Vers une conception logique et physique des bases de données avancées dirigée par la variabilité

Selma Bouarar

▶ **To cite this version:**

Selma Bouarar. Vers une conception logique et physique des bases de données avancées dirigée par la variabilité. Autre [cs.OH]. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique - Poitiers, 2016. Français. NNT : 2016ESMA0024 . tel-01470445

HAL Id: tel-01470445

https://theses.hal.science/tel-01470445v1

Submitted on 17 Feb 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.

THESE

pour l'obtention du Grade de

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

(Diplôme National — Arrêté du 25 mai 2016)

Ecole Doctorale : Science et Ingénierie pour l'Information, Mathématiques (S2IM)
Secteur de Recherche : INFORMATIQUE ET APPLICATIONS

Présentée par :

Selma BOUARAR

****

Vers une conception logique et physique des bases de données avancées dirigée par la variabilité

****

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

Soutenue le 13 décembre 2016
devant la Commission d'Examen

JURY

Rapporteurs :	**Isabelle COMYN-WATTIAU**	Professeur, ESSEC Business School, Cergy-Pontoise
	Esteban ZIMANYI	Professeur, Ecole polytechnique de Bruxelles, Belgique
Examinateurs :	**Xavier BLANC**	Professeur, Université de Bordeaux
	Parisa GHODOUS	Professeur, Université de Lyon 1
	Ladjel BELLATRECHE	Professeur, ISAE-ENSMA, Poitiers
	Stéphane JEAN	Maître de conférences, Université de Poitiers

Remerciements

Au terme de cette longue aventure, je tiens tout d'abord à exprimer toute ma profonde gratitude aux personnes suivantes :

Ladjel Bellatreche qui n'a ménagé aucun effort en voulant me faire partager sa grande passion de la recherche. Merci pour son génie qui fait de lui une vraie machine à idées pas toujours facile à suivre ☺, mais toujours enrichissante à contempler, sa confiance (parfois excessive) qu'il m'a accordée et ce jusqu'au jour de la soutenance, sa bienveillance souvent cachée sous une coque rigide ainsi que les rires et les "pleurs" en sa compagnie qui m'ont fait grandir ☺...

Stéphane Jean pour tous ses efforts entrepris afin de me corriger minutieusement rapport et articles et m'apprendre à les rédiger, pour sa rigueur, la clarté de ses idées ainsi que ses conseils avisés. Merci pour le rôle d'"*équilibreur*" qu'il a de temps en temps joué pendant ma thèse. J'ai beaucoup appris aux côtés de mes encadrants tant sur le plan humain que scientifique.

Un grand merci aux membres du jury de m'avoir fait l'honneur de s'intéresser à mes travaux : Isabelle COMYN-WATTIAU et Esteban ZIMANYI en tant que rapporteurs, ainsi qu'Xavier BLANC et Parisa GHODOUS en tant qu'examinateurs. Merci pour leurs remarques enrichissantes et encourageantes.

Un merci tout particulier à Mickael Baron, mon adorable "tuteur" qui, par son énergie, sa gentillesse et son humour, m'a toujours redonné espoir et motivation, et ce depuis le premier jour de mon arrivée au LIAS. Je tiens également à remercier Emmanuel Grolleau, Brice Chardin et Zoé Faget pour leur soutien amical, ainsi que Denis Lemonnier, Audrey Veron, Céline Portron, Jocelyne Bardeau et Frédéric Carreau pour leur professionnalisme.

En dehors de la thèse, ces trois dernières années ont été pour moi l'occasion de faire mes débuts dans l'enseignement, ce que fut l'un de mes meilleurs moments de cette tranche de vie. À ce titre, je voudrais remercier Laurent Guittet, Dominique Proudhon, Henri Bauer et le CPMEC de m'avoir fait découvrir l'enseignement et de me le faire tant apprécier.

Je n'aurai jamais pu mener cette aventure à terme sans l'appui moral et scientifique de mes chers amis qui m'ont aidé à m'aménager ~~beaucoup~~ de "parenthèses" et à ne pas me sentir comme un ovni ☺ entre autres. Mes pensées vont d'abord à mes chers collègues :

- ✦ les doyens : Selma Khouri et Yassine Ouhammou pour leur esprit perspicace et généreux dont j'ai beaucoup appris... Ces deux-là sont devenus bien plus que des collègues au fil du temps. Je remercie également Ilyes, Chedlia, Linda, Amira, Soumia, Moustapha, Zakaria, Georges, Bery et Christian;
- ✦ mes "co-peines" : Ahcène l'intelligent, Géraud le confident, Guillaume le gentil râleur, Amine l'énergie, Zouhir la mécanique, Zahira, Abdelkader, Kevin, Olga, Okba et Thomas le méchant prof 😄
- ✦ les futurs docteurs à qui je souhaite beaucoup de succès : Nassima, Lahcene, Sarar, Sayda, Yves, Paule, Aymen, Anaïs, Abdelkarim, Dat-Nguyen, Ibrahim et Anh-Toan.

Une pensée particulière pour mes équipes de badminton, mes collègues des ~~fou-rires~~ cours d'espagnol, mes différents collègues de mes différents bureaux pour avoir supporté toutes mes histoires sans fin ☺...

Merci au destin de m'avoir permis de croiser les chemins de : Dounia G., Faten R., Sarah F., Firas H., Zahra E., Hayet B., Djamel K., Lotfi S., Yasser, Max B., Amel H., Meriem T., Iman B., Philippe M., Romaric H., Mohamed R., Raoul T., Camo B., Thao T., Benjamin, Soizic, Patricia, Jérôme M. et Céline M., entre autres.

Enfin et surtout, j'aimerais remercier ma famille en commençant par ma fratrie pour exprimer tout l'amour que j'ai pour elle : Hafsa, Zineb, Salah et Amine. Merci aussi à tonton Abderrahmane Hamdane, Amel Allag, mes beaux frères et tous mes proches pour leur aide, leur amour et leur présence précieuse à mes côtés durant toutes ces années.

Mes très chers parents à qui je dédie ce titre en témoignage de gratitude. Aucune dédicace ne saurait exprimer mon grand amour éternel pour vous et ma grande reconnaissance pour les sacrifices que vous avez consentis pour mon instruction et mon bien être.

Je clos ces remerciements par une pensée pleine d'affection et de nostalgie pour ma très chère grand-mère "Berkahom" dite "Mani" qui, même loin des yeux, fait toujours partie de ma vie et à tout jamais ... Mani, Mama-Aicha, Baba-sidi et Djeddou Laid, trouvez en ce titre un petit geste de reconnaissance et que dieu - que je remercie pour tout - nous réunisse au paradis.

Que tous ceux et celles que je n'ai pas mentionnés n'en reçoivent pas moins ma gratitude.

Et si c'était à refaire ?! ...

Table des matières

Introduction générale **1**

Partie I De la nécessité d'une gestion explicite de la variabilité pour la conception des bases de données

Chapitre 1 Évolution du cycle de conception des bases de données **13**

1 Introduction 15

2 Notions relatives à la conception des BD 17

2.1 Système d'information vs Base de données vs Système de gestion . 17

2.2 Cycle de développement des bases de données 18

2.3 Notre positionnement dans ce cycle de développement 19

2.4 Acteurs intervenants dans la conception 20

2.5 Conception et modélisation 20

3 Stade de maturation : architecture 1-tier 21

3.1 Modèles physiques de données et SGBD 21
3.2 L'avènement des modèles logiques 23
3.3 L'avènement des modèles conceptuels 25
4 Stade de maturité : architecture 3-tiers 26
4.1 Cycle triptyque de conception de BD 26
4.1.1 Design conceptuel 28
4.1.2 Conception logique 29
4.1.3 Conception physique 30
4.2 Évolution horizontale de chaque étape 31
4.2.1 Variation au sein du design conceptuel 31
4.2.2 Variation au sein de la conception logique 32
4.2.3 Variation au sein de la conception physique 33
4.3 Évolution verticale : des bases de données aux entrepôts de données 35
4.3.1 Traitement transactionnel et analytique des données . . . 35
4.3.2 Modélisation multidimensionnelle 36
4.3.3 Cycle de conception de BD revisité 38
5 Stade dynamique : instabilité du cycle 42
5.1 Les principales innovations 42
5.2 La diversité des besoins non fonctionnels 43
6 Conclusion 44

Chapitre 2 Gestion de la variabilité de la conception des bases de données 47

1 Introduction 49
2 Travaux orientés vers l'aide à la décision 50
2.1 Test de BD 51
2.1.1 Classification des travaux sur le test de BD 52
2.1.2 Limites des tests de BD 54
2.2 Bancs d'essais 54

2.2.1 Les bancs d'essais existants 55
2.2.2 Limites des bancs d'essais 56
2.3 *Advisors* : outils d'aide à l'administration de BD 57
2.3.1 Conception physique et tuning des BD 57
2.3.2 Limites des advisors 60
2.4 Optimisation à base de modèles de coûts 60
2.4.1 Définitions 60
2.4.2 Travaux existants et limites 62
3 Travaux orientés vers la réutilisation 62
3.1 Automatisation : les outils CASE 63
3.1.1 Définitions 64
3.1.2 Les outils CASE existants 65
3.1.3 Limites des outils CASE 65
3.2 Standardisation : les design patterns 66
3.2.1 Définitions 66
3.2.2 Les design patterns existants 67
3.2.3 Les limites des design patterns 68
4 Travaux orientés vers la gestion explicite de la variabilité 69
4.1 Notions sur la gestion de la variabilité en génie logiciel 70
4.1.1 Lignes de produits (LdP) 70
4.1.2 Étapes de la gestion de la variabilité 71
4.1.3 LdP et ingénierie des modèles 73
4.2 Travaux existants portant sur la phase physique 73
4.2.1 Variabilité des SGBD 74
4.2.2 Moteur de requêtes 76
4.2.3 Modèle de stockage 76
4.3 Travaux existants portant sur la phase conceptuelle 76

5 Bilan et discussion 77

Partie II Approche par étapes pour la gestion de la variabilité au sein de la conception des bases de données

Chapitre 3 La conception des bases de données comme une ligne de produits 83

1 Introduction 85

2 Analyse de la variabilité de la conception des BD 86

2.1 Framework générique des LdP 86

2.2 Phase conceptuelle 87

2.3 Phase logique 88

2.4 Phase physique 91

2.5 Identification des dépendances entre les variantes 92

3 Implémentation de la variabilité de la conception des BD 93

3.1 Implémentation des features : outillage 94

3.2 Configuration des BD 95

3.3 Dérivation des BD : modélisation du problème 97

3.3.1 Passage des FM à la structure d'arbre 97

3.3.2 Annotation de l'arbre de conception et évaluation des chemins 99

3.3.3 Dérivation augmentée de BD : démarche 102

4 Approche par étapes pour la gestion de la variabilité 102

4.1 La gestion de la variabilité de la conception logique 103

4.2 Choix de feature pour incarner la variabilité au sein de la conception logique . . . 104
4.2.1 Classification des types de corrélations . . . 105
4.2.2 Domaines d'application des corrélations . . . 108
4.3 Choix des features pour refléter l'impact de la variabilité logique . 111
4.3.1 Optimisation logique . . . 111
4.3.2 Optimisation physique . . . 113
5 Conclusion . . . 116

Chapitre 4 Impact de la variabilité sur l'optimisation logique 119

1 Introduction . . . 121
2 Génération des schémas logiques ROLAP . . . 123
2.1 Décomposition des dimensions et génération des schémas . . . 124
2.2 Peuplement des tables et calcul de la taille des tables . . . 126
2.2.1 Tables en 2FN (ou plus) . . . 127
2.2.2 Tables en 1FN . . . 129
3 Impact de la variation logique sur l'optimisation logique . . . 130
3.1 Réécriture des requêtes . . . 130
3.2 Hypothèses relatives au développement des MdC . . . 132
3.3 Modèle de coût orienté temps d'exécution . . . 133
3.3.1 Coût de l'opérateur unaire de sélection . . . 134
3.3.2 Coût de l'opérateur unaire de projection . . . 134
3.3.3 Coût de l'opérateur binaire d'une jointure étoile . . . 135
3.3.4 Coût de l'opérateur binaire d'une jointure en flocon de neige . . . 136
3.4 Modèle de coût orienté consommation d'énergie . . . 137
3.4.1 Identification du niveau de modélisation et de la relation entre les paramètres . . . 137

3.4.2 Construction de notre MdC énergétique 139
3.4.3 Validation du modèle de coût 142
3.5 Modèle de coût orienté espace de stockage 142
3.6 Sélection de la meilleure variante 142
3.6.1 Sélection mono-objectif 142
3.6.2 Sélection multi-objectifs : méthode de la somme pondérée 143
4 Validation et expérimentations 144
4.1 Génération des schémas logiques 144
4.2 Évaluation de l'impact en fonction du temps d'exécution 145
4.2.1 Évaluation empirique 145
4.2.2 Analyse des résultats 146
4.2.3 Évaluation théorique 146
4.3 Évaluation de l'impact en fonction de l'énergie 149
4.3.1 Évaluation empirique 150
4.3.2 Évaluation théorique 152
4.4 Évaluation de l'impact en fonction de l'espace de stockage 154
5 Conclusion 155

Chapitre 5 Impact de la variabilité sur l'optimisation physique 157

1 Introduction 159
2 Fondements théoriques 160
3 Adaptation de l'approche de génération de graphe de requêtes 163
3.1 Construction des hypergraphes multidimensionnels 163
3.2 Ordonnancement des sous-hypergraphes 164
3.3 Transformation d'un hypergraphe en graphe 164
4 Sélection de la meilleure variante 165
4.1 Sélection mono-objectif : temps d'exécution 166
4.2 Sélection multi-objectifs : algorithme évolutionnaire 166

5 Validation et expérimentations 168
5.1 Évaluation du scénario mono-objectif 168
5.2 Évaluation du scénario multi-objectifs 169
6 Conclusion 171

Conclusion et perspectives **173**

Bibliographie **181**

Annexes **197**

Annexe A Schémas logiques générés **197**

Annexes **197**

Annexe B Requêtes utilisées **201**
1 Évaluation du temps d'exécution 201
2 Évaluation de l'énergie consommée 203
3 Évaluation de l'optimisation multi-requêtes 214
4 Apprentissage pour le MdC orienté énergie 217

Table des figures **221**

Liste des tableaux **225**

Glossaire **227**

Introduction générale

Le monde dans lequel chacun vit dépend de la façon de le concevoir.

— Arthur Schopenhauer

Ce chapitre préliminaire met en avant le contexte et la problématique qui sous-tendent notre recherche. Ainsi, nous commençons par étudier l'exemple des systèmes de gestion de bases de données pour montrer la diversité et la variabilité omniprésente dans le contexte de conception des bases de données. Nous introduisons ensuite nos contributions qui portent sur la gestion de cette variabilité ainsi que l'organisation du manuscrit.

Contexte

L'histoire des Bases de Données (*BD*) commençant par les systèmes navigationnels à la fin des années 1960 jusqu'aux systèmes les plus sophistiqués d'aujourd'hui, a notamment été marquée par la domination du modèle relationnel. Créé dans les années 1970, il s'est progressivement imposé jusqu'à devenir le paradigme incontournable au début des années 1990, et rester ainsi en tête sans partage pendant deux décennies. Même si plusieurs autres modèles ont entretemps émergé, notamment l'objet (pour les applications de conception/fabrication assistée par ordinateur) et le multidimensionnel (applications décisionnelles), ces derniers n'ont pas pu tenir tête et se sont adaptés pour se conformer au relationnel. Cette adaptation est illustrée par l'apparition des modèles hybrides relationnel-objet et les modèles multidimensionnels relationnels (à l'exemple du schéma en étoile). Dans cette optique, des structures d'optimisation (par exemple les index de jointure binaires et les vues matérialisées), sélectionnées dans la phase physique, ont été proposées pour soutenir le modèle multidimensionnel.

Ce n'est que vers la fin des années 2000, que les 25 ans de certitude du modèle relationnel deviennent réellement menacés par le NoSQL : un nouveau paradigme appuyé par le succès énorme d'Internet, brassant des quantités de données colossales mais surtout variées. Celui-ci

ne semble pas vouloir se plier aux règles relationnelles [1], et remet en question les habitudes ancrées par le règne du relationnel qui semblent être inadaptées à ce nouveau contexte de diversité. Ces règles concernent notamment les décisions prises lors de la conception des *BD* qui relevaient jusque-là de l'approche classique dite (**1-1**), et qui signifie qu'un problème est associé à une solution voire un petit nombre de solutions. Plus concrètement, pour concevoir un modèle conceptuel pour une application de *BD*, nous utilisons systématiquement le modèle Entité Association [2] pour construire son modèle conceptuel et le modèle relationnel pour le modèle logique. L'évolution informatique et technologique impose le passage de l'approche traditionnelle (**1-1**) à celle de (**1-N**) qui signifie qu'un problème peut être associé à une multitude de solutions, en termes de méthodes de conception, de déploiement, et d'implémentation.

Afin de mieux comprendre cette nouvelle Ère de multiplicité de solutions, nous nous sommes concentrés sur les Systèmes de Gestion de Bases de Données (SGBD) hébergeant les *BD*. Nous avons alors tracé l'évolution des 50 ans (1966-2016) des SGBD pour savoir comment ils se sont comportés pour contenir les différentes évolutions de conception de *BD*. Pour ce faire, nous avons mené une enquête basée sur le classement du mois d'octobre 2016 livré par le site spécialisé *DB-Engines* [3], où 316 candidats étaient en lice pour le titre du SGBD le plus populaire. Nous avons alors cherché les années de première parution de chacun [4] et avons distingué trois stades temporels : les deux premières décennies de "maturation" (1966-1986), les deux décennies de maturité (1987-2007) et les dernières années dynamiques (>2008). Les résultats illustrés dans la figure 1 montrent que :

— plus que la moitié des SGBD est inventée pendant le stade actuel [5], ce qui témoigne d'une croissance élevée des inventions et de la vulgarisation multidisciplinaire des *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 ;

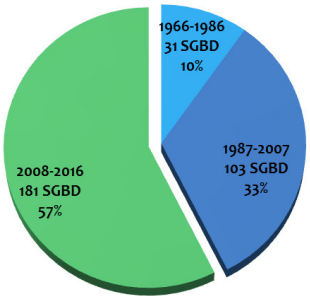
— les types des modèles de données sous-tendant les SGBD se sont multipliés au fur et à mesure que les périodes se succèdent (de 6 à 14). En effet, le paysage des *BD* a connu, ces dernières années, un spectaculaire changement avec de nouveaux fournisseurs et beaucoup de nouvelles technologies qui se 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] ;

1. du moins pour l'instant et après huit ans d'existence.
2. Ce modèle a été inventé dans les années 70 par Peter Chen
3. `http://db-engines.com/en/ranking`, publie un classement mensuel des SGBD depuis 2012.
4. précisément 315 SGBD, n'ayant pas trouvé assez de renseignements concernant le SGBD K-DB `http://db-engines.com/en/system/K-DB`. Il est important de noter qu'à défaut de trouver les années de la mise effective sur le marché, nous nous sommes parfois contentés des années de lancement de projet (conception). Cela concerne surtout les SGBD les plus récents, et a pour effet de faire avancer certaines dates d'environ 2 ans au max.
5. Le pourcentage aurait dépassé les 60% si les dates correspondaient uniquement aux dates de mise effective sur le marché.

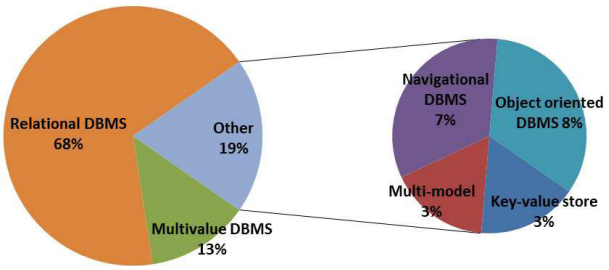
— le modèle relationnel sous-tendait environ 70% des SGBD existants lors du stade initial des SGBD, contre une proportion de 50% lors du stade mature, jusqu'à atteindre seulement 25% pendant cette dernière décennie (depuis 2008). Malgré cela, le relationnel demeure le leader en matière d'usage, grâce aux quatre éditeurs historiques : *Oracle*, *Microsoft*, *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 Top10.

Synthèse

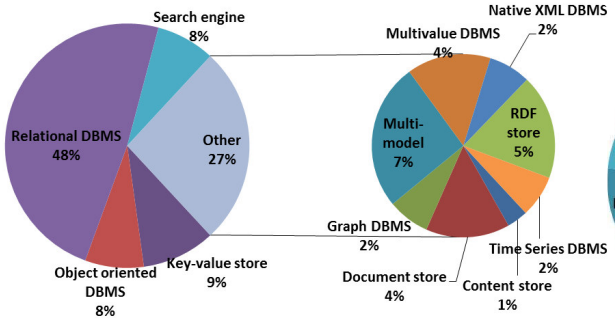
Ces inversions de courbes, cette abondance de solutions et cette diversité, matérialisées par les SGBD, ont été d'abord abstraites et contenues dans le processus triptyque (phase conceptuelle, logique et physique) de conception de *BD*. Elles ne sont que l'image de l'évolution de ce processus qui est créé autour du modèle relationnel et est demeuré stable pendant le stade de maturité. L'analyse temporelle des SGBD que nous avons effectuée est fortement liée aux évolutions de ce processus et les trois stades distinguées correspondent d'ailleurs à ses générations. En effet, le SGBD est un composant logiciel qui implémente, héberge les *BD* et donc reflète leur évolution. Chaque évolution liée au processus de conception de *BD* est traduite dans



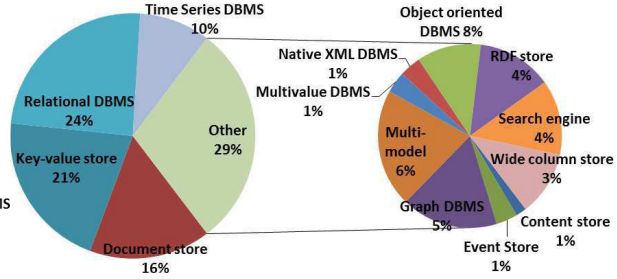
(a) Répartition du nombre de SGBD sur les 3 stades



(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 – Évolution en nombre et en type des SGBD

le SGBD dans les années qui suivent son apparition (à titre d'exemple, le modèle relationnel a été inventé en 1970 et le premier SGBD relationnel est commercialisé en 1979). Afin de soutenir l'évolution multi-échelle des *BD*, ce processus a dû s'enrichir de nouveaux formalismes de conception (p. ex. UML, sémantiques, graphes), acteurs (p. ex. *analystes* et *architectes* de *BD*), modèles (p. ex. multidimensionnel, clé-valeurs), techniques d'optimisations (p. ex. vues matérialisées et indexation), langages de requêtes (p. ex. *SPARQL*, *HiveQL*), outils (p. ex. *advisors* [8, 198, 162]), etc. Cela a clairement pour effet l'augmentation de la complexité de ce processus voire son écartement de son cycle standard. La preuve en est que quelques modèles *non-relationnels* n'exigent carrément pas de modèle de données.

Problématique

Ce processus ne cesse, de ce fait, d'augmenter en complexité et d'exiger plus d'expertise, de temps et de ressources afin de contenir les multiples évolutions et enjeux naissants. Rappelons que, paradoxalement et jusque-là, celui-ci se base essentiellement sur l'intuition, le talent et la connaissance des concepteurs dans la prise des décisions. Ces bases s'avèrent de plus en plus insuffisantes face à la croissante complexité et diversité du processus de conception, en soulevant le problème de la fiabilité et de l'exhaustivité de cette connaissance. D'un autre côté, de nouvelles solutions voient le jour en permanence, dont une grande partie peut être des adaptations d'anciens concepts à de nouveaux compromis, en l'absence d'un framework général.

Ce problème est bien connu sous le nom de la gestion de la variabilité en génie logiciel. Peu explorée dans le contexte des *BD*, elle se définit comme la capacité d'un système à s'adapter, à se configurer et à se spécialiser en fonction du contexte de son utilisation. Appliquée aux SGBD par exemple, elle permet d'offrir une solution médiane entre l'empilement peu performant d'un large éventail de fonctionnalités (*one size fits all*) [177] retrouvé dans les SGBD génériques, et le re-développement coûteux - en partant de zéro - de grandes parties des SGBD (réinvention de la roue), retrouvé dans les solutions à usage spécifique. La gestion de variabilité vient personnaliser les solutions selon le contexte, en se basant sur la réutilisation et la configurabilité des composants communs et variables préalablement définis et implémentés.

Bien que chaque *BD* est différente, par ses données, ses requêtes mais aussi par sa configuration, elles passent toutes par le même processus de conception qui définit cette configuration. Elles y partagent donc un ensemble de caractéristiques et se distinguent les unes des autres par un autre ensemble. Ainsi, les avantages de la gestion de la variabilité doivent être exploités tout au long du processus de conception de *BD*, pour contenir et refléter les évolutions multi-échelle et pas uniquement dans le contexte des SGBD (partie physique).

Nous avons mené une revue de littérature concernant les travaux explorant la gestion de variabilité dans le processus de conception des *BD*. Malgré l'importante présence de la variabilité, il en est ressorti les constats suivants :

1. elle est traitée d'une manière disjointe (mono-phase) pour quelques parties de certaines phases ;
2. la phase physique, et plus particulièrement le SGBD, a bénéficié d'une attention beaucoup plus soutenue que les autres phases ;
3. peu de travaux se sont intéressés à la phase conceptuelle et encore moins à la phase logique. Pour autant que nous sachions, celle-ci n'a pas fait l'objet d'une étude de variabilité, sauf si on lui attribue les rares travaux portant sur les langages de requêtes, un module qui se trouve à cheval entre la phase logique et physique. Les travaux de Rosenmüller et al. [152] en sont un exemple ;
4. l'absence d'une approche holistique, qui traite la variabilité du processus de conception dans son ensemble, pour **(i)** assurer la cohérence et prendre en compte les interdépendances qui lient les différentes phases, **(ii)** présenter au concepteur différents choix disponibles afin d'en choisir le meilleur, contrairement à l'approche classique où il se fiait à son expérience. Cela risquait de le faire passer à côté de ce choix, surtout dans un contexte actuel aussi variable, **(iii)** unifier le processus de conception et ainsi éviter de réinventer des solutions qui existent déjà et **(vi)** automatiser autant que possible ce dernier, afin de réduire les différents coûts.

Objectifs et Contributions

La maturité de la technologie des *BD* acquise grâce à sa vulgarisation dans la majorité des domaines, lui a permis d'être considérée comme un produit logiciel. Cela rend possible l'exploitation des avancées du domaine du génie logiciel dans la résolution des problèmes de *BD*. En effet, comme l'étude de contexte l'a montré, le besoin de gérer la diversité et la variabilité croissante au sein de la conception des *BD* devient un enjeu important. Dans ce mémoire, nous nous sommes donnés pour but de remédier aux lacunes de l'existant, en proposant en premier lieu une méthodologie de conception orientée gestion de variabilité. Pour ce faire, nous adoptons la technique des lignes de produits logiciels (LdP) (Software Product Lines). Celle-ci reprend le fonctionnement des chaînes de production au monde du logiciel, et prend donc en considération les communautés et les variabilités des applications à toutes les étapes de leur cycle de vie et pour toutes les sortes de produits. Ainsi, le développement de chaque produit (en l'occurrence une *BD*) se fait via la composition de fonctionnalités communes. Quant aux fonctionnalités (différences) qui distinguent un produit d'un autre, elles restent ouvertes et personnalisables selon le contexte du produit (BD) en question. Nous appliquons cette technique, réputée pour être efficace dans la maîtrise de la variabilité des BD, sur l'ensemble du processus de conception. Ce caractère holistique permet de: **(i)** considérer les interdépendances entre les phases, **(ii)** offrir une vision globale et "exhaustive" au concepteur, et **(iii)** augmenter l'automatisation du processus. (Chapitre 3).

Le résultat idéal de cette voie de recherche serait une boîte noire configurable, alimentée par

un ensemble d'entrées (notamment requêtes, statistiques et besoins utilisateurs/schéma), pour produire en sortie un script correspondant à une *BD* prête à être déployée dans un environnement prédéfini. L'énorme étendue de l'implémentation (configuration et dérivation des produits) nous impose donc de procéder par étapes dans la réalisation de cette vision, en consacrant cette thèse à l'étude d'un cas précis. L'implémentation finale de notre LdP (problème global) revient alors à généraliser les résultats de différents cas d'études. Nous visons à travers cette étude de cas de :

- ☞ montrer l'importance de la variabilité au niveau de la phase de conception logique, jusque-là sous-étudiée ; (Chapitres 4 et 5)
- ☞ inculquer la culture de la variabilité aux concepteurs en leur offrant une large panoplie de choix de modèles logiques ; (Chapitres 4)
- ☞ exploiter cette variabilité définie au niveau de la phase logique pour mieux optimiser logiquement les requêtes. Dans notre étude, l'optimisation logique concerne les tâches conventionnelles de l'optimiseur de requête sans recours aux structures d'optimisation souvent sélectionnées dans la phase physique. En effet, quand une requête correcte est soumise par un utilisateur, elle est convertie en une représentation interne sous forme d'arbre : plan d'exécution. Celui-ci peut avoir plusieurs représentations équivalentes, générant les mêmes résultats finaux, mais avec des performances différentes. L'optimisation logique vise à obtenir le meilleur plan sans recourir aux structures d'optimisation physiques ; (Chapitre 4)
- ☞ passer de la vision mono-phase de la variabilité à la visions multi-phase en associant la phase physique à la phase logique. Cette association est motivée par l'interdépendance forte entre les phases ; (Chapitres 3 et 5)
- ☞ s'éloigner de l'optimisation physique traditionnelle qui suppose toujours un schéma logique figé, à une optimisation prenant en compte la variabilité des schémas logiques. Dans cette étude, l'optimisation physique est représentée par le processus de sélection des vues matérialisées, une structure d'optimisation de requêtes (Chapitre 5) ;
- ☞ l'évaluation des deux optimisations (logique et physique) est effectuée à l'aide de l'utilisation de modèles de coût mathématiques, estimant les métriques de besoins non fonctionnels, à savoir la performance des requêtes, la consommation d'énergie et l'espace de stockage. Nous tenons à souligner que la gestion de variabilité proposée dans ce manuscrit est supportée par une méthodologie compréhensive et formelle à travers : la modélisation du LdP, la formalisation de la génération des modèles logiques ainsi que l'utilisation des modèles de coûts mathématiques pour les évaluations. Ces derniers sont validées avec des expérimentations empiriques ;
- ☞ par ailleurs, nous soulignons notre intérêt porté aux solutions logicielles écologiques (*Green IT*). En effet, les spécialistes des technologies *BD* ont longtemps accordé la plus grande attention à l'optimisation du temps d'exécution, suivi par le stockage en ignorant l'énergie. Le contexte actuel oblige, plus que jamais, ces spécialistes à intégrer la dimension énergétique, car un SGBD est l'un des principaux consommateurs d'énergie dans les centres de données.

Structure du mémoire

La thèse que nous défendons s'articule autour de trois contributions majeures présentées dans la deuxième partie, la première étant dédiée à l'étude de l'art.

Partie état de l'art

Le chapitre 1 se veut être un recul historique de la conception des *BD*, l'une des étapes clés du cycle de développement des *BD* et l'un des sujets clés de notre travail. Nous présentons les principaux jalons qui ont ponctué l'évolution de ce processus, à savoir les trois stades susmentionnés dans le contexte. Nous y démontrons l'évolution croissante de ce processus et ses conséquences sur le comportement des concepteurs, la vision de la conception, et la genèse de la variabilité.

La suite logique est d'examiner comment cette variabilité a été traitée jusque-là. Plusieurs propositions allant à l'encontre de sa gestion ont déjà été formulées. Par conséquent, le chapitre 2 a pour objectif de présenter un état des lieux des principales techniques visant à gérer la variabilité croissante dans le cycle de conception des *BD*. La notion de la *gestion de variabilité* doit y être abordée dans une acception large, couvrant non seulement les techniques spécialement dédiées du génie logiciel, mais aussi les différents outils, techniques, et concepts qui relèvent pour l'essentiel de la réduction de la complexité du processus de conception. En effet, nous en avons distingué trois axes : aide à la décision, réutilisation, et gestion explicite (configurabilité). Cette revue de littérature s'articule donc en trois temps en associant à chaque fois les travaux existants aux différentes phases du processus afin d'identifier les lacunes actuelles et définir nos objectifs en conséquence.

Partie contributions

Le chapitre 3 est dédié à la définition d'une méthodologie de conception de *BD* orientée gestion de variabilité. Celle-ci est basée sur la technique des lignes de produits logiciels, car elle est réputée pour être efficace dans la maîtrise de la variabilité des *BD*. Le framework résultant permet la modélisation d'un grand ensemble de *BD* comme une ligne de produits. Cela en vue de dériver des *BD* prêtes à être déployées à partir de composants communs et d'autres variables mais configurables selon les besoins de la *BD* en question. Nous suivons des étapes prédéfinies pour l'adoption de cette technique. Nous commençons donc par l'analyse de la variabilité (ingénierie du domaine) de la conception des *BD*. Nous abordons par la suite, l'implémentation de cette variabilité dans un cadre général en présentant ce à quoi doit ressembler le framework dédié. Vu l'étendue de ce dernier, nous le divisons pour mieux appréhender le problème. Nous consacrons alors la dernière section à la présentation d'une partie du problème divisée, illustrée par un cas d'étude. Celui-ci porte sur la variabilité de la conception logique, car la revue de

littérature du chapitre 2 l'a désignée comme étant la moins étudiée parmi les phases du processus de conception des *BD*. Nous présentons en particulier la forme (*normalisation*) et la cause (corrélations) de cette variabilité logique ainsi que les opérations de conception choisies pour refléter l'impact de cette variabilité. Il s'agit de l'*optimisation logique* pour l'impact intraphase et de l'*optimisation physique* pour l'impact interphase. Entretemps, l'ensemble des besoins non fonctionnels, selon lesquels l'impact va être évalué, est décrit.

Le chapitre 4 permet en premier lieu de savoir comment gérer cette variabilité au niveau de la phase logique, en mettant d'abord, à la disposition du concepteur l'ensemble des choix des schémas logiques. Le chapitre se focalise ensuite sur l'impact intraphase, c'est-à-dire l'impact de ces choix de schémas logiques (variabilité logique) sur l'optimisation logique. Afin d'évaluer cet impact selon plusieurs besoins non fonctionnels, nous présentons en second lieu nos modèles de coûts permettant de sélectionner le choix le plus pertinent. Des expérimentations qui viennent valider notre démarche, sont représentées à la fin du chapitre.

Quant à l'impact de la variabilité logique sur l'optimisation physique, celui-ci est décrit dans le chapitre 5. Nous concrétisons ce type d'optimisation par la structure des vues matérialisées. Les fondements théoriques sur lesquels repose notre approche basée sur la structure d'hypergraphe sont d'abord définis. L'approche proposée est ensuite décrite, où nous utilisons les mêmes modèles de coûts pour évaluer l'impact de chaque choix logique sur le rôle des vues matérialisées dans l'optimisation des différents besoins non fonctionnels. Nous présentons à la fin les expérimentations effectuées en vue de la validation de notre approche.

Annexe

Ce mémoire comporte également une annexe qui présente les différentes charges de requêtes utilisées par nos expérimentations, ainsi qu'une autre annexe pour la description des schémas logiques générés.

Publications

La liste suivante présente les publications et communications soumises dans le cadre de cette thèse afin de valider nos travaux. Elles peuvent s'affecter aux chapitres de la manière suivante :

— Chapitre 3 : [33, 102]

— Chapitre 4 : [31, 164, 156, 155]

— Chapitre 5 : [32, 23]

Revue Internationale

1. Amine ROUKH, Ladjel BELLATRECHE, Selma BOUARAR, Ahcène BOUKORCA, Eco-Physic: Eco-Physical Design Initiative for Very Large Databases, submitted to Information Systems Journal, Elsevier (second round) [155].

Conférences Internationales

2. Selma BOUARAR, Ladjel BELLATRECHE, Amine ROUKH, Eco-Data Warehouse Design Through Logical Variability, 43rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM) [32].
3. Ladjel BELLATRECHE, Amine ROUKH, Selma BOUARAR, Step by Step Towards Energy-aware Data Warehouse Design, 5th European Summer School on Business Intelligence (eBISS 2016), Lecture Notes in Business Information Processing, Tours, France, July 3-8, 2016, Tutorial Lectures [23].
4. Selma BOUARAR, Stéphane JEAN, Norbert Siegmund, SPL Driven Approach for Variability in Database Design, The 5th International Conference on Model & Data Engineering. MEDI 2015, pp. 332-342 [33]
5. Amine ROUKH, Ladjel BELLATRECHE, Ahcène BOUKORCA, Selma BOUARAR, Eco-DMW: Eco-Design Methodology for Data warehouses, 18th International Workshop on Data Warehousing and OLAP. DOLAP 2015, pp. 1-10 [156].
6. Selma BOUARAR, Ladjel BELLATRECHE, Stéphane JEAN, Mickael BARON, Do Rule-based Approaches Still Make Sense in Logical Data Warehouse Design?, Proceedings of the 18th East-European Conference on Advances in Databases and Information Systems. ADBIS 2014, pp. 83-96 [31].
7. Selma KHOURI, Ladjel BELLATRECHE, Ilyes BOUKHARI, Selma BOUARAR, More Investment in Conceptual Designers: Think about it!, 15th IEEE International Conference on Computational Science and Engineering. CSE 2012, pp. 88-93 [102].

Conférences Nationales

8. Selma BOUARAR, Ladjel BELLATRECHE, Stéphane JEAN, Mickael BARON, Leveraging Ontology-based Methodologies for Designing Semantic Data Warehouses, 29èmes journées Bases de Données Avancées (BDA), 2013 [164].

Première partie

De la nécessité d'une gestion explicite de la variabilité pour la conception des bases de données

Chapitre 1

Évolution du cycle de conception des bases de données

Sommaire

1 Introduction 15

2 Notions relatives à la conception des BD 17

2.1 Système d'information vs Base de données vs Système de gestion 17

2.2 Cycle de développement des bases de données 18

2.3 Notre positionnement dans ce cycle de développement 19

2.4 Acteurs intervenants dans la conception 20

2.5 Conception et modélisation 20

3 Stade de maturation : architecture 1-tier 21

3.1 Modèles physiques de données et SGBD 21

3.2 L'avènement des modèles logiques 23

3.3 L'avènement des modèles conceptuels 25

4 Stade de maturité : architecture 3-tiers 26

4.1 Cycle triptyque de conception de BD 26

4.2 Évolution horizontale de chaque étape 31

4.3 Évolution verticale : des bases de données aux entrepôts de données 35

5 Stade dynamique : instabilité du cycle 42

5.1 Les principales innovations 42

5.2 La diversité des besoins non fonctionnels 43

6 Conclusion 44

FIGURE 1 – Cafetière de *Carelman*

« Well designed objects are easy to interpret and understand. They contain visible clues to operation. Poorly designed objects can be difficult and frustrating to use »

— D. Norman, The design of Everyday Things

1 Introduction

Depuis la seconde partie du XXe siècle, une véritable révolution numérique ne cesse de déferler sur le monde, en introduisant, à grande échelle, la technologie numérique dans tous les domaines de la vie : économique, social, sanitaire et privé. Cette numérisation consiste à stocker toute information utile afin qu'elle soit efficacement exploitée par différents utilisateurs cibles. Les masses des données engendrées ne se comptent plus, tout autant que les différents nouveaux concepts autour de ce phénomène : la technologie de l'information, révolution informationnelle, troisième révolution industrielle, Big data, etc.

Au centre de ce phénomène se trouvent les bases de données (BD), qui demeurent la solution incontournable pour gérer et interroger ces données stockées, ce qui explique leur apparition quasi-simultanée avec la numérisation, et leur investissement économique très fructueux. En effet, le marché global des BD génère plusieurs milliards de dollars par an ces dernières années et il devrait atteindre, selon IDC [6], 50 milliards de dollars en 2017.

Sous l'effet de la mondialisation et de l'explosion des technologies numériques, de plus en plus d'organisations et d'individus s'intéressent à la collection, au stockage et à l'analyse des données. Le nombre et les types d'applications autour des BD se sont ainsi multipliés. À cela s'ajoute l'évolution à vive allure du matériel (*emerging hardware : GPU et SSD par exemple*), avec de nouvelles technologies apparaissant chaque année.

Afin de soutenir cette évolution multi-échelle, le processus de conception de BD a dû s'enrichir de nouveaux modèles (par exemple, Entité/Association (E/A), UML, logiques, physiques), acteurs (par exemple, analystes et architectes de BD), méthodes (par exemple, OLAP, OLTP, orienté objet, clé-valeurs), techniques d'optimisations (par exemple, vues matérialisées et indexation), langages de requêtes (par exemple, SQL, SPARQL, XQuery), outils (par exemple, les différents systèmes de gestion et advisors [7] [8, 198, 162]) et paradigmes (p. ex. NoSQL).

Ce rythme accéléré laisse présager un besoin continu en termes de nouveaux concepts et d'inventions autour du processus de conception des BD. Ce dernier ne cesse, de ce fait, d'augmenter en complexité et d'exiger plus d'expertise, de temps et de ressources afin de contenir les multiples évolutions et enjeux naissants. Rappelons que, paradoxalement et jusque là, ce processus se base essentiellement sur l'intuition, le talent et l'expertise des concepteurs dans la prise des décisions. Les avantages et les inconvénients de chaque décision sont de moins en moins évidents à soupeser, vu la complexité montante des divers composants inter-connectés de la BD.

Afin de faire gagner du temps, de coût et de qualité aux concepteurs, une stratégie de réutilisation semble nécessaire. Celle-ci doit non seulement tenir compte de la composition du processus, et ainsi des interdépendances entre les phases, mais aussi exploiter ces dernières dans des mécanismes d'aide à la décision. Ces derniers viennent consolider l'intuition et l'expertise

6. http://www.valuewalk.com/2014/07/oracle-corporation-orcl-launches-big-data-sql/
7. outils d'aide assistant les administrateurs de BD confrontés à plusieurs décisions complexes.

des concepteurs pour qu'ils puissent faire de bons compromis. Elle aura également l'avantage de réduire le syndrome de la réinvention de la roue, car de nouvelles solutions voient le jour en permanence, dont une grande partie semble être des adaptations d'anciens concepts à de nouveaux paradigmes.

Plusieurs défis se présentent cependant, qui sont autant de pistes de recherche : allant de l'aide des concepteurs dans la prise de décisions jusqu'à l'abstraction des composantes et des variations du processus de conception. Dans ce chapitre et avant de se lancer dans la résolution de ces différents défis, nous allons passer en revue le processus de conception et retracer ses évolutions les plus marquantes afin de :

— repérer les différentes composantes et points de variations en vue de cette abstraction générale du processus ;

— détecter les éventuelles lacunes de ce processus, pour positionner notre problématique de recherche ;

— prouver le besoin d'un framework général, car l'évolution montre qu'effectivement ce processus a atteint un stade où l'on doit l'abstraire et le (semi-) automatiser.

Actuellement, ce processus de conception possède un cycle composé principalement de trois phases : (1) le design [8] conceptuel, (2) la conception logique, (3) et la conception physique. L'établissement de ce dernier a connu plusieurs étapes d'évolution avant d'être communément admis comme tel par la communauté des BD. Ces étapes coïncident avec les stades de l'évolution des SGBD repérés par notre enquête (voir l'introduction générale), à quelques différences près relatives au décalage temporel qui puisse exister entre la publication de la nouvelle technique (évolution) et de son implémentation par le SGBD. Ces stades sont résumés comme suit.

— **Un stade de maturation** : caractérisé par une focalisation sur l'aspect physique fortement inspiré des systèmes de gestion de fichiers. Ces derniers ont souffert, entre autres, d'un manque de flexibilité et d'expressivité. Un peu plus tard, un aspect logique puis conceptuel se sont respectivement distingués, visant chacun à corriger les lacunes de son prédécesseur. Ces lacunes ont été révélées par l'apparition de nouveaux besoins liés à l'utilisation croissante des BD dans divers domaines.

— **Un stade de maturité** : où les bases de la conception des BD se sont consolidées théoriquement et pratiquement, avec la naissance d'importantes communautés de recherche reconnues autour de conférences internationales (*Requirement Engineering*, INFORSID, ER, CAiSE, VLDB, SIGMOD, BDA entre autres) et industrielle (par exemple, *IBM*, *Oracle*). Ce stade est notamment caractérisé par (i) la définition d'un cycle composé de trois niveaux d'abstraction initialement proposés dans l'architecture ANSI/SPARC : conceptuel, logique et physique, ainsi que (ii) une stabilité relative dans le temps avec une domination des systèmes de gestion relationnels. La stabilité n'exclut pas pour autant l'évolution inévitable des technologies BD, mais garantit que ces évolutions se conforment

8. Nous préférons l'anglicisme *design* que *conception* faute d'équivalent français qui distingue entre la conception au sens général (design) et celle relative au concepts (conception conceptuelle).

au cycle triptyque en l'adaptant à des contextes récents. Cette adaptation se réalise soit par l'ajout de nouvelles variantes à chaque étape (*évolution horizontale*), ou carrément de nouvelles étapes (*évolution verticale*).

— **Un stade dynamique (actuel)** : la stabilité régnante dans l'étape précédente est fortement perturbée par la fréquence élevée des évolutions des technologies qui remettent en cause de plus en plus le cycle standard, ainsi que l'apparition d'autres technologies qui rivalisent avec la domination du relationnel. Les technologies *NoSQL* en sont un parfait exemple, vu qu'ils n'exigent carrément pas de modèle de données.

Nous commençons d'abord par donner quelques définitions afin de mieux cerner le processus de conception dans la section 2. Nous enchaînons ensuite avec la description du stade de maturation de ce processus dans la section 3, puis son stade de maturité dans la section 4. Dans cette même section, nous présentons quelques exemples d'évolutions au sein de chaque étape (évolution horizontale) du cycle *mature* de conception, et nous passons en revue l'une des évolutions verticales majeures (qui a apporté une nouvelle *étape* au cycle : l'intégration), celle des entrepôts de données. La section 5 est dédiée au stade actuel du processus de conception des BD. Nous concluons avec la section 6.

2 Notions relatives à la conception des BD

Nous présentons ci-dessous les principales notions liées au processus de conception des BD.

2.1 Système d'information vs Base de données vs Système de gestion

Avant de rentrer dans le vif du sujet, notons que nous ne pouvons pas parler de BD sans évoquer leur *système de gestion* (*SGBD*) ainsi que leur *système d'information* (*SI*), deux notions qui lui sont inhérentes et qui prêtent souvent à confusion. Par conséquent, rappelons que la *base de données* est un ensemble structuré de données relatives à l'application en question, alors que le SGBD désigne le logiciel permettant de manipuler ces données c'est-à-dire de les stocker, interroger, analyser et mettre à jour. Puis vient le SI qui couvre, quant à lui, un périmètre plus large, celui de l'organisation (et non seulement de l'application/données) avec ses différentes couches : métier, algorithmique, infrastructure humaine et réseau, sécurité, la BD, etc. La figure 1.2 résume superficiellement la relation entre les trois notions.

Nous allons maintenant nous focaliser sur la base de donnée comme entité à part entière, qui est devenue un produit assez complexe, et doit donc suivre les étapes du cycle de développement de n'importe quel système aussi complexe (SI par exemple).

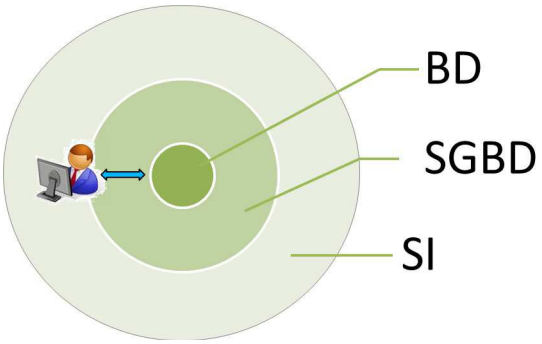


FIGURE 1.2 – Relation entre la BD, le SGBD et le SI

2.2 Cycle de développement des bases de données

La création et l'évolution des SI suivent un modèle itératif appelé le cycle de vie du développement des systèmes, qui est un processus continu de création, de maintien et d'amélioration du SI. Un cycle similaire s'applique aux BD (voir figure 1.3) : celle-ci est créée, maintenue et enrichie en suivant les étapes illustrées dans la figure 1.4, et décrites comme suit.

1. L'objectif général de l'analyse des besoins (étude initiale) consiste à : analyser la situation de l'entreprise, définir les problèmes et les contraintes, fixer les objectifs et définir la portée et les limites de la BD par rapport à l'organisation, les fonctionnalités, le matériel, etc.
2. La deuxième étape s'intéresse à la conception du modèle de la BD censée réaliser les opérations et les objectifs de l'entreprise. Celle-ci est sans doute la phase la plus critique : faire en sorte que le produit final réponde au mieux aux exigences des utilisateurs/systèmes en garantissant intégrité, sécurité, performance et récupération des données en cas

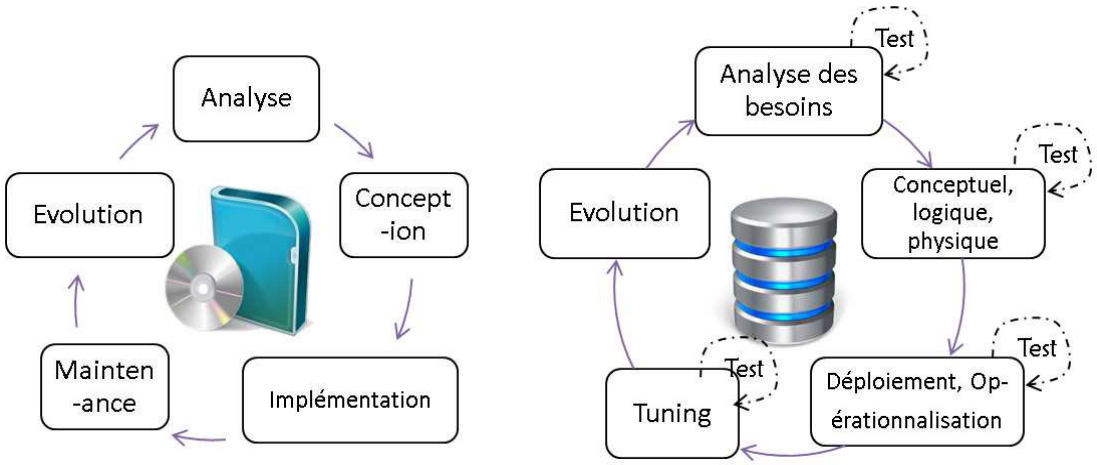


FIGURE 1.3 – Cycle de développement du SI vs. celui de la BD

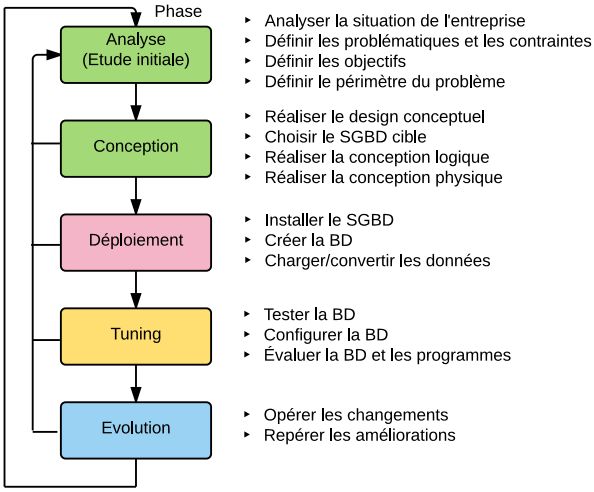


FIGURE 1.4 – Aperçu des étapes du cycle de vie des bases de données [58]

de pépin.

3. L'étape suivante consiste à l'implémentation (*déploiement*) du résultat issu de la précédente phase. Cette mise en oeuvre est faite par une série d'instructions détaillant la création des tables, attributs, domaines, vues, index, contraintes de sécurité/stockage et les directives de performance. L'implémentation inclut l'installation du SGBD, la création de la BD et le chargement/conversion des données.
4. La quatrième étape, à savoir le tuning, consiste en un processus de réglage continu de la BD dont le but est d'atteindre une performance maximale, et ce malgré l'arrivée de nouvelles requêtes/exigences nécessitant des modifications *mineures* dans les composants/-contenu de BD (sauvegarde, récupération, amélioration de la performance en ajoutant des entités/attributs, affectation des autorisations d'accès et leur entretien pour les nouveaux et les anciens utilisateurs, etc.). Ces changements ne peuvent être facilement mises en oeuvre que lorsque la conception est flexible et que la documentation est mise à jour.
5. Quand les changements évolutifs deviennent *majeurs*, l'ensemble du processus doit reprendre à nouveau. À la longue, même la BD la mieux conçue possible ne pourra pas y échapper.

Notons que chacune des étapes décrites précédemment est suivie d'un processus itératif de test, d'évaluation et de modification jusqu'à ce que son objectif soit atteint.

2.3 Notre positionnement dans ce cycle de développement

L'étape cible de notre étude est la deuxième phase de ce cycle, c'est-à-dire la *conception* (*design*), et cela peut se justifier par les raisons suivantes. (i) Elle a toujours été d'une importance vitale, dans tous les domaines, ce qu'illustre la *cafetière de Carelman* (figure 1). Une

bonne conception dans le contexte des BD permet, entre autres, de rendre leur évolution (tuning, administration ou adaptation) plus simple et plus efficace, ce qui fait perdurer leur vie et économiser des frais de mise à jour. De plus, même un très bon SGBD révèlera de piètres performances avec une BD mal conçue. (ii) Les prouesses technologiques et la vulgarisation de la numérisation ont fait exploser la quantité et les types des exigences/applications ce qui, à son tour, a multiplié les méthodes, modèles, outils et paradigmes utilisés dans la conception, et a attisé notre curiosité vis-à-vis de la gestion de cette diversité des choix auxquels les concepteurs sont confrontés. (iii) Étudier la phase de la conception, affecte forcément tout le cycle. En effet, la phase de l'analyse est essentiellement une phase de planification, axée sur la collecte des informations utiles destinées à être l'entrée de la conception. La phase de l'implémentation est plutôt mécanique à travers son processus de chargement, et quant à la phase du tuning, elle consiste à reconfigurer la conception physique de notre BD. Dans la suite de ce chapitre, nous nous focalisons sur la phase de conception qui est basée sur un cycle que nous allons montrer comment il a été créé et comment il a évolué.

2.4 Acteurs intervenants dans la conception

Concernant les intervenants dans le processus du design de BD, on y trouve notamment le concepteur qui, selon la complexité et l'étendue de la BD, peut agir individuellement comme il peut faire partie de toute une équipe de développement composée d'un chef de projet, un ou plusieurs analystes et un ou plusieurs architectes de données. On y trouve également les programmeurs d'applications ainsi que l'administrateur qui, malgré ses responsabilités essentiellement liées au déploiement et au tuning, participe également à la conception physique. Les utilisateurs finaux y interviennent aussi afin de valider le cahier des charges. Tout au long du manuscrit, nous utilisons *Concepteur* comme terme générique pour désigner les différents acteurs et le terme *requête* pour les questions des utilisateurs sur les données.

2.5 Conception et modélisation

Les BD ont dû s'adapter aux différentes évolutions de l'informatique, avant de s'approprier un cycle de conception standard. Un des enjeux majeurs de ce processus est de faire converger les différents points de vue (sur les mêmes données) de ses multiples acteurs, à savoir les concepteurs, programmeurs, administrateurs et utilisateurs finaux de la BD. Ces derniers opèrent sur des aspects/niveaux différents, et doivent donc communiquer pour aboutir à une conception fédératrice reflétant, autant que possible, le fonctionnement réel de l'organisation. Les modèles de données ont été inventés pour simplifier cette communication grâce : (i) à leur simple représentation graphique des structures de données du monde réel de nature complexe, et (ii) aux niveaux d'abstractions permettant d'aborder progressivement les détails selon l'optique des acteurs, ce qui s'avère très utile dans l'intégration de multiples points de vues (voire contradictoires) sur les données à différents niveaux de l'organisation. Ainsi, le concepteur commence

par une simple vue globale des données destinée aux utilisateurs finaux et ajoute des détails au fur et à mesure qu'il acquiert davantage de connaissances et se rapproche donc de l'implémentation confiée à l'administrateur. L'évolution des applications des BD et de leur systèmes de gestion, a toujours été animée par la quête de nouvelles méthodes en vue de modéliser et gérer des données du monde réel, de plus en plus complexes. Plusieurs modèles de données ont ainsi vu le jour, visant chacun à corriger les lacunes de son prédécesseur. La section suivante en donne un aperçu général selon leur ordre de parution chronologique, et montre leur implication dans l'identification des principales étapes du cycle de conception des BD.

3 Stade de maturation : architecture 1-tier

Les BD sont apparues vers les années 1960, et doivent leur origine à leurs prédécesseurs d'une dizaine d'années : les systèmes de gestion de fichiers. Ces derniers permettaient à chaque nouvelle application de créer ses propres fichiers de données et ses propres programmes, et sont d'ailleurs encore utilisés dans des applications à fortes contraintes comme les systèmes embarqués. Cette méthode s'est néanmoins vite révélée inefficace face à la fulgurante croissance du nombre d'informations entraînant redondance, incohérence et lourdeur d'exploitation. C'est en allant à l'encontre de ce procédé : via la centralisation, l'intégration et la structuration de ces informations stockées, que les BD ont pu s'imposer. Ces deux procédés vont de pair : *intégrer* les données *collectées* depuis différents supports d'informations en leur affectant la même description, pour les *fédérer* dans une base *centrale* évitant ainsi redondance et incohérence.

3.1 Modèles physiques de données et SGBD

La structuration (modélisation), quant à elle, a été initialement réalisée grâce à des *modèles physiques de données* (MPD) [51]. Par *modèle*, on désigne un formalisme qui permet d'abstraire la sémantique, originairement complexe, des informations manipulées, des relations/règles qui existent entre elles et des opérations portant sur elles (insertion/mises à jour). Et par *physique*, on entend l'aspect stockage (numérique) de l'ordinateur qui comprend les choix techniques entre objets (structures de données) et contraintes. Cette notion de modèles physiques est souvent liée aux SGBD. En effet, afin de gagner en souplesse, la lourde partie *progammes* permettant de manipuler efficacement les données, a été prise en charge par un système à part, le SGBD, de manière à permettre aux utilisateurs d'interroger leur BD sans se soucier de la position des données sur le support physique, de leur intégrité, de la sécurité, ou de tant d'autres tâches compliquées [132]. Le terme « modèle de données » a été introduit et défini par *Edgar Codd* pour un autre type de modèle de données (logique) [50]. Toutefois, le concept en lui même a émergé un peu plus tôt avec l'apparition de deux principaux modèles physiques ayant eu un réel mais bref succès : le modèle hiérarchique et le modèle en réseau. Très proches du niveau physique, ils se basent sur les systèmes de gestion de fichiers et considèrent donc une BD

comme un ensemble de fichiers reliés par des pointeurs.

Modèle hiérarchique. Le système IMS a été le premier système d'IBM implémentant le modèle hiérarchique en 1968, avec pour langage de manipulation *DL1* [74]. Ce dernier doit sa popularité au fait que le monde réel nous apparait souvent au travers de hiérarchies, ce qui a facilité la transcription de ce monde par ce modèle. Il consiste à stocker les données dans des *segments* comportant un ensemble de champs ayant chacun un type de données. Les segments sont reliés par des liens de 1 vers N qui font correspondre N segments fils à un segment père, c'est pourquoi ils sont organisés sous la forme d'une hiérarchie d'arbre tel que chaque noeud possède un seul parent. En effet, la figure 1.5 montre que le segment *Avion* est implémenté deux fois, car un noeud ne peut pas posséder plus d'un parent. La BD IMS consistait alors en un ensemble d'instances de ces arbres de segments, encore appelé *forêt*. Ce modèle en arbre possède deux inconvénients majeurs [177] : (1) redondance de l'information qui entraîne, à terme, des incohérences en modification, insertion et suppression de données, rendant la base peu performante, (2) La liaison père-fils (1-N) est de type *fort* c'est-à-dire que les données d'un segment fils ne peuvent exister sauf si celles de son parent existent.

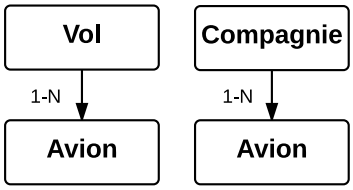


FIGURE 1.5 – Exemple d'une représentation hiérarchique

Modèle réseau. La nécessité de lier des arbres via des associations 1-N sans avoir à dupliquer les segments (lien *frère* pour ainsi diminuer la redondance), a fait naître le modèle de données réseau proposé en 1969 par le groupe DBTG du comité *CODASYL* (Committee on Data Systems Languages) permettant des organisations de liens plus générales et complexes que son prédécesseur. Plusieurs versions se sont succédées dont une version *CODASYL 1971* [179] qui reste encore aujourd'hui utilisée par certains systèmes. Le modèle CODASYL stocke les données dans une collection d'enregistrements reliés par des liens 1-N, non pas dans un arbre, mais dans un réseau ou graphe dont les noeuds sont des enregistrements. Ce modèle permet ainsi à une instance d'un enregistrement d'avoir plusieurs parents et non plus un parent unique (voir la figure 1.6 où l'enregistrement *Avion* possède effectivement deux parents : *Vol* et *Compagnie*). 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 pour aboutir à l'enregistrement contenant la donnée. Ce modèle offre une meilleure représentation des données et davantage de flexibilité que le modèle hiérarchique, ce qui a valu à son auteur Charles Bachman d'avoir le prix ACM Turing en 1973. Il comporte cependant de nombreux inconvénients comme : (1) la difficulté

d'installation et de gestion du système à cause du temps nécessaire pour l'organisation des données, (2) la redondance qui compliquait la mise à jour ne serait-ce que d'une seule structure (3) le système de navigation le long des chemins d'accès pour atteindre une donnée cible s'avérait de plus en plus compliqué pour les programmeurs d'autant plus que le volume des données augmentait [53].

Compte tenu de l'accroissement spectaculaire des besoins en informations et du matériel informatique, la nécessité d'applications et de BD plus sophistiquées s'est imposée, d'où la désuétude de ces deux modèles. Leur manque d'expressivité, de souplesse et de flexibilité engendré par la faible indépendance des programmes aux données en est la principale cause. En effet, afin d'éviter de recoder les éventuels mises à jour ou changements, ces modèles impliquent de prévoir toutes les données et opérations nécessaires avant la conception de la BD, ce qui est aujourd'hui inconcevable. Aussi, la complexité des langages de manipulation basés sur la navigation, est un reproche que l'on peut faire à ces modèles [72].

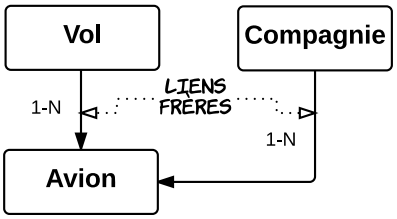


FIGURE 1.6 – Exemple d'une représentation réseau

3.2 L'avènement des modèles logiques

L'indépendance physique des données était une nécessité car une application de BD n'est jamais écrite dès le premier jet, mais requiert plusieurs mises à jour. Le modèle qui suivit les deux modèles présentés précédemment, a été proposé avec pour principal objectif d'offrir une séparation entre les deux niveaux d'abstraction des données : le niveau logique et le niveau physique [177]. C'est ainsi qu'*Edgar Codd* a proposé en 1970 le modèle *relationnel* [50] présenta une abstraction mathématique (algèbre relationnelle) de ce qui est implémenté dans la BD. La composante clé du modèle relationnel est la table (relation) : une structure simple et flexible qui stocke des instances homogènes. En ce point, la table se distingue du fichier, par sa nature *logique* qui garantit une indépendance totale entre les données et l'implémentation. En termes concrets, le concepteur ne se préoccupe guère du stockage physique des données dans la BD, il se focalise plutôt sur la structure, et c'est cette caractéristique qui a fait du modèle relationnel une véritable révolution. Ainsi, la gestion des données est réalisée par leur valeur et non par leur position souvent difficile à maintenir dans les modèles réseaux. Le modèle relationnel doit également son succès à la puissance et à la flexibilité de son langage de requêtes *SQL* qui permet à l'utilisateur de spécifier ce qui doit être fait, sans préciser comment. Il revient à la charge du *moteur SQL*; qui fait désormais partie du SGBD relationnel (SGBDR); de traduire

en coulisse les requêtes des utilisateurs en instructions pour récupérer les données demandées avec beaucoup moins d'efforts que tout autre fichier ou environnement de BD. *Edgar Codd* a distingué trois composantes devenues désormais indispensables pour tout type de modèle de données [53].

— *Partie structurelle* : représente la collection de types de structures de données. La partie structurelle dans le modèle relationnel est représentée par les domaines, relations (table de plusieurs lignes et colonnes), attributs (souvent des colonnes), tuples (lignes), clés candidates, primaires et étrangères.

— *Partie manipulation* : collection d'opérateurs et de règles d'inférence qui peuvent être appliqués sur des instances valides de la BD pour leur restitution, leur modification ou leur gestion. La partie manipulation dans le modèle relationnel est représentée par les opérateurs algébriques (select, project, join, etc) qui transforment des relations en d'autres relations.

— *Partie intégrité* : collection de règles d'intégrité qui définissent implicitement ou explicitement l'ensemble des états consistants de la base. La partie intégrité dans le modèle relationnel est représentée par les contraintes d'intégrité d'entités et d'intégrité référentielle.

L'ensemble de ces contributions ont valu à *Edgar Codd* de recevoir le prix Turing en 1981 [53]. À ses débuts, et malgré ses avantages, le modèle relationnel était considéré ingénieux mais irréalisable, surtout qu'à cette époque, les ordinateurs manquaient de puissance requise pour son implémentation. Un grand débat eut d'ailleurs lieu dans les années 70 dans de grandes conférences sur les BD (comme *SIGFIDET* et son successeur *SIGMOD*) opposant les pro-CODASYL menés par Charles Bachman et les pro-relationnel menés par Edgar Codd [177]. Des études ont montré la supériorité des performances des BD relationnelles comparées à celles des BD orientées enregistrements [79]. Ces études ont servi de base pour le développement de deux prototypes majeurs de SGBDR proposés entre 1974-1977 : *Ingres* initié par l'université de Berkeley et *SystemR* initié par le groupe de recherche d'*IBM*. Ces projets ont été les précurseurs de plusieurs champs de recherches du domaine des BD et de plusieurs contributions comme le développement des langages de requêtes, la proposition des *propriétés ACID* (acronyme pour Atomicité, Consistance, Isolation et Durabilité) pour les transactions de BD, l'optimisation des requêtes ou l'indépendance des données et les vues [79, 5].

En 1979, *Larry Ellison*, ancien PDG d'Oracle, suivant de près le travail des chercheurs d'IBM et de l'Université de Berkeley, a publié le premier SGBD basé sur le modèle relationnel. Fort heureusement, la puissance des ordinateurs a augmenté de façon exponentielle, tout comme l'efficacité des systèmes d'exploitation. Mieux encore, leur coût diminuait à mesure que leur puissance augmentait. Le débat entre les pro-CODASYL et les pro-relationnel prit fin en 1984 lorsqu'IBM, grand détenteur du marché des SGBD, annonça l'apparition de son SGBD DB2 (successeur d'IMS) développé selon le modèle relationnel. Le langage *SQL* (structured Query

Language) a été adopté comme langage standard d'interrogation et de manipulation du modèle relationnel [177].

3.3 L'avènement des modèles conceptuels

Encore une fois, les exigences accrues des transactions et d'informations ont fait augmenter la complexité des structures d'implémentation, créant ainsi un besoin d'outils de conception encore plus sophistiqués. Dans ce nouveau contexte, le modèle relationnel s'est révélé insuffisant pour réponde aux attentes des concepteurs de BD qui voulaient désormais exprimer plus de sémantique, notamment celle dégagée par les règles de gestion, considérer plus de formats possibles, et surtout avoir une méthodologie unifiée pour la conception de leurs BD [47]. Ainsi, le modèle *entités/associations* (E/A) fut proposé par *Peter Chen* en 1976 [46] en complément au modèle relationnel. Ils constituent ensemble une base pour une conception structurée, d'où sa notoriété comme étant le standard le plus largement accepté en matière de modélisation de données. En effet, le modèle E/A (i) apporte davantage de sémantique que le modèle relationnel, à titre d'exemple la représentation des cardinalités, et le lien explicite entre les entités qui est implicitement représenté dans le modèle relationnel, (ii) se base sur la représentation graphique grâce à son diagramme E/A qui est, d'un côté, plus compréhensible par les utilisateurs (voir figure 1.7), et d'un autre, plus intuitif à l'égard des concepteurs qui trouvent facile d'examiner graphiquement les structures que de les décrire formellement ou textuellement, (iii) se base sur la théorie des ensembles et des relations mathématiques, tout comme son prédécesseur, dont il se distingue pourtant au niveau de la formalisation mathématique. En termes concrets, le premier utilise le produit cartésien (constructeur de relation) pour décrire la structure des *valeurs de données* : une table est alors définie comme un produit cartésien des domaines de ses attributs. Le modèle E/A par contre, utilise le même constructeur pour décrire la structure *des entités*. Dans le modèle E/A, une relation est le produit cartésien des entités [47] et (iv) transcende, de ce fait, la couche logique du modèle relationnel avec une couche supérieure dite *conceptuelle* où il traite les entités du monde réel et ne considère pas seulement les valeurs de données. De la même façon que le niveau logique a offert une indépendance physique (matérielle) à ses modèles, le niveau conceptuel offre également une indépendance *logicielle* (logique) en plus de la physique bien entendu. L'indépendance logique (respectivement physique) est illustrée par le fait que les modèles conceptuels peuvent être traduits dans n'importe quel modèle logique (respectivement physique). Autrement dit, le modèle conceptuel ne dépend ni du matériel physique, ni du SGBD utilisé dans son implémentation : une caractéristique pertinente compte tenu des divers choix de SBGD disponibles, et abordés un peu plus loin dans ce chapitre. Le niveau d'abstraction (conceptuel) introduit par le modèle E/A a marqué une nouvelle évolution de la conception des BD. Proche du monde réel et des clients, il est rapidement devenu très utilisé par les outils de conception où la traduction du schéma E/A en un ensemble de tables relationnelles

a été automatisée. *ER/Studio* [9] et *ERwin* [10] sont deux exemples de ces outils.

