recensant le nombre d'installation par an d'un SGBD sur une machine avec un système d'exploitation. Nous pouvons aussi fusionner ou fractionner les valeurs de chaque dimension, obtenant ainsi une représentation hiérarchique comme montré dans la figure 5.5.

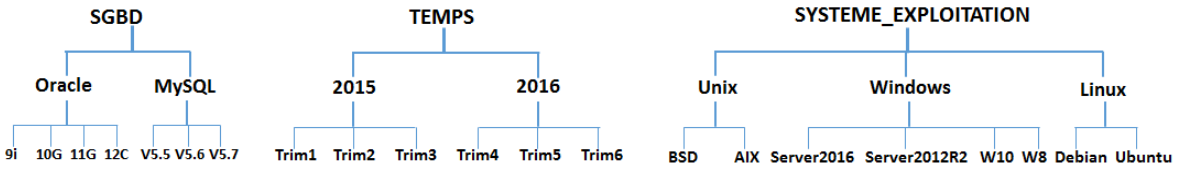


FIGURE 5.5 – Représentation hiérarchique du cube OLAP

2.6.2 Opérations OLAP pour l'interrogation de l'entrepôt de données

Les opérations de manipulation des cubes de données sont classées en deux familles. La première famille contient des opérations liées à la structure de données permettant la réorientation (*rotate*), la permutation (*switch*), la division (*split*), l'emboitement (*nest*), l'enfoncement (*push*) et le retrait (*pull*). La deuxième famille contient des opérations liées à la granularité de données. Ces opérations nécessitent des hiérarchies bien définies pour préparer les calculs automatiques. Elles permettent l'agrégation (*roll up*), le détaill (*roll down*) et la sélection sur une dimension (*slice*) ou sur plusieurs dimensions (*dice*).

2.6.2.1 Cube OLAP : Opération Roll up L'opération Roll up (également appelée opération d'agrandissement ou d'exploration) effectue l'agrégation sur un cube de données, soit en montant une hiérarchie de concept pour une dimension, soit en descendant une hiérarchie de concepts, c'est-à-dire une réduction de dimension [151]. Considérons le cube suivant illustrant le nombre d'installation de certains SGBD effectués chaque année. Supposons que nous voulons récapituler les niveaux (Oracle (9i, 10G, 11G, 12C), MySQL (V5.5, V5.6, V5.7)) de la dimension *SGBD* à partir du cube présenté dans le tableau 5.1. Pour ce faire, nous devons regrouper les colonnes et additionner les valeurs selon la hiérarchie des concepts. En faisant cela, nous obtenons le cube présenté dans le tableau 5.2 :

	Cube OLAP						
	9i	10G	11G	12C	V5.5	V5.6	V5.7
2015	13	26	5	41	17	59	11
2016	25	38	19	47	20	36	18

Tableau 5.1 – Exemple cube OLAP

	Roll up	
	Oracle	MySQL
2015	85	87
2016	129	74

Tableau 5.2 – Opération roll up

2.6.2.2 Cube OLAP : opération Roll down L'opération Roll down (aussi appelée drill down) est l'inverse du roll up. Elle permet de naviguer à partir de données moins détaillées vers des données plus détaillées. Elle peut être réalisée en décalant une hiérarchie de concept pour une dimension ou en introduisant des dimensions supplémentaires [151]. Le tableau 5.3 présente un exemple d'application de l'opération roll down à la dimension *TEMPS* effectuée sur l'exemple précédant.

2.6.2.3 Cube OLAP : opération Slice L'opération Slice permet d'effectuer une sélection sur une dimension d'un cube donné, résultant ainsi en un sous-cube [151]. Par exemple, si nous faisons la sélection SGBD = Oracle nous obtenons le cube présenté dans le tableau 5.4.

2.6.2.4 Cube OLAP : opération Dice L'opération Dice définit un sous-cube en effectuant une sélection sur deux ou plusieurs dimensions [151]. Par exemple, en appliquant la sélection

	Roll down	
	Oracle	MySQL
Trim1	25	22
Trim2	36	29
Trim3	24	36
Trim4	43	19
Trim5	39	28
Trim6	47	27

Tableau 5.3 – Opération roll down

	Slice
	Oracle
Trim1	25
Trim2	36
Trim3	24
Trim4	43
Trim5	39
Trim6	47

Tableau 5.4 – Opération slice

(TEMPS = Trim2 ou TEMPS = Trim3) et (SGBD = Oracle ou SGBD = MySQL) au cube d'origine nous obtenons le sous-cube présenté dans le tableau 5.5.

	Roll down	
	Oracle	MySQL
Trim2	36	29
Trim3	24	36

Tableau 5.5 – Opération dice

2.6.2.5 Cube OLAP : opération Pivot L'opération Pivot également appelée rotation, permet de changer l'orientation dimensionnelle du cube, c'est-à-dire faire tourner les axes pour visualiser les données à partir de différentes vues. Pivot regroupe des données de différentes dimensions. Le tableau 5.7 présente une représentation du tableau 5.6 après l'application de 'opération Pivot .

Pour résumer, les opérations OLAP typiques incluent le roll up, le roll down, le slicing, le dicing et le pivot, ainsi que certaines opérations statistiques telles que : scoping, screening, drill across et drill through.

	Pivot		
	SGBD	2015	2016
Linux	9i	06	16
	10G	10	14
	11G	02	06
	12C	15	19
Windows	9i	07	09
	10G	08	18
	11G	01	09
	12C	18	20

Tableau 5.6 – Avant l'opération pivot

	Pivot		
	SGBD	Linux	Windows
2015	9i	06	07
	10G	10	08
	11G	02	01
	12C	15	18
2016	9i	16	09
	10G	14	18
	11G	06	09
	12C	19	20

Tableau 5.7 – Après l'opération pivot

3 Notre entrepôt de données de tests non fonctionnels

Afin de bénéficier des avantages et de la maturité du domaine des entrepôts de données (par exemple la constitution d'une collection de données centralisées et la conservation de l'historique des données ce qui permet l'étude des tendances et la prise de décision), nous proposons de développer un dépôt de tests et de le matérialiser comme un entrepôt de données. Notre objectif est de fournir un dépôt libre et ouvert pour la communauté de la recherche de $\mathcal{BD}$ afin de peupler leurs expérimentations et de pouvoir les explorer facilement en exprimant différents paramètres et caractéristiques de test comme le montre la figure 5.6 qui présente une vue globale de l'ensemble des composants intervenant dans un système décisionnel. L'idée est de mettre en place un processus capitalisant l'activité de test afin d'en tirer profit à travers la reproductibilité, la comparaison et l'apprentissage.

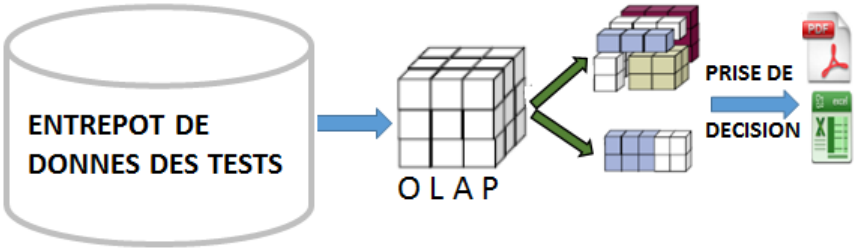


FIGURE 5.6 – Architecture d'un système décisionnel

3.1 Développement de l'entrepôt de données

Le développement d'un entrepôt nécessite l'identification de ses dimensions. Dans notre contexte, les dimensions devraient correspondre aux composantes de l'environnement de test. Nous rappelons que ces composantes ont été explicitées dans le chapitre 4. Cependant, pour

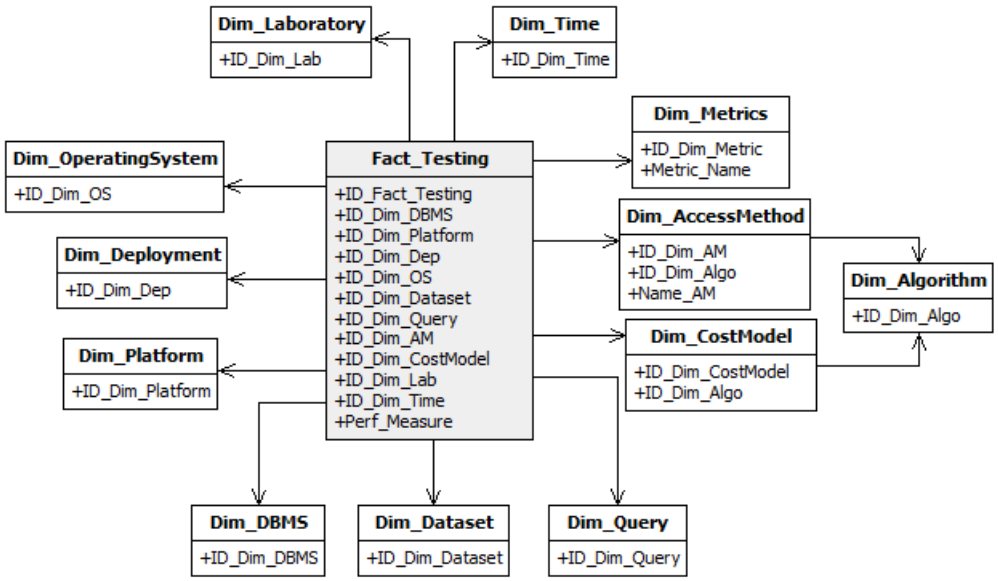


FIGURE 5.7 – Notre entrepôt de données DW_TESTS

donner plus de souplesse à notre dépôt de tests et aussi pour avoir une certaine flexibilité au moment de requêtage, nous avons également identifié les dimensions qui se trouvent au sein des composantes explicitées. Ces dimensions comprennent : *Dim_Platform*, *Dim_Deployment*, *Dim_DBMS*, *Dim_OperatingSystem*, *Dim_Dataset*, *Dim_Query*, *Dim_AccessMethod*, *Dim_-CostModel*, *Dim_Algorithm*, *Dim_Metrics* et deux dimensions d'information : *Dim_Laboratory* et *Dim_Time*. En effet, les composantes ont été transformées en dimensions comme suit :

- La composante DBMS est représentée par la dimension Dim_DBMS.
- La composante DataSet est représentée par la dimension Dim_Dataset.
- La composante Query est représentée par les dimensions Dim_Query, Dim_AccessMethod et Dim_Algorithm. Nous avons fait le choix de faire sortir les deux dernières dimensions de la dimension Dim_Query pour faciliter l'insertion et également l'interrogation.
- La composante OperatingSystem est représentée par la dimension Dim_OperatingSystem.
- La composante HardwareArchitecture est représentée par les dimensions Dim_platform et Dim_Deployment. Nous avons fait apparaitre cette dernière car d'après les articles de tests, les chercheurs mettent souvent l'accent sur le type de déploiement et alors que les informations concernant le hardware restent noyées dans les dimensions voire même ignorées.
- La composante Metric est représentée par la dimension Dim_Metrics.

- La composante TechniqueType est représentée par la dimension Dim_CostModel quand il s'agit d'un test qui fait appel à un modèle de coût.

Comme nous nous intéressons également à la traçabilité du test, nous avons concrétisé cela par l'ajout de deux dimensions qui sont Dim_Laboratory et Dim_Time. La première sert à stocker des informations identitaires de l'organisme effectuant le test, alors que la deuxième sert à stocker la date de l'obtention des résultats en l'occurrence la publication de l'article scientifique.

Les faits de notre entrepôt couvrent différentes mesures utilisées par les concepteurs tels que le coût de la CPU, le coût E/S, le coût de la consommation d'énergie et le temps de réponse. Notre entrepôt est modélisé avec un schéma en flocon comme le montre la figure 5.7 appelé DW_TESTS.

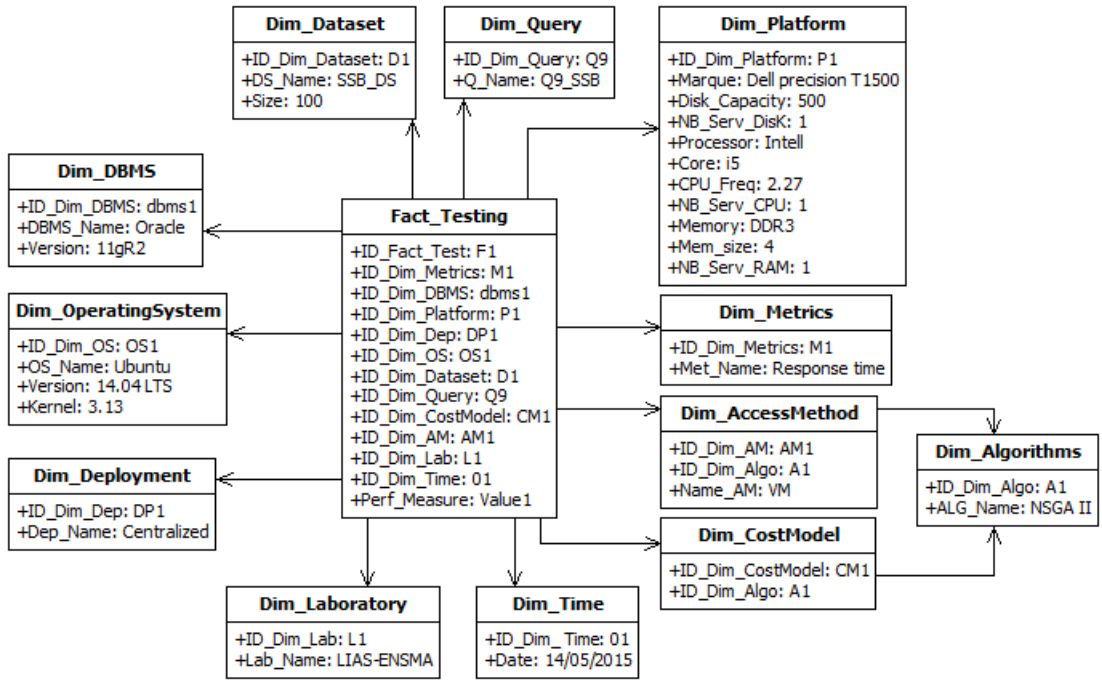


FIGURE 5.8 – Exemple d'instanciation de notre DW_TESTS

La figure 5.8 illustre un exemple d'instanciation de notre entrepôt en considérant le test présenté dans le tableau 4.1. Puisque nous visons à encourager les chercheurs à partager leurs tests, le dépôt offre une interface appropriée facilitant son utilisation. En effet, cette interface doit permettre aux utilisateurs (fournisseurs et producteurs de tests), qui s'intéressent à partager leurs résultats scientifiques, d'enrichir le dépôt sur la base du contenu existant. Par conséquent, chaque nouvel enrichissement met à jour le contenu.

3.2 Interface d'insertion et de sélection de tests

Pour interagir avec notre dépôt de tests, nous avons conçu une interface de programmation applicative (*Application Programming Interface-API*). cette interface est composé de deux parties (voir la figure 5.9) : (i) une partie qui consiste à insérer des nouveaux tests proviennent des fournisseurs spécialisés dans le test via l'interface d'insertion, (ii) une partie qui consiste à rechercher des tests en se basant sur un manifeste exprimé par le demandeur de tests via l'interface de recherche.

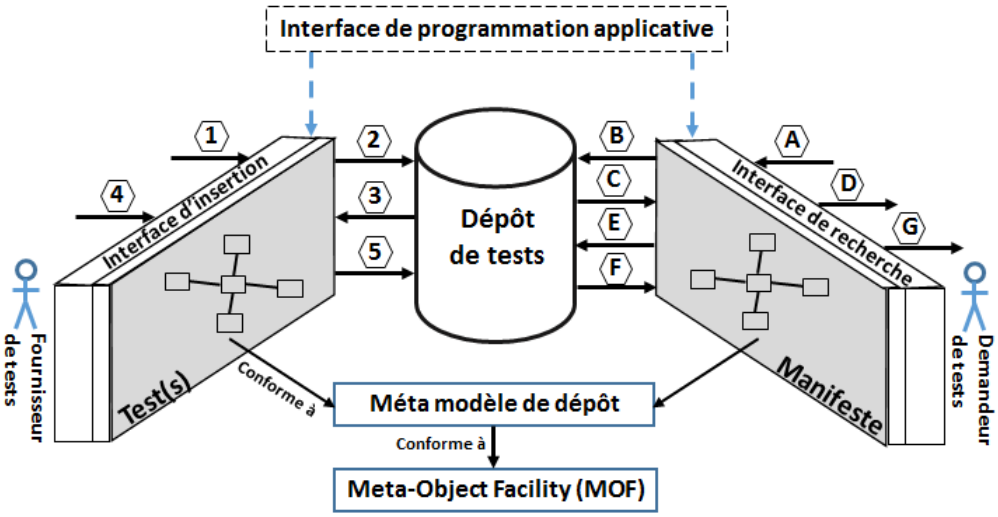


FIGURE 5.9 – Aperçu du dépôt de tests

La figure 5.10 représente un exemple d'un manifeste, dans lequel le SGBD et la métrique de performance (exemple QphH) sont manquants. Cela signifie que l'entreprise est à la recherche d'un SGBD et de performances pour son application.

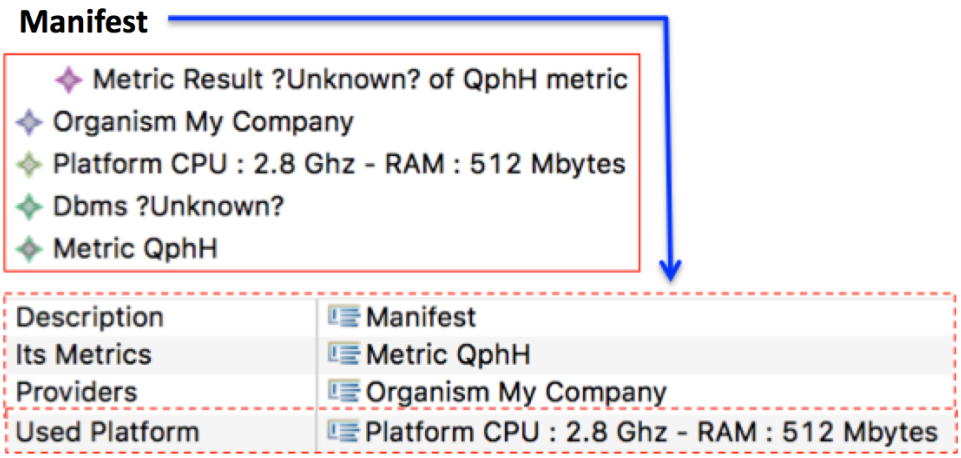


FIGURE 5.10 – Exemple de *manifeste*

3.2.1 Manifeste un moyen pour définir les exigences du déployeur

Nous considérons le scénario suivant illustrant les exigences d'un *déployeur* exprimé dans un document (que nous appelons manifeste). Ce dernier contient les informations suivantes : le schéma de sa $\mathcal{BD}$, ses besoins fonctionnels et non fonctionnels et ses contraintes, et il cherche à obtenir des recommandations afin de choisir le meilleur couple $< SGBD, plateforme >$ qui satisfait ses besoins.

En analysant les données publiées dans le manifeste qui couvrent les éléments de l'environnement de déploiement potentiel à son application ($A^{BD} :< C2V, SGBD, Plateforme, Usage, Sat >$), nous distinguons trois types de données : *des données fournies* (DF) et *des données non fournies* (DnF), *des données à recommander par notre système* (DaR).

L'ensemble des données du manifeste appartient à trois catégories liées au SGBD que nous avons dénoté par $DBMS_{features}$, à la configuration fonctionnelle ($Config$) (jeux de données, requêtes, schémas, SGBD, plateforme, etc.) et aux métriques utilisées par exprimer l'ensemble des besoins non fonctionnels ($Metric$) pour évaluer le critère de satisfaction Sat.

3.2.2 Interaction entre le manifeste et l'entrepôt

Le manifeste du *déployeur* et l'entrepôt de données interagissent fortement. Cette interaction est de type *push and pull*. Plus précisément, des données du manifeste peuvent être insérées dans l'entrepôt de données ou utilisées (avec des requêtes OLAP SQL) pour extraire des données de cet entrepôt.

Pour réaliser cette interaction, nous avons fait appel aux techniques de l'ingénierie dirigée par les modèles [166]. Une autre utilisation de ces techniques concerne le développement de l'interface d'exploitation de l'entrepôt.

En effet le paradigme de l'ingénierie dirigée par les modèles (méta-modélisation, transformation de modèles, génération de code, etc.) nous a facilités le développement du squelette de l'API utilisateur. Chaque instance d'un manifeste et chaque résultat d'une requête de départ générant un ou plusieurs environnements de tests sont des instances conformes au modèle de dépôt de tests (voir la figure 5.11). La conformité ici a le même sens que qu'on dit un programme est conforme à une grammaire d'un langage de programmation. Le modèle de dépôt de tests peut alors être vu comme un langage d'expression pour décrire les manifestes et aussi les environnements de tests.

