L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels

Amira Kerkad

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

THESE

pour l'obtention du Grade de

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

(Diplôme National — Arrêté du 7 août 2006)

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

Présentée par :

Amira KERKAD

****

L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels

****

Directeurs de Thèse : **Ladjel BELLATRECHE** et **Dominique GENIET**

Soutenue le 11/12/2013
devant la Commission d'Examen

JURY

Rapporteurs :	**Christine COLLET**	**Professeur, Institut Polytechnique de Grenoble**
	Bruno DEFUDE	**Professeur, TELECOM SudParis, Evry.**
Examinateurs :	**Pedro FURTADO**	**Professeur, Université de Coimbra**
	René SCHOTT	**Professeur, Université Henri Poincaré, Nancy**
	Ladjel BELLATRECHE	**Professeur, ISAE-ENSMA, Poitiers**
	Dominique GENIET	**Maître de conférences - HDR, Université de Poitiers**

Remerciements

Je mesure la chance qui m'a été donnée d'être encadrée par des personnes aussi impliquées, ouvertes et compétentes, qui ont cru en mes capacités pour mener à bien ce travail, et sans qui ce travail n'aurait jamais vu le jour. Qu'ils trouvent ici le témoignage de toute ma gratitude :

Ladjel BELLATRECHE, qui sait faire voler en éclat chaque idée reçue, qui s'est dévoué généreusement à ma thèse et qui m'a donné goût à la recherche. Il m'a fait énormément apprendre tant sur le plan scientifique qu'humain.

Dominique GENIET, pour son encadrement minutieux et ses qualités pédagogiques et scientifiques. Ses conseils avisés sont précieux et sa gentillesse m'est inégalée.

Un grand merci à Christine COLLET et Bruno DEFUDE de m'avoir fait l'honneur d'être rapporteurs de cette thèse, ainsi qu'à Pedro FURTADO et René SCHOTT pour avoir accepté d'être membres du jury en tant qu'examinateurs. Je suis très honorée de l'intérêt qu'ils ont porté à mes travaux.

Mes remerciements vont également :

Aux directeurs du laboratoire : à Yamine AIT-AMEUR pour ses qualités humaines hors du commun, son écoute et son dévouement, et à Emmanuel GROLLEAU pour son aide et sa sympathie. Sans oublier le défunt Guy Pierra que j'ai eu l'honneur de connaître au début de ma thèse.

À tout le personnel du laboratoire, surtout Claudine RAULT et Mickael BARON pour leur disponibilité et leur bonne humeur.

À mes ami(e)s et collègues sans exception, pour leur présence, leur respect et leur gentillesse.

Je ne saurais finir sans exprimer mes remerciements aux personnes qui me sont les plus proches : Mes parents, qui m'ont tout donné pour apprendre et pour réussir; Mes deux sœurs Ryma et Hana, qui étaient toujours présentes pour m'aider et me réconforter; Et enfin, et surtout, à Rabah, mon mari qui a été à mes côtés, qui a cru en moi, et qui a toujours su me redonner le sourire dans les moments difficiles.

À mon Algérie bien-aimée,
À tous ceux qui me sont chers :
Mes parents,
Mes sœurs,
Rabah.

Table des matières

Chapitre 1

La Technologie de Bases de Données a t-elle donné le meilleur d'elle-même?

Sommaire

1 Problématique

Au cours de la dernière décennie, la technologie des bases de données a connu un essor spectaculaire contribuant à l'apparition de nouveaux types de bases de données comme les entrepôts de données ou les bases de données scientifiques et statistiques. Autour de ces nouvelles bases de données, des conférences thématiques nationales et internationales se sont créées: DaWak [1], ACM DOLAP [2], SSDBM [3], EDA [4]. Ces bases de données sont caractérisées par quatre points : (i) leurs schémas comprennent un nombre important de tables (e.g. le schéma de SAP R/3 [5] contient *10 000 tables* [6].), (ii) le volume de données peut atteindre plusieurs Péta-octets (e.g. le cas de SAP R/3) , (iii) la complexité des requêtes comportant des opérations de jointure sur de nombreuses tables et des agrégations impliquant de nombreuses mesures, et (iv) les exigences des utilisateurs (qui sont souvent décideurs) notamment en terme de temps de réponse des requêtes. Ces contraintes ont motivé dans la communauté des bases de données la proposition de solutions avancées. Initialement, les solutions proposées ont concerné la phase physique du cycle de vie de la conception des bases de données. Cette phase est rapidement devenue un enjeu important, comme l'indique Surajit Chaudhuri le *ex-leader* du groupe de base de données de MicroSoft : *Data Management, Exploration and Mining Group* dans son papier intitulé *Self-Tuning Database Systems: A Decade of Progress* qui a eu le prix de *10 Year Best Paper Award* à la conférence VLDB'2007 [7]: *"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."* [67].

Durant la phase de conception physique, l'administrateur doit effectuer quatre tâches principales : (1) le choix des structures d'optimisation, (2) le choix de leur mode de sélection, (3) le développement des algorithmes de sélection et (4) la validation et le déploiement des solutions d'optimisation [23].

1. Choix des structures d'optimisation : il existe une large panoplie de structures d'optimisation pour la conception physique. L'identification des structures pertinentes exige un haut niveau d'expertise, car elle dépend de l'étude de la charge à optimiser et de l'environnement sur lequel celle-ci s'exécute.

2. Choix du mode de sélection : l'optimisation peut être effectuée en utilisant une ou plusieurs structures d'optimisation. Dans le premier cas, il s'agit d'une sélection isolée. Dans le deuxième, il s'agit d'une sélection multiple. Dans ce cas, plusieurs scenarii sont possibles pour combiner ces techniques [52]. Le choix de l'ensemble des structures à employer et du mode de combinaison est déterminant en matière de performance du système.

3. Développement des algorithmes de sélection : les algorithmes proposés pour chaque problème d'optimisation subissent une certaine évolution, en passant d'algorithmes simples vers des algorithmes lourds. Les approches simples sont souvent faciles à mettre en œuvre, mais donnent une faible efficacité (comme les approches de la gestion du buffer [74]). Les algorithmes lourds donnent

1. International Conference on Data Warehousing and Knowledge Discovery
2. ACM International Workshop on Data Warehousing and OLAP
3. International Conference on Scientific and Statistical Database Management
4. Journée Francophone sur les Entrepôts de Données et l'Analyse en ligne
5. Runtime System Three.
6. http://www.clearbluegroup.net/saparticles.html
7. Very Large Database Conference: `www.vldb.org`

une meilleur efficacité mais avec un temps d'optimisation élevé (comme les approches de la fragmentation horizontale [60]). Les compromis s'avèrent très rares dans le problème de la conception physique.

4. La validation et le déploiement des solutions d'optimisation : les recommandations obtenues d'un algorithme de résolution nécessitent une validation par le déploiement sur un environnement réel pour évaluer leur performance effective. Plusieurs outils commerciaux proposent ces services comme *Oracle SQL Acces Advisor* [6] et *DB2 Design Advisor* [201]. Cependant, ces outils présentent des limites liées aux structures d'optimisation et au mode de sélection fourni.

Le deuxième effort de la communauté a été consacré à la proposition d'environnements architecturaux souvent centrés sur la phase de déploiement. Les progrès technologiques de la dernière décennie ont permis l'amélioration significative de la puissance de calcul des ordinateurs. Néanmoins, cette puissance n'est souvent pas suffisante pour l'exécution de nouvelles applications très consommatrices en terme de ressources de calcul comme les entrepôts de données. Par conséquent, les processeurs sont combinés et reliés par un réseau de communication, formant ainsi des architectures parallèles (comme les clusters des bases de données ou les grilles de calcul) [46, 24, 31, 25]. Récemment, la technologie de *cloud* a émergé où plusieurs solutions de déploiement des bases de données volumineuses ont été proposées [9]. Dans ce contexte, de nouvelles structures d'optimisation (e.g. répartition des données, allocation des données, équilibrage de charge, définition d'index locaux et globaux, gestion du cache, etc.) et paradigmes de programmation (e.g. map-reduce) ont été proposées [9]. En conséquence, ce progrès technologique a souvent rendu possible la conception physique hiérarchisée (une optimisation globale et locale). Bien entendu, ces technologies coûtent cher notamment pour des petites et moyennes entreprises.

Pour améliorer la performance des applications utilisant des bases de données volumineuses, la technologie NoSQL (Not Only SQL) a été proposée [177]. Elle propose de s'en passer du schéma classique de base de données. Divers projets s'intègrent dans cette approche : MongoDB [5], CouchDB [3], Cassandra [1], BigTable (Google) [63] et HBase (Facebook) [4].

Finalement, nous constatons que la communauté des bases de données recherche systématiquement des solutions pour satisfaire les exigences de nouvelles applications consommatrices de volume de données, soit en augmentant la puissance de calcul des machines les hébergeant, soit en proposant de nouveaux modèles de conception et de stockage. Dans cette thèse, nous nous intéressons au fait suivant : certes le progrès technologique apporte des solutions performantes, mais il y a un coût à payer. Nous proposons alors de revisiter les différentes composantes du système classique de base de données en les analysant, afin de faire coopérer les structures d'optimisation pour offrir des solutions plus efficaces durant la phase de conception physique.

L'analyse de l'environnement de base de données et du processus de conception physique révèle l'existence de composantes omniprésentes : le schéma de la base de données, son support de stockage (environnement), la charge de requêtes, et les structures d'optimisation que le système de gestion de base de données offre aux administrateurs. Chaque composante comporte un ensemble d'éléments qui interagissent entre eux. Cette interaction concerne souvent les éléments d'une seule composante indépendamment des autres. La composante dédiée aux structures d'optimisation est la plus explorée par les travaux existants. Dans des travaux récents [169, 33, 201], les auteurs ont identifié les interactions entre les structures d'optimisation. Considérons par exemple les vues matérialisées et les index [21]. Elles produisent deux schémas d'optimisation redondants, partageant la même ressource (l'espace de stockage) et

nécessitent des mises à jour régulières. Cette interaction a été exploitée pour sélectionner simultanément les index et les vues matérialisées. Pour la composante schéma de base de données, certains travaux exploitent la corrélation entre les attributs pour sélectionner les vues matérialisées et les index, comme dans le cas du projet CORADD [2, 133, 134]. L'interaction des attributs a également été exploitée pour définir les hiérarchies des schémas de base de données [10], ainsi que pour la sélection d'un schéma de fragmentation verticale [113]. Dans certains travaux récents comme dans les travaux de Zilio *et al.* [201] et Sanjay *et al.* [170], l'interaction entre les structures d'optimisation a été identifiée. Les supports de stockage ont été étudiés dans un grand nombre de travaux qui cherchent à exploiter de façon efficace la multitude des supports disponibles dans un seul environnement [34, 187]. Cependant, aucun travail ne considère l'interaction des différentes composantes dans leur globalité lors de l'analyse du système.

D'autre part, l'environnement de base de données évolue de façon permanente, tant au niveau des données qu'au niveau des requêtes. Ces changements nécessitent l'adaptation du système en considérant ces changements sur les supports (e.g. augmenter la taille du buffer), les données (e.g. fragmentation horizontale), les requêtes (e.g. ordonnancement) et les structures d'optimisation (e.g. nouveau schéma d'indexation). Cependant, la conception physique a toujours été considérée comme un processus statique qui ignore le passage du système d'un état vers un autre. Il nous paraît important de rendre la conception physique évolutive en proposant des approches d'optimisation dynamique ou en-ligne, s'adaptant avec l'évolution du système.

2 Objectifs et contributions

Au laboratoire LIAS de l'ISAE-ENSMA, nous menons des travaux sur la couche physique des bases de données volumineuses depuis quelques années au sein de l'équipe Ingénierie des Données et des Modèles. Ces travaux concernent la sélection des structures d'optimisation comme le partitionnement, les index avancés, les vues matérialisées et l'allocation des données en proposant des algorithmes avancés du fait que le processus de sélection de ces structures est difficile. Dès le début de cette thèse, nous avons commencé par faire un état des lieux sur la manière de sélectionner ces structures. Étant donnée la diversité de ces dernières, nous nous sommes concentrés sur la fragmentation horizontale (FH) car elle a été largement étudiée au sein de notre équipe [22, 28, 29, 52, 31, 24, 45, 25, 51]. La première analyse nous a permis d'identifier l'absence du buffer et l'ordre d'exécution des requêtes dans le processus de sélection. Autrement dit, le modèle de coût proposé pour quantifier la qualité des solutions proposées ne prend pas en compte le contenu du buffer et suppose que les requêtes sont déjà ordonnées. Les compétences sur l'ordonnancement des tâches que nous avons au laboratoire dans l'équipe Temps Réel et Systèmes Embarqués, ont largement favorisé l'étude de l'impact de l'ordonnancement des requêtes sur le processus de sélection. Cela a débouché sur l'étude du problème joint (combiné) intégrant la gestion du buffer et l'ordonnancement des requêtes. Le traitement du problème joint nous a ramené à considérer l'interaction entre les requêtes identifiée par Timos Sellis dans son article intitulé Multiple-Query Optimization [176] afin d'élaguer l'espace de recherche de ce problème. Une autre interaction que nous avons identifiée concerne le lien entre les supports de stockage (la mémoire principale et le disque). Après avoir proposé des solutions pour le problème joint intégrant l'interaction de requêtes, nous avons intégré ces solutions pour répondre à notre problème initial qui est la fragmentation horizontale.