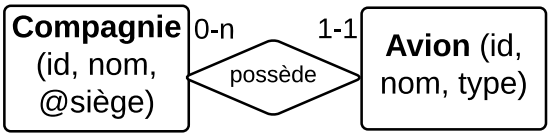


FIGURE 1.7 – Un exemple de modélisation entités/associations

Comme on voit dans cette section, le cycle de conception des BD n'a été initialement composé que d'une seule phase. La conception a donc été plus simplifiée comparée à celle d'aujourd'hui. Ce cycle s'est vu progressivement intégrer d'autres phases et évoluer, ce qui lui a permis d'atteindre un stade de maturité plus stable que nous allons décrire dans la section ci-dessous.

4 Stade de maturité : architecture 3-tiers

Ce stade doit sa maturité à son cycle bien défini, autour duquel des évolutions viennent se greffer. Cela a apporté à ce stade une certaine stabilité dans le temps avec une domination des systèmes de gestion relationnels. La stabilité n'exclut pas pour autant l'évolution inévitable des technologies BD, mais garantit que ces évolutions se conforment au cycle triptyque en l'adaptant à des contextes récents. Cette adaptation se réalise par l'ajout soit de nouvelles variantes à chaque étape (*évolution horizontale*), soit carrément de nouvelles étapes (*évolution verticale*) (voir figure 1.13), comme nous allons le voir dans la suite de ce chapitre.

4.1 Cycle triptyque de conception de BD

Le comité de planification des standards (SPARC) de l'Institut National des Standards Américains (ANSI) a défini, au milieu des années 1970, un framework dit *ANSI/SPARC* pour la modélisation des données [186]. Cette architecture universellement admise dans la communauté des BD vient résumer l'évolution *verticale* de notre cycle de conception, qui distingue les quatre niveaux suivants (illustrés dans la figure 1.8).

— Un modèle *externe* représente la vue que possède une certaine catégorie d'utilisateurs finaux sur l'environnement des données. En effet, les entreprises sont généralement divisées en plusieurs unités (finance, ventes, ressources humaines, entre autres), et chacune possède des contraintes, des exigences, des règles de gestion et donc un modèle propre à elle et externe aux autres. Cette séparation simplifie le travail des concepteurs et des programmeurs et aide à garantir des contraintes de sécurité au sein de la BD.

9. https://www.idera.com/er-studio-enterprise-data-modeling-and-architecture-tools
10. http://erwin.com/

— Le modèle *conceptuel* représente quant à lui, une vue à vol d'oiseau de toute la BD, indépendante de toute configuration matérielle ou logicielle, en intégrant ses différentes vues externes (entités, relations, contraintes, et processus). Les modèles conceptuels et externes utilisent le diagramme E/A pour leur présentation graphique, et appartiennent au même niveau d'abstraction avec des envergures différentes.

— Une fois que le SGBD est choisi, le modèle *interne* mappe les composantes du modèle conceptuel y compris ses contraintes, à leur composantes correspondantes dans le SGBD en utilisant ses constructeurs. S'il s'agit d'un SGBDR par exemple, les entités sont mappées en tables, et le schéma interne est exprimé via SQL. Cette dépendance logicielle entraine un changement du modèle interne à la moindre modification du SGBD. En revanche, une indépendance *logique* est garantie du fait que le modèle interne peut être modifié sans affecter son homologue conceptuel, ainsi qu'une indépendance matérielle, qui de la même manière, épargne le modèle interne de tout changement au niveau des supports de stockage et voire même au niveau du système d'exploitation de l'ordinateur hôte.

— Le modèle *physique* concerne le niveau d'abstraction le plus bas, en décrivant les méthodes d'accès aux données et comment ces dernières vont être stockées sur les médias de stockage (bandes ou disques magnétiques, flash, etc.), engendrant ainsi une dépendance logicielle (SGBD et système d'exploitation) et matérielle (le type des structures de stockage que l'ordinateur peut gérer). C'est pourquoi la conception physique exige aux concepteurs une connaissance détaillée du logiciel et du matériel [65].

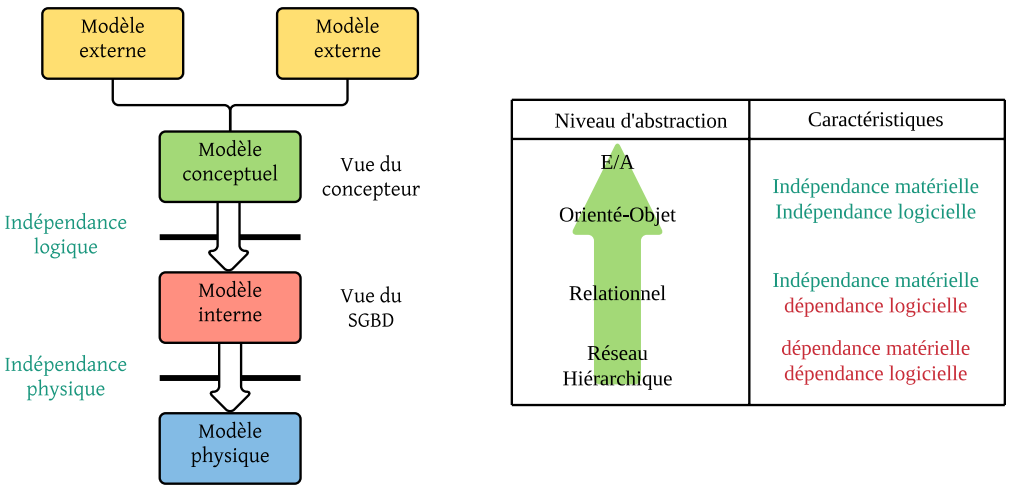


FIGURE 1.8 – L'architecture ANSI/SPARC

Il est important de noter que l'architecture ANSI/SPARC, dans sa version basique, n'évoque pas explicitement le modèle logique et se contente du modèle interne pour désigner les deux : logique et physique. Nous avons bien détaillé les deux en les séparant afin de montrer comment

l'architecture standard a servi de base pour la définition des étapes du cycle de conception de BD se composant principalement de trois étapes [58] : la phase du design conceptuel, la phase de conception logique après avoir fait le choix crucial du SGBD et enfin la phase de conception physique. Grosso modo, on peut considérer le modèle conceptuel comme l'ensemble de données telles que vues par les utilisateurs finaux, le modèle logique comme les données telles que vues par le SGBD, et le modèle physique comme les données telles que vues par le gestionnaire des périphériques de stockage du système d'exploitation sous-jacent. Le fruit de ce processus est une BD prête à être implémentée. Nous décrivons dans ce qui suit plus en détails ces trois principales étapes.

4.1.1 Design conceptuel

Le fruit du *design conceptuel* est *un modèle conceptuel de données* (MCD) qui décrit graphiquement (par une abstraction) et textuellement les principales entités, attributs, relations et contraintes du monde réel de la façon la plus réaliste possible. Pour ce faire, les concepteurs utilisent des constructeurs fournis par un modèle de données d'un haut niveau d'abstraction, indépendant donc de toute contrainte d'implémentation. Ils doivent également se baser sur la spécification des besoins collectés auprès des utilisateurs du système pendant la phase d'analyse (voir section 1.2), à laquelle le domaine de l'ingénierie des besoins s'est associé. D'ailleurs, nombreux sont les concepteurs qui ont tendance à considérer la phase d'analyse (ou du moins une partie d'elle) comme première étape du processus de conception, vu leur chevauchement [58]. Parmi les modèles de haut niveau les plus répandus, on peut citer : *E/A*, le *digramme de classes* d'UML [11] et le modèle *Express* [12].

Parallèlement, les programmeurs et analystes doivent analyser la partie applicative qui couvre les procédures aidant à réaliser les traitements de la BD et/ou transformer les données en informations utiles et exploitables. Cela peut inclure la tâche d'identification des processus qui consistent en un ensemble d'opérations complexes. Les entrées/sorties ainsi que le comportement fonctionnel de chacune des procédures/opérations doivent être spécifiés [65]. Des outils et 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* qui permettent de représenter les traitements de l'application au niveau conceptuel comme le modèle conceptuel de traitements. La conception doit également tenir compte des modifications futurs, et veiller à ce que l'investissement des entreprises dans les ressources d'information perdurera. Tout ce processus peut se résumer en trois tâches : (i) analyse de données et d'exigences, (ii) modélisation et normalisation, (iii) vérification/test du MCD.

Selon l'étendue du système, la structure centralisée ou distribuée de la BD et les préférences du concepteur, il existe deux approches classiques de conception : (i) une approche descendante

11. http://www.uml.org/
12. http://www.omg.org/spec/EXPRESS/

(top-down) qui commence par identifier les entités (ensembles de donnée) puis définit les attributs (éléments) pour chaque entité, ou inversement (ii) une approche ascendante (bottom-up) où l'on commence par identifier les attributs puis on les regroupe dans des entités. Cela dit, les deux approches sont complémentaires puisque l'approche ascendante est souvent utilisée pour les BD distribuées ou de petites BD avec peu d'entités, attributs, relations et transactions, et l'approche descendante dans le cas contraire [58].

4.1.2 Conception logique

Comme mentionné auparavant, l'objectif de cette étape est de concevoir une BD à l'échelle de l'entreprise basée sur un modèle de données particulier qui soit indépendant de tout détail physique d'implémentation. Concrètement, le concepteur établit un *modèle logique de données* (MLD) à partir du MCD. Pour ce faire, il existe des étapes à suivre de façon itérative. (i) Tous les objets (entités et contraintes) du MCD sont traduits vers les constructeurs du modèle logique choisi. Cette traduction se fait de manière directe voire automatique, en suivant des règles de traduction prédéfinies. (ii) Valider le MLD par la normalisation et l'intégrité référentielle surtout dans le cas des modèles relationnels, et d'une manière générale, vérifier sa cohérence après la phase de traduction, qui peut être amenée à ajouter de nouveaux attributs voire de nouvelles tables. (iii) Valider les contraintes d'intégrité : définition des domaines des attributs, des énumérations, des attributs obligatoires et optionnels, des droits d'accès, etc. La dernière étape est le test qui confronte le MLD obtenu aux besoins des utilisateurs en termes de données, transactions, et de sécurité.

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. 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* [13] et *Rational Rose* [14].

Notons que dans cette phase, le choix du SGBD est d'une importance capitale pour le système d'information pour la bonne et simple raison qu'il décide des pouvoirs et des limites de ce dernier. Les facteurs qui influencent la décision d'achat du SGBD varient d'une entreprise à l'autre et peuvent être d'ordre techniques, économiques ou stratégiques ; en voici les plus courants : (i) le coût qui comprend le prix d'achat initial, ainsi que celui de la maintenance et de la formation, (ii) les fonctionnalités et les outils du SGBD dédiés à l'administrateur (par exemple, les advisors), aux utilisateurs et programmeurs par exemple, l'interrogation par l'exemple ou QBE, générateurs de rapports, dictionnaires de données), à la garantie de la performance et de la sécurité lors du traitement des transactions et du contrôle de concurrence, et en général à la facilité de l'utilisation, (iii) le modèle sous-jacent qui peut être hiérarchique, réseau, relationnel, objet, clé-valeur, etc. (iv) la portabilité à travers les plateformes, systèmes d'exploitation et langages et (v) les exigences matérielles concernant entre autres les processeurs, la RAM, l'espace

13. http://www.bpmmicro.com/downloads/
14. http://www-03.ibm.com/software/products/fr/enterprise

disque.

Notons que ce choix est considéré comme une étape du cycle de conception par certains chercheurs, vu son importance [58]. Il peut avoir lieu à différents endroits du cycle de conception de BD, même si le plus courant est dans la phase de conception logique. Il peut aussi être une contrainte imposée par l'entreprise à respecter dès le début de la conception ; comme il peut également s'effectuer au début de la phase de déploiement et relever donc de la responsabilité de l'administrateur qui doit choisir la plateforme répondant le mieux aux besoins de l'application. Ce scénario est rendu possible grâce à la multiplication des SGBD, permettant une conception de BD d'une manière générique pouvant être déployée sur plusieurs SGBD.

4.1.3 Conception physique

La conception physique est un processus technique qui gère non seulement l'accessibilité aux données présentes sur le(s) support(s) de stockage mais surtout la *performance* du système, l'intégrité et la sécurité des données. Elle s'appuie sur le MLD et fournit *le modèle physique des données* (MPD) qui dépend du SGBD, du système d'exploitation, et du matériel de stockage. Elle comprend les étapes suivantes : (i) définir l'organisation du stockage des données qui passe par l'évaluation du volume, la fréquence des mises à jours, la localisation des tables, l'organisation éventuelle des *tablespaces*, l'identification des index, de leur types et des vues matérialisées [84], (ii) définir les mesures de sécurité et d'intégrité comme, entre autres, les groupes des utilisateurs, et l'ensemble des privilèges affectés à chaque individu ou groupe (iii) déterminer les mesures de performance (y compris les structures d'optimisation, voir paragraphe 1.4.2.3) en prenant en compte des caractéristiques du support de stockage, tels que le temps de recherche, la taille du secteur/bloc (page), la taille de la mémoire tampon, et le nombre de plateaux de disque et de têtes de lecture/écriture.

Afin de respecter les trois principaux niveaux imposés par l'architecture ANSI/SPARC, deux principaux langages de données sont utilisés pour assurer l'indépendance logique et physique des données : le LDD est utilisé pour spécifier le schéma conceptuel et logique, le *langage de définition du stockage* (LDS) est utilisé pour spécifier le schéma interne. Dans la plupart des SGBD actuels, il n'existe pas de langage spécifique qui joue le rôle de LDS. Le schéma interne est alors défini par une combinaison de fonctions, de paramètres et de spécifications relatives au stockage des données. Un troisième langage devrait également exister pour définir les vues des utilisateurs ; mais la plupart des SGBD utilisent le LDD pour définir les schémas conceptuels et externes. Dans les SGBD actuels, les trois types de langages cités ne sont généralement pas considérés comme des langages distincts, mais un seul langage global et intégré ayant des constructeurs pour la définition des différents schémas. Pour l'exemple des BD relationnelles, le langage SQL est utilisé en tant que langage de définition des données, de définition des vues et de manipulation des données. Le LDS est un langage qui existait dans les premières versions de SQL, mais qui a été retiré par la suite [65].

Maintenant que nous avons présenté en détails le cycle de conception des BD, qui est au centre de nos travaux, nous décrivons les deux types d'évolution qu'il a subi.

4.2 Évolution horizontale de chaque étape

La complexité croissante des problèmes du monde réel justifie le besoin continu en termes de modèles, d'outils et de techniques à la hauteur de représenter ce monde le plus fidèlement possible. Ce besoin s'est matérialisé par une multitude de propositions et de choix dans chaque niveau d'abstraction (étape du cycle de conception) afin de répondre à des types spécifiques d'applications. Ainsi, cette diversification multi-niveaux constitue un type spécifique d'évolution qui touche le cycle de conception des BD et que nous appelons évolution *horizontale* (voir figure 1.13). Dans ce qui suit, nous présentons quelques-unes de ces évolutions, sans forcément respecter leur ordre chronologique.

4.2.1 Variation au sein du design conceptuel

Plusieurs grandes conférences ont été organisées autour du thème de la modélisation conceptuelle comme la *conférence ER* dont la première édition a été organisée en 1979. À cette époque, de nouvelles applications de BD avaient déjà émergé comme par exemple la conception et fabrication assistées par ordinateur (CFAO), les télécommunications et les systèmes d'information géographiques (SIG). Ayant des exigences plus complexes, ce type d'application a entraîné l'apparition d'autres modèles dits *sémantiques* qui se voulaient être plus expressifs que le modèle relationnel [171, 87, 47]. En effet, la sémantique d'un concept ou d'un objet métier ne s'arrête pas aux propriétés descriptives, mais doit couvrir également les propriétés dynamiques, relatives à ce que l'objet en question peut faire et comment il se comporte. Les modèles les plus marquants de cette famille sont les modèles proposés par *Smith et al.* [171] et aussi *Hammer et McLeod* [87]. Dans la même lignée, les modèles objets ont suivi où effectivement, à la différence d'une entité du modèle E/A plutôt statique [47], une classe (ensemble d'objets) du modèle objet est plus autonome. En effet, elle n'est pas uniquement décrite par son contenu factuel, mais aussi par son aspect comportemental à travers les méthodes d'affichage et de mise à jour entre autres. Une autre dimension importante ajoutée par les modèles objet est l'*héritage*, une notion permettant de représenter des catégories d'objets organisées hiérarchiquement très présentes dans le monde réel. UML (*Unified Modeling Language*) a été défini comme langage de modélisation des modèles objet et a largement inspiré son prédécesseur E/A qui a été étendu dans une version sémantique appelée *extended entity relationship* (EER), plus expressive que la version originale [47].

Au milieu des années 1990, l'utilisation des objets complexes a encore pris un nouvel élan avec la révolution *internet* comme moyen de communication novateur. Dans ce contexte, le *langage de balisage extensible* (XML) est devenu la norme de facto pour l'échange efficace des données structurées, semi et non structurées. Il combine les concepts des modèles utilisés dans

les systèmes de documents avec des concepts de modélisation de BD [65]. Dans un modèle classique, le schéma de données est d'abord défini. Les instances sont créées conformément à ce schéma. Dans un modèle XML (par exemple, XMLSchema), le schéma n'a pas besoin d'être spécifié à l'avance [177]. Les instances se décrivent par elles-mêmes, où chaque attribut est marqué par une métadonnée décrivant le sens de cet attribut. Cette famille de modèles marque un retour vers les modèles conceptuels. *Peter Chen*, développeur du modèle E/A, a d'ailleurs exercé comme expert pour le groupe XML [177].

Durant la même décennie, la quantité des données stockées gérées par les applications (ingénierie/internet) est devenue importante et répartie sur plusieurs sources. Dans le but d'exploiter en même temps ces différentes données hétérogènes issues de diverses sources, des systèmes d'intégration ont été proposés, et la notion du schéma global a été ajoutée comme une vue sur les schémas locaux des sources. Cela a permis de résoudre les conflits structurels, mais pas les conflits sémantiques (nommage, contextes et mesures) résolus quant à eux grâce à la notion d'*ontologie* définie *comme une spécification formelle et explicite d'une conceptualisation partagée d'un domaine de connaissance* [83]. Des modèles *ontologiques* du web sémantique ont ainsi vu le jour comme par exemple *RDF*, *RDFS*, *OIL*, *DAML* ou *OWL*. *Tim Berners Lee*, qui a participé au développement du modèle *RDF*, considère le modèle comme dérivé de la famille des modèles E/A ou plus généralement des modèles conceptuels. L'utilisation intense, ces dernières années, des ontologies a créé un nouveau type de données appelé : *données ontologiques* ou *sémantiques* car elles référencent ces ontologies. Au début de leur apparition, leur gestion se faisait en mémoire centrale par des systèmes à l'exemple de *OWLIM* [108] ou *Jena* [41]. Afin d'assurer le passage à l'échelle, des mécanismes efficaces pour leur stockage et leur manipulation ont été proposés par de nombreux systèmes offrant une nouvelle architecture appelée *Base de Données à Base Ontologique* (BDBO). Celle-ci persiste toute la sémantique au sein de la BD sous forme d'une ontologie locale (figure 1.9) qui peut être vue comme le modèle conceptuel de la BD, ou utilisée pour l'extraire. Cela a marqué une nouvelle évolution qui consiste à persister le modèle conceptuel au sein de la BD tout comme son homologue logique. Par conséquent, des langages d'exploitation de BDBO sont apparus afin de gérer cette nouvelle couche persistée, comme notamment le langage *SPARQL* adopté par le consortium W3C comme le langage standard des BDBO [143].

La phase du design conceptuel a ainsi eu de nombreuses variantes depuis son introduction. C'est également le cas de la conception logique comme nous le voyons dans la section suivante.

4.2.2 Variation au sein de la conception logique

La nature cyclique du processus de conception des BD fait qu'une étape doit s'adapter aux changements de celle qui la précède. En effet, afin d'exporter la sémantique des nouveaux modèles conceptuels au niveau logique (au sein de la BD), le modèle relationnel a été étendu pour pouvoir représenter les nouvelles structures complexes (images, audio et vidéo). C'est ainsi

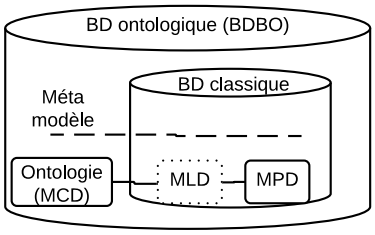


FIGURE 1.9 – BDBO vs. BD

qu'est née une nouvelle génération de BDR qui prend en charge les spécificités du modèle objet à savoir l'encapsulation des données/méthodes, l'héritage, les types de données extensibles basés sur la notion de classe voire même de document XML. D'une manière générale et grâce à sa base mathématique solide, le modèle relationnel peut être étendu pour inclure de nouvelles fonctionnalités. Cela se fait sous forme de modules optionnels, destinées à un type d'application donné comme les applications spatiales ou de traitements d'images [65]. Les SGBD correspondants sont décrits comme objet/relationnel, et, offrent la possibilité aux utilisateurs de définir leur propres types de données, opérateurs, fonctions et méthodes [65, 167].

En revanche, les SGBD purement objet ont connu un grand intérêt dans le milieu des années 80, mais semblent aujourd'hui cantonnés à des niches de marché liées à des applications répondant à des besoins spécifiques exigeant des objets encore plus complexes (par exemple, CFAO et SIG). À noter que le premier objectif des SGBD objet était de gérer le problème de dysfonctionnement ou *impedance mismatch* entre les langages de programmation orientés objet (C++) et les BDR [177]. Il s'agissait d'adapter le langage de programmation au langage du modèle de données, ou le contraire, afin de persister les objets et les structures de données utilisés par les développeurs. Ces derniers étaient habituellement soit perdus une fois que le programme se termine, soit sauvegardés par les fichiers [65] entraînant les inconvénients évoqués précédemment (voir section 1.3.1).

4.2.3 Variation au sein de la conception physique

L'évolution des modèles physiques de données concerne plusieurs aspects : matériel de stockage et de traitement, paradigmes et architecture de déploiement et enfin l'optimisation des requêtes. Les principales motivations de ces évolutions sont la sécurité et la performance qui consistent à réduire le temps de réponse du SGBD aux requêtes des utilisateurs.

Diversification matérielle

La performance d'un SGBD dépend des supports de stockage qui déterminent le temps d'accès aux données et des dispositifs de traitement qui influencent le temps d'exécution des opérations élémentaires. Durant les dernières décennies, la puissance de ces derniers ne cesse de croître d'une façon exponentielle selon la fameuse loi de *Moore*. Les facteurs suivants ont ainsi évolué d'un facteur de 10 chaque décennie : le nombre d'instructions machine s'exécutant par

seconde, le coût d'exécution du processeur, les débits en lecture/écriture des mémoires, le taux de stockage en mémoire secondaire pour une unité de coût et le taux de stockage en mémoire principale par unité de coût [12].

Malgré les différents types de supports numériques (HDD, SSD, mémoires flash, mémoires cache, serveur RAID/NAS) et processeurs dont le rapport coût/performance ne cesse de s'améliorer chaque année, des ressources illimitées ne peuvent pas exister dans la pratique. Les utilisateurs étant de plus en plus exigeants, les concepteurs continuent de travailler la partie logicielle pour optimiser l'exploitation de ces prouesses technologiques par le SGBD.

Variation des paradigmes et architecture des BD

Pendant les années 1990, des travaux de recherche se sont intéressées à l'accès en parallèle aux données décentralisées pour améliorer les performances surtout quand il s'agit de répondre à des requêtes complexes sur d'énormes volumes de données [79, 68]. Les travaux théoriques sur les BD distribuées ont conduit à la définition de plusieurs prototypes. Par exemple, un projet a été initié afin d'explorer la capacité des disques à traiter les données en parallèle et a mené au développement d'une machine de BD parallèle appelé *Gamma*. Les systèmes parallèles d'*IBM*, *Tandem*, *Oracle*, *Informix*, *Sybase*, et *AT&T* sont basées sur les recherches du projet *Gamma* [79]. Aujourd'hui, la majorité des systèmes de BD offrent la possibilité de distribuer et de répliquer les données entre les nœuds d'un réseau informatique [79]. La phase de déploiement doit prendre en compte diverses architectures matérielles (distribuées, parallèles, cluster, grille de calcul, cloud, etc.) issues des évolutions des systèmes réseaux, des dispositifs de stockage et de traitement et de la technologie des BD d'une manière générale.

Comme autre variation d'architecture, plusieurs systèmes modernes utilisent l'architecture client-serveur, dans laquelle les requêtes établies par les clients sont transmises pour être traitées au niveau du serveur. Ces systèmes se sont largement répandus, pour permettre de diviser les tâches et de décentraliser l'information [68].

Une autre variation de la conception physique, nous notons que l'émergence des BDBO a fait apparaître de nouveaux modèles physiques de données et de nouvelles architectures. Contrairement aux BD classiques, où le modèle logique est stocké selon une approche relationnelle, dans une BDBO différents modèles de stockage physique (représentation horizontale, spécifique, etc.) sont utilisés pour stocker les deux niveaux de modélisation : le niveau ontologie et le niveau des instances ontologiques. L'architecture des systèmes gérant les bases de données a aussi évolué : architecture mono-couche où les données sont manipulées au niveau de la mémoire centrale (main memory databases), architecture à deux couches représentant la méta-base et le contenude la base (utilisée par la majorité des systèmes), architecture à trois couches et architecture à 4 couches.

Multiplication des structures d'optimisation

Les BD assurent la disponibilité, l'intégrité mais également la performance d'accès aux données. Cette dernière notion est étroitement liée à l'optimisation des requêtes, qui à son tour

dépend de plusieurs paramètres essentiellement physiques pour les premières générations des BD. La croissance du volume des données, la sophistication des requêtes et du schéma, en continu, ont évidemment accru la complexité du problème d'optimisation et ce malgré la vitesse accrue des supports de stockage et l'augmentation de puissance des dispositifs de traitement. En conséquence, de nouveaux paramètres logiques et conceptuels sont désormais impliqués dans le problème d'optimisation, ce qui a donné naissance à plusieurs structures d'optimisation, dont les plus importantes initialement étaient les index et les vues matérialisées [84].

Comme cette section le montre, chaque étape du cycle de conception des BD a des variantes. De nouvelles étapes ont aussi été ajoutées à ce cycle, comme expliqué dans la section suivante.

4.3 Évolution verticale : des bases de données aux entrepôts de données

Contrairement à l'évolution horizontale, celle qualifiée de verticale augmente le cycle *mature* de conception des BD en ajoutant de nouvelles *sous-phases* (voir figure 1.13) afin de répondre aux nouvelles exigences. Nous illustrons ce type d'évolution par l'une des innovations majeures : les entrepôts de données.

Pendant les années 1980, les entreprises et les organisations ont eu à gérer des mégaoctets voire des giga-octets de données. Dans les années 1990, ce volume de données a augmenté pour atteindre les téraoctets et pétaoctets [65]. Ces données étaient stockées dans des BD disparates présentant des *ilots d'informations*. L'augmentation en continu de ce volume a créé un nouveau besoin qui est celui d'*analyser* ces quantités de données afin d'en extraire de nouvelles connaissances. Par exemple, la technique connue des ventes croisées est une technique de marketing permettant de proposer des produits ou des offres complémentaires à un achat ou une consultation d'un produit.

4.3.1 Traitement transactionnel et analytique des données

Le besoin d'analyse a fait émerger deux principaux types de BD, auxquels n'importe quelle application peut s'identifier. (i) Les transactions (par exemple, gestion des ventes) qui reflètent les opérations quotidiennes de l'entreprise dont les données doivent être immédiatement enregistrées avec précision dans le but d'être consultées. La BD qui est principalement conçue pour ce genre d'opérations quotidiennes est appelée une BD transactionnelle (opérationnelle), alias OLTP (*OnLine Transaction Processing*) pour traitement transactionnel en ligne. (ii) À l'opposé, les BD analytiques s'intéressent aux historiques de données et aux métriques utilisées pour l'extraction des décisions stratégiques. Ces décisions doivent avoir une vue unifiée et intégrée des données issues de sources (BD) hétérogènes qui se trouvent au sein de la même organisation ou bien dans différentes organisations. [65]. Un *système d'intégration* (virtuel ou matérialisé) est la solution permettant de garantir cette intégration. Pour assurer une séparation entre les données spéculatives et celles représentant l'état actuel de l'organisation, la solution choisie a été

de dupliquer les données dans une nouvelle base de type *entrepôt de données* (ED). Le concept d'ED a rapidement trouvé écho dans divers domaines et applications, et les premiers ED ont été développés au sein de projets industriels. La technologie des ED est actuellement déployée avec succès dans diverses industries : production, vente au détail, services financiers, transports, télécommunications, médecine, etc. Des outils d'analyse sont utilisés en complément aux ED afin de fournir diverses techniques d'analyse, de visualisation et de consolidation des données de l'ED. Au début des années 1990, le concept de *traitement d'analyse en ligne* (OLAP) a été proposé par *Edgar Codd* en 1993 [52] pour décrire les fonctionnalités de l'outil *Essbase* de la société Arbor Software rachetée plus tard par Oracle, outil qui offre différentes fonctionnalités d'analyse pour les BD multidimensionnelles. Plusieurs autres outils OLAP ont suivis. Un ensemble de règles ont aussi été spécifiées définissant les contraintes que doivent respecter les ED pour supporter une analyse OLAP [54]. La plupart des nouveaux produits orientés *entrepôt de données* ont par la suite été appelés outils de l'informatique décisionnelle (business intelligence). Nous définissons dans ce qui suit les principales notions relatives aux ED et nous décrivons leur cycle de conception.

4.3.2 Modélisation multidimensionnelle

Les applications d'aide à la décision à base d'ED utilisent des processus d'analyse en ligne de données OLAP répondant à des besoins d'analyse de l'information. Pour ce type d'environnement OLAP, une nouvelle approche de modélisation a été proposée : la *modélisation multidimensionnelle*. Popularisée par *Ralph Kimball* dans les années 1990, cette modélisation est aujourd'hui reconnue comme la modélisation la plus appropriée aux besoins d'analyse et de prise de décision [4]. La modélisation multidimensionnelle est une technique de conceptualisation et de visualisation des modèles de données. Elle offre une structuration et une organisation des données facilitant leur analyse. Elle a pour principal objectif d'avoir une vision multidimensionnelle des données. Les données sont ainsi organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives d'analyse. Cette modélisation est intéressante d'une part, parce qu'elle représente une modélisation assez proche de la manière de penser des analystes et d'autre part, car elle facilite la compréhension des données par les utilisateurs. Un modèle multidimensionnel est basé sur les deux concepts fondamentaux de *fait* et de *dimension*.

— Un *fait* représente le sujet ou le thème analysé. Il présente un centre d'intérêt de l'entreprise et est considéré comme un concept clé sur lequel repose le processus de prise de décision. Un fait est formé de *mesures* ou attributs du fait (atomiques ou dérivés) qui correspondent aux informations liées au thème analysé. Par exemple, pour la gestion des commandes, les mesures peuvent être la quantité du produit commandé et le montant de la commande.

— Une *dimension* représente un contexte d'analyse d'un fait. Par exemple la gestion des commandes peut être analysée selon les dimensions *Client*, *Magasin*, et *Temps*. Les dimensions se présentent sous forme d'une liste d'éléments organisés de façon *hiérar-*

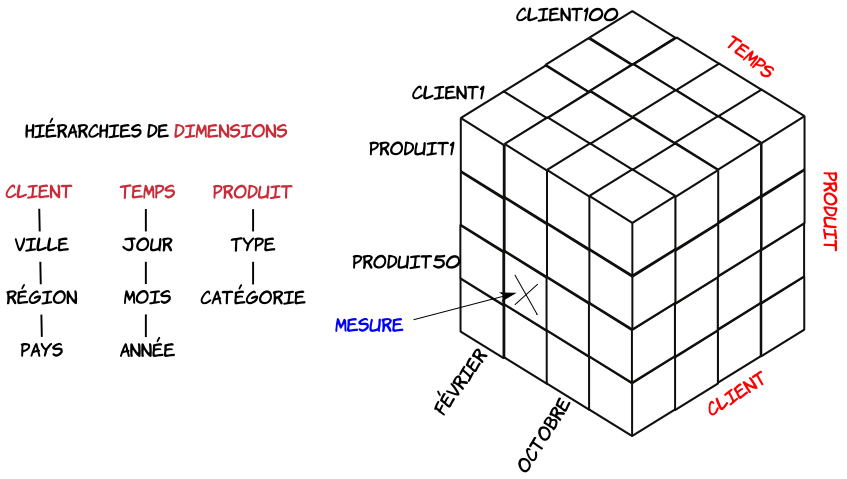


FIGURE 1.10 – Exemple d'un cube multidimensionnel des *Vente*s

chique. Par exemple, pour la dimension temps, nous pouvons avoir la hiérarchie suivante : année, semestre, trimestre, mois, semaine et jour. Les dimensions sont caractérisées par des attributs de dimensions. Une hiérarchie est dite complète si tous les objets d'un niveau de la hiérarchie appartiennent à une seule classe d'objets d'un niveau supérieur et forment cette classe.

Le constructeur des modèles multidimensionnels est appelé un *cube* de données (voir figure 1.10). Un cube de données est composé de cellules qui représentent les mesures (les attributs du fait). Le cube ci-dessous permet d'analyser les mesures selon les différentes dimensions : produit, site et temps. Les hiérarchies définies sur une dimension peuvent être simples ou multiples.

Plusieurs opérations OLAP peuvent être effectuées sur cette structure multidimensionnelle appelées opérations de restructuration qui sont : *pivot* (rotation du cube autour d'un des axes), *switch* (interchanger la position des membres d'une dimension), *split* (couper le cube en réduisant le nombre de dimensions), *nest* (imbriquer des membres de dimensions), et *push* (combiner les membres d'une dimension aux mesures du cube). D'autres opérations sont liées à la granularité permettant ainsi la hiérarchisation des données. Ces opérations sont : *roll up* qui permet de visualiser les données de manière résumée en allant d'un niveau particulier de la hiérarchie vers un niveau plus général et l'opération *drill down* qui permet de naviguer vers des données d'un niveau inférieur et donc plus détaillé.

Plusieurs auteurs ont proposé des modèles de données adaptés au paradigme multidimensionnel pour le développement d'un ED (conceptuel [159, 42], logique [39]). Une classification des modèles de données multidimensionnels proposés ainsi que les différents concepts utilisés est effectuée par *Abello et al.* [2]. Cette classification couvre les différentes phases de conception.

4.3.3 Cycle de conception de BD revisité

Partant du principe que les ED sont des BD orientées analyse, ayant une charge de requêtes et un volume de données particuliers [89], nous allons détailler les apports de cette technologie au cycle de conception des BD : (i) l'ajout d'une phase supplémentaire qui traduit l'aspect d'intégration, ainsi que (ii) les différentes adaptations *horizontales*, à commencer par la modélisation multidimensionnelle tout au long du cycle.

Évolution verticale apportée par les ED

La phase d'extraction-transformation-chargement (ETL) a été ajoutée par les ED au cycle de conception des BD. Elle consiste à extraire les données des sources, puis à les transformer afin de peupler le schéma cible de l'ED. Cette phase vient après la *conception logique* où l'on définit (i) un schéma logique de l'ED selon une plate-forme cible intégrant l'ensemble des schémas des sources de données sélectionnées, mais aussi (ii) les mappings entre ces derniers et le schéma cible. Il existe actuellement une panoplie d'outils ETL qui ont été proposés par la communauté industrielle, comme *Microsoft Data Transformation Services* (DTS), *Talend open studio*, *Oracle Warehouse Builder* (OWB), *IBM Data Warehouse Center*. D'autres outils ETL académiques ont été proposés comme le système *Potter's Wheel* [145], le système *AJAX* [70] ou l'outil *HumMer* [131]. À la fin de cette phase, vient l'étape de la conception physique où le schéma de l'ED est implémenté selon un SGBD cible. Cependant, la première génération des projets, prôné par *W.H. Inmon*, s'est focalisée sur ces dernières étapes et ne considérait pas la modélisation conceptuelle comme étape à part entière.

Plusieurs études ont montré par la suite la nécessité d'inclure une étape de modélisation conceptuelle offrant une vue abstraite du domaine étudié, et facilitant la communication entre les concepteurs et les utilisateurs de l'ED [76]. *Ralph Kimball* [104], un acteur important du domaine des ED, a d'abord fourni une vision de conception de l'ED à partir des besoins des utilisateurs. Sa proposition a été suivie par plusieurs autres études proposant des méthodes de conception montrant que la prise en compte des besoins détermine le succès ou l'échec du projet d'entreposage [148]. La phase de modélisation conceptuelle a ensuite été reconnue comme phase indispensable [77].

Évolution horizontale apportée par les ED à la phase logique

L'évolution verticale illustrée par l'ajout de la phase d'intégration influence horizontalement les autres étapes. En effet, trois principales représentations logiques orientées ED ont été proposées : ROLAP, MOLAP ou HOLAP. L'approche MOLAP (Multidimensional On-Line Analytical Processing) implémente le cube sous forme d'un tableau multidimensionnel, qui sera ensuite implémenté dans un SGBD multidimensionnel. Le principal avantage de l'approche MOLAP est sa performance en temps d'accès. Par ailleurs, l'approche ROLAP (Relational On-Line Analytical Processing) représente le modèle de l'ED selon une représentation relationnelle, qui sera ensuite implémentée en utilisant un SGBD relationnel. Chaque dimension du cube est représentée sous forme d'une table appelée table de dimension. Chaque fait est représenté par une

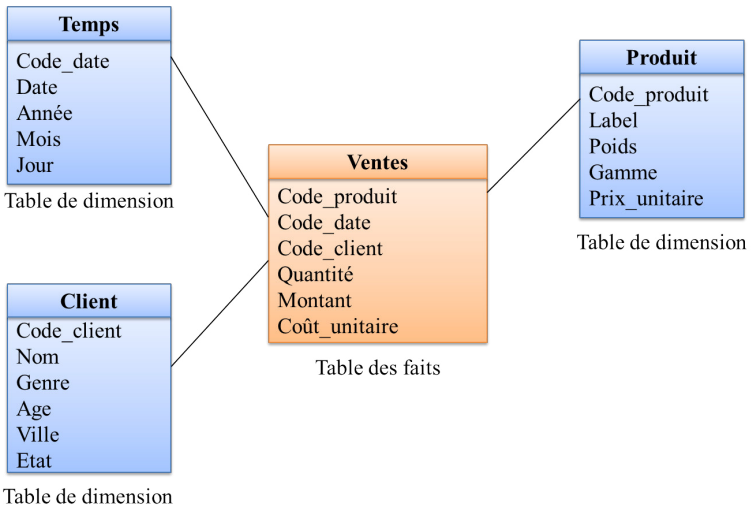


FIGURE 1.11 – Exemple d'une modélisation d'ED en étoile [12]

table de faits. Les mesures sont stockées dans les tables de faits qui contiennent les valeurs des mesures et les clés vers les tables de dimensions comme illustré par la figure 1.11.

L'inconvénient majeur des systèmes ROLAP est le temps de réponse qui peut être élevé à cause des jointures entre la table des faits souvent très large [105]. Quant à leurs avantages, nous pouvant citer le fait qu'ils sont basés sur le modèle relationnel (un standard bien maîtrisé) et leur capacité à stocker de larges quantités de données. Il existe trois types de schémas pour la modélisation des systèmes ROLAP : le schéma en étoile, le schéma en flocon de neige et le schéma en constellation.

Le schéma en étoile : les mesures sont regroupées dans une table de faits et chaque dimension du cube est représentée par une table de dimension. La table de faits est normalisée et référence les tables de dimension grâce aux clés étrangères (figure 1.11). Les tables de dimension sont généralement dénormalisées.

Le schéma en flocon de neige : dans un schéma en étoile, la table de dimension contient les informations relatives à ses hiérarchies, malgré la différence des propriétés de leur niveaux (figure 1.12). Le schéma en flocon vient décomposer une ou plusieurs tables de dimensions en plusieurs niveaux formant une hiérarchie sans toucher à la table de faits. Cette décomposition peut être considérée comme une normalisation des tables de dimensions. Ce type de schéma offre une meilleure visualisation et compréhension des données, mais peut diminuer les performances de l'ED à cause de nouvelles jointures entre la (les) table(s) de dimension et ses (leur) sous-dimensions.

Le schéma en constellation : les schémas en constellation font intervenir plusieurs tables de faits qui partagent des tables de dimensions.

L'approche HOLAP (Hybrid On-Line Analytical Processing) quant à elle, est la combinaison des approches ROLAP et MOLAP visant à bénéficier des avantages de chacune.

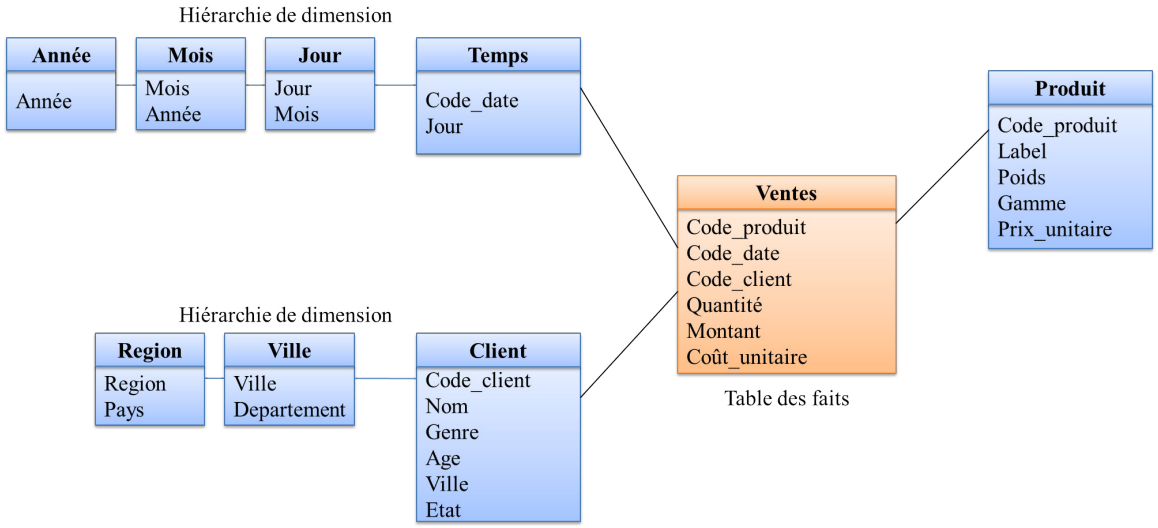


FIGURE 1.12 – Exemple d'une modélisation d'ED en flocon de neige [12]

Évolution horizontale apportée par les ED à la phase physique

Dans son article *Self-Tuning Database Systems: A Decade of Progress* (prix de 10 Year Best Paper Award à la prestigieuse conférence Very Large Databases, édition 2007), Surajit Chaudhuri indique : "*The first generation of relational execution engines were relatively simple, targeted at OLTP, making index selection less of a problem. The importance of physical design was amplified as query optimizers became sophisticated to cope with complex decision support queries*" [44]. Cette amplification peut s'expliquer par : (i) le fait que les projets d'ED sont issus originellement de l'industrie qui s'intéresse aux aspects pratiques de performance, (ii) la complexité de leur schéma, (iii) leur volume de données. En conséquence, une panoplie de structures d'optimisation a été proposée qui peut se diviser en deux grandes catégories [96]: (i) des structures redondantes qui engendrent un coût de stockage ou de maintenance supplémentaire. Comme exemples, nous citons les index, les vues matérialisées [84], la fragmentation verticale [69], la gestion du buffer [185] et (ii) des structures non redondantes qui, contrairement à la première classe, n'engendrent aucun coût de stockage en sus, du moment où ils agissent uniquement sur la répartition des données/requêtes. Comme par exemple, la fragmentation horizontale [20], le traitement parallèle [126] et l'ordonnancement des requêtes [10]. Certaines techniques sont utilisées pendant la création de la BD (fragmentation horizontale) et d'autres pendant son exploitation (vues matérialisées). Chacune d'elles peut fonctionner de façon isolée, ou en combinaison avec une autre. Cependant, la diversité des techniques et des possibilités de combinaisons rendent la tâche de sélection difficile [120].

Le stade de maturité est caractérisé par un cycle standard de conception de BD, qui pouvait contenir et représenter les évolutions de cette époque. Cependant, de plus récentes évolutions dérogent à ce constat, comme nous allons le voir dans la section suivante.

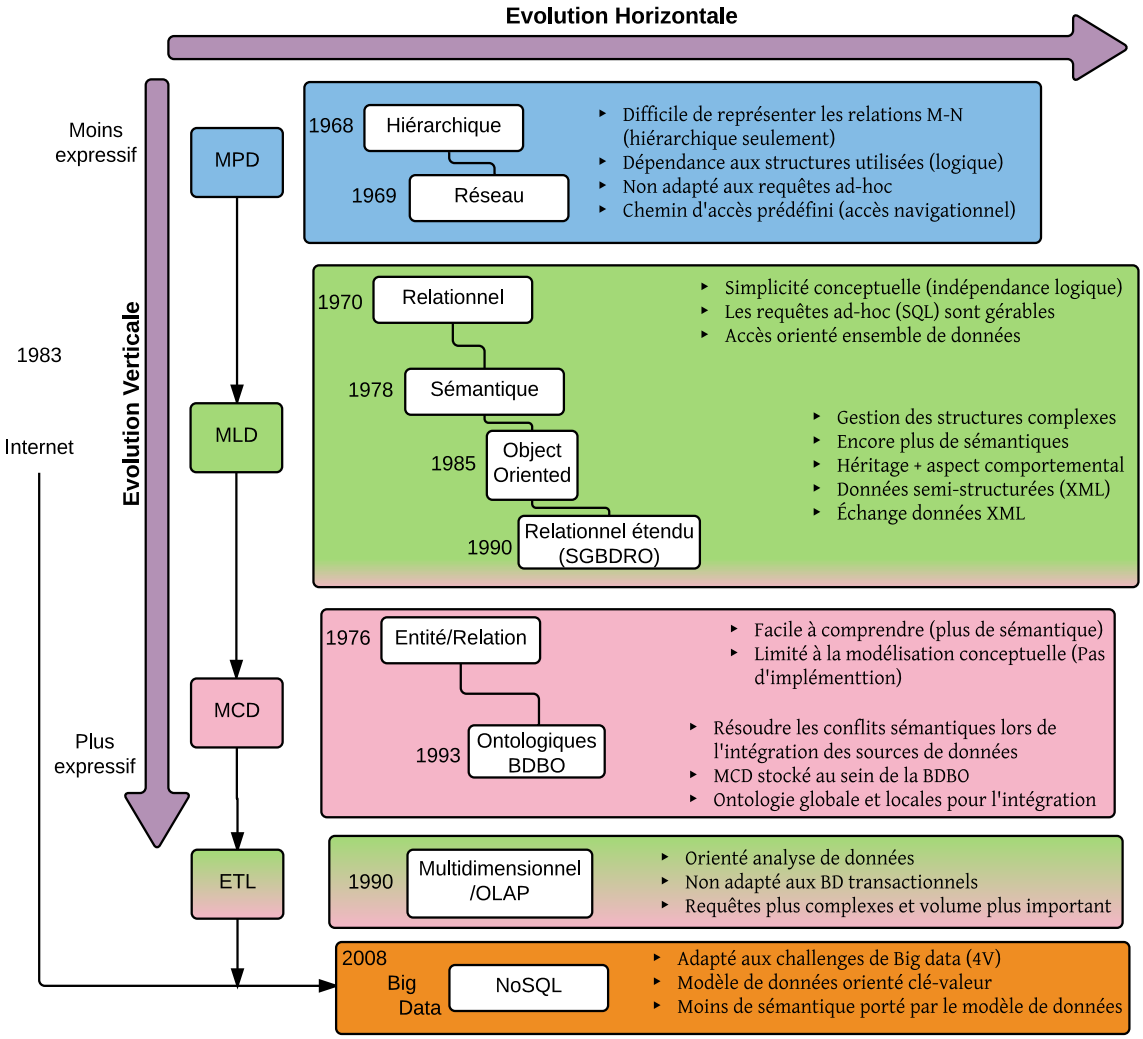


FIGURE 1.13 – Bilan des évolutions verticales/horizontales du cycle de conception des BD

5 Stade dynamique : instabilité du cycle

Comme nous le montrons dans cette section, plusieurs innovations récentes ne se conforment pas au cycle mature décrit précédemment. Aussi, le cycle mature est devenu instable et est entré dans ce que nous qualifions de "*stade dynamique*".

5.1 Les principales innovations

Depuis 1989, un groupe de chercheurs éminents se réunit périodiquement pour discuter de l'état des recherches des bases de données et les principaux challenges et tendances du domaine. Lors de la huitième édition (2008) [6], il a été déclaré que la technologie des BD était à la veille d'une nouvelle révolution liée au nouveau contexte marqué par : l'explosion des données non structurées, l'importance croissante de l'informatique décisionnelle et l'émergence des technologies *Cloud*. Ce contexte peut ainsi exiger un nouveau modèle de BD qui va éventuellement détrôner le relationnel -toujours pionnier- et il peut s'agir du *NoSQL*. Ce dernier est mieux adapté aux données non structurées des cyberespaces (e.g. *Twitter*, *Facebook*), qui ne demandent pas forcément une grande consistance et intégrité, contrairement aux données relationnelles. D'un autre côté, vu la montée en flèche du volume des transactions de données, plusieurs entreprises "*Big Data*" se sont tournées vers ce nouveau modèle. D'autres tendances se sont dégagées, notamment les BD "*en mémoire*" qui exploitent la puissance du traitement parallèle et des processeurs multi-coeurs pour accélérer les réponses décisionnelles et le *Cloud computing* qui offre un accès à des clusters éparpillés à des prix très raisonnables sans la moindre connaissance en architectures des serveurs. Il reste à voir si le modèle relationnel s'adaptera une fois de plus à ces nouveaux défis, ou si il cèdera sa place de leader au NoSQL. Il existe 4 principaux types de BD NoSQL spécifiques à différents besoins.

— Les BD clé-valeur : les données sont stockées dans une table de hachage, sous forme de couples clé-valeur. La valeur peut être de n'importe quel type, simple ou complexe.

— Les BD orientées document : sont une extension des bases orientées clé-valeur où la valeur est un document de structure libre, tel que XML ou JSON. Cette architecture lui donne une puissance dans le traitement des requêtes. Les implémentations les plus populaires sont *CouchDB* d'Apache, *RavenDB*, *Hadoop* qui utilisée par des grands noms de l'informatique tels *Yahoo* ou *Microsoft*, *MongoDB* [57].

— Les BD orientées colonnes : sont semblables aux BD relationnelles d'un point de vue architectural, à la seule différence que les BD NoSQL sont orientées colonnes et possèdent un schéma qui est dynamique. Le nombre de colonnes est variable et peut changer entre deux entrées de la table, ce qui évite d'avoir des champs de colonnes avec la valeur *NULL*. Ces BD sont capables de stocker des volumes très importants de données.

— Les BD orientées graphe : Ce modèle de représentation des données se base sur la théorie des graphes. Il permet donc de stocker les données sous forme de graphes. Il permet

aussi la modélisation, le stockage et la manipulation de données complexes liées par des relations non-triviales ou variables et convient particulièrement au stockage des données des réseaux sociaux.

5.2 La diversité des besoins non fonctionnels

Un autre changement qui est venu déstabiliser le cycle standard, est la diversité des besoins non fonctionnels. En effet, la performance (l'efficacité) des BD se mesurait essentiellement en matière de temps d'exécution des requêtes. Face aux différentes évolutions précédemment évoquées et les nouvelles exigences du marché, d'autres mesures se sont imposées, à l'instar de l'énergie et de la sécurité. De ce fait, non seulement certaines phases de conception doivent s'adapter, comme l'optimisation logique et physique [71], mais aussi un autre aspect vient rendre le processus encore plus compliqué, celui de l'interaction entre ces mesures et la nature multi-objectifs du problème. Ces mesures sont plus communément connues sous le nom de *besoins non fonctionnels* (BnF). Ainsi, la conception de BD s'apparente davantage à un problème de satisfaction des besoins non fonctionnels.

Définitions

Un logiciel ne peut être décrit uniquement à travers ses fonctions, ni à travers ses seules caractéristiques techniques. Le cahier des charges doit nécessairement comporter des caractéristiques relatives à la qualité du logiciel, appelées BnF. Comme les exigences fonctionnelles, celles-ci doivent être structurées selon leur typologie. Plusieurs typologies existent pour décrire la qualité du logiciel. Entre autres, il existe la norme *ISO/CEI 25000* qui a l'avantage de décrire la qualité du logiciel de la manière la plus complète et la moins redondante possible [56].

En ce qui concerne les aspects "non fonctionnels" (au sens courant du terme), un vocabulaire commun existe. Il est précisément et formellement défini par une norme internationale : la norme *ISO/CEI 9126* sur l'évaluation de la qualité du logiciel. La qualité est définie à l'aide de six caractéristiques se décomposant en une vingtaine de sous-caractéristiques (voir figure 1.14). Dans le contexte des BD [78], ces caractéristiques peuvent être résumées aux suivantes.

— *Fonctionnalité* : ou capacité fonctionnelle, est l'ensemble d'attributs portant sur l'existence d'un ensemble de fonctions et leurs propriétés données.
— *Utilisabilité* : évalue le système en question, à travers l'interaction entre ce dernier et des utilisateurs, pour vérifier la facilité d'utilisation et sa compréhensibilité.
— *Performance* : vérifie l'efficacité du système sous une charge donnée.
— *Résistance* : montre comment le système opère en présence de tailles maximales de données et des conditions restreintes.
— *Récupération* : vérifie la capacité du système à se rétablir d'incidents matériels et logiciels.

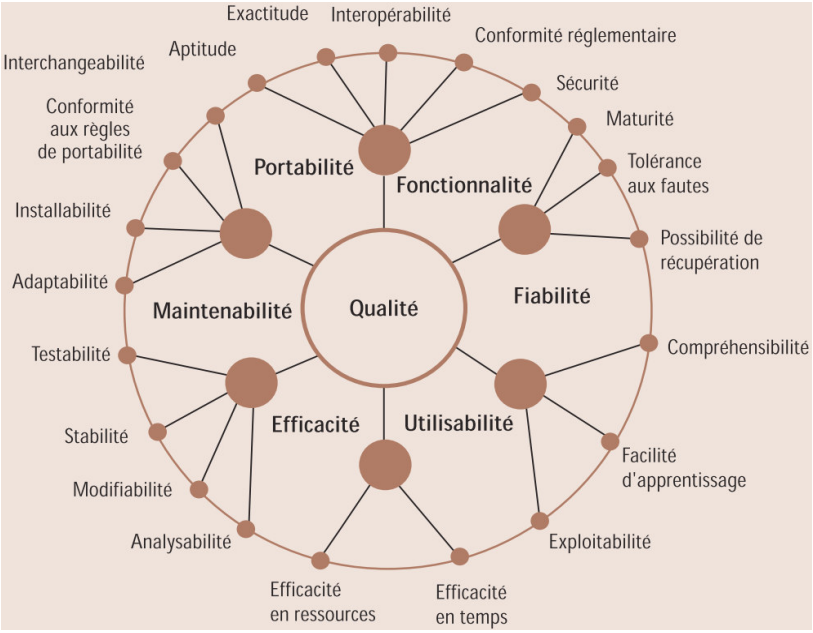


FIGURE 1.14 – La décomposition de la qualité du logiciel en caractéristiques et sous-caractéristiques (norme *ISO/CEI 9126*) [55]

— *Sécurité* : vérifie que le système protège les données tout en offrant les fonctionnalités requises.

— *Maintenabilité* : vérifie que le système peut être facilement modifié pour corriger les éventuelles erreurs, améliorer la performance, ou adapter le système à un environnement instable.

6 Conclusion

Ce chapitre se voulait être un recul historique de la conception des *BD*, l'une des étapes clés du cycle de développement des *BD* et l'un des sujets clés de notre travail. Nous avons présenté les principaux jalons qui ont ponctué l'évolution de ce processus, à savoir : le stade de maturation, le stade de maturité et le stade dynamique. Nous y avons démontré l'évolution et la complexité croissantes de ce processus, la diversité omniprésente et ses conséquences sur le comportement des concepteurs et la vision de la conception. Afin de faire face à cette diversité et ces évolutions, il devient nécessaire de :

— assister les concepteurs en :

  — leur donnant la possibilité de considérer les différents choix ;

  — les aidant à faire le bon choix ;

— automatiser, autant que possible, les tâches du cycle de conception dans son ensemble ;

— développer un framework général, qui implémente le processus de conception et qui, de plus, réduit le phénomène éventuel de réinvention de la roue.

Ces différents points correspondent au processus de la gestion de la variabilité issu du génie logiciel. En effet, qui dit diversité, dit nécessairement un besoin de faire adapter les systèmes à un contexte particulier, ce que l'on appelle la *variabilité* dans le jargon du génie logiciel. Dans le chapitre 2 suivant, nous présentons un état de l'art sur la gestion de cette variabilité (diversité) qui vise à réduire la complexité du processus de conception des BD.

Chapitre 2

Gestion de la variabilité de la conception des bases de données

Sommaire

1 Introduction 49

2 Travaux orientés vers l'aide à la décision 50

2.1 Test de BD 51

2.2 Bancs d'essais 54

2.3 *Advisors* : outils d'aide à l'administration de BD 57

2.4 Optimisation à base de modèles de coûts 60

3 Travaux orientés vers la réutilisation 62

3.1 Automatisation : les outils CASE 63

3.2 Standardisation : les design patterns 66

4 Travaux orientés vers la gestion explicite de la variabilité 69

4.1 Notions sur la gestion de la variabilité en génie logiciel 70

4.2 Travaux existants portant sur la phase physique 73

4.3 Travaux existants portant sur la phase conceptuelle 76

5 Bilan et discussion 77

1 Introduction

Dans le chapitre précédent, nous avons montré l'omniprésence de la diversité, l'abondance des évolutions et l'augmentation de la complexité du cycle de conception des BD. La diversité implique le besoin de faire adapter les systèmes à un contexte particulier, ce qui est désigné par la *variabilité* dans le vocabulaire du génie logiciel. Ce chapitre vient donc répondre à un double objectif. Le premier consiste à présenter un état des lieux des principales solutions visant à gérer la variabilité croissante dans le cycle de conception des BD. Notons que le terme de la *gestion de la variabilité* doit être vu dans une acception large, couvrant non seulement sa définition originale relevant du domaine du génie logiciel, mais aussi les différents outils, techniques et concepts qui visent à *maîtriser la diversité ainsi que les évolutions et donc réduire la complexité* du processus de conception des BD. Le deuxième objectif consiste à présenter les concepts fondamentaux liés à la gestion de la variabilité.

À ce titre, rappelons le contexte ayant amené au besoin d'une gestion de la variabilité dans la conception des BD, et ainsi aux travaux présentés dans ce mémoire.

- Face à l'abondance des solutions dont dispose actuellement les concepteurs de BD, le besoin d'outils d'aide à la décision s'accentue. Les apports des méthodes d'aide à la décision peuvent se résumer, d'une manière générale, en deux points :
  - amener les concepteurs à considérer les différents choix disponibles ;
  - les aider à choisir la solution la plus adaptée en fonction des critères établies. On parle souvent de besoins non fonctionnels (BnF) dans le contexte des BD.
- La complexité croissante du processus de conception de BD a motivé la réutilisation et l'automatisation de ce processus, en ne développant plus chaque composante séparément, mais plutôt en le concevant à partir d'éléments réutilisables, et de tâches automatisables. L'objectif principal étant de réduire le temps et les coûts, voire même de garantir la qualité de ce processus ainsi que tout les avantages d'une stratégie de réutilisation.
- Face à l'abondance des solutions de conception de BD ainsi que la complexité croissante du processus, plusieurs défis viennent entraver l'efficacité de ce dernier : distinguer ce qui est nouveau comme concept/méthode de ce qui existe déjà, bien exploiter les solutions existantes, avoir une même terminologie et surtout des lignes directrices en matière de conception. Il devient donc important d'agir à un niveau d'abstraction supérieur, celui des méthodologies fédératrices offrant : un cadre de travail unifié, des lignes directrices de conception, un entreposage de solutions, une simplicité de réutilisation, une possibilité d'automatiser et de paramétrer les composantes (configurabilité/adaptabilité), une aide à la décision, ainsi qu'une méthode de mise en oeuvre limitant considérablement les efforts - et donc les coûts - de maintenance des composantes fonctionnelles automatisées. Ces méthodologies incarnent la "gestion explicite de variabilité" et sont très prisées dans l'ingénierie logicielle. Leur adaptation au contexte de BD s'inscrit dans la vision de "BD comme produit logiciel".

Besoins	Approches	Axes de gestion de variabilité
Choisir parmi la panoplie	Advisors, Benchmark	Aide à la décision
Valider le choix	Tests	
Éviter de réinventer la roue + Unification des concepts	Design Patterns	Réutilisation (Standardisation)
Réduire les coûts et les temps	Outils CASE	Réutilisation (Automatisation)
Avoir une méthodologie	Approches de gestion de variabilité (p. ex. LdP)	Aide à la décision + Réutilisation + Configurabilité

TABLE 2.1 – Quelques approches de gestion de variabilité de la conception de BD (par axe)

Ainsi, et comme récapitulé dans le tableau 2.1, la revue de littérature qui suit porte sur la gestion de la variabilité dans la conception des BD, et s'articule en trois temps. Nous allons commencer par présenter dans la section 2 les outils et les méthodes s'inscrivant dans l'axe d'aide à la décision. Leur objectif étant d'assister et d'aider les concepteurs à choisir et/ou valider une solution. Dans la section 3, nous abordons les principales contributions visant à réduire la complexité du processus de conception en réutilisant certaines tâches (composantes) : c'est le deuxième axe de la gestion de la variabilité. La section 3 sera consacrée aux travaux appliquant des méthodologies du génie logiciel pour la gestion *explicite* de la variabilité de la conception des BD. Ils s'inscrivent dans le troisième axe, où l'on assimile les BD aux produits logiciels. Nous mettons l'accent sur les limites théoriques et pratiques de chaque proposition, afin de conclure dans la dernière section par un positionnement de nos travaux et une description des objectifs de notre proposition que nous développons dans la suite de ce mémoire de thèse.

2 Travaux orientés vers l'aide à la décision

La conception des BD, comme beaucoup d'autres systèmes d'ailleurs, est souvent assimilée à de l'art car les concepteurs se basent essentiellement sur leur intuition et leur expérience quant à la sélection d'un choix de conception. Ce principe a pour avantage la facilité. En revanche, se reposer totalement sur son ressenti présente des limites, d'autant plus que la conception des BD résulte d'un processus toujours de plus en plus complexe : le nombre de solutions et de dépendances à prendre en compte est toujours plus grand et les enjeux fonctionnels ou non (par exemple, économiques, écologiques ou de confidentialité) si importants que le processus de conception exige des outils stratégiques, surtout que de nombreux domaines proposent des appuis méthodologiques dans ce sens. Par stratégie, on entend un processus de conduite de la décision en vue d'atteindre un but précis, ce qui a été révolutionné par l'informatique en donnant lieu au domaine de l' *aide à la décision*. Il réunit des méthodes et des outils permettant de fournir des informations/alternatives utiles éventuellement hiérarchisées, et surtout de choisir parmi plusieurs alternatives, en fonction de critères préalablement établis.

Dans le contexte des BD, le terme d'aide à la décision fait penser directement à l'informatique décisionnelle qui fait partie de ce que les anglophones appellent "*business intelligence*", et qui désigne les moyens, les outils et les méthodes qui permettent de collecter et de modéliser des données d'une entreprise (entreposage des données), puis d'en dégager des informations utiles (exploration des données) en vue d'offrir une aide à la décision et une vue d'ensemble de l'activité traitée. Ces méthodes agissent au niveau des données et les décisions qui en découlent portent essentiellement sur l'activité de l'entreprise et éventuellement (par transitivité) sur des choix de conception. Elles interviennent donc après l'opérationnalisation de la BD. Or, nous nous intéressons aux choix et décisions relevant de la conception des BD, et donc aux méthodes agissant à ce niveau là. Plusieurs propositions ont été formulées dans cette optique qui sont décrites dans la section suivante.

2.1 Test de BD

Le test représente un aspect clé du processus de développement de tout produit logiciel qui consiste à détecter des erreurs. Il a bénéficié d'une grande attention de différentes communautés de recherche dont celle des BD. Nous abordons dans cette section le test dans le contexte de la conception de ces dernières. En effet, même si la grande partie s'opère principalement après le déploiement (voir section 1.2), ces tests interviennent, d'une manière itérative (méthode agile), tout le long du cycle de conception. La correction d'un schéma de BD est en grande partie déterminée au début de son cycle de développement, en l'occurrence lors de la spécification des exigences et de la conception. Les erreurs introduites à ces stades sont généralement beaucoup plus coûteuses à corriger que celles du post-déploiement. Par conséquent, il est souhaitable de détecter et corriger les erreurs le plus tôt possible dans le processus de conception voire du développement. Il est important de noter que le test des phases d'intégration (ETL) et physique, portent souvent sur les données, et ont donc besoin d'une BD qui reflète la BD réelle (le même facteur d'échelle et nombre de partitions par exemple) comme les mocks [15]. Nous adoptons ici la définition concise du test proposée par *Myers* : "*tester un programme est essayer de le faire échouer*", ce qui résume son objectif à "*déceler les défauts en déclenchant des échecs*" [129]. Cela consiste souvent à appeler les opérations que composent les logiciels à tester, avec des paramètres d'entrée et de contexte appropriés, et vérifier par la suite qu'ils retournent les résultats escomptés.

Ainsi, nous avons inclus le test comme stratégie d'aide à la décision, pour la bonne et simple raison qu'effectivement il aide les concepteurs à faire des choix. En effet, en cas d'échec de test, le concepteur est ordonné à adapter ou modifier ces choix. Dans le cas contraire, le test lui permet de valider son choix à un certain degré.

15. Le "mock" d'une BD est un objet qui la simule, en renvoyant des données prédéterminées quelle que soit la requête qu'on lui transmet.

Phase / Objet	DC	CL	ETL	CP
Schéma	Utilisabilité Sécurité	Utilisabilité Maintenabilité		Maintenabilité
Opération	Fonctionnalité Maintenabilité	Performance Fonctionnalité	Fonctionnalité Maintenabilité	Performance
Données		Performance	Fonctionnalité Résistance Récupération	Fonctionnalité Résistance Récupération

TABLE 2.2 – Les dimensions *Quoi*, *Quand* et *Comment* du test du processus de conception des BD

2.1.1 Classification des travaux sur le test de BD

D'après la revue de littérature que nous avons menée, plusieurs approches et méthodes permettant de tester la conception des BD ont été proposées. Elles peuvent être classées selon plusieurs dimensions vu la complexité du processus. Le tableau 2.2 donne un aperçu d'une classification que nous avons proposée en se basant sur les travaux de [78, 184, 63] et qui est décrite dans ce qui suit.

Positionnement dans le cycle de conception. Les approches permettant de tester les BD se distinguent par rapport à leur position dans le cycle de conception des BD. En effet, ils peuvent avoir lieu pendant la phase du design conceptuel (DC), de conception logique (CL), d'intégration (ETL), ou de conception physique (CP). Notons que cette dimension correspond à l'axe *Quand* défini dans [78].

Type d'objet testé. De nos jours, le développement de la plupart des applications est modulaire et s'appuie sur une architecture logicielle 3-tiers qui se compose de trois niveaux : le niveau supérieur gère l'interfaçage entre le client et l'application, le niveau médian concerne la couche métier, quant au niveau inférieur, il est dédié à la gestion des données de l'application (voir figure 2.1).

Dans le contexte de la conception des BD, les deux couches supérieures peuvent être subdivisées en deux composantes selon le type d'objet utilisé : (i) schémas (modèles) provenant de la phase de conception (voir section 1.4) pour la couche médiane, et modèles des besoins, par exemple, pour la couche d'interfaçage, (ii) opérations sous forme de code source implémentant les différentes fonctionnalités métier/interface de cette phase.

Si nous tenons compte du plus bas niveau de granularité, le test pendant la conception des BD peut s'opérer sur des modèles, des opérations (code source) ou des données, à l'aide de

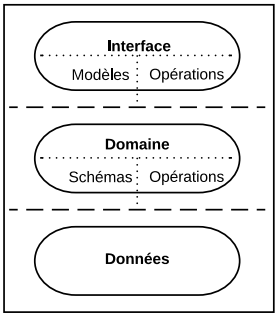


FIGURE 2.1 – Architecture 3-tiers typique des applications

métriques ou de simples comparaisons entre des entrées préétablies et des sorties prévues.