Le modèle de dépôt de tests est exprimé en Ecore [173] qui est une représentation Eclipse du MOF (Meta-Object Facility) [35]. Un extrait du modèle est illustré dans la figure 5.11. Les éléments graphiques correspondent au langage Ecore qui ressemble au diagramme de classes UML, mais avec une sémantique différente. La classe racine du modèle est le `Repository` classe. Elle regroupe les informations relatives aux tests, les méthodes d'accès, les mesures, les

35. www.omg.org/spec/MOF/2.4.1/

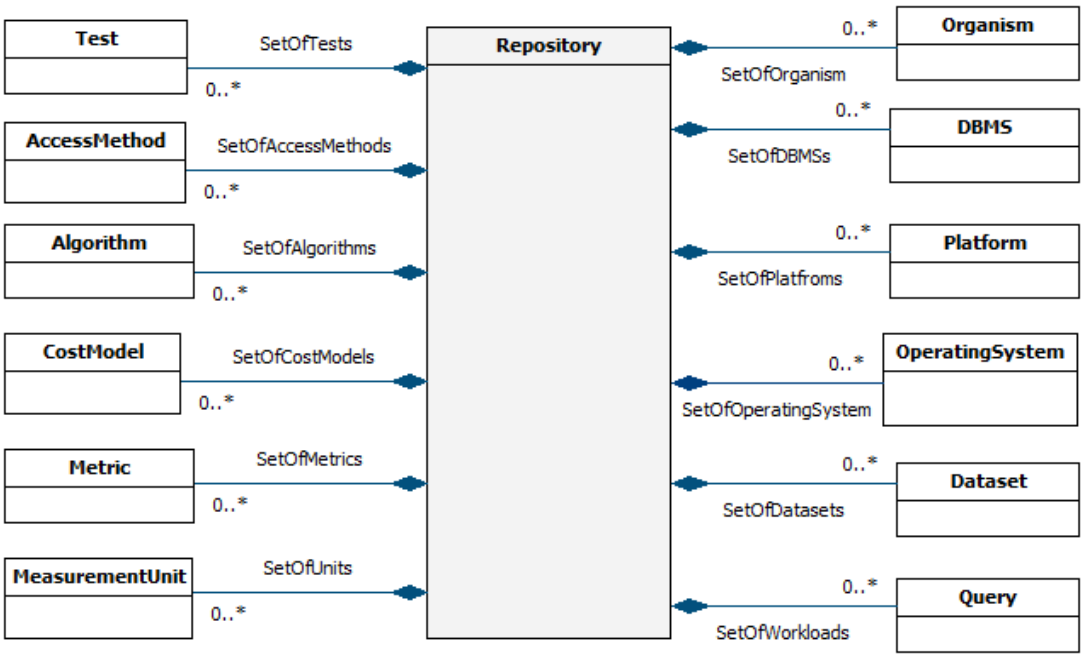


FIGURE 5.11 – Extrait du dépôt de test : entrepôt de données

plateformes, la charge de requêtes, etc.

3.2.3 Insertion des tests/instances du dépôt

Le flux général lié à l'insertion des tests dans notre entrepôts de données correspond aux étapes 1 à 5 illustrées dans la figure 5.9 :

- Étape (1) : le *déployeur* demande à jouer le rôle d'un fournisseur de test ;
- Étape (2) : l'interface fournisseur transforme la demande en un ensemble de requêtes pour sélectionner toutes les dimensions qui existent dans le dépôt de tests avec leurs valeurs à l'exception des métriques qui seront sélectionnées sans valeurs ;
- Étape (3) : l'interface fournisseur charge le résultat de l'étape (2) et le présente au *déployeur* comme une instance du dépôt ;
- Étape (4) : le *déployeur* utilise l'interface pour l'ajout de(s) nouveau(x) test(s) sur la base des informations déjà existantes dans l'instance. Le *déployeur* peut également ajouter une nouvelle valeur d'une dimension si cela est nécessaire.
- Etape (5) : l'interface du fournisseur transforme les données de l'étape (4) en un ensemble de requêtes appropriées pour insérer de nouvelles informations dans le dépôt.

3.2.4 Recherche des tests

Notons que tous les attributs utilisés dans le manifeste appartiennent au schéma de notre entrepôt. La figure 5.10 représente un exemple de manifeste, dans lequel le SGBD et la métrique de performance sont des informations manquantes. Cela signifie que l'entreprise est à la recherche d'un SGBD et de ses performances pour son application. Le processus de recherche doit considérer le manifeste en explorant l'entrepôt pour trouver un fragment de données de tests correspondant au manifeste, puis proposer à l'entreprise un SGBD. La figure 5.9 met en évidence le flux de travail décrivant le processus de recherche d'un test et ses étapes :

(A) Le *déployeur* choisit de jouer le rôle d'un chercheur de test *test seeker* ;

(B) L'interface chercheur transforme la demande en un ensemble de requêtes pour sélectionner toutes les dimensions avec leurs valeurs à l'exception des métriques qui seront sélectionnées sans valeurs ;

(C) L'interface chercheur charge le résultat de (B) et le présente au *déployeur*. Ce résultat correspond à un manifeste vide ;

(D) Le *déployeur* enrichit le manifeste en exprimant son besoin en fonction du contenu existant. Bien sûr, les *déployeurs* peuvent ajouter de nouvelles valeurs liées aux dimensions quand cela est nécessaire. Toutefois, l'ajout de nouvelles métriques n'est pas possible, parce que l'objectif est d'orienter les concepteurs vers le choix d'une configuration de test en fonction des paramètres qui existent dans le dépôt ;

(E) L'interface chercheur génère un ensemble de requêtes SQL appropriées en se basant sur le manifeste pour explorer le dépôt de test ;

(F) Basé sur les requêtes du manifeste et le contenu du dépôt, un ensemble de tests possibles et leurs configurations spécifiques sont proposées au *déployeur* via l'interface. Dans cet ensemble de tests, les informations manquantes sont remplacées par les valeurs recommandées, sont proposées au *déployeur* via l'interface. A noter que ce problème est tout à fait semblable au problème de la classification des données manquantes [189]. Plusieurs efforts de recherche ont été faits pour résoudre le problème mentionné ci-dessus. Habituellement, des algorithmes et des méthodes sont proposées pour prédire les valeurs manquantes [189]. Ces algorithmes sont définis au niveau de l'attribut et non pas au niveau des dimensions. Cela nous motive à développer notre propre algorithme. L'idée de base est d'éliminer les dimensions qui ne sont pas exprimées dans le manifeste. Sur la base des résultats obtenus, nous estimons les valeurs des attributs inconnus de dimensions. Les détails de cet algorithme sont présentés dans le chapitre 6 ;

(G) Enfin, le *déployeur* peut télécharger des informations relatives à la solution proposée. Notons que les résultats de la recherche doivent correspondre à un dépôt contenant un ou

plusieurs tests en fonction des demandes. L'objectif est de permettre aux *déployeur* de télécharger des référentiels sur mesure faisant référence à leurs besoins.

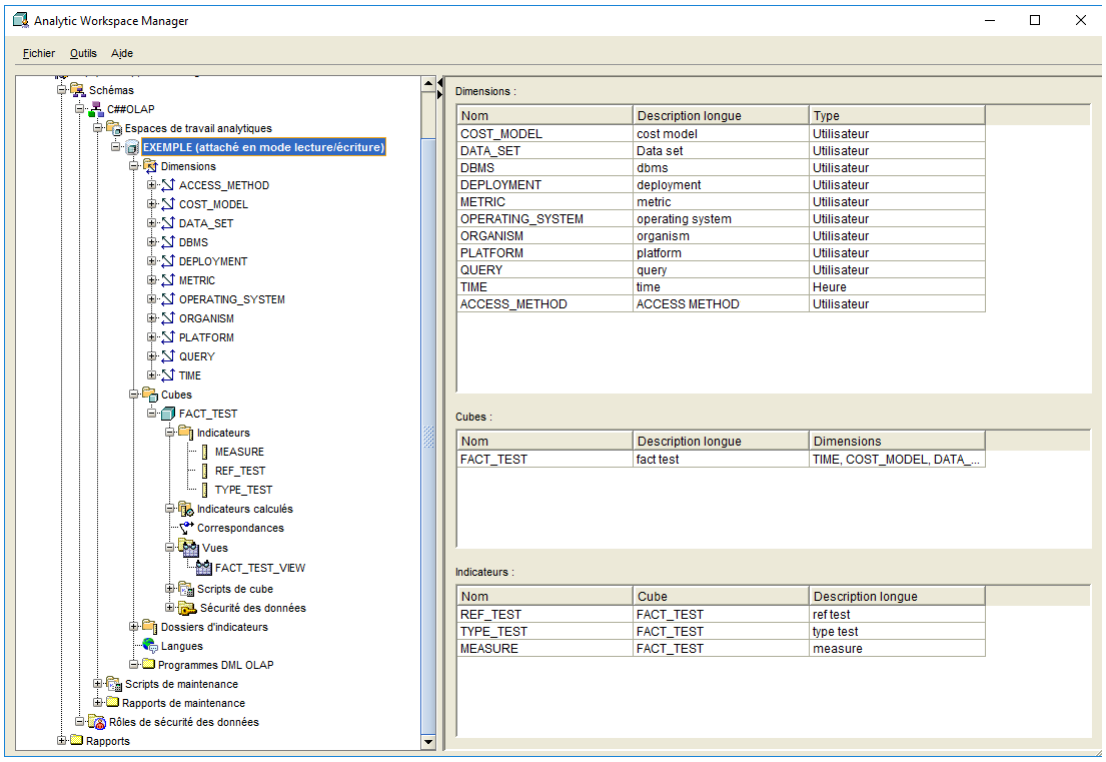
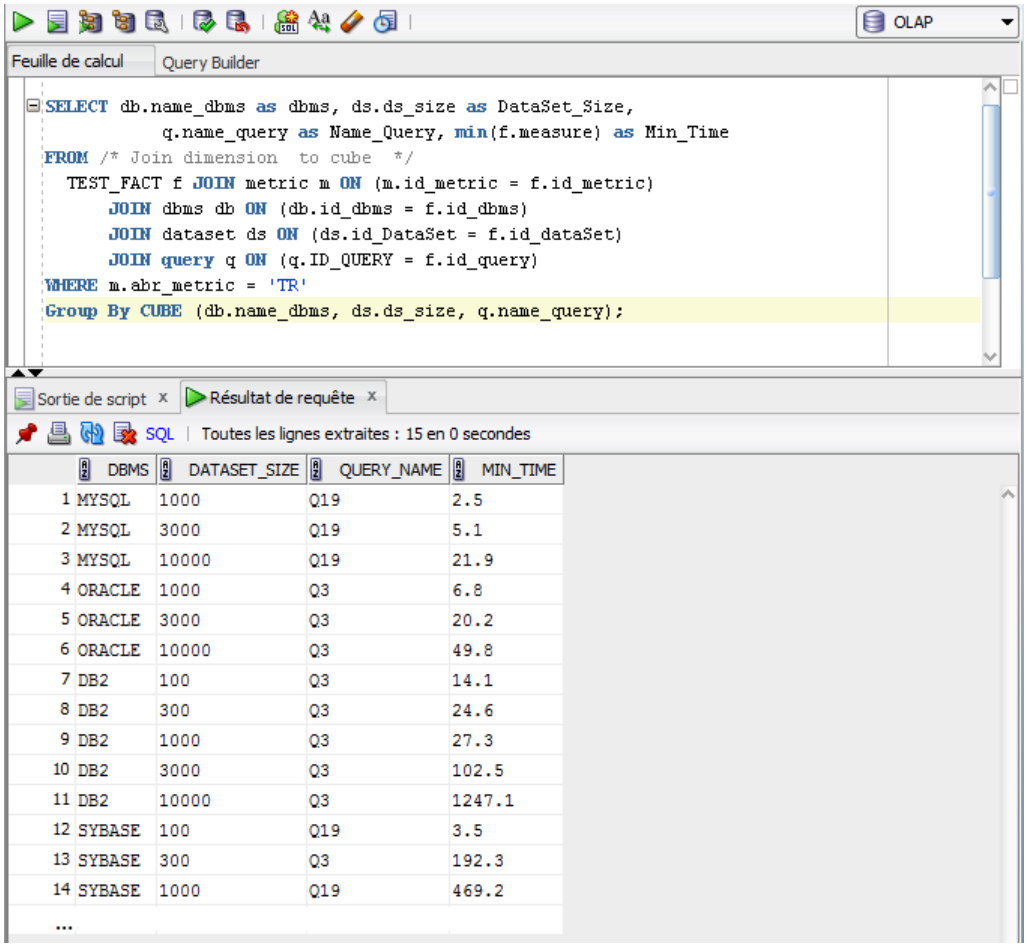


FIGURE 5.12 – Vue sur le gestionnaire d'espace de travail analytique pour Oracle OLAP

3.3 Exemples d'usage de notre Entrepôt

OLAP permet à un utilisateur d'extraire et de visualiser facilement et sélectivement des données à partir de différents points de vue. OLAP permet également de comparer les résultats de tests avec d'autres SGBD. Pour faciliter ce type d'analyse, les données OLAP sont stockées dans une $\mathcal{BD}$ multidimensionnelle. Alors qu'une $\mathcal{BD}$ relationnelle peut être considérée comme bidimensionnelle, une $\mathcal{BD}$ multidimensionnelle considère chaque attribut de données comme une dimension distincte. La figure 5.12 présente une vue sur l gestionnaire d'espace de travail analytique pour Oracle OLAP, version 11G (analytic workspace manager for Oracle OLAP). Deux requêtes OLAP sont également présentées dans les figures 5.13 et 5.14 pour montrer l'usage de notre entrepôt de données. La première requête consiste à sélectionner les tests ayant le temps de réponse minimum pour chaque SGBD, taille de $\mathcal{BD}$ et requête. La deuxième requête consiste à sélectionner les tests ayant un temps de réponse inférieur à 100 pour chaque SGBD et chaque requête.



```
SELECT db.name_dbms as dbms, ds.ds_size as DataSet_Size,
       q.name_query as Name_Query, min(f.measure) as Min_Time
FROM /* Join dimension  to cube  */
  TEST_FACT f JOIN metric m ON (m.id_metric = f.id_metric)
      JOIN dbms db ON (db.id_dbms = f.id_dbms)
      JOIN dataset ds ON (ds.id_DataSet = f.id_dataSet)
      JOIN query q ON (q.ID_QUERY = f.id_query)
WHERE m.abr_metric = 'TR'
Group By CUBE (db.name_dbms, ds.ds_size, q.name_query);
```

Sortie de script | Résultat de requête | SQL | Toutes les lignes extraites : 15 en 0 secondes

	DBMS	DATASET_SIZE	QUERY_NAME	MIN_TIME
1	MYSQL	1000	Q19	2.5
2	MYSQL	3000	Q19	5.1
3	MYSQL	10000	Q19	21.9
4	ORACLE	1000	Q3	6.8
5	ORACLE	3000	Q3	20.2
6	ORACLE	10000	Q3	49.8
7	DB2	100	Q3	14.1
8	DB2	300	Q3	24.6
9	DB2	1000	Q3	27.3
10	DB2	3000	Q3	102.5
11	DB2	10000	Q3	1247.1
12	SYBASE	100	Q19	3.5
13	SYBASE	300	Q3	192.3
14	SYBASE	1000	Q19	469.2
...				

FIGURE 5.13 – Minimum de temps de réponse par requête, SGBD et taille de BD

4 Conclusion

Dans ce chapitre, nous avons conçu un dépôt de tests, "DBLP-like" [36], qui joue le rôle d'un entrepôt de données ouvert et collaboratif sur les résultats des tests. Nous avons motivé notre proposition par plusieurs faits identifiés par des chercheurs industriels et universitaires tels que la difficulté de reproduction des résultats des tests, le coût élevé des tests, etc. Nous avons identifié des dimensions de notre entrepôt de données couvrant différents aspects de l'environnement de test. La table de fait contient également les mesures de test. L'utilisation de l'entreposage de données permet aux utilisateurs de pouvoir effectuer des rapports et visualiser les données de test à l'aide des outils OLAP. Ainsi, nous avons développé une interface utilisateur liée à la recherche et à l'évaluation des tests à partir des besoins exprimés par le *déployeur*. Actuellement, l'intégrité des données est gérée par un modérateur. Cependant, il serait aussi possible de développer des techniques garantissant la bonne qualité des données. Nous avons également développé des techniques pour vérifier l'intégrité des données à stocker afin d'assurer certaines

36. http://dblp.uni-trier.de/

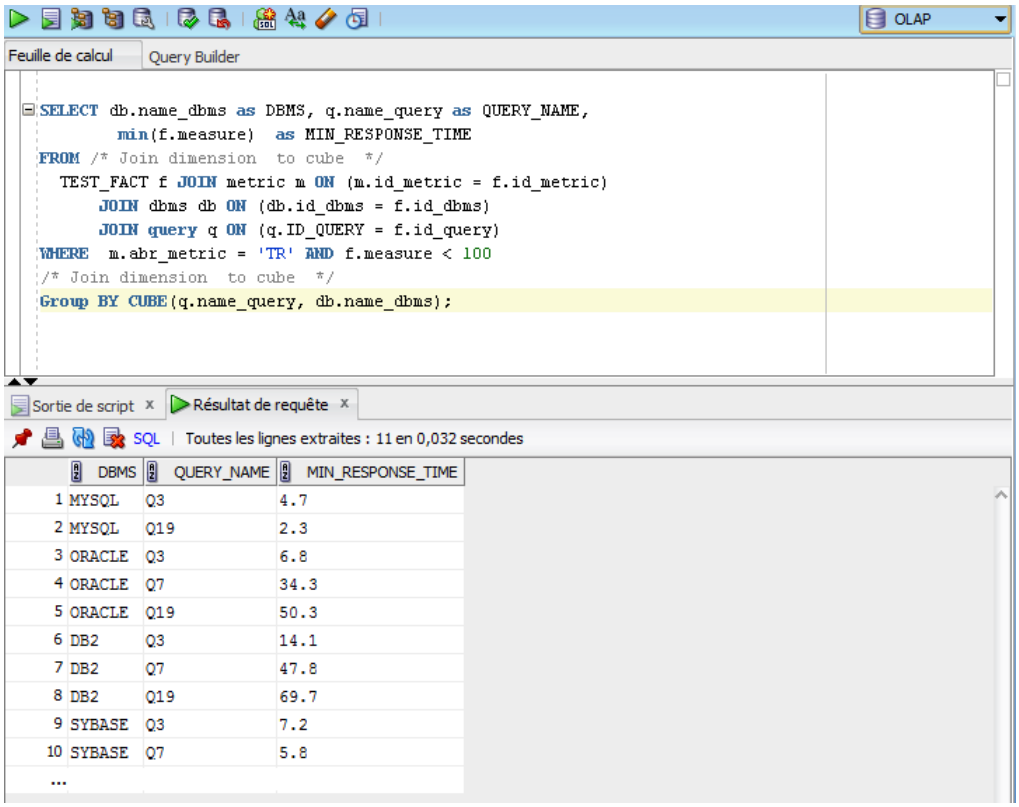


FIGURE 5.14 – Liste des SGBD ayant un temps de réponse < 100 par requête

garanties de l'utilisation de ce dépôt.

La simple utilisation telle qu'elle a été présentée dans ce chapitre n'est pas suffisante. Le dépôt de tests nécessite d'être mieux exploité afin d'obtenir un système plus complet jouant le rôle d'un système de recommandation. C'est dans cette optique que le chapitre suivant apportera plus d'éclaircissement sur la manière dont le dépôt de test peut être couplé avec l'ontologie des SBD pour répondre au problème de choix de SGBD et de plateforme.

Chapitre 6

Système de recommandation pour le déploiement des bases de données

Sommaire

1 Introduction 137

2 Définitions et concepts 137

2.1 Aide à la décision multicritère 137

2.2 Modèle de la somme pondérée 138

2.3 Apprentissage automatique et similarité 138

2.3.1 Similarité et distance 139

2.3.2 Similarité de requêtes 139

2.4 Système de recommandation 140

2.4.1 Filtrage collaboratif 141

2.4.2 Systèmes de recommandation basés sur le contenu . . 143

2.4.3 Filtrage basé sur la connaissance 143

3 Composition de notre système de recommandation 144

3.1 Phase 1 : Approche basée sur la décision multicritère 145

3.2 Phase 2 : Approche basée sur l'apprentissage automatique 146

4 Implémentation et expérimentation 146

4.1 Outils et environnement de développement 148

4.2 Évaluation et analyse des résultats de la première phase 148

4.3 Évaluation et analyse des résultats de la deuxième phase 151

4.3.1 Cas d'étude 1 : requêtes des bancs d'essai 151

4.3.2 Cas d'étude 2 : Requêtes des déployeurs 155

5 Conclusion 158

Résumé. L'entrepôt de données que nous avons construit stocke un sous-ensemble de l'espace de recherche de notre problème ($N \times M$, où N et M représentent respectivement le nombre de SGBD et de plateformes disponibles). Il stocke alors ce sous ensemble de résultats de tests non fonctionnels. Sur cet entrepôt, nous construisons un système de recommandation dédié aux *déployeurs* pour les assister à sélectionner le SGBD et sa plateforme pour leur application. En effet, ce système est basé sur notre entrepôt et sur l'ontologie des fonctionnalités des systèmes de $\mathcal{BD}$ (SBD). Le rôle de ce système de recommandation est d'orienter le *déployeur* vers la solution appropriée en fonctions de SGBD et de plateformes par rapport aux besoins. Plusieurs scénarios d'usage ont été joués pour montrer l'efficacité du système par rapport aux approches étudiées précédemment.