Lors de la phase de conception physique, l'administrateur doit sélectionner un ensemble de struc-

tures optimisant une charge de requêtes et respectant une ou plusieurs contraintes comme l'espace de stockage et/ou le coût de maintenance. Ce que nous avons proposé pour la fragmentation reste valide pour les autres structures d'optimisation classiques comme les index et les vues matérialisées. De ce fait, nous avons proposé une formalisation générique de la conception physique qui met l'accent sur les paramètres explicites que nous trouvons dans n'importe quelle formalisation et les paramètres implicites qui ressortent de l'exploitation de l'interaction entre les principales composantes de système de bases de données.

Dans cette thèse, nous avons considéré deux types d'interaction :

- L'interaction entre les composantes *support* et *requêtes*, en prenant comme cas d'étude la gestion du buffer et l'ordonnancement des requêtes.
- La contribution de l'interaction sur la composante *structures d'optimisation*, en prenant comme cas d'étude la combinaison de la fragmentation horizontale avec la gestion du buffer et l'ordonnancement des requêtes.

En analysant le processus d'optimisation, nous avons également constaté l'absence de l'aspect dynamique dans la conception physique. Cet aspect concerne les changements que l'environnement subit dans un temps fini comme l'arrivée de nouvelles requêtes, le changement des structures d'optimisation existantes ou la mise à jour des données. Pour analyser l'évolution du système, nous établissons une représentation évolutive du processus de la conception physique, suivie par un cas d'étude sur la gestion du buffer et l'ordonnancement des requêtes en-ligne.

3 Organisation de la thèse

D'un point de vue organisationnel, le reste de la thèse s'articule autour de cinq chapitres. Dans le chapitre 2, nous présentons l'état de l'art de la conception physique tout en rappelant les différents concepts requis pour ce travail, avant d'aborder la nouvelle vision qui permet d'expliciter les paramètres de la conception physique. Dans le chapitre 3, nous étudions l'interaction entre les composantes *supports* et *requêtes*. Pour illustrer ce phénomène, nous étudions le problème joint de la gestion du buffer et de l'ordonnancement des requêtes. Dans cette étude nous proposons plusieurs approches dont le but est de contourner la complexité du problème joint. Les approches proposées sont examinées en termes de complexité algorithmique, d'efficacité et de réactivité sur une base de données volumineuse. Dans le chapitre 4, l'interaction entre les structures d'optimisation est étudiée pour établir les liens existants avec les autres composantes. Le cas d'étude dans ce chapitre est la combinaison de la fragmentation horizontale avec la gestion du buffer et l'ordonnancement des requêtes. Il s'agit d'identifier les caractéristiques et les difficultés liées à la sélection multiple touchant aux différentes composantes à la fois. Le chapitre 5 aborde l'aspect dynamique et évolutif d'un système de base de données. L'analyse est effectuée en prenant le cas de la gestion du buffer et d'ordonnancement de requêtes en-ligne. Dans le chapitre 6, nous présentons notre outil d'aide à la prise de décision, que nous avons développé pour simuler les différents cas de sélection dans la conception physique. Nous concluons la thèse par un bilan général sur nos contributions, et l'évocation des perspectives qui s'offrent à l'issue de notre travail.

Les principales contributions de notre thèse sont réparties dans les chapitres 3, 4, 5 et 6. La figure 1.1 schématise les grands axes de ce travail et leur répartition dans chaque chapitre. Chaque chapitre a donné lieu à un ensemble de communications et de publications dans des conférences nationales et

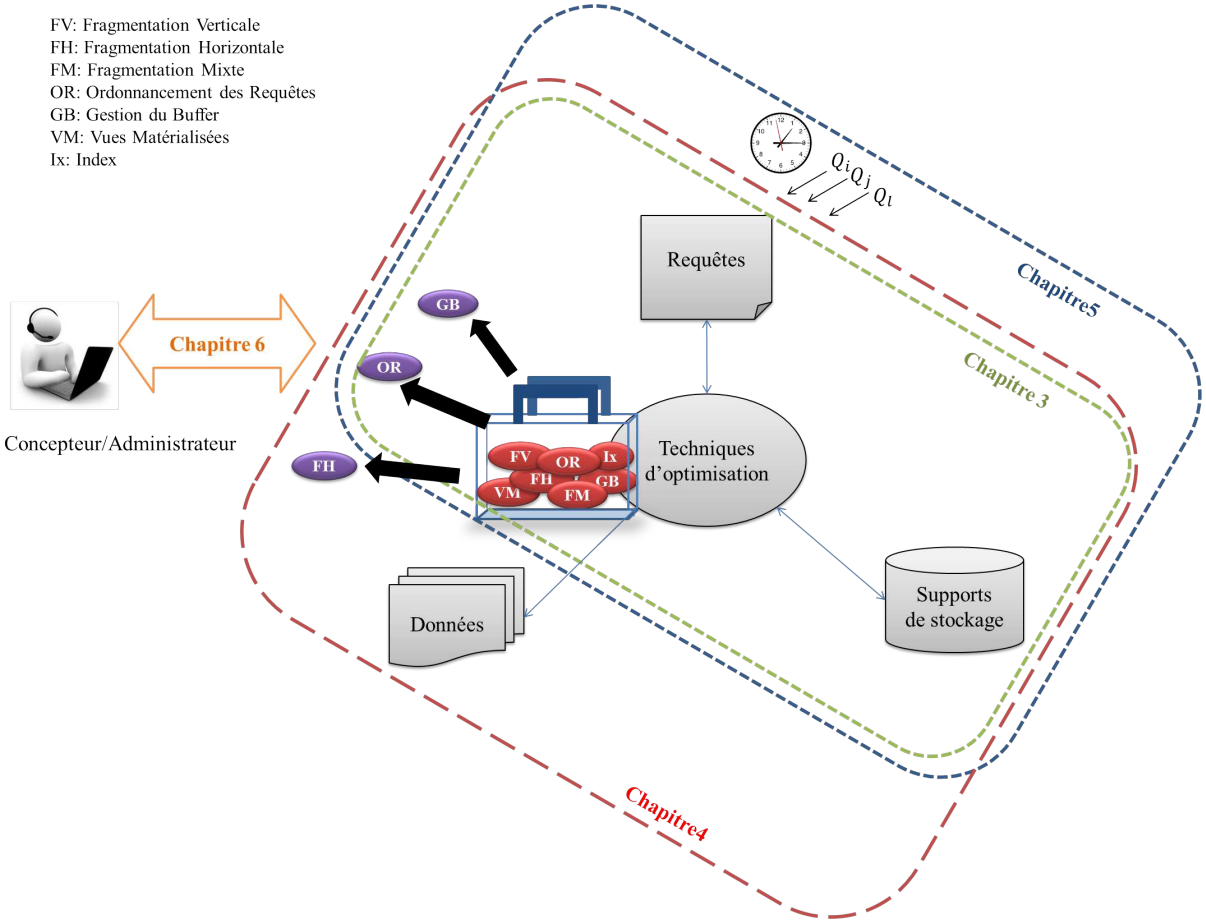

FIGURE 1.1 – La répartition des chapitres de la thèse

internationales :

Publications internationales :

Amira Kerkad, Ladjel Bellatreche, Pascal Richard, Carlos Ordonez, Dominique Geniet : A Query Beehive Algorithm for Data Warehouse Buffer Management and Query Scheduling. To appear in the International Journal of Data Warehousing and Mining (IJDWM), 2014 [128].

Communications internationales :

Amira Kerkad, Ladjel Bellatreche, Dominique Geniet : Simultaneous Resolution of Buffer Allocation and Query Scheduling Problems. 6th International Conference on Complex, Intelligent and software Intensive Systems (CISIS) 2012, pages 122-129 [125].

Amira Kerkad, Ladjel Bellatreche, Dominique Geniet : Queen-Bee: Query Interaction-Aware for Buffer Allocation and Scheduling Problem. 14th International Conference on Data Warehousing and Knowledge Discovery (DaWaK) 2012, pages 156-167 [124].

Ladjel Bellatreche, Amira Kerkad, Sebastian Bress, Dominique Geniet : RouPar: Routinely and Mixed Query-Driven Approach for Data Partitioning. OnTheMove Federated Conferences (OTM) 2013,

pages 309-326 [38].

Amira Kerkad, Ladjel Bellatreche, Dominique Geniet : Database Technology: A World of Interaction. A la conférence internationale : Database and Expert Systems Applications (DEXA) 2013, volume 1, pages 454-461 [126].

Ladjel Bellatreche, Sebastian Bress, Amira Kerkad, Ahcène Boukorca, Cheikh Salmi : The Generalized Physical Design Problem in Data Warehousing Environment: Towards a Generic Cost Model, Proceedings of 36th IEEE International Convention on information and communication technology, electronics and microelectronics (MIPRO), édité par IEEE, 2013 [30].

Ladjel Bellatreche, Salmi Cheikh, Sebastian Bress, Amira Kerkad, Ahcène Boukhorca, Jalil Boukhobza : How to Exploit the Device Diversity and Database Interaction to Propose a Generic Cost Model, 17th International Database Engineering and Applications Symposium (IDEAS) 2013, pages 142-147 [42].

Communications nationales :

Amira Kerkad, Ladjel Bellatreche, Dominique Geniet : Exploitation de l'Interaction des Requêtes OLAP pour la Gestion de Cache et l'Ordonnancement de Traitements. 8ème Journées Francophones sur les Entrepôts de Données et Analyse en Ligne (EDA) 2012, éditée par RNTI, pages 154-163 [123].

Amira Kerkad, Ladjel Bellatreche, Dominique Geniet, La Fragmentation Horizontale Revisitée : Prise en Compte de l'Interaction de Requêtes, 9ème Journées Francophones sur les Entrepôts de Données et Analyse en Ligne (EDA) 2013, pages 124-140 [127].

La répartition des publications et des communications à travers nos chapitres est la suivante :

- Chapitre 2 : [126].
- Chapitre 3 : [124, 125, 123, 42, 128].
- Chapitre 4 : [127, 38, 126].
- Chapitre 5 : [128].
- Chapitre 6 : [30, 42].

Chapitre 2

État de l'art

Sommaire

1 Introduction

Les entreprises à travers le monde connaissent une rude concurrence, où la performance des systèmes d'information et la prise de décision deviennent des enjeux stratégiques. Sous effet de la mondialisation et de l'explosion des technologies des réseaux et de télécommunication, le volume de données devient extrêmement large (plusieurs Téra-octets). Le nombre d'utilisateurs et des requêtes interrogeant les données accroît également, ce qui rend le besoin de performance crucial.

Depuis leur apparition en 1969, les bases de données ont pris une part centrale dans les systèmes d'information. Pendant les quatre dernières décennies, les bases de données (BD) évoluent sur tous les plans avec la proposition de nouveaux modèles de données (modèle hiérarchique, relationnel, etc.) [188, 77], de supports de stockage (HDD, flash, etc.) [183, 158] et d'architecture (centralisée, parallèle, Cloud, etc.) [31, 14, 190]. Les nouvelles générations de bases de données ont permis de combler un grand nombre de problèmes liés aux premiers systèmes, mais ils en ont ouvert de nouveaux problèmes et pistes de recherches liés à la performance et l'optimisation dans ces systèmes.

L'amélioration des performances d'un système de base de données se joue souvent lors de la conception physique de la base de données. Cette phase constitue donc un tournant décisif dans la vie du système. Elle a pris une place importante dans la communauté de bases de données. Les travaux du milieu académique et industriel ont permis de fournir un ensemble de structures et de méthodes d'accès permettant d'augmenter les performances des systèmes en satisfaisant les besoins des requêtes. Parmi les structures existantes, la Fragmentation Horizontale (FH), les Vues Matérialisées (VM), les Index [21]. Chacune des techniques a été traitée dans la littérature de façon isolée, ou en combinaison avec une autre. Cependant, la multitude de techniques et des combinaisons rendent la tâche du concepteur difficile.

Un grand nombre de travaux ont traité ce problème en considérant simultanément différents ensembles de structures d'optimisation. Les approches proposées restent toujours limitées en efficacité face aux besoins exponentiels de performance. Ceci a poussé un nombre de chercheurs à revoir les systèmes de base de données, et à remettre en cause les modèles de données existants.

Dans notre travail, nous revisitions le problème de la conception physique, afin de ressortir les principales limites liées aux approches d'optimisation existantes, et de fournir une nouvelle vision du problème.