Critère de test. Les tests évaluent la qualité de l'objet par rapport à un critère précis. Ces critères sont communément appelés les besoins non fonctionnels (voir section 1.5.2). Notons que cette dimension correspond à l'axe *Comment* défini dans [78].

Stratégie de test. Nous nous contentons d'évoquer deux stratégies populaires parmi d'autres, et qui se distinguent par l'angle de vue utilisé. Il s'agit des tests en *boîte blanche* ou en *boîte noire*, et d'un autre point de vue, les tests de vérification et de validation.

- Le test en *boîte noire* est une approche orientée données, qui détecte les erreurs sans interagir avec la logique interne du programme. Elle est dirigée par des scénarios "entrées-sorties", visant à spécifier quelles sorties le programme (code ou modèle) est censé produire face à des entrées prédéterminées. À l'inverse, le testeur en *boîte blanche* doit agir sur la structure interne (conception/implémentation) de l'objet à tester. Si on prend le code source comme exemple d'objet de test, la vérification peut se faire en exécutant les instructions pas à pas (débogage).
- La vérification, quant à elle, comprend l'ensemble des activités visant à s'assurer que le logiciel implémente correctement une fonction spécifique (test fonctionnel). Ce genre de tests se réalise souvent en boîte blanche. La validation, de son côté, couvre l'ensemble des activités visant à s'assurer que le logiciel réponde aux exigences des clients. Cela se réalise souvent en boite noire, ce qui signifie que le testeur ignore la structure interne de l'objet à tester, et se contente de comparer des entrées préétablies avec des sorties escomptées.

Notons que cette dimension n'est pas représentée dans le tableau 2.2, car les deux stratégies peuvent être utilisées par chaque type de test représenté.

2.1.2 Limites des tests de BD

Les tests présentent plusieurs limites, parmi lesquelles nous trouvons les suivantes.

— D'une manière générale, le test atteste la présence des erreurs mais en aucun cas leur absence, ni éventuellement leur cause.

— Tester tous les scénarios possibles relève presque de l'impossible, surtout avec la complexité croissante des BD. En effet, la qualité du test dépend de la pertinence des scénarios de tests.

— Il n'offre pas de mécanismes de réutilisation ni de configurabilité des composantes de conception.

— Il valide des choix faits, mais il intervient a posteriori (après avoir fait des choix) et donc, ne tient pas en comptes des variantes possibles.

— Il porte sur une seule variante de solutions, et ignore les alternatives, validant ainsi un choix qui n'est pas nécessairement le meilleur.

Après avoir présenté les travaux existants et les limites de test comme technique d'aide à la décision dans la conception des BD, nous allons présenter une autre technique dans la section suivante.

2.2 Bancs d'essais

Parmi les bonnes pratiques souvent adoptées par les administrateurs des BD afin de choisir un SGBD, voire la meilleure combinaison matérielle-logicielle pour leur application, figure l'utilisation des bancs d'essais communément appelés par les anglophones les "*Benchmarks*". Le benchmarking consiste à évaluer, sous forme de métriques, plusieurs configurations de SGBD édités par les meilleurs compétiteurs/entreprises reconnus comme des leaders du marché. Les résultat sont utilisés comme référence qui sert à comparer les performances globales des SGBD, mais aussi pour illustrer les avantages d'un système par rapport à un autre dans une situation donnée, constituée d'une charge de requêtes d'un type particulier d'application. Il peut donc être utilisé par les utilisateurs de BD ou les constructeurs de SGBD.

La mise en oeuvre d'un banc d'essai consiste à effectuer une série de tests sur un SGBD existant dans le but d'estimer ses performances dans certaines conditions. Pour ce faire, trois éléments sont nécessaires.

— Une base de données décrite par un schéma conceptuel et une méthode de génération de données conformes à ce schéma.

— Une charge composée d'un ensemble de requêtes de tests comportant des opérations d'interrogation et de mise à jour à lancer sur la BD, ainsi qu'un protocole décrivant le déroulement de leur exécution.

— Des métriques composées d'un ensemble de mesures simples ou composites afin d'exprimer les résultats. Parmi les mesures les plus couramment utilisées, nous pouvons citer à titre d'exemples le temps de réponse, le débit, le nombre d'objets accédés, le nombre d'entrées/sorties effectuées, l'espace mémoire ou de disque occupé, etc. Chaque mesure est liée à un BnF.

Un ensemble de critères primordiaux permettant de qualifier un banc d'essai de « bon » a été défini par Jim Gray [80] :

— la pertinence : le banc d'essai doit couvrir les besoins du plus grand nombre d'utilisateurs possible ;

— la portabilité : il doit être facile à implémenter sur les différents systèmes cibles ;

— le passage à l'échelle : il doit être possible d'étudier des BD de tailles diverses et d'augmenter l'échelle du banc d'essai ;

— la simplicité : il doit être compréhensible sinon il manquera de crédibilité.

Ces critères s'opposent souvent et la garantie de l'un diminue celle de l'autre. La conception d'un banc d'essai implique, donc, l'optimisation de ces critères.

L'industrie de l'information a vu naître une large panoplie de bancs d'essais dédiés à l'évaluation des performances des BD. Certains ont connu beaucoup de succès, d'autres moins. Dans la section suivante nous présentons quelques exemples connus.

2.2.1 Les bancs d'essais existants

À chaque évolution de BD, de nouveaux bancs d'essais sont proposés par la communauté, en se basant souvent sur ceux de l'ancienne génération, comme résumé dans le tableau 2.3. Le premier banc d'essai de BD *Wisconsin* fut créé dans le but de tester les premiers SGF et SGBD relationnels afin de comparer les performances des algorithmes des opérateurs algébriques. Par la suite, les bancs d'essais TPC (*Transaction Processing Performance Council*) se sont emparés du marché en permettant de vérifier leur application correcte par les usagers qui souhaitent voir leur produit évalué et de publier régulièrement les résultats de ces tests de performance.

Avec l'arrivée des SGBD objet, des bancs d'essais, comme *OO7*, ont été conçus pour les différentes applications d'ingénierie CAD/CAM (computer-aided design/computer-aided manufacturing). D'autres ont également été dédiés aux SGBD Objet-Relationnels comme *BUCKY*, pour permettre d'évaluer des opérations de navigation et ensemblistes relatives à l'aspect objet et relationnel des opérations OLTP. De même, les BD XML et les BDBO ont eu leurs propres bancs d'essais qui considèrent respectivement les requêtes dites de navigation nécessitant la traversée de la structure du document XML en utilisant des références ou des liens (par exemple, *XOO7*), et les données ontologiques (par exemple, *LUBM*). D'un autre côté, la complexité des requêtes des entrepôts de données et leur volume ont largement contribué au succès des bancs d'essais, avec en tête, une série adaptée des TPC. Enfin, avec l'apparition des BD en nuage

Type de BD	Banc d'essai	Base de données	Quelques métriques
OLTP	TPC-C	client-commande-produit-fournisseur	Débit des transactions Prix / Performance
OLAP	TPC-H	client-commande-produit-fournisseur	Puissance Débit
BD Objet	OO7	CAD/CAM	Débit des traversées Efficacité des mises à jour
BD XML	XOO7	CAD/CAM	Utilisation espace mémoire Temps de conversion de données Temps de réponse
BDBO	LUBM	ontologie de l'université	Utilisation espace mémoire Temps de chargement Temps de réponse + Structuration
NoSQL	YCSB	client-commande-produit-fournisseur	Latence des mises à jours Latence des lectures

TABLE 2.3 – Classification des bancs d'essais existants

(cloud) et les technologies NoSQL, de nombreux bancs d'essais ont proposé des charges de requêtes pour évaluer notamment la performance et le passage à l'échelle des multiples systèmes NoSQL existants [57] (BigTable, PNUTS, Cassandra, HBase, etc.).

Entretemps, plusieurs travaux académiques se sont intéressés au benchmarking. Nous citons pour exemples, *MalStone* [26], un banc d'essai conçu pour mesurer la performance des environnements Cloud, et *Ontodbench* [95], un autre banc d'essai pour évaluer les modèles de stockage des BD à base ontologique. Comme nous le voyons dans cette section, de nombreux bancs d'essais ont été conçus pour différents types de BD. Cependant, ils présentent des limitations évoquées dans la section suivante.

2.2.2 Limites des bancs d'essais

Les bancs d'essais guident les tests des BD et permettent d'évaluer leur performance. Malgré leur succès, surtout dans l'évaluation des algorithmes d'optimisation (sélection des index, vues matérialisées, etc.), ils présentent, toutefois, quelques limites à cause des raisons suivantes.

— Les bancs d'essais, dont l'utilisation n'est pas automatisable, servent beaucoup plus à choisir une configuration physique, et n'offrent pas de mécanismes de réutilisation ni de configurabilité des composantes de conception.

— La principale difficulté pour réaliser une évaluation d'une BD via un banc d'essais est d'utiliser un schéma et sa population qui se rapprochent le plus du schéma et données de

l'application réelle dont la performance dépend également des méthodes d'accès, taille du cache, système d'exploitation, hardware et d'autres paramètres de configuration. En effet, les bancs d'essais sont loin de représenter les cas réels d'utilisation, et sont rarement représentatifs de la performance d'une application donnée. Cependant, ils offrent des lignes directrices aidant à établir des attentes opérationnelles basiques, et s'avèrent souvent être le seul moyen d'obtenir ces lignes directrices.

— La plupart ont une structure rigide (schéma figé), ce qui limite leur utilisation dans les BD où il existe plusieurs schémas.

— Certains concepteurs de bancs d'essais ont misé sur le critère de la simplicité. Cela les a conduits à concevoir des bancs d'essais très ciblés, difficilement réutilisables dans d'autres contextes que celui dans lequel ils ont été créés. Par exemple, un banc d'essais dont la charge ne serait constituée que de requêtes d'interrogation serait mal adapté à l'évaluation des techniques d'indexation, pour lesquelles il faut non seulement mesurer les gains en temps de réponse, mais aussi la surcharge engendrée par la maintenance des index lors des mises à jour. Étendre la charge devient alors indispensable pour assurer la pertinence du banc d'essai. D'autres concepteurs ont misé sur le critère de la pertinence. Ceci se traduit par une démarche orientée vers la généricité et une grande paramétrabilité. Cela peut rendre le banc d'essai difficile à appréhender et à utiliser. Mais ce choix dépendra toujours des clients de ces bancs d'essais qui ont généralement des besoins sensiblement différents.

2.3 *Advisors* : outils d'aide à l'administration de BD

Cette catégorie représente le mieux le premier axe de la variabilité *aide à la décision*. En effet, elle englobe les *advisors*, des outils assistant les administrateurs de BD confrontés à plusieurs décisions complexes à prendre durant les phases de conception physique et tuning. Les premiers travaux dans ce sens traitaient l'élaboration d'outils et d'approches permettant de sélectionner une configuration d'index et depuis l'idée a pris de l'envergure pour inclure les différentes structures d'optimisation. Nous allons d'abord détailler la conception physique et le tuning des BD qui sont à l'origine des advisors.

2.3.1 Conception physique et tuning des BD

Le problème de la conception physique consiste à optimiser l'exploitation de la BD et plus particulièrement l'exécution des requêtes par rapport à un ou plusieurs critères non fonctionnels (BnF), en jouant sur plusieurs paramètres de la configuration physique (*structures d'optimisation* : {Index (Ix), fragmentation horizontale (FH), fragmentation verticale (FV), vues matérialisées (VM), gestion de buffer (GB), ordonnancement de requêtes (OR), clustering (CL), traitement parallèle (TP), etc.}, modèle de déploiement, logiciel (application SGBD/noyau système) et matériel (support(s) de stockage, processeur(s)) (voir figure 2.2).

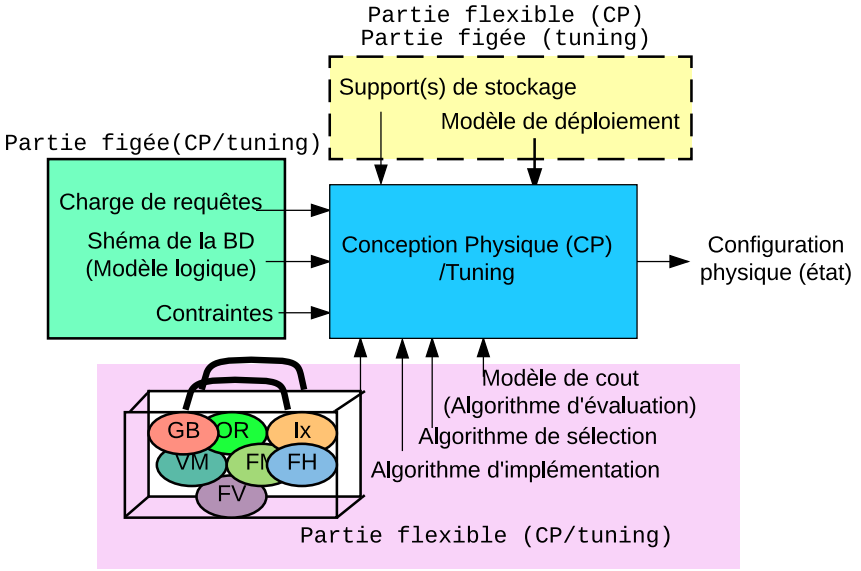


FIGURE 2.2 – Aperçu du problème de la conception physique et du tuning [16]

Le temps d'exécution (plus précisément le throughput [17]) a longtemps été le critère le plus convoité et la mesure la plus répandue de la performance des BD. Après le déploiement du modèle physique et l'opérationnalisation de la BD, on parle plutôt de *tuning* que de conception physique. Le tuning est défini comme un processus continu de réglage dans l'objectif d'atteindre une performance maximale de toutes les composantes d'un système de BD [34]. En effet, la configuration physique initiale, issue de la conception physique (pré-déploiement), doit être périodiquement entretenue afin de supporter l'évolution incontournable de la BD qui peut se manifester, entre autres, par un changement significatif au niveau : (i) du contenu des tables, (ii) de la nature et la fréquence des requêtes et des mises à jour, (iii) de l'espace de stockage, (iv) de la structure de la BD, (v) de la taille des techniques d'optimisation sélectionnées lors de la conception physique (les index, les vues matérialisées et les partitions).

Contrairement à la conception physique, le tuning agit principalement sur la partie flexible (logicielle) de la configuration physique qui est essentiellement représentée par les structures d'optimisation (voir figure 2.2). Cela consiste à sélectionner des schémas de structures d'optimisation en vue d'optimiser l'exécution des requêtes tout en respectant les contraintes prédéfinies, et d'éviter de refaire toute la conception physique. Pour ce faire, l'administrateur se trouve face à des choix critiques difficiles à faire pour les raisons suivantes.

— Chaque technique d'optimisation a plusieurs algorithmes et approches d'implémentation.

16. Les trois parties qui composent la figure sont soit qualifiées de flexible ou de figée selon que les éléments les composant puissent être contrôlées ou pas par le processus de conception physique et/ou de tuning. Par exemple, le support de stockage est défini pendant la CP, donc il appartient à une partie flexible vis-à-vis de la CP, mais il est plutôt figé pendant le tuning, donc il appartient à une partie figée vis-à-vis du tuning. La charge de requêtes quant à elle est une entrée qui ne peut être définie/paramétrée ni par la CP ni par le tuning, donc elle appartient à une partie figée vis-à-vis de ces deux processus.

17. Le throughput ou le débit représente le nombre des enregistrements lus/écrits dans un laps de temps donné.

En guise d'exemple, l'indexation peut s'implémenter, entre autres, via les arbres B-tree, le hachage ou les vecteurs binaires.

— Chaque technique possède un ensemble important de solutions candidates, dont le choix de la solution optimale représente un problème NP-Complet [120], ce qui signifie l'absence d'algorithme exhaustif proposant une solution optimale en un temps acceptable. Reprenons l'exemple des index : ces derniers peuvent être définis sur plusieurs attributs issus de plusieurs tables dont le nombre est considérable. Les algorithmes alternatifs proposés comportent généralement deux étapes : l'élagage de l'espace de recherche (définition des solutions candidates) et le choix de la solution finale.

— L'administrateur peut choisir une seule technique d'optimisation (sélection isolée) ou plusieurs (sélection multiple), car chacune a ses propres avantages/inconvénients voire même des similarités avec d'autres techniques. De même, plusieurs solutions s'offrent aux administrateurs car il existe une interaction entre certaines techniques d'optimisation où le changement dans la sélection de la structure dépendante entraine un changement dans la sélection de la structure dominante [198].

Il est important de noter qu'au bout d'un certain nombre de réglages effectués, des changement structurels s'imposent. À ce moment là, le tuning pourrait ne plus améliorer les performances mais plutôt les dégrader, car le schéma physique ne correspond plus à l'état actuel de la BD. Dans ce cas, les concepteurs doivent *refaire* la BD depuis le début en repassant par les étapes classiques du développement à savoir : analyse des besoins, conception et déploiement (voir section 1.2). À cet effet, il existe des outils [38] qui aident l'administrateur dans la détection de l'inutilité du tuning, et donc la nécessité du changement définitif de schéma physique.

Les éditeurs de SGBD commerciaux ont très vite investi dans le tuning, notamment pour les raisons suivantes.

— L'évolution continue des technologies de BD offre plus de choix de tuning à l'administrateur, ce qui complique davantage sa tâche d'administration, surtout en présence d'interaction entre les différentes composantes du SGBD.

— De nouvelles métriques de performance (énergie, sécurité, etc.) se sont imposées dans le marché mouvementé des BD, à côté du traditionnel temps d'exécution (throughput) ce qui, à son tour, a augmenté la complexité du tuning.

— L'administrateur de BD se voit affecter plusieurs tâches aussi compliquées l'une que l'autre, à gérer en même temps ce qui diminue l'efficacité et augmente les coûts de cette administration, contrairement au coût de matériel qui est en baisse constante [136].

Dans l'objectif d'aider l'administrateur à faire les bons choix dans un tel contexte, tout en réduisant la charge de travail qui lui incombe, les éditeurs des SGBD se sont tournés vers des solutions d'assistance (semi)-automatiques qui proposent des recommandations de tuning concernant un ensemble de structures d'optimisation [8, 198, 64]. Ces solutions sont connues sous le nom d'*advisor* ou d'outils d'aide et sont généralement propres à un SGBD donné. Une comparaison entre les outils de certains SGBD leaders du marché est illustrée dans le tableau

2.4. L'évaluation des recommandations générées par les advisors est réalisée soit par un modèle de coût mathématique, soit par le modèle de l'optimiseur du SGBD (décrits dans la section suivante).

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

TABLE 2.4 – Comparaison des principaux advisors[34]

Bien que les advisors ont été initialement créés pour réaliser le tuning d'une BD, leur usage peut facilement s'étendre à la conception physique où les données peuvent être remplacées par des statistiques ou des mocks de BD, et les requêtes peuvent être déduites de l'analyse préalable des besoins. Leur usage peut également s'étendre à d'autres phases [199, 162].

2.3.2 Limites des advisors

— Ces outils ne dévoilent généralement pas les algorithmes utilisés pour la sélection des techniques d'optimisation. On leur reproche souvent le manque d'interactivité et de personnalisation avec l'administrateur qui pourrait apporter son expérience afin d'améliorer la qualité des techniques d'optimisation recommandées.

— La robustesse de ces outils est mise en question dans [62] qui souligne leurs limites face aux requêtes non représentatives ou aux hypothèses non valides dans un environnement réel.

— Ces outils ne comprennent pas de mécanismes de réutilisation et de configurabilité des composantes de conception, afin d'atteindre une gestion efficace de variabilité.

2.4 Optimisation à base de modèles de coûts

Dans un environnement variable où plusieurs solutions sont possibles, les modèles de coût se sont imposés comme le noyau des advisors et des approches d'aide à la décision et de gestion de variabilité.

2.4.1 Définitions

Un modèle de coût (MdC) est une métrique qui sert à évaluer la qualité d'une solution à un problème particulier de BD. Dans un contexte où la diversité de solutions possibles est

forte, cette métrique est essentiellement utilisée pour évaluer plusieurs solutions sans avoir à les déployer sur un système de BD réel et les comparer pour enfin sélectionner la meilleure configuration qui minimise le coût d'exécution d'une charge de requêtes.

Ces métriques sont largement utilisées dans le contexte des BD traditionnelles et avancées [91, 133]. L'origine de leur utilisation remonte à l'optimisation de l'exécution de requêtes. En effet, une requête est composée d'un ensemble d'opérations algébriques, telles que les sélections (par exemples, séquentiel, index), jointures (par exemples, hachage, tri-fusion) et agrégations (par exemples, hiérarchie, ordre total) avec pour chacune son algorithme d'implémentation. Le choix d'un ordonnancement de ces opérations et des algorithmes exécutés constitue *un plan d'exécution*, qui détermine la performance de l'exécution. Par exemple, pour une requête comprenant deux jointures, un plan d'exécution peut montrer, qu'il est plus performant de commencer par réaliser la deuxième jointure, avant la première, et de les exécuter en utilisant un algorithme de *jointure par Hachage*. Une requête peut donc avoir plusieurs plans d'exécution possibles aboutissant tous au même résultat mais avec des performances variables. Le rôle de l'optimiseur consiste, par conséquent, à construire le meilleur plan d'exécution. Pour ce faire, les premiers optimiseurs se basaient sur un ensemble de règles, comme par exemple, descendre les sélections au plus bas du plan (*Push Down Selections*) afin de réduire la taille du résultat des opérations. Désignée sous le nom d'*approche dirigée par des règles* (*Rule-based Approach*), celle-ci s'avérait insuffisante face à l'augmentation de la taille à des schémas de BD impliquant un nombre de plus en plus important de tables (relations) et à la diversité des algorithmes et des exigences. Le problème de génération du plan d'exécution optimal devient alors NP-complet [150]. Une autre approche dite *dirigée par des modèles de coût* (Cost-based Approach) a vu le jour pour venir compléter la précédente. Cette approche vise à évaluer le coût des plans d'exécution afin de choisir celui ayant le coût minimal. 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 et les algorithmes choisis, jusqu'à l'obtention du coût total.

Un MCD doit souvent prendre en considération les composantes principales d'un SGBD (conception physique) [137], où la majorité des BnF se mesurent, et où les solutions se concrétisent. Ainsi, on affecte un coût pour chacune de ces composantes: les entrées-sorties (IO) des processus de lecture/écriture entre la mémoire et le support de stockage, le CPU et la communication sur le réseau. Si une solution distribuée est utilisée, de nouveaux paramètres d'entrée du MdC se sont ajoutés progressivement, et proviennent de différentes phases de conception : (1) *logiques* : requêtes, schéma de BD (attributs, instances, facteurs de sélectivité des prédicats), ainsi que les estimations (statistiques) des volumes de données manipulées par la requête. Le problème d'estimation des résultats intermédiaires a été largement abordé dans la littérature [93], (2) *modèle de déploiement* : modèle et politique de traitement de requêtes dans le cas distribué, et (3) *physiques* : algorithmes d'implémentation et support de stockage.

2.4.2 Travaux existants et limites

L'approche dirigée par les MdC des optimiseurs a grandement inspiré les chercheurs en BD dans la résolution d'autres problématiques que la recherche du plan d'exécution optimal. Nous citons, en premier, le problème de la conception physique, durant laquelle un ensemble de structures d'optimisation doit être sélectionné (comme les vues matérialisées, les index et la fragmentation). Compte tenu de l'importante taille de l'espace de recherche de chaque problème lié à la sélection d'une ou plusieurs structures d'optimisation (voir paragraphe 2.2.3.1), des méta heuristiques, comme les algorithmes génétiques ou le recuit simulé [22], dirigées par des MdC sont proposées. Le déploiement de BD distribuées a également exploité les MdC afin de réaliser efficacement plusieurs tâches telles que : la fragmentation, l'allocation, la réplication et l'équilibrage de charge [173]. Nous pouvons citer également les travaux sur : l'optimisation multi-requête qui exploite leur interaction [12], l'optimisation des BD à base ontologique [27], le routage des réseaux [128] ou l'optimisation des BD multimédia [29].

En ce qui concerne les limites de ces travaux, on peut se référer à celles des advisors, du fait du lien entre ces approches. Les modèles de coût sont ainsi une grande avancée pour le traitement de la variabilité dans les BD. Cependant, ceux-ci ne permettent pas de traiter toutes les phases du cycle de conception des BD, ni leur interaction ce qui est l'objet de nos travaux. Nous étudions maintenant la variabilité du point de vue *réutilisation*.

3 Travaux orientés vers la réutilisation

Actuellement, le développement logiciel fait face à plusieurs défis : croissance et diversité des exigences fonctionnelles et non fonctionnelles (qualité), la taille du logiciel, sa complexité, ainsi que les coûts et la durée de son développement. L'approche '*from scrach*', qui construit un nouveau système à partir de rien, est devenue ainsi inadaptée à ce nouveau contexte.

Dès lors, la réutilisation est souvent vue comme le meilleur moyen d'améliorer la qualité et la productivité indispensables au développement et à l'entretien d'un système d'information, en ne développant plus chaque composante séparément, mais plutôt en le concevant à partir d'éléments réutilisables et de tâches automatisables. Les gains de qualité résultent de l'utilisation de composantes logicielles déjà prouvées, tandis que les gains de productivité proviennent de la réduction des efforts de développement et de maintenance.

La notion de *réutilisation* désigne l'ensemble des théories, techniques, et outils permettant de construire un système en réutilisant des composants existants, ayant été produits à l'occasion de développements antérieurs. Elle est aussi ancienne que le logiciel, et a connu plusieurs évolutions illustrées dans la figure 2.3. Elle concernait initialement des sous-ensemble de code, ce qui ne s'avérait pas suffisamment rentable. Actuellement, elle s'est dotée de stratégies, d'une portée plus large (besoins opérationnels et techniques) et concerne également la conception [75]. Elle peut et doit être pratiquée tout au long du cycle de vie d'une application.

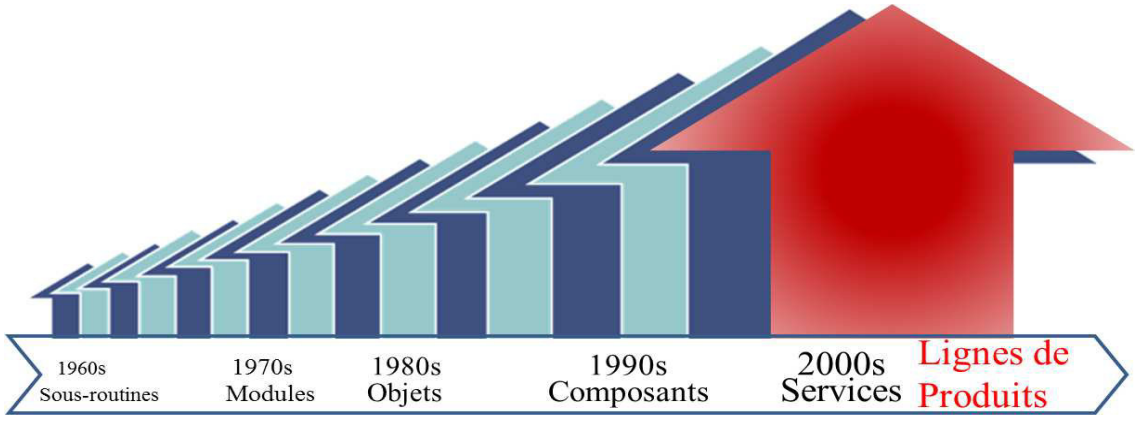


FIGURE 2.3 – L'historique de la réutilisation : d'un usage ad-hoc à systématique [49]

Dans cette section, nous traitons d'abord la réutilisation pré-"*gestion explicite de variabilité*", désignée par l'*ingénierie pour et par la réutilisation des composants*' (*en anglais, 'design for & with reuse'*) [192]. Cette dernière peut être implémentée via plusieurs techniques que nous avons réparties en deux catégories selon l'objectif visé, à savoir *l'automatisation* et *la standardisation*. Toutes les deux visent à réduire la complexité du processus de développement de BD, de plus en plus complexe et composé de nombreuses tâches fréquentes et répétitives qui ne cessent de varier. Notons que cet axe implique la réutilisation, mais pas forcément la configurabilité ni les mécanismes d'aide à la décision. Ainsi, il ne peut pas être qualifié de stratégie de réutilisation (Tableau 2.1).

3.1 Automatisation : les outils CASE

L'automatisation, au sens général du terme, désigne l'exécution et le contrôle d'un ensemble de tâches techniques par des machines fonctionnant sans ou avec aussi peu d'intervention humaine que possible. Elle concerne des tâches/processus répétitifs et fréquents afin d'en réduire le temps, le nombre, les erreurs, les coûts, la complexité, les efforts humains et d'augmenter l'efficacité/productivité du travail ainsi que la standardisation des processus. Ces caractéristiques font d'elle l'un des axes incontestables de la gestion de variabilité.

Si en 1950, les ordinateurs étaient encore peu courants, moins d'un demi-siècle plus tard, ce sont des millions d'ordinateurs qui, isolés ou en réseau, peuvent accomplir des tâches complexes, non seulement de contrôle par des boucles de rétroaction, mais aussi de pilotage de machines, de traitement de données, de circulation de l'information et de simulation (conception et fabrication assistées par ordinateur).

3.1.1 Définitions

Un outil CASE [18] désigne une approche structurée utilisant des interfaces graphiques pour automatiser les activités faisant partie du processus de développement des systèmes, et par extension, celui des BD. Ce dernier est connu pour être un procédé fastidieux et exigeant en temps, ce qui rend nécessaire l'utilisation de méthodes automatisées afin d'accélérer la conception et l'implémentation. Les outils CASE jouent alors un rôle de plus en plus important dans les projets de BD.

Selon leur champ d'action dans le cycle de développement, on distingue trois grandes catégories d'outils CASE : (i) *frontaux* (front-end) qui apportent un soutien à la planification et la conception, (ii) *internes* aux codage et implémentation (back-end), ou (iii) hybrides qui couvrent l'ensemble des activités du processus allant même jusqu'à la gestion du projet et la documentation dans certains produits. Dans le contexte des BD où la modélisation des données constitue une partie importante du processus de développement, on trouve plutôt des outils CASE frontaux. Cette classe d'outils permet au concepteur d'élaborer, en peu de temps, un modèle de la BD à construire, et d'en vérifier sa qualité en effectuant des contrôles de cohérence et de validité. Ainsi, ils permettent de détecter d'éventuelles erreurs de conception avant leur propagation d'autant plus coûteuse qu'elles sont détectées tardivement. Une nouvelle génération d'outils CASE, appelée *moteurs d'analyse et de conception*, offre au concepteur la possibilité d'adapter le processus de modélisation aux besoins spécifiques du projet de BD en question. Pour ce faire, un outil CASE de conception de BD est généralement constitué des composants suivants.

— Un environnement de développement qui contient un panel riche d'outils (essentiellement graphiques) permettant de générer et documenter les différents schémas de BD, processus d'application, rapports, flux de données, éléments d'interfaces et tout autre objet pertinent concernant le processus de développement. Cet environnement renforce, en outre, la communication entre les différents acteurs de la BD en aidant à détecter et corriger les éventuels conflits, violations de contraintes/règles et incohérences ;

— Un dictionnaire très complet de données qui garde la trace de tous les objets, relations et règles créés durant le processus de développement (descriptions d'entités, définitions d'attributs, diagrammes de flux de données, formats de rapports, etc.). Plusieurs outils CASE fournissent des interfaces qui communiquent directement avec le SGBD sous-jacent pour stocker son dictionnaire de données.

— Un répertoire intégré de données pour sauvegarder tout les éléments de développement : dictionnaire de données, résultats (par exemple, le schéma de la BD) et les différentes décisions.

18. CASE est l'acronyme de *Computer-Aided Software Engineering* pour génie logiciel assisté par ordinateur, ou atelier de génie logiciel.

3.1.2 Les outils CASE existants

PowerDesigner [158] est l'un des premiers outils à avoir proposé de développer graphiquement différents modèles de données et de les implémenter de manière automatique, quel que soit le SGBD cible. Il est, en effet, un exemple de solution visant à réduire la complexité (gérer la variabilité) des phases conceptuelle et logique de BD, en mettant à disposition du concepteur et en automatisant plusieurs techniques de conception. Son concurrent *ERwin Data Modeler* [180], est un spécialiste du modèle E/A qui réalise des diagrammes entièrement documentés pouvant être mis en correspondance avec les différents niveaux d'abstraction. Il convient aussi de mentionner *DB-Main* [86], un outil CASE générique dédié à l'étude des problématiques liées à la conception des systèmes d'information complexes. Il comprend, en plus des fonctionnalités standards, des techniques pour l'ingénierie inverse, l'intégration, la maintenance et l'évolution. Il comprend aussi un prototype d'outil d'aide à l'évolution, qui permet notamment de générer automatiquement les programmes de modification de la BD à partir des opérations de modification de ses schémas conceptuel et/ou logique.

D'un autre côté, l'utilité des outils CASE a poussé les principaux fournisseurs de SGBD relationnels, à s'y intéresser, offrant désormais des outils CASE complètement intégrés pour leur propre logiciel, aussi bien que pour les SGBD relationnels concurrents. Les outils proposés par *Oracle*, par exemple, peuvent être utilisés avec *Microsoft SQL Server*, *IBM DB2* et *SQL/DS*. Le tableau 2.5 dresse une liste non exhaustive de fournisseurs d'outils CASE.

Outil	Prix [19]	Plate-forme	Fonctionnalités			
			DC&CL&CP	OLAP	XML	Merise
ERwin	>8000$	win	✓	✓	✓	
ER/studio	~1400$	win	✓	✓		
PowerDesigner	>7000$	win	✓	✓	✓	✓
Visio	~600$	win	✓		✓	
Rational	~5000$	multi	✓			
DB-Main	0$	win	✓	✓	✓	

TABLE 2.5 – Comparaison des principaux outils CASE pour bases de données

3.1.3 Limites des outils CASE

Il ne fait aucun doute que les outils CASE ont largement contribué à l'amélioration de l'efficacité des concepteurs et développeurs de BD durant le processus de conception. Cependant, aussi perfectionnés qu'ils soient, les outils CASE doivent leur efficacité à la maîtrise des fondements de conception par leurs utilisateurs. Entre les mains d'un novice, ces outils produisent,

19. Le prix dépend de la version du produit.

au mieux, une mauvaise conception avec de jolis schémas. En effet, la modélisation de données est plus un exercice de conception que technique. Un autre inconvénient, est leur manque de mécanismes de configurabilité (adaptabilité) et d'aide à la décision.

3.2 Standardisation : les design patterns

Standardiser une technique, revient à la rendre conforme à des caractéristiques communes d'un ensemble d'approches similaires. Synonyme d'efficacité mais réductrice de diversité, elle vise à généraliser l'usage de la technique pour qu'elle puisse être adoptée par un grand nombre d'utilisateurs, facilitant ainsi l'interopérabilité, contrairement aux solutions propriétaires. Elle implique l'automatisation.

3.2.1 Définitions

> *"A pattern is an idea that has been useful in one practical context and will probably be useful in others."* [67]

Un *design pattern* ou *patron de conception*, est une structure générique qui permet de résoudre un problème spécifique mais courant. Elle est surtout utilisée pour des problèmes pouvant se décomposer en un ensemble de modules (orientés objets) indépendamment du langage de programmation, et est standardisée pour que tous puissent s'y référer. On parle parfois de "*bonnes pratiques*" (*best practices*) car c'est l'expérience qui montre que la solution est valide. À l'inverse, des anti-patterns montrent ce qu'il ne faut pas faire.

Le besoin de créer et d'utiliser des design patterns vient du fait qu' il y a souvent des problèmes de conception récurrents durant le développement des systèmes, en créer un modèle de solutions permet d'exploiter les connaissances et la précieuse expérience déjà acquises afin de gagner du temps, d'éviter aux autres développeurs de buter sur le même souci et de faciliter la maintenabilité du code, la réutilisation et l'adaptation de cette solution lors d'une nouvelle occurrence du problème. D'un autre côté, l'orienté-objet est devenu un principe incontournable dans le développement des systèmes, ce qui fait que les programmes sont souvent décomposés en plusieurs composants plus simples et réutilisables. Ces derniers peuvent devenir ingérables s'ils sont découpés sans règles cohérentes et précises. Ainsi les patrons de conception permettent une meilleure gestion des différents composants d'un programme. Cependant, les design patterns ne sont pas forcément simples à implémenter et il serait dommage de les utiliser si le problème correspondant n'est pas présent. On peut classer les design patterns en trois catégories :

— les modèles de création : ce sont des design patterns qui permettent de gérer la construction d'objet;

— les modèles de structure : ils permettent de créer des ensembles de classes cohérentes;

— les modèles de comportement : ils permettent de gérer le comportement des responsabilités entre les objets.

L'application des approches de standardisation du génie logiciel (approches *UML*, *design patterns* et *processus unifiés*) aux BD permet de tirer profit de la vaste expérience acquise dans ce dernier pour mieux résoudre les différents enjeux de conception des BD. En l'occurrence, les designs patterns ont pu être exploités dans les différentes étapes du cycle de développement de BD. En effet, cet axe, et plus particulièrement les designs patterns permettent de gérer la variabilité qui a une importance croissante dans les BD. En effet, le concept des designs patterns (et plus généralement la réutilisation/automatisation) réduit la variabilité grâce à la standardisation. L'idée est qu'un développeur de logiciel doit d'abord essayer de comprendre les techniques existantes (designs patterns) pour ensuite les réutiliser ou les améliorer. Réinventer la roue en proposant de nouvelles techniques et solutions à un problème déjà résolu n'est pas nécessaire, et augmente la variabilité. Cependant, nous admettons qu'il peut y avoir plus d'une solution pour chaque problème, qu'elle soit ad hoc ou récurrente. Mais, certaines solutions possèdent de meilleures caractéristiques que les autres. Ces dernières doivent être encapsulées dans les design patterns.

3.2.2 Les design patterns existants

La majorité des travaux de recherche exploitant les designs patterns dans le contexte des BD, se situent aux niveaux logique et physique. Ils suivent la logique relationnelle et peuvent concerner l'aspect modélisation (schéma) ou comportemental (traitement).

Au niveau conceptuel, nous pouvons inclure les schémas-patterns qui sont des modèles conceptuels (ou logiques normalisés) "*préfabriqués*" mettant à disposition des concepteurs, des schémas génériques qui peuvent être personnalisés et adaptés aux règles de gestion spécifiques d'une organisation donnée. Cette idée est d'ailleurs similaire aux ontologies, des solutions émergentes dans les BD et dans d'autres domaines. Ces modèles universels font office d'un template de plusieurs sujets capitaux à l'instar de : clients, comptes, produits, documents, et/ou les fonctions essentielles de l'entreprise telles que : achats, ventes, comptabilité, livraisons, ainsi que les applications spécifiques à l'industrie [169, 168]. Cela permet entre autres de réduire considérablement le temps voire le coût de développement de BD, en utilisant des structures universelles de qualité souvent supérieure à celles établies par les équipes internes de développement.

D'autres travaux se sont intéressés à l'utilisation des design patterns au niveau du développement de l'application [88] tandis que d'autres traitent le cas des entrepôt de données. Stathopoulou [174] présentent une approche d'interaction avec les BD, avec un classement des design patterns en trois catégories :

— *pivotement* qui permet de transformer les lignes des tables relationnelles en colonnes, et vice versa, souvent utilisé pour les flux de données;

— *matérialisation* qui permet l'implémentation d'une relation père-fils entre les tables (re-

lation *un-à-un*), l'équivalent de classe abstraite;

— *généralisation* et *spécialisation* (relation *is-a*) qui sont souvent rencontrées dans les SGBD objet.

Parmi les patterns les plus connus portant sur l'accès aux ressources persistantes, figure en tête, celui des objets d'accès aux données (DAO pour *Data Access Object*). Ce dernier vise à encapsuler le code responsable du stockage des données dans une couche indépendante du type de support utilisé (voir figure 2.1). L'idée est qu'au lieu de faire communiquer directement nos objets métier avec la BD, ou plus généralement ce qui fait office de système de stockage, ces derniers vont communiquer avec cette couche qui, à son tour, interagira avec ce système de stockage. Le tableau 2.6 répartit quelques design patterns existants sur l'ensemble des phases de conception des BD.

Phase	Design Conceptuel	Conception Logique	ETL	Conception Physique
Travaux	[169, 168]	[169, 168]	[183]	[11]

TABLE 2.6 – Répartition des design patterns proposés sur les phases de conception de BD

3.2.3 Les limites des design patterns

Les design patterns agissent à un niveau plus abstrait que les primitives de modélisation en offrant un développement de blocs réutilisables. Leur utilisation engendre des coûts en terme de taille et de temps, mais offre en contrepartie plus de flexibilité et de maintenabilité tout en évitant le phénomène de la réinvention de la roue. En revanche, il présente, entre autres, les limites suivantes.

— Couverture sporadique : on ne peut pas construire un modèle par simple combinaison de patterns. Typiquement, on va uniquement utiliser quelques modèles pour quelques problèmes qui se posent pendant la conception.

— L'identification du design pattern adapté à un problème donné reste difficile. Le problème peut, en effet, avoir des caractéristiques légèrement différentes de celui traité par un design pattern. Il devient alors difficile de savoir si le design pattern s'applique dans ce cas précis.

— La complexité des patterns : ils représentent des notions avancées et peuvent ainsi s'avérer difficiles à comprendre.

— Des inconsistances se produisent lors de la fusion des (utilisation de plusieurs) design patterns, et ce malgré les différents efforts déjà entrepris dans ce sens là.

— L'immaturité de la technologie qui est encore en évolution.

— Le manque de mécanismes de configurabilité (adaptabilité) et d'aide à la décision incite à tourner vers d'autres solutions complémentaires.

4 Travaux orientés vers la gestion explicite de la variabilité

Force est de constater que les solutions présentées jusqu'ici pour gérer la variabilité dans la conception des BD, sont des approches disséminées dans le cycle de conception. Aucune ne représente une démarche en soi. Dans un contexte évolutif de surabondance de solutions et de diversité d'outils et de phases impliquées, il devient primordial de proposer une *méthodologie fédératrice* de gestion *explicite* de variabilité.

Par *méthodologie*, on entend une *stratégie* de réutilisation qui vient compléter la réutilisation classique en mettant l'accent sur la dimension de la configurabilité (adaptabilité, flexibilité, capacité de personnalisation ou encore ré-ingénierie [20]). Celle-ci désigne la faculté d'un système à être modifié, étendu ou paramétré afin de produire de nouvelles fonctionnalités/caractéristiques ajustées aux exigences du contexte sous-jacent.

Et par gestion *explicite* de variabilité, on entend distinguer la réutilisation concernant les parties communes présentes chez tous les membres dérivés du système, de celle qui concerne les parties variables qui ne doivent être intégrées que chez certains membres. Cela permet ainsi de bien instaurer la configurabilité, et rendre efficace l'activité de développement d'un nouveau système. À l'opposé, les approches classiques de réutilisation géraient *implicitement* la variabilité, car les composants réutilisables sont créées à partir des connaissances invariantes entre plusieurs systèmes. Par conséquent, l'adaptation n'est pas déclinée et requiert plus d'efforts.

En bref, la méthodologie doit offrir aux concepteurs, un cadre de travail unifié et des lignes directrices de conception, en s'appuyant sur une stratégie de réutilisation (réutilisation + configurabilité) et des techniques d'aide à la décision pour rendre encore plus efficace l'adaptation. En effet, le choix entre les variantes (valeurs de paramètres) au moment de la configuration, peut être direct, c'est à dire fait par l'utilisateur (manuel), comme il peut faire appel à des techniques d'aide à la décision ((semi-) automatique) associées à certains points de variation.

La gestion de la variabilité est très prisée dans le domaine du génie logiciel qui devance largement celui des BD en la matière. On y trouve plusieurs méthodologies autour des stratégies de réutilisation. L'ingénierie de BD étant une branche spéciale du génie logiciel, nous visons à exploiter les progrès de ce dernier dans la gestion de la variabilité du processus de conception de BD. La section suivante sert d'introduction aux notions importantes liées à la gestion de la variabilité en génie logiciel.

20. Tout ces termes renvoient au comportement variable du système selon un contexte donné. Cependant, il existe des nuances quant à l'implémentation de la variation. La personnalisation implique une modification "manuelle" des méthodes d'agencement ou du code des composantes, contrairement à la configuration qui consiste uniquement à changer des paramètres entraînant une génération de code/agencement de composantes automatiques. La ré-ingénierie d'applications combinée avec la réutilisation de composantes permettent ce qui peut s'appeler la reconfiguration évolutive [99]. La ré-ingénierie sans aucune modification des modules est similaire à de la réutilisation basique. Nous considérons les différents termes comme équivalents tout au long de ce manuscrit.

4.1 Notions sur la gestion de la variabilité en génie logiciel

La majorité des organisations produisent des familles de systèmes assez similaires, se distinguant uniquement par quelques caractéristiques. C'est la raison pour laquelle, la réutilisation, dont l'origine remonte aux années 1960, a été et continue d'être un domaine important du génie logiciel.

Au début, chaque produit logiciel se construisait isolément, dans un but très précis. Cette approche s'est révélée coûteuse et longue, et a donc rapidement évolué vers une production de masse appelée "*lignes de production*". Des éléments constitutifs communs y sont d'abord développés, ensuite une personnalisation de masse est appliquée pour dériver les différents produits finaux. En effet, les ingénieurs logiciels [123, 139] ont très tôt eu l'ambition de composer des systèmes logiciels en assemblant des composants. Ils ont dû se focaliser au début sur les connaissances des domaines pour élaborer une réutilisation plus stratégique avec une personnalisation de masse efficace, d'où les premières approches pilotées par le domaine : programmation générative, usines logicielles et plus récemment les lignes de produits logiciels. Ces approches reposent sur des techniques importantes comme la programmation orientée-objet, programmation orientée aspect, design patterns, frameworks, générateurs de programmes, langages dédiées de programmation, et ainsi de suite (voir figure 2.3). Dans le cadre de notre étude, nous nous sommes intéressés aux *lignes de produits logiciels* (LdP) dont la supériorité sur les autres méthodologies dans la maîtrise de variabilité des BD, a été prouvée [36].

Notons que le terme *variabilité* renvoie aux endroits du processus de développement de logiciels où le comportement peut être configuré. Ces endroits sont désignés sous le nom de *points de variation*. La variabilité a été largement étudiée sous différents aspects : modélisation [170], implémentation [178], test [98], etc. Ces résultats ont été exploités par de multiples disciplines impliquant un cycle complexe de développement.

4.1.1 Lignes de produits (LdP)

Une ligne de produits logiciels (LdP), en anglais *software product line*, est un ensemble de systèmes partageant un ensemble de propriétés communes et satisfaisant des besoins spécifiques d'un domaine particulier [49]. Le principe est une transposition des chaînes de production au monde du logiciel, et doit donc prendre en considération les communautés et les variabilités des applications à toutes les étapes de leur cycle de vie et pour toutes les sortes de produits.

Ainsi, le développement de chaque produit se fait par composition de fonctionnalités (features) communes. Quant aux fonctionnalités (différences) qui distinguent un produit d'un autre, elles restent ouvertes. On appelle ces dernières des *points de variation*, un concept clé des LdP, car leur absence signifie qu'il s'agit d'un seul produit.

Les LdP incarnent une réutilisation systématique, à grande échelle avec une personnalisation de masse. Elle offre donc de multiples avantages, dont la haute flexibilité, productivité,

qualité, prédictibilité, maintenabilité, et ainsi de suite. La personnalisation de masse permet le report de certaines décisions de conception à un stade ultérieur, ce qui permet d'optimiser la stratégie métier. Cependant, mettre à jour une LdP s'avère bien plus compliqué que l'évolution des produits logiciels individuels.

Les LdP se sont imposées comme une approche multi-disciplinaire à succès : *VLSI* (intégration à très grande échelle) [188], systèmes avioniques [60], téléphonie mobile [125], dispositifs médicaux[114], système de commande des moteurs [115] et téléports [49]. Une communauté des LdP s'est ainsi fondée à travers des conférences dédiées telles que SPLC (*Software Product Line Conference*) [21] qui est à sa 20ème édition en cette année, ainsi que différents projets européens [22]. Le fameux *Software Engineering Institue* des états unis a même crée une action spéciale spécifique à l'ingénierie des LdP [23], et il y a publié diverses expériences industrielles autour des LdP.

4.1.2 Étapes de la gestion de la variabilité

La gestion explicite de la variabilité signifie la capacité d'exprimer les variations possibles d'une application, en se basant sur deux composantes : l'architecture de composition, et la configurabilité. Il existe plusieurs approches pour mener à bien cette tâche [189].

L'introduction de la variabilité dans une famille de produits logiciels, doit passer par des étapes illustrées dans la figure 2.4 [170], et détaillées dans les sections suivantes.

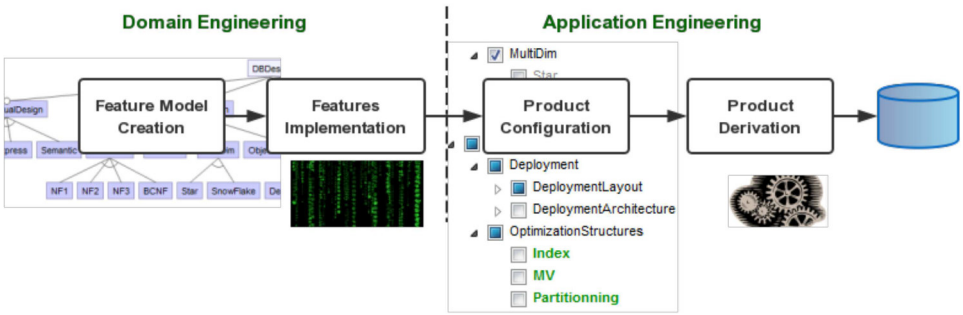


FIGURE 2.4 – Étapes de la gestion de variabilité

4.1.2.1 Identification de la variabilité. La première étape consiste à détecter les caractéristiques communes et variables entre les produits de la famille et à définir l'architecture permettant la réutilisation systématique. Cela rentre dans le spectre de l'ingénierie de domaine, et implique les deux sous-tâches suivantes.

21. http://splc.net/history.html
22. FAMILIES entre autres. https://itea3.org/project/families.html
23. http://www.sei.cmu.edu/productlines/

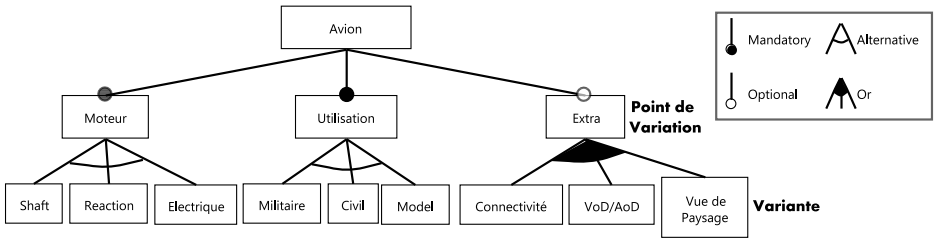


FIGURE 2.5 – Aperçu d'un diagramme de caractéristiques des avions

1. Modélisation : plusieurs techniques de modélisation orientées gestion de variabilité [170] existent dans la littérature, comme l'*Analyse de domaine orientée features* (*FODA*) [97], Koalish [14] ou VSL [18]. La variabilité est facilement identifiée si le système est modélisé en utilisant le concept de "*feature*" (caractéristique ou fonctionnalité), défini comme "*une unité logique d'un comportement déterminé par un ensemble d'exigences fonctionnelles et non fonctionnelles*" [30]. Elle peut être une exigence client, une fonctionnalité technique, un groupe de fonctionnalités, ou simplement une caractéristique non fonctionnelle. Notons que cette notion de feature, couvre la notion des points de variation et de variante. La figure 2.5 montre un exemple d'un diagramme de *features*. Celui-ci prend toujours la structure d'un arbre dont les noeuds représentent les features, et les arcs ainsi que les groupements des features représentent la variabilité. Il englobe des features *obligatoires* ou *optionnelles*. Chacun noeud peut avoir des "enfants" sous forme de groupes de features *exclusifs* ou *alternatifs*. Les règles suivantes sont appliquées, lors de la spécification des features à inclure dans une variante (binding).

- Si une feature parente est contenue dans une variante, toutes les features enfants obligatoires doivent être également incluses (n de n) au moment du binding. Elles sont dénotées par un cercle noir, comme *Moteur* et *Utilisation* dans la figure 2.5.
- Un ensemble de features optionnelles peut aussi être inclus (m de n / $0 \leq m \leq n$). Celles-ci sont dénotées par un cercle blanc, comme *Extra* dans la figure 2.5.
- Une et une seule feature appartenant à un groupe "exclusif" (*XOR*) doit être sélectionnée (1 de n). Cela est symbolisé par un demi cercle blanc connectant les arcs enfants. C'est le cas de *Moteur* avec ses enfants *Shaft*, *Réaction*, *Électrique*.
- Au moins une feature appartenant à un groupe "alternatif" (*OR*) (m de n / $1 \leq m \leq n$) doit être sélectionnée. Un demi cercle noir connectant les arcs symbolise ce type, comme c'est le cas pour *Extra* avec son groupe d'enfants (*Connectivité*, *VoD/AoD*, *Vue de Paysages*).
- Afin d'augmenter l'expressivité de ces modèles, de nouvelles versions ont vu le jour, comme celle qui intègre les cardinalités [59] (Un *Avion*, par exemple, peut avoir entre 2 et six *Moteur*), et celle qui offre un modèle orthogonale de variabilité [142].

2. Deuxième étape: une fois que les features sont identifiées, toutes leur combinaisons ne sont pas forcément valides, c'est à dire qu'elles ne correspondent pas forcément à un produit réel de la famille. En effet, certains choix sont incompatibles entre eux, d'autres sont liés. Afin d'effectivement préciser ces combinaisons valides, des contraintes de cohérence permettant de faciliter

les choix lors de la dérivation, doivent être ajoutées. Ces contraintes sont de type *impliquer* et *exclure*. À titre d'exemple, *Utilisation militaire* exclut la feature *Extra*, et *Utilisation civile* implique *Moteur électrique*. Notons qu'un diagramme de caractéristiques doit être enrichi par les contraintes pour être qualifié de *modèle de features* (FM). Ce dernier s'impose comme moyen fondamental pour spécifier et raisonner sur la variabilité et la commonalité des LdPs en termes de caractéristiques.

4.1.2.2 Implémentation de la variabilité. La deuxième étape consiste à construire un produit particulier, et donc à définir la manière dont les produits finaux vont être dérivés. Il s'agit de choisir, parmi la panoplie qui existe [178], une technique appropriée de réalisation de la variabilité. Cela rentre dans le spectre de l'ingénierie de l'application et se déroule en deux sous-étapes :

— la configuration : l'utilisateur commence d'abord par figer (sélectionner) les features vis-à-vis de la variabilité définie, qui correspondent à ses exigences. Un choix particulier lors de la dérivation d'un produit peut exclure ou exiger d'autres choix (les contraintes);

— la dérivation : chaque feature est instanciée et remplacée par son rôle (code, script, test, documentation, modèle, etc.).

Une dernière étape peut également être considérée. Elle concerne la maintenance du modèle des features, par l'ajout, la suppression et la modification de ses composantes.

4.1.3 LdP et ingénierie des modèles

L'ingénierie des LdP et l'ingénierie de modèles (IDM) partagent les mêmes préoccupations, à savoir l'augmentation de la productivité et de la qualité des produits logiciels, automatisation de leur développement, séparation des connaissances du domaine de celles de la technologie, et ainsi de suite. Cependant, LdP et IDM sont deux techniques complémentaires, du fait que l'IDM permet l'automatisation de la chaine de production des LdP [157]. Nous avons opté pour la technique des LdP car l'IDM favorise la création des modèles des systèmes en question à un niveau supérieur d'abstraction, tandis que les LdP représentent un processus logiciel qui met plutôt l'accent sur la *réutilisation* organisée à travers une architecture logicielle commune.

Malgré le succès des LdP et l'émergence de la gestion de variabilité, les travaux l'abordant dans le contexte des applications BD demeurent rares. Parmi ceux-ci, on retrouve le travail de Broneske et al. [36] qui explore les méthodes actuelles de la maîtrise de la variabilité dans les SGBD, ainsi que d'autres travaux que nous résumons dans ce qui suit.

4.2 Travaux existants portant sur la phase physique

La conception physique a toujours été le premier centre d'intérêt de la recherche en BD. 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 (modèles logique et de déploiement) à traduire vers la sémantique du SGBD cible. La complexité et la diversité des tâches y sont aussi pour beaucoup. D'ailleurs, la majorité des BnF y sont évalués, notamment le plus convoité : la performance.

La gestion de la variabilité n'a pas dérogé à cette règle. En effet, plusieurs travaux s'y sont intéressés, touchant ainsi aux différents éléments de la conception physique, avec en tête le SGBD, mais aussi le moteur des requêtes, le modèle de déploiement et ainsi de suite.

4.2.1 Variabilité des SGBD

Initialement, les SGBD ont été relativement statiques, uniformes et tout changement nécessaire exigeait une modification complexe et onéreuse du code source (par exemple, les SGBD hiérarchiques). Peu à peu et compte tenu de l'adversité croissante des application de BD, les développeurs ont d'abord opté pour des SGBD à usage général, qui sont riches en fonctionnalités et conçus pour répondre aux besoins d'une large gamme d'applications : BD plus grandes que la mémoire centrale, architecture client-serveur, points de contrôle et récupération, etc. *Oracle Database* [24] en est un parfait exemple.

En revanche, n'importe quel principe ne peut être généralisé au delà d'un certain degré, car la généralité introduit souvent un "surplus" fonctionnel au détriment de la performance et de la clarté, d'autant plus qu'elle s'avère inappropriée dans les environnements à forte contraintes (avec des exigences très particulières comme les systèmes embarqués). Par conséquent, des solutions à usage spécifique ont vu le jour, basées sur le principe qu'une application ne peut atteindre ses meilleures performances que si elle repose sur un framework "mean and lean" (pour "maigre et aigre") qui correspond exactement aux scénarios spécifiques de l'application. Par exemple, *PicoDBMS* [144] est un SGBD "léger" dédié à la gestion des cartes à puces et *TinyDB* [118] est un langage de requêtes spécifique aux réseaux de capteurs qui facilite la description des données aux utilisateurs, sans connaître la façon dont les données sont traitées.

Cependant, ce genre de solutions requiert souvent un re-développement - en partant de zéro - de grandes parties des SGBD, conduisant à une duplication des efforts d'implémentation et des coûts/durées de développement, voire même une réduction de la qualité logicielle.

Entre l'empilement d'un large éventail de fonctionnalités et la réinvention de la roue pour chaque nouveau scénario, les deux solutions présentent des limites en termes de gestion de variabilité puisqu'elles ne fournissent pas une stratégie de réutilisation de composantes appartenant à des SGBD similaires. En effet, la gestion de la variabilité se concrétise en ralliant la généralité de la première solution et la spécificité de la deuxième, grâce à la configurabilité des *solutions personnalisables*. Ces dernières sont utilisées pour générer des architectures extensibles ou des SGBD "sur mesure", en offrant un degré élevé de réutilisation/configurabilité. Elles peuvent être réalisées en utilisant diverses techniques [151], comme récapitulé ci-dessous.

24. https://www.oracle.com/DB/index.html

- Les approches par composants permettent de façonner partiellement ou totalement des SGBD à partir d'un ensemble de composants, de façon à répondre à des exigences spécifiques. On distingue différents degrés d'ajustement, donnant lieu à trois types de SGBD "*ajustables*" : (i) les extensibles, qui étendent des SGBD existants par des fonctionnalités non standards [15], (ii) d'autres fournissent les fonctionnalités standardisées sous forme de services [138], et (iii) les configurables (personnalisables), composent des SGBD non-standards à partir de composants réutilisables à l'instar de *KIDS* [73], et de *COMET* [135]. Cependant, l'aspect transversal (en anglais, cross-cutting concerns) de certaines fonctionnalités, la consommation considérable de la mémoire, ainsi que la perte de performance causée par la modularisation ont été les facteurs fatals à ce type d'approches [151]. Il convient également de mentionner le générateur GENESIS [17] comme pionnier des SGBD personnalisables, voire même l'origine de la *programmation orientée caractéristiques* (FOP) abordée un peu plus bas. La complexité de ce générateur et son non-support de la programmation orientée objet ont beaucoup réduit son usage. Pour pallier à ces lacunes, d'autres approches plus approfondies en termes de personnalisation ont été associées à celles-ci, notamment les approches de la programmation orientées aspects (POA). Ces dernières gèrent mieux la modularisation des fonctionnalités transversales, difficilement encapsulées dans des composants. Tesanovic et al. [181] ont effectivement exploité la POA dans la configuration des SGBD par la modularisation des fonctionnalités transversales dans des aspects.
- Les préprocesseurs constituent une autre approche de développement de SGBD configurables qui est exempte des lacunes précédentes. *Berkeley* [25] (SGBD embarqué d'Oracle), est un exemple de SGBD codé en langage C en utilisant les directives *#ifdef*. Le côté négatif des préprocesseurs réside cependant dans leur effet polluant du code source, mais surtout la maintenance et l'évolution difficiles du logiciel.
- Les approches basées sur la programmation orientée caractéristiques (Feature-oriented programming alias FOP) représentent également des approches de personnalisation de logiciel assez similaire à la POA avec, comme atout clé, la modularisation des caractéristiques d'un logiciel. Elles permettent de générer différentes applications en composant ces caractéristiques, aboutissant ainsi à des lignes de produits d'applications similaires. POA et FOP sont conceptuellement différentes et certaines études avancent que les techniques collaboratives, dont la FOP, peuvent être préférées pour le développement des logiciels hautement personnalisables [151]. En pratique, les besoins fonctionnels d'un SGBD sont représentés par des caractéristiques pouvant être implémentées de façon modulaire. Rosenmüller et al. [151] ont développé une ligne de produits de logiciels embarqués de gestion de données, qui répond aux exigences spéciales de diverses applications, génère, en peu de temps, différentes variantes "sur mesure" de SGBD et réduit l'utilisation de ressources en n'incluant que les fonctionnalités requises, sans toucher à la performance. Ces travaux ont permis de réduire de 50% la taille du code source d'une variante basique de *Berkeley* et d'augmenter sa performance de

25. `http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html`

16%. Cela tout en offrant un haut niveau de personnalisation, qui a même concerné de petits composants du SGBD remanié.

4.2.2 Moteur de requêtes

Afin d'éviter les traditionnels moteurs de requêtes alourdis de fonctionnalités, Soffner et al. [172] ont proposé des optimiseurs de requêtes offrant des solutions (génération des plans d'exécution des requêtes et exécution) pour différents contextes et types de systèmes. Dans un cadre plus général, Rosenmüller et al. [152] ont montré comment SQL peut être décomposé afin d'en créer une famille extensible de dialectes spéciaux, une sorte de SQL à la carte. En effet, ses extensions actuelles pour des domaines spéciaux (par exemple, les réseaux de capteurs ou traitement de flux) sont plus ou moins indépendants du standard, en outre le SGBD sous-jacent ne supporte qu'une partie de ce standard. Personnaliser SQL requiert la personnalisation de ses différents éléments à savoir la syntaxe du langage, le parseur, l'optimiseur, voire même le système du stockage sous-jacent. Pour un scénario particulier, le développeur peut décider des fonctionnalités nécessaires du langage, et choisir par conséquent, le SGBD adéquat qu'il soit standard ou fabriqué sur mesure.

4.2.3 Modèle de stockage

Flexs [190] est un langage déclaratif de description des caractéristiques macroscopiques des modèles physiques de données. Il permet aux utilisateurs de configurer un SGBD pour utiliser un modèle précis de stockage physique de données, ce qui fait de lui un bon exemple de gestion explicite de variabilité de déploiement de données, un élément de la conception physique.

4.3 Travaux existants portant sur la phase conceptuelle

Il existe peu de travaux sur l'adaptation du schéma conceptuel de BD dans des lignes de produits ou des concepts similaires. En effet, un outil client de BD "sur mesure" nécessite aussi bien un schéma conceptuel "sur mesure", afin d'obtenir un SI consistant et valide.

Dans cette optique, Siegmund et al. [166] se sont intéressés à l'intégration des vues afin de générer un schéma de BD global et consistant à partir de différents schémas locaux. Ils rendent possible la séparation d'aspects dès le niveau conceptuel, en décomposant le schéma en caractéristiques (FOP). Ils ont proposé un plugin *eclipse* pour annoter les éléments appartenant à chaque caractéristique. Ce travail a été étendu pour décrire une approche globale de modélisation et génération d'un schéma de BD et des programmes client. Cependant, ils n'ont pas pris en compte l'impact du contexte sur l'applicabilité des caractéristiques. Dans cette perspective, Mori et al. [127] ont développé une approche pour l'adaptation du schéma conceptuel à un contexte donné. Entretemps, Zaid et al. [3] ont proposé une approche basée sur les LdP pour

modéliser la variabilité des modèles des données en utilisant le modèle E/A et la persistance comme une caractéristique. Quant à Khedri et al. [101], ils ont proposé une nouvelle méthode de modélisation basée sur les LdP et la programmation orientée *delta*, destinée à gérer la variabilité présente dans les modèles conceptuels de BD. Le schéma conceptuel est ainsi construit en utilisant un script SQL de définition de données pour décrire les modules noyaux et "delta". Le module de base contient les caractéristiques obligatoires de la famille de produits, et les modules "delta" en contiennent les alternatives ou optionnelles. Avant la génération automatique du modèle de données correspondant à une configuration valide de produit, les inconsistances des modules "delta" doivent être identifiées en prenant en compte les contraintes d'intégrité. Plus récemment, Humblet at al. [90] ont étendu l'outil CASE de conception de BD *DB-Main* [149], en développant le plugin *SVL*. Ce dernier facilite aux concepteurs la modélisation FOP, sa correspondance aux éléments du schéma de BD, et le développement d'un nouveau schéma de BD à partir des caractéristiques sélectionnées.

5 Bilan et discussion

La figure 2.6 illustre la répartition des travaux de gestion de variabilité; présentés dans ce chapitre; sur le cycle de conception de BD. Cela montre le caractère mono-phase de ces travaux. D'une manière générale, nous avons distingué trois axes de gestion de variabilité : aide à la décision, réutilisation et gestion explicite de variabilité (qui réunit l'aide à la décision, la réutilisation et ajoute la configurabilité). Une comparaison entre ces axes est résumée dans le tableau 2.7. Par *implicite*, nous voulons dire que ces travaux n'étaient pas initialement conçus pour la gestion de la variabilité, et donc la traitent d'une manière partielle. En effet, les travaux orientés *aide à la décision* offrent une gestion implicite avec des outils qui aident l'administrateur dans la sélection des choix de conception. Ils n'offrent cependant quasiment pas de configurabilité car il s'agit d'outils spécifiques à un problème très précis. En outre, nous distinguons deux catégories concernant les travaux orientés *réutilisation*. La première catégorie couvre l'aspect de la standardisation via les design patterns. Ces derniers offrent une approche générique de réutilisation, qui limite la configurabilité, mais qui opère sur un niveau d'abstraction élevé. La deuxième catégorie couvre l'aspect de l'automatisation représenté par les outils CASE. Ces derniers offre une réutilisation spécifique au domaine, avec un degré moyen de configurabilité et une gestion implicite de variabilité. Les travaux orientés *gestion explicite de variabilité* semblent les plus

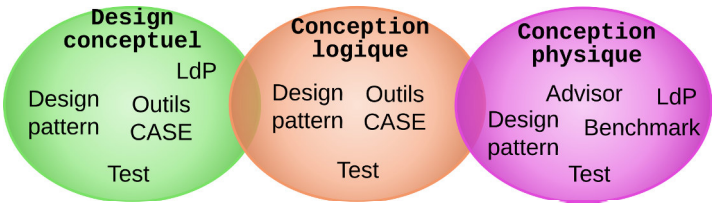


FIGURE 2.6 – Répartition des travaux de gestion de variabilité sur le cycle de conception de BD

Solution	Type de réutilisation	Contexte de réutilisation	Configu-rabilité	Niveau abstraction	Gestion de variabilité	Type
Test, advisor, benchmark	- (aide à la décision)	-	~	-	implicite	outil
Design patterns	standardisation	générique	~	++	implicite	approche
Outils CASE	automatisation	spécifique à l'application	+	~	implicite	outil
LdP	axée sur la variabilité	spécifique au domaine	++	+	explicite	méthodo-logie

TABLE 2.7 – Comparaison d'approches de gestion de variabilité de la conception de BD

prometteurs. Il s'appuient sur une méthodologie de conception qui offre des niveaux élevés de configurabilité et d'abstraction, et une réutilisation spécifique au domaine mais plus générique que les outils CASE. En effet, afin de couvrir tout le cycle de conception de BD, on est obligé de passer par une méthodologie.

Nous avons ensuite mené une revue de littérature concernant les travaux explorant la gestion explicite de variabilité dans le processus de conception des *BD*, dont un résumé est donné dans le tableau 2.8. Malgré l'importante présence de la variabilité, les constats suivants ont été établis.