1 Introduction

Afin de proposer au *déployeur* des conseils pertinents par rapport à ses attentes, les systèmes de recommandation peuvent offrir au *déployeur* un assistanat convenable relatif à ses activités de recherche d'informations. D'ailleurs, les systèmes de recommandation peuvent être utilisés pour sélectionner le meilleur SGBD et la meilleure plateforme à travers l'exploitation soit à partir de l'ontologie des systèmes de base de données et/ou soit de l'entrepôt des données de tests.

Dans ce contexte, nous proposons un nouveau système de recommandation s'articulant sur deux phases : la première est basée sur le dépôt présenté dans le chapitre 5, et la deuxième phase est basée sur l'ontologie présentée dans le chapitre 4. A titre de rappel, notre objectif est d'offrir aux *déployeur* la possibilité de prendre une bonne décision dans le choix du SGBD et de la plateforme. L'exploitation du dépôt des tests et de l'ontologie sera assurée respectivement par des techniques d'apprentissage automatique (*machine learning*) et des techniques de la décision multicritère. Pour bien expérimenter les études et solutions antérieures, nous présentons un prototype d'aide à la décision, appelé "Système de recommandation pour la sélection des SGBD". Ce prototype implémente les approches que nous allons présenter dans ce chapitre. Notre système permet notamment de suggérer des recommandations pour choisir le meilleur SGBD en garantissant de satisfaire le maximum des critères exprimés par le *déployeur*, à savoir les jeux de données, les charges de requêtes et les plateformes.

Le reste de ce chapitre est organisé comme suit : dans la section 2, nous présentons quelques définitions relatives aux techniques étudiées (la décision multicritère, la méthode de la somme pondérée, l'apprentissage automatique et la similarité). Nous exposons divers principaux types de systèmes de recommandation dans la section 2.4. Dans la section 3, nous détaillons notre système de recommandation, ses phases et ses composants. Nous détaillons les aspects liés à l'implémentation et l'environnement d'expérimentation dans la section 4. Nous analysons par la suite les résultats obtenus pour montrer l'efficacité de nos approches.

2 Définitions et concepts

Cette section présente les techniques d'aide à la décision multicritère (*Multi-criteria decision making-MCDM*) et l'une de ces meilleures méthodes, appelée le modèle de la somme pondérée (*Weighted sum model-WSM*) [10].

2.1 Aide à la décision multicritère

Les techniques d'aide à la décision multicritère ont pris une ampleur croissante dans la recherche opérationnelle grâce à leur capacité à évaluer d'une manière subjective les perfor-

mances des outils informatiques et mathématiques. Au cours des dernières années, plusieurs travaux se sont basés sur des outils MCDM afin de résoudre de nombreux problèmes liés à l'énergie, l'environnement, la gestion de qualité, la sécurité et à la gestion des risques. D'ailleurs, les méthodes dites MCDM ont été conçues pour proposer des solutions alternatives selon les préférences des utilisateurs et de les classer selon un ordre de préférence subjectif. De plus, MCDM est un terme générique pour désigner les méthodes existantes permettant d'aider les gens à prendre des décisions en fonction de leurs préférences [132].

2.2 Modèle de la somme pondérée

La somme pondérée est un ancien modèle de MCDM et il peut probablement le plus utilisé [182]. Il est connu comme le meilleur et le plus simple des modèles d'aide à la décision multicritère [10]. En effet, il combine les différents objectifs et les poids correspondant à ces objectifs pour créer un score unique pour chaque solution alternative afin de les rendre comparables. Par la suite, la meilleure alternative est celle qui a le score maximum de WSM. Les différents objectifs sont supposés être positifs [102]. Étant donné un ensemble de m alternatives dénotées $A_1, A_2, A_3, ..., A_m$ et un ensemble de n critères de décision dénotés $C_1, C_2, C_3, ..., C_n$. Nous supposons que tous les critères sont des critères de prestations, et w_j désigne le poids relatif de l'importance du critère C_j et a_{ij} désigne la valeur de la performance de $i-$ème alternative quand elle est évaluée en termes de j-ème critère. Alors, le total de l'importance de la i-ème alternative, dénoté par $A_i^{WSM-score}$, est défini par l'équation suivante [182] :

$$A_i^{WSM-score} = \sum_{j=1}^{n} w_j a_{ij}, \ for\ i = 1, 2, 3, ...m \tag{1}$$

Il est noté que les poids devraient être normalisés de sorte que $\sum_{j=1}^{n} w_j = 1$.
Par conséquent, la meilleure solution est celle qui satisfait (dans le cas de maximisation) la formule suivante :

$$A_*^{WSM-score} = \max_i \sum_{j=1}^{n} w_j a_{ij}, \ for\ i = 1, 2, 3, ...m \tag{2}$$

2.3 Apprentissage automatique et similarité

L'apprentissage automatique offre la possibilité de concevoir des algorithmes permettant de faire des prédictions ou de calculer des suggestions en se basant sur des informations liées à l'entrée et la sortie du processus d'apprentissage. De plus, les exemples les plus courants d'apprentissage automatique sont liés aux systèmes de recommandation en ligne telles que Amazon et Netflix, ainsi que la détection de fraude, qui représente l'une des utilisations la plus importante dans le monde des affaires.

2.3.1 Similarité et distance

Définition 1

*La **similarité** permet de comparer deux objets pour déterminer les relations les plus importantes et utiles entre eux [63].*

Définition 2

*La **distance** représente la mesure inverse de la similarité. Plusieurs fonctions de distance existent telles que la distance euclidienne définie comme suit :*
Soient $P_1(x_1, x_2, ..., x_k)$ et $P_2(y_1, y_2, ..., y_k)$ deux points d'un espace vectoriel, la distance entre P_1 et P_2 est donnée par l'équation suivante :

$$Distance = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{3}$$

Soient a et b deux valeurs de mise à l'échelle. a et b sont similaires si elles vérifient la relation suivante [65] :

$$\begin{array}{ll} Relation\ relative : \frac{a}{b} \approx 1 & \text{si } \frac{a}{b} \in [1-\epsilon, \frac{1}{1-\epsilon}] \\ Relation\ Absolue : a - b \approx 0 & \text{si } |a-b| \in [0, \epsilon] \end{array} \tag{4}$$

et ϵ est la plus petite valeur dans l'échelle de a ou b. Parmi les deux relations ci-dessus, la relation relative correspond mieux à notre problème. Par conséquent, la similarité peut être assimilée au rapport entre les mesures estimées et les mesures réelles.

Définition 3

*La **normalisation** représente une propriété de la similarité et exige que toutes les valeurs appartiennent à l'intervalle [0, 1]. Il existe diverses formules de normalisation dans les statistiques. Soit $X = \{x_1, x_2, \ldots, x_n\}$ un échantillon de n objets évalués. La valeur normalisée de l'objet x_i est donnée par :*

$$X_{i,\ 0\ a\ 1} = \frac{x_i - X_{Min}}{X_{Max} - X_{Min}} \tag{5}$$

Où x_i représente chaque point de la donnée i ; X_{Min} et X_{Max} représentent respectivement le minimum et le maximum parmi tous les points de données, $X_{i,\ 0\ a\ 1}$: le point de données normalisé entre 0 et 1. Dans le cas où la distance (D) est normalisée, la similarité S peut être exprimée de la façon suivante : $S = 1 - D$.

2.3.2 Similarité de requêtes

Les fonctions de similarité que nous avons présentées précédemment ne concernent que les valeurs numériques. Dans le cas du calcul de la similitude entre les requêtes, nous utilisons l'algorithme *PLASTIC SIMCHECK* [84] qui est basé sur le vecteur de fonctionnalités de requête

que nous avons détaillé dans la section 3.3 du chapitre 4. L'algorithme prend en entrée deux vecteurs à comparer et renvoie un booléen indiquant l'existence ou non d'une similarité entre ces requêtes. Cet algorithme fonctionne en deux étapes : (i) comparaison des vecteurs et (ii) établissement d'une correspondance entre les tables. Dans la première étape, les deux vecteurs de requête sont comparés sur la base du nombre de tables, du nombre total de prédicats de jointure et de la séquence de degré de requête. C'est seulement en cas d'égalité entre ces attributs que l'on passe à la deuxième étape. Cette deuxième étape établit des correspondances une à une entre les tables des deux requêtes et crée ainsi des paires de tables compatibles. Deux tables sont dites compatibles si elles ont les mêmes valeurs en termes de degré de table, comptage d'index de jointure et nombre de prédicats. Ensuite, pour chaque ensemble de tables compatibles, *SIM-CHECK* calcule la distance entre elles, en utilisant deux équations de distance, soit en fonction de la taille réelle et la taille effective estimée (Équation 6), soit en fonction de la taille effective (Équation 7)

$$Distance(Q1, Q2) = \sum_{i=1}^{k} \frac{\mid ETS^i_1 - ETS^i_2 \mid}{max(TS^i_1, TS^i_2)} \tag{6}$$

$$Distance(Q1, Q2) = \sum_{i=1}^{k} \frac{\mid ETS^i_1 - ETS^i_2 \mid}{max(ETS^i_1, ETS^i_2)} \tag{7}$$

Où TS^i_1, ETS^i_1, TS^i_2 et ETS^i_2 ($i = 1..k$) représentent les tailles réelles et effectives de k tables impliquées respectivement dans les requêtes Q1 et Q2. Cela nous conduit à estimer le facteur de sélectivité. Elle s'agit d'un coefficient représentant le rapport entre le nombre d'enregistrements sélectionnés (taille effective) et le nombre d'enregistrements total d'une table (taille réelle). Si la sélectivité vaut 1, tous les enregistrements sont sélectionnés [28]. Si elle vaut 0, aucun enregistrement n'est sélectionné.

2.4 Système de recommandation

Les systèmes de recommandation sont généralement définis comme des modules exploités par des applications ou des outils et des techniques fournissant des suggestions pour que les items soient utiles à un utilisateur. Un item représente le terme général utilisé pour désigner ce que le système recommande aux utilisateurs [156]. Les suggestions concernent différents processus de prise de décision, tels que l'article à acheter, le SGBD à choisir, etc. Ils sont devenus des applications fondamentales et ont été largement utilisés dans plusieurs domaines comme le commerce électronique (Cdiscount, Fnac, Amazon, etc.), les jeux vidéos (Steem, Microsoft Xbox Live, etc.), la vente des musiques (Deezer...), la vidéo à la demande (Netflix, Allocine, etc.), le tourisme (Expedia...) et la santé [127]. Ces systèmes fournissent des suggestions qui taillent efficacement les grands espaces d'information afin que les utilisateurs soient orientés vers les éléments difficiles à trouver répondant le mieux à leurs exigences et préférences dans

un temps de recherche très réduit [147].
Plusieurs types de systèmes de recommandation existent. Nous citons comme exemple : les systèmes de recommandation collaboratifs, le systèmes de recommandation basés sur le contenu, les systèmes de recommandation basés sur la démographie, les systèmes de recommandation basés sur l'utilité et les systèmes de recommandation basés sur la connaissance. Ces méthodes ont parfois été combinées dans des systèmes de recommandation hybrides [44]. Ce genre de systèmes utilise des informations différentes pour proposer des recommandations. En effet, à titre d'exemple, le filtrage collaboratif s'articule sur la similarité de comportement ou d'usage entre les utilisateurs. D'autre part, le filtrage basé sur le contenu utilise des informations statiques sur des utilisateurs ou des items tandis que celui basé sur la connaissance dépend des informations qui sont fournies par les utilisateurs [137].

2.4.1 Filtrage collaboratif

D'après la communauté des utilisateurs du système, le filtrage collaboratif est la technique de recommandation la plus populaire et la plus réussie dans les systèmes de recommandation [147]. En effet, ce genre de système s'appuie sur les opinions d'autres utilisateurs qui partagent les mêmes idées pour fournir des recommandations d'items ou des prédictions. Les opinions des utilisateurs peuvent être obtenues explicitement auprès des utilisateurs ou en utilisant des mesures implicites [155]. Dans le filtrage collaboratif, l'analyse complexe des caractéristiques des items recommandés n'est pas nécessaire [122].

Les algorithmes de recommandation de filtrage collaboratif peuvent être divisés en deux catégories. Algorithmes de recommandation de filtrage collaboratif basés sur la mémoire [94] et algorithmes de recommandation de filtrage collaboratif basés sur les modèles [127]. La première consiste à faire des recommandations en explorant la $\mathcal{BD}$ des évaluations des utilisateurs et en calculant des poids qui nous permettent de faire des prédictions concernant l'utilisateur actif [165]. Ces poids sont calculés à l'aide de multiples algorithmes à savoir ceux basés sur la corrélation et ceux basés sur la similarité de vecteurs [127]. Cette catégorie est divisée en deux variantes : centrée sur l'utilisateur et centrée sur les attributs des items [122]. La deuxième catégorie estime un modèle qui est alors utilisé pour faire des prédictions en utilisant les données des évaluations des utilisateurs. Plusieurs modèles probabilistes sont utilisés dans ce type, à titre d'exemple, le modèle à base de *clusters* et le modèle à base de réseau bayésien. Le modèle à base de *clusters* consiste à regrouper (en clusters) les utilisateurs ayant les mêmes goûts, et aussi de regrouper (en *clusters*) les items portant sur les mêmes sujets. Le modèle à base de réseau bayésien consiste à représenter un ensemble de variables avec une distribution de probabilité de dépendance où les nœuds représentent les variables et les arcs représentent les dépendances directes entre variables [165]. Les systèmes de recommandation basés sur le filtrage collaboratif pourraient rencontrer trois difficultés. (i) La contrainte liée à la diffusion des documents qui dépend obligatoirement des évaluations d'utilisateurs, freine l'insertion des nouveaux documents. (ii) Le démarrage à froid d'un nouveau système représente un goulot d'étranglement à cause

	Oracle	PostgreSQL	MySQL	MSQL Server	MongoDB	Cassandra
Entreprise1	8		3	9		6
Entreprise2	9	7	5	7	7	2
Entreprise3	9			6		
Entreprise4	5	4		8	5	
Entreprise5	8	6	6			1

Tableau 6.1 – Matrice SGBD entreprises

	Oracle	PostgreSQL	MySQL	MSQL Server	MongoDB	Cassandra
~~Entreprise1~~	~~08~~		~~03~~	~~09~~		~~06~~
Entreprise2	09	07	05	07	07	02
~~Entreprise3~~	~~09~~			~~06~~		
Entreprise4	05	04		08	05	
Entreprise5	08	06	06		? 07	01

Tableau 6.2 – Filtrage collaboratif basé sur l'utilisateur

du fait que tous les documents se considèrent comme des nouveaux. (iii) La rareté de certain profiles qui implique le risque de ne pas recevoir des recommandations et/ou ne pas avoir des propositions auprès des autres utilisateurs.

Le tableau 6.1 présente une évaluation globale d'utilisation de six SGBD par cinq entreprises. Les entreprises représentent les utilisateurs et les SGBD représentent les items. Si nous voulons prédire l'évaluation de *MongoDB* avec l'*entreprise5*, comme cité auparavant, nous avons deux méthodes, une méthode basée sur les utilisateurs et une autre basée sur les items. Nous commençons par la première et nous éliminons les utilisateurs qui n'ont pas évalué *MongoDB*. Ensuite nous cherchons les k-voisins de l'*entreprise5* qui ont évalué au moins quatre SGBD (parce que *Entreprise5* a évalué quatre SGBD). Le résultat obtenu montre que l'évaluation de l'*entreprise2* est la plus proche et nous pouvons prédire que l'*entreprise5* donnera une note de sept à *MongoDB* (voir le tableau 6.2).

Si nous utilisons la deuxième méthode basée sur les items, nous commençons tout d'abord par trouver les autres items évalués par *Entreprise5* qui sont les plus proche voisins à *MongoDB*. Ensuite, nous prédisons la note probable de *MongoDB* en fonction des évaluations de l'*entreprise5* pour les items voisins les plus proches de *MongoDB*. Le résultat montre que l'évaluation de *MongoDB* est plus proche de celle de *PostgreSQL*. Nous pouvons donc prédire que l'*entreprise5* donnera une note de six à *MongoDB* (voir le tableau 6.3).

	Oracle	PostgreSQL	MySQL	MSQL Server	MongoDB	Cassandra
Entreprise1	08		03	09		06
Entreprise2	09	07	05	07	07	02
Entreprise3	09			06		
Entreprise4	05	04		08	05	
Entreprise5	08	06	06		? 06	01

Tableau 6.3 – Filtrage collaboratif basé sur l'item

2.4.2 Systèmes de recommandation basés sur le contenu

Ce type est basé sur l'analyse des jeux de documents et/ou de descriptions d'items évalués précédemment par un utilisateur et construit un modèle ou un profil d'intérêts d'utilisateur en fonction des caractéristiques des objets évalués par cet utilisateur [156]. En se basant sur un ensemble de données, des algorithmes d'apprentissage automatique supervisés sont appliqués pour créer ce modèle d'utilisateur [11]. L'objectif est de recommander de nouveaux éléments intéressants qui sont similaires à ceux qui étaient utilisés antérieurement [5]. Le processus de recommandation consiste essentiellement à faire correspondre les attributs du profil utilisateur aux attributs d'un objet contenu. Le résultat est un jugement de pertinence qui représente le niveau d'intérêt de l'utilisateur pour cet objet [156]. L'idée est de calculer la similarité entre le contenu des caractéristiques associées aux éléments et les préférences des utilisateurs. L'efficacité de ce type s'appuie sur l'utilisation fréquente du système qui offre plus de pertinence des items.

2.4.3 Filtrage basé sur la connaissance

Ce type de systèmes s'appuie sur la sémantique des données (ontologies) [138] pour recommander des éléments basés sur des connaissances spécifiques de domaine au sujet de la manière dont certaines caractéristiques d'élément répondent aux besoins et aux préférences des utilisateurs et à la manière dont l'élément est utile pour l'utilisateur. Dans ces systèmes, une fonction de similarité évalue à quel point les besoins de l'utilisateur correspondent aux recommandations. Dans ce cas, le score de similarité peut être directement interprété comme l'utilité de la recommandation pour l'utilisateur. Les systèmes basés sur la connaissance tente à fonctionner mieux que d'autres au début de leur déploiement, mais s'ils ne sont pas équipés de composantes d'apprentissage, ils peuvent être dominés par d'autres méthodes [156]. Ce type utilise également le raisonnement pour générer une recommandation en effectuant des substitutions entre des attributs et des données similaires qui répondent aux exigences de l'utilisateur [181].

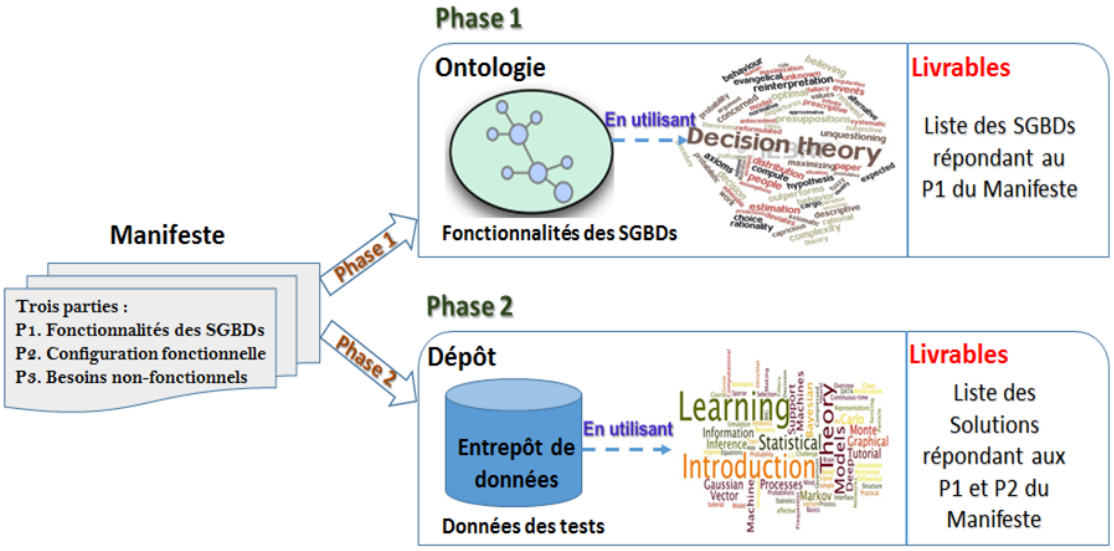


FIGURE 6.1 – Structure du système de recommandation

3 Composition de notre système de recommandation

Dans l'introduction de cette thèse, nous avons soulevé une problématique majeure liée au choix des solutions les plus adaptées à une application donnée. Nous croyons que les systèmes de recommandation peuvent aider les entreprises dans ce choix. Le scénario de recommandation dans notre contexte est basé essentiellement sur le manifeste fourni par le *déployeur*. Comme indiqué dans le chapitre 5, le manifeste représente la $\mathcal{BD}$ avec les caractéristiques liées au schéma, aux ensembles de données, à la charge de requêtes, à la plateforme, au système d'exploitation et à l'architecture de déploiement. Il serait intéressant d'avoir un système qui permet de recommander les meilleures solutions (plateforme et SGBD), qui répondent aux exigences de l'entreprise (manifeste). Par conséquence, la recommandation basée sur le contenu qui consiste à analyser la description des ressources afin de déterminer lesquelles sont susceptibles d'être utiles ou intéressantes (pour un *déployeur* donné) est la plus adaptée à notre problème.