Ce chapitre présente l'état de l'art sur la conception physique. Il est organisé en six sections. La première section est consacrée à la conception physique. Dans cette section, nous présentons les bases de données et leur évolution, les entrepôts de données comme un cas d'étude des systèmes de bases de données modernes, et les structures d'optimisation existantes. La deuxième section présente en détail le mode de sélection isolée. Nous prenons le cas d'étude de la fragmentation horizontale comme structure d'optimisation. Dans la troisième section, nous présentons la sélection multiple en prenant comme cas d'étude, la combinaison de la Gestion du Buffer (GB) et l'Ordonnancement des Requêtes (OR). La quatrième section conclu l'étude avec une discussion sur les constats et les déductions que l'on a ressorti de l'état de l'art et des travaux reliés. Une nouvelle vision est proposée suite à cet analyse, dans la section cinq. Nous terminons le chapitre par un bilan synthétisant l'étude effectuée, et présentant le positionnement de notre travail par rapport aux travaux existants.

2 La conception physique

La conception physique ($\mathbb{CP}$) consiste à établir une configuration physique des données, de leurs types, et surtout la sélection des techniques d'optimisation sur le support de stockage [21, 52].

Avec la croissance du volume de données et la complexité des requêtes, la phase de conception physique est devenue primordiale pour les systèmes de bases de données en général et aux entrepôts de données en particulier. Selon Chaudhuri S. et Narasayya V. [67], la $\mathbb{CP}$ n'est nullement un problème nouveau, mais elle reste toujours un problème ouvert. Son importance majeure est de déterminer la façon dont une requête peut s'exécuter efficacement sur le système en exploitant les capacités de l'optimiseur et du moteur d'exécution. Cette importance s'est amplifiée car les optimiseurs deviennent de plus en plus sophistiqués afin de répondre aux besoins transactionnels et décisionnels. Avec l'évolution des techniques d'optimisation et d'exécution des requêtes dans les SGBD, l'administrateur ne peut employer les mêmes modèles simplistes, et son choix des index et d'autres structures d'optimisation devient crucial [21, 67].

Les deux premiers systèmes relationnels qui ont adopté l'analyse des alternatives dans un logiciel dit *optimiseur* sont : System-R et INGRES [94]. L'efficacité des optimiseurs a été largement étudiée depuis la fin des années 70' [47, 69]. Ces études montrent l'importance de tels outils dans l'amélioration des performances des systèmes en général et des requêtes en particulier. Ces études ont favorisé l'adoption des optimiseurs dans plusieurs produits de gestion de bases de données ce qui a contribué à leur amélioration [35].

Dans cette section, nous détaillons les éléments nécessaires pour le processus de $\mathbb{CP}$, en commençant par présenter l'évolution des bases de données, leurs besoins en performance et le traitement de leurs requêtes. Nous présentons par la suite, les entrepôts de données comme un exemple de bases de données avancées. En dernier lieu, nous étudions les techniques d'optimisation en détaillant les différents modes de sélection existants.

2.1 Les bases de données : évolution et besoins

Dans cette section, nous présentons l'histoire d'apparition des bases de données et leur évolution, les tendances affectant leur évolution et les traitements des requêtes sur un SGBD typique.

2.1.1 Évolution des systèmes de bases de données

Pour présenter l'évolution et les dernières générations de systèmes de BD, il convient de faire un retour en arrière pour rappeler la genèse des bases de données telle que la présente Bill Inmon [117] :

L'origine des bases de données revient aux premiers programmes informatiques (années 50'), sauf que ces origines ne sont pas évidentes. Avant, les programmes ont été transformés en applications, ce qui induit l'utilisation des fichiers de données collectées, stockées et manipulées par d'autres programmes. Ces fichiers ont été souvent stockés dans des bandes magnétiques et accédés d'une façon séquentielle. La limite de ces fichiers, était le fait de parcourir la totalité des données pour accéder à un sous-ensemble restreint des données requises. Une autre lacune majeure, était l'accès grotesque et inefficace aux données

lors de la fusion de plusieurs fichiers. Ainsi, les chercheurs ont commencé à revoir la façon dont les données sont organisées.

Quelques années plus tard, le stockage sur disque est devenu courant, et on a pu accéder directement sans avoir à parcourir la totalité des fichiers. C'est ainsi que l'idée des bases de données a surgit. Le terme base de données est apparu en 1969. Depuis les premiers travaux [76], les théoriciens ont défini une base de données comme *"un endroit unique où tous les traitements de données stockées, se produisent"*. Cette définition est restée invariante à travers les générations [117].

La technologie des bases de données évolue constamment depuis quatre décennies. Avec la croissance en volume et la sophistication des systèmes de bases de données, les utilisateurs finaux ont pu créer leur propres transactions et requêtes selon leur besoins, une chose qui n'était pas envisageable quelques années auparavant. Une requête est le terme attribué à une question de l'utilisateur sur les données.

La gestion des données implique à la fois, de définir des structures de stockage pour les données, et de fournir les mécanismes de manipulation efficaces pour celles-ci. Le point fort des BD vient des connaissances et de la technologie développée durant plusieurs décennies, intégrés aux logiciels spécialisés appelés les Systèmes de Gestion de Bases de Données (SGBD). Depuis l'apparition des premières bases de données (la fin des années 1960') [76], les SGBD ont connu plusieurs améliorations. La première amélioration concerne la proposition des SGBD centralisés sous différents modèles [77, 71, 185, 78]. Ces nouveaux SGBD ont été conçus pour surpasser les limites des bases de données traditionnelles, et pour satisfaire les besoins en données structurées. Cette génération de SGBD permet de fournir une organisation et un contrôle d'accès aux données. La limite majeure de ces bases de données, est l'obligation de stocker l'information localement. Cet aspect est devenu le goulet d'étranglement dans ses systèmes. De plus, avec le développement des plate-formes réseaux, les chercheurs ont proposé les BD distribuées comme alternatives aux systèmes centralisés [64, 142, 174, 49, 62, 32, 178, 114, 105, 31]. Parallèlement à cette évolution des architectures, les structures de données ont également évolué en passant des modèles hiérarchiques [188] et réseaux [109, 13], vers le modèle relationnel proposé par Codd [77] en 1970. La décennie suivante, plusieurs types de BD ont été proposés comme les BD orientée-objet [15, 16, 137, 129, 130, 12, 48, 159, 7, 102, 98, 37], résultant de l'apparition de la programmation objet. Vers les années 90', les données et les traitements informatisés se sont largement propagés. Cependant, les données manquaient d'intégrité, et le contrôle des données était restreint. Les entrepôts de données (ED) sont nées des besoins de cet environnement [116]. Les ED est un type différent de bases de données contenant l'histoire des données, une chose souvent indésirables pour les BD traditionnelles, car elle réduit la performance et occupe un volume important sur les supports. De plus, à leur lancement, les théoriciens n'appréciaient pas les ED car ils ne vérifient pas les règles établies pour les BD, comme la normalisation. Toutefois, les entrepôts sont granulaires et flexibles, ce qui leur a permis de devenir un standard dans l'infrastructure de traitement de l'information [117]. Plusieurs types de bases de données ont été proposés suite à l'apparition des entrepôts de données, comme les BD scientifiques et statistiques [138, 18, 162].

Les années 2000' étaient une décennies marquée par de nombreuses évolutions, où beaucoup de systèmes ont vu le jour comme les BD sur grille [194], et les BD Cloud [14, 57, 190, 9].

Actuellement, les bases de données deviennent un besoin crucial dans tous les traitements. Elles nécessitent une gestion rigoureuse pour leur bon fonctionnement. Le SGBD intégré aux bases de données, est un outil puissant pour la création et la gestion des volumes importants de données de façon efficace,

et permettant la persistance sécurisée pour une longue durée. Ses principales fonctions sont :

- la création de nouvelles BD par l'utilisateur, en spécifiant leur schéma (logique, structures de données, etc.) en utilisant un langage dit langage de définition de données (LDD).
- l'interrogation des données à l'aide des requêtes établies par l'utilisateur, et la modification de celles-ci par un langage spécifique appelé langage de manipulation de données (LMD).
- le stockage sécurisé des volumes importants de données (des Giga-octets, voire des Téra-octets), pendant toute la durée de vie de la BD, ainsi que la modification de celles-ci par les utilisateurs autorisés.
- le contrôle d'accès aux données par plusieurs utilisateurs à la fois, tout en gardant leur cohérence.

Pour résumer l'évolution des BD, nous présentons la classification établie par Ullman *et al.* [101], où plusieurs classes d'évolution ont été identifiées :

Les premiers systèmes de BD :

Les premières bases de données ont vu le jour vers les années 60'. Le stockage des données dans ces bases est orienté fichier (*file-based systems*). Cependant, ces systèmes ne garantissent pas la persistance des données pour une longue durée. De plus, ils ne supportent pas l'accès concurrent. Ces systèmes ont été utilisés pour les opérations bancaires et les réservations pour les compagnies aériennes.

Les premières générations utilisaient plusieurs modèles pour décrire la structure des données, parmi ces modèles, il y a le modèle hiérarchique [188] et le modèle réseau [109, 13]. Ce dernier a été standardisé par le rapport établi par CODASYL (Committee on Data Systems and Languages) [13]. Le problème de ces modèles réside dans leur incapacité de supporter les langages de requêtes de haut niveau, dits langage de quatrième génération (e.g. SQL).

Les systèmes relationnels :

Suite à la publication effectuée par Codd E. F. en 1970 [77], les systèmes de BD ont changé considérablement. Codd propose que les SGBD doivent présenter à l'administrateur une vue des données organisées comme tables appelées *relations*. En coulisses, il existe une structuration de données plus complexe permettant l'accès rapide aux données, et une réponse prompte aux requêtes. Contrairement aux anciens systèmes, l'utilisateur n'est pas concerné par la façon dont les données sont stockées. De plus, les requêtes peuvent être exprimées par des langages de haut-niveau. SQL (*Structured Query Language*) fût le langage le plus largement utilisé dans les modèles relationnels. Vers les années 1990', les systèmes relationnels sont devenus la norme des BD. Toutefois, les bases de données ont continué à évoluer, et de nouvelles approches de gestion de données ont été proposées [48, 159, 102, 98, 138, 18, 162, 14].

La puissance des modèles relationnels en terme de structuration et de gestion des données, a permis une explosion de l'utilisation des BD, et leur volume est devenu de plus en plus important. Ceci a induit le besoin d'optimiser les requêtes dans ces systèmes, et la nécessité de trouver des méthodes d'accès plus efficaces.

Les systèmes de plus en plus petits :

Au départ, les SGBD étaient volumineux, coûteux et implémenté sur les anciens ordinateurs. Leur volume était indispensable pour stocker les giga-octets de données. Aujourd'hui, il devient possible d'implanter un système sur un seul disque dans un ordinateur personnel. Ainsi, les SGBD supportant le modèle relationnel sont devenus disponible pour des machines plus petites (les nouveaux ordinateurs personnels) [73].

Les systèmes de plus en plus grands :

D'un autre côté, avec la disponibilité des supports de stockage de plus en plus rapides et larges, et de moins en moins coûteux, les quelques giga-octets de données ont augmenté pour atteindre les téra-octets par base de données [162, 57, 26].

Dans les nouvelles générations, les BD ne se limitent plus au stockage simple des données (entier, caractères, etc.), mais ils permettent de stocker des images, audio et vidéo en passant vers les bases de données multi-média [193].

La manipulation des BD larges nécessite des technologies avancées. Les SGDB supposent donc que les données deviennent trop grandes pour se loger en mémoire principale. Pour remédier à ce problème de volume, deux principales solutions ont été proposées : (1) Le *stockage tertiaire* [161, 34], ou plusieurs supports de stockage peuvent être employés à la fois, et (2) le *traitement parallèle*, où les données sont traitées plus rapidement par plusieurs unités de calcul dites *machines parallèles* [151, 99, 31].

Architecture Client-Serveur et Multi-niveaux :

Plusieurs variétés des 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 [97, 103].

Intégration de l'information :

Comme l'information devient essentielle pour la vie de tous les jours, une même source d'information devient exploitée de différentes façons. Les chercheurs ont par conséquent pensé à l'intégration des sources de données [68, 189, 56, 8, 148, 41]. L'une des solutions proposées est la création des entrepôts de données, où l'information est obtenue à partir de plusieurs sources hétérogènes, avec un processus d'extraction, de transformation et d'intégration adéquat [131, 66, 135, 95, 83, 82, 20, 150].