Étape	Cible	Entrée	Technique
Design conceptuel	Schéma conceptuel	Schéma global	LdP [101, 166]
Conception logique	Langage de requêtes	Exigences d'un schéma logique	LdP [152]
Conception physique	Optimiseur de requêtes	Exigences d'un schéma physique	LdP [172]
	modèle de déploiement	Exigences d'un modèle de stockage	Grammaire [190]
	SGBD	Exigence d'un type de BD	Composants[73],FOP[17] Preprocesseurs,AOP[181],LdP[151]
L'ensemble du cycle	Conception de BD	Exigences d'une application BD	Notre LdP

TABLE 2.8 – Classification des travaux orientés *gestion explicite de variabilité* de conception de BD

1. La variabilité est traitée d'une manière disjointe (mono-phase) pour quelques parties de certaines phases.
2. La phase physique, et plus particulièrement le SGBD, a bénéficié d'une attention beaucoup plus soutenue que les autres phases.
3. Peu de travaux se sont intéressés à la phase conceptuelle et encore moins à la phase logique. Pour autant que nous sachions, celle-ci n'a pas fait l'objet d'une étude de variabilité, sauf si on lui attribue les rares travaux portant sur les langages de requêtes, un module qui se trouve à cheval entre la phase logique et physique. Les travaux de Rosenmüller et al. [152] en sont un exemple.
4. L'absence d'une approche holistique, qui traite la variabilité du processus de conception dans son ensemble, pour **(i)** assurer la cohérence et prendre en compte les interdépendances qui lient les différentes phases, **(ii)** présenter au concepteur différents choix disponibles afin d'en choisir le meilleur, contrairement à l'approche classique où il se fiait à son expérience. Cela risquait de le faire passer à côté de ce choix, surtout dans un contexte actuel aussi variable, **(iii)** unifier le processus de conception et ainsi éviter de réinventer des solutions qui existent déjà et **(vi)** automatiser autant que possible ce dernier, afin de réduire les différents coûts.

Positionnement de nos contributions

Dans ce mémoire, nous nous sommes donnés pour but de remédier aux lacunes de l'existant, en proposant en premier lieu une méthodologie de conception orientée gestion de variabilité. Pour ce faire, nous adoptons la technique des lignes de produits logiciels (LdP). Nous appliquons cette technique, réputée pour être efficace dans la maîtrise de la variabilité des BD, sur l'ensemble du processus de conception. Ce caractère holistique permet de: **(i)** considérer les interdépendances entre les phases, **(ii)** offrir une vision globale et "exhaustive" au concepteur, et **(iii)** augmenter l'automatisation du processus. (Chapitre 3).

Le résultat souhaitable de cette voie de recherche serait une boîte noire configurable, alimentée par un ensemble d'entrées (notamment requêtes, statistiques et besoins utilisateurs/schéma), pour produire en sortie un script correspondant à une *BD* prête à être déployée dans un environnement prédéfini. L'énorme étendue de l'implémentation (configuration et dérivation des produits) nous impose donc de procéder par étapes dans la réalisation de cette vision, en consacrant cette thèse à l'étude d'un cas précis. L'implémentation finale de notre LdP (problème global) revient alors à généraliser les résultats de différents cas d'études. Nous visons à travers cette étude de cas de :

☞ montrer l'importance de la variabilité au niveau de la phase de conception logique, jusque-là sous-étudiée ; (Chapitres 4 et 5)

☞ inculquer la culture de la variabilité aux concepteurs en leur offrant une large panoplie de choix de modèles logiques ; (Chapitres 4)

- ☞ exploiter cette variabilité définie au niveau de la phase logique pour mieux optimiser logiquement les requêtes. Dans notre étude, l'optimisation logique concerne les tâches conventionnelles de l'optimiseur de requête sans recours aux structures d'optimisation souvent sélectionnées dans la phase physique. En effet, quand une requête correcte est soumise par un utilisateur, elle est convertie en une représentation interne sous forme d'arbre : plan d'exécution. Celui-ci peut avoir plusieurs représentations équivalentes, générant les mêmes résultats finaux, mais avec des performances différentes. L'optimisation logique vise à obtenir le meilleur plan sans recourir aux structures d'optimisation physiques ; (Chapitre 4)
- ☞ passer de la vision mono-phase de la variabilité à la visions multi-phase en associant la phase physique à la phase logique. Cette association est motivée par l'interdépendance forte entre les phases ; (Chapitres 3 et 5)
- ☞ s'éloigner de l'optimisation physique traditionnelle qui suppose toujours un schéma logique figé, à une optimisation prenant en compte la variabilité des schémas logiques. Dans cette étude, l'optimisation physique est représentée par le processus de sélection des vues matérialisées, une structure d'optimisation de requêtes (Chapitres 5) ;
- ☞ l'évaluation des deux optimisations (logique et physique) est effectuée à l'aide de l'utilisation de modèles de coût mathématiques, estimant les métriques de besoins non fonctionnels, à savoir la performance des requêtes, la consommation d'énergie et l'espace de stockage. Nous tenons à souligner que la gestion de variabilité proposée dans ce manuscrit est supportée par une méthodologie compréhensive et formelle à travers : la modélisation du LdP, la formalisation de la génération des modèles logiques ainsi que l'utilisation des modèles de coûts mathématiques pour les évaluations. Ces derniers sont validées avec des expérimentations empiriques ;
- ☞ par ailleurs, nous soulignons notre intérêt porté aux solutions logicielles écologiques (*Green IT*). En effet, les spécialistes des technologies *BD* ont longtemps accordé la plus grande attention à l'optimisation du temps d'exécution, suivi par le stockage en ignorant l'énergie. Le contexte actuel oblige, plus que jamais, ces spécialistes à intégrer la dimension énergétique, car un SGBD est l'un des principaux consommateurs d'énergie dans les centres de données.

Deuxième partie

Approche par étapes pour la gestion de la variabilité au sein de la conception des bases de données

Chapitre 3

La conception des bases de données comme une ligne de produits

Sommaire

1 Introduction 85

2 Analyse de la variabilité de la conception des BD 86

2.1 Framework générique des LdP 86

2.2 Phase conceptuelle 87

2.3 Phase logique 88

2.4 Phase physique 91

2.5 Identification des dépendances entre les variantes 92

3 Implémentation de la variabilité de la conception des BD 93

3.1 Implémentation des features : outillage 94

3.2 Configuration des BD 95

3.3 Dérivation des BD : modélisation du problème 97

4 Approche par étapes pour la gestion de la variabilité 102

4.1 La gestion de la variabilité de la conception logique 103

4.2 Choix de feature pour incarner la variabilité au sein de la conception logique 104

4.3 Choix des features pour refléter l'impact de la variabilité logique 111

5 Conclusion 116

1 Introduction

Dans le chapitre précédent, nous avons montré comment la variabilité a été traitée dans chaque phase de conception de BD. Pour aller plus loin dans notre réflexion, nous avons identifié la nécessité d'avoir une méthodologie compréhensive et complète d'intégration de la gestion de la variabilité dans ce processus de conception. Dans ce chapitre, nous présentons notre première contribution dédiée à la définition de cette méthodologie de conception orientée gestion de variabilité. Pour ce faire, nous adoptons la technique des lignes de produits logiciels, réputée pour être efficace dans la maîtrise de la variabilité des BD (entre autres) [36], et ce sur l'*ensemble* du processus de conception. Ce caractère holistique permet de (i) considérer les interdépendances entre les phases, (ii) offrir une vision globale et exhaustive au concepteur, et (iii) augmenter l'automatisation du processus. De plus, contrairement à la vision classique [66] qui, entre autres, fixe la sélection du SGBD avant la conception logique, les LdP offrent plus d'indépendance en rendant possible le report de ce choix à un point ultérieur.

Le framework que nous proposons est dédié à l'adoption des LdP pour la conception des BD. Il permet ainsi la modélisation d'un grand ensemble de BD comme une ligne de produits en vue de réaliser une analyse orientée variabilité. Malgré le fait que chaque BD est différente, par ses données, ses requêtes mais aussi par sa configuration, elles passent toutes par le même processus de conception qui définit cette configuration. Elles y partagent donc un ensemble de caractéristiques et se distinguent les unes des autres par un autre ensemble de caractéristiques. Ainsi, l'analyse doit couvrir toutes les phases de ce cycle de conception.

La gestion de variabilité ne se limite pas seulement à l'analyse (but *descriptif*), mais doit s'étendre à son implémentation (configuration et dérivation des produits), qui est une activité importante dans une approche supportant les LdP. Le résultat idéal de cette voie de recherche serait une boîte noire configurable, alimentée par un ensemble d'entrées (notamment requêtes, statistiques et besoins utilisateurs/schéma) et produisant en sortie un script correspondant à une BD prête à être déployée dans un environnement prédéfini. Notons qu'il existe deux scénarios pour l'utilisation de cette LdP. Le premier scénario concerne la conception d'une nouvelle BD/ED ce qui est caractérisé par l'absence des données (pouvant être remplacées par des statistiques, ou des BD/ED mocks [26]) et l'absence des requêtes pouvant être déduites à partir des besoins [21, 102, 94] (*principe de Pareto*). Le deuxième scénario concerne une BD/ED déjà déployée (en production). Les données ainsi que les requêtes réelles sont alors disponibles.

L'envergure de ce problème (configuration et dérivation des produits) nous impose de restreindre notre étude à un cas précis. Ce cas doit cependant permettre de montrer : (i) comment cette dérivation de BD (produit logiciel) peut se réaliser, (ii) l'importance de la gestion *multi-phase* de la variabilité dans la conception des BD, qui a l'avantage de prendre en compte les interdépendances liant les phases (par opposition à la mono-phase de l'état de l'art), et qui par

26. Le "mock" d'une BD est un objet qui la simule, en renvoyant des données prédéterminées quelle que soit la requête qu'on lui transmet.

la même occasion (iii) doit jeter la lumière sur l'importance de la variabilité de la conception logique, jusque là sous-étudiée.

Dans ce chapitre, nous commençons d'abord par l'analyse de la variabilité (ingénierie du domaine) de la conception des BD. Cette analyse proposée dans la section 2, est faite en deux étapes : modélisation de la LdP et identification des dépendances. Nous abordons par la suite, dans la section 3, l'implémentation de cette variabilité dans un cadre général et le framework associé. Vu l'étendue de ce dernier, la section 4 est consacrée à la présentation de notre cas d'étude, portant sur la variabilité de la conception logique, qui montre la gestion de variabilité multi-phase. Nous concluons dans la section 5.

2 Analyse de la variabilité de la conception des BD

L'adoption des LdP commence d'abord par une description de la variabilité, souvent réalisée à l'aide des *modèles des features* (FM). Ces derniers définissent les principaux objectifs du processus d'ingénierie du domaine des LdP [142], à savoir : le domaine, la famille de produits à dériver, les points communs et les points de variabilité.

Dans nos travaux, nous ciblons le domaine de la conception des BD : l'étape clé de leur processus de développement et dont l'étude de variabilité impacte tout le cycle (voir paragraphe 1.2.3). Les produits finaux à dériver représentent, dès lors, différents types de BD conçues en adéquation avec les besoins fonctionnels et non fonctionnels des utilisateurs, et prêtes à être déployées dans un environnement prédéfini. Par ailleurs, l'étude de la variabilité doit couvrir tout le cycle de conception, composé principalement des trois étapes : conceptuelle, logique et physique (voir section 1.4), ce qui donne un FM assez grand. Pour cette raison, nous procédons par étape en proposant un FM pour chaque étape. L'union de l'ensemble des modèles constitue le modèle global.

Il est important de noter que notre modélisation est loin d'être unique ou exhaustif. Cependant, nous estimons avoir défini un niveau de granularité de base, suffisant pour atteindre nos objectifs tout en donnant un aperçu global sur le processus de conception. De plus, nous insistons sur l'aspect *collaboratif* du framework, qui permet aux utilisateurs d'enrichir et d'étendre les modèles.

2.1 Framework générique des LdP

Notre LdP est modélisée avec des modèles de features, définis par le couple:

$$\mathcal{FM} : \langle \mathcal{F}, \mathcal{CI} \rangle, \text{ tel que :}$$

$\mathcal{F} = \{f_1, ..., f_n\}$ est un ensemble de features, et $\mathcal{CI}$ un ensemble de contraintes d'intégrité (formules propositionnelles expliquées plus loin) définies sur l'ensemble des features (caractéris-

tiques) $\mathcal{F}$, et ayant pour format :

$$\mathcal{CI}_i \rightarrow \langle f_1 \rangle \; exclut|exige|avantage|desavantage \; \langle f_2 \rangle$$

Une feature peut représenter du code, un test, un script, une architecture, une documentation, des exigences, etc. Elle peut aussi être interactive, nécessitant une entrée ou une intervention de l'utilisateur. Elle est définie par le triplet :

$$\forall f \in \mathcal{F}, f : \langle t, \mathcal{FG}, \mathcal{I} \rangle \text{ tel que :}$$

1- t est le type de feature parmi obligatoire, optionnel ou indéfini si cette dernière appartient à un groupe alternatif ou exclusif $\mathcal{FG}_{or}/\mathcal{FG}_{xor}$.

2- chaque feature peut être parente de plusieurs groupes de features $\mathcal{FG}$ (variantes, enfants) :

$$\mathcal{FG} \rightarrow \mathcal{FG}_{and}|\mathcal{FG}_{or}|\mathcal{FG}_{xor}|f|\varnothing, \text{ avec :}$$

- $\mathcal{FG}_{and} \rightarrow (\langle f_1 \rangle \wedge \langle f_2 \rangle)^+$: ce que signifie que tous les features enfants de type "obligatoire" doivent être sélectionnées à chaque fois que ce parent est sélectionné.

- $\mathcal{FG}_{or} \rightarrow (\langle f_1 \rangle \vee \langle f_2 \rangle)^+$: signifie qu'au moins une feature doit être sélectionnée (groupe alternatif).

- $\mathcal{FG}_{xor} \rightarrow (\langle f_1 \rangle \; xor \; \langle f_2 \rangle)^+$: implique la sélection d'une et d'une seule feature parmi les enfants (groupe exclusif).

- Si $\mathcal{FG} = \varnothing$, alors f est une variante (noeud feuille), sinon elle se qualifie de *point de variation* en sus.

3- $\mathcal{I}$ est l'éventuelle entrée requise pour l'implémentation de la feature. Il peut s'agir de requêtes, statistiques, schémas, valeurs de paramètres, et ainsi de suite.

Nous utilisons ce framework pour modéliser la variabilité de chacune des principales phases du cycle de conception des BD.

2.2 Phase conceptuelle

Le design conceptuel consiste en l'élaboration d'une représentation structurée et abstraite de données, destinée aux utilisateurs. En d'autres termes, cette représentation n'inclut pas de détails sur la manière dont les données sont stockées ou les opérations sont implémentées. Plusieurs langages de modélisation peuvent être utilisés, notamment : Entité/Association (EA), Unified Modeling Language (UML), Express, etc. Il est important de noter que l'expressivité et la qualité du modèle conceptuel dépend fortement des compétences du concepteur, plutôt que du formalisme utilisé. Ainsi, cette phase ne peut s'automatiser totalement. En effet, la variabilité porte sur les formalismes utilisés (voir figure 3.1), ainsi que l'appartenance (locale ou globale)

des entités du schéma [100, 166] (voir figure 3.2) mais pas le contenu. On imagine mal qu'un même concepteur puisse réaliser un modèle conceptuel pour la même application, en utilisant plusieurs formalismes (E/A, UML, express) pour ensuite les comparer et choisir le meilleur en termes de contenu. En d'autres termes, la vision du domaine ne va pas évoluer en changeant de formalisme. Le concepteur peut cependant privilégier un formalisme par rapport à un autre pour l'expressivité de ce dernier, son aisance d'utilisation, ou simplement pour une question d'affinité. Aussi, après avoir terminé le design conceptuel, il peut définir la portée des entités, c'est à dire en assigner une partie à des applications spécifiques, et définir une autre partie comme centrale, qui doit être présente quelle que soit l'application [100, 166]. Cela représente le deuxième type de variabilité qui peut concerner cette phase, illustré par la figure 3.2 où les entités du domaine de la scolarité sont annotées selon leur appartenance aux applications dérivées (offre de formation, répartition des classes, lectures recommandées, etc.). L'aperçu du FM présenté dans la figure 3.1 est formalisé comme une instance du framework des LdP décrit précédemment, comme suit :

Exemple 1 (*instanciation du framework*)

$\mathcal{FM}_{\mathcal{DC}} : \langle \mathcal{F}_{\mathcal{DC}}, \mathcal{CI}_{\mathcal{DC}} \rangle|$
- $\mathcal{F}_{\mathcal{DC}} : \langle obligatoire, \mathcal{FG}_{and}, \varnothing \rangle|$
  - *t est indéfini car* $f_{e/a}$ *appartient à un* $\mathcal{FG}_{or}$.
  - $\mathcal{FG}_{and} = \{f_{Formalisme}, f_{entite}\}|$
    - $f_{Entite} : \langle optionnel, \varnothing, entité \rangle$
    - $f_{Formalisme} : \langle obligatoire, \mathcal{FG}_{or}, cahier\ des\ charges \rangle|$
    - $\mathcal{FG}_{or} = f_{e/a} \vee f_{uml} \vee f_{express} \vee f_{semantique}|$
      - $f_{er} : \langle indéfini, \varnothing, \varnothing \rangle$, *alors* $f_{e/A}$ *est une variante mais pas un point de variation.*
- $\mathcal{CI}_{\mathcal{DC}}$ *est décrit un peu plus bas.*

2.3 Phase logique

La conception logique est une phase de transformation (mapping) du modèle conceptuel en un modèle logique lié à l'implémentation de BD dans un SGBD spécifique. Cette phase peut ainsi être automatisée par les outils de conception de BD (outils CASE). La variabilité au sein de

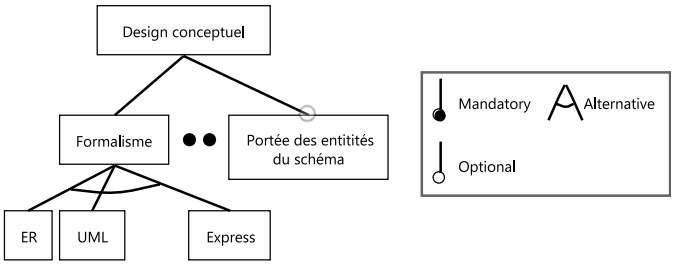


FIGURE 3.1 – Extrait du modèle des features du design conceptuel des BD

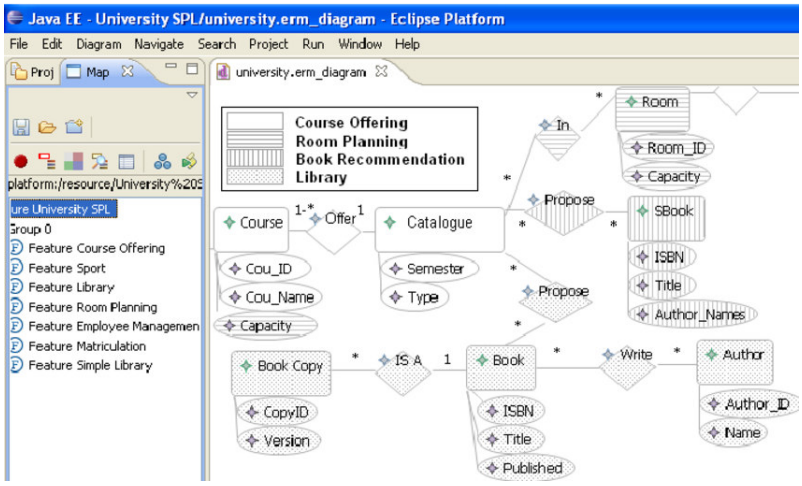


FIGURE 3.2 – Exemple de variabilité de la portée des entités du schéma conceptuel [166]

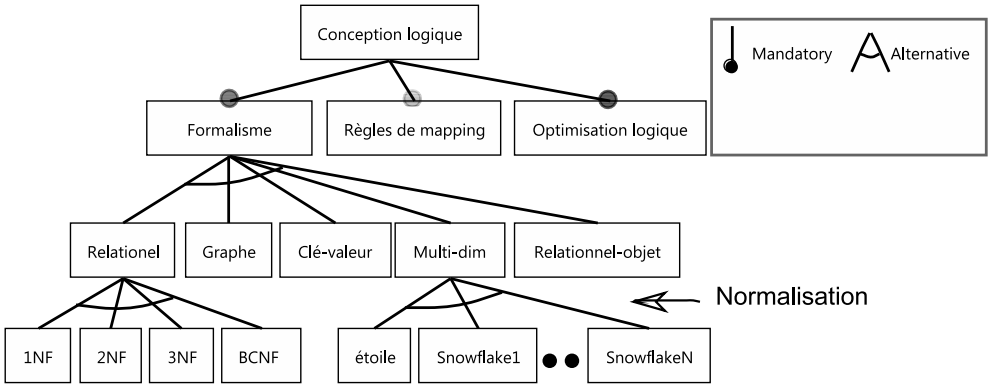


FIGURE 3.3 – Extrait n°1 du modèle des features de la conception logique des BD

cette phase, peut concerner quatre features (points de variation) : le formalisme, les mappings, la normalisation et l'optimisation logique.

— Modèle de données : divers formalismes existent pour représenter logiquement les données, qui peuvent être classés dans trois catégories : (1) modèles orientés enregistrements comme le relationnel, multidimensionnel et réseau, (ii) orientés-objet comme l'objet et le relationnel-objet, et (iii) orientés NoSQL comme les modèles clé-valeur, graphes, documents ou colonnes (voir figure 3.3).

— Les règles de mapping : pendant la transformation du modèle conceptuel en modèle logique, certains concepts possèdent plusieurs façons d'être transformés. Par exemple, la relation de généralisation/spécialisation et ses différentes contraintes ainsi que les constructeurs d'union ont plusieurs transformations possibles. La figure 3.4 illustre la variabilité de ce point. L'exemple 2 montre les options A et B du mapping. Notons que chaque option a ses avantages et ses inconvénients.

— La normalisation des données : souvent rencontrée dans les BD relationnelles, elle consti-

tue elle aussi, un point de variation, puisque le choix d'une forme normale n'est pas évident car il influence la performance de la BD. La normalisation donne lieu à différentes formes normales (1FN, 2FN, 3FN, FNBC) dans le contexte OLTP, et à différents schémas multidimensionnels dans l'OLAP [66].

— L'optimisation logique : une fois que le schéma logique est implémenté sur la BD, des *optimisations logiques* sont d'abord définies, en exploitant les propriétés offertes par le langage de requêtes utilisées (à l'exemple de la commutativité de la sélection, la distribution de la jointure sur l'union, etc.). Cela concerne les tâches conventionnelles de l'optimiseur de requête sans recours aux structures d'optimisation souvent sélectionnées dans la phase physique. En effet, quand une requête correcte est soumise par un utilisateur, elle est convertie en une représentation interne sous forme d'arbre : plan d'exécution. Celui-ci peut avoir plusieurs représentations équivalentes, générant les mêmes résultats finaux, mais avec des performances différentes [71, 72]. La variabilité de ce point concerne les techniques utilisées (règles algébriques et/ou MdC) afin d'obtenir le meilleur plan avant de recourir aux structures d'optimisation physiques et est expliqué dans la section 3.4.3.

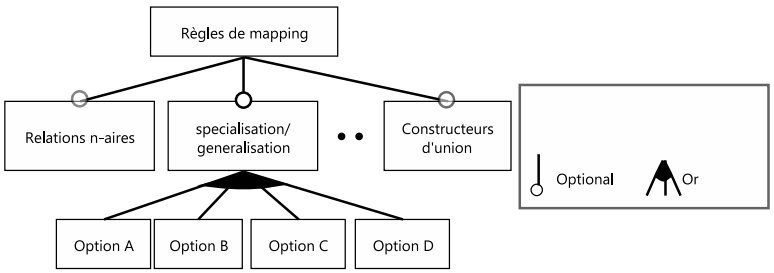


FIGURE 3.4 – Extrait n°2 du modèle des features de la conception logique des BD

Exemple 2 (*options d'interprétation de l'héritage dans la logique relationnelle*)

Prenons l'entité "Employé" qui possède les attributs suivants : nom, ssn, date_naissance, adresse, type_job, et qui est spécialisée en deux sous-entités comme suit : Ingénieur avec type_ing comme attribut, Technicien avec tgrade comme attribut. Cet héritage peut être implémenté dans le modèle relationnel suivant différentes options [66] par exemple :

***Option A** : en utilisant les trois tables suivantes :*
Employé(ssn, nom, date_naissance, adresse, type_job),
Ingénieur(ssn,type_ing),
Technicien(ssn,tgrade).
***Option B** : en utilisant une seule table :*
Employé(ssn, nom, date_naissance, adresse, type_job, type_ing, tgrade).

Exemple 3 (*instanciation du framework*)

$\mathcal{FM}_{\mathcal{CL}} : \langle \mathcal{F}_{\mathcal{CL}}, \mathcal{CI}_{\mathcal{CL}} \rangle |$
-$\mathcal{F}_{\mathcal{CL}} : \langle$ *indéfini*, $\mathcal{FG}_{xor}, \mathcal{DC} \rangle | \mathcal{DC}$ *est le modèle conceptuel issu de la phase précédente.*

-$\mathcal{FG}_{xor}$: {hiérarchique, ..., objet-relationnel}.
-$\mathcal{CI}_{\mathcal{CL}}$ est décrit un peu plus bas.

2.4 Phase physique

La dernière étape est la conception physique, pendant laquelle sont spécifiés des structures internes de stockage, des index, des chemins d'accès, des paramètres physiques et l'organisation des fichiers de la BD. En parallèle de ces activités, les programmes opérationnels sont implémentés. La variabilité peut ainsi concerner différents niveaux comme illustré dans la figure 3.5 et détaillé ci-dessous.

— Plateforme de déploiement : fait la distinction entre le traitement local et distribué des données. Cela définit entre autre, le nombre de sites abritant les données distribuées et décrit les politiques d'exécution de requêtes et de distribution de données. Il s'agit de déploiement central, parallèle et distribué. La plateforme centralisée est caractérisée par le fait que la BD est stockée dans un seul support de stockage. La distribuée implique que les données sont stockées dans plusieurs supports de stockage à travers divers sites connectés via un réseau informatique. Lorsque les données sont identifiées dans différents sites de la plateforme distribuée on parle de l'architecture répliquée. Quant à la parallèle, elle se distingue de la distribuée par le fait que chaque support de stockage possèdent ses propres CPU.

— Architecture de déploiement (éventuelle) : définit comment stocker les autres éléments que les données dans une seule BD. Il s'agit notamment des BD sémantiques destinées à stocker les données en même temps que leur sémantique (ontologie), voire même le modèle d'ontologie utilisé (PLIB, RDFS, etc.). Plusieurs architectures sont proposées dans la littérature (I, II, III) [175].

— Le "layout" de stockage désigne la façon dont les données sont physiquement organisées. À titre d'exemple, le stockage traditionnel qualifié d'horizontal (*Row-store*) consiste à stocker les données par lignes. Plus récemment le stockage vertical (*Column-store*) pro-

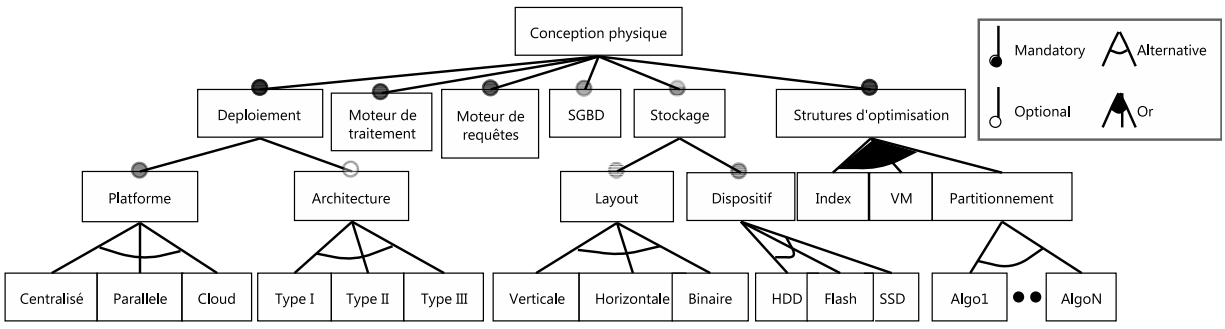


FIGURE 3.5 – Extrait du modèle des features de la conception physique des BD

pose le stockage des données par colonne. D'autres modèle de stockage existent tels que le binaire, stockage en colonne, en grille, etc.

— Structures d'optimisation, également désignées sous le nom de structures de la conception physiques : sont des outils et des techniques conçus pour optimiser le traitement des requêtes. Nous pouvons citer par exemple : les index, les vues matérialisées, l'ordonnancement de requêtes, le partitionnement, la gestion de buffer, et ainsi de suite. De plus, chaque structure peut être implémentée en utilisant différents algorithmes (voir section 2.2.3.1).

— Dispositif de stockage : détermine la technologie utilisée pour le stockage des données comme les disques durs (HDD), disques électroniques (SSD), disques en RAID, mémoires centrales, mémoires flash, etc. Il s'agit d'une fonctionnalité bas niveau (matérielle).

— Moteur de requêtes : une feature à cheval entre la conception logique et physique qui concerne le traitement des requêtes.

— Moteur de traitement : désigne les composants matériels qui exécutent les différentes opérations élémentaires, comme CPU, GPU et FPGA. Il s'agit également d'une fonctionnalité bas niveau (matérielle).

Exemple 4 (*instanciation du framework*)

$\mathcal{FM}_{\mathcal{CP}} : \langle \mathcal{F}_{\mathcal{CP}}, \mathcal{CI}_{\mathcal{CP}} \rangle |$
- $\mathcal{F}_{\mathcal{CP}} : \langle$ *indéfini*, $\mathcal{FG}_{and}, \mathcal{CL} \rangle |\mathcal{CL}$ *est le modèle logique issu de la phase précédente.*
  - $\mathcal{FG}_{and} = \{f_{deploiement}, ..., f_{structures-optimisation}\}$.
    - $f_{deploiement} : \langle obligatoire, \mathcal{FG}_{and}, \mathcal{CL} \rangle$
      - $\mathcal{FG}_{and} = f_{platform}, f_{architecture}$
      - $f_{platform} = \langle obligatoire, \mathcal{FG}_{xor}, I \rangle$
        - $\mathcal{FG}_{xor} =$ {*Centralisé, Parallèle, Cloud*}
      - $f_{architecture} = \langle optionnel, \mathcal{FG}_{xor}, I \rangle$
        - $\mathcal{FG}_{xor} =$ {*Type I, Type II, Type III*}

2.5 Identification des dépendances entre les variantes

La chaîne que forment les étapes du processus de conception de BD montre une certaine interdépendance entre ces dernières, et transitivement entre leur variabilité. L'exemple le plus trivial est l'ordre chronologique qui impose au design conceptuel d'être réalisé avant le logique, et ce dernier avant le physique. Plusieurs autres dépendances existent, liées notamment au déploiement physique et au design. Nous notons que certaines dépendances concernent une seule phase et que d'autres lient différentes phases entre-elles. Elles peuvent être flexibles ou strictes. C'est à partir de ces dépendances que l'architecture du LdP, qui détermine la composition des produits finaux, se développe.

Notre framework prend en considération deux types de dépendances : (i) strictes exprimées à l'aide des relations *exige* et *exclut*, et (ii) flexibles qui interviennent en vue de conseiller et assister les concepteurs pendant le processus de configuration via le couple de relations *avantage/désavantage*. Elles sont le fruit de l'expérience acquise par les concepteurs, et peuvent être enrichies grâce à leur collaboration.

Ces dépendances sont primordiales au processus de configuration, parce qu'elles définissent l'architecture du LdP, les produits valides en éliminant les inconsistances, et donc réduisent le nombre de combinaisons (produits) possibles. Dans l'exemple 5, nous présentons quelques dépendances qui régissent notre LdP :

Exemple 5

-$\mathcal{CI}_{\mathcal{DC}}$: ⟨*sémantique* ***exige*** *architecture déploiement*⟩
-$\mathcal{CI}_{\mathcal{CL}}$: ⟨*multidimensionnel* ***avantage*** *vertical*⟩
- $\mathcal{CI}_{\mathcal{CL}}$: ⟨ *optimisation logique* **exige** *formalisme logique*⟩
-$\mathcal{CI}_{\mathcal{CP}}$: ⟨*flash* ***désavantage*** *vue matérialisée*⟩
-$\mathcal{CI}_{\mathcal{CP}}$: ⟨*MySQL* ***exclut*** *vue matérialisée*⟩
- $\mathcal{CI}_{\mathcal{CP}}$: ⟨ *structures-optimisation* **exige** *déploiement* et *stockage*⟩
- $\mathcal{CI}_{\mathcal{CL/CP}}$: ⟨*multidimensionnel* **avantage** *vue matérialisée*⟩.

3 Implémentation de la variabilité de la conception des BD

La deuxième étape d'adoption des LdP est la génération des produits finaux de l'application qui passe par le processus d'ingénierie de l'application. Rappelons que les produits finaux de notre LdP sont des BD conçues prêtes à être déployées dans un environnement prédéfini. Le résultat idéal de cette voie de recherche serait une boîte noire configurable, alimentée par un ensemble d'entrées (notamment requêtes, statistiques et schémas), pour produire en sortie un script correspondant à une BD prête à être déployée, qui peut être enrichi via l'ajout des procédures stockées et des triggers avant le déploiement effectif de la BD. À première vue, notre LdP peut être assimilé à un outil CASE de conception de BD, comme *PowerDesigner* [158] par exemple, car ce dernier génère aussi des scripts correspondants à certains choix de conception. Nous notons cependant que *PowerDesigner* et les outils équivalents s'intéressent principalement à l'aspect modélisation. Ils n'offrent pas la prise en compte des différentes variantes ni leur interdépendances, et n'aident pas le concepteur à faire un choix parmi ces variantes qui existent tout au long du cycle de conception. Notre objectif, quant à lui, peut être assimilé à une sorte d'advisor holistique, qui accompagne le concepteur à travers toutes les étapes du processus en lui offrant un haut degré de configurabilité et de réutilisation.

À notre connaissance, il n'existe pas à l'heure actuelle un tel outil permettant de choisir finement la configuration des BD, ou du moins la plus proche de l'environnement technique et non-fonctionnel de l'application BD. Ce choix repose actuellement sur les connaissances des

experts et soulève le problème de l'exhaustivité et de la fiabilité de cette connaissance. Le cadre logiciel présenté dans cette thèse est une solution proposée pour concevoir un outil configurable de conception des BD, qui a pour objectif de gérer sa variabilité, un challenge auquel sont confrontés les concepteurs de nos jours. Nous donnons une ébauche du processus d'ingénierie de l'application en expliquant ses deux étapes de configuration et dérivation, et nous présentons une vue d'ensemble de notre framework.

3.1 Implémentation des features : outillage

La manipulation des LdP a attiré beaucoup d'attention ces dernières années. Plusieurs outils se sont donc développés [116, 61]. Certains se sont intéressés uniquement à la modélisation de la variabilité (ingénierie du domaine), et d'autres plus complets, se sont également intéressés à son implémentation (ingénierie d'application). Le tableau 3.1 donne un aperçu de quelques outils assez connus, conçus pour la gestion de la variabilité basée sur FODA (*Analyse de domaine orientée features*).

Outil	**Développé par**	**Utilisé**	**Publié**
CaptainFeature	Fachhochschule Kaiserlautern Allemagne	Académique	2002
Pure::Variants	Pure-Systems Allemagne	Industriel	2003
FeatureIDE	Université de Magdeburg Allemagne	Académique	2005
RequiLine	Université technique d'Aix-la-Chapelle Allemagne	A&I [27]	2005
XFeature	P&P Software GmbH avec l'école polytechnique fédérale de Zurich Suisse	A&I	2005
MOSKitt	Projet gvCASE Espagne	Académique	2008
Feature Modeling Tool	Université de Valladolid et de Burgos Espagne	Académique	2008
Feature Model DSL	Gunther Lenz et Christoph Wienands	Académique	2008
CVM Tool	Projet européen ATESST Allemagne	A&I	2009

TABLE 3.1 – Quelques outils de gestion de variabilité basés sur FODA [61]

Nous avons choisi l'outil *FeatureIDE* [182], basé sur *Eclipse IDE* car c'est un des plugins open-source les plus complets trouvés dans la littérature [61]. Il prend en charge différents langages d'implémentation des LdP notamment *AspectJ* [103] et *AHEAD* [16]. Nous avons implémenté notre framework avec le compositeur *AHEAD* qui fournit la composition des fichiers de types *Jak*. Ce type étend le langage *Java* avec des mots-clés issus de la programmation orientée *features* (POF).

En premier lieu, les features ainsi que les dépendances sont modélisées en utilisant l'éditeur. *FeatureIDE* crée deux répertoires : *features* qui contient un dossier destiné à l'implémentation

27. A&I : Académique & Industriel

de chaque feature préalablement définie, et *configs* pour sauvegarder les différentes configurations matérialisées par des combinaisons valides de features (voir figure 3.6).

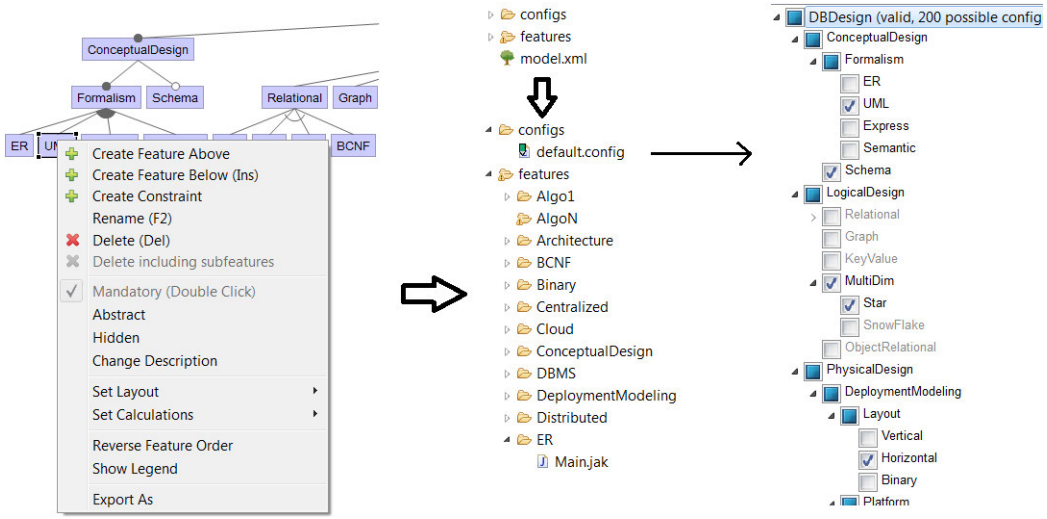


FIGURE 3.6 – Aperçu de l'implémentation de *FeatureIDE* de nos FM de conception des BD

Les différentes features ainsi que les dépendances des FM issues de l'ingénierie du domaine, doivent être implémentées. Vu l'ampleur de la tâche, nous nous concentrons sur un cas d'étude particulier étudié dans la section 3.4. Le reste des features peut être implémenté au fur et à mesure que le framework est utilisé et enrichi par de futurs concepteurs.

3.2 Configuration des BD

Une fois que les features sont implémentées, le concepteur peut configurer la BD désirée, en sélectionnant une combinaison valide de choix. Ces derniers sont représentés sous forme d'un ensemble de cases à cocher (voir figure 3.7), qui peuvent être : (i) mises en gras après la sélection préalable d'autres cases, afin de recommander ces choix (dépendances *avantage/désavantage*), (ii) désactivées en cas de dépendance de type *exclut*, ou (iii) automatiquement sélectionnées en cas de dépendance de type *exige*.

La particularité avantageuse de notre framework (des LdP) est qu'il aide les concepteurs lors de la sélection (configuration), non seulement grâce aux recommandations, mais surtout en utilisant des algorithmes avancés (modèles de coûts) pour l'évaluation de ces choix. Il permet, de plus, d'être le plus exhaustif possible dans la présentation de ces choix au concepteur, et de prendre en compte les éventuelles interdépendances.

Ce mécanisme d'aide à la décision pendant le processus de configuration, porte sur les points de variation restant *open*, dont aucune variante n'a été sélectionnée par le concepteur, car il manque de critères efficaces servant à la comparaison et au choix entre ces variantes. En revanche, il y a des choix plus évidents ou simplement imposés, qui sont entièrement confiés

au concepteur, sans passer par une évaluation algorithmique. Il s'agit, entre autres, du modèle logique de données à utiliser, qui dépend du type des données à manipuler et du type d'application (ses objectifs). En effet, le *relationnel* est plutôt destiné à la gestion des données structurées manipulées par des applications à but transactionnel, le *multidimensionnel relationnel* pour des données toujours structurées mais manipulées en vue d'un traitement décisionnel, l'*objet* pour la modélisation des structures de données complexes, manipulées par les applications *CFAO* [28], le *clé-valeur* et plus généralement le *NoSQL* pour des données non structurées émanant du web, et ainsi de suite. Par conséquent, il incombe au concepteur de détecter les types des données manipulées par l'application, et ainsi de choisir le modèle logique de données adapté. Un autre exemple concerne le choix du SGBD, qui peut être imposé par le cahier des charges, ou par des contraintes financières. Nous constatons alors que l'évaluation par modèle de coût d'un point de variation concerne les feuilles des modèles des features, et où il existe un large nombre de variantes. C'est par exemple le cas du choix des structures d'optimisation à mettre en place. Un modèle de coût permet d'estimer la qualité de chaque solution et ainsi d'identifier les meilleures.

La validité de la configuration établie jusque là, peut ensuite être vérifiée à l'aide de solveurs SAT (par exemple, pour vérifier qu'une caractéristique a été choisie si elle est requise par une autre, ou qu'une caractéristique parent d'une caractéristique sélectionnée a été indiquée). Quant à la configuration (la sélection des variantes) des points de variation restant open, nous allons

28. Conception et Fabrication Assistée par Ordinateur

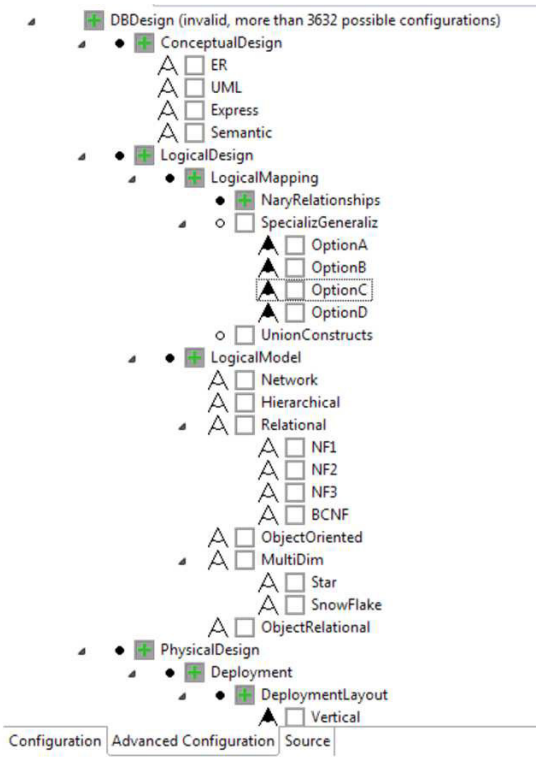


FIGURE 3.7 – Aperçu de la configuration des BD

l'aborder dans la section suivante.

3.3 Dérivation des BD : modélisation du problème

La dérivation classique des LdP consiste à exécuter la configuration établie par le concepteur. L'exécution fait appel à l'implémentation de chaque feature sélectionnée, et intègre l'ensemble suivant l'architecture du LdP, définie grâce aux dépendances.

Cependant, quand le concepteur manque de directives tranchantes vis-à-vis de certains points de variation, ces derniers restent open, ce qui donne lieu à plusieurs configurations de modèles de features pouvant être valides. Dans ce cas, de nombreux types de BD peuvent correspondre à l'application. Afin d'aider le concepteur à affiner cette liste de BD potentielles, traditionnellement fait en se fiant à ses intuitions, il va falloir évaluer ces dernières (axe d'aide à la décision de la gestion de la variabilité). L'évaluation implique un algorithme avancé (voir section 2.2.3.1) et se fait par rapport à un critère donné. Il s'agit des besoins non fonctionnels de l'application, comme par exemple la performance de l'exécution, la consommation de l'énergie, ou l'espace de stockage.

Cependant l'évaluation doit porter sur tout le processus au même temps, pour tenir compte des interdépendances des phases. La sélection des meilleures alternatives revient donc à trouver la meilleure combinaison de ces variantes open. L'espace de recherche à évaluer peut devenir très large. Afin de modéliser notre problème, nous pouvons exploiter la structure d'arbre des modèles des features, l'une des structures de données les plus efficaces, fréquemment utilisées dans la recherche opérationnelle afin d'aider à identifier le meilleur chemin. Ainsi, un chemin peut désigner une combinaison de ces variantes open. D'une manière plus générale, la conception de BD correspond dorénavant à un chemin, et plus précisément le plus optimal en termes des BnF parmi les chemins composant l'arbre global.

La sélection de la meilleure configuration pour la conception des BD, peut donc se ramener à la résolution du problème de la recherche du meilleur chemin dans l'arbre global des features. Il existe un ensemble d'algorithmes classiques pour résoudre ce type de problème que nous pouvons exploiter. Nous présentons dans ce qui suit, les étapes qui permettent de transformer notre problème initial à ce problème d'optimisation en recherche opérationnelle en expliquant la démarche.

3.3.1 Passage des FM à la structure d'arbre

Nous commençons d'abord par la transformation des modèles de features (FM) en structure d'arbre. La définition de l'ordre entre les noeuds de l'arbre est guidée par deux paramètres. (1) Les dépendances des différents FM issues de l'analyse de la variabilité du processus de conception, par exemple l'ordre chronologique qui régit les étapes et qui stipule que le design conceptuel vient avant le logique avant le physique. En ce qui concerne la sélection du SGBD,

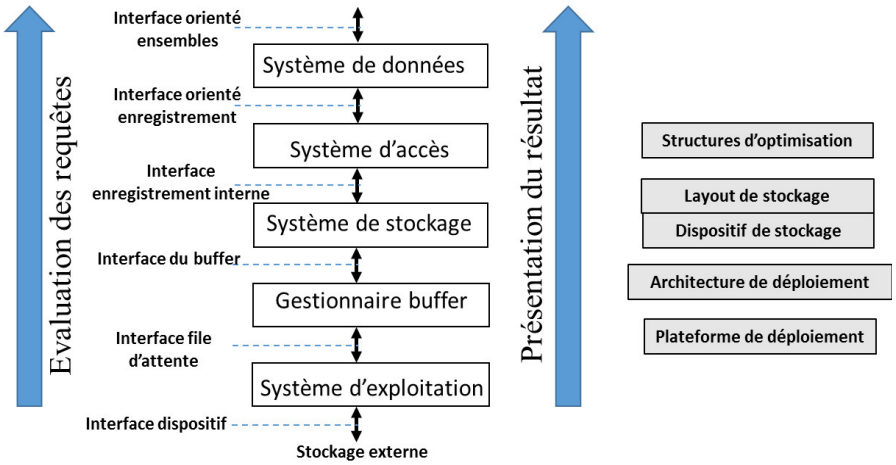


FIGURE 3.8 – L'architecture en cinq couches des systèmes de BD

nous distinguons deux possibilités : soit cette sélection est réalisée au début de la conception logique (l'approche actuelle [66]), soit plus tard dans le processus (pendant ou à la fin de la conception physique) après le raffinement des besoins utilisateurs. (2) L'architecture indépendante des données des systèmes de BD, composée de cinq couches, décrite dans la figure 3.8. Elle détermine l'ordre au sein de la phase physique, selon lequel le traitement des requêtes se déroule à travers les différentes couches d'un SGBD.

Maintenant que l'ordre de features est déterminé, nous définissons ci-dessous un ensemble de règles pour la gestion de ce processus de transition des FM à l'arbre global de conception.

- Seules les features feuilles (variantes) sont transformées en noeuds d'arbre. Leur point de variation est quant à lui, transformé en noeud neutre comme expliqué dans le point suivant.
- Un noeud neutre est créé à la fin de chaque groupe de features (point de variation) dans le but de réduire le nombre d'arêtes qui interconnectent les groupes frères de features. Ces noeud sont symbolisés par des formes octogonales dans la figure 3.9.
- Si le point de variation représente un groupe alternatif ($\mathcal{FG}_{or}$) comme les structures d'optimisation dans la figure 3.5, une arête est créée partant du noeud précédent (qui est neutre dans la plupart des cas), vers les variantes (noeuds feuilles). Ensuite, une arête relie chaque variante à ses noeuds successeurs, y compris le noeud neutre. Le même principe est appliqué si la feature est optionnelle.
- Certaines variantes ajoutent d'autres noeuds qui ne sont pas explicitement représentés dans le modèle des features. À titre d'exemple, la variante "Cloud" (ou *distribué*) du point de variation *plateforme de déploiement* de la figure 3.5 doit créer autant de noeuds que le nombre de sites impliqués. Cette information est implémentée avec la feature (l'entrée $\mathcal{I}$ du triplet f du framework générique des LdP)

La figure 3.9 donne un aperçu de l'arbre issu de cette transformation.

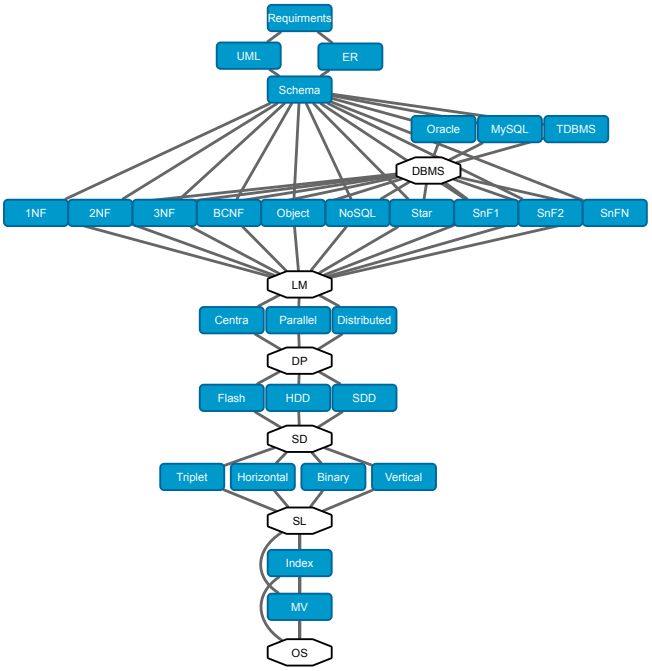


FIGURE 3.9 – Un aperçu simplifié de l'arbre correspondant au processus de conception des BD

3.3.2 Annotation de l'arbre de conception et évaluation des chemins

Une fois que l'arbre est construit, les noeuds ainsi que les arêtes doivent être annotés. L'annotation consiste à assigner aux arêtes et éventuellement aux noeuds, une valeur communément appelée un *poids*, d'où le nom de *pondération* désignant le processus d'annotation dans le jargon des graphes et des arbres. La détermination du poids des arêtes et des noeuds, ainsi que l'évaluation des chemins à base de ces poids, dépend évidemment du critère évaluation, représenté dans notre contexte par les besoins non fonctionnels. À ce sujet, nous distinguons les deux types suivants.

3.3.2.1 BnF orientés utilisateurs finaux. Ce type de besoin concerne la qualité des éléments de conception faisant office de moyen de communication entre les concepteurs et les parties prenantes. Il s'agit, plus particulièrement, des modèles et des interfaces destinés aux utilisateurs finaux de la BD. La qualité de ces éléments de conception se mesure par rapport aux utilisateurs cibles (clients). L'utilisabilité, la compréhensibilité, l'expressivité, le coût financier, la sécurité (partiellement, car la définition des entités publics et privées peut se faire via la modélisation) et la satisfaction des besoins fonctionnels sont de parfaits exemples de ce genre de BnF, que l'on trouve particulièrement dans le design conceptuel et le design de la partie applicative de la BD.

Par ailleurs, la prédiction et l'évaluation de ce genre de BnF et donc, de la qualité des ces éléments de conception (modèles), ont été largement étudiées par de nombreux travaux qui ont été classifiés par Lemaitre et al. [112] en trois cadres : hiérarchiques, sémiotiques, et de causalité. Ces cadres englobent des méthodologies, métriques, patrons, et recommandations proposées pour la prédiction et l'évaluation des BnF orientés utilisateurs.

3.3.2.2 BnF orientés base de données. La qualité recherchée par ce deuxième type de BnF, se mesure par rapport au fonctionnement de la BD elle même. Elle concerne donc surtout la partie processus (optimisation), que nous trouvons notamment au sein des phases logique (normalisation et partitionnement) et physique (déploiement et optimisation), plutôt que dans la partie modélisation de la phase conceptuelle. Nous pouvons résumer ce genre de BnF avec le mot *performance*, une notion traditionnellement réservée principalement au temps d'exécution des requêtes, que l'on peut également affecter à l'espace de stockage nécessaire, un critère primordial à l'époque où les dispositifs de stockage étaient encore cher et peu performants. Cette notion s'est étendue pour couvrir d'autres critères émanant de l'évolution des technologies BD, comme par exemple, la consommation de l'énergie. Quant à la prédiction et l'évaluation de ce type de BnF, elle se ramène souvent à des problèmes d'optimisation et les concepteurs se reposent souvent sur des métriques et des algorithmes avancés appelés modèle de coût (voir section 2.2.4).

Il est important de noter qu'il existe des BnF qui peuvent être orientés utilisateurs et BD en même temps. Un exemple est le BnF concernant la sécurité des données, qui implique d'abord la distinction entre les entités privées et publiques au niveau conceptuel, afin d'être validées par les utilisateurs finaux (orienté utilisateur). Une définition des politiques de protection et de confidentialité est ensuite établie au niveau physique (orienté BD). La figure 3.10 illustre la répartition des deux types des BnF sur le cycle de conception des BD.

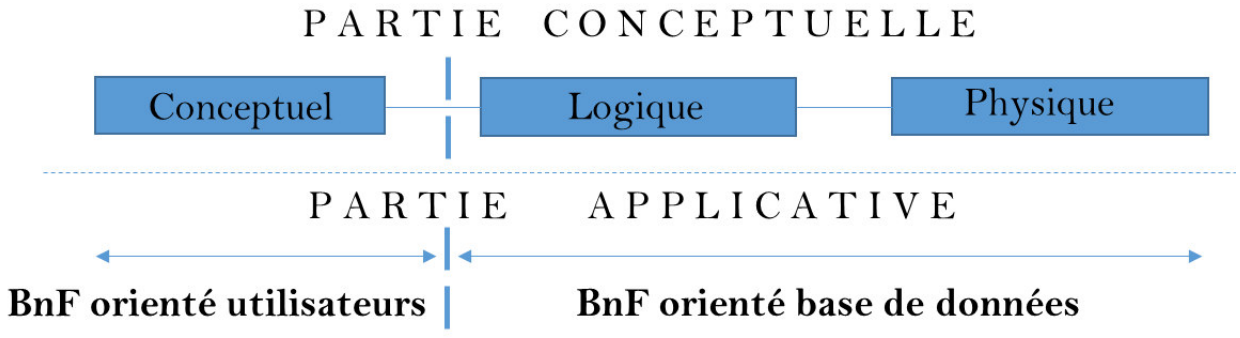


FIGURE 3.10 – Répartition des types de BnF sur le cycle de conception des BD

3.3.2.3 Pondération des noeuds et évaluation des chemins en fonction du type des BnF. Quand il s'agit des BnF orientés utilisateurs, l'annotation des composants de l'arbre se fait

à l'aide des métriques définies pour chaque élément et l'évaluation des chemins consiste à additionner ces poids, à la manière de l'exemple 6 ci-dessous, où chaque noeud composant un chemin, peut être pondéré par un "sous-poids" et le poids total d'un chemin correspond à la somme de ces sous-poids.

Exemple 6 (*pondération de l'arbre de conception en vue d'évaluer un BnF orienté utilisateurs*)

Si l'on prend l'exemple du coût financier du processus de conception des BD comme BnF, alors certains noeuds notamment le SGBD et les dispositifs de stockage seront annotés par des poids correspondants, par exemple, aux prix fixés par le marché ou aux frais de la montée en compétence nécessaire pour la maîtrise des technologies. Dans ce cas, le poids d'un chemin correspond à la somme des coûts de chaque noeud.

Le processus devient plus compliqué dans le cas des BnF orientés BD. En effet, comme expliqué précédemment, la prédiction et l'évaluation se font généralement grâce aux modèles de coûts. Ces derniers requièrent des entrées qui peuvent provenir de différentes parties du processus de conception de BD. Les noeuds ne peuvent être pondérés que lorsque la définition de toute la configuration correspondante a été faite, en d'autres termes, jusqu'à ce que tout le chemin soit parcouru. Et comme nous visons le traitement de ce processus dans sa globalité, la pondération doit donc se répartir sur plusieurs noeuds. Les noeuds et/ou arêtes concernées se verront affecter un paramètre requis pour l'utilisation du modèle de coût (*MdC*), comme illustré dans l'exemple 7.

Exemple 7 (*Pondération de l'arbre de conception en vue d'évaluer un BnF orienté BD*)

En pratique, le MdC conçu pour la prédiction du temps d'exécution dépend du plan d'exécution des requêtes, c'est à dire de l'ordre de leurs opérations algébriques. Ces opérateurs sont déterminés par le modèle logique de données, et possèdent chacun une formule algébrique pour le calcul du MdC en question. Cette formule change en fonction de l'algorithme d'implémentation des opérateurs, des méthodes d'accès et de la structure d'optimisation utilisée. Les paramètres d'entrée du MdC proviennent de différentes phases de conception (logique, déploiement et physique) (voir section 2.2.4) : schéma de BD (attributs, instances, facteurs de sélectivité des prédicats), requêtes ainsi que les estimations (statistiques) des volumes de données manipulées par la charge[29]*, modèle de déploiement, politique de traitement de requêtes dans le cas distribué, support de stockage, et ainsi de suite. Par conséquent, le processus d'annotation et du calcul du poids d'un chemin se répartit sur les features comme suit :*

- *le formalisme conceptuel ne possède pas d'impact direct sur le temps d'exécution des requêtes et sur les BnF orientés BD d'une manière générale. Basée sur la modélisation, l'évaluation de la qualité de cette feature rentre dans le cadre des BnF orientés utilisateur comme la compréhensibilité, l'utilisabilité ou la sécurité;*
- *le schéma conceptuel peut intervenir dans le calcul du MdC, car il détermine le schéma local concerné par l'application en question, et donc spécifie les requêtes devant être considérées dans la charge globale ;*

— *le formalisme logique* détermine les opérateurs logiques à considérer à travers l'analyse des requêtes ;

— *la plateforme de déploiement*, *le modèle de stockage*, et éventuellement *la normalisation logique* et *l'architecture de déploiement* permettent la réécriture finale des requêtes. En effet, les requêtes doivent être écrites selon le déploiement final des données qui dépend de toutes ces features ;

— *le dispositif de stockage* et *moteur de traitement* alimentent le MdC par différents paramètres matériaux nécessaires au calcul, comme par exemple, la taille de la mémoire, du bloc, de page, le nombre des noeuds dans les environnements non-centralisés, la fréquence du calcul ;

— *les structures d'optimisation* qui déterminent la formule à utiliser dans le calcul.

3.3.3 Dérivation augmentée de BD : démarche

Tous les paramètres évoqués précédemment, qui sont relatifs à la pondération des noeuds/arêtes de l'arbre de conception, doivent être intégrés et implémentés au sein des features. Le processus de dérivation, est alors divisé en trois étapes : (1) l'évaluation des chemins de conception en se basant sur les différents paramètres stockés dans les features en fonction des BnF (et donc du MdC) utilisés, (2) la désignation du chemin optimal, ayant le meilleur poids et (3) l'exécution classique de la configuration issue du chemin optimal. Rappelons que cette exécution fait appel à l'implémentation de chaque feature sélectionnée et intègre l'ensemble suivant l'architecture des LdP. Notons que nous n'excluons pas le scénario où le concepteur peut ne pas laisser de points de variation open, en ayant fixé les variantes tout au long du processus. Dans ce cas, la dérivation se réduit à l'exécution de cette configuration (l'étape 3).

L'implémentation globale de notre framework de LdP, ainsi que le processus de dérivation enrichi pour l'aide à la décision, est basé sur une démarche en plusieurs étapes, comme expliqué dans la section suivante.

4 Approche par étapes pour la gestion de la variabilité

Aborder d'une seule traite l'implémentation globale de notre LdP revient à implémenter plusieurs features, fortement dépendantes, appartenant à différentes phases de la conception des BD. En suivant le principe du "diviser pour régner", vu l'étendue de la tâche, nous commençons d'abord par se focaliser sur la phase logique de ce cycle de vie, élaborer son arbre avec ses chemins potentiels. Nous étudions comment les évaluer selon plusieurs BnF et comment évaluer

29. Selon qu'il s'agisse du premier ou du second scénario (présence ou absence de données/requêtes).

ses interdépendances. L'implémentation finale de notre LdP (problème global) revient alors à généraliser les résultats de ce cas d'étude vers d'autres phases et features.

4.1 La gestion de la variabilité de la conception logique

Nous nous intéressons à la variabilité de la conception logique car notre revue de littérature présentée dans le chapitre 2, a révélé que la variabilité au sein de la phase logique a été la moins étudiée en comparaison avec les autres phases. Nous voulons en effet, démontrer le fait que même si plusieurs problèmes de performance (au sens large) peuvent être résolus avec un ajustement fin du modèle physique, quelques-uns sont causés par la non-optimisation du schéma logique [78, 104, 113]. Nous commençons d'abord par instancier les objectifs principaux de la *gestion de la variabilité holistique* dans la phase de la conception logique :

1. mettre à la disposition du concepteur les variantes possibles du modèle logique ;
2. offrir un *mécanisme d'aide à la décision orienté BnF* pour assister le concepteur à sélectionner le modèle logique le plus adéquat avec les besoins de la BD. Ce mécanisme doit évaluer les variantes selon les BnF avant d'en proposer la meilleure ;
3. offrir un mécanisme de *réutilisation*, pour accélérer et (semi-)automatiser la solution au problème ;
4. offrir un mécanisme de *configurabilité* qui permet de faire adapter la sélection du modèle logique selon le contexte d'application.
5. Ce que l'on entend par *holistique* se résume dans la prise en compte des *interdépendances* présentes entre les phases de conception. L'étude de cas doit donc s'étaler sur au moins deux phases.

La réutilisation et la configurabilité de la BD (points 3 et 4) sont offertes par l'implémentation de la technique des lignes de produits. L'énumération des modèles logiques, le mécanisme d'aide à la sélection orientée BnF du modèle logique, la prise en compte des interdépendances (points 1, 2 et 5) doivent être soulevés et résolus dans notre étude de cas.

D'un autre côté, notre attention s'est d'abord portée vers les besoins non fonctionnels orientés BD (performance, énergie et espace de stockage), car ils sont traités par les différents axes de recherche de notre laboratoire (LIAS) et parce que ces derniers couvrent une partie plus importante du cycle de conception (voir figure 3.10). En même temps, les BnF orientés BD interviennent dès la conception logique, ce qui restreint l'étude de l'interdépendance sur les phases logique et physique.

Par conséquent, le reste du manuscrit est consacré à l'étude de la variabilité au sein de la phase logique, son impact intraphase et interphase (sur la phase physique), ainsi que l'évaluation de ces impact en matière de BnF orientés BD : la performance, l'énergie consommée, et l'espace de stockage. Nous commençons par présenter la forme choisie de cette variabilité logique, ainsi que les objets l'ayant engendrée dans la section suivante. Nous enchaînons ensuite

avec la présentation des "opérations" cibles choisies pour représenter l'impact da la variabilité logique à évaluer. Il s'agit des optimisations logiques et physiques expliquées dans la section 3.4.3.

4.2 Choix de feature pour incarner la variabilité au sein de la conception logique

Comme nous l'avons déjà indiqué, la variabilité au sein de la phase logique peut concerner trois dimensions (features): (1) le formalisme utilisé (le modèle relationnel par exemple), (2) les différentes traductions du modèle conceptuel vers le modèle logique, (3) la normalisation du modèle logique obtenu en utilisant les dépendances qui peuvent exister entre les différentes propriétés du modèle logique et (4) l'optimisation logique. Il est indispensable de rappeler que le modèle logique obtenu dans cette phase est implémenté dans la BD (souvent appelé modèle physique dans le jargon de la communauté française des BD [72]). Sur ce dernier, des *optimisations logiques* sont d'abord définies, en exploitant les propriétés offertes par le langage de requêtes utilisé (à l'exemple de la commutativité de la sélection, la distribution de la jointure sur l'union, etc.). L'optimisation physique vient ensuite appuyer le schéma implémenté de la BD en faisant appel à des structures physiques de données.

En ce qui concerne le *formalisme*, la variabilité est souvent guidée par le SGBD utilisé pour persister la BD et qui détermine les *règles de mappings* à appliquer pour traduire le modèle conceptuel à un modèle logique de données. Le choix du SGBD dépend du type des données à manipuler et du type de l'application (ses objectifs) (voir section 3.3.2). Quant à la *normalisation*, elle est souvent respectée par la plupart des méthodologies de conception de BD. Elle se repose sur la notion de corrélation [30], qui peut être définie comme une relation pouvant lier des propriétés et/ou des entités. Ces corrélations représentent une des sources majeures de la variabilité des BD en général, et les entrepôts de données (ED), en particulier. En effet, contrairement à la conception des BD, celle des ED est moins sensible à l'utilisation des corrélations. Cela signifie que les concepteurs ont souvent tendance à normaliser les BD transactionnelles, une habitude moins fréquente dans les ED. En revanche, l'exploitation des ED est plus consommatrice des corrélations, à travers l'utilisation des opérations d'analyse à l'instar du *drill down* et du *roll up*. L'*optimisation logique*, quant à elle, concerne plutôt les requêtes définies sur un schéma logique souvent figé. Faire varier ce schéma pourrait impacter l'optimisation de ces requêtes, ce que nous allons montrer par nos propositions théoriques et expérimentales. Quant à cette variation, et pour toutes les raisons susmentionnées, nous proposons de la traiter au moyen de la *normalisation* du schéma logique dans le contexte des ED. Celle-ci est basée sur les corrélations qui seront détaillées dans la section suivante.

30. alias : dépendances, relations voire contraintes d'intégrité

4.2.1 Classification des types de corrélations

Afin de bien illustrer chaque type de corrélation, nous utilisons un aperçu du modèle conceptuel du benchmark *Star Schema Benchmark* (SSB) présenté dans la figure 3.11 (et qui sera utilisé dans nos futures expérimentations). Nous l'avons obtenu par rétro-conception sur son modèle logique d'origine. Nous utilisons également la *Logique de Description* (LD) [31] afin de présenter formellement ces corrélations. Nous devons d'abord définir une fonction pour la représentation des axiomes terminologiques du formalisme LD. Il s'agit de la fonction Ref définie comme suit :

Définition 1 (*La fonction Ref*)

$$Ref : C \cup R \rightarrow (Operator, Exp(C, R)) \text{ telle que :}$$

- *Ref est une fonction qui associe à chaque classe un opérateur (d'inclusion ou d'équivalence) et une expression Exp sur d'autres classes et propriétés.*
- *C désigne les concepts, qui sont une généralisation de la notion de classe du langage UML.*
- *R désigne les roles, une généralisation de la notion de propriété du langage UML.*
- *Les opérateurs (operator) peuvent être une inclusion ($\sqsubseteq$) or une équivalence ($\equiv$).*
- *$Exp(C, R)$ est une expression définie sur les concepts et les roles en utilisant l'ensemble des constructeurs LD comme la restriction, l'union, l'intersection et ainsi de suite. Exp peut être la fonction identité qui associe à une classe la même classe (la classe se définit par elle-même comme la plus grande classe de la hiérarchie Thing). Par exemple, la classe Order du modèle conceptuel SSB est définie comme sous classe de la classe Thing, comme suit : Ref(Order) = ($\sqsubset$, Thing).*

Classification des corrélations En se basant sur la fonction *Ref* définie précédemment, nous proposons de classifier les corrélations dans les catégories suivantes.

1. *Relation de définition ou d'équivalence (DEF)* : quand une classe est définie en fonction d'autres objets via les opérateurs de *LD*, pour apporter plus de précision.

31. La logique de description est constituée d'une famille de langages de représentation de connaissance, ayant pour objectif de représenter la connaissance terminologique d'un domaine d'application d'une manière formelle et structurée. LD est reconnue d'être parmi les langages les plus aptes à représenter des formalismes de modélisation basés sur la notion des classes de données, et actuellement utilisés dans l'analyse des BD et des systèmes d'information [40]

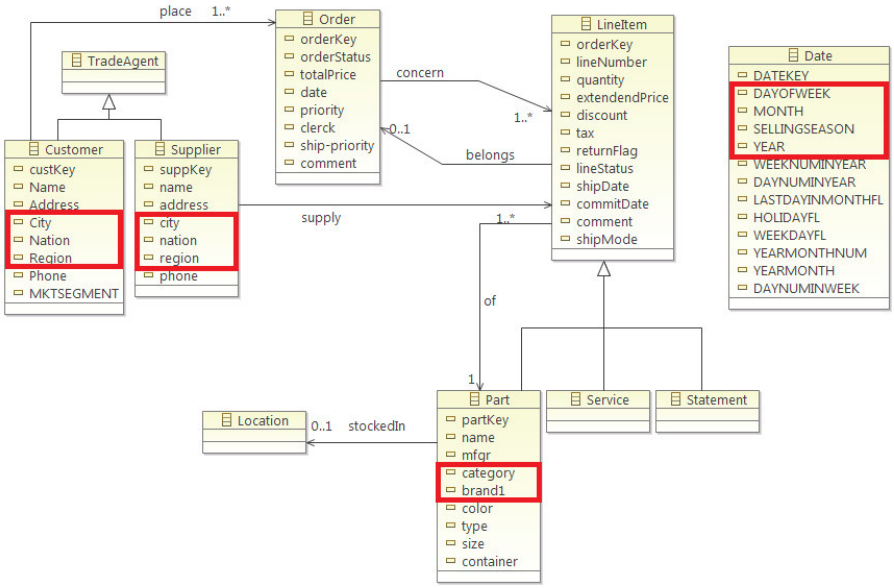


FIGURE 3.11 – Un extrait du modèle conceptuel du benchmark SSB

Exemple 8

$$Ref(Supplier) \rightarrow (\sqsubseteq, TradeAgent \sqcap \forall supply(TradeAgent, LineItem))$$

Cet exemple exprime le fait que le fournisseur "Supplier" est défini comme étant un agent commercial "TradeAgent" qui fournit "supply" des lignes de commandes "LineItem".