Notre système de recommandation est basé sur deux phases, ainsi illustré dans la figure 6.1. (i) L'utilisation de notre ontologie des SBD pour choisir les meilleurs SGBD assurant les fonctionnalités ($\in DBMS_{features}$) demandées par le *déployeur*, (ii) La satisfaction de la configuration fonctionnelle ($\in Config$) et les exigences non fonctionnelles ($\in Metric$) en utilisant notre entrepôt de tests. Rappelons que les fonctionnalités, la configuration fonctionnelle et les métriques sont exprimées dans le manifeste. Pour chaque phase nous utilisons des techniques qui conviennent à la nature du problème : dans la première phase, nous faisons appel à la décision multicritère. Dans la deuxième phase, nous intégrons l'apprentissage automatique qui s'avère être mieux adapté aux fouilles de données. Dans la première phase, tous les SGBD sont candidats tandis qu'au deuxième phase, seuls ceux livrés dans la première phase.

n°	Critère C_j	Poids w_j	w_j normalisé
1	Stockage en mémoire	1	0.07
2	Données géospatiales	2	0.13
3	Agrégation de pipeline	5	0.33
4	Procédures stockées	4	0.27
5	Indexation plein texte	3	0.20

Tableau 6.4 – Critères et poids associés

3.1 Phase 1 : Approche basée sur la décision multicritère

Cette phase permet de trouver les systèmes garantissant la première partie du manifeste ($\in DBMS_{features}$). Ces systèmes doivent également assurer le maximum des exigences fournies par le manifeste. Il est à noter que notre problème est tout à fait semblable aux problèmes traités par la décision multicritère pour évaluer un certain nombre d'alternatives en termes d'un certain nombre de critères et enfin de choisir la meilleure alternative. Pour atteindre cet objectif, notre approche proposée est basée sur le couplage entre l'ontologie présentée auparavant et la méthode de la somme pondérée. L'idée est de construire une matrice de décision (A) à partir des instances de notre ontologie, tandis que les alternatives (A_i) représentent les SGBD, les critères C_j (les critères de vecteur C) représentent les fonctionnalités ($\in DBMS_{features}$) et la valeur de la performance a_{ij} prend 0 en cas d'absence de la fonctionnalité et 1 dans le cas contraire, et w_j désigne le poids relatif de l'importance du critère C_j. Par exemple, considérons le scénario suivant. un *déployeur* cherche un système garantissant les critères suivants : (i) gestion des données géospatiales, (ii) stockage en mémoire, (iii) utilisation des procédures stockées, (iv) beaucoup de requêtes de recherche de texte qui nécessitent une optimisation en utilisant l'indexage pour la recherche en mode plein texte et (v) l'agrégation Pipelines.
Afin de résoudre ce problème, on associe un poids à chaque critère en fonction des priorités exprimées par le *déployeur*. Un poids maximum pour le critère le plus prioritaire et un minimum pour le moins prioritaire. Par exemple, le *déployeur* a attribué le poids 5 au troisième critère (agrégation de pipeline), le poids 4 au quatrième critère (Procédure stockée), le poids 3 au cinquième critère (index plein texte), le poids 2 au deuxième critère (données géospatiales) et le poids 1 au premier critère (stockage en mémoire). Le tableau 6.4 résume le choix fait par le *déployeur*.

Nous appliquons la formule (1), où A est notre matrice de SGBD et C est notre vecteur de critères normalisé (voir la dernière colonne du tableau 6.4). Ensuite, nous classons les alternatives selon le score maximum du WSM et nous choisissons celle correspond à la valeur du score maximum.

3.2 Phase 2 : Approche basée sur l'apprentissage automatique

Nous avons tenté d'utiliser une technique de régression linéaire pour résoudre notre problème. Cependant, les résultats obtenus étaient médiocres en termes de prédiction. Ceci est dû au nombre de tests de SGBD stockés dans notre entrepôt de données qui n'est pas suffisamment grand pour la prédiction. Le manque d'un nombre suffisant d'instances de tests est la cause principale des mauvais résultats. Par exemple, dans notre cas, nous avons utilisé 40 tests pour Microsoft SQL Server et 10 tests pour Oracle.
Afin d'éviter le problème lié au nombre de tests, nous avons utilisé un algorithme basé sur la similarité entre le manifeste et les tests stockés dans l'entrepôt de données.

Notre algorithme se déroule selon les étapes suivantes (voir aussi les étapes du tableau 6.5) :

- **Etape 1 :** Analyser le manifeste exprimé par le *déployeur* des $\mathcal{BD}$ pour identifier la présence ou l'absence d'une dimension de l'entrepôt de données et de déterminer les dimensions concernées par le manifeste.
- **Etape 2 :** Extraire le fragment de données satisfaisant ces dimensions en utilisant les opérations classiques de l'entreposage de données.
- **Etape 3 :** Normaliser toutes les valeurs de toutes les dimensions en utilisant la formule (5).
- **Etape 4 :** Calculer la similarité entre le manifeste et chaque instance figurant dans le fragment de l'étape 2. Notons que chaque instance représente un test.
- **Etape 5 :** Sélectionner les meilleures propositions en se basant sur les résultats. En effet, les tests sont triés en fonction de la similarité des résultats pour chaque SGBD.
- **Etape 6 :** Le *déployeur* peut choisir son SGBD préféré en fonction des résultats obtenus, comme il peut croiser les résultats avec d'autres critères (tel que le prix).

4 Implémentation et expérimentation

Dans le but d'évaluer l'efficacité de notre système de recommandation et ses deux phases utilisant les approches présentées ci-dessus, nous avons expérimenté l'implémentation des deux approches. Pour bien comprendre le fonctionnement de notre système. Des scénarios de cas d'utilisations ont été joués concernant que la sélection des meilleurs SGBD et/ou plateformes. Le but est de recommander à chaque fois les meilleurs SGBD et/ou plateformes. Dans la suite, nous présentons notre prototype, l'environnement de développement, l'évaluation et l'analyse des résultats obtenus pour chaque phase.

<table>
<tr><th>Étape d'algorithme</th><th>Exemple</th></tr>
<tr><td>Étape 1
Entrée : Manifeste</td><td>

Organism My Company
Platform CPU: 2.8 Ghz -Memory: 768 Gbytes
DBMS ?Unknown
Data Set TPC-H datasets -Size : 800 GB
Mteric QphH

- Platform dimension
- DBMS dimension
- Dataset dimension
- Metrics dimension
</td></tr>
<tr><td>Étape 2
Input : DW_TEST</td><td>Sortie :
<table>
<tr><th>SGBD</th><th>Test</th><th>Taille</th><th>CPU</th><th>Mémoire</th><th>QphH</th></tr>
<tr><td rowspan="5">MS SQL Server</td><td>Test1</td><td>1000</td><td>2,8</td><td>1536</td><td>588831</td></tr>
<tr><td>Test2</td><td>3000</td><td>2,5</td><td>3072</td><td>725686</td></tr>
<tr><td>Test3</td><td>3000</td><td>2,5</td><td>3072</td><td>700392</td></tr>
<tr><td>Test4</td><td>3000</td><td>2,8</td><td>3072</td><td>461837</td></tr>
<tr><td>Test5</td><td>10000</td><td>2,8</td><td>4096</td><td>652239</td></tr>
<tr><td rowspan="5">Oracle</td><td>Test6</td><td>1000</td><td>1,5</td><td>64</td><td>9853</td></tr>
<tr><td>Test7</td><td>3000</td><td>2,88</td><td>512</td><td>198907</td></tr>
<tr><td>Test8</td><td>3000</td><td>3</td><td>1024</td><td>205792</td></tr>
<tr><td>Test9</td><td>10000</td><td>1,5</td><td>288</td><td>108099</td></tr>
<tr><td>Test10</td><td>30000</td><td>1,6</td><td>1024</td><td>156960</td></tr>
<tr><td rowspan="5">DB2</td><td>Test11</td><td>100</td><td>3,6</td><td>4</td><td>1894</td></tr>
<tr><td>Test12</td><td>300</td><td>3</td><td>32</td><td>10165</td></tr>
<tr><td>Test13</td><td>1000</td><td>1,7</td><td>32</td><td>20221</td></tr>
<tr><td>Test14</td><td>1000</td><td>1,9</td><td>32</td><td>26156</td></tr>
<tr><td>Test15</td><td>3000</td><td>2,6</td><td>16</td><td>38672</td></tr>
</table>
</td></tr>
<tr><td>Étape 3 et 4
Input : Le tableau ci-dessus avec les formules suivantes : 3, 5 et 6</td><td>
<table>
<tr><th>SGBD</th><th>Test</th><th>Taille</th><th>N1</th><th>CPU</th><th>N2</th><th>Mémoire</th><th>N3</th><th>QphH</th><th>Distance</th><th>N</th><th>S</th></tr>
<tr><td rowspan="5">MS SQL Server</td><td>Test1</td><td>1000</td><td>0,03</td><td>2,8</td><td>0,62</td><td>1536</td><td>0,37</td><td>588831</td><td>0,19</td><td>0,17</td><td>0,83</td></tr>
<tr><td>Test2</td><td>3000</td><td>0,10</td><td>2,5</td><td>0,48</td><td>3072</td><td>0,75</td><td>725686</td><td>0,59</td><td>0,52</td><td>0,48</td></tr>
<tr><td>Test3</td><td>3000</td><td>0,10</td><td>2,5</td><td>0,48</td><td>3072</td><td>0,75</td><td>700392</td><td>0,59</td><td>0,52</td><td>0,48</td></tr>
<tr><td>Test4</td><td>3000</td><td>0,10</td><td>2,8</td><td>0,62</td><td>3072</td><td>0,75</td><td>461837</td><td>0,57</td><td>0,50</td><td>0,50</td></tr>
<tr><td>Test5</td><td>10000</td><td>0,33</td><td>2,8</td><td>0,62</td><td>4096</td><td>1,00</td><td>652239</td><td>0,87</td><td>0,77</td><td>0,23</td></tr>
<tr><td rowspan="5">Oracle</td><td>Test6</td><td>1000</td><td>0,03</td><td>1,5</td><td>0,00</td><td>64</td><td>0,01</td><td>9853</td><td>0,64</td><td>0,57</td><td>0,43</td></tr>
<tr><td>Test7</td><td>3000</td><td>0,10</td><td>2,88</td><td>0,66</td><td>512</td><td>0,12</td><td>198907</td><td>0,10</td><td>0,09</td><td>0,91</td></tr>
<tr><td>Test8</td><td>3000</td><td>0,10</td><td>3</td><td>0,71</td><td>1024</td><td>0,25</td><td>205792</td><td>0,14</td><td>0,12</td><td>0,88</td></tr>
<tr><td>Test9</td><td>10000</td><td>0,33</td><td>1,5</td><td>0,00</td><td>288</td><td>0,07</td><td>108099</td><td>0,70</td><td>0,62</td><td>0,38</td></tr>
<tr><td>Test10</td><td>30000</td><td>1,00</td><td>1,6</td><td>0,05</td><td>1024</td><td>0,25</td><td>156960</td><td>1,13</td><td>1,00</td><td>0,00</td></tr>
<tr><td rowspan="5">DB2</td><td>Test11</td><td>100</td><td>0,00</td><td>3,6</td><td>1,00</td><td>4</td><td>0,00</td><td>1894</td><td>0,42</td><td>0,37</td><td>0,63</td></tr>
<tr><td>Test12</td><td>300</td><td>0,01</td><td>3</td><td>0,71</td><td>32</td><td>0,01</td><td>10165</td><td>0,20</td><td>0,18</td><td>0,82</td></tr>
<tr><td>Test13</td><td>1000</td><td>0,03</td><td>1,7</td><td>0,10</td><td>32</td><td>0,01</td><td>20221</td><td>0,55</td><td>0,49</td><td>0,51</td></tr>
<tr><td>Test14</td><td>1000</td><td>0,03</td><td>1,9</td><td>0,19</td><td>32</td><td>0,01</td><td>26156</td><td>0,46</td><td>0,41</td><td>0,59</td></tr>
<tr><td>Test15</td><td>3000</td><td>0,10</td><td>2,6</td><td>0,52</td><td>16</td><td>0,00</td><td>38672</td><td>0,22</td><td>0,19</td><td>0,81</td></tr>
<tr><td></td><td>Manifest</td><td>800</td><td>0,02</td><td>2,8</td><td>0,62</td><td>768</td><td>0,19</td><td></td><td>0,00</td><td>0,00</td><td>1,00</td></tr>
</table>
</td></tr>
<tr><td>Étape 5 et 6
Résultat :</td><td>
<table>
<tr><th>SGBD</th><th>QphH</th><th>S</th></tr>
<tr><td>MS SQL Server</td><td>588831</td><td>0,81</td></tr>
</table>
</td></tr>
</table>

Tableau 6.5 – Exemple de processus de notre système de recommandation

4.1 Outils et environnement de développement

Pour la mise en œuvre de notre prototype et la réalisations des expérimentations, nous avons utilisé une machine DELL PRECISION T1700 sous Windows 7 professionnel 64 bits. cette machine est dotée d'un processeur Intel Core i7-4770 de 3.40 Ghz et 8 Go de RAM. Nous avons utilisé les outils logiciels suivants :

- Le langage Java avec l'environnement de développement Eclipse. Ce choix a été motivé par les avantages qu'offre ce langage en termes de portabilité, de robustesse ainsi que la disponibilité de nombreuses bibliothèques ;
- Le SGBD Oracle version 12c pour concevoir l'entrepôt de données de tests. Ce choix a été principalement argumenté par la disponibilité de l'option OLAP offrant en particulier un moteur analytique OLAP, des espaces de travail et un gestionnaire d'espace de travail analytique (*Analytic Workspace Manager-AWM*) ;
- Le Système Protégé [37] pour construire l'ontologie, il offre une simple interface graphique et dispose des raisonneurs et la possibilité d'interroger l'ontologie à travers SPARQL. Il peut lire et sauvegarder des ontologies dans les formats : RDF(S), OWL, et XML Schéma.

4.2 Évaluation et analyse des résultats de la première phase

Afin de construire notre ontologie, nous avons utilisé *Protégé-OWL*. Il est très populaire dans le domaine du Web sémantique et très utilisé dans la recherche en informatique.

A=

A \ C	INM	SCX	CPX	FTX	GSD	PTX	MVW	CMP	PIP	SPR	TRG	MPR	REP
MySQL	1	1	1	1	1	0	0	1	1	1	1	0	1
PostgreSQL	0	1	1	1	1	1	1	1	1	1	1	0	1
Oracle	0	1	1	1	1	1	1	1	1	1	1	1	1
MongoDB	1	1	1	1	1	0	1	1	1	1	0	1	1
Cassandra	0	1	1	0	0	0	1	1	0	1	1	1	1
VoltDB	0	1	1	0	0	1	0	0	0	1	0	1	1
Redis	0	1	0	0	1	0	0	1	0	0	1	0	1
MSQL server	0	1	1	1	1	1	1	1	0	1	1	0	1
Neo4j	1	1	0	1	1	0	0	1	0	0	1	0	1
DynamoDB	0	1	1	0	0	0	0	1	0	1	1	1	1
Memcached	1	0	0	0	0	0	0	0	0	0	0	1	0

FIGURE 6.2 – Matrice des SGBD et ses fonctionnalités

37. http://protege.stanford.edu/

Nous avons instancié 11 SGBD de différentes classes dans notre ontologie : Oracle [38], MySQL [39], PostgreSQL [40], MongoDB [41], Cassandra [42], VoltDB [43], Redis [44], Microsoft SQL Server [45]. Neo4j [46], DynamoDB [47] et MemcacheDB [48]. La figure 6.2 montre un extrait des instances de notre ontologie incluant les fonctionnalités suivantes: INM (stockage en mémoire), SCX (Index secondaire), CPX (Index composé), FTX (Index plein texte) , GSD (Données géospatiales), PTX (Index partiel), MVW (Vue matérialisée), CMP (Compression), PIP (Pipelines), SPR (Procédure stockée), TRG (Déclencheur), MPR (MapReduce) et REP (Réplication).

Nous considérons le manifeste présenté dans la figure 6.3 avec ses trois parties. La première partie représente le même scénario précédent de la section 3.1 avec cinq critères et leur poids (comme dans le tableau 6.4). La deuxième partie et la troisième partie représentent respectivement la configuration fonctionnelle et les métriques (par exemple temps de réponse).

Pour compléter la matrice de décision de la figure 6.2, nous interrogeons notre ontologie des SBD en utilisant le langage de requête SPARQL pour chaque critère. Les figures 6.4 et 6.5 présentent deux requêtes concernant respectivement les critères du stockage en mémoire et de la gestion des données géospatielles. Ensuite, nous multiplions chaque colonne du tableau 6.2 par son poids. Par exemple, les valeurs de la colonne PIP sont multipliées par 0.33 (poids Pipelines). Puis nous mettons les résultats dans une matrice A^* (partie gauche de la figure 6.6). Enfin, nous comptons les valeurs de chaque ligne (SGBD) et nous trions les résultats de performance suivant leur score. Dans ce cas, les résultats montrent que les meilleures alternatives sont MySQL et MongoDB (parce qu'elles ont le score maximum de WSM qui est égal à 1). En outre, les solutions PostgreSQL et Oracle ont le même score de WSM qui est égal à 0.93, comme c'est illustré dans la figure 6.6.

Nous avons présenté notre proposition portant sur l'introduction d'outils mathématiques dans le processus de sélection de SGBD. Ensuite, nous avons montré un scénario de sélection d'un système selon les contraintes exprimées. Cependant, cette méthode dépend de la base de connaissances des instances. Le manque d'informations et de détails pour les SGBD rend ce processus incomplet. Ceci dit, nous pouvons conclure que notre modèle répond parfaitement en tenant en compte les systèmes instanciés.