2.1.2 Les tendances affectant l'évolution des bases de données

L'évolution des systèmes de bases de données résulte de plusieurs facteurs. En 1999, un groupe de chercheurs s'est réuni afin d'examiner les prospects pour les futures recherches dans le domaine de bases de données. Le rapport résultant a établi une critique des systèmes antérieurs, et a souligné leurs points forts [179]. Une autre mise au point sur l'évolution des bases de données a été effectuée en 1995 [106]. Il en ressort les constatations suivantes [180]:

- *La communauté de recherche en base de données joue un rôle fondamental dans la création de l'infrastructure technologique sur laquelle les bases de données évoluent.*
- *Les applications de bases de données des générations futures sont affectées par l'explosion de l'information numérique des cinq dernières années, et feront ressortir de nouveaux problèmes de recherche.*
- *Un nouveau mandat de recherche pour les bases de données est assuré par le développement technologique du passé proche, à savoir l'explosion des capacités des supports de stockage, des vitesses de traitements, et de la télécommunication.*
- *Il y a un besoin accru d'aide de l'industrie et des gouvernements, pour la recherche en base de données afin de répondre à ces challenges.*

Silberschatz *et al.* [180] ont étudié l'évolution des SGBD jusqu'au 21-ième siècle. Dans leur étude,

les auteurs ont exposé les réalisations accomplies dans le domaine des bases de données, et ont prédit l'avenir de ces systèmes. Aussi, ils ont énuméré les principaux aspects pouvant affecter leur évolution :

Les tendances technologiques :

Durant les dernières décennies, plusieurs paramètres de performance ont connu des améliorations exponentielles. Les facteurs suivants ont évolué d'un facteur de 10 chaque décennie : le nombre d'instructions machine s'exécutant par seconde, le coût d'exécution du processeur, le taux de stockage en mémoire secondaire pour une unité de coût et le taux de stockage en mémoire principale par unité de coût.

Ces améliorations en coût/performance des composantes critiques fournissent, chaque année, des solutions pour un certain nombre de problèmes, et en crée de nouveaux types de services et produit qui était, quelques années auparavant, hors de portée.

Les tendances des architectures de bases de données :

Elles ont connu moins de changements que les systèmes et les supports de stockage. Toutefois, les générations d'architectures ont créé une révolution dans les systèmes de BD, en changeant la façon dont la donnée et la base sont structurées et exploitées. Parmi ces évolutions, nous citons les architectures centralisée [77, 71, 185, 78], distribuée [64, 142, 174, 49, 62, 32, 178, 114, 105, 31], et parallèle [151, 99, 31].

La recherche et le climat commercial :

Les compagnies ayant traditionnellement soutenu considérablement les recherches fondamentales, ont dû réduire l'effort de recherche, car les marges de profits sur certains services (e.g. certains moyens de télécommunication) ou produits (e.g. certains supports de stockage) ont diminué. À la place, la recherche a été réorientée vers des projets à court terme, destinés non pas aux prototypes, mais comme un travail frontalier destiné au marché. D'autre part, plusieurs indices montrent que la recherche en base de données est considérée positivement par le gouvernement et l'industrie. Lindberg *et al.* montrent que les BD prennent une place prépondérante dans les systèmes d'information dans l'avenir [141]. Les entreprises ont déjà commencé à mettre les systèmes de BD au centre de leurs préoccupations.

Le grand flux d'information à portée de main : (*The Information Superhighway Just Rolled Through Your Living Room*)

Ceci signifie que le Web, qui était *"sous-estimées"* auparavant, a pris part d'une grande utilisation de données. Le nombre des utilisateurs est de plus en plus croissant, et les données échangées sur le Web atteignent des volumes vertigineux. C'est le cas des messageries instantanées et des réseaux sociaux de nos jours [100, 139, 50].

2.1.3 Traitement des requêtes

Les utilisateurs peuvent interroger ou modifier les données existantes. Ces opérations sont dites *requêtes*, et nécessitent d'être exprimées dans un langage de manipulation de données (LMD). Le langage SQL est devenu le langage le plus utilisé [180]. La manipulation des données comporte à la fois : la réponse aux requêtes et le traitement des transactions.

La réponse aux requêtes nécessite l'analyse et l'optimisation de chaque requête par le compilateur.

Le plan d'exécution résultant sera traité par le SGBD comme une série d'opérations, afin de fournir la réponse finale. Le moteur d'exécution établit l'ensemble des opérations en effectuant les lectures et écritures requises des données dans la base.

Le traitement de transactions concerne les ensembles de requêtes et les actions du LMD (e.g. SQL), qui doivent s'exécuter comme unité. L'effet de chaque transaction doit être préservé même en cas d'échec du système. Ce type de traitement est répandu dans les traitements bancaires par exemple, où la totalité des transactions doivent prendre effet sur le système [58].

La figure 2.1 illustre l'architecture générale d'un système de gestion de base de données typique [101]. Cette architecture montre les principales composantes d'un SGBD, qui sont :

- Le compilateur : il constitue l'interface entre l'utilisateur et le moteur d'exécution. Il permet de vérifier la syntaxe des requêtes avant de les diriger vers le moteur d'exécution.
- Le moteur d'exécution : il permet de répondre aux requêtes en appelant les gestionnaires de stockage et de buffer pour le chargement des données requises.
- Le gestionnaire de buffer : il gère la mémoire buffer par l'allocation et le remplacement des données, des méta-données et d'autres structures auxiliaires.
- Le gestionnaire de transactions : il est responsable de l'exécution des transactions tout en assurant la cohérence des données en cas d'échec du système.
- Le gestionnaire de stockage : il permet d'effectuer les lectures et écritures sur les données persistantes sur les supports de stockages, dans le but de répondre aux requêtes traitées par le moteur d'exécution.

Le traitement d'une requête suit le chemin le plus à gauche de cette architecture pour l'obtention des données voulues sur la mémoire secondaire. Toutefois, le gestionnaire de buffer peut aiguiller le processus de traitement de la requête vers le buffer, si des données pertinentes y sont logées au moment du traitement.

2.2 Les entrepôts de données relationnels

Dans cette section, nous présentons les entrepôts de données comme l'un des types de BD modernes. Nous détaillons ensuite ses différents modèles, ainsi que les types de requêtes s'exécutant sur ces systèmes.

2.2.1 Entrepôts de données

Un entrepôt de données tel que le définit Bill Inmon [116] est une collection de données orientées sujet, non volatiles, intégrées, historisées et utilisées pour supporter un processus d'aide à la décision. Selon Ralph Kimball [131], un entrepôt de données doit être conçu pour être compréhensible (par l'analyseur). Dans ce but, Kimball a proposé la méthodologie de modélisation dimensionnelle (*Dimentional Modeling* ou *Kimball Modeling* en anglais), qui est devenue par la suite un standard dans le domaine décisionnel [132].

Parmi les principales caractéristiques d'un entrepôt de données, son volume important et ses requêtes complexes destinées aux applications d'analyse. Ces dernières rendent le temps de réponse élevé, ce qui ne convient pas aux décideurs qui demandent un temps raisonnable pour pouvoir améliorer les perfor-

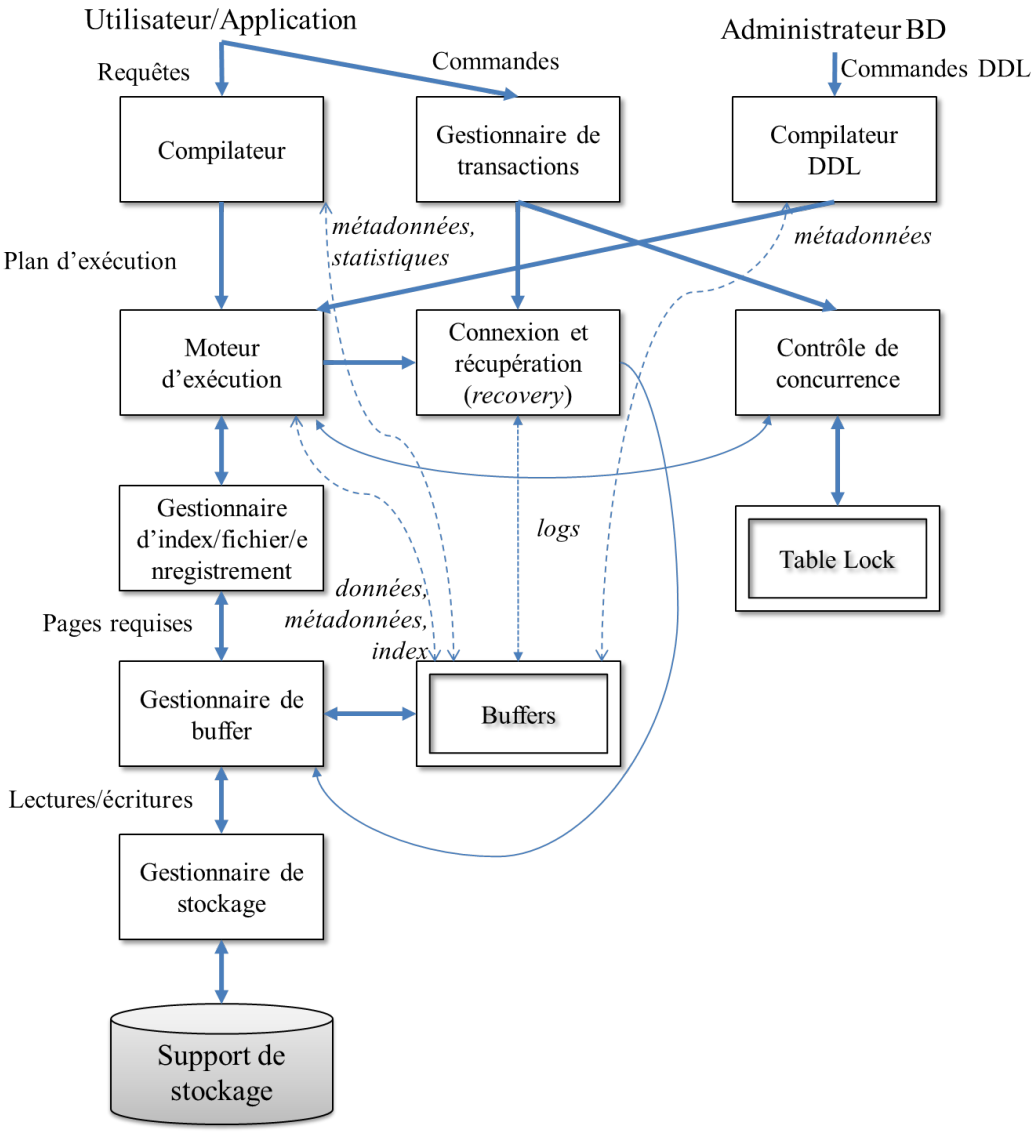

FIGURE 2.1 – Architecture générale d'un SGBD

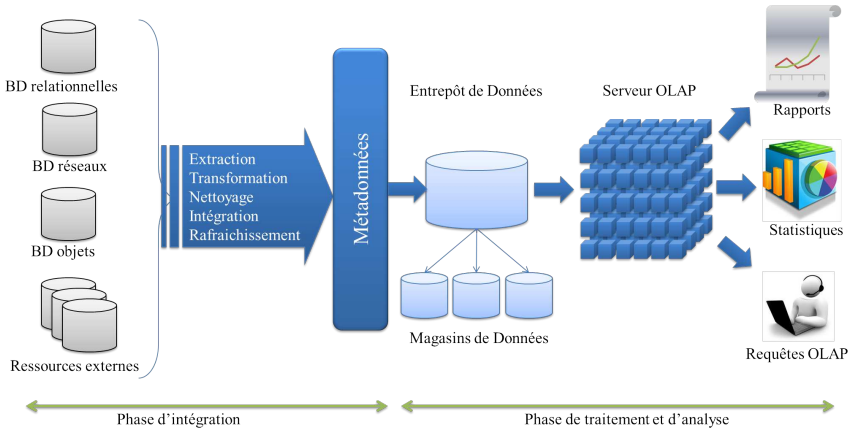

FIGURE 2.2 – Architecture d'un entrepôt de données

mances décisionnelles de l'entreprise.

Une requête peut être exécutée sous différents plans d'exécutions possibles et avec plusieurs structures d'optimisation sans altérer le résultat final de celle-ci, seule la performance change entre un plan et un autre. En partant de ce principe, et pour permettre d'assurer un temps de réponse réduit, les administrateurs (ou les concepteurs) sont amenés à faire une bonne conception physique de l'entrepôt.

2.2.2 Modèles d'Entrepôts de Données

Le but principal d'un entrepôt de données est d'historiser les données pour des fins d'analyse et de prise de décisions nécessaires pour l'entreprise. Les données stockées dans un entrepôt proviennent de plusieurs sources hétérogènes [41] : bases de données classiques, bases de données objets, bases de données réseaux ou d'autres sources externes.

La construction d'un entrepôt de données comporte deux phases : la phase d'intégration, et la phase d'analyse. Dans la phase d'intégration, les données sont collectées de plusieurs sources hétérogènes pour être nettoyées et intégrées afin de construire l'entrepôt. Dans la phase d'analyse, l'entrepôt conçu est exploité pour répondre aux requêtes des décideurs. Ces requêtes permettent d'effectuer des analyses sur les données afin de permettre la prise de décision (cf. figure 2.2).

Nous établissons une comparaison entre une base de données classique et un entrepôt de données comme le montre le tableau 2.1. Cette comparaison révèle les aspects analytiques et les besoins en performance liés aux entrepôts.