2. *Dépendance d'inclusion (DI)*[32] : représente le fait qu'un concept est subsumé par un autre. Quand cela concerne les attributs et non les classes, on peut affecter une autre application à ce genre de corrélation : les clés étrangères. Elles expriment le fait que le domaine d'un attribut doit être un sous-ensemble inclus dans le domaine d'un autre attribut corrélé (clé primaire).

Exemple 9

$$Ref(Customer) \rightarrow (\sqsubseteq, TradePart)$$

Cet exemple exprime le fait que le client "Customer" est une spécialisation d'un agent commercial "TradeAgent".

3. *Dépendance fonctionnelles (DF/DC)*[33] : quand un ensemble de classes (ou de leurs instances) **déterminent** un autre ensemble de même type. Par exemple, étant donné un ensemble d'instances de la classe A, il lui correspondrait un et un seul ensemble d'instances de la classe B. Deux catégories de dépendances existent [43] : dépendances basées sur les

32. aussi appelé : relation *is-a*, ou relation de subsomption.
33. Nous notons *DF* quand il s'agit d'attributs, et *DC* quand il s'agit des classes.

instances et dépendances statiques (basées sur les définition des classes). Cette derniére catégorie est spécifique aux BD sémantiques où les définitions de classes sont disponibles. Ce type de relations permet une normalisation de données en vue de réduire la redondance et de garantir plus de cohérence.

Exemple 10

$$Ref(Customer.custKey) \rightarrow (Customer.Name)$$

Cet exemple définit une dépendance fonctionnelle qui stipule que la clé primaire d'une table relationnelle doit déterminer tous les autres attributs de la même table. En l'occurrence, la clé primaire du client "Customer" détermine (entre autres) son nom.

4. *Dépendance multivaluée (DM)* : spécifique aux attributs, elle est considérée comme étant une généralisation du concept des dépendances fonctionnelles. Formellement, ce qui distingue ce type du précédent, c'est la suppression de la notion de détermination (unicité de correspondance). Autrement dit, à un ensemble de valeurs d'attributs, il peut correspondre un ou plusieurs ensembles de même type (notion plus génerale). À titre d'exemple, les attributs *city* et *nation* sont corrélés, si on prend la ville de Montréal comme *city*, le pays (*nation*) correspondant peut être soit Canada, soit France. Ce type de relations permet une meilleure précision et des travaux ont montré son grand impact sur les performances des requêtes [48, 107]

5. *Contraintes d'intégrité conditionnelles (CIC)* : spécifiques aux attributs, elles représentent les contraintes d'intégrité CI [34] algébriques ou sémantiques régissant les instances des relations (données de la BD) impliquant plus d'un attribut [147]. Notons que les corrélations définies jusque là (définitions et dépendances) sont considérées comme de simples contraintes d'intégrité devant être valides sur l'ensemble de la relation, contrairement aux CIC. Ces dernières sont accompagnées de conditions à satisfaire, pour qu'elles prennent effet. Cela fait descendre le niveau d'application des CI du domaine des attributs au niveau valeurs. En pratique, juste un sous-ensemble du domaine des attributs concernés est impliqué.

Exemple 11

$$Customer.City = Poitiers \rightarrow LineItem.discount > 20\%$$

Cet exemple stipule que si les clients sont poitevins (Customer.City = Poitiers) alors le discount associé à toutes les lignes de commande doit être supérieur à 20%.

Nous distinguons deux principales catégories de CIC. La première concerne les dépendances fonctionnelles conditionnelles [28], qui comme leur nom l'indique, ne sont appli-

34. Les contraintes d'intégrité définissent des conditions ou des propositions qui doivent être maintenues vraies, comme par exemple Part.size>0, qui spécifie que la taille du produit doit toujours être une valeur positive.

quées que sur un sous-ensemble d'une relation dont les tuples suivent un pattern spécifique.

Exemple 12 (*dépendances fonctionnelles conditionnelles*)

$$[Customer.City = Poitiers, Customer.Name] \rightarrow [Customer.phone]$$

Cet exemple stipule que seuls les clients poitevins ont un nom qui permet de déterminer leur numéro de téléphone.

La deuxième catégorie désigne des règles d'association qui ne s'appliquent que sur des valeurs particulières de certains attributs [7].

Exemple 13 (*règles d'association*)

$$Part.color = \text{`}red' \rightarrow Part.size = 50$$

Cet exemple exprime que la taille des produits "Part" de couleur rouge est égale à 50.

6. *Relation de Hiérarchie (H)* : peut être assimilée à la relation *partie-tout*[35]. Ce type de relations est le sujet d'une discipline appelée *Méréologie* basée sur le calcul des prédicats du premier ordre et une ontologie formelle pour les traitements. Une *partie-tout* est une relation d'ordre partiel, réflexive, transitive et antisymétrique qui lie les attributs des dimensions des schémas multidimensionnels (ED). L'hiérarchie *{industrie, catégorie, produit}* en est un exemple. La particularité de ce type, vient du fait qu'ici, nous ne tenons pas compte de la correspondance entre les attributs éléments, car nous envisageons par la sélection de ces derniers, une construction de nouvelles classes normalisées, tandis que dans les deux types précédents, nous envisageons plutôt d'exploiter ces relations comme des règles à satisfaire sur l'ensemble de données, via la vérification de la correspondance entre les attributs dominants et dépendants pour les dépendances fonctionnelles DC, DF, DM et l'utilisation de cette correspondance dans un processus (définition des structures d'optimisation par exemple) pour les autres types.

4.2.2 Domaines d'application des corrélations

Les corrélations représentent une des sources majeures de la variabilité des BD car elles interviennent dans plusieurs domaines relevant de la conception et de l'exploitation de la BD. En effet, elles sont d'une grande importance pour l'analyse automatique des systèmes d'information (exploitation des BD). Cette analyse s'inscrit dans un processus de découverte de relations implicites entre les données de la base, potentiellement utiles à la description de leur comportement actuel, et transitivement à la prédiction de leur comportement futur. C'est la spécialité des domaines de l'extraction des connaissances et de l'exploration des données [7], où la détection

35. `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole`

de ces relations sert à construire des modèles de description et de prédiction automatiques. Ces relations ont également servi de base à l'auto-administration des SGBD (conception et tuning des BD), où l'on oeuvre pour rendre ce dernier plus autonome en automatisant le plus possible de tâches administratives, facilitant ainsi le maintien, l'utilisation et surtout la réduction des coûts d'administration des systèmes. La preuve en est que de plus en plus de fournisseurs de SGBD commerciaux considèrent l'autonomie de l'administration (*self-tuning* [36]) comme un élément crucial à intégrer dans le SGBD [109, 81]. Les optimiseurs de requêtes qui exploitent ces relations entre les données de la base sont une composante importante de cette technologie émergente [37, 106]. D'un autre côté, l'exploitation des corrélations s'est également avérée être utile dans un autre domaine qui est la rétro-ingénierie [140] (conception des BD).

L'importance des corrélations a attiré l'attention de la communauté académique autant bien que celle de la communauté industrielle. Elles en ont fait l'objet de plusieurs travaux, pouvant porter soit sur : (i) l'automatisation de leur découverte car leur détection n'est souvent pas évidente pour les concepteurs, particulièrement quand il s'agit de manipuler de gros volumes de données, (ii) leur exploitation dans les différents domaines que nous venons d'évoquer.

4.2.2.1 Découverte (extraction) des corrélations.

Ci dessous, nous donnons un aperçu de quelques travaux concernant la découverte des corrélations.

— *BHUNT*[37] est une technique basée sur les données qui vise à automatiquement détecter les relations algébriques entre les attributs de type numérique de la BD. Par la suite, elle offre cette information à l'optimiseur des requêtes sous forme de contraintes avec en plus une estimation de sélectivité associée. Cette information est d'une grande utilité pour l'amélioration de l'estimation des coûts, la proposition de nouveaux chemins d'accès ainsi que la considération d'autres structures d'optimisation comme les vues matérialisées et les stratégies de partitionnement.

— *CORDS* [92], est un outil basé sur la technique précédente afin de découvrir automatiquement des corrélations statistiques et des dépendances multivaluées liant des attributs de type numérique et non numérique.

— Deux algorithmes sont proposés dans les travaux menés par Mannila et al. [121] pour trouver une couverture d'un ensemble de dépendances fonctionnelles qu'une table relationnelle peut déceler. Ces algorithmes sont appliqués dans le contexte de l'optimisation des requêtes des BD relationnelles, ainsi que dans l'intelligence artificielle.

— Le fameux éditeur *Talend* [37] offre lui aussi une fonctionnalité d'analyse de corrélations entre les colonnes d'une table en utilisant des fonctions spécifiques de calculs.

4.2.2.2 Exploitation des corrélations

36. http://research.microsoft.com/en-us/projects/autoadmin/default.aspx

37. Talend: open source data management and data integration systems software solutions. `http://www.talend.com/`

— Stohr et al. [176] proposent une méthode de fragmentation de la table des faits des entrepôts de données parallèles de type étoile, en se basant sur la structure hiérarchique des dimensions afin d'optimiser le temps de traitement des requêtes.

— Anderlik et al. [13] exploitent les hiérarchies de subsomption des concepts ontologiques comme noyau des hiérarchies de *Roll-up* (dimensions sémantiques) utilisées pour l'aggrégation des données.

— Kimura et al. [106] permettent la sélection des index cluster, ayant de fortes dépendances avec les attributs issus des prédicats de sélection composant les requêtes. En effet, cela permet de réduire la taille des index secondaires définis sur ces attributs, et donc d'améliorer la performance. Des structures de vues matérialisées sont ensuite choisies sur la base de ces index corrélés (cluster et secondaires), ce qui réduit le nombre des vues matérialisées possibles.

Nous pouvons affecter les travaux précédents et d'autres plus récents aux différents types de corrélation, que nous répartissons sur le cycle de conception des BD comme résumé dans le tableau 3.2 suivant.

Travaux \ Phases	Conceptuelle	Logique	Physique	OLAP	Exploitation
Anderlik & al. [13]				*DEF*/*DI* Roll-up	
Stohr & al. [176]		*H* Fragmentation			
Kimura & al. [106]			*DF*/*DM* MV/index		
Brown & al. [37]			*CIC* optimiseurs		
Agrawal & al. [7]					*CIC* Data-mining
Petit & al. [140]	*DI*/*DF* schéma ER	*DI*/*DF* schéma relationnel			*DI*/*DF* Rétro-ingénierie
Liu & al. [117]			*DF*/*DM*/*CIC*		
Hoang-vu & al. [134]		*DF*/*DM*			
Haas & al. [85]		tout type	tout type		
Naumann & al. [130]		*DF*/*DM*/*CIC*	*DF*/*DM*/*CIC*		*DF*/*DM*/*CIC*

TABLE 3.2 – Travaux connexes sur l'exploitation des corrélations tout au long du cycle de conception des BD

Maintenant que nous avons vu la forme ainsi que les objets engendrant la variabilité logique à laquelle nous nous intéressons, nous allons présenter les opérations (features) que nous avons choisies pour refléter cette variabilité. Ces opérations nous permettent d'évaluer l'impact de

cette dernière.

4.3 Choix des features pour refléter l'impact de la variabilité logique

La variabilité logique est incarnée par la feature de la *normalisation* basée sur les corrélations qui existent entre les propriétés. Cependant, il existe des interdépendances entre les différentes features que ce soit au niveau intraphase (par exemple le *formalisme logique* influence les *règles de mappings*), ou au niveau interphase (par exemple le *formalisme logique* influence le *déploiement* de la phase physique) (voir section 3.2.5). Ces interdépendances font que la variabilité d'une feature ait un impact sur ses features dépendantes. Ainsi, nous devons choisir deux features dépendantes à notre variabilité logique pour refléter son impact intra et inter.

Sur le plan *intraphase*, la seule feature locale à la phase logique et qui est dépendante de la *normalisation* du schéma, est l'*optimisation logique*. En effet, le *formalisme logique* définit les *règles de mappings* utilisées pour traduire un schéma conceptuel vers le schéma logique (par exemple, le modèle relationnel [72, 71] et le modèle orienté objet [111]). Il définit également les règles du processus de la *normalisation* de ce schéma traduit. Ainsi, le *formalisme logique* et les *règles de mappings* sont des features dominantes par rapport à la *normalisation*, c'est à dire que c'est elles qui l'impactent mais pas l'inverse.

Sur le plan *interphase*, nous avons choisi de lier la phase logique à l'*optimisation physique* qui représente l'un des objectifs principaux de la phase physique. Dans notre étude, nous nous intéressons au processus de la sélection des vues matérialisées. Nous allons donc définir toutes ces notions des optimisations logiques et physiques ci-dessous.

Le traitement et l'optimisation des requêtes ont été l'un des sujets de recherche les plus actifs des systèmes d'information et des BD, peu importe leur génération (traditionnelles, orientées objet, XML, graphiques, ED, etc.). Ces deux notions sont souvent associées à la conception physique, étant la dernière étape et jouant ainsi le rôle d'*entonnoir*. Les problèmes liés à l'optimisation des requêtes et à la conception physique sont tous deux connus comme des tâches NP-difficiles [120]. En effet, ils impliquent plusieurs paramètres qui, même s'ils sont captés pendant la phase physique, appartiennent à différentes phases du cycle de vie de conception des BD : conceptuel, logique, déploiement et physique. Par ailleurs, ces problèmes cherchent à répondre à différents types de BnF (stockage, maintenance, énergie). Nous en distinguons deux principaux types de ces problèmes : optimisation logique et physique détaillées comme suit.

4.3.1 Optimisation logique

Quand une requête est soumise par un utilisateur, elle est d'abord parsée en utilisant un ensemble de règles de grammaire, pour vérifier si elle est correcte en matière de syntaxe et de droits d'accès. Si les tests sont réussis, elle serait convertie en une représentation interne sous forme d'arbre (plan d'exécution). Celui-ci peut avoir plusieurs représentations équivalentes, gé-

nérant les mêmes résultats finaux, mais avec des performances différentes. Raison pour laquelle, un module de *réécriture* vient y effectuer un nombre d'optimisations offertes par les propriétés du langage utilisé (algèbre du modèle relationnel par exemple), afin de le transformer à un plan plus performant vis-à-vis d'un BnF [141]. Cette approche est dite *dirigée par les règles*. Une autre approche a vu le jour pour venir compléter l'ancienne, dite *dirigée par des modèles de coût*, qui permet d'évaluer via un modèle de coût, tous les plans possibles avant de choisir le meilleur (voir section 2.2.4). Les deux approches opèrent au niveau logique. La figure 3.12 résume les différentes étapes impliquées (optimiseur de requêtes).

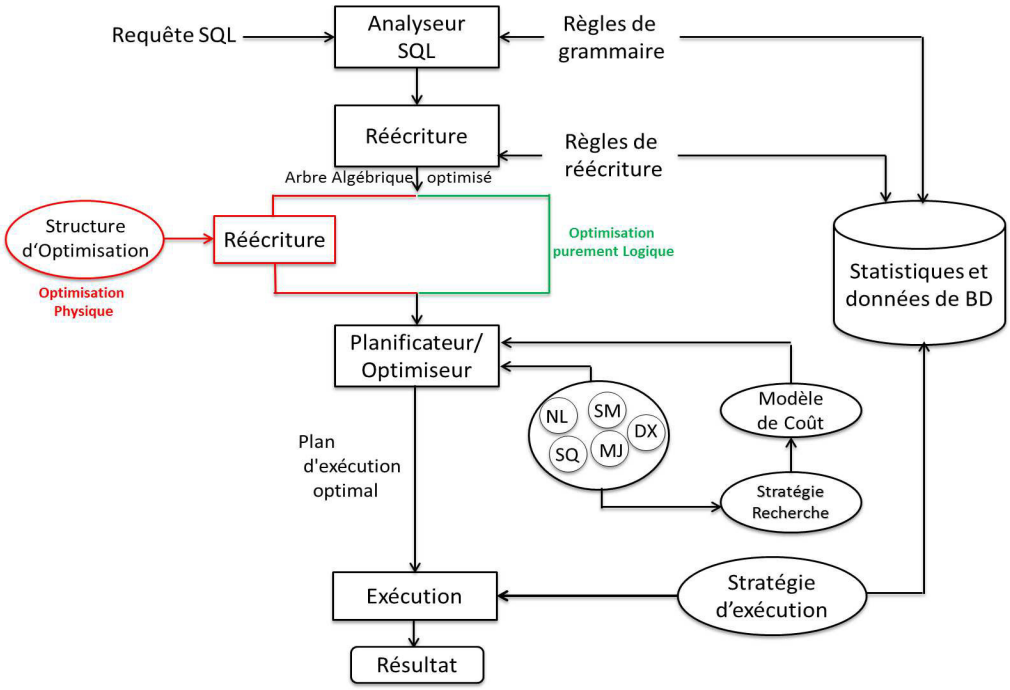


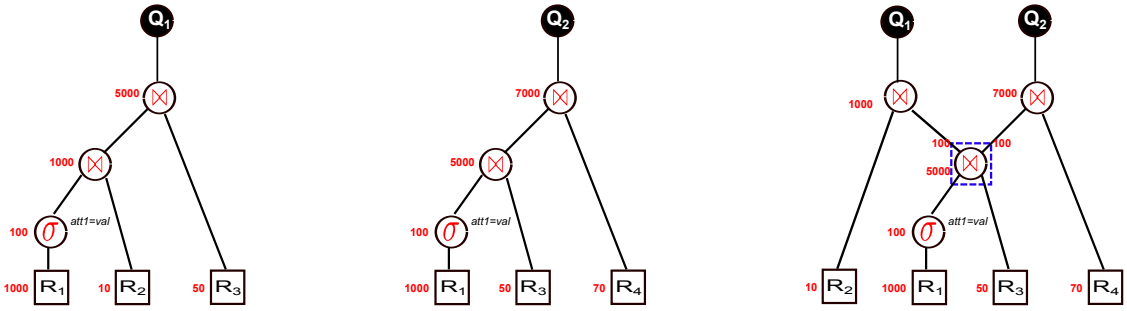
FIGURE 3.12 – Fonctionnement de l'optimiseur des requêtes (optimisation logique et physique)

Un autre type d'optimisation logique peut se distinguer si les requêtes sont traitées par lots. En effet, leur interaction matérialisée par la présence de résultats intermédiaires (sous-expressions) communs, peut être exploitée dans la réduction du coût global de la charge des requêtes. Cette question est connue sous le nom de l'optimisation multi-requêtes (OMR).

Exemple 14 (*Optimisation multi-requêtes OMR*)

Pour illustrer le processus de sélection du meilleur plan de requêtes de manière isolée et conjointe, considérons les relations R_1, R_2, R_3 et R_4 ayant respectivement un coût de scan de 1000, 10, 50 et 70 unités en supposant l'absence de toute structure d'optimisation. L'optimiseur de requêtes suggère les meilleurs plans individuels pour chaque requête et le SGBD utilise ces plans pour exécuter les requêtes de façon indépendante. Comme représenté par la figure 3.13a et 3.13b, les plans optimaux de q_1 et q_2, sont $(\sigma_{att_1=val}(R_1) \bowtie R_2) \bowtie R_3$ et $(\sigma_{att_1=val}(R_1) \bowtie R_3) \bowtie R_4$ respectivement. Le coût total des deux requêtes est de 20380

unités. Cependant, on remarque que le résultat intermédiaire $(\sigma_{att_1=val}(R_1) \bowtie R_3)$ *peut être partagé par les deux requêtes. Comme le montre la figure 3.13c, le plan global a un coût total de 14430 unités. Ainsi, en utilisant OMR, le coût d'exécution est réduit d'environ 30 %.*



(a) Plan individuel de (Q_1)

(b) Plan individuel de (Q_2)

(c) Plan multi-requêtes $(Q_1 \& Q_2)$

FIGURE 3.13 – Avantage de la technique d'optimisation multi-requêtes

Dans l'exemple ci-dessus, la fusion des plans est simple. Mais avec une grande charge de requêtes, la fusion vers un plan global unique, où les résultats des sous-expressions communes doivent être partagées autant que possible, devient un problème difficile [163]. Plusieurs algorithmes heuristiques ont été donc proposés pour améliorer le temps de génération du plan global.

Formalisation 1 (*Optimisation multi-requêtes : OMR*)

Soit :

- $Q = \{q_1, .., q_n\}$ *un ensemble donné de requêtes à optimiser ;*
- $P_i = \{p_1, .., p_{k_i}\}$ *un ensemble de plans individuels possibles pour chaque requêtes* q_i *;*

OMR consiste à trouver un plan d'exécution global en sélectionnant un seul plan de chaque P_i *tel que le coût de traitement de toutes les requêtes est réduit au minimum.*

Par ailleurs, la modélisation de l'OMR a été exploitée par le processus d'optimisation basé sur les structures physiques telles que les *vues matérialisées* [195], ce qui montre la forte dépendance entre les optimisations logiques et physiques tel que discuté ci-dessous.

4.3.2 Optimisation physique

Dans les premières générations des BD, la majorité des travaux étaient principalement axés sur les optimisations logiques. Ceci est dû à la *simplicité* des requêtes de cette époque. En revanche, ces optimisations se sont révélées insuffisantes face aux BD de plus en plus larges, et des requêtes complexes impliquant des jointures et des agrégations [45]. Ce nouveau contexte

a amplifié l'importance des optimisations physiques (et de la conception physique de manière générale). Celles-ci sont assurées par des structures avancées de stockage physique et de méthode d'accès aux données, telles que les vues matérialisées, le partitionnement des données, l'indexation avancée, la compression de données, etc. Les requêtes doivent alors être réécrites en fonction de ces structures préalablement définies (voir figure 3.12), dans le but d'optimiser l'exécution. Dans cette section, nous nous focalisons sur une structure redondante et largement utilisée dans l'optimisation des requêtes OLAP des ED : les vues matérialisées, car nous allons l'exploiter plus loin (voir chapitre 5).

Définition 2

Vue virtuelle*: est une définition d'une relation logiquement construite à partir des tables de BD ou d'autres vues.*
Vue matérialisée (VM) *utilisée pour pré-calculer et stocker des données agrégées. Elle peut également être utilisée pour pré-calculer des jointures avec ou sans agrégation. Ainsi, les VM sont avantageuses pour des requêtes avec des jointures coûteuses et agrégations. Une fois sélectionnées, toutes les requêtes seront réécrites en fonction de ces dernières. Une réécriture de la requêtes q à l'aide de vues est une expression de requête q' faisant référence à ces vues. Ce processus est fait par l'optimiseur de manière transparente. La sélection de la meilleure réécriture d'une requête donnée est dirigée par les coûts (voir figure 3.12)*

L'administrateur BD ne peut pas matérialiser toutes les vues possibles, puisqu'il est contraint par certaines ressources comme l'espace disque, le temps de maintenance des vues et de la réécriture des requêtes. La sélection des vues devient ainsi un problème défini comme suit :

Formalisation 2 (*Problème de sélection de vues à matérialiser (PSV)*)

Soit :

- *$Q = \{q_1, q_2, ..., q_n\}$ un ensemble de requêtes fréquentes, avec pour chaque q_i une fréquence d'accès aux données f_i $(1 \leq i \leq n)$;*
- *C un ensemble de contraintes à l'exemple du coût de stockage et/ou de maintenance ;*
- *Un ensemble de besoins non fonctionnels BnF à l'exemple du coût de traitement, énergie consommée, voire coût de stockage et/ou de maintenance* [38]

Le problème consiste à sélectionner un ensemble de vues à matérialiser en respectant les contraintes C et en satisfaisant les BnF.

Plusieurs travaux ont été proposés pour la résolution du PSV, et qui se distinguent les uns des autres par rapport aux dimensions suivantes (résumées dans la figure 3.14).

1. Structures de données à utiliser pour représenter l'espace de recherche, souvent très large, des vues candidates à la matérialisation. Nous pouvons citer à titre d'exemple :

38. Les coûts de stockage et/ou de maintenance peuvent être considérés comme contraintes ou comme BnF (objectif)

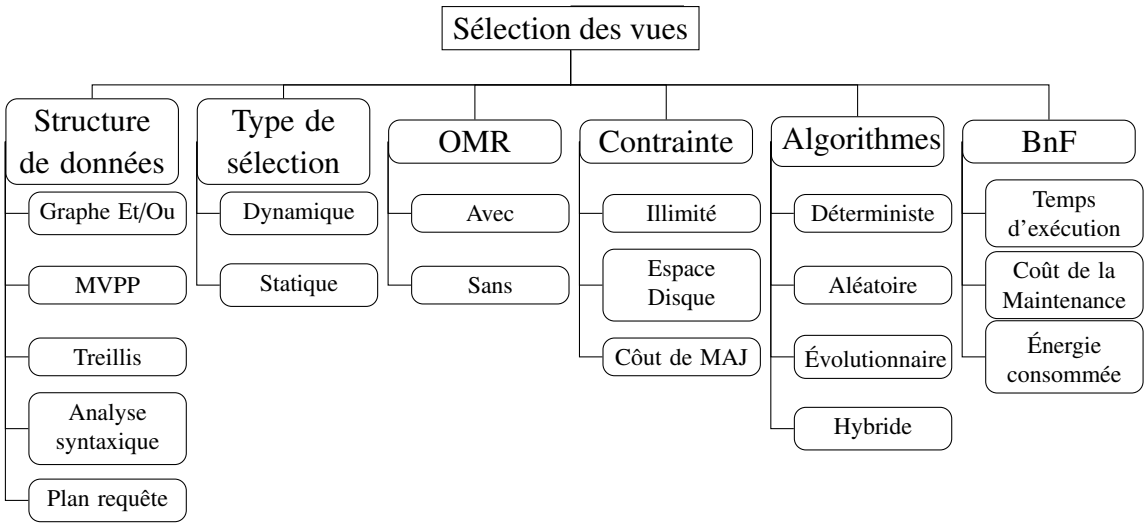


FIGURE 3.14 – Classification des techniques de sélection de vues à matérialiser [9]

- *multi view processing plan* (MVPP) : graphe orienté acyclique, dont la racine représente les résultats des requêtes, les feuilles représentent les relations de base et les noeuds intermédiaires représentent les différentes opérations algébriques (voir figure 3.13) ;
- graphe Et/Ou orienté acyclique, composé de deux types de noeuds : (i) opération comme la sélection, jointure, projection, (ii) équivalence regroupe un ensemble d'expressions logiques équivalentes.

2. BnF : le temps d'exécution était le premier BnF convoité. Ensuite, et parcequ'une VM stocke des données sujettes à changement, une mise à jour (entière ou incrémentale) s'impose afin de maintenir la synchronisation entre les deux structures (tables d'origine et VM). Ce processus a un coût pouvant être très élevé, et peut donc être considéré comme un BnF. Et plus récemment, nous avons intégré la consommation d'énergie comme BnF à titre égal [156].
3. La sélection des vues peut se baser ou pas sur l'optimisation multi-requêtes. Celle-ci peut en effet offrir le plan global optimal qui servirait de base à la sélection de structures physiques. À titre d'exemple, le travail pionnier de Yang et al. [195], qui fusionne d'abord les plans (logiques) des jointures de chaque requête dans un MVPP (plan global) optimal. Les noeuds les plus bénéfiques de ce dernier seront matérialisés. Ce point incarne parfaitement l'interaction entre les optimisations logiques et physiques.
4. La traditionnelle formalisation du PSV figeait la charge des requêtes en entrée. Une tendance plus réaliste considère une arrivée dynamique des requêtes.
5. Les premiers travaux ne considéraient aucune contrainte, mais la taille croissante des VM, a imposé l'intégration de leur espace de stockage comme une contrainte. Idem pour le coût de la maintenance.
6. Plusieurs types d'algorithmes ont été proposés dans le cadre du PSV, allant des algorithmes déterministes aux évolutionnaires.

5 Conclusion

Ce chapitre a été dédié à la définition d'une méthodologie de conception de BD orientée gestion de variabilité. Celle-ci est basée sur la technique des LdP logiciels, car elle est réputée pour être efficace dans la maîtrise de la variabilité des BD. Le framework résultant permet la modélisation d'un grand ensemble de BD comme une ligne de produits. Cela en vue de dériver des BD prêtes à être déployées à partir de composants communs et d'autres variables mais configurables, selon les besoins de la BD en question. Nous avons suivi les étapes prédéfinies pour l'adoption de cette technique, en commençant par l'analyse de la variabilité de la conception des BD. Celle-ci implique la modélisation de la variabilité du problème (conception des BD) à l'aide des modèles des features et l'identification des interdépendances. Nous avons abordé par la suite, l'implémentation de cette variabilité dans un cadre général. En effet, nous avons présenté le framework associé à la dérivation orientée BnF des BD prêtes à être déployées. Cette dérivation est assimilée au problème de la recherche du meilleur chemin dans l'arbre global des features. Vu l'étendue de ce framework, nous avons choisi d'étudier un cas restreint. La dernière section à été donc dédiée à la présentation de ce cas d'étude dont l'arbre associé est présenté dans la figure 3.15. Il porte sur la variabilité de la conception logique, car la revue de littérature du chapitre 2 l'a désignée comme étant la phase la moins étudiée parmi les phases du processus de conception.

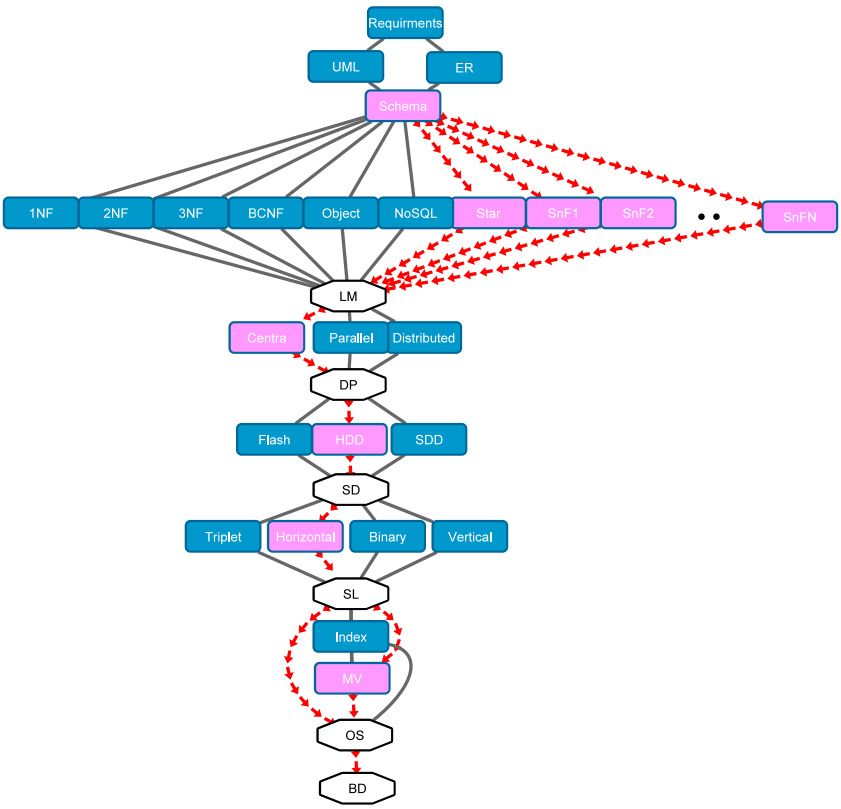


FIGURE 3.15 – Arbre associé à notre cas d'étude de la variabilité logique

Nous avons d'abord présenté la feature choisie pour incarner cette variabilité (*normalisation* : `{star, SnF1, SnF2, ..., SnFn}` de l'arbre illustré par la figure 3.15) et sa cause (les corrélations). Puis, nous avons présenté les features que nous avons choisies pour montrer l'impact de cette variabilité logique. Il s'agit des optimisations logique et physique. Dans un premier temps, nous allons étudier dans le chapitre 4 suivant, cette variabilité logique incarnée par la normalisation et basée sur les corrélations, ainsi que son impact et l'évaluation de ce dernier sur l'optimisation logique (passage du `SL` à `OS` de la figure 3.15 qui représente la non considération des structures d'optimisation physique). Dans un second temps, nous allons étudier un peu plus loin dans le chapitre 5, l'impact de cette variabilité sur l'optimisation physique des BD (passage du `SL` à `MV` de la figure 3.15 qui représente la considération des vues matérialisées incarnant l'optimisation physique).

Chapitre 4

Impact de la variabilité sur l'optimisation logique

Sommaire

1 Introduction 121

2 Génération des schémas logiques ROLAP 123

2.1 Décomposition des dimensions et génération des schémas 124

2.2 Peuplement des tables et calcul de la taille des tables 126

3 Impact de la variation logique sur l'optimisation logique 130

3.1 Réécriture des requêtes 130

3.2 Hypothèses relatives au développement des MdC 132

3.3 Modèle de coût orienté temps d'exécution 133

3.4 Modèle de coût orienté consommation d'énergie 137

3.5 Modèle de coût orienté espace de stockage 142

3.6 Sélection de la meilleure variante 142

4 Validation et expérimentations 144

4.1 Génération des schémas logiques 144

4.2 Évaluation de l'impact en fonction du temps d'exécution 145

4.3 Évaluation de l'impact en fonction de l'énergie 149

4.4 Évaluation de l'impact en fonction de l'espace de stockage . . . 154

5 Conclusion 155

1 Introduction

L'effet de l'omniprésence de la variabilité sur l'ensemble du cycle de vie de conception de BD et la difficulté de la considérer sur cet ensemble, nous impose de procéder par étape. Ainsi, nous consacrons le reste du manuscrit à l'étude d'un cas restreint qui doit, tout de même, montrer l'intérêt de la gestion holistique de la variabilité de conception. Pour de nombreuses raisons, celui-ci a été choisi pour traiter essentiellement la variabilité au sein de la phase logique (peu étudiée) qui vise à mettre à la disposition du concepteur l'ensemble des choix des schémas logiques et à en choisir le meilleur en matière de BnF (voir section 3.4). Nous rappelons qu'il existe deux scénarios pour l'application de nos propositions. Le premier scénario concerne la conception d'un nouvel ED ce qui est caractérisé par l'absence des données (pouvant être remplacées par des statistiques, ou des ED mocks) et l'absence des requêtes pouvant être déduites à partir des besoins [21, 102, 94]. Le deuxième scénario concerne un ED déjà déployé (en production). Les données ainsi que les requêtes réelles sont alors disponibles. Le nouveau schéma est peuplé à partir de l'ancien (migration de données).

Rappelons que cette variabilité est associée au paradigme "un problème à plusieurs solutions" (1-N). Si nous projetons ce point de vue sur la phase logique, les questions auxquelles nous devons répondre sont :

1. est-il possible d'énumérer les *N* schémas logiques possibles pour une application de BD à concevoir et étudier leur impact sur l'optimisation logique des requêtes ?
2. comment choisir le meilleur schéma satisfaisant un ou plusieurs BnF ?

L'énumération de tous les schémas logiques peut se faire en exploitant les corrélations que nous avons indiquées dans le chapitre précédent. Pour répondre à la deuxième question, plusieurs scénarii sont possibles : l'adoption d'une (i) solution aléatoire (par affinité) dans le choix du schéma logique sans aucune base fondée, (ii) solution dirigée par des règles prédéfinies, (iii) solution dirigée par l'expérience du concepteur, et (iv) solution basée sur un modèle de coût. Dans cette thèse, nous avons choisi la solution dirigée par les MdC. Ces derniers étant largement utilisés dans la phase d'optimisation de requêtes dans les BD. Notons que les différents algorithmes proposés sont amenés à être pluggés dans nos FM.

Ce chapitre vient donc répondre à nos deux questions, en commençant par l'énumération des *N* schémas logiques possibles (question *1*) ainsi que leur génération et peuplement automatiques. Ce processus est détaillé dans la section 2. Quant au processus de la sélection du choix (question *2*), celui-ci est fortement lié à l'évaluation de l'impact de chaque schéma candidat. En effet, chaque schéma a un impact intraphase (local) que nous avons matérialisé par l'optimisation logique. Celle-ci restructure les plans d'exécution des requêtes avant d'en choisir le meilleur. Ces plans étant directement impactés par la variation du schéma logique, nous abordons cet impact (réécriture des requêtes) dans la première partie de la section 3. Le schéma ayant le meilleur impact sur l'optimisation logique en fonction de BnF sera sélectionné. Cela revient à choisir le chemin ayant le poids minimal de l'arbre illustré par la figure 4.1 où les sché-

mas logiques possibles sont représentés par les features `{star, SnF1, SnF2, ..., SnFn}` et le passage du SL à OS représente la non considération des structures d'optimisation physique (représentation implicite de l'optimisation logique dans l'aperçu).

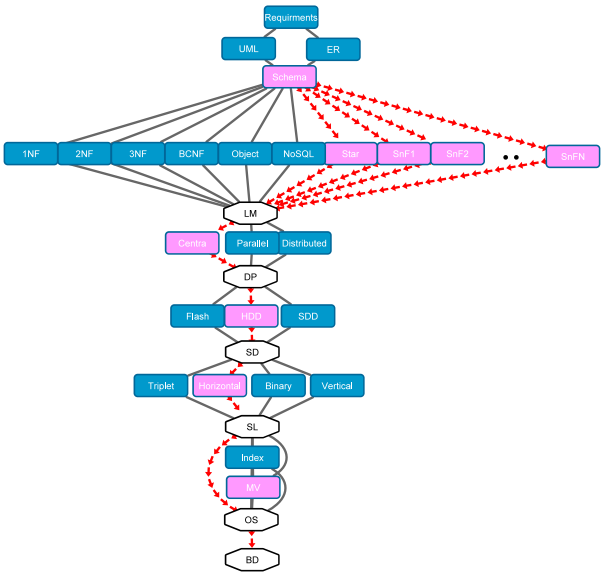


FIGURE 4.1 – Aperçu de l'arbre associé à l'impact de la variabilité logique sur l'optimisation logique

La pondération de ces chemins se fait au moyen des différents modèles de coûts que nous décrivons dans la même section. Ces derniers permettent l'évaluation de cet impact, en fonction du temps d'exécution, stockage mais aussi de l'énergie. En effet, les spécialistes des technologies BD ont longtemps accordé la plus grande attention à l'optimisation du temps d'exécution, suivi par le stockage en ignorant l'énergie. Le contexte actuel oblige plus que jamais ces spécialistes à intégrer la dimension énergétique, car les SGBD sont l'un des principaux consommateurs d'énergie dans les centres de données. Le conseil de défense des ressources naturelles rapporte que la consommation de ces derniers s'élève à plus de 91 milliards de kilowatt-heures en 2013, l'équivalent de 34 grandes centrales à charbon, avec un coût annuel de 9 milliards de dollars et des émissions de *CO2* de plusieurs dizaines de millions de tonnes par an [39]. Des statistiques qui devraient continuer à augmenter avec l'utilisation massive de données numériques. Face à cette situation, l'attention des spécialistes BD se tourne de plus en plus vers des solutions matérielles et logicielles écologiques (*Green IT*). Dans cette optique, le choix du schéma logique en terme de l'énergie consommée par ces requêtes peut être considéré comme l'une des solutions logicielles. Enfin, la section 4 démontre la faisabilité et la validation de notre méthode avant de conclure le chapitre avec la section 5.

39. `https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy`

2 Génération des schémas logiques ROLAP

L'une des étapes clés dans la gestion de la variabilité, est de prendre en considération les choix disponibles et pertinents, et de les exposer au concepteur de manière exhaustive. Bien qu'ils peuvent dissimuler la bonne alternative, certains choix sont habituellement écartés par le concepteur soit par exigence ou par un retour d'expérience, soit, de manière moins fondée, par affinité ou par ignorance. Or, plus le degré de la variabilité augmente, plus la probabilité de survenance de ce scénario est importante, ce qui est le cas de la conception des BD (voir chapitre 1). En effet, cela a motivé le passage de l'approche classique "*1-1*", qui signifie qu'un problème est associé à une solution ou à un petit nombre de solutions, à l'approche "*1-N*" qui signifie qu'un problème est associé à une multitude de solutions. Nous allons illustrer ceci dans notre cas d'étude qui porte sur la variabilité logique. Nous y montrons que même si plusieurs problèmes de performance peuvent être résolus avec un ajustement fin du modèle physique, quelques-uns sont causés par la non-optimisation du schéma logique [78, 104, 113].

Par ailleurs, la conception des ED est basée sur la notion des schémas multidimensionnels en étoile ou en flocon de neige [104]. Rappelons que le schéma en étoile est caractérisé par une table de faits centrale entourée d'un ensemble de tables de dimensions. Lorsque ces dimensions sont de plus divisées (normalisées) en tables de sous-dimensions, on parle d'un schéma en flocon de neige (voir paragraphe 1.4.3.2). Le choix entre ces deux modélisations a été sujet à débat [104, 113].

Il est important de souligner que, contrairement aux idées reçues, un pur schéma en étoile peut parfois entraîner de sérieux problèmes de performance. Cela peut se produire par exemple, quand une table de dimension dé-normalisée devient très grande et pénalise la jointure en étoile. À l'inverse, une petite table de sous-dimension n'engendre pas un coût important de jointure car elle peut être stockée de façon permanente dans une mémoire tampon. Si ce n'est pas le cas, les "petites" clefs (comme un entier avec un *auto-incrément*) impliquées dans les dites-jointures, font que l'exécution reste rapide. En outre, une structure en étoile existe toujours au centre d'une structure flocon de neige, ce qui permet l'utilisation d'une jointure en étoile efficace pour satisfaire une partie d'une requête. Enfin, certaines requêtes n'accèdent pas aux données des tables de sous-dimensions externes. Ces requêtes s'exécutent alors plus rapidement car elles interrogent un schéma en étoile qui contient des tables de dimensions plus petites. À cela s'ajoute les avantages classiques de la normalisation (cohérence, moins de redondance, gain d'espace de stockage) [66, 71].

De ce fait, un schéma en flocon de neige est plus performant qu'un schéma en étoile dans certaines circonstances, et vice versa [122]. Au final, cela revient à chercher le meilleur compromis entre la performance liée au temps d'exécution des requêtes et la taille de stockage (BnF). Toutefois, d'autres BnF voient le jour vu le contexte évolutif de conception de BD, et cela change la donne par rapport à ce compromis classique. Nous allons étudier l'exemple de la consommation d'énergie.

Par ailleurs, afin de mettre à la disposition du concepteur la panoplie des schémas logiques possibles, du plus dé-normalisé (étoile) au plus normalisé possible (flocon de neige), nous montrons dans la section suivante, comment exploiter les corrélations pour générer ces différents schémas. Ces corrélations font partie des connaissances du concepteur qui vient de concevoir le schéma conceptuel de la BD en question. Dans le scénario où le concepteur veut étudier l'évolution de son schéma de sa BD (c'est à dire qu'il a des données sur lesquelles il peut se baser), il peut utiliser les algorithmes d'extraction de corrélations (voir section 3.4.2).

2.1 Décomposition des dimensions et génération des schémas

Soit $\mathcal{ED} = \{F, D_1, D_2, ..., D_n\}$, le modèle conceptuel multidimensionnel de la BD, où F désigne la table des faits, et D_i les tables de dimensions. Nous déduisons les résultats suivants :

— chaque dimension D_i ayant h attributs hiérarchiques, peut être décomposée 2^{h-1} fois. Soit *localisation* = {*Continent*, *Nation*, *Cité*} une hiérarchie, il y aura donc 4 (2^{3-1}) possibilités de décomposition de la table, comme illustré dans la figure 4.2 ;

— concernant l'ensemble des dimensions $(D_1, D_2, ..., D_n)$, il y aura comme décompositions possibles le produit des possibilités de chaque dimension : $\prod_{\{d=1\}}^{n} 2^{h_d-1}$. Si l'on suppose avoir 3 tables de dimensions avec chacune 2,3 et 4 attributs hiérarchiques respectivement, il y aura un total de $2^{2-1} \times 2^{3-1} \times 2^{4-1} = 64$ alternatives.

Pour pouvoir générer les différentes décompositions possibles pour une hiérarchie, nous proposons l'algorithme 1. Il repose sur la construction d'un arbre. Chaque noeud est composé d'une chaine de caractères fixe `"fix"` non-décomposable, et d'une autre variable `"var"` décomposable. Chaque caractère correspond à un attribut hiérarchique. La figure 4.3 déroule l'algorithme 1 sur un exemple d'une dimension qui possède 4 attributs hiérarchiques $\{a_1, a_2, a_3, a_4\}$ et devrait donc avoir 8 (2^{4-1}) décompositions possibles.

Il va falloir maintenant générer les différents schémas correspondants à ces différentes décompositions. Nous proposons alors l'algorithme 2 qui consiste à créer les tables pour chaque décomposition et calculer le produit cartésien de ces tables.

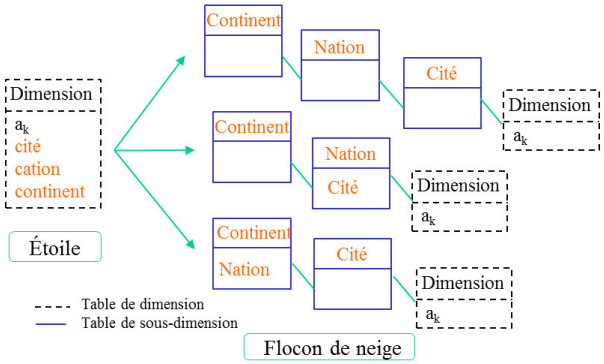


FIGURE 4.2 – Les décompositions possibles pour une hiérarchie de trois niveaux

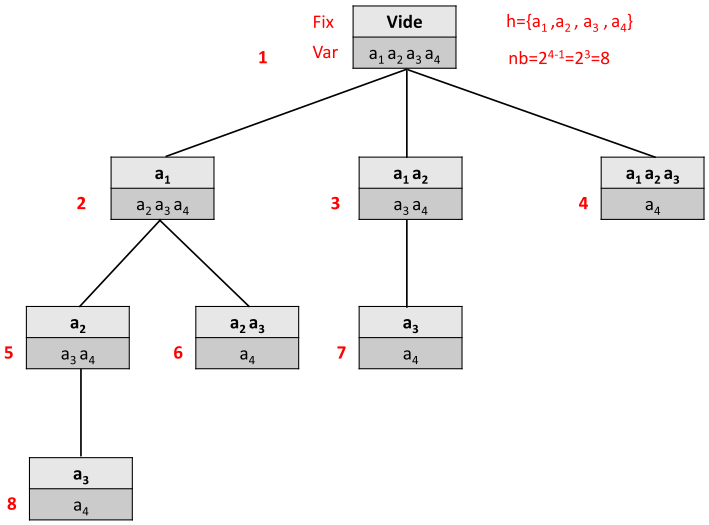


FIGURE 4.3 – Exemple de l'arbre pour la génération des décompositions possibles

Algorithm 1 Génération des différentes décompositions possibles d'une hiérarchie

Input: Une hiérarchie qui contient n attributs hiérarchiques : $h = \{a_1, a_2, \ldots, a_n\}$
Output: L'ensemble des différentes décompositions

```
1:  int nbFils = 0;
2:  String fix ← "";
3:  String var ← "a1a2 ... an";
4:  procedure CreerArbre
5:      if var.length() != 1 then nbFils = var.length()-1;
6:          for i := 1 → var.length() − 1 do
7:              String f = var.substring(0, i);
8:              String v = var.substring(i);
9:              Arbre sousArbre = new Arbre(f, v);
10:             this.Fils[i-1] = sousArbre;
11:             this.Fils[i-1].CreerArbre();
12:         end for
13:     end if
14: end procedure
15: procedure Arbre(String fix, String var)
16:     this.fix = fix;
17:     this.var = var;
18:     this.Fils = new Arbre[var.length()-1];
19:     this.nbFils = 0;
20: end procedure
```

Algorithm 2 Génération des différents schémas logiques d'un ED

Input: ED un schéma d'entrepôt de données en étoile
Output: Un ensemble des différents schémas logiques en flocon de neige correspondants

```
1: for chaque table de dimension dans ED do
2:     Générer l'arbre des différentes décompositions des attributs hiérarchiques ;
3:     for chaque décomposition do
4:         Créer les nouvelles tables de sous-dimensions ;
5:         Mettre à jour les attributs ;
6:         Ajouter la décomposition courante à la liste correspondante à la dimension en cours ;
7:     end for
8: end for
9: Calculer le produit cartésien des différentes décompositions hiérarchiques de chaque dimension ;
10: Construire le nouveau schéma correspondant à chaque élément appartenant à ce produit cartésien ;
```

Maintenant que nous avons généré les schémas logiques possibles, nous devons expliquer comment redistribuer automatiquement les anciennes données sur les nouvelles tables, si le concepteur se trouve dans le scénario où un ED existe déjà. Nous devons également calculer les nouvelles tailles des tables de dimension et de sous-dimension pour le calcul de nos MdC comme expliqué dans la section suivante.

2.2 Peuplement des tables et calcul de la taille des tables

La décomposition des tables de dimensions implique une redistribution de leurs lignes, et donc un changement de la taille de ces tables. Cette nouvelle information est cruciale pour calculer nos modèles de coût et migrer des données si le concepteur se retrouve dans le deuxième scénario (évolution de l'ED). Dans notre approche, nous manipulons deux types de tables : (i) la table de dimension d'origine (bordure en pointillé dans la figure 4.2), et (ii) les tables de sous-dimension nouvellement créées (bordure en bleu continu dans la figure 4.2). Notons que la taille d'une table est calculée en fonction de deux paramètres liés au nombre et à la taille des tuples (ou lignes) comme suit :

$$Taille(T) = tailleTuple \times nbTuples$$

Ci-dessous, nous expliquons la façon dont ces paramètres changent pour les deux types de table au sein de chaque schéma logique généré. Nous distinguons deux principaux scénarios : soit les tables sont au moins en deuxième forme normale (2FN), soit elles sont en première forme normale (1FN).

2.2.1 Tables en 2FN (ou plus)

Le calcul se fait en fonction du type de la table. En ce qui concerne les tables de dimensions d'origine, le nombre de lignes ne change pas car nous avons supposé que la table d'origine est au moins en 2FN. En ce qui concerne la taille des tuples, celle-ci change car il y a moins d'attributs qu'avant la normalisation. Ainsi, la nouvelle taille de la dimension en question est :

$$\begin{aligned} Taille(D'_i) &= Taille(D_i) - \sum_{k=1}^{n_{d_i}} (Taille(h_k)) + Taille(a_{FK}) \\ &= nbTuples \times (tailleTuple - (Taille(h_k)) + Taille(a_{FK})) \end{aligned}$$

tel que (h_k) désigne l'un des n_{d_i} attributs hiérarchiques de la dimension D_i et a_{FK} représente l'attribut de la clé étrangère, qui permet de lier la table de dimension à ses tables de sous dimension.

Passons maintenant aux tables de sous-dimension nouvellement créées. En premier lieu, la taille de la ligne est la somme des tailles des attributs hiérarchiques h_i de la sous-dimension concernée (SD), en plus de la taille de la clé étrangère (si celle-ci est présente) :

$$tailleTuple = \sum_{i=1}^{n_{sd}} (Taille(h_i)) + Taille(a_{FK})$$

Le calcul de tuples est plus complexe car il dépend du type de corrélation entre les attributs formant la hiérarchie. On distingue trois scénarios principaux :

1. la *hiérarchie* (H) qui définit un ordre partiel entre les attributs hiérarchiques est le seul type de corrélation qui puisse les lier. En d'autres termes, pour chaque valeur de l'attribut du niveau supérieur (le *tout* (whole)), il existe uniformément les mêmes valeurs de domaine pour l'attribut du niveau inférieur (le *partie* (part)). Un exemple concret de cette catégorie, est la hiérarchie `Date {année, mois, jour de la semaine}`, où quel que soit l'*année* (respectivement, le *mois*), le nombre de valeurs (cardinal) de *mois* est toujours égal à 12 (respectivement, le cardinal de *jour* est toujours égal à 7). Dans ce cas, le nombre de lignes consiste en le produit cartésien de la cardinalité du domaine de chaque attribut hiérarchique de la sous-dimension actuelle :

   $$nbTuples = (\prod_{i=1}^{n_{sd}} |domaine(h_i)|) = domaine(h_1) \times domaine(h_2) \times \cdots \times domaine(h_{n_{sd}})$$

   Par exemple, en considérant deux ans seulement, l'application numérique sur l'exemple de la hiérarchie `Date {année, mois, jour}` donne $(2 * 12 * 7)$ combinaisons possibles.
2. Outre la corrélation de type *hiérarchie* (H), il existe une dépendance fonctionnelle (DF) [40] entre les attributs. Ceci signifie qu'il est possible de faire correspondre à une valeur de l'attribut du niveau supérieur, un seul ensemble déterminé de valeurs de l'attribut du niveau

40. les ensembles de valeurs de l'attribut du niveau inférieur, qui peuvent être associées aux valeurs de l'attribut du niveau supérieur, sont disjoints

inférieur. Conformément à l'exemple précédent, nous prenons une hiérarchie `Date` plus complexe {`année, saison, mois, jour`}. Nous remarquons que l'attribut `saison` possède une corrélation de type *DF* avec l'attribut `mois`. En effet, toutes les valeurs de `saison` peuvent être liées à seulement trois mois spécifiques. Considérons un exemple plus concret, où la corrélation de dépendance est entièrement appliquée : l'exemple de la hiérarchie `Emplacement {continent, pays, ville}` dans laquelle, à chaque valeur d'attribut de niveau supérieur, seulement un ensemble spécifique de valeurs d'attribut de niveau inférieur peut être associé (par exemple, la valeur `Europe` de l'attribut `continent` possède un ensemble spécifique de valeurs de l'attribut `Pays` telles que la France, l'Espagne et l'Allemagne, ce qui est différent de l'ensemble des valeurs associées à la valeur `Amérique` de l'attribut `continent`. Dans ce cas, le nombre de lignes est :

$$\begin{aligned} nbTuples &= \max(|domaine(h_i)|) \\ &= \max(|domaine(h_{SD_1})|, |domaine(h_{SD_2})|, \ldots, |domaine(h_{SD_{n_{SD}}})|) \end{aligned}$$

Considérant 2 valeurs de `continent`, 4 valeurs de `pays` et 8 de `ville`, le nombre de lignes possibles serait égal à $nbRows = \max(2, 4, 8) = 8$. Notons également que l'application numérique pour la hiérarchie `Date {année, saison, mois, jour}`, où `saison` dépend fonctionnellement de `mois`, donne le résultat suivant si l'on considère que |domaine(année)|=2 :

$$\begin{aligned} 2 \times \max(|domaine(h_i)|) \times 7 &= 2 \times \max(|domaine(saison, mois)|) \times 7 \\ &= 2 \times 12 \times 7 \text{ combinaisons possibles au lieu de } 2 \times 4 \times 12 \times 7 \end{aligned}$$

3. Pour le dernier scénario, nous considérons les corrélations de type hiérarchique (*H*) et multivalué (*DM*) [41] entre les attributs. Prenons l'exemple de la hiérarchie *nationalité* suivante: {`nationalité, langues parlées`}, dans laquelle deux nationalités différentes peuvent partager les mêmes langues. Ainsi, le nombre de lignes est approximativement égale à :

$$nbTuples = |domaine(h_i)| \times \delta(DM)$$

Définition 3 ($\delta(DM)$)

Si l'on considère une corrélation de type DM entre deux attributs A et B, tel que A est l'attribut dominant et B est l'attribut dépendant, alors $\delta(DM)$ est le nombre moyen des valeurs que B peut avoir pour une valeur de A. Par conséquent, la valeur de $\delta(DM)$ doit être inférieure à la cardinalité du domainee de l'attribut dépendant ($\left|domaine(h_{dependant})\right|$)

Par exemple, sachant qu'il y a une corrélation *DM* entre les attributs `nationalité` (*A*) et `langue parlée` (*B*), et que chaque nationalité peut avoir en moyenne trois langues parlées possibles alors $\delta(MD) = 3$.

41. Contrairement au cas précédent, les ensembles de valeurs de l'attribut de niveau inférieur ne sont pas disjoints.

Il est à noter que ce cas peut, dans des cas particuliers, être considéré comme faisant partie du premier scénario. Il s'agit du cas où $\delta(MD)$ est aussi grand que le cardinal du domaine d'attribut concerné, en d'autres termes, les ensembles de valeurs des attributs de niveau inférieur sont totalement non-disjoints (les mêmes).

Nous notons également qu'en cas où les données de base sont présentes (deuxième scénario), la requête suivante permet de vérifier le nombre de tuples de la table sous-dimension, et transitivement le type de corrélation qui existe entre deux attributs.

```
1  SELECT distinct(1er attribut), 2ème attribut
2  FROM dimension
3  ORDER BY 1er attribut ;
```

2.2.2 Tables en 1FN

Contrairement au cas précédent, le nombre de lignes des tables de dimension d'origine sera également affectées. En effet, une instance donnée d'une relation pourrait apparaître plus d'une fois [42] provoquant une redondance, qui peut se prolonger au cardinal de l'attribut responsable. Nous pouvons nous inspirer du scénario précédent, et définir $\delta(Attribut)$ comme le nombre maximal de valeurs appartenant à ce domaine d'attribut, et qui peut être associé à une instance (individuelle) de la relation concernée. Ou tout simplement suggérer que $\delta(Attribut)$ est égal à la moyenne de son cardinal de domaine. Dans les deux cas, il s'agit d'une approximation surestimée, où le nombre de lignes devient :

$$nbTuples' = \lceil nbTuples/\delta(Attribut) \rceil$$

Il ressort ainsi que la connaissance du concepteur du domaine est fondamentale pour la détermination de la taille des tables. Cependant, cette connaissance est facile à acquérir, car elle concerne les paramètres suivants : taille des attributs (ce qui implique la taille du tuple), la taille du domaine de l'attribut, le nombre de lignes pour les tables d'origine, le type de corrélations reliant les attributs hiérarchiques et éventuellement la mesure de delta. Lors de la conception de tout système d'information, le concepteur doit avoir une idée approximative sur ces valeurs. Dans le scénario où le concepteur veut étudier l'évolution de son schéma de son ED (c'est-à-dire qu'il a des données sur lesquelles il peut se baser), il peut interroger son ED par de simples requêtes afin d'obtenir ces différentes informations.

Maintenant que nous avons généré les schémas logiques possibles issus de la normalisation, calculé les nouvelles tailles des tables et éventuellement les avoir peuplé, nous allons dans un premier temps étudier l'impact sur l'optimisation logique (réécriture des requêtes), puis nous procédons à l'évaluation de cet impact via la présentation de nos modèles de coûts dans la section suivante.

42. Un attribut dépend d'une partie de la clé seulement.

3 Impact de la variation logique sur l'optimisation logique

Après avoir délimité l'espace de recherche, il est (logiquement) nécessaire de définir une stratégie fondée de sélection, plus fiable que la traditionnelle. On tombe sur un problème classique d'optimisation où la stratégie exhaustive qui évalue empiriquement (réellement) l'ensemble des solutions candidates est de loin la plus sûre, mais la plus écartée. En effet, il est inconcevable de déployer physiquement toutes les configurations afin de réellement évaluer les effets des choix de conception et des changements (évolutions) possibles, d'où le recours aux modèles de coût pour une évaluation théorique proche de la réelle, à des fins de prédiction.

D'un autre côté, nous avons choisi d'évaluer dans un premier temps l'impact des schémas candidats sur l'optimisation logique en matière de temps d'exécution, énergie et stockage. Rappelons que ce type d'optimisation agit sur les plans d'exécution des requêtes en les restructurant en vue d'une meilleure performance. Par conséquent, nous commençons d'abord par présenter l'impact de la variabilité du schéma logique sur cette optimisation. Cet impact est incarné par la réécriture des requêtes d'origine et la définition du meilleur plan d'exécution correspondant, suite à la variation du schéma logique initial (en étoile). Ensuite, nous présentons nos MdC utilisés pour évaluer chaque schéma ainsi que la méthode de sélection du meilleur schéma.

3.1 Réécriture des requêtes

La charge des requêtes $Q = \{q_1, q_2, ..., q_m\}$ est constituée de l'ensemble des requêtes les plus fréquentes/importantes à soumettre. Le concepteur doit en posséder des connaissances qui peuvent êtres issues soit de la phase de conception où les requêtes peuvent se déduire à partir des besoins utilisateurs analysés [21, 102, 94], soit bien après le déploiement (charge réelle) pour l'étude de l'évolution de la BD.

Ces requêtes doivent être réécrites en fonction du schéma logique sous-jacent. En effet, les attributs référencés par les requêtes peuvent se déplacer vers de nouvelles tables pendant l'éclatement des dimensions en sous-dimensions. L'algorithme 3 suivant décrit ce processus de réécriture de requêtes.

Après la réécriture syntaxique de chaque requête, nous devons générer le meilleur plan d'exécution correspondant. Pour ce faire, il existe des règles algébriques prédéfinies à appliquer, comme celle qui consiste à descendre les sélections le plus bas possible dans l'arbre [71]. Cependant, l'ordre le plus délicat qui reste à définir est celui entre les jointures s'il y en a plusieurs. Pour notre étude, nous adaptons les mêmes règles appliquées dans [19, 24] où l'ordre entre les opérations algébriques est le suivant : sélection, jointure, projection et agrégation. Quant à l'ordre des jointures multiples, celui-ci y est basé sur la notion du *minimum share* [19, 24] défini comme suit :

Définition 4 (*Share d'une table de dimension*)

Le share (F, a_k, D) *est défini comme le nombre d'instances de la table des faits* F *pour*

Algorithm 3 Réécriture syntaxique des requêtes en fonction du schéma logique

Entrées: – Le schéma logique de l'entrepôt de données ED ;
– La charge d'origine du schéma en étoile : $Q = \{q_1, q_2, ..., q_m\}$

$Q = \{q'_1, q'_2, ..., q'_m\}$
for chaque requête q_i dans la charge Q **do**
    Extraire les attributs de la clause "*Select*" de q_i ;
    Chercher ces attributs dans les tables du nouveau schéma ED (entrée) ;
    Définir en conséquence les clauses "*Select*" et "*From*" de la nouvelle requête (q'_i) ;
    Extraire les attributs de la clause "*Where*" de q_i ;
    Chercher ces attributs dans les tables du nouveau schéma ED (entrée) ;
    Définir les clauses "*Where*" de la nouvelle requête (q'_i), et mettre à jour sa clause "*from*" ;
    Gérer les extra-jointures résultants de l'éclatement des dimensions ;
end for

chaque instance utilisée de la table de dimension D à travers l'attribut a_k, clé étrangère dans la table des faits. Le share (F, D) est la moyenne des share (T, a_k, D). Autrement dit, le Share d'une table de dimension est le nombre d'instances pour lesquelles la valeur de la clé étrangère dans la table des faits est égale à une valeur de cette clé dans la table de dimension. Notons que dans le cas d'une répartition uniforme des données dans les tables, on a :

$$share_{D_i} = \frac{||F||}{||D_i||} \quad / \quad ||T|| \text{ est la taille en pages de la table } T$$

Exemple : prenons la table de dimension Customer et la table des faits Lineorder de l'exemple de la figure 1; la commande SQL suivante permet de calculer le share (Lineorder, Customer).

```
SELECT AVG(nbr) AS moyenne
FROM (SELECT DISTINCT l_custkey, COUNT(*) AS nbr
FROM lineorder
GROUP BY l_custkey);
```

L'optimisation des jointures en étoile (OLAP) peut être réalisée par la minimisation de la taille des résultats intermédiaires pendant le processus d'évaluation [124]. Ainsi, la première jointure à mettre dans le plan d'exécution est celle qui implique la table de dimension possédant le *minimum share* avec la table des faits. La jointure de cette table minimise la taille des résultats intermédiaires, car un *share* élevé entraine le parcours de la quasi-totalité de la table des faits lors d'une interrogation par une requête. À noter que seules les requêtes en étoile sont prises en charge par nos travaux de référence [19], or nous devons également, gérer les requêtes en flocon de neige. Nous expliquons donc, dans ce qui suit, comment adapter ces travaux pour couvrir nos besoins (section 3.3.4).

3.2 Hypothèses relatives au développement des MdC

Notre espace de recherche est désormais constitué de plusieurs schémas logiques, dont chacun possède sa propre charge de requêtes. Maintenant, il va falloir pondérer chaque schéma avec une métrique témoignant de sa qualité, et qui aiderait donc le concepteur à sélectionner le schéma répondant à ses besoins. Cette métrique est fournie par les MdC que nous allons définir dans la section suivante, mais avant cela, nous présentons les hypothèses que nous avons considérées pour établir ces MdC.

1. L'exécution d'une requête revient à exécuter ses différents opérateurs agencés dans un plan d'exécution (PE), et dont le coût d'exécution ($Cout(op)$) dépend de trois mesures : le temps de chargement et d'écriture de données de/vers le disque ($Cout_{E/S}$), le nombre des cycles processeur nécessaires au traitement des fonctions arithmétiques et de comparaison ($Cout_{CPU}$), et le temps de communication ($Cout_{COM}$) entre les sites du réseau dans les environnements non centralisés.
2. Chaque opérateur est responsable de la transmission de ses résultats à son successeur dans le plan, à travers la mémoire vive.
3. L'environnement de travail est centralisé, la BD est déployée dans une seule station de travail. Par conséquent, le coût de communication est ignoré.

$$
\begin{aligned}
Cout(op) &= Cout_{E/S} + Cout_{CPU} + \cancelto{0}{Cout_{COM}} \\
Cout_{E/S}(q_i) &= \sum_{op \in PE} Cout_{E/S}(op) \\
Cout_{CPU}(q_i) &= \sum_{op \in PE} Cout_{CPU}(op) \\
Cout_{temps_{CPU}} &\lll Cout_{temps_{E/S}} \ldots \blacktriangle
\end{aligned}
$$

4. Nous ne considérons pas des requêtes de définition de données (`CREATE, DROP, ALTER`), ni les requêtes de manipulation de données (`INSERT, DELETE, UPDATE`). Les requêtes considérées sont de types transactionnelles et analytiques (étoile et/ou flocon de neige), ayant la forme générale suivante :

```
SELECT [ DISTINCT ou ALL] colonnes
FROM tables
[WHERE predicats ]
[GROUP BY expression ]
[ HAVING condition ]
[ORDER BY colonnes ]
```

5. Afin de réduire la complexité des calculs des statistiques, plusieurs SGBD présument que la distribution des valeurs des attributs dans les tables, suit une loi uniforme [146].

6. Les approches conventionnelles d'optimisation estiment le coût de chaque requête indépendamment des autres en utilisant des fonctions de coût d'Entrées/Sorties [161].

3.3 Modèle de coût orienté temps d'exécution

Le temps d'exécution communément appelé performance des requêtes, a longtemps été le BnF le plus recherché, et dont l'optimisation a fait l'objet de plusieurs études. Dans ce contexte, nous nous basons sur les travaux de [161, 19, 12] pour la définition d'une version étendue de leur MdC, qui prend désormais en compte des requêtes en flocon de neige aussi bien que des requêtes en étoile (voir paragraphe3.3.4). À noter que nous avons ignoré le $Cout_{CPU}$ dans nos calculs du temps d'exécution, car sa valeur est négligeable devant le $Cout_{E/S}$ qui est de loin le plus dominant (▲). Ainsi, le $Cout_{temps}$ d'un opérateur op peut être représenté soit par le nombre de pages écrites ou lues pendant son exécution, soit par le temps de son exécution en secondes, comme suit :

$$Cout_{temps}(op) = Cout_{E/S}(op) = \begin{cases} P_{op}, & \text{si le coût est représenté par le nombre de pages } E/S \\ T_{op}, & \text{si le coût est représenté par le temps écoulé} \end{cases}$$

$$P_{op} = P_{entree} + P_{interm} + P_{sortie}$$

$$T_{op} = T_{entree} + T_{interm} + T_{sortie}$$

Où P_{entree} et T_{entree} sont respectivement le nombre de pages utilisées et le temps consommé dans la lecture de(s) entrée(s) de l'opérateur en question. P_{sortie} et T_{sortie} sont à l'inverse le nombre de pages et le temps requis pour le transfert des résultats générés à l'opérateur suivant dans l'arbre ou au disque. On suppose que le flux final de sortie est écrit sur disque pour un usage ultérieur. P_{interm} et T_{interm} sont le nombre de pages et le temps requis pour le stockage disque des résultats intermédiaires, et peuvent être nuls pour certains opérateurs. T_{op} peut être directement estimé à partir de P_{op} en calibrant les paramètres ED pour définir le temps moyen de transfert disque d'une page.

$$T_{entree} = \begin{cases} P_{entree} \times t_{disk} & \text{si opérateurs unaires} \\ P_{entree_1} \times t_{disk} + P_{entree_2} \times t_{disk} & \text{si opérateurs binaires} \end{cases}$$

$$T_{sortie} = P_{sortie} \times t_{disk}$$

$$T_{interm} = P_{interm} \times t_{disk}$$

Nous allons estimer le $Cout_{E/S}$ pour chaque type d'opérateur en considérant les trois composantes et en utilisant les paramètres résumés dans le tableau 4.1.