38. http://docs.oracle.com/
39. https://www.mysql.fr/
40. http://www.postgresql.org/
41. http://www.mongodb.org/
42. http://cassandra.apache.org/
43. https://voltdb.com/
44. http://redis.io/
45. https://www.microsoft.com/fr-fr/server-cloud/products/sql-server/
46. http://neo4j.com/
47. http://aws.amazon.com/fr/dynamodb/
48. http://memcachedb.org/

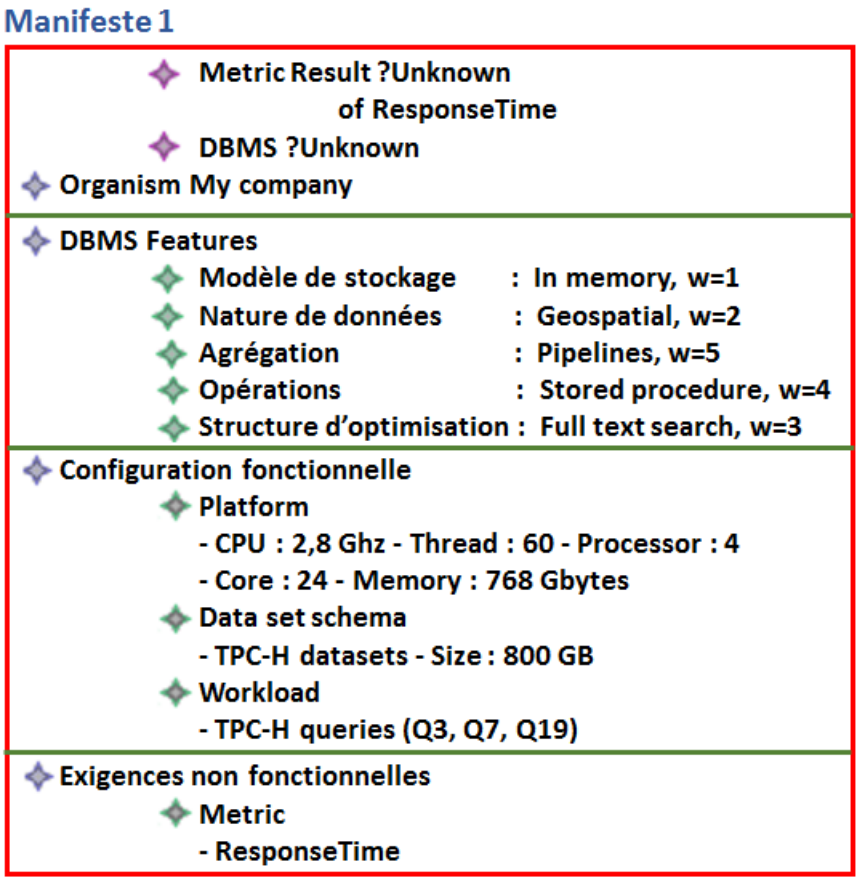


FIGURE 6.3 – Extrait du manifeste avec ses trois parties

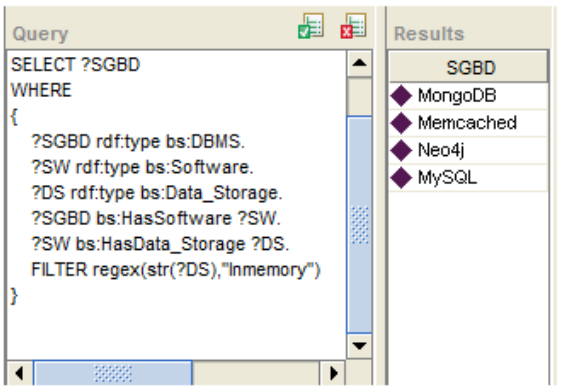


FIGURE 6.4 – Requête SPARQL du critère "stockage en mémoire"

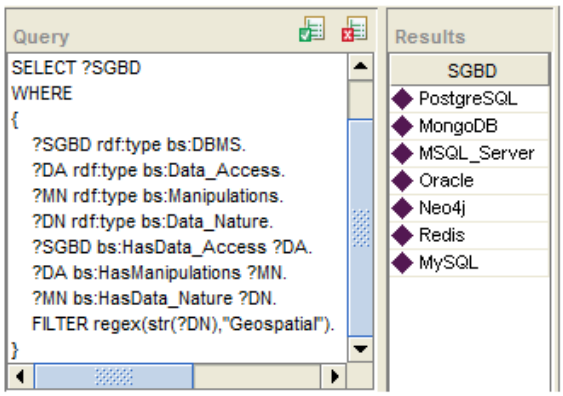


FIGURE 6.5 – Requête SPARQL récupérant les SGBD supportant les "données géospatielles"

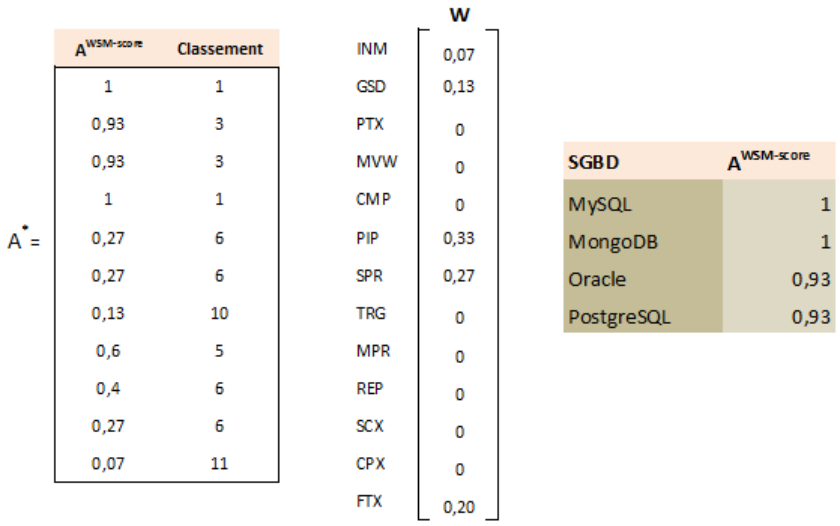


FIGURE 6.6 – Résultat de notre proposition basée sur WSM

4.3 Évaluation et analyse des résultats de la deuxième phase

Pour ce cas d'étude, nous considérons les données réelles de test disponibles sur le site Web de TPC du banc d'essai TPC-H [49]. Notons que TPC-H utilise un schéma de $\mathcal{BD}$ simple composé de huit tables : PART(P), PARTSUPP(PS), SUPPLIER(S), CUSTOMER(C), LINEITEM(L), NATION(N), REGION(R) et ORDERS(O) et la charge de requêtes se compose de 22 requêtes. Dans nos expérimentations, nous avons instancié des tests de quatre SGBD connus : Oracle, Microsoft SQL Server, DB2 et Sybase. Nous nous sommes basés sur le temps de réponse des requêtes dans le cas d'un seul flux. Ces données sont insérées dans notre dépôt via l'interface fournisseur (environ dix tests pour chaque SGBD). Deux cas d'études sont pris en considération.

4.3.1 Cas d'étude 1 : requêtes des bancs d'essai

Dans le premier cas d'étude, nous supposons que le *déployeur* cherche à estimer le temps de réponse ou d'autre métrique pour des requêtes standards telles que les requêtes de TPC-H. Pour ce cas, trois scénarios de manifeste sont pris en considération comme il est indiqué par le tableau 6.6. Le point d'interrogation indique la/les composante(s) recherchées.

4.3.1.1 Scénario 1 : pas d'inconnus Ce scénario consiste à comparer les résultats obtenus de la simulation avec notre approche par rapport aux résultats de l'expérimentation réelle pour le même test. En effet, nous avons utilisé dix (10) tests de Microsoft SQL Server avec le banc d'essai TPC-H (voir le tableau 6.7). Nous souhaitons comparer les temps de réponse des requêtes de TPC-H ($Q3$, $Q7$, $Q19$) obtenus pour les deux méthodes de tests en utilisant la même

49. http://www.tpc.org/tpch/results/tpch_perf_results.asp

	Jeux de données	Charge de requêtes	Plateforme	SGBD
Scénario 1	✓	✓	✓	✓
Scénario 2	✓	✓	✓	?
Scénario 3	✓	✓	?	?

Tableau 6.6 – Scénarios de l'étude expérimentale

SQL Server	Size	N_1	Q_3	Q_7	Q_{19}	CPU	N_2	Proc	N_3	Threads	N_4	Cores	N_5	Mémoire	N_6	TR_Q3	TR_Q7	TR_Q19	D	S
Test1	1000	0,00	1	1	1	2,2	0,00	2	0,00	88	0,16	44	0,33	512	0,00	3,80	2,50	1,7	1,1120	0,6186
Test2	1000	0,00	1	1	1	2,8	1,00	4	0,33	120	0,33	60	0,47	1536	0,29	4,70	2,80	2,3	0,3333	0,1854
Test3	1000	0,00	1	1	1	2,3	0,17	2	0,00	72	0,07	120	1,00	2048	0,43	7,40	3,50	3,3	1,0523	0,5854
Test4	3000	0,22	1	1	1	2,5	0,50	4	0,33	144	0,47	72	0,58	3072	0,71	12,00	7,40	5,5	0,8438	0,4694
Test5	3000	0,22	1	1	1	2,5	0,50	4	0,33	144	0,47	72	0,58	3072	0,71	13,60	6,90	5,1	0,8438	0,4694
Test6	3000	0,22	1	1	1	2,8	1,00	4	0,33	120	0,33	60	0,47	3072	0,71	22,90	12,10	7,3	0,5867	0,3263
Test7	10000	1,00	1	1	1	2,8	1,00	8	1,00	240	1,00	120	1,00	4096	1,00	40,50	24,00	23,4	1,7977	1,0000
Test8	10000	1,00	1	1	1	2,5	0,50	4	0,33	144	0,47	72	0,58	3072	0,71	42,00	25,00	21,9	1,2894	0,7173
Test9	10000	1,00	1	1	1	2,8	1,00	4	0,33	120	0,33	6	0,00	3072	0,71	65,40	37,00	44,3	1,2325	0,6856
Test10	10000	1,00	1	1	1	2,4	0,33	8	1,00	160	0,56	80	0,65	4096	1,00	108,90	143,40	575,9	1,6548	0,9205
Prediction	1000	0,00	1	1,00	1	2,8	1,00	4	0,33	60	0,00	60	0,47	1536	0,29	5,57	3,32	2,73	0,0000	0,19
Exp. réelle	1000					2,8		4		60		60		1536		5,4	2,8	2,5		

Tableau 6.7 – Dix tests de Microsoft SQL server avec TPC-H

plateforme et la même taille de la $\mathcal{BD}$ (voir les deux dernières lignes du tableau 6.7).

La figure 6.7 montre que les valeurs réelles sont très proches des valeurs approximatives pour les trois requêtes. D'abord, l'écart entre les résultats de l'expérimentation réelle et la simulation varie entre 4% et 16% (96%=5.4/5.57 et 84%=2.8/3.32). Cela vient de notre dépendance à l'égard de certaines dimensions, en particulier les dimensions ayant des valeurs numériques. Nous pensons si on introduit toutes les dimensions notamment celles ayant des valeurs catégorielles et l'intégration de nouvelles techniques de calcul de similarité à savoir la similarité entre les systèmes d'exploitation et entre les SGBD, le résultat devient plus proche de la réalité.

4.3.1.2 Scénario 2 : SGBD inconnu La lecture du manifeste de la figure 6.8 montre que le *déployeur* souhaite connaître le temps de réponse des requêtes de TPC-H ($Q3$, $Q7$, $Q19$) en fonction de la plateforme et de l'ensemble de données. En se référant au résultat lié à la métrique temps de réponse, nous pouvons recommander une liste de SGBD qui correspondent aux besoins du *déployeur*. Puisque parmi les quatre SGBD stockés dans l'entrepôt, Oracle et Mir-

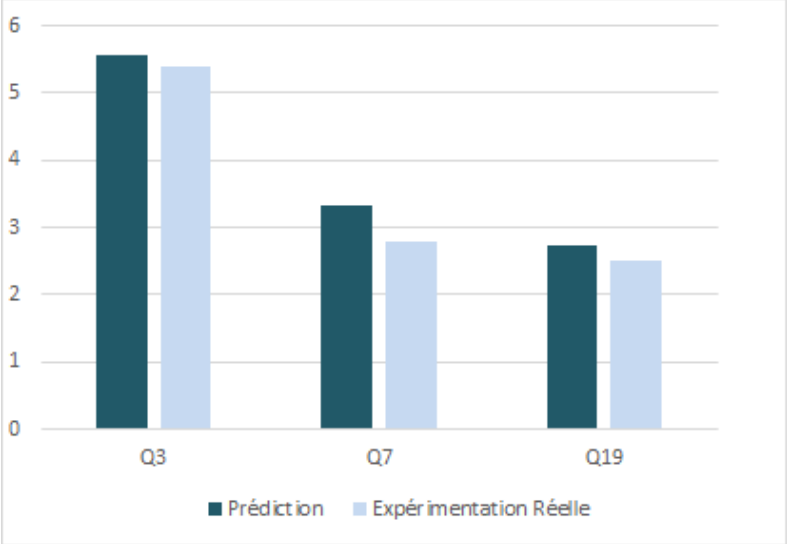


FIGURE 6.7 – Résultats correspondant au scénario 1

	Oracle	MS SQL Server	DB2	Sybase
$Q3$	6.80	5.40	102.50	35.50
$Q7$	34.30	2.80	677.80	37.50
$Q19$	50.30	2.50	262.20	19.30
Similarité (S)	0.74	0.81	0.48	0.49
Distance (D)	0.26	0.19	0.52	0.51

Tableau 6.8 – Temps de réponse réel pour $Q3$, $Q7$, $Q19$ de quatre SGBD : Oracle, Microsoft SQL Server, DB2 et Sybase

crosoft SQL Server figurent également dans les instances de notre ontologie, cela nous permet alors de nous focaliser uniquement sur ces deux SGBD.

Les tableaux 6.8 et 6.9 présentent respectivement les valeurs réelles et les valeurs approchées du temps de réponse des requêtes $Q3$, $Q7$, $Q19$ avec Microsoft SQL Server, Oracle, DB2 et Sybase. Ainsi, selon les résultats obtenus, nous pouvons trier les SGBD. En première position, nous trouvons Microsoft SQL Server qui montre des performances de vitesse (temps de réponse) $Q3 = 6.42sec$, $Q7 = 3.33sec$ et $Q19 = 2.97sec$. En deuxième position, nous trouvons Oracle avec un temps de réponse $Q3 = 8.56sec$, $Q7 = 43.21sec$ et $Q19 = 63.37sec$. DB2 et Sybase montrent des temps de réponse très élevés par rapport à Microsoft SQL Server et Oracle, en plus Sybase a prouvé sa performance par rapport à Oracle pour la requête $Q19$ (= 29.14 sec). Notons que nous avons utilisé ici une simple méthode pour calculer la valeur approchée (valeur approchée = valeur réelle * $(1 + \alpha)$ où α représente la distance D). Par conséquent, suite aux résultats obtenus, nous pouvons recommander Microsoft SQL Server pour répondre au manifeste présenté dans la figure 6.8.

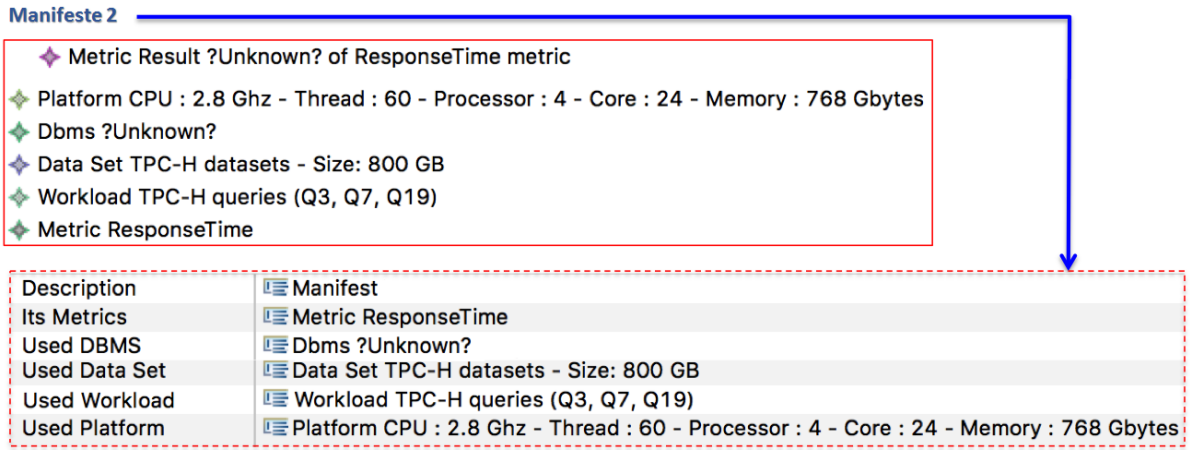


FIGURE 6.8 – Extrait du manifeste correspondant au scénario 2

	Oracle	MS SQL Server	DB2	Sybase
$Q3$	8.56	6.42	155.80	53.60
$Q7$	43.21	3.33	1030.25	56.62
$Q19$	63.37	2.97	398.54	29.14

Tableau 6.9 – Temps de réponse approché pour $Q3$, $Q7$, $Q19$ de quatre SGBD : Oracle, Microsoft SQL Server, DB2 et Sybase

4.3.1.3 Scénario 3 : SGBD et plateforme inconnues Ce scénario consiste à chercher à la fois un SGBD et une plateforme. Son manifeste est représenté par la figure 6.9.

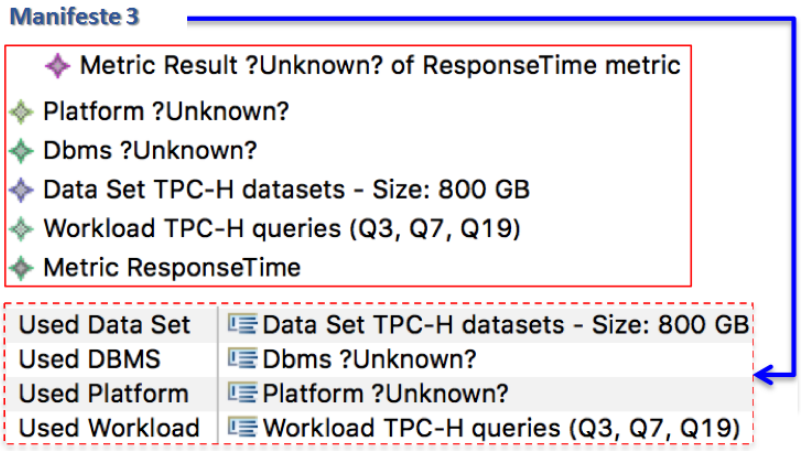


FIGURE 6.9 – Extrait du manifeste correspondant au scénario 3

Supposons que ce *déployeur* utilise la même configuration utilisée dans le scénario 2, à l'exception de la plateforme qui est manquante. Nous tenons à préciser que dans le scénario 3, le *déployeur* ne néglige pas la dimension de la plateforme, mais il cherche une plateforme et un SGBD qui répondent parfaitement au manifeste.

Le tableau 6.10 représente les résultats obtenus en termes de temps de réponse des requêtes $Q3$, $Q7$ et $Q19$. Ces temps de réponse sont classés en fonction de SGBD et les configurations de plateforme. Nous pouvons constater que Microsoft SQL Server est le meilleur SGBD en fonction des temps de réponse calculés. De plus, ces résultats sont liés à la configuration de la plateforme suivante (CPU : 2.8 GHz - Processeur : 4 - Threads : 120 - Cores : 60 - Mémoire : 1536 GB).

4.3.2 Cas d'étude 2 : Requêtes des déployeurs

La lecture du manifeste de la figure 6.10 montre que le *déployeur* souhaite connaître le temps de réponse de la requêtes (Qb du schéma S présenté dans le chapitre 4 (voir la figure 4.7) en fonction de la plateforme et de l'ensemble de données. En outre, en se référant au résultat lié à la métrique temps de réponse, nous pouvons recommander une liste de SGBD qui correspondent aux besoins du *déployeur*. Notons que dans ce cas d'étude, Qb est une requête exprimée par le *déployeur*, ce qui nous conduit à calculer la similarité entre les requêtes.

Avant de détailler cette étude de cas, nous recherchons d'abord les requêtes TPC-H compatibles avec la requête Qb. Étant donné que l'outil PLASTIC traite essentiellement des requêtes simples (sélection, projection et opérations de jointure). Puisque NT de Qb = 2, nous ne prenons pas en compte les requêtes Q1, Q2, Q3, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q18, Q20 et Q21 qui

	Oracle	Microsoft SQL Server	DB2	Sybase
CPU	1.3	2.8	1.9	2.8
Processeur	64	4	8	2
Threads	64	120	32	4
Cores	64	60	16	4
Mémoire	256	1536	32	16
Q3	6.86	4.84	27.84	1529.45
Q7	34.64	2.88	153.40	613.96
Q19	50.80	2.36	166.87	502.04
Similarité	0.99	0.97	0.98	0.93
Distance	0.01	0.03	0.02	0.07

Tableau 6.10 – SGBD et plateformes sélectionnées sur la base des temps de réponse des requêtes $Q3$, $Q7$, $Q19$

ont respectivement comme NT (1, 5, 3, 6, 1, 5, 7, 6, 4, 3, 3, 5, 4) car ils sont différents de 2. Quant à Q15 représente la création de la vue et ne correspond pas à notre exemple (sélection, projection et jointure).

	NT	NJP	DS	Compatible
Qb	2	1	($T1$, $T2$)=(1, 1)	Yes
Q4	2	1	(L,O)=(1, 1)	Yes
Q12	2	1	(L,O)=(1,1)	Yes
Q13	2	1	(C,O)=(1,1)	Yes
Q14	2	1	(P,L)=(1,1)	Yes
Q16	2	1	(P,PS)=(1,1)	Yes
Q17	2	1	(P,L)=(1,1)	Yes
Q19	2	1	(P,L)=(1,1)	Yes
Q22	2	1	(C,O)=(1,1)	Yes

Tableau 6.11 – Similarité entre *Qb* et les requêtes de TPC-H

Nous présentons dans le tableau 6.11 une comparaison entre le vecteur de caractéristiques de la requête *Qb* et chaque vecteur de caractéristique des requêtes restantes ($Q4$, $Q12$, $Q13$, $Q14$, $Q16$, $Q17$, $Q19$, $Q22$). Ces requêtes sont instanciées avec leurs fonctionnalités dans notre entrepôt de données. Nous constatons que toutes les requêtes sont compatibles avec *Qb* (égalité totale pour NT, NJP et DS).