Selon Ralph Kimball [132], la représentation multidimensionnelle est la plus adaptée pour le processus d'analyse. Un tel modèle permet d'analyser les données sous plusieurs axes, et d'y faciliter l'accès contrairement au cas normalisé souvent employé dans les bases de données classiques. Ce modèle est constitué de deux types de tables :

– La table des faits : c'est une table contenant les différentes mesures requises dans la phase d'analyse. Par exemple, une table des faits *Ventes* comporte plusieurs mesures comme *Quantité*, *Montant* ou encore *Coût_unitaire* (cf. figure 2.4).

Caractéristiques	Base de Données	Entrepôt de Données
Opération	Gestion courante	Analyse et aide à la décision
Modèle	Souvent Entité-Relation	Schéma en étoile, flocon de neige ou en constellation
Normalisation	Fréquente	Rare
Données	Courantes, brutes	Historiques, agrégées
Mise à jour	Immédiate	Différée
Perception	Bidimensionnelle	Multidimensionnelle
Requêtes	Lecture et écriture	Lecture et rafraîchissement
Volume	Des Giga Octets	Jusqu'à quelques Téra Octets

TABLE 2.1 – Comparaison entre les bases de données classiques et les entrepôts de données

Temps / Ville / Produit	Trimestre1			Trimestre2			Trimestre3			Trimestre4			Total		
	Po	Pa	L	Po	Pa	L	Po	Pa	L	Po	Pa	L	Po	Pa	L
Tablette	22	68	60	18	72	34	22	62	35	33	32	62	95	234	191
Smart phone	88	120	90	80	120	90	56	175	59	81	57	39	305	472	278
Clavier	6	49	30	16	42	20	34	43	24	18	16	59	74	130	133
Total	116	237	180	114	234	144	112	280	118	132	105	160	474	836	602

Po: Poitiers; Pa: Paris; L: Lyon

FIGURE 2.3 – Représentation MOLAP par un tableau Multidimensionnel

– Les tables de dimension : c'est un ensemble de tables contenant les axes à considérer dans la phase d'analyse. Par exemple, les dimensions *Temps*, *Produit* et *Client* de la figure 2.4.

Un entrepôt de données peut être modélisé de deux manières différentes : (1) le modèle *multidimensionnel* MOLAP (*Multidimensional OnLine Analytical Processing*), où un tableau multidimensionnel à *n* dimensions est utilisé pour représenter les données (cf. figure 2.3), et (2) le modèle *Relationnel* ROLAP (*Relational OnLine Analytical Processing*), qui offre plusieurs types de schémas plus adaptés aux besoins multidimensionnels, analytiques et décisionnels comparé au modèle normalisé en 3-ième Forme Normale (3FN).

Le modèle ROLAP est le plus utilisé pour modéliser un entrepôt de données. Dans ce modèle, si l'entrepôt est représenté par une table des faits et plusieurs dimensions, le schéma obtenu est dit *schéma en étoile* (cf. figure 2.4). Cependant, dans certains cas, les dimensions sont munies d'hiérarchies permettant de passer d'un niveau moins détaillé au plus détaillé. Par exemple, la dimension *Temps* contient l'attribut *année* qui peut être détaillé par les valeurs de l'attributs *mois*, qui lui-même peut être détaillé par les valeurs de l'attribut *jour* : année ⇒ mois ⇒ jour.

Dans le cas où les dimensions comportent des hiérarchies, le schéma est dit en *flocon de neige* comme le montre l'exemple dans la figure 2.5. Un autre type de schéma existe mais qui est moins répandu que les deux premiers : c'est le schéma en constellation. Dans ce type de schéma, il existe plusieurs tables des faits qui ne sont pas concernées par le même ensemble de dimension [52]. Parmi les schémas que nous venons de présenter, le modèle en étoile est le plus répandu dans les entrepôts de données.

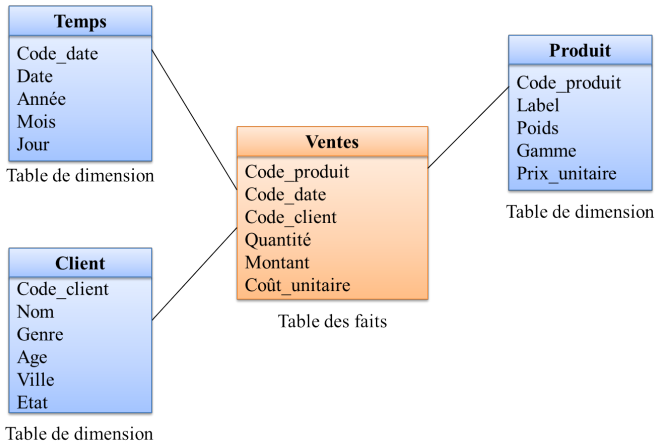

FIGURE 2.4 – Exemple d'un entrepôt de données modélisé par un schéma en étoile

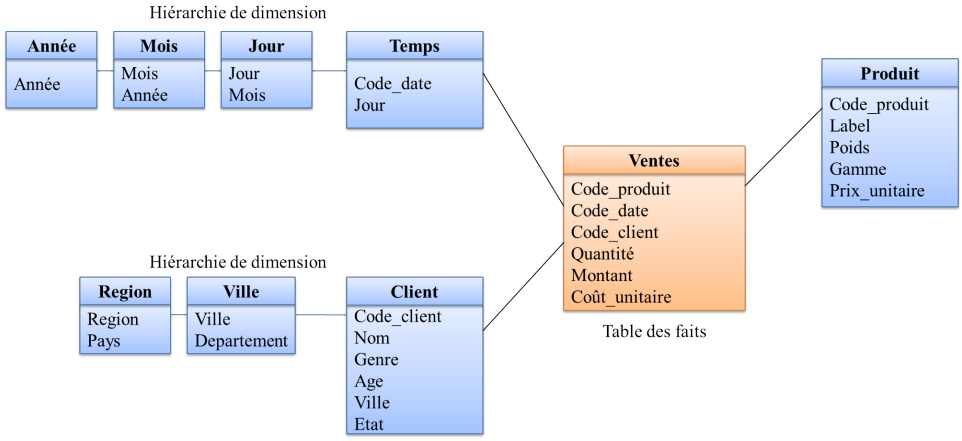

FIGURE 2.5 – Exemple d'un entrepôt de données modélisé par un schéma en flocon de neige

Abordons maintenant les différents types de requêtes s'exécutant sur ces schémas. Dans la littérature, deux types de requêtes sont distingués : (1) les requêtes transactionnelles OLTP (*OnLine Transactional Processing*), qui opèrent sur les bases de données classiques, et (2) les requêtes analytiques OLAP (*On-Line Analytical Processing*) qui opèrent sur les entrepôts de données. Dans ce dernier type, les requêtes sont principalement utilisées pour l'interrogation des données car la mise à jour dans les entrepôts de données n'est pas courante et est effectuée uniquement pour alimenter les tables [52].

2.2.3 La charge de requêtes de jointure en étoile

La charge de requêtes est un élément central dans les systèmes de bases de données car elle exprime les besoins des utilisateurs (ou administrateurs). Dans les entrepôts de données, les requêtes OLAP sont employées afin d'extraire les données requises pour l'analyse et la prise de décision. Elles s'intéressent donc à l'interrogation des données et non pas à l'écriture de celles-ci. Sur l'entrepôt de données représenté par un schéma en étoile, les requêtes employées sont les requêtes de jointures en étoile. Leur principale caractéristique est la multitude de jointures qui passent toutes par la table des faits car aucun lien direct n'existe entre les tables de dimension. Dans ces requêtes, plusieurs sélections sont effectuées sur les dimensions, suivies par des jointures avec la table des faits afin de filtrer ses instances. Plusieurs agrégats sont effectués également afin de permettre d'analyser les faits selon les mesures inclues dans le schéma.

La requête de jointure en étoile suit la forme générique suivante :

```
Select [att1, att2, .. attn ] , AGR1, AGR2, .. AGRm
From F, D1, .. D_k
Where clé1=F.clé1 and clé2=F.clé2 and ... clék=F.clék
and P1 and ... Pp
[Group by att1, att2, .. attn ]
[Order by att1, att2, .. attn ][Dsc][Asc]
```

Dans cette requête, les éléments {att1, att2, ... attn} représentent les attributs, les opérations {AGR1, AGR2, ... AGRm} représentent les agrégats (SUM, AVG, MIN, MAX, etc.) et les éléments {P1, P2 ... Pp} représentent les prédicats de sélection sur les dimensions.

Chaque requête admet un ensemble de plans d'exécution possibles, qui donnent tous le même résultat mais avec des performances variables. Le problème de génération du plan d'exécution optimal est $\mathcal{NP}$-complet [112, 164, 118, 70]. Il existe un nombre de règles permettant d'améliorer les performances des plans dites *Rule-Based Approaches*. Parmi ces règles, il y a la descente des sélections (*Push Down Selections*) qui permet de réduire la taille des relations et des résultats intermédiaires manipulés.

En 1988, Timos Sellis a souligné l'importance de considérer l'interaction entre les requêtes lors de l'optimisation d'une charge [176]. Suite à ce travail, la conception physique a pris une nouvelle tournure, où l'optimisation ne considère plus les requêtes isolées, mais la totalité de la charge. Cette approche est connue sous le nom de l'optimisation multi-requêtes, ou *Multi-Query Optimization* (MQO), et a été intégré dans une grande partie des travaux de la conception physique. Pour ressortir l'interaction des requêtes, une nouvelle représentation a été proposée par Yang *et al.* [199] afin d'identifier les sous-

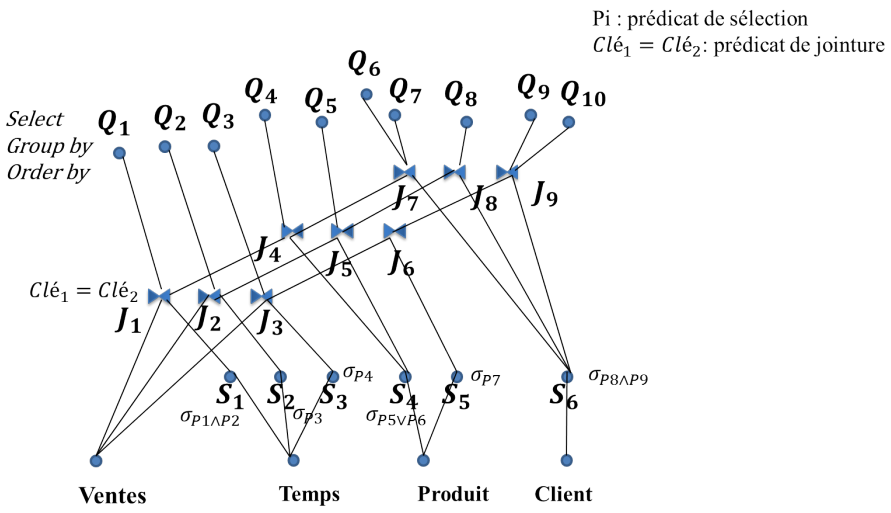

FIGURE 2.6 – Exemple de Multi-View Processing Plan simplifié

expressions partagées entre les requêtes de la même charge. Cette représentation est un graphe obtenu par la fusion des plans de toutes les requêtes, appelé : *Multi-View Processing Plan* (MVPP).

Exemple. Soit une charge de 10 requêtes de jointure en étoile. Chaque requête peut être exécutée suivant un plan d'exécution établi par l'optimiseur. Supposons que pour chaque requête, l'optimiseur fait descendre les sélections en bas du plan, et laisse l'ensemble d'agrégats et de projections à la fin du traitement de chacune. La fusion des plans obtenus pour la charge globale fournit le MVPP de la figure 2.6 contentant un ensemble de *nœuds* tels que :

- Les nœuds feuilles du graphe sont les tables du schéma $\{Ventes, Temps, Produit, Client\}$;
- En faisant descendre les sélections pour réduire la taille des résultats intermédiaires, nous obtenons un niveau contenant les sélections au-dessus des tables : $\{S_1, S_2 \ldots S_8\}$;
- Les nœuds binaires représentent les opérations de jointure $\{J_1, J_2 \ldots J_8\}$;
- Les nœuds finaux unaires représentent les traitements effectués sur les résultats finaux (agrégats, group by, order by et projection) [8] $\{gop_1, gop_2 \ldots gop_{10}\}$.

2.3 Les structures d'optimisation

Il existe une large panoplie de structures d'optimisation supportées par les SGBD. Chacune de ces structures est liée à un problème d'optimisation $\mathcal{NP}$-Complet [27, 54, 40]. Ainsi, pour chaque structure d'optimisation, il existe un ensemble d'algorithmes et d'approches proposés dans la littérature.