Paramètre	Description
PS	Taille de la page
$\|\|R\|\|$	Taille en pages d'une relation R
$\|R\|$	Cardinalité (nombre de tuples) de la relation R
$L_t(R)$	Longueur moyenne de tuple de la relation R
$L_A(R)$	La longueur moyenne de l'attribut A de la relation R
$f_p(R)$	Facteur de sélectivité du prédicat p défini sur R

TABLE 4.1 – Paramètres des fonctions de l'estimation des coûts

3.3.1 Coût de l'opérateur unaire de sélection

Soit p un prédicat de sélection défini sur la relation R, et dont le résultat est dénoté par $\sigma_p(R)$. Le facteur de sélectivité f_p d'un prédicat p désigne le pourcentage (entre 0 et 1) des tuples le satisfaisant :

$$f_p(R) = \frac{|\sigma_p(R)|}{|R|} \quad \text{(définition du facteur de sélectivité)}$$

$$P_{entree} = \begin{cases} ||R|| & \text{, si } R \text{ est non triée} \\ \prod_{j>0} f_{p_j} \times ||R|| & \text{, si } R \text{ est triée} \end{cases}$$

$$P_{interm} = 0$$

$$||\sigma_p(R)|| = P_{sortie} = \prod_{j>0} f_{p_j} \times ||R||$$

3.3.2 Coût de l'opérateur unaire de projection

Soit A un attribut ou un ensemble d'attributs de la relation R, et dont le résultat de projection est dénoté par $\pi_A(R)$

$$P_{entree} = ||R||$$

$$P_{interm} = \begin{cases} 0, \text{si } P_R \leq M \text{ ou pas de tri effectué} \\ P_R \times log_{M-1}(P_R), \text{sinon} \end{cases}$$

$$\pi_A(R) = P_{sortie} = \begin{cases} ||A||, \text{si les tuples doubles supprimés par l'optimiseur} \\ ||\Pi_A(R)||, \text{sinon} \end{cases}$$

3.3.3 Coût de l'opérateur binaire d'une jointure étoile

La jointure est l'opération algébrique la plus coûteuse. Soit $R_1(A, b) \bowtie R_2(b, C)$ une jointure naturelle entre R_1 et R_2, avec b l'attribut commun, A et C un ensemble d'attributs. La cardinalité dépend des valeurs de b. De manière générale, on pose b_{R1} les attributs b de R_1, b_{R2} ceux de R_2, r_1 un tuple de R_1 et r_2 un tuple de R_2. Si $\|b_{R1}\| > \|b_{R2}\|$, alors la valeur de b de r_2 doit apparaître dans les valeurs de b de R_1. Donc, la probabilité que r_1 et r_2 partagent la même valeur de b est $1/\|b_{R1}\|$ et vice versa. D'autre part, le nombre de comparaisons possibles entre les pairs r_1 et r_2 est de $\|R_1\| \times \|R_2\|$. Soit B un ensemble d'attributs communs et J_{R1R2} la probabilité que deux tuples vérifient la condition de jointure, alors la cardinalité d'une jointure naturelle est définie comme suit :

$$P_{sortie} = \|R_1 \bowtie R_2\| = \begin{cases} = \|(max(|R_1|, |R_2|)/max(|b_{R1}|, |b_{R_2}|))\| \\ = \|(max(|R_1|, |R_2|)/max(|bi_{R1}|, |bi_{R_2}|))\|, \forall bi \in B \\ = (\|max(|R_1|, |R_2|) \times J_{R1R2}\|) \end{cases}$$

Rappelons que dans le contexte des ED en étoile, toutes les jointures (et en particulier la première) passent pat la table (centrale) des faits F. Soit D_j la j-ème table de dimension et n le nombre de jointures dans le plan d'exécution de q_i, l'ordre d'exécution des jointures de q_i est :

$$F \bowtie D_1 \bowtie D_2 \bowtie \cdots \bowtie D_n$$

Notons que nous optons pour un ordre basé sur la notion du "*minimum share*" (voir définition 4) car ce dernier effectue en premier les opérations ayant la plus petite taille des résultats intermédiaires. Ainsi, nous calculons P_{entree} et P_{interm} de toutes les jointures naturelles et les agrégations de q_i en une seule passe par la somme des trois types d'opérations suivants :

— la première jointure, notée PJ;

— la jointure intermédiaire, notée JI;

— les opérations finales de groupement et d'agrégation, notées AG.

Le calcul des coûts de ces opérations (P_{entree} et P_{interm}) dépendent du type d'algorithme de jointure utilisé parmi tri-fusion, boucles imbriquées et hachage. Dans notre étude, nous supposons que la jointure est effectuée par hachage. Par conséquent, les coûts de ces opérations sont calculés comme suit [71] :

$$\begin{aligned} Cout_{PJ}(noeud) &= 3(Yao(F) + Yao(D_1)) \\ Cout_{JI_i}(noeud) &= 3(Yao(Res_i) + Yao(D_{i+1})) \\ Cout_{AG}(noeud) &= 2 \times (AGR + GB)(|Res_n|) \end{aligned}$$

$$\text{Tel que} \quad AGR = \begin{cases} 1, \text{si agrégation} \\ 0, \text{sinon} \end{cases} \quad , \quad GB = \begin{cases} 1, \text{si groupement} \\ 0, \text{sinon} \end{cases}$$

et *Yao* est une fonction définie comme suit :

soit R une relation ou une sous-relation résultant d'un ensemble d'opérations antérieures. La fonction *yao* permet de fournir une estimation du nombre de pages de R à charger, contenant toutes les instances pertinentes [191], en utilisant les lois de probabilité.

Cette fonction utilise trois paramètres pour chaque opération : le nombre de pages de la relation R concernée par la lecture, le nombre de tuples total dans R et le nombre d'instances pertinentes de R. Pour calculer la taille du résultat intermédiaire, qui dépend des opérations précédentes dans le plan d'exécution de q_i, nous utilisons le *share* de la dimension et la proportion des faits retournés par l'opération précédente. Rappelons que $\sigma_p(R)$ est le résultat des prédicats de sélection p sur R.

$$yao(R) = \begin{cases} yao(||F||, |F|, share_{D_1} \times \sigma_p(D_1)) & \textit{Si R est table des faits} \\ yao(||D_i||, |D_i|, \sigma_p(D_i)) & \textit{Si R est table de dimension} \\ yao(||Res_i||, |Res_i|, \sigma_p(Res_i)) & \text{Si R est résultat intermédiaire} \end{cases}$$

$$\text{Tel que} \quad ||\sigma_p(R)|| = \prod_{j>0} f_{p_j} \times ||R|| \quad \text{et} \quad \sigma_p(Res_i) = \lceil \frac{share_{D_{i-1}} \times \sigma_p(D_{i-1}) \times \sigma_p(D_i)}{||F||} \rceil$$

3.3.4 Coût de l'opérateur binaire d'une jointure en flocon de neige

Les requêtes en flocon de neige sont distinguées par l'existence de jointures entre les tables de dimensions et de sous-dimensions de la manière suivant :

$$D \bowtie SD_1 \bowtie SD_2 \bowtie \cdots \bowtie SD_k$$

Cependant, il y a un ordre hiérarchique prédéfini entre les jointures de chaque dimension et ses sous-dimensions. En effet, on commence d'abord par les jointures entre sous-dimensions situées le plus à l'extérieur ($SD_{k-1} \bowtie SD_k$), et on remonte jusqu'à la dernière jointure entre la sous-dimension la plus interne et la dimension ($D \bowtie SD_1$). Nous distinguons alors l'existence de deux types de jointures, et nous proposons d'y étendre la notion du *share* pour estimer le nombre de pages pertinentes, comme suit :

— les premières sous-jointures entre les sous-dimensions, notées Psj avec $share_{SD_k} = \frac{||SD_{k-1}||}{||SD_k||}$. Notons que si les tailles des sous-dimensions sont petites, le *share* aura une valeur neutre de 1 ;

— la dernière sous-jointure entre la dimension et sa première sous-dimension, notée Dsj avec $share_{SD_1} = \frac{||D||}{||SD_1||}$

$$yao(R) = \begin{cases} yao(\|F\|, |F|, share_{D_1} \times \sigma_p(share_{SD_{1_1}} \times \sigma_p(SD_{1_1})) & \textit{Si R table des faits} \\ yao(\|D_i\|, |D_i|, \sigma_p(share_{SD_{i_1}} \times \sigma_p(SD_{i_1}))) & \textit{Si R table de dimension} \\ yao(\|Res_i\|, |Res_i|, \sigma_p(Res_i)) & \text{Si R résultat intermédiaire} \end{cases}$$

Tel que $\|\sigma_p(SD_{i_1})\| = \prod_{j=2}^{k} f_{p_j} \times \sigma_p(SD_{1_j}) \times \cancelto{1}{share_{SD_{1_j}}}$

et $\sigma_p(Res_i)$ reste le même.

3.4 Modèle de coût orienté consommation d'énergie

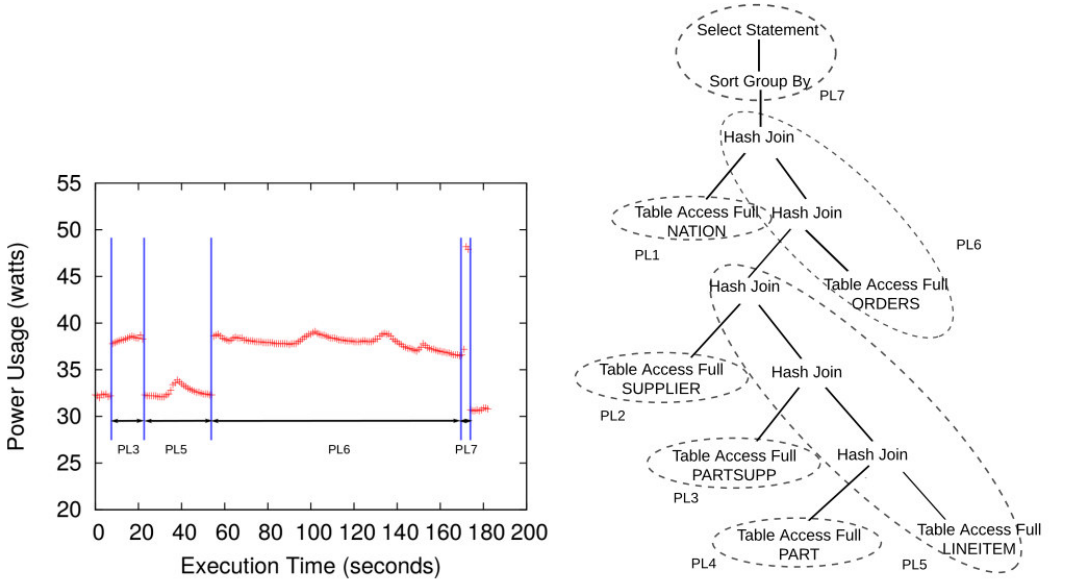
L'estimation de l'énergie dans le contexte des SGBD est une tâche complexe, car elle doit considérer tous les composants matériels des serveurs hôtes. Le coût du processeur ne peut d'ailleurs pas être ignoré comme c'est le cas pour le MdC orienté temps du traitement des requêtes. En effet, l'une des principales difficultés pour définir un MdC énergétique est l'identification des paramètres clés impliqués dans la consommation de l'énergie.

C'est dans cette optique que nous avons adapté une approche d'apprentissage automatique pour la définition de notre MdC, basée sur l'exécution de différentes requêtes simples et complexes, avec différentes échelles de données. Cette stratégie empirique est effectivement utilisée lorsque les paramètres du BnF sont inconnus au préalable. Nous expliquons ci-dessous, les étapes suivies pour le développement de notre MdC en vue d'évaluer les différentes charges de requêtes des différents schémas logiques en étoile ou en flocon de neige. Les mêmes hypothèses que le MdC orienté temps d'exécution sont reprises.

3.4.1 Identification du niveau de modélisation et de la relation entre les paramètres

Nous nous basons sur les travaux de Roukh et al. [154] qui, à travers des expérimentations intensives, ont identifié un niveau de modélisation plus précis pour l'estimation de la dépense énergétique lors de l'exécution des requêtes.

3.4.1.1 Stratégie et mode d'exécution des requêtes. Les expérimentations ont révélé que de nombreuses requêtes ont différents segments de consommation d'énergie au cours de leur exécution. En effet, la consommation d'énergie varie en passant d'un pipeline à un autre, et a généralement tendance à être stable au cours de l'exécution d'un pipeline (voir figure 4.4.a). Celui-ci étant une stratégie implémentée par le SGBD, divise la requête en un ensemble de segments, permettant chacun l'exécution simultanée d'une séquence contiguë d'opérateurs SQL (mode itératif). Ce constat va à l'encontre de l'hypothèse émise par Xu et al. [193] stipulant que le coût de l'énergie est stable pendant l'exécution d'une seule requête dans un système. Afin de gagner en précision, l'estimation doit être effectuée à un niveau plus précis, celui des pipelines.

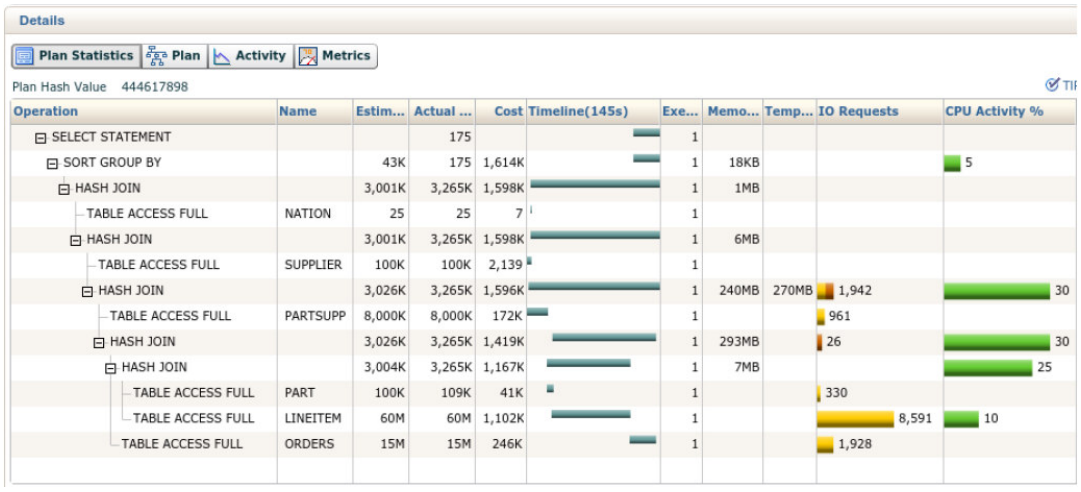


(a) La segmentation de la consommation d'énergie suivant les pipelines

(b) La segmentation du plan d'exécution suivant les pipelines

FIGURE 4.4 – La consommation d'énergie et le plan d'exécution d'une requête exécutée avec pipeline [154]

La détermination des segments énergie-pipelines est effectuée à l'aide du *module de surveillance SQL temps réel* intégré dans le SGBD [165], qui nous donne en temps réel les statistiques à chaque étape du plan d'exécution avec le temps écoulé (voir figure 4.5). Les pipelines sont créés de manière progressive, à partir des opérateurs feuilles du plan comme l'accès séquentiel jusqu'au dernier. À chaque opérateur bloquant [43], comme le tri et le groupage, le pipe-



Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 444617898

Operation	Name	Estim...	Actual ...	Cost	Timeline(145s)	Exe...	Memo...	Temp...	IO Requests	CPU Activity %
SELECT STATEMENT			175			1				
SORT GROUP BY		43K	175	1,614K		1	18KB			5
HASH JOIN		3,001K	3,265K	1,598K		1	1MB			
TABLE ACCESS FULL	NATION	25	25	7		1				
HASH JOIN		3,001K	3,265K	1,598K		1	6MB			
TABLE ACCESS FULL	SUPPLIER	100K	100K	2,139		1				
HASH JOIN		3,026K	3,265K	1,596K		1	240MB	270MB	1,942	30
TABLE ACCESS FULL	PARTSUPP	8,000K	8,000K	172K		1			961	
HASH JOIN		3,026K	3,265K	1,419K		1	293MB		26	30
HASH JOIN		3,004K	3,265K	1,167K		1	7MB			25
TABLE ACCESS FULL	PART	100K	109K	41K		1			330	
TABLE ACCESS FULL	LINEITEM	60M	60M	1,102K		1			8,591	10
TABLE ACCESS FULL	ORDERS	15M	15M	246K		1			1,928	

FIGURE 4.5 – Exemple de rendu du module de surveillance SQL d'Oracle lors de l'exécution d'une requête

43. Un opérateur est bloquant s'il ne peut produire aucun tuple sans lire au moins une de ses entrées.

line en cours se termine, et un nouveau commence. Pour une jointure de hachage, l'opérateur de jointure est inclus dans le pipeline du noeud de calcul de jointure, et le noeud de la construction de la table de hachage est la racine d'un autre pipeline. La Figure 4.4.b montre le plan de pipeline d'une requête.

— Les pipelines *PL1*, *PL2*, *PL3*, *PL4* font la lecture séquentielle des tables de données et construisent leurs tables de hachage en même temps. Ce sont des opérateurs bloquants.

— Le pipeline *PL5* fait la jointure par hachage (opérateur non-bloquant) entre la table `LINE ITEM` et le *PL1* puis le résultat avec les autres pipelines. À la fin, le pipeline construit la table de hachage du résultat final.

— Le pipeline *PL6* fait la jointure par hachage entre *PL5* et la table `ORDERS` puis le résultat avec le *PL1*.

— Le pipeline *PL7* calcul l'opérateur bloquant qui est le tri de *PL6* et affiche le résultat final de la requête.

Dans la même veine, la dépense énergétique augmente à mesure que l'on augmente le degré de concurrence des requêtes, car le système devient plus chargé. La prise en compte du mode d'exécution (mode isolé/concurrent) dans la modélisation est donc cruciale pour prédire avec précision la consommation énergétique d'une charge de requête. En effet, tous les pipelines de toutes les requêtes doivent être déterminés, et l'énergie consommée doit être estimée à chaque transition.

3.4.1.2 Relation entre la taille de données et l'énergie consommée.

L'autre hypothèse contredite par les expérimentations est celle qui stipule que la consommation de l'énergie active d'une requête est positivement liée à sa taille [194]. En effet, plusieurs requêtes interrogeant une petite taille de données peuvent parfois consommer plus d'énergie que les mêmes requêtes interrogeant une plus grande taille de données. La dépense énergétique se révèle être indépendante de la taille de données, et la manipulation de larges fichiers ne se traduit pas forcément par une plus grande consommation énergétique. Cela dépend plutôt des types des requêtes (gourmandes en CPU ou en E/S), et établit une relation *non-linéaire* entre ces deux éléments (consommation énergétique et taille de données).

3.4.2 Construction de notre MdC énergétique

Chaque opérateur algébrique requiert, de la même manière que pour le MdC orienté temps d'exécution de requêtes, des tâches d'E/S et de processeur, ayant respectivement $Cout_{E/S}$ comme nombre (coût) d'E/S et $Cout_{CPU}$ comme coût en fonction de cycles processeur. $Cout_{energie}$ est le coût d'énergie active consommée afin de finir ces tâches. Cependant les $Cout_{E/S}$ et $Cout_{CPU}$

sont d'abord directement prélevés à partir du SGBD pendant la phase d'apprentissage.

$$Cout_{energie}(op) = Cout_{E/S} + Cout_{CPU} + \cancelto{0}{Cout_{COM}} \quad / \qquad Cout_{COM} = 0 \text{ car environnement centralisé}$$
$$Energie = Puissance \times Temps \ldots \blacklozenge$$
$$Puissance = {}^{Energie}/_{Temps}\ ^{44}$$

Contrairement aux travaux précédents sur la modélisation de la consommation énergétique qui opéraient leur modèle au niveau d'opérateurs SQL, notre MdC opère au niveau pipeline. Le type de l'opérateur SQL ou son implémentation physique ne sont pas représentés car leur modélisation est complexe et diffère selon les SGBD. Le débit et la taille de données ne sont pas représentés non plus [110]. Ainsi, notre MdC peut être facilement intégré dans l'optimiseur des SGBD existants.

$$Q = \sum_{i=1}^{n}(q_i) \quad / \qquad Puissance(Q) = \frac{\sum_{i=1}^{n}(Puissance(q_i) \times Temps(q_i))}{Temps(Q)}$$
$$q_i = \sum_{j=1}^{k}(Pl_j) \quad / \qquad Puissance(q_i) = \frac{\sum_{j=1}^{k}(Puissance(Pl_j) \times Temps(Pl_j))}{Temps(q_i)}$$

La construction empirique de notre MdC passe ensuite par la définition du modèle (équation) de régression.

$$Puissance(Pl_j) = \beta_{cpu} \times \sum_{k=1}^{n_j} Cout_{CPU_k} + \beta_{e/s} \times \sum_{k=1}^{n_j} Cout_{E/S_k}$$
$$n_j : \text{nombre d'opérateurs algébriques de } Pl_j$$

La relation entre les paramètres du modèle n'étant pas linéaire (en particulier celle entre la taille des données et l'énergie consommée), la technique de régression linéaire simple, traditionnellement utilisée [194, 110] ne peut donc pas être appliquée. Par conséquent, nous avons utilisé la régression polynomiale, une technique plus appropriée dans le cas de non-linéarité des paramètres.

$$Puissance(Pl_j) = \beta_0 + \beta_1(Cout_{CPU}) + \beta_2(Cout_{E/S}) + \beta_3(Cout_{E/S})^2 + \beta_4(Cout_{CPU})^2 + \cdots + \beta_p(Cout_{E/S})^p \beta_{p'}(Cout_{CPU})^p + \epsilon \qquad \ldots \qquad \bigstar$$

La résolution de cette équation revient à estimer les coefficients de régression β en fonction des données d'apprentissage en utilisant la méthode des moindres carrés. À cet effet, nous

44. Nous notons que les deux termes *énergie* et *puissance* sont employés de façon interchangeable tout au long du manuscrit. Plus précisément, l'énergie est notre fonction objective, et le terme est plutôt employé dans la partie théorique, contrairement à la puissance qui est plutôt réservée à l'usage expérimental car notre appareil de mesure renvoie des valeurs de puissance. Celle-ci peut être convertie en énergie grâce à l'équation (♦).

avons créé notre charge de requête caractérisée par (i) l'interrogation de données appartenant à des schémas en étoile et en flocon de neige, de différentes tailles (ii) des requêtes avec des opérations gourmandes en processeur et (iii) des requêtes avec des opérations gourmandes en Entrées/Sorties. Celles-ci peuvent impliquer la lecture d'une seule table ou plusieurs, une simple fonction d'agrégation (*count(*)*) à des fonctions plus complexes, des jointures à un seul prédicat ou à plusieurs avec des opérations de tri/regroupement (voir annexe B, section 2 et section 4).

La première étape consiste donc en la division des plans d'exécution des requêtes d'apprentissage en un ensemble de segments d'énergie/pipelines suivant les opérateurs bloquant et semi-bloquant. Pour ce faire, nous faisons appel au module de surveillance SQL temps réel du SGBD [165]. Nous lançons ensuite les différentes exécutions, dans une station de travail qui contient la BD déployée (serveur de données). La station est équipée avec la dernière version du SGBD Oracle 11gR2 sous Ubuntu Server 14.04 LTS avec noyau Linux 3.13. Nous avons désactivé les tâches de fond inutiles, et vidé le cache du système et le buffer d'Oracle avant chaque exécution de requête, et ce dans le but de réduire les influences indésirables. À noter également que les techniques de modification dynamique de la performance des composants matériels pour une meilleure efficacité énergétique (comme la tension-fréquence dynamique du processeur (DVFS)) [25] ne sont pas appliquées dans nos expérimentations. Les valeurs de l'énergie consommée par le système sont récupérées au moyen d'un wattmètre "Watts UP? Pro ES" [45] ayant une résolution maximale d'une seconde. Le wattmètre est directement connecté entre la prise d'alimentation et la station de travail sur laquelle les BD sont déployées afin de mesurer la consommation d'énergie globale (figure 4.6). Les valeurs mesurées sont enregistrées et traitées dans une machine de contrôle séparée connectée avec le wattmètre via un câble USB. Les $Cout_{E/S}$ et $Cout_{CPU}$ sont directement prélevés à partir du SGBD. Notons que pour certaines fonctions d'agrégation, notre SGBD ne calcule pas leur coût de CPU malgré leur consommation énergétique importante. Nous proposons de calculer manuellement les cycles de CPU requis pour leur exécution, en utilisant les coûts de notre machine pour faire les instructions de base (ADD, SUB, MUL, etc.). Les estimations obtenues sont ensuite multipliés par le nombre de lignes. À la fin, nous appliquons l'équation de régression (★) à l'aide du logi-

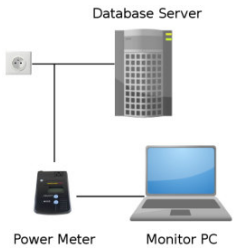


FIGURE 4.6 – Architecture d'expérimentations

ciel R [46] pour trouver les paramètres β_i du modèle. Cette équation sera désormais utilisée pour

45. https://www.wattsupmeters.com/secure/products.php?pn=0&wai=0&more=3
46. http://www.r-project.org/

prédire la consommation énergétique des requêtes interrogeant notre système. Ces prédictions peuvent ensuite être utilisées pour améliorer l'efficacité énergétique des SGBD, par exemple en intégrant le modèle dans des techniques d'optimisation (voir chapitre 5), l'ajustement de la fréquence de la tension dynamique (DVFS), l'amélioration des algorithmes, et ainsi de suite.

3.4.3 Validation du modèle de coût

Le modèle ainsi obtenu doit être validé en testant son efficacité suivant différents scénarios. Étant donné le coût d'énergie estimé (E) par notre modèle, nous le comparons avec la dépense énergétique réelle (M) du système, mesurée avec le wattmètre. Pour quantifier la précision du modèle, nous avons utilisé la métrique du taux d'erreur suivante :

$$Erreur = \frac{|M - E|}{M}$$

$$Erreur_{moy} = \frac{1}{n} \times \sum_{i=1}^{n} \frac{|M_i - E_i|}{M_i}$$

3.5 Modèle de coût orienté espace de stockage

L'estimation de l'espace de stockage d'un ED, consiste à calculer la taille des tables de son schéma. Le calcul se fait directement en s'appuyant sur les statistiques fournies par le SGBD ou par le concepteur. Ce calcul ainsi que la déduction de la taille des schémas en flocon de neige à partir de son schéma en étoile sont détaillés dans la section 4.2.

3.6 Sélection de la meilleure variante

Le concepteur doit maintenant choisir parmi la multitude des schémas logiques possibles. Ce choix est devenu plus fiable car ne se base plus uniquement sur l'intuition des concepteurs, mais s'appuie désormais sur les différents modèles de coûts susmentionnés. Notons que l'évaluation de tous les schémas semble naïve, mais rappelons qu'il s'agit d'un processus à long terme (non fréquent). Il est effectué une seule fois pendant la conception de la BD, ou en vue de la faire évoluer après le subissement d'importants changements.

3.6.1 Sélection mono-objectif

La sélection se fait en terme d'un seul BnF parmi : le temps d'exécution des requêtes, l'énergie consommée ou l'espace de stockage. Le concepteur doit utiliser le MdC propre au BnF choisi pour évaluer les différents schémas logiques, avant de sélectionner celui ayant la valeur minimale du BnF.

$$Sch_{BD} = Min_{BnF}(Sch_1, Sch_2, \ldots, Sch_n) \quad \text{avec} \quad BnF \in \{temps, energie, stockage\}$$

3.6.2 Sélection multi-objectifs : méthode de la somme pondérée

La sélection multi-objectifs représente le scénario où le concepteur veut choisir son schéma logique vis-à-vis de plusieurs BnF, ce qui est souvent le cas. Ces derniers étant contradictoires, leur optimisation simultanée revient à trouver le schéma ayant le meilleur compromis. Afin de fournir le compromis souhaité, nous proposons utiliser la méthode de la *somme pondérée des fonctions objectives*. Dans cette méthode d'agrégation, nous calculons la somme pondérée des fonctions de coût normalisées pour agréger les objectifs (BnF) en une seule fonction objective équivalente à optimiser. Cette méthode est définie comme suit :

$$\begin{aligned} minimize \quad y = f(x) &= \sum_{i=1}^{k} \omega_i \times f_i(\overrightarrow{x}) / \sum_{i=1}^{k} \omega_i = 1 \\ f_{compromis} &= {}^1\!/_3 \times f_{temps} + {}^1\!/_3 \times f_{energie} + {}^1\!/_3 \times f_{stockage} \\ f_{temps} &= {}^3\!/_4 \times f_{temps} + {}^1\!/_4 \times f_{energie} + {}^0\!/_4 \times f_{stockage} \end{aligned}$$

Où ω_i sont les coefficients de pondération représentant l'importance relative des fonctions de coûts f_i. f_{temps}, $f_{energie}$, $f_{stockage}$ représentent respectivement les fonction de coût de temps, d'énergie et du stockage. $f_{compromis}$ est un exemple d'équilibrage entre les trois BnF cibles, quand à f_{temps}, elle est un exemple de compromis orienté temps d'exécution. Pour normaliser le vecteur des valeurs de fonctions objectives, nous utilisons la formule suivante:

$$x' = \begin{cases} \frac{(max'-min')}{(max-min)} * (x - min) + min' & \text{si } x \in [min, max], x' \in [min', max'] \\ \\ \frac{x-min}{max-min} & \text{si } x \in [min, max], x' \in [0, 1] \end{cases}$$

Où x est la valeur d'origine et x' est la valeur normalisée. Il est important de noter que cette technique n'est efficace pour trouver le minimum simultané (compromis) des fonctions membres, que quand la courbe de la frontière de *Pareto*, est d'allure convexe.

Définition 5 (*Optimum de Pareto*)

Dans l'optimisation multiobjectif, il n'existe pratiquement pas de solution qui, à la fois, minimise toutes les fonctions objectives. Par conséquent, des solutions qui ne peuvent pas être améliorées dans l'un des objectifs sans dégrader au moins l'un des autres objectifs sont appelées des optimums de Pareto [197].

Plus formellement, une solution x_1 est dite dominée par une autre solution x_2 si :

$$f_i(x_1) \leq f_i(x_2)\ \forall i \in \{1, 2, \ldots, k\}\ \wedge\ \exists f_j(x) : f_j(x_1) < f_j(x_2)$$

Une solution x_1 est appelée optimum de Pareto s'il n'existe pas une autre solution qui la domine. L'ensemble des résultats optimaux de Pareto est appelé la frontière de Pareto.

4 Validation et expérimentations

L'impact de la variabilité du schéma logique sur le traitement des requêtes a été évalué à travers un ensemble d'expériences que nous avons menées. Dans cette dernière section, nous commençons d'abord par présenter les expérimentations orientées temps d'exécution, puis celles orientées énergie.

4.1 Génération des schémas logiques

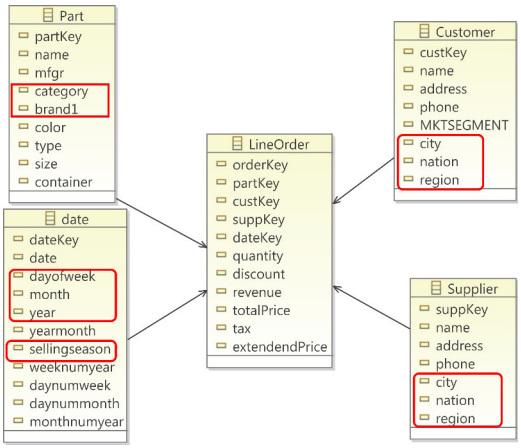


FIGURE 4.7 – Schéma logique multidimensionnel du banc-d'essai SSB

Comme illustré dans la figure 4.7, les hiérarchies que nous avons choisies sont les suivantes :

— pour la table `Customer, H={city, nation, region}` ;

- pour la table `Supplier, H={city, nation, region}` ;
- pour la table `Part, H={brand, category}` ;
- pour la table `Customer, H={dayofweek, month, sellingseason, year}.`

Par application numérique, nous obtenons le nombre des schémas logiques possibles, que nous créons, peuplons un à un et réécrivons la charge de requêtes originale pour chacun. Nous donnons une brève description des schémas générés dans l'annexe A.

$$H(Customer) \times H(Part) \times H(Supplier) \times H(Date) = 2^{3-1} \times 2^{2-1} \times 2^{3-1} \times 2^{4-1}$$
$$= 256 \text{ schémas logiques possibles.}$$

4.2 Évaluation de l'impact en fonction du temps d'exécution

Nous menons une étude de cas basée sur le banc d'essai *Star Schema Benchmark* (SSB) [47] destiné aux systèmes d'aide à la décision qui examinent de grandes quantités de données, et dont le schéma logique est représenté dans la figure 4.7, et la charge est donnée dans l'annexe B section 1.

La station de travail est équipée avec la dernière version du SGBD Oracle 11gR2 ayant pour taille de bloc (page) 8192 *bytes*, et différents facteurs d'échelle de données hébergées sur une machine avec 4GB de RAM. Les principales statistiques sont extraites à partir de la définition du schéma : nombre et taille des tuples de chaque table, la taille du domaine des attributs, le type de corrélation qui lie les attributs hiérarchiques, et éventuellement la valeur delta.

4.2.1 Évaluation empirique

Nous commençons d'abord par évaluer empiriquement l'impact de la variabilité logique sur l'optimisation logique, en déléguant ce type d'optimisations (comme l'implémentation et l'ordre des jointures) à l'optimiseur des requêtes du SGBD *Oracle*, et en désactivant les structures physiques avancées d'optimisation comme les VM. Nous avons soumis la charge d'origine de requêtes du *SSB* sur les 256 schémas avec 1GB puis 10GB comme facteurs d'échelle. Les résultats sont décrits par les figures 4.8 et 4.9 suivantes, où le schéma en étoile original est le premier (numéro 1) suivi par les différentes normalisations numérotées de 2 à 256.

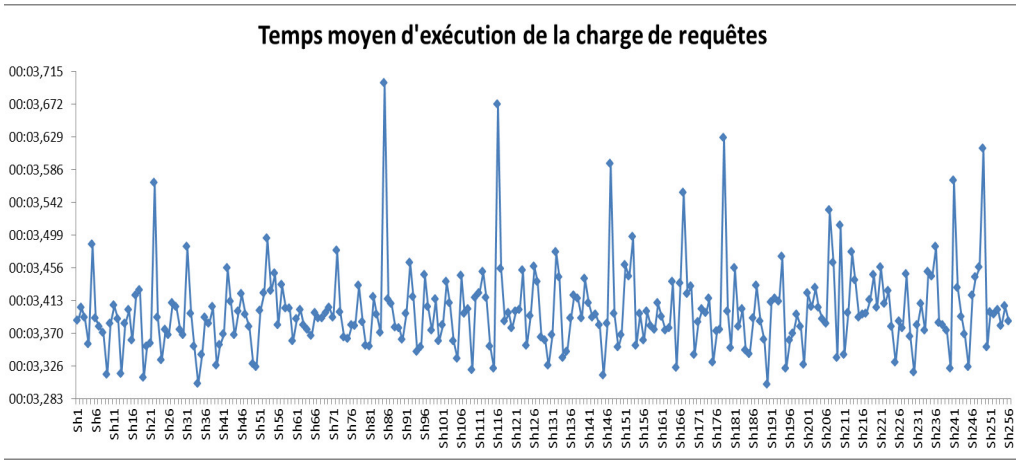


FIGURE 4.8 – Impact de la variabilité logique sur la performance des données de 1G

47. https://github.com/electrum/ssb-dbgen

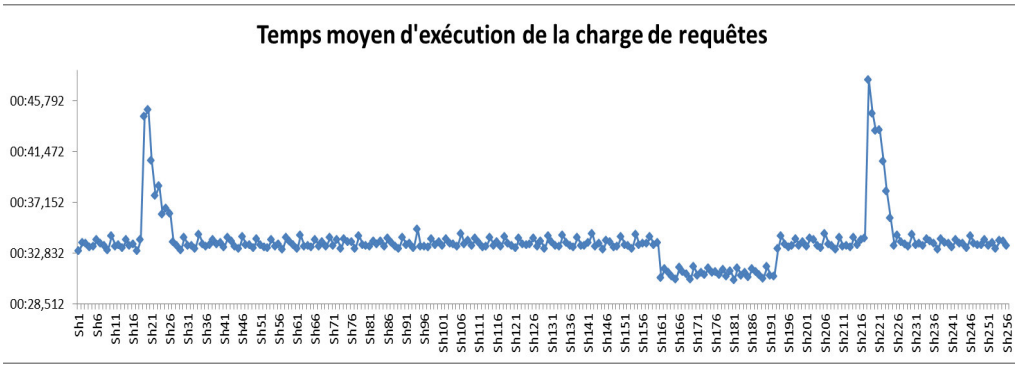


FIGURE 4.9 – Impact de la variabilité logique sur la performance des données de 10G

4.2.2 Analyse des résultats

Par le biais de ces premières expériences, nous voulons montrer l'importance de la conception logique :

- les résultats montrent effectivement que la variation du schéma logique a bien un impact sur les performances ;
- ils démentent le fait que le schéma en étoile (le plus dé-normalisé possible) est le plus performant. En effet, dans les deux expérimentations, le schéma en étoile (n°1) n'est pas l'optimal. Le schéma n°190 et n°17 pour 1G et 10G respectivement sont les plus performants et le schéma en étoile est classé bien derrière à la 112ème et 4ème place avec un surplus de quelques centaines de ms. Nous analysons l'effet des caractéristiques des schémas (table et taille) sur la performance pendant l'évaluation théorique ;
- l'impact d'un même schéma logique sur la performance d'une même charge de requêtes peut être optimal avec une taille de données et non-optimal avec une autre taille, à l'image du schéma n°181. Idem, nous analysons cet effet dans la deuxième vague d'expériences ;
- de manière générale, nous démentons le fait que la performance d'un ED est strictement liée à la conception physique, et qui fait que certains projets d'ED s'en contentent actuellement (en passant du schéma conceptuel au physique directement). Or, même si la plupart des problèmes de performance peuvent être résolus avec un ajustement fin du modèle physique, quelques problèmes de performance sont causés par la non-optimisation du schéma logique [78]. il faut, en effet, éviter la confusion entre la phase conceptuelle et logique, qui réduit cette dernière à la transformation d'un schéma devant répondre aux besoins utilisateurs, vers le modèle de données du SGBD.

4.2.3 Évaluation théorique

Dans la pratique, le concepteur ne peut pas déployer physiquement toutes les configurations afin de réellement évaluer les effets des choix de conception et des changements (évolution), d'où le recours aux MdC. Dans cette sous section, nous voulons :

— analyser les caractéristiques des schémas (taille des tables, attributs, etc.) qui influencent la performance des requêtes ;

— évaluer les schémas en utilisant notre MdC et comparer les résultats obtenus avec les résultats empiriques afin de valider notre MdC.

De la même manière, l'évaluation théorique consiste à appliquer le MdC sur chaque requête de chaque schéma, tout en tenant compte des différents paramètres nécessaires (statistiques BD). Chaque schéma possède 13 caractéristiques : une taille, et 12 valeurs de coûts (une pour chaque requête soumise). Les besoins des utilisateurs peuvent intervenir à cette étape pour indiquer les requêtes les plus importantes. Ensuite, selon cette information et l'ensemble de résultats, les schémas ayant le ratio de *temps d'exécution/taille* le plus petit (meilleur compromis) seront choisis. Notons que le facteur d'échelle utilisé pour cette évaluation théorique est de 10.

Afin de bien analyser les caractéristiques influençant la performance, nous examinons la charge requête par requête. La charge SSB est composée de 4 catégories de requêtes, chacune a en moyenne 3 requêtes. La figure 4.10 montre leurs coûts d'exécution sur le schéma en étoile d'entrée.

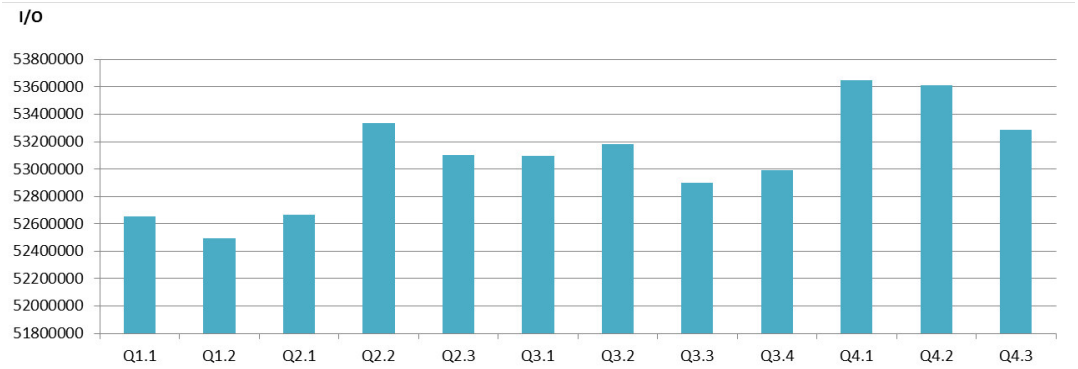


FIGURE 4.10 – Les coûts de requêtes d'origine de SSB dans le schéma étoile

1. Catégorie 1 : les requêtes ont des restrictions portant sur une seule dimension impliquant l'attribut hiérarchique `année` de la hiérarchie `date{dayofweek,month,sellingsea son,year}`. Les petits coûts appartiennent aux schémas dont la dimension `Dates` est effectivement normalisée. Cela dit, il n'existe pas une grande différence entre les coûts. Ce résultat peut être expliqué par la petite taille de la table de dimension `Dates` d'origine (dé-normalisée). Cette dernière est fixe quelque soit le facteur d'échelle, et dont la lecture peut se faire en mémoire centrale.
2. Catégorie 2 : ce type de requête a des restrictions sur les deux dimensions `Supplier` et `Part` impliquant leurs hiérarchies respectives : `localisation {city, nation, region}` et `catégorie {brand, category}`. Contrairement aux attentes, nous constatons que le coût minimum ne correspond pas au schéma en étoile. Il correspond plutôt aux schémas dont la table *Part* est normalisée en utilisant la hiérarchie `catégories`. Cela peut

être expliqué par un gain important en matière d'espace de stockage suite à cette normalisation, et nous pouvons également relier cela au fait que la granularité de cette hiérarchie est faible (2 contre 4 pour la hiérarchie de `Dates`).

3. Catégorie 3 : les restrictions portent sur trois dimensions: `Dates`, `Customer` et `Supplier` impliquant leurs hiérarchies : `localisation` de `Customer`, `localisation` de `Supplier` et `catégorie` de `Part`. Les petits coûts appartiennent à des schémas où les dimensions `Part`, `Dates` et `Supplier` sont normalisées. La même explication s'applique concernant l'espace de stockage gagné par la normalisation de cette dernière table. En effet, le plus grand rapport entre l'ancienne et la nouvelle taille des tables revient à la table `Part`, puis `Supplier`, puis `Dates` et enfin `Customer`.
4. Catégorie 4 : toutes les tables de dimensions sont impliquées. En général, la même logique s'applique, c'est à dire que les coûts minimaux appartiennent à des schémas où `Part` est normalisée, puis augmentent suivant le gain en matière de taille de la dimension (la normalisation des autres tables `Supplier`, `Dates` et `Customer`), la taille des sous-dimensions et le niveau de décomposition des sous-dimensions. En effet, la taille des données détermine le gain en termes de stockage de tables de dimensions, qui diminuera le coût de la jointure avec la table des faits, si les jointures engendrées par les sous-dimensions ne sont pas coûteuses. Ce dernier point dépend de la taille des sous-dimensions (domaine et taille des attributs hiérarchiques) ainsi que du niveau de décomposition.

Nous avons constaté que les meilleurs coûts de l'ensemble des requêtes sont obtenus avec le 17ème schéma, dont seule la dimension `Part` est normalisée. La figure 4.11 montre la différence entre les coûts correspondants au schéma d'origine (en étoile) et le 17ème schéma.

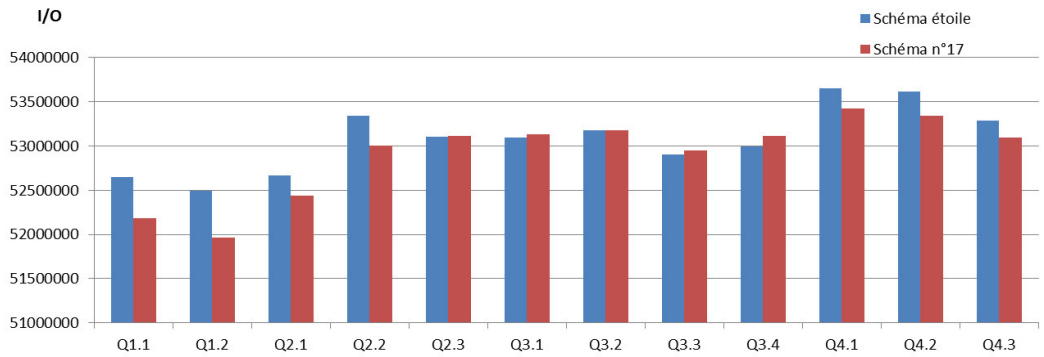


FIGURE 4.11 – Comparaison entre les coûts des requêtes dans le meilleur schéma (n°17) et le schéma en étoile

Afin de valider notre MdC, nous établissons une comparaison entre les coûts théoriques et empiriques, comme présentée dans la figure 4.12. Ces résultats doivent en principe être similaires à ceux représentés dans la figure 4.11. Mais bien que n'étant pas identiques dans l'ensemble, les résultats demeurent cohérents au sens où le meilleur schéma en terme d'exécution

de requêtes est le même. Ainsi, les requêtes $Q1, Q2, Q4$ sont exécutées plus rapidement lorsqu'elles sont soumises au 17ème schéma. Quant aux requêtes $Q3.2$ et $Q3.3$, elles sont exécutées plus rapidement lorsqu'elles sont soumises au schéma initial. Ces constatations sont vraies dans une certaine mesure quel que soit le type d'évaluation adoptée (évaluation théorique se basant sur le modèle de coût ou évaluation empirique utilisant le SGBD réel). Cependant, ils ne sont pas strictement identiques à cause de la surestimation du coût de certaines requêtes (en l'occurrence ici $Q2, Q3, Q4$). Les différences peuvent être dues au nombre de jointures présentes dans les requêtes concernées. En effet, les requêtes $Q2, Q3$ et $Q4$ contiennent plusieurs jointures qui doivent être ordonnées. Cette surestimation du coût de certaines requêtes résulte de notre choix d'une technique d'ordonnancement se basant sur la valeur du *Share* (voir section 3.1).

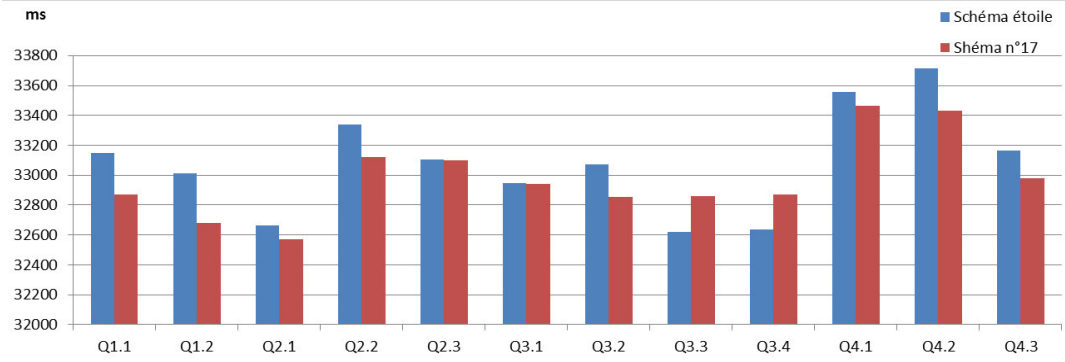


FIGURE 4.12 – Comparaison entre l'exécution réelle de la charge SSB sur le schéma étoile et le meilleur schéma (17)

4.3 Évaluation de l'impact en fonction de l'énergie

De la même manière, nous commençons d'abord par une évaluation empirique pour démontrer de manière générale l'existence de l'impact du modèle logique sur la consommation énergétique. Et puis nous validons notre MdC à travers une comparaison entre les évaluations empirique et théorique. Pour l'énergie, nous avons utilisé un autre environnement et une autre charge de requêtes que ceux utilisés pour l'évaluation du temps d'exécution, et ce pour des raisons expliquées ci-dessous.

Configuration matérielle et logicielle. Afin de mesurer l'énergie consommée en temps réel, nous utilisons une architecture client/serveur basée sur les sockets implémentés en langage C, impliquant deux machines :

— La station de travail (partie client de l'application) où l'ED est déployé et les requêtes sont exécutées. L'alimentation de cette machine passe par le wattmètre "*Watts UP? Pro ES*" (voir section 3.4.2) qui est relié à la prise murale (voir figure 4.6). Nous avons utilisé le système d'exploitation *Ubuntu Server 14.04 LTS* avec noyau *Linux 3.13* avec la dernière

version du SGBD Oracle 11gR2. La station de travail est *Dell PowerEdge R210 II* ayant un processeur *Intel Xeon E3-1230 V2* de 3.30GHz, 10 Go de mémoire DDR3 et une configuration disque dur de 2x500 Go. Afin de minimiser les influences indésirables, nous y avons désactivé les tâches de fond inutiles, et vidé le cache du système et le buffer d'Oracle avant chaque exécution de requête à l'aide des requêtes suivantes.

```
--Vider buffer Oracle
ALTER SYSTEM FLUSH SHARED_POOL ;
ALTER SYSTEM FLUSH BUFFER_CACHE ;
```

```
#!/bin/bash
#Vider pagecache, dentries et inodes (noyau linux)
sync && echo 3 | tee /proc/sys/vm/ drop_caches
```

C'est au niveau de cette machine qu'on récupère le plan d'exécution avec toutes les informations notamment les coûts E/S et CPU, temps d'exécution des opérateurs, et découpage en pipelines. avec la commande Oracle :

```
SELECT DBMS_SQLTUNE.report_sql_monitor(
sql_id => '526mvccm5nfy4',
type => 'ACTIVE',
report_level => 'ALL') AS report
FROM dual;
```

— Une machine de contrôle (partie serveur de l'application), ayant sa propre alimentation électrique, est reliée au wattmètre via un câble USB. Elle permet de lancer/arrêter les expérimentations, et d'enregistrer les valeurs de l'énergie consommée par la station de travail et capturées par le wattmètre.

Jeu de données. Pour cette première vague d'expérimentations, nous avons utilisé les données *SSB* avec un facteur d'échelle égal à 10. Cependant, nous avons créé une nouvelle charge avec 30 requêtes basées sur la charge initiale de *SSB*, de façon à ce que différentes catégories de requêtes puissent s'exécuter, celles qui épuisent le processeur, celles qui sont gourmandes en opérations d'entrées/sorties et des requêtes hybrides (voir Annexe B section 2).

4.3.1 Évaluation empirique

Nous commençons d'abord par évaluer empiriquement l'impact de la variabilité logique sur l'optimisation logique, en déléguant ce type d'optimisation (comme l'implémentation et l'ordre des jointures) à l'optimiseur des requêtes du SGBD *Oracle*, et en désactivant les structures physiques avancées d'optimisation comme les VM. L'ensemble des 256 schémas logiques sont déjà déployés, cependant les nouvelles requêtes doivent être réécrites selon chaque schéma

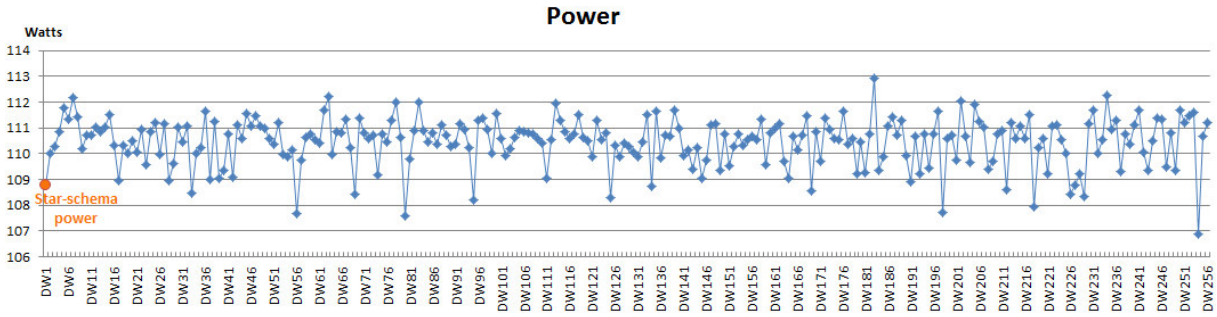


FIGURE 4.13 – Impact de la conception logique sur la consommation énergétique de l'ED

(30*256 = 7680 requêtes en tout) et exécutées. Le temps d'exécution (à partir d'*Oracle*) et la consommation de la puissance (wattmètre) sont prélevés et enregistrés.

Nous nous concentrons en premier lieu sur la fonction objective de la puissance consommée, et dont les résultats sont illustrés dans la figure 4.13 qui confirme effectivement que la variation du schéma logique a un impact sur la consommation de l'énergie, et encore mieux, elle démontre que le schéma en étoile est loin d'être le plus écologique.

Nous avons constaté que la co-normalisation des plus petites tables de dimensions (`Supplier (2000*SF); Dates (2556)`) en présence d'opérations fortement consommatrices du CPU représente un désavantage certain en terme de consommation énergétique. À l'inverse, le nombre de jointures (coûts des E/S) ainsi que le nombre d'opérations fortement consommatrices de CPU (par exemple les opérations d'agrégation et de tri) n'ont pas d'influence directe sur la consommation énergétique. Une explication possible de ce phénomène est que la majeure partie du temps d'exécution des requêtes correspond à du traitement au niveau du CPU, la lecture des données étant rapide due à la faible taille des fichiers. À l'inverse, lorsque la majeure partie du temps d'exécution des requêtes est écoulée dans l'attente que les données soient disponibles (à cause du transfert de données entre la mémoire et le disque), une plus faible consommation énergétique est relevée.

Dans un deuxième temps, nous considérons deux fonctions objectives représentant la performance des requêtes et la consommation énergétique. Nous mettons ensuite en évidence la relation entre ces deux fonctions qui prend la forme d'une courbe convexe ainsi qu'illustré en figure 4.14. Cela permet de montrer l'existence de schémas logiques qui optimisent les deux BnF tout en approuvant notre choix de méthode de sélection des schémas (somme pondérée, voir section 3.6.2). En effet, nous avons trouvé 18 schémas de *Pareto* (voir définition 5).

Afin d'assister les concepteurs d'ED dans leur choix d'un schéma correspondant au mieux à leurs besoins, nous proposons, dans un premier temps, d'utiliser une méthode considérant la somme pondérée des deux fonctions objectives, qui permet de formuler le compromis souhaité entre les BnF cibles (voir section 3.6.2).

$$minimize \quad y = f(x) = 1/2 \times f_{temps} + 1/2 \times f_{energie}$$

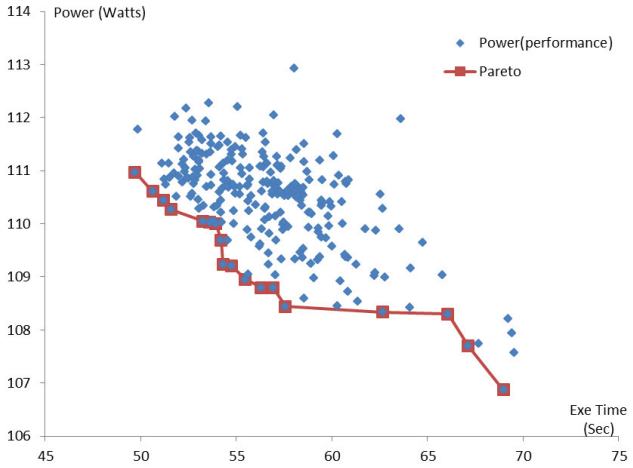


FIGURE 4.14 – Impact de la conception logique sur la performance et la puissance de l'ED

En effet, un schéma éco-performant aurait $\omega_{pow} = \omega_{perf} = 1/2$ alors qu'un schéma orienté performance aurait $\omega_{perf} > \omega_{pow}$, contrairement à un schéma orienté énergie ($\omega_{pow} > \omega_{perf}$). Cette technique est particulièrement bien adaptée lorsque le front de *Pareto* est convexe, ce qui est le cas pour notre courbe (voir figure 4.14). Nous avons appliqué cette fonction sur les résultats normalisés de tous les schémas en fonction du temps et de puissance. Le tableau 4.2 résume les résultats de certains schémas logiques de référence. Il s'agit (dans l'ordre) du schéma : le plus écologique, le plus performant, le moins écologique, le moins performant, en étoile et enfin le schéma du compromis.

Caractéristique du schéma	Numéro schéma	Temps exécution	Puissance consommée
Min_puissance	Sh254	68,9667	106,873
Min_temps	Sh161	49,7	110,974
Max_puissance	Sh183	58,0333	112,933
Max_temps	Sh80	69,5333	107,574
Schéma étoile	Sh1	56,931	108,792
Schéma compromis	Sh228	54,3	109,034

TABLE 4.2 – Les caractéristiques de certains schémas logiques de référence

4.3.2 Évaluation théorique

Nos expérimentations empiriques précédentes ont duré presque 10 jours (7680 requêtes), ce qui confirme la nécessité de l'utilisation d'un MdC (simulateur). Pour ce faire et comme

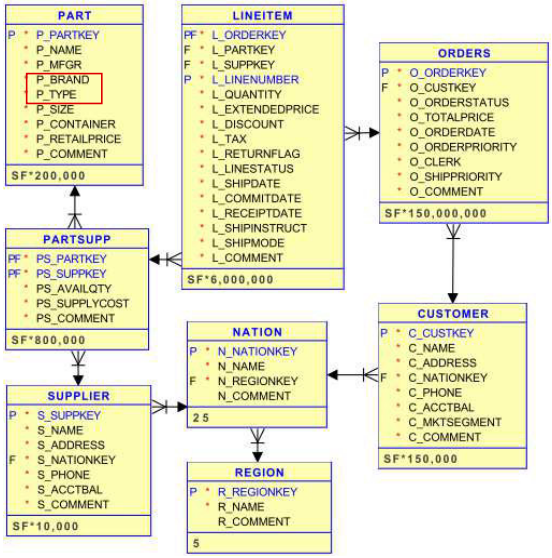


FIGURE 4.15 – Schéma du benchmark TPC-H

expliqué dans la section 3.4.2, une phase d'apprentissage est nécessaire pour identifier les coefficients du modèle de régression β. Ce modèle permet d'estimer/prédire l'énergie consommée par une BD sans avoir à la déployer. Notons que la phase d'apprentissage est cruciale car c'est elle qui permet de se rapprocher le plus possible de la consommation énergétique réelle de la BD. Nous nous basons sur les jeux de données et les requêtes du benchmark décisionnel *TPC-H*[48], avec une taille (facteur d'échelle) de 10 Go et 100 Go. Le schéma présente un modèle *produits-commandes-fournisseurs* réparti sur 2 tables de faits et 6 tables de dimensions, avec une charge de 22 requêtes décisionnelles caractérisées par un large volume de données et un degré élevé de complexité. Nous avons généré un schéma normalisé (plus le schéma d'origine), en décomposant la hiérarchie `brand, type` de la table de dimension `Part`, le reste des tables étant déjà quasi-normalisées (voir figure 4.15). Nous avons donc un total de 4 schémas : 2 en étoile (10 et 100GB) et 2 en flocon de neige.

Nous avons créé une charge de requêtes basée sur les données de TPC-H pour la phase d'apprentissage de façon à ce que différentes catégories de requêtes puissent s'exécuter, celles qui épuisent le processeur, celles qui sont gourmandes en opérations d'entrées/sorties et des requêtes hybrides. Elles sont présentées dans l'annexe B section 4. Nous récupérons à chaque exécution de requête des différents schémas, des statistiques liées à la consommation énergétique du système et aux pipelines. Celles-ci sont utilisées par le logiciel *R* pour estimer les paramètres du modèle. Les coefficients finaux du MdC d'énergie sont présentés dans l'annexe B.

Par la suite, nous avons exécuté la charge originale du TPC-H (22 requêtes), nous avons prélevé la puissance réelle de l'exécution de chaque requête avant de calculer leur puissance

48. `http://www.tpc.org/tpch/`

prédite par notre MdC. Nous notons que la plupart des requêtes contiennent plus de 4 pipelines. La moyenne des résultats de comparaison entre les valeurs réelles et prédites de puissance consommée lors de l'exécution de la charge originale du TPC-H sont présentés dans la Table 4.3. Comme nous pouvons le voir dans le tableau, l'erreur moyenne est généralement très minimale (1,6% pour l'ensemble de données de 10Go et 2,1% pour celui de 100Go), et l'erreur maximale est au-dessous de 5%.

Requête	Erreur (%)		Requête	Erreur (%)	
	Sh			Sh	
	10G	100G		10G	100G
Q1	1,2	0,9	Q12	1,9	0,05
Q2	10,1	8,9	Q13	12,8	6,1
Q3	1,0	0,1	Q14	0,4	0,6
Q4	0,5	0,3	Q15	2,3	1,0
Q5	1,2	0,6	Q16	0,4	1,7
Q6	1,3	0,1	Q17	0,2	1,4
Q7	1,1	0,7	Q18	1,9	3,7
Q8	0,5	0,1	Q19	0,6	1,0
Q9	1,9	0,8	Q20	1,8	2,1
Q10	0,6	1,2	Q21	0,9	0,4
Q11	4,8	0,3	Q22	0,007	2,1

TABLE 4.3 – Erreurs d'estimation d'énergie pour les requêtes du benchmark TPC-H avec différents tailles et schémas de BD.

4.4 Évaluation de l'impact en fonction de l'espace de stockage

Contrairement aux discussions précédentes, les résultats concernant l'impact de la variation logique sur l'espace de stockage sont bien connus, et plus faciles à évaluer. Tel que prévu, le processus de normalisation réduit l'espace de stockage, en particulier dans le cas de grandes tables de dimension et/ou de hiérarchies de taille importante. Par conséquent, les schémas en flocon de neige sont adaptés dans le cas d'applications contraintes en espace, ainsi qu'illustré dans la figure 4.16. Ce gain en stockage pourrait aussi être propagé aux contraintes de stockage de structures d'optimisation.

En effet, 19 des 256 schémas ont une taille minimale (4857 pages de moins que pour le schéma en étoile). Ils correspondent aux schémas les plus normalisés.

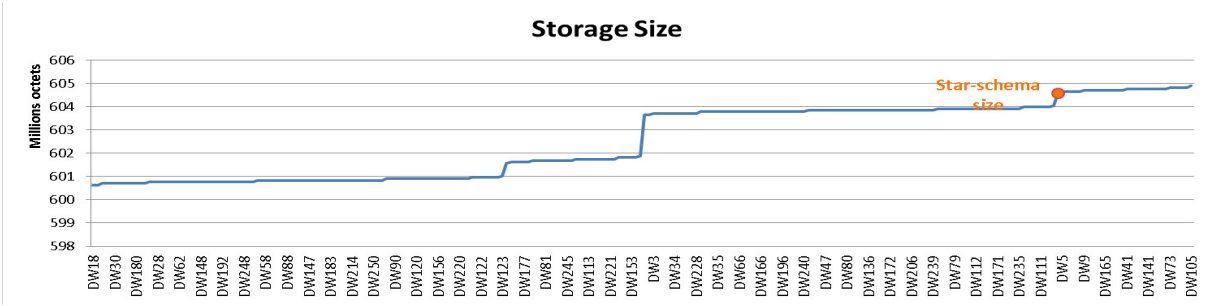


FIGURE 4.16 – Impact de la variabilité logique sur la taille d'implémentation de l'ED

5 Conclusion

L'implémentation de la vision de *conception de BD comme une LdP* est une tâche longue et complexe, c'est pourquoi nous nous sommes focalisés sur le cas de la conception logique peu étudiée. Dans cette optique, ce chapitre permet en premier lieu de décrire comment gérer la variabilité au niveau de la phase logique, en expliquant d'abord, comment exploiter les corrélations afin de générer les différents schémas logiques candidats à être le meilleur schéma. Puis, il se focalise sur l'impact *intraphase*, c'est-à-dire l'impact de ces modèles logiques sur l'optimisation logique. Cet impact doit être évalué afin d'aider le concepteur à choisir parmi cette panoplie de schémas logiques (pondérer les chemins de l'arbre correspondant). Cette évaluation est traditionnellement faite en fonction des BnF classiques que sont le temps d'exécution et l'espace de stockage. Nous avons également considéré la consommation de l'énergie dans cette évaluation. Pour ce faire, nous présentons nos MdC permettant de sélectionner le choix le plus pertinent et ce en fonction d'un seul ou de plusieurs BnF (sélection mono ou multi-objectives). Des expérimentations théoriques et empiriques en vue de valider notre démarche sont représentées à la fin du chapitre.

Les résultats obtenus nous montre que l'implémentation de la vision de *conception de BD comme une LdP* est faisable. En effet, nos algorithmes de génération/peuplement des schémas logiques ainsi que nos MdC d'évaluation de différents BnF, sont pluggables pour implémenter les features de nos FM. Cela se fait grâce aux outils orientés gestion de variabilité (*par exemple FeatureIDE*). En outre, les résultats mettent en avant l'importance de la variabilité logique, jusque là sous-estimée, à travers la détection de son impact sur les BnF orientés BD. Ils incitent également les concepteurs à rompre avec quelques habitudes liées à cette conception logique, comme par exemple le fait de figer un schéma logique (variabilité ignorée) et de favoriser le schéma en étoile. Ces habitudes ne sont plus valables dans un contexte aussi diverse, évolutif et (donc) variable et où la performance n'est plus limitée au temps d'exécution uniquement. Maintenant, il va falloir montrer que la gestion holistique de la variabilité de conception de BD est également faisable. Cela requiert l'étude de l'impact interphase présenté dans le chapitre 5 suivant.

Chapitre 5

Impact de la variabilité sur l'optimisation physique

Sommaire

1 Introduction 159
2 Fondements théoriques 160
3 Adaptation de l'approche de génération de graphe de requêtes 163
3.1 Construction des hypergraphes multidimensionnels 163
3.2 Ordonnancement des sous-hypergraphes 164
3.3 Transformation d'un hypergraphe en graphe 164
4 Sélection de la meilleure variante 165
4.1 Sélection mono-objectif : temps d'exécution 166
4.2 Sélection multi-objectifs : algorithme évolutionnaire 166
5 Validation et expérimentations 168
5.1 Évaluation du scénario mono-objectif 168
5.2 Évaluation du scénario multi-objectifs 169
6 Conclusion 171

1 Introduction

Dans le chapitre précédent, nous avons montré notre approche pour gérer la variabilité logique : considération des choix des modèles logiques disponibles et sélection du modèle le plus adéquat en matière de BnF. La sélection se fait en évaluant l'impact intraphase. Afin de montrer la gestion *holistique* de la variabilité sur le même cas, il faut de plus, considérer l'interdépendance entre les phases de conception des BD. Nous avons choisi la phase physique pour refléter l'impact *interphase* de la variabilité logique, car c'est la dernière phase et elle joue ainsi le rôle d'*entonnoir*. La phase physique contient plusieurs features. Nous avons choisi de considérer l'*optimisation physique* car c'est une tâche directement liée aux BnF. Nous nous sommes plus précisément focalisés sur le problème de sélection des vues matérialisées (VM). Parmi les variantes de schéma logique possibles, le schéma ayant le meilleur impact sur l'optimisation physique en fonction de BnF sera sélectionné. Considérons l'arbre illustré par la figure 5.1 où les schémas logiques possibles sont représentés par les features `{star, SnF1, SnF2, ..., SnFn}` et le passage du `SL` à `MV` représente la considération des VM comme structures d'optimisation physique. Le problème de la sélection du meilleur schéma logique avec les VM adaptées revient à choisir le chemin de cet arbre ayant le poids minimal.