Dans la deuxième étape, nous calculons la distance entre *Qb* et chaque requête acceptée dans l'étape précédente en fonction des tables impliquées dans la requête. Pour cela, nous avons utilisé l'équation (7) en tenant en compte les tailles effectives de $T1$ et $T2$ du schéma S et les

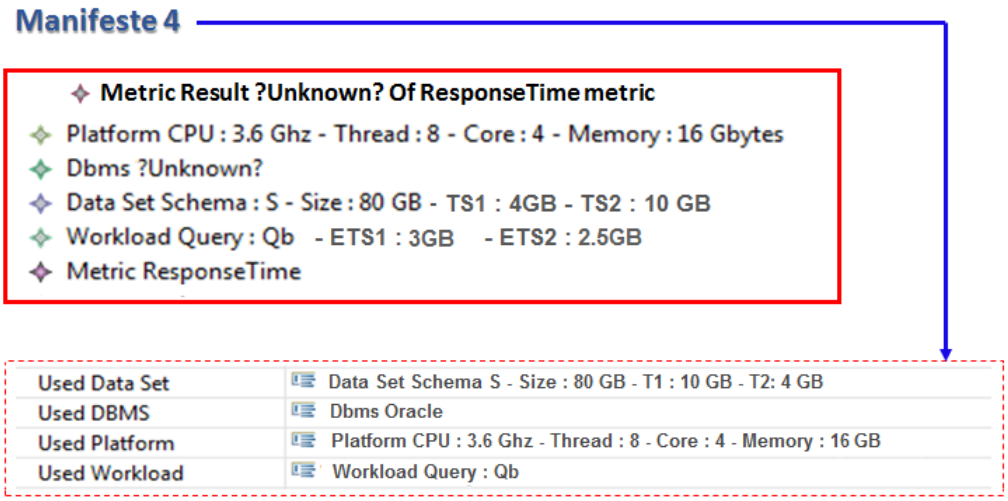


FIGURE 6.10 – Extrait du manifeste correspondant au deuxième cas d'étude

Table	Taille (MB)	Table	Taille (MB)
REGION (R)	$4 * 10^{-4}$	PART (P)	30
NATION (N)	$2. * 10^{-3}$	PARTSUPP (PS)	110
SUPPLIER (S)	2	ORDERS (O)	149
CUSTOMER (C)	26	LINEITEM (L)	641

Tableau 6.12 – Taille des tables de TPC-H (SF=1)

tailles effectives des tables de chaque requête ($Q4$, $Q12$, $Q13$, $Q14$, $Q16$, $Q17$, $Q19$ et $Q22$) comme c'est montré dans le tableau 6.12.

Soient que x_1 et y_1 les tailles effectives des tables $T1$ et $T2$ respectivement de la requête Qb. Soient x_2, y_2 les tailles effectives des tables d'une requête Q parmi ($Q4$, $Q12$, $Q13$, $Q14$, $Q16$, $Q17$, $Q19$, $Q22$). En utilisant l'équation (7) pour calculer la distance, nous obtiendrons :

$$Distance(Qb, Q) = \frac{|\, x_1 - x_2 \,|}{max(x_1, x_2)} + \frac{|\, y_1 - y_2 \,|}{max(y_1, y_2)} \tag{8}$$

Si nous prenons les deux parties de cette somme :

$$\frac{|\, x_1 - x_2 \,|}{max(x_1, x_2)} = 1 - \frac{x_2}{x_1} \;\; (si \;\; x_1 > x_2) \;\; ou \;\; = 1 - \frac{x_1}{x_2} \;\; (si \;\; x_1 < x_2). \tag{9}$$

$$\frac{|\, y_1 - y_2 \,|}{max(y_1, y_2)} = 1 - \frac{y_2}{y_1} \;\; (si \;\; y_1 > y_2) \;\; ou \;\; = 1 - \frac{y_1}{y_2} \;\; (si \;\; y_1 < y_2). \tag{10}$$

Le but est de choisir les requêtes permettant d'avoir des distances minimales (≈ 0). Par conséquent, le problème revient à résoudre les deux équations :

Top-3	SGBD	CPU	Threads	Cores	Mem	Q16	Ref	Sim	Qb
1	Oracle	3.33	12	24	192	1.72	[164]	0.06	1.82
2	SQL Server	3.3	4	8	10	4.4	TPC	0.09	4.79
3	DB2	3.6	4	2	4	63.4	TPC	0.02	64.66

Tableau 6.13 – Résultats du manifeste 4

$$x_1 \approx x_2. \quad et \quad y_1 \approx y_2. \tag{11}$$

Il s'agit de définir les tailles effectives des tables impliquées dans la requête en se basant sur les tailles effectives de la requête Qb. Rappelons que la taille effective d'une table représente la multiplication de la taille réelle de cette table et le facteur de sélectivité. Selon les instances stockées dans notre entrepôt de données notamment la dimension *Dim_Query*, nous avons trouvé que les tailles effectives des tables impliquées dans la requête $Q16$ pour $SF = 100$ (Taille effective de P = Facteur de sélectivité * SF*30= 2.7GB, Facteur de sélectivité= 0.09157 et taille effective de PS=Facteur de sélectivité*110*100=2.5GB, Facteur de sélectivité=0.0228928) vérifient les deux dernières équations, sachant que comme illustré dans la figure 6.10, la taille effective de $ETS1$= 3GB et de $ETS2$=2.5GB. En conséquent, la requête Q_{16} est la plus similaire à Qb. En effet, nous utilisons $Q16$ pour prédire le temps de réponse de Qb.

Le tableau 6.13 représente les résultats obtenus liés au temps de réponse de Qb en se basant sur le temps de réponse de $Q16$ avec Microsoft SQL Server, Oracle et DB2 et sur les tests de TPC-H et d'autres réalisés dans notre laboratoire. D'ailleurs, selon les résultats obtenus, nous pouvons trier les SGBD trouvés. En première position, nous trouvons Oracle qui montre des performances de rapidité (temps de réponse) Qb = 1,82 sec, (Temps approximatif = Temps réel * (1+ distance)). Dans la deuxième positions, nous trouvons Microsoft SQL Server avec Qb = 4.79. Cependant, le temps approximatif de Qb de DB2 est très élevé, bien que son test soit le plus semblable à celui de notre manifeste par rapport aux tests d'Oracle et de Microsoft SQL Server. Par conséquent, nous pouvons recommander Oracle pour satisfaire ce manifeste.

5 Conclusion

Dans le but de répondre parfaitement à notre problématique, nous avons utilisé notre dépôt contenant les environnements de test et aussi notre ontologie des systèmes de base de données (SBD). Pour bien bénéficier de ces deux éléments, nous avons proposé dans ce chapitre un système de recommandation. Cela représente une aide de décision aux *déployeurs* dans le choix de leurs solutions (SGBD et plateforme). Notre système contient deux phases : la première phase permet d'interroger l'ontologie des SBD avec l'utilisation de la méthode de la somme pondérée. La deuxième phase permet d'exploiter l'entrepôt de données de tests avec l'usage de

l'apprentissage automatique en se basant sur la similarité. En effet, nous avons essayé de fédérer la communauté de $\mathcal{BD}$ sur l'importance des données de test disponibles et de les motiver à construire un dépôt qui peut jouer le rôle d'un entrepôt de données de test. Nous avons également détaillé les aspects liés à l'implémentation et l'environnement d'expérimentation. Par la suite, nous avons présenté quelques cas d'étude en analysant les résultats obtenus pour montrer l'efficacité de notre système de recommandation.

Chapitre 7

Conclusion générale et Perspectives

Sommaire

1 **Conclusion** **163**

2 **Perspectives** **165**

1 Conclusion

Les systèmes d'information industriels sont devenus de plus en plus complexes suite à l'hétérogénéité et la diversité de leurs composants matériels et logiciels. La maîtrise de cette complexité nécessite des activités interdisciplinaires de l'ingénierie système allant de la conception à la vérification et la validation. Ces systèmes requièrent des tests d'analyses, dédiés à des métriques spécifiques, qui peuvent être effectués à différentes phases de développement afin de conclure sur un choix de conception et/ou une décision prise et éventuellement la comparer avec d'autres possibilités concurrentes. Récemment, les communautés de tests s'intéressent de plus en plus aux environnements de tests dans un but de reproductibilité pour la comparaison, la transparence et pour la capitalisation du savoir-faire. Dans cette perspective, cette thèse s'intéresse aux données de tests, leurs caractéristiques ainsi que leurs environnements afin de les mettre au service de systèmes de recommandation pour le déploiement de systèmes de base de données. Compte tenu de l'explosion des volumes de données et la diversification des applications, divers systèmes de gestion de bases de données (SGBD) et de plateformes d'exécution sont proposés. En effet, une des décisions primordiales, au moment du déploiement d'une nouvelle base de données (ou de la migration d'un système existant suite à son évolution) est la sélection des SGBD et/ou des plateformes les plus appropriés par rapport aux différentes exigences comme le temps de réponse, la consommation énergétique, la sécurité, etc. En se basant sur une ou plusieurs métriques, les tests permettent alors d'évaluer un éventuel choix de SGBD en quantifiant les métriques vis-à-vis la configuration du système en termes de schéma, de charge de requêtes, de plateforme d'exécution, de structures d'optimisation, etc. Les résultats issus de ces tests pourraient ainsi être comparés aux exigences fixées afin de prendre une décision. Cependant, la solution naïve est très coûteuse vue qu'elle nécessite une utilisation exhaustive de toutes les plateformes. pour faire face à cette problématique, nous avons étudié les systèmes de recommandation, les entrepôts de données et les ontologies qui ont été appliquées dans de nombreux domaines de manipulation de quantité énorme de données historiques. Cette thèse a donné lien à trois axes essentiels pour répondre à l'archivage, la reproductibilité et la réutilisation des tests.

1.1 Analyse des solutions existantes pour le problème de sélection des SGBD

Nous avons élaboré un état de l'art pour mettre l'accent sur les Causes de problème de sélection du SGBD et de plateforme provenant de l'évolution de cycle de vie. Nous avons présenté trois principales évolutions. La revue de la littérature concernant le problème de sélection des SGBD a montré la croissance de la complexité de ce problème. Nous avons distingué quatre principaux types de solutions. Les solutions basées sur les résultats des tests (test matériel de toutes les plateformes et la simulation via les modèles de coût), les solutions basées sur le retour de l'expérience, les solutions basées sur la classification des SGBD et la décomposition

fonctionnelle des SGBD pour la réduction de l'espace de recherche de notre problème. Par la suite, nous avons discuté les lacunes de chaque type de solutions en montrant que ces solutions traitent le problème évoqué de manière partielle. Par conséquent, nous avons visé d'expliciter les composantes des solutions basées sur les résultats des tests et de celles basées sur la classification des SGBD. Notre choix est motivé par l'existence d'une mine de données de tests, l'absence de méthodologie pour les réutiliser et la diversité des fonctionnalités des systèmes de bases de données notamment les SGBD et les plateformes. Cette explicitation nous a permis de définir les composantes de l'environnement de tests et de les exploiter dans le but de proposer une solution qui consiste à recommander des systèmes répondant aux exigences des *déployeurs* selon des indicateurs numériques et avec un minimum de coût.

1.2 Conception d'une ontologie des systèmes de base de données

Suite à la diversité et la complexité des SBD, les *déployeurs* rencontrent de plus en plus des difficultés pour répondre efficacement et rapidement aux besoins. A cet effet, nous avons fait un recensement des fonctionnalités techniques offertes par les SBD, en se basant sur le fonctionnement interne, les fonctionnalités communes et non communes des systèmes de base de données. Nous avons souligné l'absence d'une base de connaissance représentant les systèmes de gestion de bases de données et leurs fonctionnalités. Par conséquent, nous avons développé une ontologie du domaine pour décrire de manière hiérarchique tous les concepts et propriétés d'un système de bases de données. L'ontologie proposée nous a permis d'instancier une vingtaine de SGBD.

1.3 Développement d'un dépôt de données de tests

Dans la deuxième contribution, nous avons présenté un dépôt persistant les données des environnements de test. Ce dépôt permet d'ajouter des nouveaux tests et/ou de chercher des tests suivant la configuration fonctionnelle et les exigences non fonctionnelles fournies par les entreprises. Ce dépôt a été implémenté sous format d'un entrepôt de données. Il est structuré en plusieurs tables de dimensions : SGBD, plateforme, système d'exploitation, architecture de déploiement, jeux de données, requêtes, algorithmes et métriques et une table des faits pour stocker tous les éléments des tests et ainsi les résultats.

1.4 Constitution d'un système de recommandation

La troisième contribution permet d'estimer le résultat d'un test à travers un système de recommandation. Ce système s'appuie sur le dépôt des données de tests et sur l'ontologie des fonctionnalités des SBD. La prise de la bonne décision pour choisir une meilleure solution se déroule en deux phases. La première phase est consacrée à l'utilisation de l'ontologie des

SBD en appliquant les techniques de la décision multicritères, en particulier le modèle de la somme pondérée pour trouver le SGBD supportant le maximum de contraintes exprimées par le *déployeur*. La deuxième phase est consacrée à l'exploitation de l'entrepôt de données de tests en utilisant les techniques de l'apprentissage automatique, et en particulier la similarité pour prédire des tests.

2 Perspectives

De nombreuses perspectives concernent l'extension de notre système de recommandation peuvent être envisagées. Dans cette section, nous présentons succinctement celles qui nous paraissent être les plus intéressantes.

2.1 Validation de l'ontologie

Bien évidemment, nous avons conçu une ontologie pour les systèmes de base de données et en particulier les fonctionnalités des systèmes de gestion de bases de données. La validation de cette ontologie a été effectuée par les membres de notre équipe pour répondre à des besoins particuliers concernant l'usage dans des différents travaux. Nous prévoyons d'instancier tous les SGBD et nous envisageons de faire appel à des experts du domaine des bases de données pour sa validation, sa standardisation et son intégration dans le Common Warehouse Metamodel (CWM). Nous prévoyons aussi d'intégrer notre ontologie avec des systèmes de gestion des bases de données relationnelles comme Oracle Semantic Store1 ou Virtuoso qui offrent un stockage, une manipulation et un raisonnement efficaces et supportent plusieurs formalismes (RDF, OWL, etc) tout en assurant les propriétés ACID. Dans cette thèse, nous avons utilisé les ontologies pour expliciter le sens de nos dimensions de test non fonctionnels. Par contre le mécanisme de raisonnement offert par les ontologies n'est pas utilisé. Vu le lien existant à la fois entre les supports de stockage et les supports de traitement, l'utilisation de raisonnement pourrait proposer des solutions de substitution ou de remplacement de matériel.

2.2 Diversification de l'étude expérimentale

Notre système de recommandation représente un premier pas pour choisir une meilleure solution (SGBD, plateforme, etc.) dans la phase de déploiement des bases de données. En effet, nous croyons que son utilisation dans différents cas va nous permettre d'identifier des relations directes entre le manifeste et les dimensions de l'entrepôt de données. L'idée envisageable est de savoir l'impact des paramètres du manifeste sur les résultats des tests afin de les intégrer comme des règles de raisonnement dans l'ontologie proposée.

2.3 Intégration de la similarité catégorielle

Dans le système de recommandation, nous sommes focalisés sur la similarité entre les items numériques (taille de mémoire, fréquence de CPU, taille de la base de données, etc). A court terme, nous prévoyons d'intégrer la similarité entre des objets catégoriels. Nous allons étendre notre ontologie pour nous permettre de calculer la similarité entre les SGBD (Oracle, MongoDB), de calculer la similarité entre les systèmes d'exploitation (Windows, Linux), de claculer la similarité entre les structures d'optimisation (index, vues matérialisés). Cette extension va nous permettre d'enrichir notre algorithme et d'estimer des tests pour des SGBD qui ne sont jamais testés.

2.4 Approfondissement des techniques de la décision multicritère

En cas de l'enrichissement de l'ontologie par d'autres règles de raisonnement et d'extraire les dépendances entre les fonctionnalités des SGBD, il serait intéressant d'utiliser d'autres méthodes de la décision multicritère. Nous envisageons de faire une comparaison entre les résultats obtenus en consolidant avec un nombre important de scénarios afin de connaitre les relations entre les besoins et les systèmes. Ce type de connaissance consiste à élaborer des nouvelles classifications basées sur l'usage et le profil utilisateur et recommande les fondateurs des systèmes de produire des SGBD selon les profils. En effet, le coût d'acquisition, de formation des SGBD commerciaux et le temps d'installation et de maintenance peuvent être diminués.

2.5 Automatisation de la lecture des données de tests

Nous envisageons de développer des outils pour la lecture et l'apprentissage des données de tests à partir des articles scientifiques en exploitant des techniques issues de traitement automatique du langage naturel. Ces outils nous permettent d'extraire automatiquement les dimensions (avec leurs propriétés) et les résultats impliquées dans les tests.

Bibliographie

[1] Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, Magdalena Balazinska, Philip A. Bernstein, Michael J. Carey, Surajit Chaudhuri, Jeffrey Dean, AnHai Doan, Michael J. Franklin, Johannes Gehrke, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, H. V. Jagadish, Donald Kossmann, Samuel Madden, Sharad Mehrotra, Tova Milo, Jeffrey F. Naughton, Raghu Ramakrishnan, Volker Markl, Christopher Olston, Beng Chin Ooi, Christopher Ré, Dan Suciu, Michael Stonebraker, Todd Walter, and Jennifer Widom. The beckman report on database research. *Commun. ACM*, 59(2):92–99, 2016.

[2] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, C. Erwin, Eduardo F. Galvez, M. Hatoun, Anurag Maskey, Alex Rasin, A. Singer, Michael Stonebraker, Nesime Tatbul, Ying Xing, R. Yan, and Stanley B. Zdonik. Aurora: A data stream management system. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, page 666, 2003.

[3] Daniel J Abadi, Samuel R Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980. ACM, 2008.

[4] Veronika Abramova and Jorge Bernardino. Nosql databases: Mongodb vs cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, pages 14–22. ACM, 2013.

[5] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[6] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling multidimensional databases. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K.*, pages 232–243, 1997.

[7] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 496–505, 2000.

[8] Youcef Aklouf, Guy Pierra, Yamine Aït Ameur, and Habiba Drias. PLIB ontology for B2B electronic commerce. In *Enhanced Interoperable Systems. Proceedings of the 10th ISPE International Conference on Concurrent Engineering (ISPE CE 2003), July 26-30, 2003, Madeira, Portugal*, pages 269–278, 2003.

[9] Ioannis Alagiannis, Manos Athanassoulis, and Anastasia Ailamaki. Scaling up analytical queries with column-stores. In *Proceedings of the Sixth International Workshop on Testing Database Systems*, page 8. ACM, 2013.

[10] Hamdan O Alanazi, Abdul Hanan Abdullah, and Moussa Larbani. Dynamic weighted sum multi-criteria decision making: Mathematical model. *International Journal of Mathematics and Statistics Invention*, 1:16–18, 2013.

[11] Haifa Alharthi and Diana Inkpen. Content-based recommender system enriched with wordnet synsets. In *Computational Linguistics and Intelligent Text Processing - 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part II*, pages 295–308, 2015.

[12] Frédéric Andrès, Michel Couprie, and Yann Viémont. A multi-environment cost evaluator for parallel database systems. In *Proceedings of the Second International Symposium on Database Systems for Advanced Applications*, pages 126–135. World Scientific Press, 1991.

[13] Alexander A. Anisimov. Review of the data warehouse toolkit: the complete guide to dimensional modeling (2nd edition) by ralph kimball, margy ross. john wiley & sons, inc. 2002. *SIGMOD Record*, 32(3):101–102, 2003.

[14] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: a stream data management benchmark. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 480–491. VLDB Endowment, 2004.

[15] Melyssa Barata, Jorge Bernardino, and Pedro Furtado. Ycsb and tpc-h: Big data and decision support benchmarks. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 800–801. IEEE, 2014.

[16] Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. Cost-model oblivious database tuning with reinforcement learning. In *Database and Expert Systems Applications - 26th International Conference,*

DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part I, pages 253–268, 2015.

[17] Daniel Bausch, Ilia Petrov, and Alejandro P. Buchmann. Making cost-based query optimization asymmetry-aware. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware, DaMoN 2012, Scottsdale, AZ, USA, May 21, 2012*, pages 24–32, 2012.

[18] Sean Bechhofer. Owl: Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

[19] Catherine M. Beise. Revisiting database resource choice: A framework for DBMS course tool selection. In *Connecting the Americas. 12th Americas Conference on Information Systems, AMCIS 2006, Acapulco, México, August 4-6, 2006*, page 266, 2006.

[20] L. Bellatreche. Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique des entrepôts de données". Phd. thesis, Université de Clermont-Ferrand II, December 2000.

[21] Ladjel Bellatreche, Soumia Benkrid, Alain Crolotte, Alfredo Cuzzocrea, and Ahmad Ghazal. The f&a methodology and its experimental validation on a real-life parallel processing database system. In *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2012, Palermo, Italy, July 4-6, 2012*, pages 114–121, 2012.

[22] Ladjel Bellatreche and Kamel Boukhalfa. Yet another algorithms for selecting bitmap join indexes. In *Data Warehousing and Knowledge Discovery, 12th International Conference, DAWAK 2010, Bilbao, Spain, August/September 2010. Proceedings*, pages 105–116, 2010.

[23] Ladjel Bellatreche, Kamel Boukhalfa, and Zaia Alimazighi. Simulph. d.: A physical design simulator tool. In *DEXA*, pages 263–270. Springer, 2009.

[24] Ladjel Bellatreche, Kamel Boukhalfa, and Mukesh K. Mohania. Pruning search space of physical database design. In *Database and Expert Systems Applications, 18th International Conference, DEXA 2007, Regensburg, Germany, September 3-7, 2007, Proceedings*, pages 479–488, 2007.

[25] Ladjel Bellatreche, Salmi Cheikh, Sebastian Breß, Amira Kerkad, Ahcène Boukhorca, and Jalil Boukhobza. How to exploit the device diversity and database interaction to propose a generic cost model? In *Proceedings of the 17th International Database Engineering & Applications Symposium*, pages 142–147. ACM, 2013.