Une structure d'optimisation SO_i choisie pour la conception physique doit être supportée par le SGBD hôte. La structure possède une implémentation spécifique sur le support de stockage. Selon le type d'implémentation, nous distinguons deux classes de structures d'optimisation : les structures redon-

8. pour des raisons de simplification, nous supposons que les opérations d'agrégation et de tri s'effectuent sur le résultat final.

dantes, et les structures non-redondantes. Dans la section qui suit, nous allons détailler les deux types avec quelques exemples de chaque classe.

2.3.1 Classification des structures d'optimisation

Dans la littérature, les structures d'optimisation existantes sont classées selon la redondance liée à leur implémentation sur le support physique [52]. Ainsi, si une structure n'engendre pas un coût de stockage ou de maintenance supplémentaire, alors la structure est dite non-redondante. Dans le cas contraire, elle est dite redondante.

2.3.1.1 Structures d'optimisation redondantes Dans cette classe, nous citons la fragmentation verticale [152, 98], la fragmentation mixte [153, 202], les vues matérialisées [17, 199], les index [191, 156, 119, 171] et la gestion du buffer [81, 93, 187]. Dans toutes ces techniques, un coût de stockage supplémentaire est considéré afin d'implémenter les structures sur les données existantes. De plus, la mise à jour des données (insertion, suppression ou modification) influe sur ces structures et engendre un coût de maintenance supplémentaire.

2.3.1.2 Structures d'optimisation non-redondantes Dans cette classe, nous trouvons la fragmentation horizontale [60, 37, 144], le traitement parallèle [151] et l'ordonnancement des requêtes [11, 143] . Contrairement aux structures redondantes, celles-ci ne nécessitent aucun coût de stockage supplémentaire, car elles portent sur la répartition des données ou des requêtes.

2.3.2 Formalisation classique

Le problème de la conception physique est formalisé dans la littérature de la façon suivante [23] :

Étant donnés les trois entrées suivantes :
- Une charge de m requêtes $Q = \{Q_1, Q_2, \dots Q_m\}$;
- Un ensemble de l structures d'optimisation $\mathcal{SO} = \{SO_1, SO_2, \dots SO_l\}$ supportées par le SGBD;
- Un ensemble de contraintes liées à $\mathcal{SO}$: $Cont = \{Cont_1, Cont_2, \dots Cont_l\}$ où chaque contrainte $Cont_i$ $(0 \leq i \leq l)$ est associée à une structure d'optimisation SO_i [9].

Le problème de la conception physique ($\mathbb{PCP}$) consiste à sélectionner des schémas de structures d'optimisation afin de réduire le coût d'exécution de la charge de requêtes Q tout en vérifiant l'ensemble des contraintes définies.

2.3.3 Modes de sélection des structures d'optimisation

Parmi les instances possibles issues de la formalisation du $\mathbb{PCP}$, nous distinguons deux modes de sélection des structures d'optimisation [23] :
- **Mode isolé :** où l'administrateur sélectionne une seule structure d'optimisation pour satisfaire sa charge de requêtes Q. Plus formellement : $\|SO\| = 1$

9. Pour simplifier la formalisation, nous supposons que chaque structure a une seule contrainte

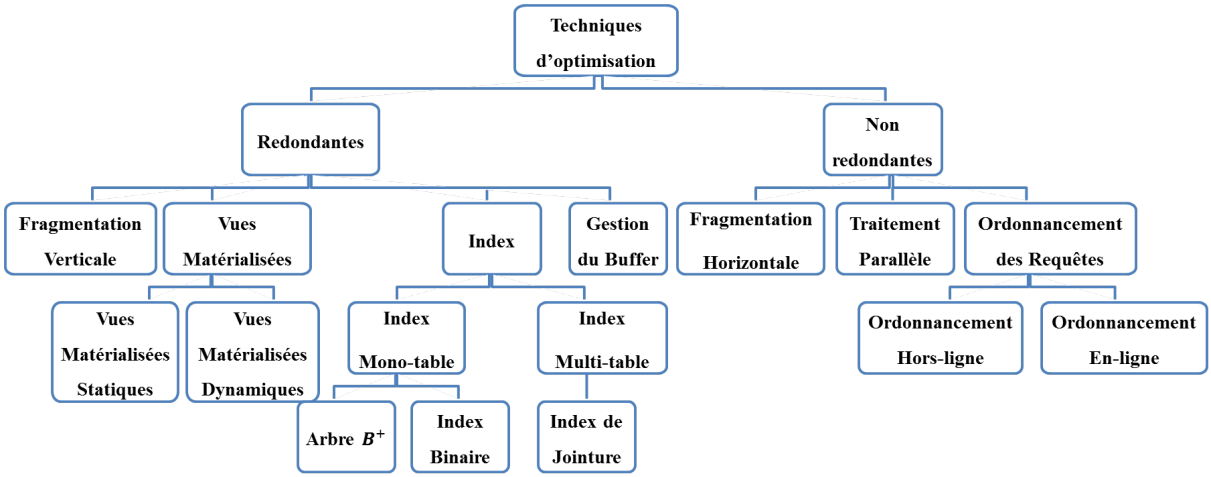

FIGURE 2.7 – Classification des techniques d'optimisation

– **Mode multiple :** où l'administrateur sélectionne plusieurs structures d'optimisation à la fois pour satisfaire sa charge. Plus formellement : $||SO|| > 1$

3 Sélection isolée des structures d'optimisation

Dans cette section, nous décrivons la sélection isolée en prenant le cas d'étude de la fragmentation horizontale. Nous présentons le principe de cette technique ainsi que son évolution à travers les générations de BD. Nous donnons ensuite la démarche de formalisation classique du problème. Une étude détaillée des travaux existants est fournie, suivie d'une synthèse de ces travaux. Nous terminons cette section par une discussion sur les limites de la sélection isolée.

3.1 La sélection isolée

Ce type de sélection a été largement étudié et adopté dans les bases de données traditionnelles [60, 152, 121]. Plusieurs travaux se sont intéressés à la sélection isolée des index [79, 191, 96, 73, 192, 65, 39], de fragmentation horizontale [60, 36, 37, 144] et verticale [80, 152, 44], des vues matérialisées [110, 149], etc. En plus de ces travaux, les entrepôts de données ont été considérés dans la sélection des index de jointures binaires (IJB) et multi-tables [198, 87], les vues matérialisées [17, 199], la compression des IJB [197], etc. La plupart des techniques d'optimisation sont prouvées comme étant liées aux problèmes $\mathcal{NP}$-complets [27, 54, 40].

Dans la section suivante, nous analysons la sélection isolée à travers un exemple. Pour cela, nous prenons le cas d'une des structures d'optimisations les plus répandues dans les EDR. Il s'agit de la fragmentation horizontale.

3.2 Cas d'étude : Fragmentation Horizontale

Nous proposons d'instancier le mode de sélection isolée en prenant le cas d'optimisation par la fragmentation horizontale. Dans un premier temps, nous donnons l'ensemble des définitions et des principes liées à cette technique. Nous étudions également son évolution, suivie par la démarche adoptée pour l'optimisation sur les entrepôts de données. Nous détaillons ensuite les différents travaux réalisés sur la fragmentation horizontale, et nous terminons par synthétiser l'ensemble de ces travaux.

3.2.1 La fragmentation horizontale : principe et définitions

La fragmentation horizontale (FH) est une technique d'optimisation non redondante qui consiste à découper une table en plusieurs sous-ensembles disjoints de tuples. Cette technique vise à optimiser les opérations de sélection en réduisant ainsi la taille des résultats intermédiaires qui sont souvent coûteuses dans les bases de données volumineuses. Dans les entrepôts de données relationnels, il existe deux types de fragmentation horizontale : la Fragmentation Horizontale Primaire (FHP) et la Fragmentation Horizontale Dérivée (FHD).

La fragmentation primaire : Le processus de partitionnement sélectionne les tables de dimension candidates afin d'appliquer la FHP sur celles-ci. La FHP consiste à découper une table en fonction de ses propres attributs. Ce type de partitionnement permet d'optimiser les sélections des requêtes de jointure en étoile ce qui réduira le coût global de la charge en conséquence.

Exemple. Soit un schéma en étoile contenant une table des faits *Ventes* et trois tables de dimension : *Client, Produit* et *Temps*. Si la table Client est candidate au partitionnement, elle peut être découpée par la FHP sur l'attribut *genre* de la façon suivante (cf. figure 2.8) :

- $Client_F = \sigma_{\text{genre = féminin}}(Client)$
- $Client_M = \sigma_{\text{genre = masculin}}(Client)$

L'exécution d'une requête impliquant les ventes féminines sera effectuée par l'optimiseur de la façon suivante :

```
SELECT Count(*)
FROM Client PARTITION(ClientF) C, Ventes V
WHERE V.CID=C.CID
```

L'opération de sélection a été optimisée par l'accès à la partition Client.

La fragmentation dérivée : Dans les EDR, la fragmentation horizontale doit se propager dans le schéma en étoile. La FHP ne suffit pas pour partitionner l'entrepôt, c'est pour cela que la fragmentation dérivée est appliquée sur la table des faits. La FHD consiste à fragmenter une table en fonction des attributs d'une autre table. Ce type de partitionnement permet d'optimiser les jointures.

Exemple. Dans le schéma de l'exemple précédent, la table des faits *Ventes* peut être découpée selon les fragments de la table *Client* pour obtenir les ventes féminines et les ventes masculines séparément (cf. figure 2.9) :

- $Ventes_F = (Ventes \rhd\!\lhd Client_F)$

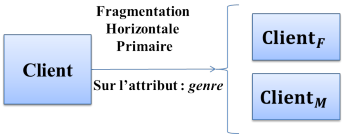

FIGURE 2.8 – Principe de la Fragmentation Horizontale Primaire sur une table de dimension

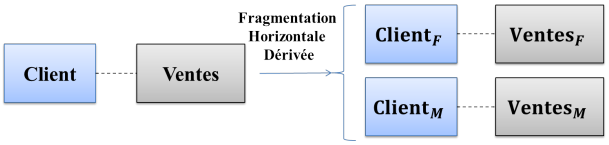

FIGURE 2.9 – Principe de la Fragmentation Horizontale Dérivée sur la table des faits

– $Ventes_M = (Ventes \ltimes Client_M)$

Par conséquent, la recherche des ventes féminines sera optimisée de la façon suivante :

```
SELECT Count(*)
FROM Ventes PARTITION(VentesF)
```

Car la sélection et la jointure sont déjà établies dans la FHD. D'autre part, la recherche des ventes effectuées par les clients parisiens sera exécutée par l'optimiseur sur les deux partitions de la table des faits obtenues, de la façon suivante :

```
SELECT Count(*)
FROM Client PARTITION(ClientF) C, Ventes PARTITION(VentesF) V
WHERE V.CID=C.CID AND C.Ville='Paris'
UNION ALL
SELECT Count(*)
FROM Client PARTITION(ClientM) C, Ventes PARTITION(VentesM) V
WHERE V.CID=C.CID AND C.Ville='Paris'
```

Le schéma en étoile après la fragmentation horizontale avec ses deux types sera éclaté en plusieurs sous-schémas en étoile. Le nombre de sous-schémas est égal au nombre de fragments des faits N obtenus par la dérivation de la FHP sur les tables de dimension. Il est obtenu par le produit des fragments de dimension autour de la table des faits.

$$N = \prod_{i=1}^{d} Frag_i$$

Où $Frag_i$ représente le nombre de fragments de la table de dimension $D_i \in \{D_1, D_2, \ldots D_d\}$.

Dans l'exemple précèdent, la FH engendre deux fragments de la table *Client* et ainsi deux fragments de *Ventes*. Par conséquent, le nombre de sous-schémas est égal à deux (cf. figure 2.10).

Exemple. La FHP de la table *Produit* selon la catégorie des produits en vente, peut engendrer trois fragments de la dimension *Produit* tels que :

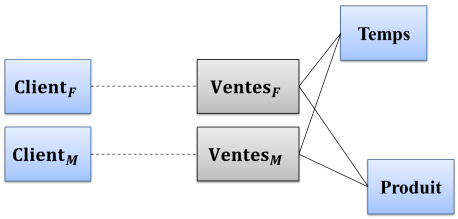

FIGURE 2.10 – Schéma d'un entrepôt de données partitionné par FHP et FHD

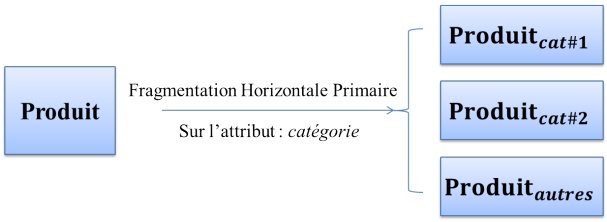

FIGURE 2.11 – Fragmentation horizontale primaire sur la table de dimension Produit

- $Produit_1 = \sigma_{\text{categorie = 'cat\#1'}}(Produit)$
- $Produit_2 = \sigma_{\text{categorie = 'cat\#2'}}(Produit)$
- $Produit_3 = \sigma_{\text{categorie}\notin\{'cat\#1','cat\#2'\}}(Produit)$

Comme les deux catégories *cat#1* et *cat#2* ne sont pas les seules valeurs de l'attribut catégorie, alors un autre fragment est rajouté contenant les autres valeurs possibles (cf. figure 2.11). Ceci permet d'assurer la complétude qui est une des règles de correction de la fragmentation que nous discutons dans la Section 3.2.2.