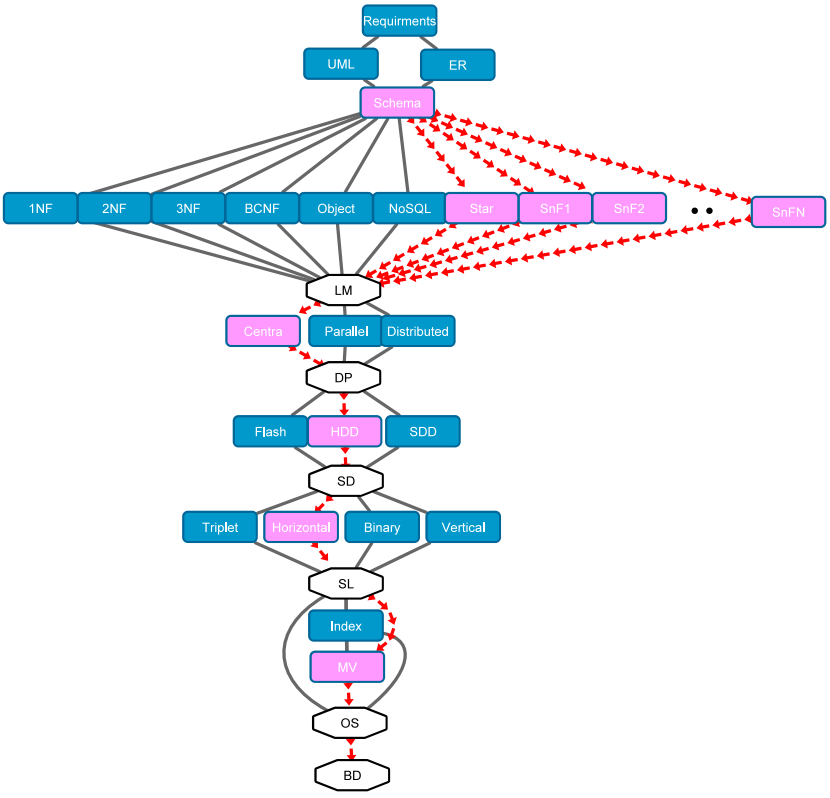


FIGURE 5.1 – Extrait de l'arbre modélisant l'impact interphase de la variabilité logique sur l'optimisation physique

La pondération des chemins de l'arbre considéré se fait au moyen de MdC qui permettent l'évaluation de cet impact, en fonction de BnF tels que le temps d'exécution et/ou l'énergie consommée. Dans le cadre de notre étude, nous ne déléguons pas la sélection des VM aux advisors, mais nous proposons un algorithme qui effectue cette tâche connue comme un problème NP-complet et ayant fait l'objet de plusieurs travaux de recherche [120]. Le processus de sélection des vues se base sur trois éléments que sont [35] :

1. une structure de données pour capturer l'interaction entre les requêtes (optimisation multi-requêtes : OMR), comme le *graphe Et/Ou* ou le *Multi-View Processing Plan* [120]. Celle-ci doit ordonner les opérations algébriques des requêtes sous forme d'un graphe acyclique en commençant par les tables de base sous forme de feuilles jusqu'aux résultats de requêtes sous forme de racines, et en passant par des noeuds intermédiaires : opérations unaires (comme la sélection/projection) et binaires (comme la jointure/union). La définition de l'ordre optimal entre les noeuds intermédiaires détermine l'efficacité de la structure ;
2. les algorithmes qui exploitent une telle structure pour choisir la meilleure configuration de VM (par exemple les algorithmes déterministes ou probabilistes) ;
3. les modèles de coûts qui estiment la qualité des configurations de VM en matière de BnF.

Nous commençons par présenter dans la section 2 les fondements de l'approche que nous utilisons pour la construction de la structure de données orientée OMR qui est basée sur les hypergraphes (point 1). Nous enchaînons avec l'étude de l'impact de la variabilité logique sur ce processus de construction dans la section 3 (point 1). Nous présentons ensuite dans la section 4 notre algorithme pour la sélection de la meilleure configuration de vues matérialisées en matière de BnF (notamment le temps d'exécution et la consommation énergétique) (point 2). Quant aux MdC utilisés pour l'évaluation des configurations de vues matérialisées (point 3), nous utilisons les mêmes que ceux utilisés pour l'évaluation de l'impact logique (voir section 3.3). Nous terminons le chapitre avec des expérimentations présentées dans la section 5 avant de le conclure dans la section 6.

2 Fondements théoriques

Les optimiseurs de requêtes des BD ont été initialement conçus pour optimiser des requêtes en mode isolé. L'augmentation du nombre de requêtes et des utilisateurs exploitant la BD (mode concurrent) et l'identification des interactions entre ces requêtes, ont fait émerger le besoin d'une optimisation multi-requêtes (voir section 3.4.3). Les applications des ED ont favorisé cette interaction de par leur structure multidimensionnelle, centrée sur la table de faits, qui fait que les jointures passent systématiquement par cette table. La difficulté de cette optimisation, considérée comme un problème NP-difficile [120], est l'identification des expressions communes entre les requêtes et l'évaluation de la qualité de ce plan global. Pour le résoudre, des solutions basées sur la fusion des arbres algébriques des requêtes ont été proposées pour la

présentation du plan global de la charge ainsi que les intéractions entre les requêtes. Celles-ci souffrent du problème de passage à l'échelle. Boukorca et al. [35] ont proposé l'exploitation de la théorie de graphes comme solution à ce problème, ce qui permet de considérer un très grand nombre de requêtes. Leur méthode passe par les étapes suivantes.

Construction de l'hypergraphe de jointure

Après avoir analysé la charge des requêtes pour extraire les différentes opérations algébriques (noeuds de sélection, jointure, projection et agrégation), l'approche proposée s'intéresse aux opérations de jointure qui sont les plus coûteuses pour les requêtes OLAP [196]. Chaque requête est représentée par une hyperarête H_i qui contient les différents noeuds de jointures représentés par des sommets, comme illustré dans la figure 5.2. Par exemple, la requête $Q16$ impliquant trois jointures représentées par les noeuds 20, 22 et 28, est une hyperarête dont les sommets ce sont ces derniers noeuds. L'hypergraphe He représente la charge des requêtes et englobe les différentes hyperarêtes $Q1, Q2, Q5, Q7, Q9, Q10, Q14, Q16, Q23, Q24, Q27, Q28$. L'objectif est de maximiser la réutilisation des résultats de ces jointures par les différentes requêtes.

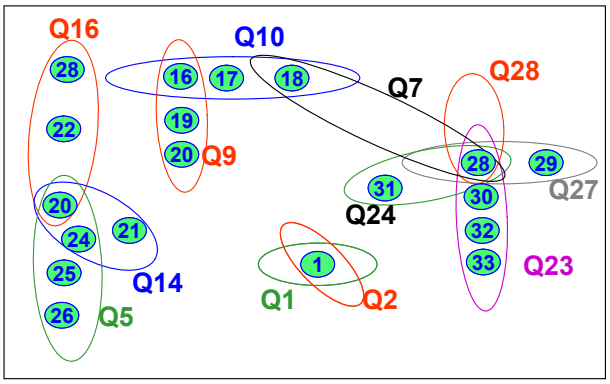


FIGURE 5.2 – Exemple d'hypergraphe de jointures He_1 [9]

Partitionnement de l'hypergraphe

La deuxième étape consiste à partitionner l'hypergraphe construit précédemment en utilisant un algorithme adapté, afin de générer des composantes connexes He_{sub} (figure 5.3). Par définition, chaque composante connexe est un ensemble de noeuds de jointures, pouvant avoir des liens entre eux et (réutilisation possible) et pouvant être traités indépendamment des autres composantes. Le principe étant de diviser un hypergraphe en k nouveaux hypergraphes, de telle sorte que le nombre d'hyperarêtes coupées, c'est à dire d'hyperarête ayant des sommets dans deux sections, soit minimal.

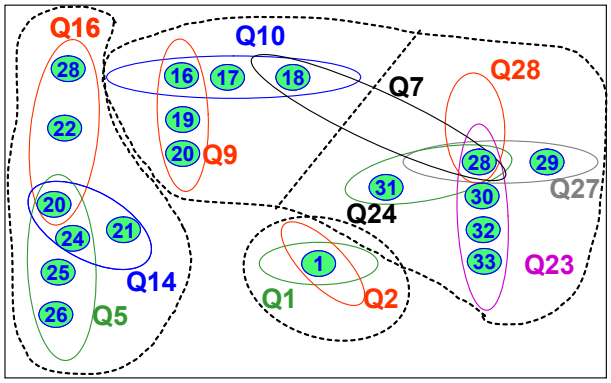


FIGURE 5.3 – Résultat du partitionnement de l'hypergraphe He_1 [9]

Transformation d'un hypergraphe en graphe

La troisième étape consiste à trouver un ordre entre les jointures de chaque sous-hypergraphe He_{sub} pour les transformer en un graphe orienté et ainsi construire le plan global d'exécution des requêtes. Cet ordonnancement se base sur une *fonction bénéfice* qui dépend de la stucture d'optimisation à sélectionner. Pour les VM et le temps d'exécution par exemple, on cherche d'abord le noeud pivot qui va maximiser le bénéfice de réutilisation de son résultat par les différentes requêtes par rapport au coût de sa première exécution. Le noeud qui a le maximum de bénéfice sera ajouté au graphe, et supprimé de l'hypergraphe et ainsi de suite jusqu'à épuisement des noeuds (figure 5.4).

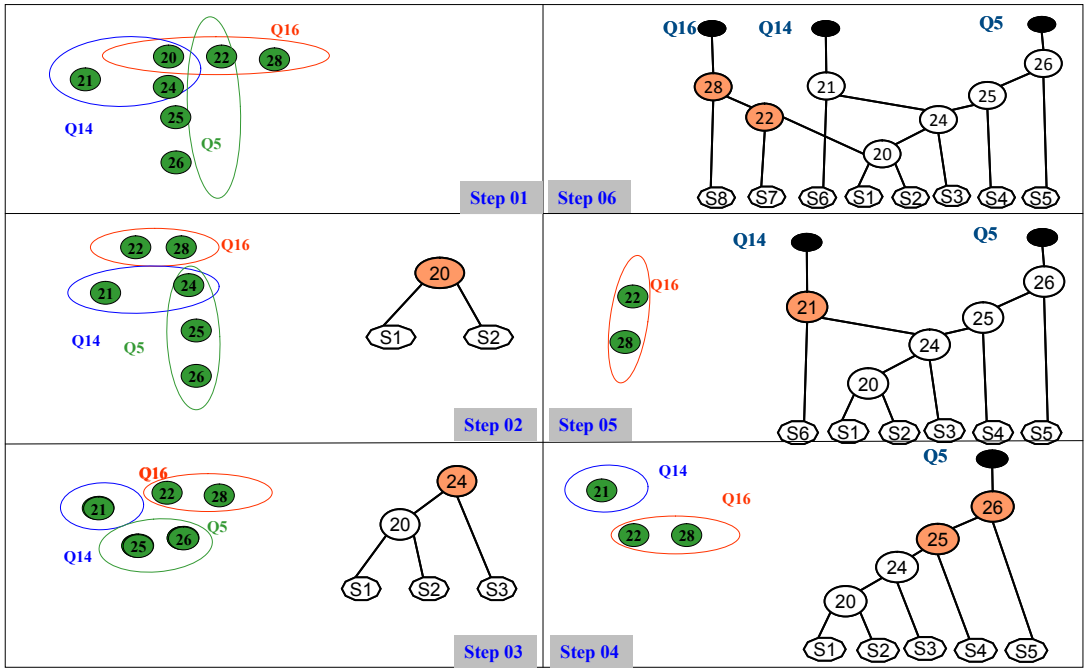


FIGURE 5.4 – Étapes de transformation d'un hypergraphe à un graphe orienté (plan global) [9]

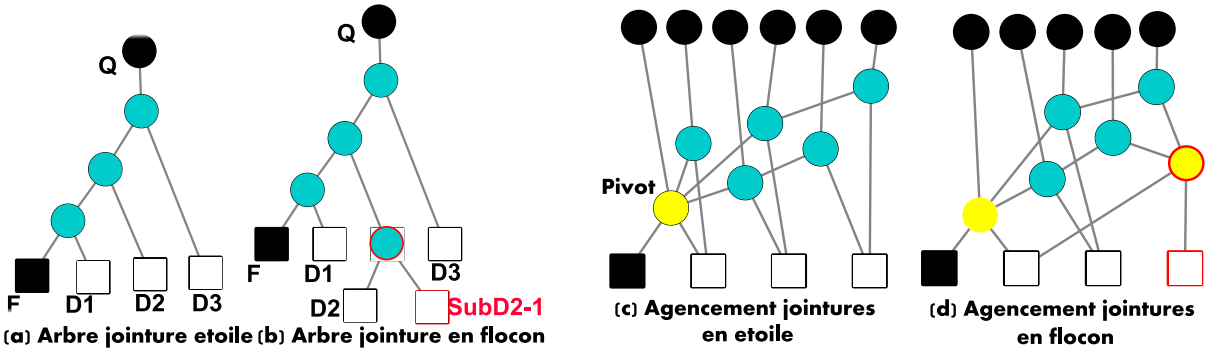


FIGURE 5.5 – Les agencements possibles des noeuds de jointure dans le plan global

Cette approche ne considère que les modèles multidimensionnels en étoile et ne traite donc pas les requêtes en flocon de neige. La section suivante explique notre contribution qui consiste à adapter cette approche à notre cas d'étude qui traite les deux types de schémas multidimensionnels.

3 Adaptation de l'approche de génération de graphe de requêtes

La variabilité de la conception logique est incarnée par la génération des différents schémas logiques qui incluent le schéma en étoile aussi bien que les schémas en flocon de neige. Ces derniers, non considérés par l'approche de base de Boukorca et al. [35], requièrent la modification des différentes étapes. Nous expliquons nos propositions concernant cette modification, étape par étape, dans les sections suivantes.

3.1 Construction des hypergraphes multidimensionnels

La considération de schémas en flocons de neige engendre l'existence de jointures supplémentaires entre les tables de dimensions et de sous-dimension qui n'impliquent pas la table de faits. Nous désignons ce type de jointures par "*jointure-dimension*" par opposition aux jointures en étoile désignées par *jointure-fait*. Comme illustré dans la figure 5.5, le deuxième et le quatrième agencement de noeuds sont impossibles dans l'approche de base et fréquents dans la notre. Nous avons en effet, plusieurs noeuds qui peuvent représenter le noeud de départ dans une composante connexe. Ce problème peut être contourné par la *fonction bénéfice* abordée dans la sous-section suivante.

3.2 Ordonnancement des sous-hypergraphes

Les *jointures-dimensions* introduisent un ordre partiel car ils doivent précéder les *jointures-faits* (les tables de sous-dimensions doivent être joints avant leur tables de dimensions). Cet ordre doit être considéré pendant l'ordonnancement des noeuds de jointure des sous-hypergraphes He_{sub} (étape n°3 du processus de construction du plan global) de manière à ce que ces deux règles soient respectées :

— la table de dimension "*mère*" ($D2$ dans la figure 5.5) ne doit apparaître dans une *jointure-fait* qu'après ses *jointures-dimensions* ;

— les *jointure-dimensions* externes doivent figurer avant les *jointure-dimensions* internes.

Le respect de ces deux règles est garanti par la fonction bénéfice suivante dont le but est de trouver le noeud pivot et donc d'ordonner les sous-hypergraphes He_{sub}. Soient n_{f_i} une *jointure-fait*, n_{e_i} une *jointure-dimension*, k le nombre de *jointure-dimension* impliquées par une *jointure-fait*, *nbr* le nombre de requêtes utilisant le noeud en question, *cout* le coût de traitement de ce noeud, alors :

$$\begin{cases} cout_{total}(n_{f_i}) &= cout(n_{f_i}) + \sum_{j=1}^{k} cout(n_{e_j}) \\ benefit(n_{e_i}) &= (nbr-1) * cout(n_{e_i}) - cout(n_{e_i}) \\ benefit(n_{f_i}) &= (nbr-1) * cout(n_{f_i}) + \\ & \quad \sum_{j=1}^{k}((nbr-1) * cout(n_{e_j})) \\ & \quad -cout(n_{f_i}) - \sum_{j=1}^{k} cout(n_{e_j}) \ldots (\blacktriangledown) \end{cases}$$

La table de faits étant très large par rapport aux autres tables, et le nombre de réutilisation d'une *jointure-fait* étant toujours supérieur ou égal à celui de ses *jointures-dimensions*, une *jointure-dimension* ne peut donc pas être le noeud pivot. En effet, le bénéfice de réutilisation de cette dernière est toujours inférieur à celui d'une *jointure-fait*. Pour cette raison, nous incluons les bénéfices des *jointures-dimensions* dans celles des *jointures-faits* correspondantes (équation ▼). La définition du noeud pivot d'un hypergraphe devient alors décrite par l'algorithme 4. Celui-ci décrit la recherche du noeud qui va maximiser non seulement le bénéfice de réutilisation de son résultat par les différentes requêtes par rapport au coût de sa première exécution, mais aussi le bénéfice de ses jointures-dimensions. Le noeud qui a le maximum de bénéfice sera ajouté au graphe GRAPHE, et supprimé de l'hypergraphe et ainsi de suite jusqu'à épuisement des noeuds.

3.3 Transformation d'un hypergraphe en graphe

Cette étape est décrite par l'algorithme 5 qui commence par chercher le noeud pivot de bénéfice maximal. Si ce dernier est utilisé par toutes les hyperarêtes (toutes les requêtes), alors le noeud pivot sera supprimé de l'hypergraphe et ajouté au graphe GRAPHE avec tous ses éventuels noeuds de *jointures-dimensions*. Le reste des étapes restent les mêmes que celles

Algorithm 4 chercherPivot(HyperGraphe He)

```
1: beneficemax ← 0;
2: for all s_i ∈ S do                                   ▷ S ensemble de sommets
3:     if s_i instanceof jointure-fait then
4:         nbr ← nbrUse(s_i);          ▷ avoir le nombre hyperarêtes qui utilisent le sommet s_i
5:                                          ▷ le nombre de réutilisations est nbr − 1
6:         cost ← coutExecution(s_i); ▷ le coût d'exécution de ce sommet s_i , dans le graphe G
7:         for j = 1 to k do     ▷ k est le nombre de jointures-dimensions de cette jointure-fait
8:             cost ← cost + coutExecution(s_{i_j})
9:         end for
10:        benefice ← (nbr − 1) × cost − cost        ▷ le bénéfice de réutilisation de sommet s_i
11:        if benefice < beneficemax then
12:            beneficemax ← benefice;
13:            pivot ← s_i ;
14:        end if
15:    end if
16: end for
17: return pivot
```

d'origine : après la suppression du noeud pivot, les hyperarêtes ne reliant plus aucun sommet sont supprimées de l'hypergraphe. Si le noeud pivot n'est pas utilisé par toutes les hyperarêtes, alors il sera ajouté au graphe GRAPHE, et l'hypergraphe sera partitionné en deux hypergraphes disjoints : $He1$ contenant les hyperarêtes utilisant le noeud pivot et $He2$ contenant les autres hyperarêtes. $He1$ et $He2$ seront transformés en un graphe en suivant les mêmes étapes précédentes 5.4.

Maintenant que nous pouvons représenter le graphe des requêtes des schémas logiques en étoile ou en flocon de neige, nous allons pouvoir évaluer l'impact de ces derniers sur l'efficacité des VM dans l'optimisation physique. L'évaluation se fait en termes de temps d'exécution et/ou de l'énergie consommée. La sélection de la meilleure variante est expliqué dans la section suivante.

4 Sélection de la meilleure variante

Dans l'optique d'étudier l'impact interphase de la variabilité logique sur l'optimisation physique, la meilleure variante est le modèle logique permettant d'obtenir des VM optimisant le mieux possible les BnF de l'ED. Nous étudions deux scénarios. Le premier concerne le cas où le concepteur ne s'intéresse qu'au BnF lié au temps d'exécution. Le second scénario concerne le cas où il s'intéresse aux deux BnF que sont le temps d'exécution et l'énergie consommée en même temps.

Algorithm 5 transformerHyperGraphe(HyperGraphe He)

```
while S non vide do  ▹ He = {S, HA}, S est l'ensemble des sommets, HA est l'ensemble des hyperarêtes
    pivot gets chercherPivot(He);                                              ▹ Algorithme 4
    if pivot ∈ tous(hyper ∈ HA) then
        ajouterSommetAuGraphe(pivot);        ▹ Ajouter le sommet pivot au graphe GRAPHE
        for j = 1 to k do                    ▹ k est le nombre de jointures-dimensions de ce pivot
            ajouterSommetAuGraphe(s_ij)
        end for
        supprimerSommet(pivot);              ▹ Supprimer le sommet pivot de S de He
        for all hyper_i ∈ HA do
            if |hyper_i| = 0 then
                supprimerHyperArete(hyper_i);   ▹ Supprimer les hyperarêtes n'ayant pas de sommets
            end if
        end for
    else
        partitionnerPivot(pivot, He1, He2);     ▹ Algorithme de partitionnement [35]
        ajouterSommetAuGraphe(pivot);
        transformerHyperGraphe(He1);
        transformerHyperGraphe(He2);
    end if
end while
```

4.1 Sélection mono-objectif : temps d'exécution

Dans ce scénario, la meilleure variante est le modèle logique permettant d'obtenir des VM optimisant le mieux possible le temps d'exécution de l'ED. Pour ce faire, nous créons la structure de graphe des requêtes pour chaque schéma logique possible. Ensuite, le noeud pivot (le plus avantageux) de chaque graphe est matérialisé, les requêtes réécrites selon chaque schéma sont exécutées et les résultats en termes de temps d'exécution sont comparés. Le schéma ayant le plus petit temps d'exécution est sélectionné comme étant la meilleure variante.

4.2 Sélection multi-objectifs : algorithme évolutionnaire

Dans le cas où le concepteur veut optimiser le temps d'exécution en même temps que la consommation énergétique, la meilleure variante est le modèle logique permettant d'obtenir des VM optimisant le mieux possible le compromis souhaité entre le temps d'exécution et l'énergie consommée de l'ED.

Matérialiser le noeud pivot n'a pas de sens si l'on cherche à réduire la consommation énergé-

Algorithm 6 Sélection du schéma logique ayant le meilleur impact physique

Input: BnF, $\mathcal{LM}$: un ensemble de schémas logiques / $\mathcal{LM}_i = \{F, D_j, SubD_{jk}\} / j \in \{1..n\}, k \in \{1..n_k\}, \mathcal{Q}_i = \{q_{i_1}, q_{i_2}, ..., q_{i_m}\}$

Output: Un ensemble de noeuds de jointure à matérialiser (configuration de VM)

1: **for** $\mathcal{LM}_i \in \mathcal{LM}$ **do**
2: générer le graphe de requêtes $\mathcal{Q}_i$ correspondantes ;
3: **if** BnF = *performance* **then**
4: le noeud pivot de chaque of composante connexe du graphe est matérialisé ;
5: **else** ▷ BnF = *énergie* & *temps d'exécution*)
6: Annoter chaque noeud de jointure par son temps d'exécution et sa consommation d'énergie ;
7: Appliquer un AE afin de sélectionner les vues candidates pour être matérialisées car elles optimisent le *temps d'exécution* et la *consommation d'énergie* ;
8: Appliquer la somme pondérée sur cet ensemble de VM pour sélectionner une seule VM ;
9: **end if**
10: **end for**

tique, car il est choisi pour avantager le temps d'exécution sans prendre en compte la dimension énergétique. Matérialiser tous les noeuds de jointure du graphe n'est pas réalisable non-plus à cause du coût de stockage engendré et du fait que cela peut entraîner les plus importantes dépenses énergétiques [156]. Tester 2^n configurations possibles de VM (n est le nombre des noeuds de jointure) pour trouver les solutions *Pareto* est impossible, surtout dans le contexte des ED impliquant un très grand nombre de jointures. Les algorithmes évolutionnaires (AE) par contre sont appropriés aux problèmes d'optimisation multi-objectifs où un grand espace de recherche est impliqué et plusieurs alternatives de compromis peuvent être générés en une seule passe d'exécution. L'idée générale derrière les AE est de simuler le processus d'évolution de la nature dans lequel les *individus les plus aptes* peuvent survivre après plusieurs générations [197]. Leur but est de trouver un ensemble de solutions se rapprochant le plus possible de la frontière de *Pareto*. Pour la sélection du meilleur modèle logique ayant le meilleur impact sur l'optimisation physique dans les deux scénarios, nous suivons l'algorithme 6.

Notons que les AE multi-objectifs produisent un ensemble de solutions. Pour donner au concepteur la possibilité de choisir la meilleure solution parmi cet ensemble selon le compromis souhaité (entre le temps et l'énergie), nous proposons d'utiliser la méthode des sommes pondérées. Il est important de noter que l'application de la somme pondérée dès le départ pour évaluer toutes les solutions candidates est impossible, même pour un petit nombre de tables à cause de la NP-Complétude du problème. Pour cette raison, nous avons employé les AE afin de générer un ensemble de solutions *Pareto* uniquement, puis nous avons appliqué la somme pondérée afin de ne choisir qu'une seule solution.

5 Validation et expérimentations

Nous utilisons la même configuration matérielle utilisée dans nos expérimentations pour l'évaluation de la consommation énergétique dans la section 4.4.3. Concernant les données, nous utilisons celles du benchmark SSB avec un facteur d'échelle égal à 10. Nous avons également créé une charge de requêtes orientée optimisation multi-requêtes (voir annexe B).

5.1 Évaluation du scénario mono-objectif

Nous nous focalisons dans ce scénario sur le problème de sélection de VM en considérant la variabilité logique. Nous avons adapté l'outil JAVA de Boukorca et al. [35] pour qu'il génère le graphe de requêtes suivant le modèle logique en étoile ou en flocon. Nous l'avons équipé avec nos MdC mathématiques pour estimer le temps d'exécution et l'énergie. La figure 5.6 présente les résultats de nos simulations (en I/O) pour les différents schémas/requêtes avec ou sans les VM. Nous constatons que le schéma 5 par exemple, enregistre un petit nombre d'I/O avec la définition des VM ($26 * 10^7$ I/O). Ce nombre augmente en absence de VM ($32 * 10^7$ I/O). Ce résultat se généralise sur l'ensemble des schémas. Cela témoigne de l'importance de la matérialisation des vues sur le temps d'exécution, ce qui est un résultat escompté qui peut prouver partiellement la cohérence de notre MdC. Un autre constat concerne les schémas entre eux. Autrement dit, l'optimisation de temps d'exécution (I/O) faite grâce au VM change selon le schéma logique : il existe environ $34 * 10^7$ I/O de différence entre les schémas 5 et 16 en présence de MV, et une différence de plus de $40 * 10^7$ de I/O en absence de MV. Cela témoigne de l'importance de l'impact de la variabilité logique sur l'optimisation physique.

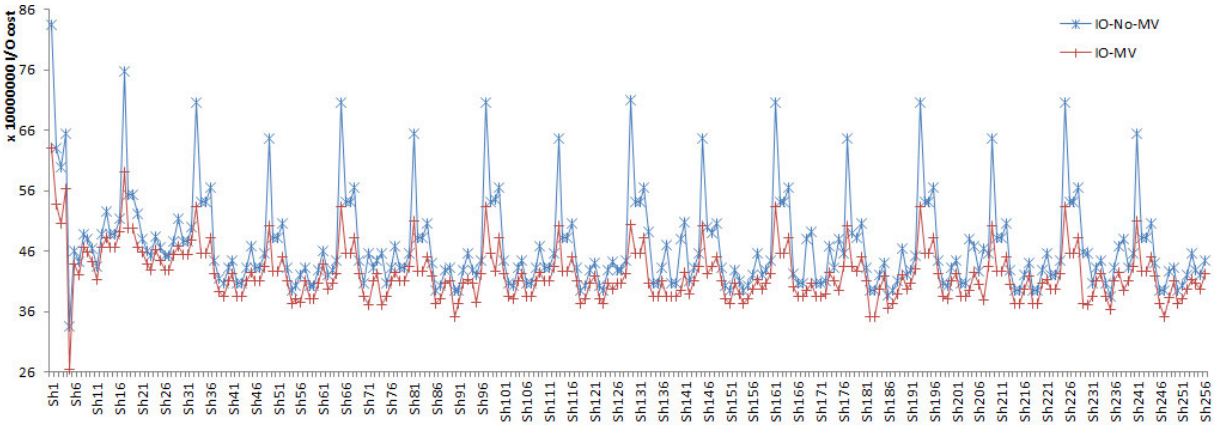


FIGURE 5.6 – L'impact de la variabilité logique sur la performance des VM

5.2 Évaluation du scénario multi-objectifs

Au lieu de tester de façon combinatoire toutes les configurations (256 schémas, 30 requêtes et *n* vues générées par les algorithmes évolutifs, pour chaque schéma), il est plus sensé de sélectionner tout d'abord trois schémas logiques générés par notre simulateur en utilisant nos MdC : un premier schéma orienté temps d'exécution, un deuxième orienté énergie et un troisième orienté compromis *50/50*. Nous avons créé pour chaque schéma son graphe de requêtes. En utilisant *MOEA* [49] que nous avons intégré au simulateur, l'ensemble des noeuds *Pareto* à matérialiser pour chaque schéma est généré. Nous nous sommes contentés de l'utilisation des paramètres par défaut de l'AE et qui sont décrits dans le tableau 5.1, car ces derniers nous ont prouvé l'existence de l'impact de la variation logique. Pour chacune des configurations de VM générées (jointures) par le framework *MOEA*, le simulateur calcule le temps et l'énergie consommés à l'aide des MdC. Afin de sélectionner une seule configuration de vues avec le compromis souhaité, nous appliquons la méthode de la somme pondérée sur cet ensemble de Pareto avec les compromis suivants : *Power-MV* pour les vues orientées énergie, *Time-MV* pour les vues orientées temps d'exécution et *tradoff-MV* pour les vues orientées 50% temps et 50% énergie. Notons que les coûts d'E/S ont été convertis en valeurs temporelles (ms).

Paramètre	Valeur
Type d'encodage	chaîne de bits
Méthode de sélection	tournoi binaire
taille de la sélection du tournoi	2
Type de croisement	Half Uniform
Probabilité de croisement	1,0
Type de mutation	Flip Bit
Probabilité de mutation	0,01
Nombre maximal d'évaluations	10,000

TABLE 5.1 – Les paramètres de l'algorithme évolutionnaire

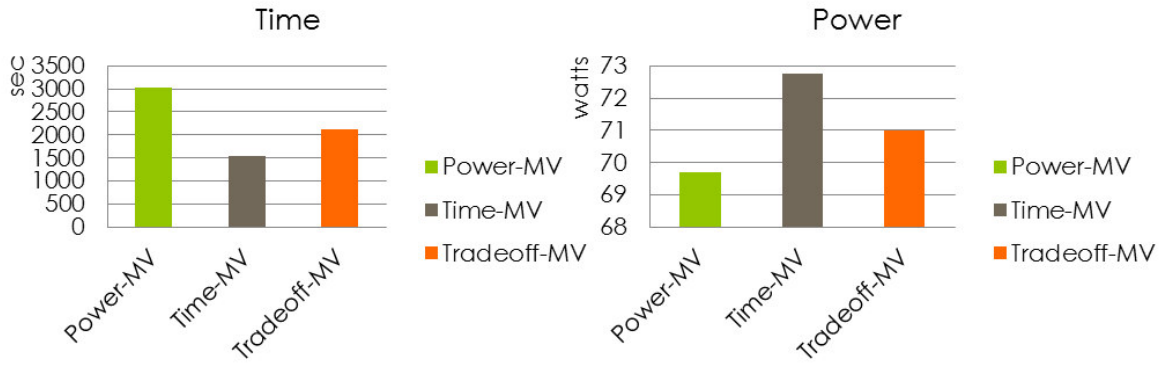
Nos expériences (dont certaines sont décrites dans la figure 5.7) montrent les faits suivants.

— Pour orienter un concepteur vers un BnF donné à des stades antérieurs, il doit combiner le compromis approprié du schéma logique avec les structures d'optimisation. En effet, en choisissant un schéma logique orienté compromis (figure 5.7-a), nous constatons que les VM orientés compromis, donnent bien des valeurs moyennes de temps et d'énergie. De même, dans la figure 5.7-b, en choisissant un schéma logique orienté temps, nous constatons que les VM orientés temps, donnent les meilleurs valeurs de temps par rapport aux autres VM orientés énergie ou compromis. La même chose s'applique sur le schéma orienté énergie (figure 5.7-c) où les VM orientés énergie donnent en effet la meilleure valeur en comparant avec les VM orientées temps et compromis.

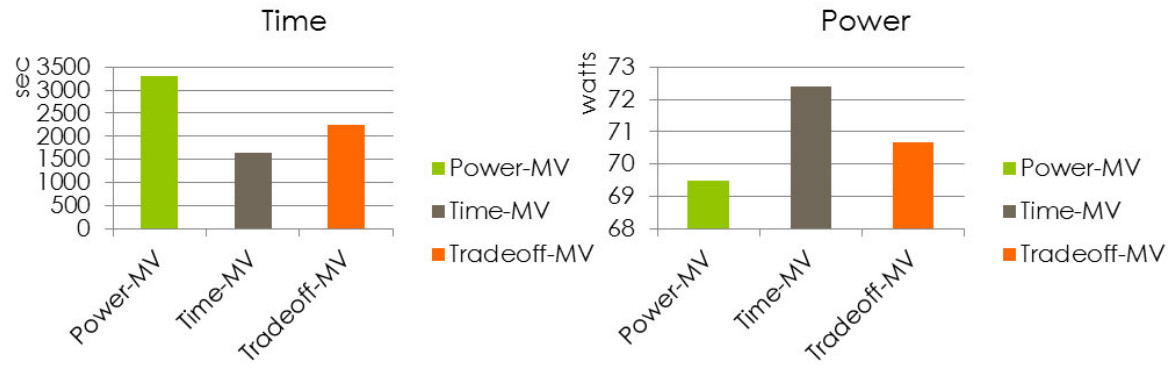
49. Bibliothèque Java pour les AE multi-objectifs. `Www.moeaframework.org`

— Les schémas logiques destinés à améliorer un BnF (performance/énergie), ne donnent pas forcément les valeurs optimales en présence de structures d'optimisation. Cependant, ils ne donnent pas les pires valeurs. En effet, il est vrai que les VM orientées temps dans le schéma orienté temps (figure 5.7-b) donnent la meilleure valeur en comparant avec les autres VM du même schéma, mais cette valeur n'est pas l'optimale, car celle-ci est obtenue avec les VM orientées temps dans le schéma logique orienté énergie.

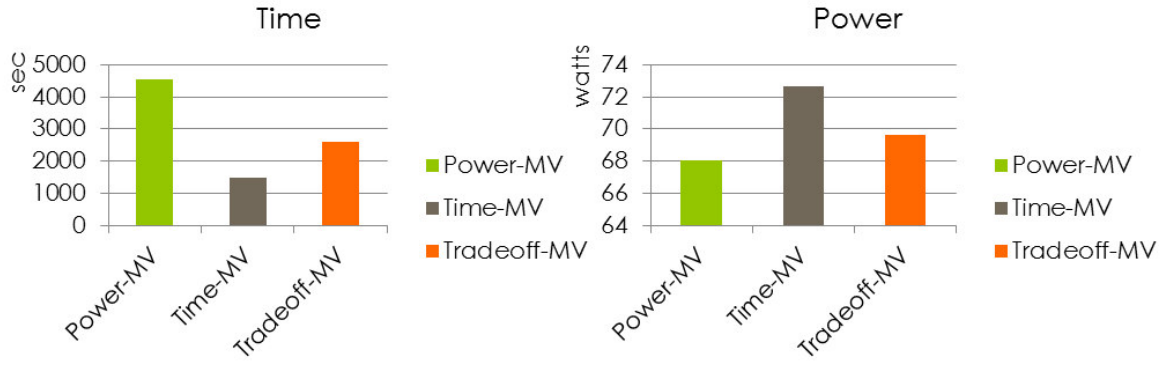
— Ces résultats confirment le besoin d'un processus holistique de conception de BD orientée-variabilité où les interdépendances entre les phases doivent être considérées.



(a) Schéma logique orienté compromis 50/50



(b) Schéma logique orienté temps



(c) Schéma logique orienté énergie

FIGURE 5.7 – Impact de la variabilité logique associée aux VM sur l'énergie et le temps dépensés

6 Conclusion

Pour étudier l'impact interphase de la variabilité logique, nous avons choisi l'optimisation physique comme cible étant une tâche directement liée aux BnF, et plus précisément la technique des VM. Nous avons adapté une approche flexible basée sur les hypergraphes pour la représentation de graphe des requêtes en étoile et en flocon de neige. La sélection des VM

pour l'optimisation de temps d'exécution de chaque schéma revient à trouver le noeud pivot du graphe de requêtes correspondant. Le schéma logique ayant les meilleurs résultats est sélectionné avec les VM à matérialiser. En ce qui concerne la sélection multi-objectifs, nous avons fait appel au framework MOEA qui implémente un AE pour générer les configurations de VM optimisant le temps et l'énergie, puis nous appliquons la méthode de la somme pondérée afin de choisir une seule configuration. Les résultats montrent que les schémas logiques destinés à améliorer un BnF (performance/énergie), ne donnent pas forcément les valeurs optimales en présence de structures d'optimisation. Cependant, ils ne donnent pas les pires valeurs.

Ce chapitre montre l'impact de la variabilité logique sur le problème de sélection de VM et d'une manière plus générale l'importance de la gestion holistique de la variabilité de conception de BD. En effet, les résultats obtenus nous montre que l'implémentation de la vision de *conception de BD comme une LdP* est faisable car nos algorithmes de génération de graphes de requêtes des schémas logiques ainsi que nos MdC d'évaluation de différents BnF et les méthodes de sélection, sont pluggables pour implémenter les features de nos FM. Cela se fait grâce aux outils orientés gestion de variabilité (*par exemple FeatureIDE*). En outre, les résultats mettent en avant l'importance de la variabilité logique, jusque là sous-estimée, à travers la détection de son impact sur les BnF orientés BD.

Conclusion et perspectives

Nous présentons dans ce chapitre un bilan général synthétisant nos principales contributions, ainsi qu'un ensemble de perspectives qui s'ouvrent à l'issue de notre étude.

Conclusion

Dans cette thèse nous nous sommes intéressés au problème de la gestion de la variabilité dans le contexte de la conception des BD. Les principales contributions de notre travail sont les suivantes :

État de l'art

Nous avons mené une enquête sur l'évolution technologique des SGBD, basée sur le classement du mois d'octobre 2016 livré par le site spécialisé *DB-Engines* [50]. Nous avons distingué trois stades temporels : *maturation*, *maturité* et *dynamique*. Ces stades coïncident avec les stades de l'évolution du processus de conception des BD, à quelques différences près relatives au décalage temporel qui puisse exister entre la publication de la nouvelle technique (évolution) et de son implémentation par le SGBD. Cette enquête nous a permis d'établir un lien fort entre l'évolution des SGBD et celle du cycle de conception des BD.

Cette dernière a en effet, fait l'objet d'une synthèse des principales évolutions l'ayant concernée qui a mis en avant les mêmes stades temporels mais également l'omniprésence de la variabilité, et le besoin urgent d'une méthodologie visant à la gérer. Cela est incarné par la multitude de choix de conception dont font face les concepteurs et les administrateurs, et ce tout au long du processus de conception.

Cette diversité a pour conséquence l'augmentation de la complexité du processus de conception. C'est pour cette raison que nous avons élaboré une synthèse des principaux travaux pou-

50. `http://db-engines.com/en/ranking`

vant entrer dans le cadre de la gestion de la variabilité de la conception des BD. Nous en avons distingué deux principales catégories : des travaux orientée gestion implicite de la variabilité et des d'autres orientés gestion explicite. Par *implicite*, nous voulons dire que ces travaux n'étaient pas initialement conçus pour la gestion de la variabilité, et donc la traitent d'une manière partielle. Ces derniers s'articulent autour de deux axes : la réutilisation et l'aide à la décision. Quant aux travaux orientés gestion explicite de variabilité, nous avons choisi les technique des lignes de produits. Celui-ci a révélé que les différents outils et approches proposés s'intéressent aux phases de conception de manière séparée, et ne fournissent pas de gestion globale de conception des BD ignorant ainsi les interdépendances entre les phases.

Le cœur de notre travail a été présenté dans la deuxième partie de ce manuscrit que nous avons organisé en trois principales contributions.

Conception des BD comme une ligne de produits

Le besoin de la maîtrise de la variabilité dans le contexte de la conception des BD, nous a motivé à proposer une méthodologie pour la gestion holistique de cette variabilité. Celle-ci se base sur la technique des lignes de produits réputée pour être efficace dans ce domaine et permet de voir l'ensemble des phases de conception d'une BD comme une LdP. Ainsi, les différents choix de conception sont exposés aux concepteurs en offrant en plus un mécanisme pour les aider dans la sélection du meilleur choix. Par ailleurs, le caractère *holistique* de cette vision permet de considérer les interdépendances qui puissent exister entre les phases de ce processus. Cette vision de BD comme une LdP est comparable à une boîte noire configurable, alimentée par un ensemble d'entrées (requêtes, statistiques et besoins utilisateurs/schéma), pour produire en sortie un script correspondant à une *BD* prête à être déployée dans un environnement prédéfini.

La réalisation de cette vision passe par des étapes prédéfinies. La première étape consiste à analyser la variabilité de la conception des BD en identifiant ses features et leur interdépendances à l'aide des *modèles des features*. L'avantage de l'utilisation de ces modèles est l'existence de plusieurs outils (comme *FeatureIDE*) permettant d'implémenter ces features, leur interdépendances et dériver des configurations (BD prêtes à être déployées), ce qui représente la deuxième étape de la réalisation de notre vision. Le choix entre ses configurations peut être représenté par un arbre dont les chemins sont pondérés selon plusieurs BnF (temps d'exécution, énergie et espace de stockage). Le chemin ayant le poids minimal représente la BD à sélectionner. À première vue, notre LdP peut être assimilé à un outil CASE de conception de BD, comme *PowerDesigner* [158] par exemple, car ce dernier génère aussi des scripts correspondants à certains choix de conception. Nous notons cependant que *PowerDesigner* et les outils équivalents s'intéressent principalement à l'aspect modélisation. Ils n'offrent pas la prise en compte des différentes variantes ni leur interdépendances, et n'aident pas le concepteur à faire un choix parmi ces variantes qui existent tout au long du cycle de conception. Notre objectif, quant à lui, peut être assimilé à une sorte d'*advisor holistique*, qui accompagne le concepteur à travers toutes

les étapes du processus en lui offrant un haut degré de configurabilité et de réutilisation et l'automatisant autant que possible. Cet *advisor* peut être utilisé pour la conception d'une nouvelle BD/ED ou d'une BD/ED déjà déployé (sous production).

L'envergure de la réalisation de cette vision nous impose de se focaliser sur la gestion de la variabilité de la conception logique, jusque là sous-étudiée (cas restreint).

Étude de l'impact de la variabilité sur l'optimisation logique des BD

Nous instancions les objectifs de la gestion de la variabilité de la conception des BD dans la phase logique. Ainsi, nous expliquons d'abord comment exploiter les corrélations afin de générer différents schémas logiques (choix de conception) à être considérés par le concepteur. Puis, nous étudions l'impact de cette variabilité de schéma sur l'optimisation logique des requêtes (impact intraphase). Celui-ci doit être évalué afin d'aider le concepteur à choisir parmi cette panoplie de schémas logiques (pondérer les chemins de l'arbre correspondant). Cette évaluation est traditionnellement faite en fonction des BnF classiques que sont le temps d'exécution et l'espace de stockage. Nous avons également intégré la dimension énergétique dans cette évaluation. Afin d'automatiser cette dernière, nous proposons des modèles de coûts relatifs à chaque BnF. Le choix du meilleur schéma correspond au schéma ayant le meilleur impact (les meilleures valeurs renvoyées par les MdC et qui servent à pondérer les chemins) et ce en fonction d'un seul ou de plusieurs BnF (sélection mono ou multi-objectives). Les résultats sont appuyés par des expérimentations théoriques et empiriques.

Notons que le fait de varier le schéma logique ait un impact sur les BnF de la BD peut être un résultat escompté. Cependant, le but principal de nos propositions est d'inculquer cette habitude de conception orientée variabilité aux concepteurs en leur offrant des algorithmes pour automatiser cette gestion.

Étude de l'impact de la variabilité sur l'optimisation physique des BD

Afin d'instancier les objectifs de la gestion de la variabilité *holistique* de la conception des BD dans la phase logique, nous devons considérer un impact *interphase*. Notre choix s'est porté sur l'optimisation physique comme feature pour refléter ce type d'impact, étant une tâche directement liée aux BnF. Cette optimisation s'appuie sur plusieurs techniques physiques (comme les index, le partitionnement). Nous avons choisi les vues matérialisées. Nous avons utilisé une approche basée sur les hypergraphes pour la représentation du plan global des requêtes. Cette structure efficace est exploitée pour la sélection des vues à matérialiser selon un ou plusieurs BnF (sélection mono ou multi-objectives), et ce pour chaque schéma logique candidat. Nous utilisons un algorithme évolutionnaire dans le cas de la sélection multi-objective. Le schéma enregistrant les meilleurs valeurs de BnF est choisi. Nous utilisons les mêmes modèles de coûts pour évaluer l'impact de chaque choix logique sur le rôle des vues matérialisées dans l'optimi-

sation des différents besoins non fonctionnels.

À travers ces deux dernières contributions, nous avons montré que l'implémentation de la vision de *conception de BD comme une LdP* est faisable. En effet, nos algorithmes de génération/peuplement des schémas logiques, réécriture des requêtes, génération de graphes de requêtes (plan global) ainsi que nos MdC d'évaluation de différents BnF, sont pluggables sous forme de features implémentées par les outils orientés gestion de variabilité (*par exemple FeatureIDE*). Ces deux dernières contributions nous ont également permis de montrer l'importance de la variabilité logique, jusque là sous-étudiée. Elles nous ont permis aussi de rompre avec quelques habitudes liées à cette conception logique et qui consistent à figer un schéma (variabilité ignorée) et à favoriser le schéma en étoile. Ces habitudes ne sont plus valables dans un contexte où la performance n'est plus limitée au temps d'exécution uniquement et d'une manière générale dans un contexte aussi diverse, évolutif et donc variable. L'implémentation de tout le LdP de la conception des BD requiert la généralisation de ce cas d'étude à d'autres features.

Perspectives

Les travaux présentés dans ce manuscrit laissent envisager de nombreuses perspectives. Dans cette section, nous présentons celles qui nous paraissent être les plus importantes, suivant qu'elles s'inscrivent dans le court ou le moyen/long terme.

Perspectives à court terme

Les perspectives à court terme concernent notamment la réalisation de nouvelles expérimentations, la prise en compte de plus de corrélations, et l'étude de la variabilité sur d'autres phases.

Expérimentations plus poussées pour consolider les résultats de notre étude

Notre proposition de BD comme une LdP est une sorte de premier pas dans la gestion holistique de la variabilité du processus de conception des BD. Pour cette raison, nous étions plus dans la preuve de l'existence des différentes conséquences de la gestion holistique (interdépendance entre les phases de la conception logique et physique, intégration de la dimension énergétique et interdépendance entre les BnF sur ces phases), que dans l'évaluation poussée de ces conséquences avec plus d'expérimentations Ainsi, la première perspective porte sur la mise en oeuvre d'expérimentations plus solides dans l'étude des conséquences de la variabilité. Cela peut concerner plusieurs aspects.

— Augmentation du nombre de requêtes, et de la taille des données (100GB) pour l'éva-

luation de l'impact (intra et inter-phase) de la variabilité logique, que ce soit en matière de temps d'exécution ou de l'énergie consommée. Cela peut s'inscrire dans la vision de "BigQuery" traité par notre collègue *Boukorca* [9].

— Utilisation d'un schéma plus complexe pour l'évaluation de l'impact de la variabilité logique (TPC-DS [51]), mais aussi pour la validation de nos MdC.

— Changement de configuration matérielle pour la validation de notre MdC orienté énergie : réduction de mémoire, changement de processeur et de SGBD. Cela conduirait à un changement de comportement de consommation d'énergie qui est fortement liée à la configuration matérielle.

— Proposition d'une heuristique pour l'élagage de l'espace de recherche des schémas logiques à évaluer. L'assimilation de la dérivation des BD au problème de la recherche du meilleur chemin dans l'arbre des features nous offre la possibilité d'exploiter les algorithmes de ce domaine (à l'exemple de A*). Il serait intéressant de pouvoir identifier les schémas candidats les plus pertinents à travers des caractéristiques que nous leur affectons.

— Intégrer la contrainte d'espace lié au problème de sélection de VM.

La prise en compte des différents types de corrélations

La présentation multidimensionnelle des ED a connu un vif succès au sein des entreprises où elle est confrontée à l'avis des décideurs. Malinowski et Zimányi [119] ont proposé une classification des différents types de hiérarchies, en s'inspirant de situations réelles. Divers types de hiérarchies sont représentés : symétriques/asymétriques, strictes/non strictes, multiples, parallèles, etc. Il serait intéressant d'adapter notre algorithme de génération de différents décompositions possibles d'une hiérarchie (voir algorithme 4.1 pour qu'il prenne en considération ces types d'hiérarchies. Dans la même lignée, et dans les serveurs de BD avec des charges de requêtes mixtes (OLAP et OLTP), il faut également adapter notre algorithme pour gérer à la fois la décomposition des hiérarchies OLAP et la décomposition OLTP classique des formes normales. Cette dernière implique les dépendances fonctionnelles et multi-valuées.

Par ailleurs, dans les BD sémantiques à base ontologique, on distingue deux types de concept : *canoniques* et *non canoniques*, où le dernier est défini en fonction du premier en utilisant des constructeurs offerts par les modèles d'ontologies [82]. Des corrélations sont alors crées entre les propriétés mais surtout entre les *classes* du modèle conceptuel représenté par l'ontologie. Ces corrélations liant les concepts non-canoniques à ceux canoniques doivent également être pris en compte par notre algorithme 4.1.

51. http://www.tpc.org/tpcds/

Généralisation de la démarche de gestion de variabilité dans le cycle de vie

Dans notre thèse, nous nous sommes focalisés sur l'étude de la variabilité de la conception logique et de son impact interphase sur l'optimisation de la conception physique en outre de son impact intraphase. L'implémentation de la vision BD comme une LdP revient à généraliser ce genre d'études sur l'ensemble des features du processus de conception des BD (voir figure 8). Ces features sont identifiées lors de l'analyse holistique de la variabilité (modèle de features). À titre d'exemple, nous pouvons s'intéresser à l'étude de l'impact de la variabilité logique sur

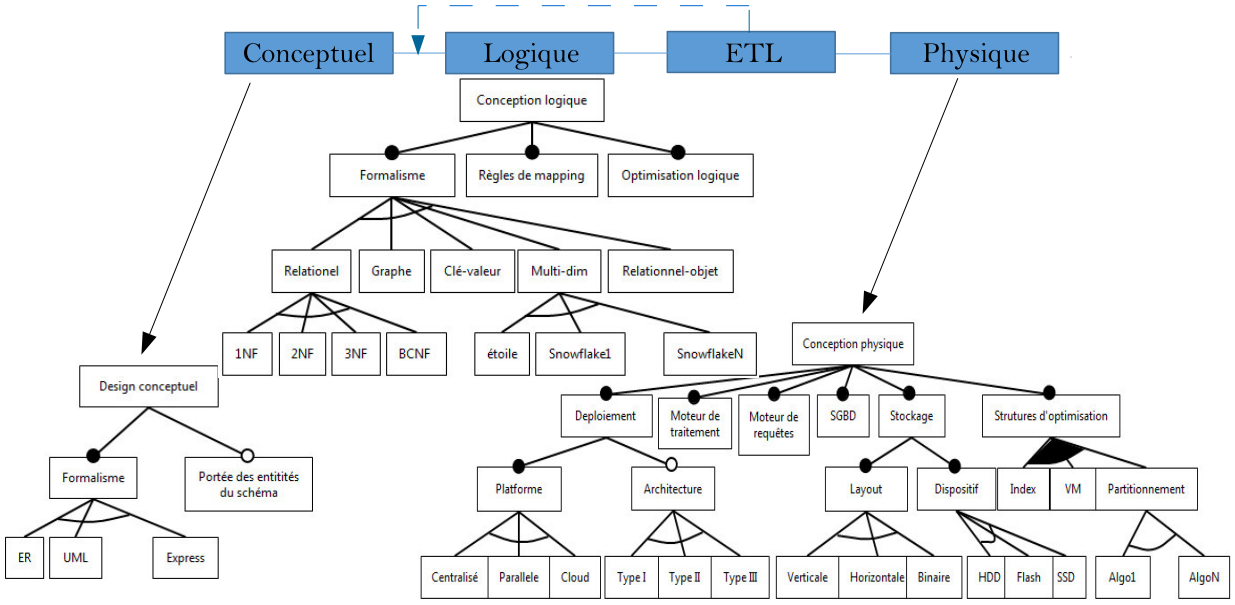


FIGURE 8 – Gestion holistique de la variabilité de la conception des BD

la phase *ETL* des ED, plus précisément sur la définition des opérateurs ETL. L'impact sur le *déploiement* peut également être automatiquement déduit car il peut être identifié pour chaque flux ETL logique [187]. Un autre exemple concerne la prise en compte des structures d'optimisation autres que les vues matérialisées pour refléter la variabilité logique. Le partitionnement horizontal est facilement intégrable, car le processus de la sélection des partitions se base sur le graphe de requêtes. Il suffit juste de trouver la fonction du bénéfice correspondante qui permet de partitionner l'hypergraphe.

Perspectives à moyen et long terme

Les perspectives à moyen et long terme concernent notamment notre ligne de produits, et donc la modélisation et l'implémentation de la gestion de la variabilité du processus de conception des BD. En effet, la réalisation de ces deux tâches s'inscrit dans la durée.

Analyse de la variabilité de conception des BD

Les FM que nous avons proposés ne sont pas exhaustifs, comme par exemple les features *SGBD* et *moteur de requêtes* de la phase physique. Il va falloir élaborer des analyses de ces aspects absents et enrichir les FM correspondants ainsi que leurs dépendances. Cet enrichissement peut conduire à des FM très larges dont la gestion d'interdépendances peut devenir très complexe. Pour pallier à ce problème, il est nécessaire d'adapter des techniques de *variabilité multi-dimensionnelle* [160, 153].

Conception collaborative dans les BD

L'état de l'art concernant le cycle de conception des BD que nous avons élaboré, a montré l'augmentation de la complexité de ce processus. Notre proposition de BD comme une LdP témoigne également de la complexité de la gestion de la variabilité holistique de la conception des BD. En effet, les principales propositions de cette thèse découlent d'un important constat : le cycle de conception des BD a évolué, mais les méthodes de conception proposées ne s'adaptent pas à cette évolution. Ce constat nous motive à reconsidérer la *conception séquentielle* actuelle des BD comme une *conception collaborative*. L'effort collectif dans sa gestion de variabilité : l'identification des features, des interdépendances et leur implémentation, est une réponse à la complexité de ce processus. Dans cette optique, il serait intéressant de mettre en ligne notre LdP et inciter les chercheurs à collaborer pour son enrichissement.

Bibliographie

[1] Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, Magdalena Balazinska, Philip A. Bernstein, Michael J. Carey, Surajit Chaudhuri, 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, January 2016.

[2] Alberto Abelló, José Samos, and Félix Saltor. A framework for the classification and description of multidimensional data models. In *Database and Expert Systems Applications*, pages 668–677. Springer, 2001.

[3] Lamia Abo Zaid and Olga De Troyer. Towards modeling data variability in software product lines. In *Enterprise, Business-Process and Information Systems Modeling: 12th International Conference, BPMDS 2011, and 16th International Conference, EMMSAD 2011, held at CAiSE 2011, London, UK, June 20-21, 2011. Proceedings*, pages 453–467, 2011.

[4] Christopher Adamson. *Mastering data warehouse aggregates: solutions for star schema performance*. Wiley, 2012.

[5] R. et al. Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, and Gerhard Weikum. The claremont report on database research. *SIGMOD Rec.*, 37(3):9–19, September 2008.

[6] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy,

Joseph M. Hellerstein, Yannis E. Ioannidis, Henry F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, and Gerhard Weikum. The claremont report on database research. *SIGMOD Record*, 37(3):9–19, 2008.

[7] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[8] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database tuning advisor for microsoft sql server 2005. In *In Proceedings of the International Conference on Very Large Databases, (VLDB'04)*, pages 1110–1121, 2004.

[9] Boukorca Ahcène. *Hypergraphs in the Service of Very Large Scale Query Optimization - Application: Data Warehousing. To appear*. PhD thesis, Université de Poitiers, dec 2016.

[10] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Interaction-aware scheduling of report-generation workloads. *The VLDB Journal*, 20(4):589–615, 2011.

[11] Scott W. Ambler and Pramodkumar J. Sadalage. *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Professional, 2006.

[12] Kerkad Amira. *L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels*. PhD thesis, dec 2013.

[13] S. Anderlik, B. Neumayr, and M. Schrefl. Using domain ontologies as semantic dimensions in data warehouses. *In Proc. of LNCS*, pages 88–101, 2012.

[14] Timo Asikainen, Timo Soininen, and Tomi Männistö. A koala-based approach for modelling and deploying configurable software product families. In *Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003, Revised Papers*, pages 225–249, 2003.

[15] Sandeepan Banerjee, Vishu Krishnamurthy, and Ravi Murthy. All your data: The oracle extensibility architecture. In *Compontent Database Systems*, pages 71–104. 2001.

[16] Don Batory. Feature models, grammars, and propositional formulas. In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pages 7–20, 2005.

[17] D.S. Batory, J.R. Barnett, J.F. Garza, K.P. Smith, K. Tsukuda, B.C. Twichell, and Wise. Genesis: an extensible db management system. *Software Engineering, IEEE*, 1988.

[18] Martin Becker. Towards a general model of variability in product families. In *in Proceedings of the First Workshop on Software Variability Management*, 2003.

[19] L. Bellatreche, K. Boukhalfa, P. Richard, and K.Y. Woameno. Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(4):1–23, 2009.

[20] L. Bellatreche, K. Karlapalem, and A. Simonet. Algorithms and support for horizontal class partitioning in object-oriented databases. *In the Distributed and Parallel Databases Journal*, 8(2):155–179, April 2000.

[21] L. Bellatreche, S. Khouri, I. Boukhari, and R. Bouchakri. Using ontologies and requirements for constructing and optimizing data warehouses. In *(MIPRO)*, pages 1568–1573, 2012.

[22] Ladjel Bellatreche, Kamel Boukhalfa, and Pascal Richard. Primary and referential horizontal partitioning selection problems: Concepts, algorithms and advisor tool. In *Integrations of Data Warehousing, Data Mining and Database Technologies - Innovative Approaches.*, pages 258–286. 2011.

[23] Ladjel Bellatreche, Amine Roukh, and Selma Bouarar, editors. *Step by Step Towards Energy-aware Data Warehouse Design - 5th European Summer School on Business Intelligence (eBISS 2016), Tours, France, July 3-8, 2016, Tutorial Lectures*, Lecture Notes in Business Information Processing. Springer, 2016.

[24] Ladjel Bellatreche and Komla Yamavo Woameno. Dimension table driven approach to referential partition relational data warehouses. In *DOLAP 2009, ACM 12th International Workshop on Data Warehousing and OLAP, Hong Kong, China, November 6, 2009, Proceedings*, pages 9–16, 2009.

[25] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011.

[26] Collin Bennett, Robert L. Grossman, David Locke, Jonathan Seidman, and Steve Vejcik. Malstone: Towards a benchmark for analytics on large data clouds. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 145–152, 2010.

[27] Mbaiossoum Bery. *Conception physique des bases de données à base ontologique : le cas des vues matérialisées*. PhD thesis, dec 2014.

[28] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[29] Christian Böhm. A cost model for query processing in high dimensional data spaces. *ACM Trans. Database Syst.*, 25(2):129–178, 2000.

[30] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co., 2000.

[31] Selma Bouarar, Ladjel Bellatreche, Stéphane Jean, and Mickaël Baron. Do rule-based approaches still make sense in logical data warehouse design? In *ADBIS*, pages 83–96, 2014.

[32] Selma Bouarar, Ladjel Bellatreche, and Amine Roukh. Eco-data warehouse design through logical variability. In *Proceedings of the 43rd International Conference on Current Trends in Theory and Practice of Computer*

Science, SOFSEM '17, page to appear, 2017.

[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] K. 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 de Poitiers, july 2009.

[35] Ahcène Boukorca, Ladjel Bellatreche, Sid-Ahmed Benali Senouci, and Zoé Faget. Coupling materialized view selection to multi query optimization: Hyper graph approach. *IJDWM*, 11(2):62–84, 2015.

[36] David Broneske, Sebastian Dorok, Veit Köppen, and Andreas Meister. Software design approaches for mastering variability in database systems. In *Proceedings of the 26th GI-Workshop Grundlagen von Datenbanken, Bozen-Bolzano, Italy, October 21st to 24th, 2014.*, pages 47–52, 2014.

[37] Paul G. Brown and Peter J. Hass. Bhunt: automatic discovery of fuzzy algebraic constraints in relational data. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 668–679, 2003.

[38] Nicolas Bruno and Surajit Chaudhuri. To tune or not to tune?: A lightweight physical design alerter. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 499–510, 2006.

[39] Foresman S. Bulos D. Getting started with adapt tm olap database design, white paper. 1998.

[40] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[41] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *13th international conference on World Wide Web (WWW'2004)*, pages 74–83, 2004.

[42] Jose Maria Cavero, Mario Piattini, and Esperanza Marcos. Midea: A multidimensional data warehouse methodology. *ICEIS (1)*, pages 138–144, 2001.

[43] C. Chakroun, L. Bellatreche, and Y. Ait-Ameur. Ontological concepts dependencies driven methodology to design ontology-based databases. *Technical Report*, 2011.

[44] S. Chaudhuri and V. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, pages 3–14. VLDB Endowment, 2007.

[45] Surajit Chaudhuri and Vivek R. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the International Conference on Very Large Databases*, pages 3–14, 2007.

[46] P. Chen. The entity relationship model - towards a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[47] 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.

[48] Xuedong Chen, Patrick O'Neil, and Elizabeth O'Neil. Adjoined dimension column clustering to improve data warehouse query performance. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 1409–1411, 2008.

[49] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2001.

[50] E. F. Codd. A relational model of data for large shared data banks. volume 13, pages 377–387, 1970.

[51] Edgar Frank Codd. Data models in database management. In *ACM SIGMOD Record*, volume 11, pages 112–114. ACM, 1980.

[52] Edgar Frank Codd. Data models in database management. In *ACM SIGMOD Record*, volume 11, pages 112–114. ACM, 1980.

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

[54] EF Codd, SB Codd, and CT Salley. Providing olap (on-line analytical processing). 1993.

[55] Y. Constantinidis. *Expression des besoins pour le système d'information: Guide d'élaboration du cahier des charges - Préface de Michel Volle*. Solutions d'entreprise. Eyrolles, 2011.

[56] Y. Constantinidis. *Expression des besoins pour le SI: Guide d'élaboration du cahier des charges*. Eyrolles, 2015.

[57] 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*, SoCC '10, pages 143–154, 2010.

[58] C. Coronel, S. Morris, and P. Rob. *Database Systems: Design, Implementation, and Management*. Cengage Learning, 10th edition, 2012.

[59] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. *Staged Configuration Using Feature Models*, pages 266–283. 2004.

[60] Frank Dordowsky and Walter Hipp. Adopting software product line principles to manage software variants in a complex avionics system. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 265–274, 2009.

[61] Mohammed El Dammagh and Olga De Troyer. Feature modeling tools: Evaluation and lessons learned. In *Proceedings of the 30th International Conference on Advances in Conceptual Modeling: Recent Developments and New Directions*, ER'11, pages 120–129, 2011.

[62] K. El Gebaly and A. Aboulnaga. Robustness in automatic physical database

design. *In 11th International Conference on Extending Database Technology (EDBT'08)*, pages 145–156, 2008.

[63] Neveen ElGamal, Ali ElBastawissy, and Galal Galal-Edeen. Data warehouse testing. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 1–8, New York, NY, USA, 2013. ACM.

[64] Iman Elghandour, Ashraf Aboulnaga, Daniel C. Zilio, Fei Chiang, Andrey Balmin, Kevin S. Beyer, and Calisto Zuzarte. An xml index advisor for DB2. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1267–1270, 2008.

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

[66] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition, 2010.

[67] M. Fowler. *Analysis Patterns: Reusable Objects Models*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[68] M. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. *In Proceeding of the International Conference on Management of Data, ACM-SIGMOD'96*, pages 149–160, 1996.

[69] C. W. Fung, K. Karlapalem, and Q. Li. Cost-driven evaluation of vertical class partitioning in object oriented databases. *In Fifth International Conference On Database Systems For Advanced Applications (DASFAA'97), Melbourne, Australia*, pages 11–20, April 1997.

[70] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: an extensible data cleaning tool. *ACM SIGMOD Record*, 29(2):590, 2000.

[71] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[72] G. Gardarin. *Bases de données*. Best of Eyrolles. Eyrolles, 2003.

[73] Andreas Geppert, Stefan Scherrer, and Klaus R. Dittrich. Kids: Construction of database management systems based on reuse. Technical report, 1997.

[74] IBM Form Number GH20-1260. The information management system/virtual storage general information. *IBM Syst. J.*, 1978.

[75] Allen Goldberg. Reusing software developments. In *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, SDE 4, pages 107–119, 1990.

[76] M. Golfarelli. Data warehouse lifecycle and design. In *Encyclopedia of Database Systems*, pages 658–664. Springer US, 2009.

[77] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model: a conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(02n03):215–247, 1998.

[78] Matteo Golfarelli and Stefano Rizzi. Data warehouse testing: A prototype-

based methodology. *Information & Software Technology*, 53(11):1183 – 1198, 2011.

[79] 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.

[80] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[81] Katarina Grolinger and Miriam AM C apretz. Autonomic database management: State of the art and future trends. (23), 2012.

[82] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[83] T.R. Gruber. A translation approach to portable ontology specifications. *In Knowledge Acquisition*, 5(2):199–220, 1993.

[84] Himanshu Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.

[85] Peter J Haas, Ihab F Ilyas, Guy M Lohman, and Volker Markl. Discovering and exploiting statistical properties for query optimization in relational databases: A survey. *Statistical Analysis and Data Mining*, 1(4):223–250, 2009.

[86] Jean-Luc Hainaut, Vincent Englebert, Jean Henrard, Jean-Marc Hick, and Didier Roland. Database evolution: the db-main approach. In *Entity-Relationship Approach - ER'94, Business Modelling and Re-Engineering, 13th International Conference on the Entity-Relationship Approach, Manchester, U.K., December 13-16, 1994, Proceedings*, pages 112–131, 1994.

[87] 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.

[88] David C. Hay. *Data Model Patterns: A Metadata Map (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[89] Joseph M. Hellerstein, Michael Stonebraker, and Rick Caccia. Independent, open enterprise data integration. *IEEE Data Engineering Bulletin*, 22(1):43–49, 1999.

[90] Mathieu Humblet, Dang Vinh Tran, Jens H. Weber, and Anthony Cleve. Variability management in database applications. In *Proceedings of the 1st International Workshop on Variability and Complexity in Software Design*, VACE '16, pages 21–27, 2016.

[91] Seung-won Hwang and Kevin Chen-Chuan Chang. Optimizing top-k queries for middleware access: A unified cost-based approach. *ACM Trans. Database Syst.*, 32(1):5, 2007.

[92] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 647–658, 2004.

[93] Yannis E. Ioannidis and Viswanath Poosala. Histogram-based approximation of set-valued query-answers. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 174–185, 1999.

[94] Stéphane Jean, Idir Aït-Sadoune, Ladjel Bellatreche, and Ilyès Boukhari. On using requirements throughout the life cycle of data repository. In *Database and Expert Systems Applications - 25th International Conference, DEXA 2014, Munich, Germany, September 1-4, 2014. Proceedings, Part II*, pages 409–416, 2014.

[95] Stéphane Jean, Ladjel Bellatreche, Carlos Ordonez, Géraud Fokou, and Mickaël Baron. Ontodbench: Interactively benchmarking ontology storage in a database. In *Conceptual Modeling - 32th International Conference, ER 2013, Hong-Kong, China, November 11-13, 2013. Proceedings*, pages 499–503, 2013.

[96] Boukhalfa Kamel. *De la conception physique aux outils d'administration et de tuning des entrepôts de donnés*. PhD thesis, ENSMA et Université de Poitiers, juillet 2009.

[97] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[98] Christian Kästner, Alexander von Rhein, Sebastian Erdweg, Jonas Pusch, Sven Apel, Tillmann Rendel, and Klaus Ostermann. Toward variability-aware testing. In *4th International Workshop on Feature-Oriented Software Development*, pages 1–8, 2012.

[99] Abdelmadjid KETFI. *Generic Approach for the Dynamic Reconfiguration of Component-based Software*. Theses, Université Joseph-Fourier - Grenoble I, December 2004.

[100] Niloofar Khedri and Ramtin Khosravi. Handling database schema variability in software product lines. In *Proceedings of the 2013 20th Asia-Pacific Software Engineering Conference (APSEC) - Volume 01*, APSEC '13, pages 331–338, 2013.

[101] Niloofar Khedri and Ramtin Khosravi. Incremental variability management in conceptual data models of software product lines. In *2015 Asia-Pacific Software Engineering Conference, APSEC 2015, New Delhi, India, December 1-4, 2015*, pages 222–229, 2015.

[102] S. Khouri, L. Bellatreche, I. Boukhari, and S. Bouarar. More investment in conceptual designers: Think about it! In *15th International Conference on Computational Science and Engineering (CSE), IEEE*, pages 88–93, 2012.

[103] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, pages 327–353, 2001.

[104] Ralph Kimball. *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[105] Ralph Kimball, Margy Ross, et al. The data warehouse toolkit: the complete guide to dimensional modelling. *New York ua*, 2002.

[106] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. Zdonik. Coradd: Correlation aware database designer for materialized views and indexes. *PVLDB*, 3(1):1103–1113, 2010.

[107] Hideaki Kimura, George Huo, Alexander Rasin, Samuel Madden, and Stanley B. Zdonik. Correlation maps: A compressed access method for exploiting soft functional dependencies. *Proc. VLDB Endow.*, 2(1):1222–1233, August 2009.

[108] 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.