[26] Ladjel Bellatreche, Alfredo Cuzzocrea, and Soumia Benkrid. *F&a*: A methodology for effectively and efficiently designing parallel relational data warehouses on heterogenous database clusters. In *DaWak*, pages 89–104, 2010.

[27] Ladjel Bellatreche and Amira Kerkad. Query interaction based approach for horizontal data partitioning. *IJDWM*, 11(2):44–61, 2015.

[28] Soumia Benkrid. *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données : application aux plateformes parallèles. (Deployment, full phase in the data warehouse life cycle : application to parallel platforms)*. PhD thesis, École nationale supérieure de mécanique et d'aérotechnique, Chasseneuil-du-Poitou, Poitiers, France, 2014.

[29] Nabila Berkani, Ladjel Bellatreche, and Selma Khouri. Towards a conceptualization of ETL and physical storage of semantic data warehouses as a service. *Cluster Computing*, 16(4):915–931, 2013.

[30] Brett Berlin. DBMS selection and evaluation: Perspectives and practical issues. In *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 31 - June 2, 1978*, pages 37–38, 1978.

[31] Anthony C. Bloesch and Terry A. Halpin. Conquer: A conceptual query language. In *15th International Conference on Conceptual Modeling (ER)*, pages 121–133, 1996.

[32] Selma Bouarar. *Vers une conception logique et physique des bases de données avancées dirigée par la variabilité. (Towards a Variability-Aware Logical and Physical Database Design)*. PhD thesis, École nationale supérieure de mécanique et d'aérotechnique, France, 2016.

[33] Selma Bouarar, Stéphane Jean, and Norbert Siegmund. SPL driven approach for variability in database design. In *Model and Data Engineering - 5th International Conference, MEDI 2015, Rhodes, Greece, September 26-28, 2015, Proceedings*, pages 332–342, 2015.

[34] Rima Bouchakri, Ladjel Bellatreche, and Khaled-Walid Hidouci. Static and incremental selection of multi-table indexes for very large join queries. In *ADBIS*, pages 43–56, 2012.

[35] Kamel Boukhalfa. *De la conception physique aux outils d'administration et de tuning des entrepôts de données*. PhD thesis, École nationale supérieure de mécanique et d'aérotechnique, Chasseneuil-du-Poitou, Poitiers, France, 2009.

[36] Lahcène Brahimi. Vers une suite décisionnelle dédiée aux données de tests. In *Actes du 8 e Forum Jeunes Chercheurs du congrès INFORSID*, page 61, 2016.

[37] Lahcène Brahimi, Ladjel Bellatreche, and Yassine Ouhammou. A recommender system for DBMS selection based on a test data repository. In *Advances in Databases and Information Systems - 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28-31, Proceedings*, pages 166–180, 2016.

[38] Lahcène Brahimi, Ladjel Bellatreche, and Yassine Ouhammou. Coupling multi-criteria decision making and ontologies for recommending dbms. In *22nd International Conference on Management of Data, COMAD 2017, Chennai, India, March 08-10*, 2017.

[39] Lahcène Brahimi, Yassine Ouhammou, Ladjel Bellatreche, and Abdelkader Ouared. More transparency in testing results: Towards an open collective knowledge base. In *Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1-3*, pages 1–6, 2016.

[40] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[41] Dan Brickley and Ramanathan V Guha. Resource description framework (rdf) schema specification 1.0: W3c candidate recommendation 27 march 2000. *W3C recommendation, http://www.w3.org/TR/2000/CR-rdf-schema-20000327/*, 2000.

[42] Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. *W3C recommendation, https://www.w3.org/TR/rdf-schema/*, 25:2004–2014, 2014.

[43] Assia Brighen, Ladjel Bellatreche, Hachem Slimani, and Zoé Faget. An economical query cost model in the cloud. In *Database Systems for Advanced Applications - 18th International Conference, DASFAA 2013, International Workshops: BDMA, SNSM, SeCoP, Wuhan, China, April 22-25, 2013. Proceedings*, pages 16–30, 2013.

[44] Robin D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User-Adapt. Interact.*, 12(4):331–370, 2002.

[45] Luca Cabibbo and Riccardo Torlone. Querying multidimensional databases. In *International Workshop on Database Programming Languages*, pages 319–335. Springer, 1997.

[46] C. Marie-Françoise Canut, Sirinya On-at, André Péninou, and Florence Sèdes. Construction du profil social de l'utilisateur dans un contexte dynamique. application d'une méthode de pondération temporelle. *Ingénierie des Systèmes d'Information*, 21(2):65–94, 2016.

[47] Michael J Carey, David J DeWitt, Jeffrey F Naughton, Mohammad Asgarian, Paul Brown, Johannes E Gehrke, and Dhaval N Shah. The bucky object-relational benchmark. In *ACM SIGMOD Record*, volume 26, pages 135–146. ACM, 1997.

[48] Rick Cattell. Scalable SQL and nosql data stores. *SIGMOD Record*, 39(4):12–27, 2010.

[49] S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. *1982 ACM SIGMOD International Conference on Management of Data*, pages 128–136, 1982.

[50] Brigitte Chaput, Ahcene Benayache, Catherine Barry, and Marie-Héléne Abel. Une expérience de construction d'ontologie d'application pour indexer les ressources d'une formation en statistique. *Actes des Trente-sixiémes Journées Françaises de Statistique (SFDS'04), Montpellier, France*, 2004.

[51] Surajit Chaudhuri and Vivek Narasayya. Self-tuning database systems: a decade of progress. In *VLDB*, pages 3–14, 2007.

[52] Surajit Chaudhuri and Vivek R. Narasayya. Microsoft index tuning wizard for SQL server 7.0. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 553–554, 1998.

[53] Peter Chen. Software pioneers. chapter Entity-relationship modeling: historical events, future trends, and lessons learned, pages 296–310. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[54] Peter Pin-Shan Chen. The entity-relationship model toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[55] Rada Chirkova and Chen Li. Materializing views with minimal size to answer queries. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 38–48, 2003.

[56] Kristina Chodorow. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[57] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[58] E. F. Codd, S. B. Codd, and C. T. Salley. Providing olap (on-line analytical processing) to user-analysts: An it mandate. *White paper, http://www.arborsoft.com/essbase/wht_paper/coddTOC.html*, 1993.

[59] Edgar Frank Codd. Relational database: a practical foundation for productivity. *Communications of the ACM*, 25(2):109–117, 1982.

[60] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[61] John J. Cornyn, William R. Smith, Aaron H. Coleman, and William R. Svirsky. Life cycle cost models for comparing computer family architectures. In *American Federation*

of Information Processing Societies: 1977 National Computer Conference, June 13-16, 1977, Dallas, Texas, USA, pages 185–199, 1977.

[62] Timothy L Cross, Ronald J Lane, et al. Selecting a database management system for agricultural record keeping. Technical report, [Corvallis, Or.]: Oregon State University Extension Service;[Washington, DC]: US Dept. of Agriculture, 1988.

[63] Valerie V Cross and Thomas A Sudkamp. *Similarity and compatibility in fuzzy set theory: assessment and applications*, volume 93. 2002.

[64] Benoît Dageville, Dinesh Das, Karl Dias, Khaled Yagoub, Mohamed Zaït, and Mohamed Ziauddin. Automatic SQL tuning in oracle 10g. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 1098–1109, 2004.

[65] Philippe Dague and Louise Travé-Massuyès. Raisonnement causal en physique qualitative. *Intellectica*, 38:247–290, 2004.

[66] J. Darmont and P. Marcel. EntrepÃ´ts de donnÃ©es et olap, analyse et dÃ©cision dans l'entreprise. In *Les BigData Ã dÃ©couvert, CNRS Editions*, pages 132–133, 2017.

[67] Souripriya Das, Eugene Inseok Chong, Zhe Wu, Melliyal Annamalai, and Jagannathan Srinivasan. A scalable scheme for bulk loading large RDF graphs into oracle. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 1297–1306, 2008.

[68] Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan. Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 13–20. ACM, 2013.

[69] Nedim Dedic and Clare Stanier. An evaluation of the challenges of multilingualism in data warehouse development. In *ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems, Volume 1, Rome, Italy, April 25-28, 2016*, pages 196–206, 2016.

[70] Eric Deehr, Wen-Qi Fang, H. Reza Taheri, and Hai-Fang Yun. Performance analysis of database virtualization with the TPC-VMS benchmark. In *Performance Characterization and Benchmarking. Traditional to Big Data - 6th TPC Technology Conference, TPCTC 2014, Hangzhou, China, September 1-5, 2014. Revised Selected Papers*, pages 156–172, 2014.

[71] Hondjack Dehainsala. *Explicitation de la sémantique dans lesbases de données : Base de données à base ontologique et le modèle OntoDB*. PhD thesis, University of Poitiers, France, 2007.

[72] David J. DeWitt. The wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* 1993.

[73] David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *Proceedings of the 18th International Conference on Very Large Data Bases*, VLDB '92, pages 27–40, 1992.

[74] Akon Dey. Scalable transactions across heterogeneous nosql key-value data stores. *PVLDB*, 6(12):1434–1439, 2013.

[75] Kshitij A Doshi, Tao Zhong, Zhongyan Lu, Xi Tang, Ting Lou, and Gang Deng. Blending sql and newsql approaches: reference architectures for enterprise big data challenges. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on*, pages 163–170. IEEE, 2013.

[76] Christian Düntgen, Thomas Behr, and Ralf Hartmut Güting. Berlinmod: a benchmark for moving object databases. *VLDB J.*, 18(6):1335–1368, 2009.

[77] Ramez Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008.

[78] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 3rd Edition*. Addison-Wesley-Longman, 2000.

[79] Chimène Fankam. *OntoDB2 : un système flexible et efficient de base de données à base ontologique pour le web sémantique et les données techniques. (OntoDB2)*. PhD thesis, École nationale supérieure de mécanique et d'aérotechnique, Poitiers, France, 2009.

[80] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: an ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[81] IBM Center for Advanced Studies of Rome. Thinking ontologies at ibm center for advanced studies, 2006.

[82] Mark S. Fox. The TOVE project towards a common-sense model of the enterprise. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, 5th International Conference, IEA/AIE - 92, Paderborn, Germany, June 9-12, 1992, Proceedings*, pages 25–34, 1992.

[83] Georges Gardarin, Fei Sha, and Zhao-Hui Tang. Calibrating the query optimizer cost model of iro-db, an object-oriented federated database system. In *12èmes Journées Bases de Données Avancées, 27-30 Août 1996, Cassis (Informal Proceedings).*, pages 37–56, 1996.

[84] Antara Ghosh, Jignashu Parikh, Vibhuti S. Sengar, and Jayant R. Haritsa. Plan selection based on query clustering. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 179–190, 2002.

[85] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59, 2002.

[86] Ioana Giurgiu, Mirela Botezatu, and Dorothea Wiesmann. Comprehensible models for reconfiguring enterprise relational databases to avoid incidents. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1371–1380. ACM, 2015.

[87] Martin Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 21–26. IEEE, 2007.

[88] Matteo Golfarelli and Stefano Rizzi. Data warehouse testing: A prototype-based methodology. *Information and Software Technology*, 53:1183–1198, 2011.

[89] Matteo Golfarelli and Stefano Rizzi. Temporal data warehousing: Approaches and techniques. In *Integrations of Data Warehousing, Data Mining and Database Technologies - Innovative Approaches.*, pages 1–18. 2011.

[90] J GRAY. Database systems: A textbook case of research paying off. *Committee on the Fundamentals of Computer Science: Challenges and Opportunities, editors, Computer Science: Reflections on the Field, Reflections from the Field*, pages 80–88, 1995.

[91] Thomas R Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[92] Venkat N Gudivada, Dhana Rao, and Vijay V Raghavan. Nosql systems for big data management. In *Services (SERVICES), 2014 IEEE World Congress on*, pages 190–197. IEEE, 2014.

[93] Emna Guermazi, Mounir Ben Ayed, and Hanêne Ben-Abdallah. Adaptive security for cloud data warehouse as a service. In *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on*, pages 647–650. IEEE, 2015.

[94] Guibing Guo, Jie Zhang, and Daniel Thalmann. A simple but effective method to incorporate trusted neighbors in recommender systems. In *User Modeling, Adaptation, and Personalization - 20th International Conference, UMAP 2012, Montreal, Canada, July 16-20, 2012. Proceedings*, pages 114–125, 2012.

[95] Kunyi Guo, Ke Yu, Shanchen Pang, Dan Yang, Jun Huang, Yunni Xia, Xin Luo, and Jia Li. On the performance and power consumption analysis of elastic clouds. *Concurrency and Computation: Practice and Experience*, 28(17):4367–4384, 2016.

[96] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[97] Rashmi Gupta, Omesh Kumar, and Aditi Sharma. Article: Analytical study of various high performance computing paradigms. *International Journal of Applied Information Systems*, 1(9):16–21, April 2012. Published by Foundation of Computer Science, New York, USA.

[98] David Haertzen. *The Analytical Puzzle: Data Warehousing, Business Intelligence and Analytics*. Technics Publications, 2012.

[99] Florian Haftmann, Donald Kossmann, and Eric Lo. A framework for efficient regression tests on database applications. *VLDB J.*, 16:145–164, 2007.

[100] Michael Hammer and Dennis Mc Leod. Database description with sdm: a semantic database model. *ACM Transactions on Database Systems (TODS)*, 6(3):351–386, 1981.

[101] WAQAR HASAN and et al. Open issues in parallel query optimization, 1996.

[102] Florian Helff, Le Gruenwald, and Laurent d'Orazio. Weighted sum model for multi-objective query optimization for mobile-cloud database environments. In *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016.*, 2016.

[103] Joseph M Hellerstein and Michael Stonebraker. *Readings in database systems*. MIT Press, 2005.

[104] Florian Holzschuher and René Peinl. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13, Genoa, Italy, March 22, 2013, Workshop Proceedings*, pages 195–204, 2013.

[105] Ian Horrocks et al. Daml+oil: A description logic for the semantic web. *IEEE Data Eng. Bull., https://www.w3.org/TR/daml+oil-reference*, 25(1):4–9, 2002.

[106] William H Inmon. *Building the data warehouse*. John wiley & sons, 2005.

[107] Stéphane Jean, Guy Pierra, and Yamine Aït Ameur. Domain ontologies: A database-oriented analysis. In *Web Information Systems and Technologies, International Conferences, WEBIST 2005 and WEBIST 2006. Revised Selected Papers*, pages 238–254, 2006.

[108] Nick Jenkins. A software testing primer. *An Introduction to Software Testing*, 2008.

[109] Amira Kerkad. *L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels*. PhD thesis, ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique-Poitiers, 2013.

[110] Amira Kerkad, Ladjel Bellatreche, and Dominique Geniet. Queen-bee: Query interaction-aware for buffer allocation and scheduling problem. In *Data Warehousing and Knowledge Discovery - 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings*, pages 156–167, 2012.

[111] Selma Khouri. *Cycle de vie sémantique de conception de systèmes de stockage et manipulation de données. (Semantic lifecycle desing for data storage and manioulation systems)*. PhD thesis, École nationale supérieure de mécanique et d'aérotechnique, Chasseneuil-du-Poitou, near Poitiers, France, 2013.

[112] Ralph. Kimball. *The data warehouse lifecycle toolkit*. Wiley Pub., 2008.

[113] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. Owlim–a pragmatic semantic repository for owl. In *Web Information Systems Engineering–WISE 2005 Workshops*, pages 182–192. Springer, 2005.

[114] Ioannis Konstantinou, Evangelos Angelou, Christina Boumpouka, Dimitrios Tsoumakos, and Nectarios Koziris. On the elasticity of nosql databases over cloud management platforms. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 2385–2388, 2011.

[115] Anton Kos, Saso Tomazic, Jakob Salom, Nemanja Trifunovic, Mateo Valero, and Veljko Milutinovic. New benchmarking methodology and programming model for big data processing. *IJDSN*, 11:271752:1–271752:7, 2015.

[116] Charles W Krueger. Introduction to the emerging practice of software product line development. *Methods and Tools*, 14(3):3–15, 2006.

[117] Wilburt Labio, Dallan Quass, and Brad Adelberg. Physical database design for data warehouses. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE), April 7-11, 1997 Birmingham U.K.*, pages 277–288, 1997.

[118] Ora Lassila, Ralph R Swick, et al. Resource description framework (rdf) model and syntax specification. *W3C recommendation, http://www.w3.org/TR/REC-rdf-syntax/*, 1998.

[119] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[120] Scott T Leutenegger and Daniel Dias. *A modeling study of the TPC-C benchmark*, volume 22. ACM, 1993.

[121] Vincent Link, Steffen Lohmann, and Florian Haag. Ontobench: Generating custom OWL 2 benchmark ontologies. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II*, pages 122–130, 2016.

[122] Yi Liu, Jun Feng, and Jiamin Lu. Collaborative filtering algorithm based on rating distance. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017, Beppu, Japan, January 5-7, 2017*, page 66, 2017.

[123] Frederick H. Lochovsky and Dennis Tsichritzis. User performance considerations in DBMS selection. In *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data, Toronto, Canada, August 3-5, 1977.*, pages 128–134, 1977.

[124] Guy M. Lohman, C. Mohan, Laura M. Haas, Dean Daniels, Bruce G. Lindsay, Patricia G. Selinger, and Paul F. Wilms. Query processing in r*. In *Query Processing in Database Systems*, pages 31–47. Springer, 1985.

[125] Oscar López, Miguel A Laguna, and Francisco J García. Metamodeling for requirements reuse. In *Anais do WER02-Workshop em Engenharia de Requisitos, Valencia, Spain*, 2002.

[126] Andreas Lübcke. *Automated query interface for hybrid relational architectures*. PhD thesis, Dissertation, Magdeburg, Universität, 2016, 2016.

[127] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 287–296, 2011.

[128] Cristina Maier, Debabrata Dash, Ioannis Alagiannis, Anastasia Ailamaki, and Thomas Heinis. Parinda: an interactive physical designer for postgresql. In *EDBT*, pages 701–704, 2010.

[129] Elzbieta Malinowski and Esteban Zimányi. OLAP hierarchies: A conceptual perspective. In *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*, pages 477–491, 2004.

[130] Stefan Manegold, Peter A. Boncz, and Martin L. Kersten. Generic database cost models for hierarchical memory systems. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 191–202, 2002.

[131] Frank Manola, Eric Miller, Brian McBride, et al. Rdf primer. *W3C recommendation, https://www.w3.org/TR/2004/REC-rdf-primer-20040210/*, 10(1-107):6, 2004.

[132] Abbas Mardani, Ahmad Jusoh, Khalil MD Nor, Zainab Khalifah, Norhayati Zakwan, and Alireza Valipour. Multiple criteria decision-making techniques and their applications–a review of the literature from 2000 to 2014. *Economic Research-Ekonomska Istraživanja*, 28(1):516–571, 2015.

[133] Bery Leouro Mbaiossoum. *Conception physique des bases de données à base ontologique : le cas des vues matérialisées. (Physicaly Design of Ontology-Based Databases)*. PhD thesis, École nationale supérieure de mécanique et d'aérotechnique, Chasseneuil-du-Poitou, Poitiers, France, 2014.

[134] Brian McBride. Jena: Implementing the RDF model and syntax specification. In *SemWeb*, 2001.

[135] Deborah L. McGuinness, Richard Fikes, James A. Hendler, and Lynn Andrea Stein. DAML+OIL: an ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5):72–80, 2002.

[136] Stuart E. Middleton, Nigel Shadbolt, and David De Roure. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22(1):54–88, 2004.

[137] Bamshad Mobasher, Robin D. Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Techn.*, 7(4):23, 2007.

[138] Antonio Moreno, Aida Valls, David Isern, Lucas Marin, and Joan Borràs. Sigtur/e-destination: ontology-based personalized recommendation of tourism and leisure activities. *Engineering Applications of Artificial Intelligence*, 26(1):633–651, 2013.

[139] Shamkant B Navathe. Evolution of data modeling for databases. *Communications of the ACM*, 35(9):112–123, 1992.

[140] Sudsanguan Ngamsuriyaroj and Rangsan Pornpattana. Performance evaluation of TPC-H queries on mysql cluster. In *24th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2010, Perth, Australia, 20-13 April 2010*, pages 1035–1040, 2010.

[141] Thi-Van-Anh Nguyen, Sandro Bimonte, Laurent d'Orazio, and Jérôme Darmont. Cost models for view materialization in the cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012*, pages 47–54, 2012.

[142] Philipp Obermeier and Lyndon Nixon. A cost model for querying distributed rdf-repositories with sparql. In *Proceedings of the Workshop on Advancing Reasoning on the Web: Scalability and Commonsense Tenerife, Spain*, 2008.

[143] Tamer M. Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, third edition*. Springer, 2011.

[144] Sangwon Park. Flash-aware cost model for embedded database query optimizer. *J. Inf. Sci. Eng.*, 29(5):947–967, 2013.

[145] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs, and Billie Rinaldi. Ycsb++: benchmarking and performance debugging advanced features in scalable table stores. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 9. ACM, 2011.