Les trois fragments de la table produit engendrent six fragments de faits, et ainsi six sous-schémas en étoile à partir de l'ancien schéma. Le résultat de la fragmentation dérivée est illustré dans la figure 2.12. La table *Ventes* sera partitionnée une deuxième fois pour donner les deux fragments suivants :

- $Ventes_{F,cat\#1} = (Ventes_F \ltimes Produit_1)$
- $Ventes_{F,cat\#2} = (Ventes_F \ltimes Produit_2)$
- $Ventes_{F,autres} = (Ventes_F \ltimes Produit_3)$
- $Ventes_{M,cat\#1} = (Ventes_M \ltimes Produit_1)$
- $Ventes_{M,cat\#2} = (Ventes_M \ltimes Produit_2)$
- $Ventes_{M,autres} = (Ventes_M \ltimes Produit_3)$

Le schéma final de l'entrepôt obtenu par la fragmentation horizontale est présenté dans la figure 2.13.

3.2.2 Règles de correction

Le processus de fragmentation horizontale s'effectue suivant des règles de correction pour que la fragmentation d'une table soit valide. Il existe trois règles pour vérifier la validité d'un schéma de fragmentation : la complétude, la disjonction et la reconstruction [52].

Soit T une table fragmentée en k fragments $\{T_1, T_2 \ldots T_k\}$.

1. Complétude : toute instance i de la table initiale T doit appartenir au moins à un de ses fragments T_s. Cette règle garantit que toutes les instances existent, et qu'aucune instance n'est perdue dans

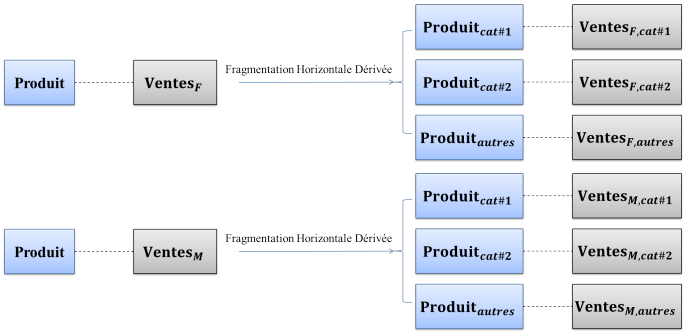

FIGURE 2.12 – Fragmentation horizontale dérivée sur la table Ventes

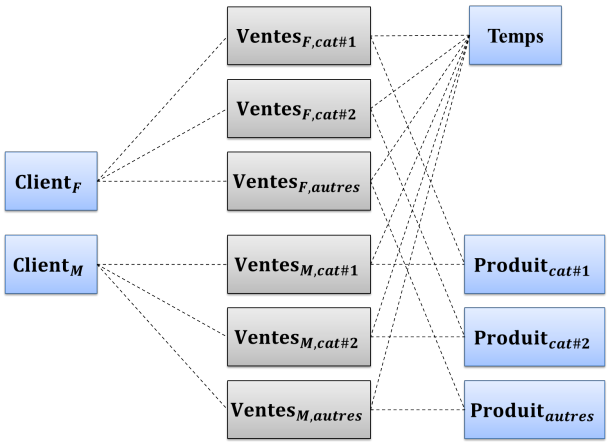

FIGURE 2.13 – Le schéma de Fragmentation Horizontale final

le schéma fragmenté.

2. Disjonction : toute instance i appartenant à un fragment T_k ne doit pas appartenir à un autre fragment T_l. Autrement dit, l'intersection entre les fragments de la table T donne toujours l'ensemble vide. C'est grâce à cette règle que la fragmentation horizontale appartient à la classe des techniques d'optimisation non redondantes.
3. Reconstruction : après la fragmentation d'une table T en k fragments $\{T_1, T_2 \dots T_k\}$, elle doit pouvoir être reconstruite à partir de ses fragments en utilisant une opération donnée (dans les types de fragmentations actuels, cette opération est l'union) : $T = \cup_{j=1}^{k} T_k$. Cette règle garantit que la fragmentation soit réversible.

3.2.3 Évolution la fragmentation horizontale

La fragmentation horizontale a pris une place importante dans l'optimisation dans les systèmes de bases de données. Vers la fin des années 70, les chercheurs ont commencé par l'adopter dans la conception logique des bases de données pour passer par la suite au niveau physique. Les différents types de bases de données ont été adressés : bases de données centralisées [60, 37, 170, 28, 29, 144], distribuées [61, 160, 92, 121, 140, 161, 154] et parallèles [151, 99, 46, 31, 24]. Avec l'apparition des entrepôts de données, le besoins de la fragmentation horizontale s'est accru. La fragmentation horizontale a été l'objet de plusieurs travaux, en combinaison avec d'autres techniques comme les vues matérialisées [43], les index [84, 22, 59] ou encore la fragmentation verticale [153, 170, 59].

En parallèle à ces approches et travaux de recherches, l'industrie a grandement contribué dans l'évolution de la fragmentation horizontale. Les SGBD commerciaux se sont intéressés à cette technique et l'ont adopté en intégrant des commandes DDL et des primitives permettant de partitionner les objets au niveau physique. Parmi ces SGBD, nous citons Oracle, SQL Server et TERADATA. Au fil des temps, deux modes de partitionnement sont apparus dans les SGBD : le mode simple et le mode composé.

a) Le mode simple : comporte trois types de partitionnement supportés par les SGBD : partitionnement par liste (LIST), par hachage (HASH) ou par intervalle (RANGE).

Le mode simple est défini par une clé de fragmentation C et un ensemble de valeurs $V\{v_1, v_2, \dots v_n\}$ de cette clé. La clé C peut être un ou plusieurs attributs de la table. Pour simplifier la compréhension de la définition, nous supposons que C comporte un seul attribut. Les éléments de V sont une ou plusieurs valeurs du domaine de valeurs Dom_i de l'attribut a_i représenté par la clé C. Ainsi, l'ensemble V est une décomposition de Dom_i en plusieurs sous-domaines stables. Chaque sous-domaine stable représente une partition de la table. Lors d'une insertion d'une nouvelle instance, le système identifie la partition concernée en comparant la valeur de la clé C et les éléments de V définis [52].

Les trois types définis dépendent de la façon dont l'ensemble V est décomposé. Le type LIST consiste à énumérer les valeurs de chaque élément $v_j \in V$. Le type RANGE permet de définir les éléments de V comme étant des intervalles de valeurs. Pour le HASH, une fonction de hachage interne au système est employée. Pour ce type, la clé C et le nombre de partitions doivent être fournis pour établir la fonction de hachage.

b) Le mode composé : ce mode comporte une combinaison de tous les types de fragmentation en mode simple. Ainsi, on peut effectuer : RANGE-LIST, LIST-LIST, RANGE-HASH, etc.

c) Le mode *référence* pour la fragmentation dérivée : Ce mode permet d'effectuer une fragmentation horizontale sur une table selon les fragments d'une autre table. Ce qui est souvent le cas de la table des faits dans le schéma en étoile. Dans ce mode, la fragmentation peut être simple (LIST, RANGE, HASH) ou composée (RANGE-LIST, LIST-LIST, RANGE-HASH, etc.).

Pour illustrer les différents modes de partitionnement, nous considérons les exemples suivants :

Exemple. La création de la table *Produit* partitionnée en mode simple par un LIST sur l'attribut catégorie s'effectue de la façon suivante :

```
CREATE TABLE PRODUIT
(TID NUMBER, Categorie VARCHAR2(10), Label Number)
PARTITION BY LIST(Categorie)
(PARTITION Produit1 VALUES('cat#1'),
PARTITION Produit2 VALUES ('cat#2'),
PARTITION Produit3 VALUES (DEFAULT));
```

La création de la table *Client* partitionnée en mode composé, par un RANGE-LIST sur les attributs Age (RANGE) et Genre (LIST) s'effectue de la façon suivante :

```
CREATE TABLE CLIENT
(CID NUMBER, Nom Varchar2(20), Genre CHAR, Age Number)
PARTITION BY RANGE (Age)
SUBPARTITION BY LIST (Genre)
SUBPARTITION TEMPLATE (SUBPARTITION Female VALUES ('F'),
SUBPARTITION Male VALUES ('M'))
(PARTITION Client_0_60 VALUES LESS THAN (61),
PARTITION Client_60_120 VALUES LESS THAN (MAXVALUE));
```

La création de la table *Vente* partitionnée par une FHD en mode référence, selon les fragments de la table *Client* s'effectue de la façon suivante :

```
CREATE TABLE VENTES
(CID number(6), Date DATE , Montant Number(8,2)
CONSTRAINT Client_fk FOREIGN KEY (CID) REFERENCES Client(CID))
PARTITION BY REFERENCE(Client_fk);
```

3.2.4 Démarche classique de la fragmentation horizontale

La fragmentation horizontale a évolué avec l'apparition de nouvelles générations de bases de données. La démarche a également évolué pour s'adapter aux nouveaux problèmes rencontrés à chaque environnement. Dans les entrepôts de données, la FH est effectuée selon la démarche suivante [52] :

1. Phase de préparation de la fragmentation :
   - Extraction de l'ensemble des prédicats de sélection [60, 161];
   - Identification des tables candidates pour participer à la FH;
   - Génération d'un ensemble de prédicats complet et minimal [160];
   - Découpage du domaine de chaque attribut en sous-domaines;

2. Phase de sélection d'un schéma de Fragmentation Horizontale :
   - La génération des schémas de Fragmentation Horizontale;
   - L'évaluation de chaque schéma;
   - La sélection du meilleur schéma;
3. Phase de fragmentation des tables de dimension par une FH Primaire;
4. Phase de fragmentation de la table des faits par une FH Dérivée.

La phase de préparation de la fragmentation est primordiale car elle fournit l'ensemble des prédicats requis sur lesquels les tables vont être partitionnées. A partir de ces prédicats, la sélection du meilleur schéma est effectuée selon l'algorithme employé. Une fois ce schéma est retourné, la FH Primaire est déployée sur les tables de dimension pour être propagée par la suite sur la table des faits par une FH Dérivée.

Cependant, l'approche impose un traitement avancé des prédicats et ne peut être facilement assurée par l'administrateur. De plus, cette démarche ignore l'interaction entre les requêtes qui a un impact considérable sur le déroulement du processus de partitionnement [176, 38, 127].

3.2.5 Problème de sélection d'un schéma de fragmentation horizontale

Le problème de fragmentation horizontale a été formalisé dans la littérature selon la formalisation classique du problème de la conception physique [20, 51, 26].

Étant données les entrées suivantes :
- Un entrepôt de données ayant un schéma en étoile : une table des faits F et un ensemble de tables de dimension $D = \{D_1, D_2, \ldots, D_d\}$;
- Une charge de requêtes Q;

Et soit la contrainte de maintenance suivante :
- Un seuil W fixé par l'administrateur, limitant le nombre de sous-schémas permis;

Le problème de la fragmentation horizontale vise à obtenir un ensemble $D' \subseteq D$ de dimensions partitionnées par une FHP, et un ensemble de N fragments de faits $F_1, \ldots, F_N$ obtenus par une FHD, tels que le coût d'exécution de Q soit minimal, et la contrainte $N \leq W$ soit vérifiée.

3.2.6 Travaux sur la fragmentation horizontale

Comme nous l'avons présenté dans a Section 3.2.3, la fragmentation horizontale est l'une des techniques d'optimisation les plus largement adoptées dans la phase de conception physique. Cette technique a évolué parallèlement aux bases de données et aux supports de stockage, en commençant par les bases de données classiques jusqu'aux bases de données Cloud.

La fragmentation horizontale a la particularité d'être utilisée au niveau logique et physique. Au niveau logique, elle a été largement considérée comme une technique de conception de bases de données réparties. Avec le développement des applications décisionnelles, la FH est supportée par la majorité des SGBD. Contrairement aux autres structures d'optimisation (comme les index et les vues matérialisées), où la sélection des schémas d'optimisation se fait généralement lorsque la base de données est opérationnelle (ou créée), la sélection d'un schéma de fragmentation d'une base de données (ou un entrepôt

de données) se décide avant sa création. Cette situation rend sa sélection plus sensible que les autres structures. De plus, elle peut également être combinée avec d'autres structures d'optimisation comme les index, les vues matérialisées et le traitement parallèle [170, 201].

L'idée sous-jacente à la fragmentation est de pouvoir constituer des ensembles de données partielles dont les n-uplets (ou instances) ont des propriétés géographiques communes. Le mot géographique peut être perçu à des échelles différentes selon le contexte d'utilisation : centralisé, réparti, et parallèle. La fragmentation horizontale permet ainsi d'augmenter le parallélisme inter-requêtes et intra-requêtes.