[109] Eugene Krayzman. Development of self-tuning and autonomic databases and latest achievements, 2005.

[110] Mayuresh Kunjir, Puneet K. Birwa, and Jayant R. Haritsa. Peak power plays in database engines. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 444–455, 2012.

[111] Byung S. Lee. Normalization in oodb design. *SIGMOD Rec.*, 24(3):23–27, September 1995.

[112] Jonathan Lemaitre and Jean-Luc Hainaut. A combined global-analytical quality framework for data models. In Sourrouille Jean-Louis and Staron Miroslaw, editors, *Proceedings of the 3rd International Workshop on Quality in Modeling (QiM'08 @ MODELS'08)*, pages 46–58, 2008.

[113] Mark Levene and George Loizou. Why is the snowflake schema a good data warehouse design? *Information Systems*, 28(3):225–240, 2003.

[114] Liang Liang, Zhiqiang Hu, and Xiangyun Wang. An open architecture for medical image workstation, 2005.

[115] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[116] Liana Barachisio Lisboa, Vinicius Cardoso Garcia, Daniel Lucrédio, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira, and Renata Pontin de Mattos Fortes. A systematic review of domain analysis tools. *Inf. Softw. Technol.*, 52(1):1–13, January 2010.

[117] Hai Liu, Dongqing Xiao, Pankaj Didwania, and Mohamed Y. Eltabakh. Exploiting soft and hard correlations in big data query optimization. *Proc. VLDB Endow.*, 9(12):1005–1016, August 2016.

[118] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[119] Elzbieta Malinowski and Esteban Zimányi. Hierarchies in a multidimensional model: From conceptual modeling to lo-

gical representation. *Data Knowl. Eng.*, 59(2):348–377, 2006.

[120] Imene Mami and Zohra Bellahsene. A survey of view selection methods. *SIGMOD Rec.*, 41(1):20–29, April 2012.

[121] Heikki Mannila and Kari-Jouko Raiha. Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering*, 12(1):83 – 99, 1994.

[122] Tim Martyn. Reconsidering multi-dimensional schemas. *SIGMOD Rec.*, 33(1):83–88, 2004.

[123] M. D. McIlroy. Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*, 1968.

[124] Benjamin J. McMahan, Guoqiang Pan, Patrick Porter, and Moshe Y. Vardi. Projection pushing revisited. In *Advances in Database Technology - EDBT 2004: 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004*, pages 441–458, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[125] Parastoo Mohagheghi and Reidar Conradi. An empirical investigation of software reuse benefits in a large telecom product. *ACM Trans. Softw. Eng. Methodol.*, 17(3):13:1–13:31, June 2008.

[126] H. Molina, W. J. Labio, J. L. Wiener, and Y. Zhuge. Distributed and parallel computing issues in data warehousing. *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, page 7, June 1998.

[127] Marco Mori and Anthony Cleve. Towards highly adaptive data-intensive systems: A research agenda. In *Advanced Information Systems Engineering Workshops - CAiSE 2013 International Workshops, Valencia, Spain, June 17-21, 2013. Proceedings*, pages 386–401, 2013.

[128] Murtaza Motiwala, Amogh Dhamdhere, Nick Feamster, and Anukool Lakhina. Towards a cost model for network traffic. *SIGCOMM Comput. Commun. Rev.*, 42(1):54–60, 2012.

[129] Glenford J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[130] Felix Naumann. Data profiling revisited. *ACM SIGMOD Record*, 42(4):40–49, 2014.

[131] Felix Naumann, Alexander Bilke, Jens Bleiholder, and Melanie Weis. Data fusion in three steps: Resolving schema, tuple, and value inconsistencies. *IEEE Data Eng. Bull*, 29(2):21–31, 2006.

[132] Shamkant B Navathe. Evolution of data modeling for databases. *Communications of the ACM*, 35(9):112–123, 1992.

[133] Rimma Nehme and Nicolas Bruno. Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1137–1148, 2011.

[134] Hoang Vu Nguyen, Emmanuel Müller, Periklis Andritsos, and Klemens Böhm. Detecting correlated columns in relational databases with mixed data types. In *Proceedings of the 26th International Conference on Scientific and Statistical*

Database Management, page 30. ACM, 2014.

[135] Dag Nyström, Aleksandra Tesanovic, Ra Tesanovic, Mikael Nolin, Christer Norström, and Jörgen Hansson. Comet: A component-based real-time database for automotive systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems*, pages 1–8, 2004.

[136] F. Oliveira, K. Nagaraja, R. Bachwani, R. Bianchini, R. P. Martin, and T. D. Nguyen. Understanding and validating database system administration. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 19–19, 2006.

[137] M. Tamer Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[138] M. Tamer Özsu and Benjamin Bin Yao. Building component database systems using CORBA. In *Compontent Database Systems*, pages 207–236. 2001.

[139] D.L. Parnas. On the design and development of program families. *Software Engineering, IEEE Transactions on*, SE-2(1):1–9, March 1976.

[140] Jean-Marc Petit, Farouk Toumani, Jean-Francois Boulicaut, and Jacques Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proceedings of the Twelfth International Conference on Data Engineering*, ICDE '96, pages 218–227, 1996.

[141] Evaggelia Pitoura. Query optimization. In *Encyclopedia of Database Systems*, pages 2272–2273. Springer US, 2009.

[142] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering - Foundations, Principles, and Techniques*. Springer, 2005.

[143] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. *Available at* `http://www.w3.org/TR/rdf-sparql-query/`.

[144] Philippe Pucheral, Luc Bouganim, Patrick Valduriez, and Christophe Bobineau. Picodbms: Scaling down database techniques for the smartcard. *VLDB J.*, 10:120–132, 2001.

[145] Raman01. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[146] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy Lohman. Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 558–569, 2002.

[147] W. Rasdorf, K. Ulberg, and J. Baugh Jr. A structure-based model of semantic integrity constraints for relational data bases. *In Proc. of Engineering with Computers*, 2:31–39, 1987.

[148] Stefano Rizzi, Alberto Abelló, Jens Lechtenbörger, and Juan Trujillo. Research in data warehouse modeling and design: dead or alive? In *Proceedings of the 9th ACM international workshop*

on Data warehousing and OLAP, pages 3–10. ACM, 2006.

[149] Didier Roland, Jean-Luc Hainaut, Jean-Marc Hick, Jean Henrard, and Vincent Englebert. Database engineering processes with DB-MAIN. In *Proceedings of the 8th European Conference on Information Systems, Trends in Information and Communication Systems for the 21st Century, ECIS 2000, Vienna, Austria, July 3-5, 2000*, pages 244–251, 2000.

[150] Daniel J. Rosenkrantz and Harry B. Hunt III. Processing conjunctive predicates and queries. In *Proceedings of the Sixth International Conference on Very Large Data Bases - Volume 6*, VLDB '80, pages 64–72, 1980.

[151] Marko Rosenmüller, Sven Apel, Thomas Leich, and Gunter Saake. Tailor-made data management for embedded systems: A case study on berkeley db. *Data & Knowledge Engineering (DKE)*, 68:1493–1512, 2009.

[152] Marko Rosenmüller, Christian Kästner, Norbert Siegmund, Sagar Sunkle, Sven Apel, Thomas Leich, and Gunter Saake. SQL à la Carte – Toward Tailor-made Data Management. In *13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 117–136, 2009.

[153] Marko Rosenmüller, Norbert Siegmund, Thomas Thüm, and Gunter Saake. Multi-dimensional variability modeling. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 11–20, 2011.

[154] Amine Roukh and Ladjel Bellatreche. Eco-processing of OLAP complex queries. In *Big Data Analytics and Knowledge Discovery - 17th International Conference, DaWaK 2015, Valencia, Spain, September 1-4, 2015, Proceedings*, pages 229–242, 2015.

[155] Amine ROUKH, Ladjel BELLATRECHE, Selma BOUARAR, and Ahcène BOUKORCA. Eco-physic: Eco-physical design initiative for very large databases. *Information Systems Journal*, page To appear, 2017.

[156] 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*, pages 1–10, 2015.

[157] Jean-Claude Royer and Hugo Arboleda. *Model-Driven and Software Product Line Engineering*. Iste Series. Wiley, 2013.

[158] SyBase SAP. Sap sybase powerdesigner. `http://go.sap.com/product/data-mgmt/powerdesigner-data-modeling-tools.html`, 2013.

[159] Carsten Sapia, Markus Blaschka, Gabriele Höfling, and Barbara Dinter. Extending the e/r model for the multidimensional paradigm. *Advances in Database Technologies*, pages 1947–1947, 1999.

[160] Reimar Schröter, Norbert Siegmund, and Thomas Thüm. Towards modular analysis of multi product lines. In *Proceedings of the 17th International Software Product Line Conference Co-*

located Workshops, SPLC '13 Workshops, pages 96–99, 2013.

[161] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, pages 23–34, New York, NY, USA, 1979. ACM.

[162] Thibault Sellam and Martin L. Kersten. Meet charles, big data query advisor. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[163] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, March 1988.

[164] Bouarar Selma, Bellatreche Ladjel, Jean Stéphane, and Baron Mickael. Leveraging ontology-based methodologies for designing semantic data warehouses. In *Proceedings des 29èmes journées Bases de Données Avancées*, oct 2013.

[165] Koltakov Sergey. Oracle: Real-time sql monitoring. White paper, Oracle Corporation, 2009.

[166] Norbert Siegmund, Christian Kästner, Marko Rosenmüller, Florian Heidenreich, Sven Apel, and Gunter Saake. Bridging the gap between variability in client application and db schema. In *Datenbanksysteme in Business, Technologie und Web (BTW 2009), 13. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 2.-6. März 2009, Münster, Germany*, pages 297–306, 2009.

[167] Avi Silberschatz, Mike Stonebraker, and Jeff Ullman. Database research: achievements and opportunities into the 1st century. *ACM sIGMOD record*, 25(1):52–63, 1996.

[168] L. Silverston. *The Data Model Resource Book: A Library of Universal Data Models by Industry Types*. Number vol. 2 in Data modeling and design / Wiley. Wiley, 2001.

[169] L. Silverston. *The Data Model Resource Book: A Library of Universal Data Models for All Enterprises*. Number vol. 1 in Data modeling and design. Wiley, 2001.

[170] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Inf. Softw. Technol.*, 49(7):717–739, July 2007.

[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] Michael Soffner, Norbert Siegmund, Marko Rosenmüller, Janet Siegmund, Thomas Leich, and Gunter Saake. A variability model for query optimizers. In *Databases and Information Systems VII - Selected Papers from the Tenth International Baltic Conference, DB&IS 2012, Vilnius, Lithuania, July 8-11, 2012*, pages 15–28, 2012.

[173] Benkrid Soumia. *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données : application aux plateformes parallèles*. PhD thesis, jun 2014.

[174] Eugenia Stathopoulou and Panos Vassi-

liadis. Design patterns for relational databases. *Citeseer*, pages 1–38, 2009.

[175] Jean Stéphane. *OntoQL, un langage d'exploitation des bases de données à base ontologique*. PhD thesis, Université de Poitiers, dec 2007.

[176] T. Stohr, H. Martens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. *In Proc. of VLDB*, pages 273–284, 2000.

[177] Michael Stonebraker and Ugur Cetintemel. "one size fits all": An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 2–11, 2005.

[178] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35(8):705–754, July 2005.

[179] Robert W. Taylor and Randall L. Frank. Codasyl data-base management systems. *ACM Comput. Surv.*, 8(1):67–103, March 1976.

[180] Computer Associates Technologies. Erwin® data modeler. `http://erwin.com/products/data-modeler`, 2015.

[181] Aleksandra Tesanovic, Ke Sheng, and Jörgen Hansson. Application-tailored database systems: A case of aspects in an embedded database. In *8th International Database Engineering and Applications Symposium (IDEAS 2004), 7-9 July 2004, Coimbra, Portugal*, pages 291–301, 2004.

[182] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Sci. Comput. Program.*, 79:70–85, January 2014.

[183] Kalle Tomingas, Margus Kliimask, and Tanel Tammet. Data integration patterns for data warehouse automation. pages 41–55, 2014.

[184] Albert Tort, Antoni Olivé, and Maria-Ribera Sancho. An approach to test-driven development of conceptual schemas. *Data & Knowledge Engineering (DKE)*, 70(12):1088 – 1111, 2011.

[185] D-N Tran, P. C. Huynh, Y. C. Tay, and Anthony K. H. Tung. A new approach to dynamic self-tuning of database buffers. *Trans. Storage*, 4:3:1–3:25, 2008.

[186] 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.

[187] Vasiliki Tziovara, Panos Vassiliadis, and Alkis Simitsis. Deciding the physical implementation of etl workflows. In *Proceedings of the ACM Tenth International Workshop on Data Warehousing and OLAP*, DOLAP '07, pages 49–56, 2007.

[188] Sriramkumar Venugopalan, Vivek Joshi, Luis Zamudio, Matthias Goldbach, Gert Burbach, Ralf Van Bentum, and Sriram Balasubramanian. Sram read current variability and its dependence on transistor statistics. In *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference (CICC)*, pages 1–4, 2013.

[189] Karina Villela, Adeline Silva, Tassio Vale, and Eduardo Santana de Almeida.

A survey on software variability management approaches. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 147–156, 2014.

[190] Hannes Voigt, Alfred Hanisch, and Wolfgang Lehner. Flexs - A logical model for physical data layout. In *New Trends in Database and Information Systems II - Selected papers of the 18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events, ADBIS 2014 Ohrid, Macedonia, September 7-10, 2014 Proceedings II*, pages 85–95, 2014.

[191] K-Y. Whang, G. Wiederhold, and D. Sagalowicz. Estimating block accesses in database organizations: a closed noniterative formula. *Commun. ACM*, 26:940–944, 1983.

[192] Yuanyuan Xu and Bruno Traverson. Procédés de réutilisation pour les lignes de produits logiciels. In *Actes du XXVIème Congrès INFORSID, Fontainebleau, France, 27 au 30 mai 2008*, 2008.

[193] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Power modeling in database management systems. Technical report, University of South Florida, Department of Computer Science and Engineering, 2012.

[194] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Dynamic energy estimation of query plans in database systems. In *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, 8-11 July, 2013, Philadelphia, Pennsylvania, USA*, pages 83–92, 2013.

[195] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the International Conference on Very Large Databases*, pages 136–145. Morgan Kaufmann Publishers Inc., 1997.

[196] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 136–145, 1997.

[197] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1:32–49, 2011.

[198] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. Db2 design advisor: Integrated automatic physical database design. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 1087–1097, 2004.

[199] Tao Zou, Ronan Le Bras, Marcos Antonio Vaz Salles, Alan J. Demers, and Johannes Gehrke. Cloudia: a deployment advisor for public clouds. *VLDB J.*, 24(5):633–653, 2015.

Annexe A

Schémas logiques générés

Le déploiement des 256 schémas n'impliquent au final que : la table de faits d'origine, les quatre tables de dimensions d'origine et 35 nouvelles tables représentant les dimensions éclatées et les sous-dimensions créées. Cela fait un total de 40 tables partagées à déployer sur le SGBD, dont les différentes combinaisons constituent les 256 schémas. Rappelons que la charge des requêtes est réécrite selon chaque schéma logique. Ce sont ces requêtes qui définissent et distinguent les schémas déployés entre eux. Ci-dessous la charge relative à la création de l'ensemble de schémas logiques (39 requêtes, celle liée à la table de faits d'origine n'y figure pas, car cette table ne change pas).

```
1 CREATE TABLE "C_CITYDimNATION3"(H_C_NATIONID_FK INTEGER,
     H_C_CITYID INTEGER, H_C_CITY VARCHAR(10) );
2 CREATE TABLE "C_CITYDimREGION3"(H_C_REGIONID_FK INTEGER,
     H_C_CITYID INTEGER, H_C_CITY VARCHAR(10) );
3 CREATE TABLE "C_NATIONDimREGION3"(H_C_REGIONID_FK INTEGER,
     H_C_NATIONID INTEGER, H_C_NATION VARCHAR(15) );
4 CREATE TABLE "C_NATIONDimREGION4"(H_C_REGIONID_FK INTEGER,
     H_C_NATIONID INTEGER, H_C_NATION VARCHAR(15), H_C_CITY
     VARCHAR(10) );
5 CREATE TABLE "C_REGIONDim2"(H_C_REGIONID INTEGER, H_C_REGION
     VARCHAR(12) );
6 CREATE TABLE "C_REGIONDim3"(H_C_REGIONID INTEGER, H_C_REGION
     VARCHAR(12),H_C_NATION VARCHAR(15) );
7 CREATE TABLE "DIM_CUSTOMER"(C_CUSTKEY INTEGER,C_NAME VARCHAR(25)
     ,C_ADDRESS VARCHAR(40),H_C_CITY VARCHAR(10),H_C_NATION
     VARCHAR(15),H_C_REGION VARCHAR(12),C_PHONE VARCHAR(15),
     C_MKTSEGMENT VARCHAR(20) );
8 CREATE TABLE "DIM_CUSTOMER-CITY"(C_CUSTKEY INTEGER,C_NAME
     VARCHAR(25),C_ADDRESS VARCHAR(40),C_PHONE VARCHAR(15),
     C_MKTSEGMENT VARCHAR(20),H_C_CITYID_FK INTEGER);
```

```
9  CREATE TABLE "DIM_CUSTOMER-NATION"(C_CUSTKEY INTEGER,C_NAME
       VARCHAR(25),C_ADDRESS VARCHAR(40),C_PHONE VARCHAR(15),
       C_MKTSEGMENT VARCHAR(20),H_C_NATIONID_FK INTEGER);
10 CREATE TABLE "DIM_DATES"(D_WEEKDAYFL INTEGER,D_HOLIDAYFL INTEGER
       ,D_LASTDAYINMONTHFL INTEGER,D_LASTDAYINWEEKFL INTEGER,
       H_D_SEASON VARCHAR(12),D_WEEKNUMINYEAR INTEGER,
       D_MONTHNUMINYEAR INTEGER,D_DAYNUMINYEAR INTEGER,
       D_DAYNUMINMONTH INTEGER,D_DAYNUMINWEEK INTEGER,D_YEARMONTH
       VARCHAR(7),D_YEARMONTHNUM INTEGER,H_D_YEAR INTEGER,H_D_MONTH
       VARCHAR(9),H_D_DAY VARCHAR(10),D_DATE VARCHAR(18),D_DATEKEY
       INTEGER);
11 CREATE TABLE "DIM_DATES-DAY"(D_WEEKDAYFL INTEGER,D_HOLIDAYFL
       INTEGER,D_LASTDAYINMONTHFL INTEGER,D_LASTDAYINWEEKFL INTEGER,
       D_WEEKNUMINYEAR INTEGER,D_MONTHNUMINYEAR INTEGER,
       D_DAYNUMINYEAR INTEGER,D_DAYNUMINMONTH INTEGER,D_DAYNUMINWEEK
        INTEGER,D_YEARMONTH VARCHAR(7) ,D_YEARMONTHNUM INTEGER,
       D_DATE VARCHAR(18),D_DATEKEY INTEGER,H_D_DAYID_FK INTEGER);
12 CREATE TABLE "DIM_DATES-SEASON"(D_WEEKDAYFL INTEGER,D_HOLIDAYFL
       INTEGER,D_LASTDAYINMONTHFL INTEGER,D_LASTDAYINWEEKFL INTEGER,
       D_WEEKNUMINYEAR INTEGER,D_MONTHNUMINYEAR INTEGER,
       D_DAYNUMINYEAR INTEGER,D_DAYNUMINMONTH INTEGER,D_DAYNUMINWEEK
        INTEGER,D_YEARMONTH VARCHAR(7),D_YEARMONTHNUM INTEGER,D_DATE
        VARCHAR(18),D_DATEKEY INTEGER,H_D_SEASONID_FK INTEGER);
13 CREATE TABLE "DIM_DATES-YEAR"(D_WEEKDAYFL INTEGER,D_HOLIDAYFL
       INTEGER,D_LASTDAYINMONTHFL INTEGER,D_LASTDAYINWEEKFL INTEGER,
       D_WEEKNUMINYEAR INTEGER,D_MONTHNUMINYEAR INTEGER,
       D_DAYNUMINYEAR INTEGER,D_DAYNUMINMONTH INTEGER,D_DAYNUMINWEEK
        INTEGER,D_YEARMONTH VARCHAR(7) ,D_YEARMONTHNUM INTEGER,
       D_DATE VARCHAR(18) ,D_DATEKEY INTEGER,H_D_YEARID_FK INTEGER);
14 CREATE TABLE "DIM_PART"(P_PARTKEY INTEGER,P_NAME VARCHAR(22),
       P_MFGR VARCHAR(6),H_P_CATEGORY VARCHAR(7),H_P_BRAND VARCHAR
       (9),P_COLOR VARCHAR(11),P_TYPE VARCHAR(25),P_SIZE INTEGER,
       P_CONTAINER VARCHAR(20) );
15 CREATE TABLE "DIM_PART-BRAND"(P_PARTKEY INTEGER,P_NAME VARCHAR
       (22),P_MFGR VARCHAR(6),P_COLOR VARCHAR(11),P_TYPE VARCHAR(25)
       ,P_SIZE INTEGER,P_CONTAINER VARCHAR(20),H_P_BRANDID_FK
       INTEGER);
16 CREATE TABLE "DIM_SUPPLIER"(S_SUPPKEY INTEGER,S_NAME VARCHAR(25)
       ,S_ADDRESS VARCHAR(25),H_S_CITY VARCHAR(10),H_S_NATION
       VARCHAR(15),H_S_REGION VARCHAR(12),S_PHONE VARCHAR(30));
```

```
17 CREATE TABLE "DIM_SUPPLIER-CITY"(S_SUPPKEY INTEGER,S_NAME
     VARCHAR(25),S_ADDRESS VARCHAR(25),S_PHONE VARCHAR(30),
     H_S_CITYID_FK INTEGER);
18 CREATE TABLE "DIM_SUPPLIER-NATION"(S_SUPPKEY INTEGER,S_NAME
     VARCHAR(25),S_ADDRESS VARCHAR(25),S_PHONE VARCHAR(30),
     H_S_NATIONID_FK INTEGER);
19 CREATE TABLE "D_DAYDimMONTH3"(H_D_MONTHID_FK INTEGER,H_D_DAYID
     INTEGER,H_D_DAY VARCHAR(10));
20 CREATE TABLE "D_DAYDimMONTH4"(H_D_MONTHID_FK INTEGER,H_D_DAYID
     INTEGER,H_D_DAY VARCHAR(10),H_D_YEAR INTEGER);
21 CREATE TABLE "D_DAYDimMONTH5"(H_D_MONTHID_FK INTEGER,H_D_DAYID
     INTEGER,H_D_DAY VARCHAR(10),H_D_YEAR INTEGER,H_D_SEASON
     VARCHAR(12));
22 CREATE TABLE "D_MONTHDim2"(H_D_MONTHID INTEGER,H_D_MONTH VARCHAR
     (9));
23 CREATE TABLE "D_MONTHDim3"(H_D_MONTHID INTEGER,H_D_MONTH VARCHAR
     (9),H_D_DAY VARCHAR(10));
24 CREATE TABLE "D_MONTHDim4"(H_D_MONTHID INTEGER,H_D_MONTH VARCHAR
     (9),H_D_DAY VARCHAR(10),H_D_YEAR INTEGER);
25 CREATE TABLE "D_SEASONDimDAY3"(H_D_DAYID_FK INTEGER,H_D_SEASONID
      INTEGER,H_D_SEASON VARCHAR(12));
26 CREATE TABLE "D_SEASONDimMONTH3"(H_D_MONTHID_FK INTEGER,
     H_D_SEASONID INTEGER,H_D_SEASON VARCHAR(12));
27 CREATE TABLE "D_SEASONDimYEAR3"(H_D_YEARID_FK INTEGER,
     H_D_SEASONID INTEGER,H_D_SEASON VARCHAR(12));
28 CREATE TABLE "D_YEARDimDAY3"(H_D_DAYID_FK INTEGER,H_D_YEARID
     INTEGER,H_D_YEAR INTEGER);
29 CREATE TABLE "D_YEARDimDAY4"(H_D_DAYID_FK INTEGER,H_D_YEARID
     INTEGER,H_D_YEAR INTEGER,H_D_SEASON VARCHAR(12));
30 CREATE TABLE "D_YEARDimMONTH3"(H_D_MONTHID_FK INTEGER,H_D_YEARID
      INTEGER,H_D_YEAR INTEGER);
31 CREATE TABLE "D_YEARDimMONTH4"(H_D_MONTHID_FK INTEGER,H_D_YEARID
      INTEGER,H_D_YEAR INTEGER,H_D_SEASON VARCHAR(12));
32 CREATE TABLE "P_BRANDDimCATEGORY3"(H_P_CATEGORYID_FK INTEGER,
     H_P_BRANDID INTEGER,H_P_BRAND VARCHAR(9));
33 CREATE TABLE "P_CATEGORYDim2"(H_P_CATEGORYID INTEGER,
     H_P_CATEGORY VARCHAR(7));
34 CREATE TABLE "S_CITYDimNATION3"(H_S_NATIONID_FK INTEGER,
     H_S_CITYID INTEGER,H_S_CITY VARCHAR(10));
35 CREATE TABLE "S_CITYDimREGION3"(H_S_REGIONID_FK INTEGER,
```

```
    H_S_CITYID INTEGER,H_S_CITY VARCHAR(10));
36 CREATE TABLE "S_NATIONDimREGION3"(H_S_REGIONID_FK INTEGER,
    H_S_NATIONID INTEGER,H_S_NATION VARCHAR(15));
37 CREATE TABLE "S_NATIONDimREGION4"(H_S_REGIONID_FK INTEGER,
    H_S_NATIONID INTEGER,H_S_NATION VARCHAR(15),H_S_CITY VARCHAR
    (10));
38 CREATE TABLE "S_REGIONDim2"(H_S_REGIONID INTEGER,H_S_REGION
    VARCHAR(12));
39 CREATE TABLE "S_REGIONDim3"(H_S_REGIONID INTEGER,H_S_REGION
    VARCHAR(12),H_S_NATION VARCHAR(15));
```

Annexe B

Requêtes utilisées

Dans cette annexe, nous présentons les différentes charges de requêtes que nous avons manipulées et créées. Chacune est destinée à une problématique précise (variabilité). Nous distinguons principalement 4 charges :

1. La charge originale de SSB, utilisée pour l'évaluation de l'impact de la variabilité sur l'optimisation logique, en fonction du temps d'exécution/taille de stockage de données (voir sous section 4.4.2 et 4.4.4).
2. Une charge que nous avons créée à partir de la charge initiale de *SSB*, pour l'évaluation de l'impact de la variabilité sur l'optimisation logique en fonction de l'énergie consommée. Cette charge contient des requêtes qui épuisent le processeur, celles qui sont gourmandes en opérations d'entrées/sorties et des requêtes hybrides (voir sous section 4.4.3).
3. Une charge que nous avons créée à partir de la charge initiale de *SSB*, pour l'évaluation de l'impact de la variabilité sur l'optimisation physique en fonction du temps/énergie. Cette charge représente des requêtes exécutées simultanément de façon concurrente pour l'OMR (voir section 5.5).
4. Enfin, un aperçu de la charge utilisée dans la phase d'apprentissage du développement de notre MdC orienté consommation d'énergie. Celle-ci est basé sur la charge du benchmark TPC-H (voir sous-section 4.4.3.2).

Ci-dessous la liste des requêtes de chaque type, dans le même ordre que la liste précédente

1 Évaluation du temps d'exécution

```
1
2 select /*Q1.1*/ sum(lo_extendedprice*lo_discount) as revenue
    from lineorder, date where lo_orderdate = d_datekey and
    d_year = 1993 and lo_discount between1 and 3 and lo_quantity
    < 25;
```

```
3 select /*Q1.2*/ sum(lo_extendedprice*lo_discount) as revenue
    from lineorder, date where lo_orderdate = d_datekey and
    d_yearmonthnum = 199401 and lo_discount between 4 and 6 and
    lo_quantity between 26 and 35;
4 select /*Q1.3*/ sum(lo_extendedprice*lo_discount) as revenue
    from lineorder, date where lo_orderdate = d_datekey and
    d_weeknuminyear = 6 and d_year = 1994 and lo_discount between
     5 and 7 and lo_quantity between 26 and 35; -- non considérée
5 select /*Q2.1*/ sum(lo_revenue), d_year, p_brand1 from lineorder
    , date, part, supplier where lo_orderdate = d_datekey and
    lo_partkey = p_partkey and lo_suppkey = s_suppkey and
    p_category = 'MFGR#12' and s_region = 'AMERICA' group by
    d_year, p_brand1 order by d_year, p_brand1;
6 select /*Q2.2*/ sum(lo_revenue), d_year, p_brand from lineorder,
     dates, part, supplier where lo_orderdate = d_datekey and
    lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand
     between 'MFGR#2221' and 'MFGR#2228' and s_region = 'ASIA'
    group by d_year, p_brand order by d_year, p_brand;
7 select /*Q2.3*/ sum(lo_revenue), d_year, p_brand1 from lineorder
    , date, part, supplier where lo_orderdate = d_datekey and
    lo_partkey = p_partkey and lo_suppkey = s_suppkey and
    p_brand1 = 'MFGR#2221' and s_region = 'EUROPE' group by
    d_year, p_brand1 order by d_year, p_brand1;
8 select /*Q3.1*/ c_nation, s_nation, d_year, sum(lo_revenue) as
    revenue from customer, lineorder, supplier, date where
    lo_custkey = c_custkey and lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and c_region = 'ASIA' and s_region =
     'ASIA' and d_year >= 1992 and d_year <= 1997 group by
    c_nation, s_nation, d_year order by d_year asc, revenue desc;
9 select /*Q3.2*/ c_city, s_city, d_year, sum(lo_revenue) as
    revenue from customer, lineorder, supplier, date where
    lo_custkey = c_custkey and lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and c_nation = 'UNITED␣STATES' and
    s_nation = 'UNITED␣STATES' and d_year >= 1992 and d_year <=
    1997 group by c_city, s_city, d_year order by d_year asc,
    revenue desc;
10 select /*Q3.3*/ c_city, s_city, d_year, sum(lo_revenue) as
    revenue from customer, lineorder, supplier, date where
    lo_custkey = c_custkey and lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and (c_city='UNITED␣KI1' or c_city='
```

```
   UNITED␣KI5') and (s_city='UNITED␣KI1' or s_city='UNITED␣KI5')
    and d_year >= 1992 and d_year <= 1997 group by c_city,
   s_city, d_year order by d_year asc, revenue desc;
11 select /*Q3.4*/ c_city, s_city, d_year, sum(lo_revenue) as
   revenue from customer, lineorder, supplier, date where
   lo_custkey = c_custkey and lo_suppkey = s_suppkey and
   lo_orderdate = d_datekey and (c_city='UNITED␣KI1' or c_city='
   UNITED␣KI5') and (s_city='UNITED␣KI1' or s_city = 'UNITED␣KI5
   ') and d_yearmonth = 'Dec1997' group by c_city, s_city,
   d_year order by d_year asc, revenue desc;
12 select /*Q4.1*/ d_year, c_nation, sum(lo_revenue - lo_supplycost
   ) as profit
13 from date, customer, supplier, part, lineorder where lo_custkey
   = c_custkey and lo_suppkey = s_suppkey and lo_partkey =
   p_partkey and lo_orderdate = d_datekey and c_region = '
   AMERICA' and s_region = 'AMERICA' and (p_mfgr = 'MFGR#1' or
   p_mfgr = 'MFGR#2') group by d_year, c_nation order by d_year,
    c_nation
14 select /*Q4.2*/ d_year, s_nation, p_category, sum(lo_revenue -
   lo_supplycost) as profit from date, customer, supplier, part,
    lineorder where lo_custkey = c_custkey and lo_suppkey =
   s_suppkey and lo_partkey = p_partkey and lo_orderdate =
   d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA'
   and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1'
   or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category
   order by d_year, s_nation, p_category
15 select /*Q4.2*/ d_year, s_city, p_brand1, sum(lo_revenue -
   lo_supplycost) as profit from date, customer, supplier, part,
    lineorder where lo_custkey = c_custkey and lo_suppkey =
   s_suppkey and lo_partkey = p_partkey and lo_orderdate =
   d_datekey and c_region = 'AMERICA' and s_nation = 'UNITED␣
   STATES' and (d_year = 1997 or d_year = 1998) and p_category =
    'MFGR#14' group by d_year, s_city, p_brand1 order by d_year,
    s_city, p_brand1
```

2 Évaluation de l'énergie consommée

```
1 select /* q0 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(LO_QUANTITY)
   , avg(LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum
```

```
    (LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum
    (LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY
    * 0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder, dates where
    lo_orderdate = d_datekey and d_year = 1998 and lo_discount
    >= 1 and lo_discount <= 3 and lo_quantity < 25 group by
    LO_SHIPMODE, LO_ORDERPRIORITY order by LO_SHIPMODE,
    LO_ORDERPRIORITY;
2 select /* q1 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(lo_quantity)
   as sum_qty, sum(lo_extendedprice) as sum_base_price, sum(
   lo_extendedprice * (1 - lo_discount)) as sum_disc_price, sum(
   lo_extendedprice * (1 - lo_discount) * (1 + lo_tax)) as
   sum_charge, avg(lo_quantity) as avg_qty, avg(lo_extendedprice
   ) as avg_price, avg(lo_discount) as avg_disc, count(*) as
   count_order from lineorder, dates where lo_orderdate =
   d_datekey and d_year = 1998 and lo_discount >= 1 and
   lo_discount <= 3 and lo_quantity < 25 group by LO_SHIPMODE,
   LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
3 select /* q2 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(LO_QUANTITY),
    avg(LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
   LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
   LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
   LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
   0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
   LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
   LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
   LO_ORDTOTALPRICE)+1), count(*) from lineorder, dates where
   lo_orderdate = d_datekey and d_year = 1998 group by
   LO_SHIPMODE, LO_ORDERPRIORITY order by LO_SHIPMODE,
   LO_ORDERPRIORITY;
4 select /* q3 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(lo_quantity)
   as sum_qty, sum(lo_extendedprice) as sum_base_price, sum(
   lo_extendedprice * (1 - lo_discount)) as sum_disc_price, sum(
   lo_extendedprice * (1 - lo_discount) * (1 + lo_tax)) as
   sum_charge, avg(lo_quantity) as avg_qty, avg(lo_extendedprice
   ) as avg_price, avg(lo_discount) as avg_disc, count(*) as
   count_order from lineorder, dates where lo_orderdate =
```

```
    d_datekey and d_year = 1998 group by LO_SHIPMODE,
    LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
5 select /* q4 */ sum(lo_quantity) as sum_qty, sum(
    lo_extendedprice) as sum_base_price, sum(lo_extendedprice *
    (1 - lo_discount)) as sum_disc_price, sum(lo_extendedprice *
    (1 - lo_discount) * (1 + lo_tax)) as sum_charge, avg(
    lo_quantity) as avg_qty, avg(lo_extendedprice) as avg_price,
    avg(lo_discount) as avg_disc, count(*) as count_order, d_year
     from lineorder, dates, part, supplier where lo_orderdate =
    d_datekey and lo_partkey = p_partkey and lo_suppkey =
    s_suppkey and p_brand = 'MFGR#3438' and s_region = 'ASIA'
    group by d_year order by d_year;
6 select /* q5 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(LO_QUANTITY),
     avg(LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
    LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
    0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder, part where
    lo_partkey = p_partkey and p_brand = 'MFGR#3438' group by
    LO_SHIPMODE, LO_ORDERPRIORITY order by LO_SHIPMODE,
    LO_ORDERPRIORITY;
7 select /* q6 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(LO_QUANTITY),
     avg(LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
    LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
    0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder, dates, part,
    supplier where lo_orderdate = d_datekey and lo_partkey =
    p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#3438
    ' and s_region = 'ASIA' group by LO_SHIPMODE,
    LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
8 select /* q7 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(LO_QUANTITY),
     avg(LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
```

```
   LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
   LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
   0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
   LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
   LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
   LO_ORDTOTALPRICE)+1), count(*) from lineorder, dates, part,
   supplier where lo_orderdate = d_datekey and lo_partkey =
   p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#5227
   ' and s_region = 'AMERICA' group by LO_SHIPMODE,
   LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
9  select /* q8 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(lo_quantity)
   as sum_qty, sum(lo_extendedprice) as sum_base_price, sum(
   lo_extendedprice * (1 - lo_discount)) as sum_disc_price, sum(
   lo_extendedprice * (1 - lo_discount) * (1 + lo_tax)) as
   sum_charge, avg(lo_quantity) as avg_qty, avg(lo_extendedprice
   ) as avg_price, avg(lo_discount) as avg_disc, count(*) as
   count_order from lineorder, part, supplier where lo_partkey =
    p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR
   #5227' and s_region = 'AMERICA' group by LO_SHIPMODE,
   LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
10 select /* q9 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(LO_QUANTITY),
    avg(LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
   LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
   LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
   LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
   0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
   LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
   LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
   LO_ORDTOTALPRICE)+1), count(*) from lineorder, dates, part,
   supplier where lo_orderdate = d_datekey and lo_partkey =
   p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#5227
   ' and s_region = 'AMERICA' group by LO_SHIPMODE,
   LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
11 select /* q10 */ LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION,
   C_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
```

```
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder,customer, supplier, dates where lo_custkey =
    c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
    d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and
    d_year >= 1992 and d_year <= 1997 group by LO_SHIPMODE,
    LO_ORDERPRIORITY, C_NATION, C_REGION order by LO_SHIPMODE,
    LO_ORDERPRIORITY, C_NATION, C_REGION;
12  select /* q11 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
     - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder, supplier where lo_suppkey = s_suppkey and s_region
     = 'AMERICA' group by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION
    , S_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION;
13  select /* q12 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
     - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder, supplier, dates where lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and s_region = 'AFRICA' group by
    LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION, S_REGION order by
    LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION, S_REGION;
14  select /* q13 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
     - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
```

```
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder, supplier, dates where lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and s_region = 'AFRICA' and d_year
    >= 1992 and d_year <= 1997 group by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION order by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION;
15  select /* q14 */ LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION,
    C_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
     - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder,customer, supplier, dates where lo_custkey =
    c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
    d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA'
    and d_year >= 1992 and d_year <= 1997 group by LO_SHIPMODE,
    LO_ORDERPRIORITY, C_NATION, C_REGION order by LO_SHIPMODE,
    LO_ORDERPRIORITY, C_NATION, C_REGION;
16  select /* q15 */ LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION,
    C_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
     - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder,customer, supplier, dates where lo_custkey =
    c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
    d_datekey and c_nation = 'INDONESIA' and s_nation = '
    INDONESIA' and d_year >= 1992 and d_year <= 1997 group by
    LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION, C_REGION order by
    LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION, C_REGION;
17  select /* q16 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
```

```
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder,customer, supplier, dates where lo_custkey =
    c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
    d_datekey and c_nation = 'ALGERIA' and s_nation = 'ALGERIA'
    and d_year >= 1992 and d_year <= 1997 group by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION order by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION;
18  select /* q17 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder, supplier where lo_suppkey = s_suppkey and s_nation
    = 'ALGERIA' group by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION
    , S_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION;
19  select /* q18 */ LO_SHIPMODE, LO_ORDERPRIORITY, sum(lo_quantity)
    as sum_qty, sum(lo_extendedprice) as sum_base_price, sum(
    lo_extendedprice * (1 - lo_discount)) as sum_disc_price, sum(
    lo_extendedprice * (1 - lo_discount) * (1 + lo_tax)) as
    sum_charge, avg(lo_quantity) as avg_qty, avg(lo_extendedprice
    ) as avg_price, avg(lo_discount) as avg_disc, count(*) as
    count_order from lineorder, supplier where lo_suppkey =
    s_suppkey and s_nation = 'ALGERIA' group by LO_SHIPMODE,
    LO_ORDERPRIORITY order by LO_SHIPMODE, LO_ORDERPRIORITY;
20  select /* q19 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, C_NATION, C_REGION, sum(LO_QUANTITY), avg(
    LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
```

```
    LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
    0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder,customer,
    supplier where lo_custkey = c_custkey and lo_suppkey =
    s_suppkey and c_nation = 'ALGERIA' and s_nation = 'ALGERIA'
    group by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION, S_REGION,
    C_NATION, C_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY,
    S_NATION, S_REGION, C_NATION, C_REGION;
21  select /* q20 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, C_NATION, C_REGION, sum(LO_QUANTITY), avg(
    LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
    LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
    0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder,customer,
    supplier, dates where lo_custkey = c_custkey and lo_suppkey =
     s_suppkey and lo_orderdate = d_datekey and c_nation = '
    ALGERIA' and s_nation = 'ALGERIA' and d_year >= 1992 and
    d_year <= 1997 group by LO_SHIPMODE, LO_ORDERPRIORITY,
    S_NATION, S_REGION, C_NATION, C_REGION order by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION, C_NATION, C_REGION;
22  select /* q21 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, C_NATION, C_REGION, sum(LO_QUANTITY), avg(
    LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
    LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
    0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder,customer,
    supplier, dates where lo_custkey = c_custkey and lo_suppkey =
     s_suppkey and lo_orderdate = d_datekey and c_nation = '
    INDONESIA' and s_nation = 'INDONESIA' and d_year >= 1992 and
    d_year <= 1997 group by LO_SHIPMODE, LO_ORDERPRIORITY,
```

```
   S_NATION, S_REGION, C_NATION, C_REGION order by LO_SHIPMODE,
   LO_ORDERPRIORITY, S_NATION, S_REGION, C_NATION, C_REGION;
23 select /* q22 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
   S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
   avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
   LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
   lineorder,dates, customer, supplier, part where lo_custkey =
   c_custkey and lo_suppkey = s_suppkey and lo_partkey =
   p_partkey and lo_orderdate = d_datekey and c_region = 'AFRICA
   ' and s_region = 'AFRICA' and (d_year = 1998 or d_year =
   1997) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#3') group by
   LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION, S_REGION order by
   LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION, S_REGION;
24 select /* q23 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
   S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
   avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
   LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
   lineorder, dates, supplier, part where lo_suppkey = s_suppkey
    and lo_partkey = p_partkey and lo_orderdate = d_datekey and
   s_region = 'ASIA' and (d_year = 1996 or d_year = 1997) and (
   p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#3') group by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION order by LO_SHIPMODE,
   LO_ORDERPRIORITY, S_NATION, S_REGION;
25 select /* q24 */ LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION,
   C_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
```

```
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder, dates, customer, supplier, part where lo_custkey =
     c_custkey and lo_suppkey = s_suppkey and lo_partkey =
    p_partkey and lo_orderdate = d_datekey and c_region = 'AFRICA
    ' and s_region = 'AFRICA' and (d_year = 1997 or d_year =
    1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#3') group by
    LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION, C_REGION order by
    LO_SHIPMODE, LO_ORDERPRIORITY, C_NATION, C_REGION;
26  select /* q25 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, C_NATION, C_REGION, sum(LO_QUANTITY), avg(
    LO_QUANTITY), sum(LO_QUANTITY - LO_EXTENDEDPRICE), sum(
    LO_QUANTITY * LO_EXTENDEDPRICE - 1), avg(LO_DISCOUNT * (
    LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX +1)), sum(LO_TAX)*sum(
    LO_EXTENDEDPRICE), sum(LO_QUANTITY * 0.5), avg(LO_QUANTITY *
    0.2), sum(LO_ORDTOTALPRICE), avg(LO_SHIPPRIORITY), sum(
    LO_ORDTOTALPRICE - LO_EXTENDEDPRICE), avg(LO_SHIPPRIORITY *
    LO_EXTENDEDPRICE - 1), sum(LO_SHIPPRIORITY)/(sum(
    LO_ORDTOTALPRICE)+1), count(*) from lineorder, dates,
    customer, supplier where lo_custkey = c_custkey and
    lo_suppkey = s_suppkey and lo_orderdate = d_datekey and
    c_region <> 'AFRICA' and s_region <> 'AFRICA' and (d_year <>
    1998) group by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, C_NATION, C_REGION order by LO_SHIPMODE,
    LO_ORDERPRIORITY, S_NATION, S_REGION, C_NATION, C_REGION;
27  select /* q26 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
    S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
     - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
    , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
    +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
    0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
    LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
    avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
    LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
    lineorder,dates, supplier, customer where lo_suppkey =
    s_suppkey and lo_custkey = c_custkey and lo_orderdate =
    d_datekey and s_region <> 'AFRICA' and c_region <> 'AFRICA'
    and (d_year <> 1998) group by LO_SHIPMODE, LO_ORDERPRIORITY,
    S_NATION, S_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY,
    S_NATION, S_REGION;
```

```
28 select /* q27 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
   S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
   avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
   LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
   lineorder,dates, supplier where lo_suppkey = s_suppkey and
   lo_orderdate = d_datekey and s_region <> 'AMERICA' and (
   d_year <> 1998) group by LO_SHIPMODE, LO_ORDERPRIORITY,
   S_NATION, S_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY,
   S_NATION, S_REGION;
29 select /* q28 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
   S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
   avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
   LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
   lineorder,dates, supplier where lo_suppkey = s_suppkey and
   lo_orderdate = d_datekey and s_region <> 'AMERICA' and (
   d_year <> 1998) group by LO_SHIPMODE, LO_ORDERPRIORITY,
   S_NATION, S_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY,
   S_NATION, S_REGION;
30 select /* q29 */ LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
   S_REGION, sum(LO_QUANTITY), avg(LO_QUANTITY), sum(LO_QUANTITY
    - LO_EXTENDEDPRICE), sum(LO_QUANTITY * LO_EXTENDEDPRICE - 1)
   , avg(LO_DISCOUNT * (LO_TAX +1)), avg(LO_QUANTITY / (LO_TAX
   +1)), sum(LO_TAX)*sum(LO_EXTENDEDPRICE), sum(LO_QUANTITY *
   0.5), avg(LO_QUANTITY * 0.2), sum(LO_ORDTOTALPRICE), avg(
   LO_SHIPPRIORITY), sum(LO_ORDTOTALPRICE - LO_EXTENDEDPRICE),
   avg(LO_SHIPPRIORITY * LO_EXTENDEDPRICE - 1), sum(
   LO_SHIPPRIORITY)/(sum(LO_ORDTOTALPRICE)+1), count(*) from
   lineorder,dates, supplier where lo_suppkey = s_suppkey and
   lo_orderdate = d_datekey and s_region <> 'AFRICA' and (d_year
    <> 1998) group by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
```

```
   S_REGION order by LO_SHIPMODE, LO_ORDERPRIORITY, S_NATION,
   S_REGION;
```

3 Évaluation de l'optimisation multi-requêtes

```
1  select sum(lo_extendedprice*lo_discount) as revenue from
      lineorder, dates where lo_orderdate = d_datekey and d_year =
      1993 and lo_discount >= 1 and lo_discount <= 3 and
      lo_quantity < 25
2  select count(*) from lineorder, dates where lo_orderdate =
      d_datekey and d_year = 1993 and lo_discount >= 1 and
      lo_discount <= 3 and lo_quantity < 25
3  select sum(lo_extendedprice*lo_discount) as revenue from
      lineorder, dates where lo_orderdate = d_datekey and d_year =
      1993
4  select count(*) from lineorder, dates where lo_orderdate =
      d_datekey and d_year = 1993
5  select sum(lo_revenue), d_year from lineorder, dates, part,
      supplier, nations,regions where s_nationkey = f_n_nationkey
      and f_n_regionkey = f_r_regionkey and lo_orderdate =
      d_datekey and lo_partkey = p_partkey and lo_suppkey =
      s_suppkey and p_brand = 'MFGR#2221' and F_R_REGIONNAME = '
      ASIA' group by d_year order by d_year
6  select sum(lo_revenue) from lineorder, part where lo_partkey =
      p_partkey and p_brand = 'MFGR#2221'
7  select avg(lo_revenue), d_year from lineorder, dates, part,
      supplier where lo_orderdate = d_datekey and lo_partkey =
      p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221
      ' and s_region = 'ASIA' group by d_year order by d_year
8  select sum(lo_revenue), d_year from lineorder, dates, part,
      supplier where lo_orderdate = d_datekey and lo_partkey =
      p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221
      ' and s_region = 'EUROPE' group by d_year, p_brand order by
      d_year, p_brand
9  select sum(lo_revenue) from lineorder, part, supplier where
      lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand
       = 'MFGR#2221' and s_region = 'EUROPE'
10 select count(*), d_year from lineorder, dates, part, supplier
      where lo_orderdate = d_datekey and lo_partkey = p_partkey and
```

```
    lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and
    s_region = 'EUROPE' group by d_year order by d_year
11 select c_nation, s_nation, d_year, sum(lo_revenue) as revenue
    from lineorder,customer, supplier, dates where lo_custkey =
    c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
    d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and
    d_year >= 1992 and d_year <= 1997 group by c_nation, s_nation
    , d_year order by d_year asc, revenue desc
12 select s_nation, sum(lo_revenue) as revenue from lineorder,
    supplier where lo_suppkey = s_suppkey and s_region = 'ASIA'
    group by s_nation order by revenue desc
13 select s_nation, count(*) as revenue from lineorder, supplier
    where lo_suppkey = s_suppkey and s_region = 'ASIA' group by
    s_nation order by revenue desc
14 select s_nation, d_year, sum(lo_revenue) as revenue from
    lineorder, supplier, dates where lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and s_region = 'ASIA' and d_year >=
    1992 and d_year <= 1997 group by s_nation, d_year order by
    d_year asc, revenue desc
15 select c_nation, s_nation, d_year, avg(lo_revenue) as
    avg_revenue from lineorder,customer, supplier, dates where
    lo_custkey = c_custkey and lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and c_region = 'ASIA' and s_region =
     'ASIA' and d_year >= 1992 and d_year <= 1997 group by
    c_nation, s_nation, d_year order by d_year asc, avg_revenue
    desc
16 select c_nation, s_nation, d_year, count(*) from lineorder,
    customer, supplier, dates where lo_custkey = c_custkey and
    lo_suppkey = s_suppkey and lo_orderdate = d_datekey and
    c_region = 'ASIA' and s_region = 'ASIA' and d_year >= 1992
    and d_year <= 1997 group by c_nation, s_nation, d_year order
    by d_year asc
17 select c_city, s_city, d_year, sum(lo_revenue) as revenue from
    lineorder,customer, supplier, dates where lo_custkey =
    c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
    d_datekey and c_nation = 'UNITED␣STATES' and s_nation = '
    UNITED␣STATES' and d_year >= 1992 and d_year <= 1997 group by
     c_city, s_city, d_year order by d_year asc, revenue desc
18 select s_city, sum(lo_revenue) as revenue from lineorder,
    supplier where lo_suppkey = s_suppkey and s_nation = 'UNITED␣
```

```
   STATES' group by s_city order by revenue desc
19 select s_city, avg(lo_revenue) as avg_revenue from lineorder,
   supplier where lo_suppkey = s_suppkey and s_nation = 'UNITED␣
   STATES' group by s_city order by avg_revenue desc
20 select c_city, s_city, count(*) from lineorder,customer,
   supplier where lo_custkey = c_custkey and lo_suppkey =
   s_suppkey and c_nation = 'UNITED␣STATES' and s_nation = '
   UNITED␣STATES' group by c_city, s_city
21 select c_city, s_city, d_year, avg(lo_revenue) as avg_revenue
   from lineorder,customer, supplier, dates where lo_custkey =
   c_custkey and lo_suppkey = s_suppkey and lo_orderdate =
   d_datekey and c_nation = 'UNITED␣STATES' and s_nation = '
   UNITED␣STATES' and d_year >= 1992 and d_year <= 1997 group by
    c_city, s_city, d_year order by d_year asc, avg_revenue desc
22 select c_city, s_city, d_year, count(*) from lineorder,customer,
    supplier, dates where lo_custkey = c_custkey and lo_suppkey
   = s_suppkey and lo_orderdate = d_datekey and c_nation = '
   UNITED␣STATES' and s_nation = 'UNITED␣STATES' and d_year >=
   1992 and d_year <= 1997 group by c_city, s_city, d_year order
    by d_year asc
23 select d_year, s_nation, p_category, sum(lo_revenue -
   lo_supplycost) as profit from lineorder,dates, customer,
   supplier, part where lo_custkey = c_custkey and lo_suppkey =
   s_suppkey and lo_partkey = p_partkey and lo_orderdate =
   d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA'
   and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1'
   or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category
   order by d_year, s_nation, p_category
24 select d_year, s_nation, p_category, sum(lo_revenue -
   lo_supplycost) as profit from lineorder, dates, supplier,
   part where lo_suppkey = s_suppkey and lo_partkey = p_partkey
   and lo_orderdate = d_datekey and s_region = 'AMERICA' and (
   d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or
   p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category
   order by d_year, s_nation, p_category
25 select d_year, s_nation, p_category, avg(lo_revenue -
   lo_supplycost) as avg_profit from lineorder, dates, customer,
    supplier, part where lo_custkey = c_custkey and lo_suppkey =
    s_suppkey and lo_partkey = p_partkey and lo_orderdate =
   d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA'
```

```
    and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1'
    or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category
    order by d_year, s_nation, p_category
26  select d_year, s_nation, p_category, count(*) from lineorder,
    dates, customer, supplier, part where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey and lo_partkey = p_partkey and
    lo_orderdate = d_datekey and c_region = 'AMERICA' and
    s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and
     (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year,
    s_nation, p_category order by d_year, s_nation, p_category
27  select d_year, s_nation, count(*) from lineorder,dates, supplier
     where lo_suppkey = s_suppkey and lo_orderdate = d_datekey
    and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998)
     group by d_year, s_nation order by d_year, s_nation
28  select s_nation, count(*) from lineorder,supplier where
    lo_suppkey = s_suppkey and s_region = 'AMERICA' group by
    s_nation order by s_nation
29  select d_year, s_nation, sum(lo_revenue) as revenue from
    lineorder,dates, supplier where lo_suppkey = s_suppkey and
    lo_orderdate = d_datekey and s_region = 'AMERICA' and (d_year
     = 1997 or d_year = 1998) group by d_year, s_nation order by
    d_year, s_nation
30  select sum(lo_revenue), p_brand from lineorder, part, supplier
    where lo_partkey = p_partkey and lo_suppkey = s_suppkey and
    p_brand = 'MFGR#2221' and s_region = 'ASIA' group by p_brand
    order by p_brand
```

4 Apprentissage pour le MdC orienté énergie

Ci-dessous un aperçu de la charge.

```
1   select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
    O_ORDERPRIORITY, sum(L_QUANTITY), avg(L_QUANTITY), sum(
    L_QUANTITY - L_EXTENDEDPRICE), sum(L_QUANTITY *
    L_EXTENDEDPRICE - 1), avg(L_DISCOUNT * (L_TAX +1)), avg(
    L_QUANTITY / (L_TAX +1)), sum(L_TAX)/sum(L_EXTENDEDPRICE),
    sum(L_QUANTITY * 0.5), avg(L_QUANTITY * 0.2), sum(
    O_TOTALPRICE), avg(O_SHIPPRIORITY), sum(O_TOTALPRICE -
    L_EXTENDEDPRICE), avg(O_SHIPPRIORITY * L_EXTENDEDPRICE - 1),
    sum(O_SHIPPRIORITY)/sum(O_TOTALPRICE), count(*) from lineitem
```

```
   , orders where L_SUPPKEY <= 100000 and L_SUPPKEY >= 5000 and
   o_orderstatus = 'P' and l_orderkey = o_orderkey group by
   L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
   order by L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY;
2 select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY, sum(L_QUANTITY), avg(L_QUANTITY), sum(
   L_QUANTITY - L_EXTENDEDPRICE), sum(L_QUANTITY *
   L_EXTENDEDPRICE - 1), avg(L_DISCOUNT * (L_TAX +1)), avg(
   L_QUANTITY / (L_TAX +1)), sum(L_TAX)/sum(L_EXTENDEDPRICE),
   sum(L_QUANTITY * 0.5), avg(L_QUANTITY * 0.2), sum(
   O_TOTALPRICE), avg(O_SHIPPRIORITY), sum(O_TOTALPRICE -
   L_EXTENDEDPRICE), avg(O_SHIPPRIORITY * L_EXTENDEDPRICE - 1),
   sum(O_SHIPPRIORITY)/sum(O_TOTALPRICE), count(*) from lineitem
   , orders where L_SUPPKEY <= 1000 and L_SUPPKEY >= 500 and
   l_orderkey = o_orderkey group by L_SHIPMODE, L_SHIPINSTRUCT,
   O_ORDERSTATUS, O_ORDERPRIORITY order by L_SHIPMODE,
   L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;
3 select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY, sum(L_QUANTITY), avg(L_QUANTITY), sum(
   L_QUANTITY - L_EXTENDEDPRICE), sum(L_QUANTITY *
   L_EXTENDEDPRICE - 1), avg(L_DISCOUNT * (L_TAX +1)), avg(
   L_QUANTITY / (L_TAX +1)), sum(O_TOTALPRICE), avg(
   O_SHIPPRIORITY), sum(O_TOTALPRICE - L_EXTENDEDPRICE), avg(
   O_SHIPPRIORITY * L_EXTENDEDPRICE - 1), count(*) from lineitem
   , orders where L_SUPPKEY <= 100000 and L_SUPPKEY >= 5000 and
   o_orderstatus = 'P' and l_orderkey = o_orderkey group by
   L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
   order by L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY;
4 select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY, sum(L_QUANTITY), avg(L_QUANTITY), sum(
   L_QUANTITY - L_EXTENDEDPRICE), sum(L_QUANTITY *
   L_EXTENDEDPRICE - 1), avg(L_DISCOUNT * (L_TAX +1)), avg(
   L_QUANTITY / (L_TAX +1)), sum(O_TOTALPRICE), avg(
   O_SHIPPRIORITY), sum(O_TOTALPRICE - L_EXTENDEDPRICE), avg(
   O_SHIPPRIORITY * L_EXTENDEDPRICE - 1), count(*) from lineitem
   , orders where L_SUPPKEY <= 1000 and L_SUPPKEY >= 500 and
   l_orderkey = o_orderkey group by L_SHIPMODE, L_SHIPINSTRUCT,
   O_ORDERSTATUS, O_ORDERPRIORITY order by L_SHIPMODE,
```

```
   L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;
5  select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY, sum(L_QUANTITY), avg(L_QUANTITY), sum(
   L_QUANTITY - L_EXTENDEDPRICE), sum(L_QUANTITY *
   L_EXTENDEDPRICE - 1), sum(O_TOTALPRICE), avg(O_SHIPPRIORITY),
    count(*) from lineitem, orders where L_SUPPKEY <= 100000 and
    L_SUPPKEY >= 5000 and o_orderstatus = 'P' and l_orderkey =
   o_orderkey group by L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS
   , O_ORDERPRIORITY order by L_SHIPMODE, L_SHIPINSTRUCT,
   O_ORDERSTATUS, O_ORDERPRIORITY;
6  select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY, sum(L_QUANTITY), avg(L_QUANTITY), sum(
   L_QUANTITY - L_EXTENDEDPRICE), sum(L_QUANTITY *
   L_EXTENDEDPRICE - 1), sum(O_TOTALPRICE), avg(O_SHIPPRIORITY),
    count(*) from lineitem, orders where L_SUPPKEY <= 1000 and
   L_SUPPKEY >= 500 and l_orderkey = o_orderkey group by
   L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
   order by L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS,
   O_ORDERPRIORITY;
7  select L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, sum(L_QUANTITY
   ), avg(L_QUANTITY), sum(O_TOTALPRICE), count(*) from lineitem
   , orders where L_SUPPKEY <= 100000 and L_SUPPKEY >= 5000 and
   o_orderstatus = 'F' and l_orderkey = o_orderkey group by
   L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS order by L_SHIPMODE
   , L_SHIPINSTRUCT, O_ORDERSTATUS;
```

Table des figures

1 Évolution en nombre et en type des SGBD 3

1 Cafetière de *Carelman* 14
1.2 Relation entre la BD, le SGBD et le SI 18
1.3 Cycle de développement du SI vs. celui de la BD 18
1.4 Aperçu des étapes du cycle de vie des bases de données [58] 19
1.5 Exemple d'une représentation hiérarchique 22
1.6 Exemple d'une représentation réseau 23
1.7 Un exemple de modélisation entités/associations 26
1.8 L'architecture ANSI/SPARC 27
1.9 BDBO vs. BD 33
1.10 Exemple d'un cube multidimensionnel des *Vente*s 37
1.11 Exemple d'une modélisation d'ED en étoile [12] 39
1.12 Exemple d'une modélisation d'ED en flocon de neige [12] 40
1.13 Bilan des évolutions verticales/horizontales du cycle de conception des BD . . 41
1.14 La décomposition de la qualité du logiciel en caractéristiques et sous-caractéristiques (norme *ISO/CEI 9126*) [55] 44

2.1 Architecture 3-tiers typique des applications 53
2.2 Aperçu du problème de la conception physique et du tuning 58
2.3 L'historique de la réutilisation : d'un usage ad-hoc à systématique [49] 63
2.4 Étapes de la gestion de variabilité 71
2.5 Aperçu d'un diagramme de caractéristiques des avions 72

2.6 Répartition des travaux de gestion de variabilité sur le cycle de conception de BD 77

3.1 Extrait du modèle des features du design conceptuel des BD 88
3.2 Exemple de variabilité de la portée des entités du schéma conceptuel [166] . . . 89
3.3 Extrait n°1 du modèle des features de la conception logique des BD 89
3.4 Extrait n°2 du modèle des features de la conception logique des BD 90
3.5 Extrait du modèle des features de la conception physique des BD 91
3.6 Aperçu de l'implémentation de *FeatureIDE* de nos FM de conception des BD . 95
3.7 Aperçu de la configuration des BD 96
3.8 L'architecture en cinq couches des systèmes de BD 98
3.9 Un aperçu simplifié de l'arbre correspondant au processus de conception des BD 99
3.10 Répartition des types de BnF sur le cycle de conception des BD 100
3.11 Un extrait du modèle conceptuel du benchmark SSB 106
3.12 Fonctionnement de l'optimiseur des requêtes (optimisation logique et physique) 112
3.13 Avantage de la technique d'optimisation multi-requêtes 113
3.14 Classification des techniques de sélection de vues à matérialiser [9] 115
3.15 Arbre associé à notre cas d'étude de la variabilité logique 116

4.1 Aperçu de l'arbre associé à l'impact de la variabilité logique sur l'optimisation logique 122
4.2 Les décompositions possibles pour une hiérarchie de trois niveaux 124
4.3 Exemple de l'arbre pour la génération des décompositions possibles 125
4.4 La consommation d'énergie et le plan d'exécution d'une requête exécutée avec pipeline [154] 138
4.5 Exemple de rendu du module de surveillance SQL d'Oracle lors de l'exécution d'une requête 138
4.6 Architecture d'expérimentations 141
4.7 Schéma logique multidimensionnel du banc-d'essai SSB 144
4.8 Impact de la variabilité logique sur la performance des données de 1G 145
4.9 Impact de la variabilité logique sur la performance des données de 10G 146
4.10 Les coûts de requêtes d'origine de SSB dans le schéma étoile 147
4.11 Comparaison entre les coûts des requêtes dans le meilleur schéma (n°17) et le schéma en étoile 148

4.12 Comparaison entre l'exécution réelle de la charge SSB sur le schéma étoile et le meilleur schéma (17) 149
4.13 Impact de la conception logique sur la consommation énergétique de l'ED . . . 151
4.14 Impact de la conception logique sur la performance et la puissance de l'ED . . 152
4.15 Schéma du benchmark TPC-H 153
4.16 Impact de la variabilité logique sur la taille d'implémentation de l'ED 155

5.1 Extrait de l'arbre modélisant l'impact interphase de la variabilité logique sur l'optimisation physique 159
5.2 Exemple d'hypergraphe de jointures He_1 [9] 161
5.3 Résultat du partitionnement de l'hypergraphe He_1 [9] 162
5.4 Étapes de transformation d'un hypergraphe à un graphe orienté (plan global) [9] 162
5.5 Les agencements possibles des noeuds de jointure dans le plan global 163
5.6 L'impact de la variabilité logique sur la performance des VM 168
5.7 Impact de la variabilité logique associée aux VM sur l'énergie et le temps dépensés 171

8 Gestion holistique de la variabilité de la conception des BD 178

Liste des tableaux

2.1 Quelques approches de gestion de variabilité de la conception de BD (par axe) . 50

2.2 Les dimensions *Quoi*, *Quand* et *Comment* du test du processus de conception des BD 52

2.3 Classification des bancs d'essais existants 56

2.4 Comparaison des principaux advisors[34] 60

2.5 Comparaison des principaux outils CASE pour bases de données 65

2.6 Répartition des design patterns proposés sur les phases de conception de BD . . 68

2.7 Comparaison d'approches de gestion de variabilité de la conception de BD . . . 78

2.8 Classification des travaux orientés *gestion explicite de variabilité* de conception de BD 78

3.1 Quelques outils de gestion de variabilité basés sur FODA [61] 94

3.2 Travaux connexes sur l'exploitation des corrélations tout au long du cycle de conception des BD 110

4.1 Paramètres des fonctions de l'estimation des coûts 134

4.2 Les caractéristiques de certains schémas logiques de référence 152

4.3 Erreurs d'estimation d'énergie pour les requêtes du benchmark TPC-H avec différents tailles et schémas de BD. 154

5.1 Les paramètres de l'algorithme évolutionnaire 169

Glossaire

BD : Base de données
BDBO : Base de Données à Base Ontologique
BnF : Besoin non fonctionnel
CPU : Processeur
CFAO : Conception et fabrication assistées par ordinateur
AE : Algorithme Evolutionnaire
ED : Entrepôt de données
E/S : Entrées/Sorties
ETL : Processus d'extraction, transformation et chargement des données
FM : Modèle des features
LIAS : Laboratoire d'Informatique et d'Automatique pour les Systèmes
LdP : Ligne de Produits
MCD : Modèle Conceptuel de Données
MLD : Modèle Logique de Données
MPD : Modèle Physique de Données
MdC : Modèle de Coût
mvpp : multi-view processing plan
OMR : Optimisation Multi-Requêtes
OLAP : On-Line Analytical Processing
OLTP : On-Line Transactional Processing
PSV : Problème de sélection de vues à matérialiser
SGBD : Système de Gestion de Base de données
SSB : Star Schema Benchmark
SO : Structure d'Optimisation
VM : Vue Matérialisée

Vers une conception de bases de données orientée variabilité : le cas de la conception logique

Présentée par :

Selma BOUARAR

Directeurs de Thèse :

Ladjel BELLATRECHE et **Stéphane JEAN**

Résumé. Le processus de conception des BD ne cesse d'augmenter en complexité et d'exiger plus de temps et de ressources afin de contenir la diversité des applications BD. Rappelons qu'il se base essentiellement sur le talent et les connaissances des concepteurs. Ces bases s'avèrent de plus en plus insuffisantes face à la croissante diversité de choix de conception, en soulevant le problème de la fiabilité et de l'exhaustivité de cette connaissance. Ce problème est bien connu sous le nom de la gestion de la variabilité en génie logiciel. S'il existe quelques travaux de gestion de variabilité portant sur les phases physique et conceptuelle, peu se sont intéressés à la phase logique. De plus, ces travaux abordent les phases de conception de manière séparée, ignorant ainsi les différentes interdépendances. Dans cette thèse, nous présentons d'abord la démarche à suivre afin d'adopter la technique des lignes de produits et ce sur l'ensemble du processus de conception afin de (i) considérer les interdépendances entre les phases, (ii) offrir une vision globale au concepteur, et (iii) augmenter l'automatisation. Vu l'étendue de la question, nous procédons par étapes dans la réalisation de cette vision, en consacrant cette thèse à l'étude d'un cas choisi de façon à montrer : (i) l'importance de la variabilité de la conception logique, (ii) comment la gérer en offrant aux concepteurs l'exhaustivité des choix, et la fiabilité de la sélection, (iii) son impact sur la conception physique (gestion multi-phase), (iv) l'évaluation de la conception logique, et de l'impact de la variabilité logique sur la conception physique (sélection des vues matérialisées) en termes des besoins non fonctionnel(s) : temps d'exécution, consommation d'énergie voire l'espace de stockage.

Mots-clés : Bases de données--Conception logique et physique, Gestion de la variabilité; Lignes de produits, Entrepôt de données, Sélection de vues matérialisées, Besoins non fonctionnels -- temps d'exécution-- consommation d'énergie--espace de stockage.

Abstract. The evolution of computer technology has strongly impacted the database design process which is henceforth requiring more time and resources to encompass the diversity of DB applications. Note that designers rely on their talent and knowledge, which have proven insufficient to face the increasing diversity of design choices, raising the problem of the reliability and completeness of this knowledge. This problem is well known as variability management in software engineering. While there exist some works on managing variability of physical and conceptual phases, very few have focused on logical design. Moreover, these works focus on design phases separately, thus ignore the different interdependencies. In this thesis, we first present a methodology to manage the variability of the whole DB design process using the technique of software product lines, so that (i) interdependencies between design phases can be considered, (ii) a holistic vision is provided to the designer and (iii) process automation is increased. Given the scope of the study, we proceed step-by-step in implementing this vision, by studying a case that shows: (i) the importance of logical design variability (iii) its impact on physical design (multi-phase management), (iv) the evaluation of logical design, and the impact of logical variability on the physical design (materialized view selection) in terms of non-functional requirements: execution time, energy consumption and storage space.

Keywords: Databases -- logical and physical design, Variability management, Software product lines, Data warehousing, Materialized view selection, Nonfunctional requirements -- execution time-- energy consumption-- storage space.