[146] Mauro Pezzè and Cheng Zhang. Automated test oracles: A survey. *Advances in Computers*, 95:1–48, 2015.

[147] Romain Picot-Clémente. *Une architecture générique de Systèmes de recommandation de combinaison d'items : application au domaine du tourisme. (A generic framework for recommender systems generating combination of items : application to the tourism domain)*. PhD thesis, University of Burgundy, Dijon, France, 2011.

[148] G Pierra, E Sardet, JC Potier, G Battier, JC Derouet, N Willmann, and A Mahir. Exchange of component data: the plib (iso 13584) model, standard and tools. *Proceedings of the CALS EUROPE*, 98:160–176, 1998.

[149] Guy Pierra. Context-explication in conceptual ontologies: the PLIB approach. In *Enhanced Interoperable Systems. Proceedings of the 10th ISPE International Conference on Concurrent Engineering (ISPE CE 2003), July 26-30, 2003, Madeira, Portugal*, pages 243–253, 2003.

[150] Guy Pierra. The PLIB ontology-based approach to data integration. In *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, pages 13–18, 2004.

[151] Jaroslav Pokorný and Peter Sokolowsky. A conceptual modelling perspective for data warehouses. In *Electronic Business Engineering, 4. Internationale Tagung Wirtschaftsinformatik, Saarbrücken, 3.-5. März 1999*, page 35, 1999.

[152] Jing Quan, Yingjie Shi, Ming Zhao, and Wei Yang. The implications from benchmarking three big data systems. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 31–38, 2013.

[153] Alexander Rasin and Stanley B. Zdonik. An automatic physical design tool for clustered column-stores. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 203–214, 2013.

[154] Eric Redmond and Jim R Wilson. Seven databases in seven weeks. *The Pragmatic Bookshelf*, 2012.

[155] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*, pages 175–186, 1994.

[156] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. 2011.

[157] Peter Rob and Carlos Coronel. *Database systems - design, implementation, and management (2. ed.)*. Boyd and Fraser, 1995.

[158] A. Rochfeld and H. Tardieu. MERISE: an information system design and development methodology. *Information & Management*, 6(3):143–159, 1983.

[159] Colette Rolland. Ingénierie des Besoins : L'Approche L'Ecritoire. *Journal Techniques de l'Ingénieur*, 2003.

[160] Marko Rosenmüller, Christian Kästner, Norbert Siegmund, Sagar Sunkle, Sven Apel, Thomas Leich, and Gunter Saake. Sql á la carte-toward tailor-made data management. In *BTW*, pages 117–136. Citeseer, 2009.

[161] Marko Rosenmüller, Norbert Siegmund, Christian Kästner, and Syed Saif ur Rahman. Modeling dependent software product lines. In *Proceedings of the GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE)*, pages 13–18. Citeseer, 2008.

[162] Marko Rosenmüller, Norbert Siegmund, Horst Schirmeier, Julio Sincero, Sven Apel, Thomas Leich, Olaf Spinczyk, and Gunter Saake. Fame-dbms: tailor-made data management solutions for embedded systems. In *Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*, pages 1–6. ACM, 2008.

[163] Amine Roukh, Ladjel Bellatreche, Selma Bouarar, and Ahcene Boukorca. Eco-physic: Eco-physical design initiative for very large databases. *Information Systems*, 2017.

[164] Amine Roukh, Ladjel Bellatreche, Ahcène Boukorca, and Selma Bouarar. Eco-dmw: Eco-design methodology for data warehouses. In *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP, DOLAP 2015, Melbourne, VIC, Australia, October 19-23, 2015*, pages 1–10, 2015.

[165] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295, 2001.

[166] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

[167] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1.*, pages 23–34, 1979.

[168] Yingjie Shi, Xiaofeng Meng, Jing Zhao, Xiangmei Hu, Bingbing Liu, and Haiping Wang. Benchmarking cloud-based data management systems. In *Proceedings of the second international workshop on Cloud data management*, pages 47–54. ACM, 2010.

[169] Avi Silberschatz, Mike Stonebraker, and Jeff Ullman. Database research: achievements and opportunities into the 1st century. *ACM sIGMOD record*, 25(1):52–63, 1996.

[170] Alkis Simitsis, Panos Vassiliadis, and Timos K. Sellis. Optimizing ETL processes in data warehouses. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 564–575, 2005.

[171] John Miles Smith and Diane CP Smith. Database abstractions: aggregation and generalization. *ACM Transactions on Database Systems (TODS)*, 2(2):105–133, 1977.

[172] Ian Sommerville and Gerald Kotonya. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[173] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2008.

[174] Michael Stonebraker and Rick Cattell. 10 rules for scalable performance in'simple operation'datastores. *Communications of the ACM*, 54(6):72–80, 2011.

[175] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.

[176] Vijayan Sugumaran and Veda C. Storey. The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Trans. Database Syst.*, 31(3):1064–1094, 2006.

[177] Rebecca Taft, Manasi Vartak, Nadathur Rajagopalan Satish, Narayanan Sundaram, Samuel Madden, and Michael Stonebraker. Genbase: a complex analytics genomics benchmark. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 177–188, 2014.

[178] David Taniar, Clement H. C. Leung, Wenny Rahayu, and Sushant Goel. *High Performance Parallel Database Processing and Grid Databases*. Wiley Publishing, 2008.

[179] Anna Thanopoulou, Paulo Carreira, and Helena Galhardas. Benchmarking with TPC-H on off-the-shelf hardware - an experiments report. In *ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 1, Wroclaw, Poland, 28 June - 1 July, 2012*, pages 205–208, 2012.

[180] Albert Tort, Antoni Olivé, and Maria-Ribera Sancho. An approach to test-driven development of conceptual schemas. *Data Knowl. Eng.*, 70:1088–1111, 2011.

[181] Brendon Towle and Clark Quinn. Knowledge based recommender systems using explicit user models. In *Proceedings of the AAAI Workshop on Knowledge-Based Electronic Markets*, pages 74–77, 2000.

[182] Evangelos Triantaphyllou. Multi-criteria decision making methods. In *Multi-criteria Decision Making Methods: A Comparative Study*, pages 5–21. Springer, 2000.

[183] Dennis Tsichritzis and Anthony Klug. The ansi/x3/sparc dbms framework report of the study group on database management systems. *Information systems*, 3(3):173–191, 1978.

[184] Craig Ulmer, Greg Bayer, Yung Ryn Choe, and Diana Roe. Exploring data warehouse appliances for mesh analysis applications. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[185] Syed Saif ur Rahman, Veit Köppen, and Gunter Saake. Cellular dbms: An attempt towards biologically-inspired data management. *JDIM*, 8(2):117–124, 2010.

[186] Alejandro A. Vaisman and Esteban Zimányi. *Data Warehouse Systems - Design and Implementation*. Data-Centric Systems and Applications. Springer, 2014.

[187] Panos Vassiliadis and Timos K. Sellis. A survey of logical models for OLAP databases. *SIGMOD Record*, 28(4):64–69, 1999.

[188] Luis Eduardo Bautista Villalpando, Alain April, and Alain Abran. Cloudmeasure: A platform for performance analysis of cloud computing systems. In *9th IEEE International Conference on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2, 2016*, pages 975–979, 2016.

[189] Kiri Wagstaff. Clustering with missing values: No imputation required. *Classification, Clustering, and Data Mining Applications*, pages 649–658, 2004.

[190] Jian Wei, Kai Chen, Yi Zhou, Qu Zhou, and Jianhua He. Benchmarking of distributed computing engines spark and graphlab for big data analytics. In *Second IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2016, Oxford, United Kingdom, March 29 - April 1, 2016*, pages 10–13, 2016.

[191] Chris Welty, Deborah L McGuinness, and Michael K Smith. Owl web ontology language guide. *W3C recommendation, W3C (February 2004), http://www.w3.org/TR/2004/REC-owl-guide-20040210*, 2004.

[192] Paul Westerman. *Data Warehousing: Using the Wal-Mart Model*. Morgan Kaufmann, 2000.

[193] Wentao Wu, Yun Chi, Shenghuo Zhu, Jun'ichi Tatemura, Hakan Hacigümüs, and Jeffrey F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 1081–1092, 2013.

[194] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 136–145, 1997.

[195] Masaharu Yoshioka. ABRIR at NTCIR-9 geotime task usage of wikipedia and geonames for handling named entity information. In *Proceedings of the 9th NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access, NTCIR-9, National Center of Sciences, Tokyo, Japan, December 6-9, 2011*, 2011.

[196] Jidong Zhai, Feng Zhang, Qingwen Li, Wenguang Chen, and Weimin Zheng. Characterizing and optimizing TPC-C workloads on large-scale systems using SSD arrays. *SCIENCE CHINA Information Sciences*, 59(9):92104, 2016.

[197] Jian Zhang, Chen Xu, and S. C. Cheung. Automatic generation of database instances for white-box testing. In *25th International Computer Software and Applications Conference (COMPSAC 2001), Invigorating Software Development, 8-12 October 2001, Chicago, IL, USA*, pages 161–165. IEEE, 2001.

[198] Ning Zhang, Junichi Tatemura, Jignesh M Patel, and Hakan Hacigümüş. Towards cost-effective storage provisioning for dbmss. *Proceedings of the VLDB Endowment*, 5(4):274–285, 2011.

[199] D. C. Zilio, J. Rao, S. Lightstone, G. M Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. pages 1087–1097, August 2004.

Table des figures

1.1 Aperçu de l'article de Edgar Frank « Ted » Codd 4
1.2 Quelques SGBD NoSQL 8
1.3 Extrait de l'article montrant les questions relatives au problème de sélection des SGBD 10
1.4 Le *déployeur* et ses entrées 11
1.5 Évolution en nombre et en type des SGBD 12
1.6 Catégories de choix de SGBD et de plateformes 13
1.7 Le Schéma de la base DBLP 14
1.8 Recommandations à partir des articles scientifiques 15
1.9 Une approche de résolution de notre problème 16
1.10 Notre démarche 17
1.11 Quelques dimensions de tests extraites de [7] 18
1.12 Aperçu globale des contributions de la thèse 19

2.1 Types d'évolution du cycle de conception des BD : horizontale, verticale, interne 29
2.2 Les phases du cycle de vie et leurs rôles 29
2.3 Exemple d'une représentation hiérarchique 30
2.4 Exemple d'une représentation en réseau 30
2.5 Un exemple de diagramme entités/associations 32
2.6 Evolution verticale du cycle de conception des $\mathcal{BD}$ 33
2.7 L'architecture ANSI/SPARC 34
2.8 Différentes composantes d'un SGBD 36

2.9 Évolution interne du cycle de conception des BD 37
2.10 *Architetcures matérielles usuelles* 41
2.11 *Exemple d'un cluster d'ordinateurs* 43
2.12 *Exemple de l'infrastructure grille de calcul* 44
2.13 *Exemple de l'infrastructure cloud de calcul* 45
2.14 *Phase physique dans le cycle de vie* 47
2.15 Evolution horizontale du cycle de conception des $\mathcal{BD}$ 49
2.16 Architecture d'une BDBO 51
2.17 *Cliché des résultats de TPC-C* 53

3.1 Démarche d'élagage de résolution de notre problème 59
3.2 Schéma du banc d'essai YCSB 65
3.3 Schéma du banc d'essai TPC-C 67
3.4 Schéma du banc d'essai TPC-H 68
3.5 Schéma du SSB 69
3.6 Schéma du banc d'essai BUCKY 69
3.7 Évolution des modèles de coût 73
3.8 Classification issue de *DB-Engines* 75
3.9 Classification des SGBD suivant les propriétés ACID, CAP et BASE 77
3.10 Classification des systèmes selon le théorème de CAP 78
3.11 Classification selon le nombre d'utilisateurs et le volume de données 79

4.1 Extrait du modèle de l'environnement de test mettant l'accent sur la classe `Test` et ses composantes 91
4.2 Extrait du modèle de l'environnement de test mettant l'accent sur la composante `Metric` 92
4.3 Extrait du modèle de l'environnement de test mettant l'accent sur la composante `CostModel` 93
4.4 Modèle structurel de base de données relationnelle 94
4.5 Modèle structurel de base de données NoSQL 94
4.6 Extrait du modèle de l'environnement de test mettant l'accent sur les composantes `DataSet` (schéma) et `Query` 97
4.7 Modèle de schéma S et graphe de requêtes Qa et Qb 97

4.8 Extrait du modèle de l'environnement de test mettant l'accent sur les trois composantes d'un SBD . . . 98
4.9 Vue en graphe des trois classes de l'ontologie des systèmes de $\mathcal{BD}$ (SBD) . . . 105
4.10 Vue en graphe de la classe des SGBD . . . 106
4.11 Vue en graphe de la sous-classe Data Access . . . 107
4.12 Vue en graphe de la sous-classe Software . . . 108
4.13 Vue en graphe de la classe Hardware . . . 109
4.14 Liste des SGBD supportant le langage de programmation "Matlab" . . . 110

5.1 Architecture des entrepôts de données . . . 117
5.2 Entrepôt de données : modèle en étoile . . . 119
5.3 Entrepôt de données : modèle en flocon . . . 119
5.4 Vue d'un cube OLAP à trois dimensions . . . 121
5.5 Représentation hiérarchique du cube OLAP . . . 121
5.6 Architecture d'un système décisionnel . . . 124
5.7 Notre entrepôt de données DW_TESTS . . . 125
5.8 Exemple d'instanciation de notre DW_TESTS . . . 126
5.9 Aperçu du dépôt de tests . . . 127
5.10 Exemple de *manifeste* . . . 127
5.11 Extrait du dépôt de test : entrepôt de données . . . 129
5.12 Vue sur le gestionnaire d'espace de travail analytique pour Oracle OLAP . . . 131
5.13 Minimum de temps de réponse par requête, SGBD et taille de BD . . . 132
5.14 Liste des SGBD ayant un temps de réponse < 100 par requête . . . 133

6.1 Structure du système de recommandation . . . 144
6.2 Matrice des SGBD et ses fonctionnalités . . . 148
6.3 Extrait du manifeste avec ses trois parties . . . 150
6.4 Requête SPARQL du critère "stockage en mémoire" . . . 150
6.5 Requête SPARQL récupérant les SGBD supportant les "données géospatielles" 150
6.6 Résultat de notre proposition basée sur WSM . . . 151
6.7 Résultats correspondant au scénario 1 . . . 153
6.8 Extrait du manifeste correspondant au scénario 2 . . . 154

6.9 Extrait du manifeste correspondant au scénario 3 155
6.10 Extrait du manifeste correspondant au deuxième cas d'étude 157

Liste des tableaux

3.1 Test de cycle de vie des bases de données 60
3.2 Comparaison des principaux advisors[35] 62
3.3 Différents bancs d'essai 64
3.4 Recensement des travaux de performance de $\mathcal{BD}$ en mode cloud 71
3.5 Recensement des travaux de performance de $\mathcal{BD}$ *Big Data* 72
3.6 Recensement des travaux des modèles de coût 74
3.7 Comparaison entre SGBD Relationnels, NoSQL et NewSQL 77

4.1 Environnement de test 89
4.2 Environnement de test selon le formalisme 91

5.1 Exemple cube OLAP 122
5.2 Opération roll up 122
5.3 Opération roll down 123
5.4 Opération slice 123
5.5 Opération dice 123
5.6 Avant l'opération pivot 124
5.7 Après l'opération pivot 124

6.1 Matrice SGBD entreprises 142
6.2 Filtrage collaboratif basé sur l'utilisateur 142
6.3 Filtrage collaboratif basé sur l'item 143
6.4 Critères et poids associés 145

6.5 Exemple de processus de notre système de recommandation 147
6.6 Scénarios de l'étude expérimentale 152
6.7 Dix tests de Microsoft SQL server avec TPC-H 152
6.8 Temps de réponse réel pour $Q3$, $Q7$, $Q19$ de quatre SGBD : Oracle, Microsoft SQL Server, DB2 et Sybase 153
6.9 Temps de réponse approché pour $Q3$, $Q7$, $Q19$ de quatre SGBD : Oracle, Microsoft SQL Server, DB2 et Sybase 154
6.10 SGBD et plateformes sélectionnées sur la base des temps de réponse des requêtes $Q3$, $Q7$, $Q19$ 156
6.11 Similarité entre Qb et les requêtes de TPC-H 156
6.12 Taille des tables de TPC-H (SF=1) 157
6.13 Résultats du manifeste 4 158

Glossaire

$\mathcal{BD}$ **:** Base de Données
CaaS: Communication-as-a-Service
CQL: Cassandra Query Language
CWM: Common Warehouse Metamodel
DBaaS: Database-as-a-Service
DSMS: Data Stream Management System
ED : Entrepôt de Données
ENSMA : École Nationale Supérieure de Mécanique et d'Aérotechnique
ETL: Extract-Transform-Load
IaaS: Infrastructure-as-a-Service
IMS: Information Management System
IO: Input Output
JDBC: Java DataBase Connectivity
LIAS : Laboratoire d'Informatique et d'Automatique pour les Systèmes
LRB: Linear Road Benchmark
MCD : Modèle Conceptuel des Données
MDA: Model-Driven Architecture
MLD : Modèle Logique des Données
MPD : Modèle Physique des Données
MPP: Massive Parallel Processing
NaaS: Network-as-a-Service
NAS: Network Attached Storage
NewSQL : SGBDR modernes
NoSQL: Not only SQL
NUMA: Non Uniform Memory Architecture
ODBC: Open DataBase Connectivity
OLAP: On-Line Analytical Processing
OLTP: On-line Transaction Processing
ORM: Object Relational Mapping

OWL: Ontology Web Language
PaaS: Plarform-as-a-Service
RDF: Ressource Description Framework
RDFS: RDF Schéma
SaaS: Software-as-a-Service
SAN: Storage Area Network
SBD : Système de base de données
SGBD : Système de Gestion de Bases de Données
SGBDR : SGBD Relationnel
SPARQL: A recursive acronym for SPARQL Protocol and RDF Query Language
SPL: Software Product Lines
SQL: Structured Query Language
SSB: Star Schema Benchmark
STaaS: Storage-as-a-Service
TPC: Transaction Processing Performance Council
TPC-H: TPC Benchmark H
TPC-C: TPC Benchmark C
XML: eXtensible Markup Language
YCSB: Yahoo! Cloud Serving Benchmark

Résumé

Le choix d'un système de gestion de bases de données (SGBD) et de plateforme d'exécution pour le déploiement est une tâche primordiale pour la satisfaction des besoins non-fonctionnels (comme la performance temporelle et la consommation d'énergie). La difficulté de ce choix explique la multitude de tests pour évaluer la qualité des bases de données (BD) développées. Cette évaluation se base essentiellement sur l'utilisation des métriques associées aux besoins non fonctionnels. En effet, une mine de tests existe couvrant toutes les phases de cycle de vie de conception d'une BD. Les tests et leurs environnements sont généralement publiés dans des articles scientifiques ou dans des sites web dédiés comme le TPC (*Transaction Processing Council*). Par conséquent, cette thèse contribue à la capitalisation et l'exploitation des tests effectués afin de diminuer la complexité du processus de choix. En analysant finement les tests, nous remarquons que chaque test porte sur les jeux de données utilisés, la plateforme d'exécution, les besoins non fonctionnels, les requêtes, etc. Nous proposons une démarche de conceptualisation et de persistance de toutes ces dimensions ainsi que les résultats de tests. Cette thèse a donné lieu aux trois contributions. (1) Une conceptualisation basée sur des modélisations descriptive, prescriptive et ontologique pour expliciter les différentes dimensions. (2) Le développement d'un entrepôt de tests multidimensionnel permettant de stocker les environnements de tests et leurs résultats. (3) Le développement d'une méthodologie de prise de décision basée sur un système de recommandation de SGBD et de plateformes.

Mots clés : Gestion de bases de données, entrepôt de données, OLAP (informatique), Ontologies (informatique), Systèmes de recommandation (informatique), apprentissage automatique, Prise de décision.

Abstract

Choosing appropriate database management systems (DBMS) and/or execution platforms for given database (DB) is complex and tends to be time- and effort-intensive since this choice has an important impact on the satisfaction of non-functional requirements (e.g., temporal performance or energy consumption). Indeed, a large number of tests have been performed for assessing the quality of developed DB. This assessment often involves metrics associated with non-functional requirement. That leads to a mine of tests covering all life-cycle phases of the DB's design. Tests and their environments are usually published in scientific articles or specific websites such as Transaction Processing Council (TPC). Therefore, this thesis has taken a special interest to the capitalization and the reutilization of performed tests to reduce and mastery the complexity of the DBMS/platforms selection process. By analyzing the test accurately, we identify that tests concern: the data set, the execution platform, the addressed non-functional requirements, the used queries, etc. Thus, we propose an approach of conceptualization and persistence of all dimensions as well as the results of tests. Consequently, this thesis leads to the following contributions. (1) The design model based on descriptive, prescriptive and ontological concepts to raise the different dimensions. (2) The development of a multidimensional repository to store the test environments and their results. (3) The development of a decision-making methodology based on a recommender system for DBMS and platforms selection.

Key words: database management system, data warehouse, OLAP technology , Ontologies, Recommender systems (Information filtering), Machine learning , Decision making.