Les travaux qui ont étudié la fragmentation horizontale proviennent de deux milieux différents : académique et industriel.

La FH est considérée comme une technique importante dans la conception physique [170]. Pour cette raison, les éditeurs de SGBD commerciaux (Oracle, DB2, SQL Server, Sybase, etc.) ou non commerciaux (PostgreSQL, MySql, etc.) s'intéressent de plus en plus à la fragmentation, en proposant des commandes au niveau du langage de définition de données (LDD) pour la supporter. Plusieurs modes de fragmentation ont été proposés (range, list, hash et composé). Récemment, Oracle 11g a fait évoluer la fragmentation horizontale en proposant plusieurs modes :

1. le partitionnement par une colonne virtuelle (virtual column partitioning) dans lequel, une table est fragmentée en utilisant un attribut virtuel, qui est défini par une expression utilisant un ou plusieurs attributs. Cette colonne est stockée seulement dans les méta-données.
2. Le partitionnement par référence (referential partitioning) : permet de fragmenter une table en utilisant une autre table (à condition qu'il y ait une relation de type père-fils entre les deux tables [60]. Étant donné que la fragmentation horizontale préserve le schéma logique des tables et des vues, ceci implique que toutes les opérations effectuées sur les objets d'origine sont aussi applicables sur leurs partitions.

Plusieurs opérations ont été définies pour manipuler des partitions. Nous citons l'opération de fusion de deux partitions (MERGE PARTITION), l'opération d'éclatement d'une partition en deux partitions (SPLIT PARTITION) et l'opération de conversion d'une partition en une table (EXCHANGE PARTITION).

Exemple. La fusion de deux fragments de la table Produit en une seule partition est effectuée par la fonction MERGE PARTITION de la façon suivante :

```
ALTER TABLE Produit
MERGE PARTITIONS Produit1, Produit2 INTO PARTITION Produit1_2;
```

L'éclatement d'une partition de la table Produit en deux fragments est effectué par la fonction SPLIT PARTITION de la façon suivante :

```
ALTER TABLE Produit
MERGE PARTITIONS Produit1, Produit2 INTO PARTITION Produit1_2;
ALTER TABLE Produit
   SPLIT PARTITION Produit4 VALUES ('cat#4', 'cat#5')
   INTO
    ( PARTITION Produit4,
      PARTITION Produit5
);
```

Au niveau académique, le problème de sélection de schéma FH a suscité beaucoup d'intérêt sur le plan développement d'algorithmes et de méthodologies. Après avoir exploré la littérature, nous proposons une classification des algorithmes existants en deux catégories : *Approches sans contrainte* et *Approches avec contrainte*. Les premiers travaux sur la FH se trouvent dans la première catégorie. Ces travaux sélectionnent un schéma de partitionnement d'une base de données sans prendre en considération le nombre de fragments générés. Nous identifions deux classes d'approches dans cette catégorie : *approches basées sur la génération des minterms* [60, 161] et *approches basées sur l'affinité* [121].

Les approches basées sur les minterms commencent par une table $\mathcal{T}$ et un ensemble de prédicats $\{p_1, \ldots, p_n\}$ des requêtes les plus fréquentes défini sur $\mathcal{T}$ et retourne un ensemble de fragments horizontaux. Les principales étapes de cette approche sont : (1) la génération des prédicats minterms $\mathcal{M} = \{m|m = \wedge(1 \leq k \leq n)P_k^*\}$, où P_k^* est soit p_k ou $\neg p_k$, (2) la simplification des minterms dans $\mathcal{M}$ et l'élimination de ceux inutiles et (3) la génération des fragments définis comme $\sigma_{m_i}(\mathcal{T})$ (σ est une opération de sélection). Cette approche est simple mais l'algorithme a une grande complexité. Pour n prédicats simples, l'algorithme génère 2^n minterms. Cette approche a été utilisée par Papadomanolakis et Ailamaki [162]. Afin de réduire cette complexité, une autre approche [121] a été proposée adaptant l'algorithme de partitionnement vertical de Navathe et Ra [152]. Les prédicats ayant une grande affinité sont regroupés ensemble. L'affinité entre deux prédicats p_i et p_j est calculée comme la somme des fréquences des requêtes y accédant simultanément. Chaque groupe fournit un fragment horizontal comme une conjonction de ses prédicats. Cet algorithme a une complexité réduite [122], mais il ne prend en considération que les fréquences d'accès pour la génération des fragments horizontaux et ignore d'autres paramètres comme la taille des tables et les facteurs de sélectivité de prédicats.

Les concepteurs utilisant les approches basées sur les *minterms* et l'*affinité*, ne connaissent pas le nombre de fragment au préalable. Ils doivent attendre la fin d'exécution de l'algorithme pour déterminer ce nombre. Ayant le besoin d'avoir un nombre de fragments raisonnable pour réduire les coûts de maintenance, de nouvelles approches basées sur la contrainte du *seuil* ont été proposées. Ce seuil représente le nombre maximal de fragments que l'administrateur de base de données (DBA) permet au système de générer. Le principal objectif des approches *avec contrainte* est de partitionner une table en N fragments tels que N soit inférieur ou égal au seuil fixé W pendant l'optimisation de la charge. Ainsi, en plus des ensembles de prédicats de sélection, ces approches nécessitent que l'administrateur fixe la valeur du seuil W. Deux classes principales existent dans cette catégorie : les méthodes *basées sur le coût* et les méthodes *basées sur les techniques du data mining*.

Les approches *basées sur le coût* commencent par un ensemble de schéma de partitionnement potentiel d'une table T et utilisent un modèle de coût comme métrique, i.e. le nombre des Entrées/Sorties (E/S) requises pour exécuter un ensemble de requêtes. Puis, elles calculent le coût sur chaque schéma [37]. Le schéma ayant le coût minimal est sélectionné comme solution finale. L'administrateur peut ainsi quantifier le gain obtenu par cette solution. Le Hill Climbing, Recuit Simulé et les Algorithmes Génétiques sont des exemples de cette catégorie.

Les approches *basées sur les techniques du Data Mining* ont été proposées dans le contexte des entrepôts de données XML [145]. Elles utilisent l'algorithme de *K-Means clustering* pour grouper les prédicats de sélection pouvant partitionner l'entrepôt. Le seuil W (nombre de fragments) est donné comme entrée pour l'algorithme *K-means* [86].

En nous penchant sur les travaux de fragmentation, nous avons identifié trois points : (1) la majorité

FIGURE 2.14 – Classification des travaux sur la fragmentation horizontale

des travaux est basée sur un ensemble de requêtes fréquentes ; (2) tous les attributs de sélection utilisés par les algorithmes de sélection ont la même probabilité d'être utilisés pour partitionner la base de données, et (3) les modèles de coût utilisés pour quantifier la qualité du schéma de fragmentation sont simples et estiment le coût de requêtes indépendamment de beaucoup de paramètres comme le contenu du buffer ou l'ordre des requêtes.

3.2.7 Synthèse des travaux de la fragmentation horizontale

La fragmentation horizontale est une technique omniprésente dans la conception physique des bases de données. Dans les différentes générations des systèmes de bases de données, depuis les BD traditionnelles jusqu'au BD Cloud, de multiples approches ont été proposées. Dans la première génération des travaux, des approches ont été proposées sans contrainte de maintenance : les approches basées sur les minterms [60, 161] et d'autres basées sur l'affinité des prédicats [121]. Dans la deuxième génération, la contrainte de maintenance a été introduite est deux nouvelles approches ont été proposées : les approches basées sur modèle de coût [37] et d'autres basées sur les techniques du Data Mining [145]. La figure 2.14 résume cette classification.

Cependant, vers la fin des années 80', une nouvelle caractéristique importante a été dévoilée dans le travail de Sellis [176]. Cette caractéristique est l'interaction entre les requêtes. Cette propriété typique dans les bases de données impose la considération de la totalité de la charge durant le processus d'optimisation. La majeure partie des techniques d'optimisation a considéré cette propriété [85, 184, 199, 111, 146, 186, 72, 196, 11, 143, 136]. Cependant, toutes les approches existantes pour la fragmentation horizontale ignorent l'interaction entre les requêtes. L'extraction des prédicats est effectuée sur des plans de requêtes isolées et toutes les requêtes auront la même probabilité de participer dans le processus de partitionnement.

D'autre part, les dernières générations de bases de données (comme les bases de données Cloud) sont confrontées à de nouvelles difficultés. Parmi ces difficultés, nous citons le volume croissant des données, le nombre important des requêtes et les changements fréquents de l'environnement. Cependant, la lourdeur des algorithmes existants (méta-heuristiques) et l'absence de l'aspect dynamique empêchent le passage à l'échelle dans de tels environnements.

Nous avons également analysé l'évolution des solutions proposées pour la fragmentation horizontale. Au départ, les solutions étaient lourdes car le processus énumère tous les prédicats de la charge pour la génération des minterms. Les approches dirigées par affinité permettent de contourner cette lourdeur par la proposition d'un algorithme simple. Toutefois, l'efficacité n'était pas satisfaisante. Les algorithmes

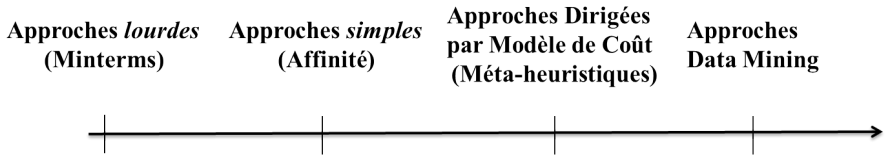

FIGURE 2.15 – Évolution des approches de résolution de la fragmentation horizontale

dirigés par Modèle de Coût ont été inspirés des optimiseurs, et ont permis d'atteindre de hauts niveaux de performance. Ces approches se basent sur des méta-heuristiques lourdes comme la génétique. Une nouvelle proposition a été initiée dans des travaux récents, qui se base sur les techniques du Data Mining. Toutefois, les approches dirigées pas Modèle de Coût reste les plus efficaces. La figure 2.15 résume cette évolution.

3.3 Limites de la sélection isolée

Le problème majeur du mode de sélection isolée est que chacune des techniques correspond à un profil particulier. Par exemple, la fragmentation horizontale est plus adaptée pour un profil donné de prédicat et de facteurs de sélectivité. D'un autre côté, les index s'avèrent plus adaptés pour d'autres types de prédicats et de facteurs de sélectivité [170]. En plus de cette limite, des travaux récents [170, 201], ont identifié l'existence d'interaction entre les structures d'optimisation. Comme la fragmentation horizontale et les index, ou la fragmentation horizontale et les vues matérialisées.

Ces limites liées à la sélection isolée ont motivé l'adoption d'un autre mode de sélection permettant de tirer profit de plusieurs structures d'optimisation à la fois, tout en exploitant leur interaction. Ce mode correspond à la sélection multiple.

4 Sélection multiple des structures d'optimisation

Dans cette section, nous étudions la sélection multiple en prenant comme exemple le cas de la gestion du buffer et l'ordonnancement des requêtes. Nous détaillons les deux techniques de façon isolée ainsi que leur combinaison. Nous terminons la section pas un bilan et une discussion sur les limites de la sélection multiple.

4.1 La sélection multiple

Dans quelques travaux récents, l'existence d'interaction entre les structures d'optimisation a été dévoilée [43, 201, 170]. Le travail de Zilio *et al.* a permis d'établir des dépendances entre les structures d'optimisation existantes dans la conception physique [201]. Deux types de dépendance ont été définis :

Soit deux structures d'optimisation SO_1 et SO_2 supportées par un SGBD.

– **Dépendance Forte :** si le changement d'une structure d'optimisation SO_2 entraîne le changement de SO_1, alors SO_1 *dépend fortement de* SO_2.

	Index	Vues Matérialisées	Fragmentation Horizontale
Index	/	Forte	Faible
Vues Matérialisées	Forte	/	Faible
Fragmentation Horizontale	Faible	Forte	/

TABLE 2.2 – Table des dépendances entre les structures d'optimisation

– **Dépendance Faible :** si le changement d'une structure d'optimisation SO_2 n'entraîne pas le changement de SO_1, alors SO_1 *dépend faiblement de* SO_2.

Zilio *et al.* [201] ont a montré que cette relation de dépendance n'est pas symétrique. Ainsi, entre deux structures d'optimisation SO_1 et SO_2, il peut y avoir trois types de liaisons :

– Forte-Forte : où SO_1 dépend fortement de SO_2, et vice versa;
– Forte-Faible : où $S0_1$ dépend fortement de SO_2, tandis que SO_2 dépend faiblement de SO_1;
– Faible-Faible : où SO_1 dépend faiblement de SO_2, et vice versa.

En se basant sur ce travail, Boukhalfa K.[52] a proposé trois implémentations différentes pour la sélection multiple :

– Indépendante : consiste à traiter chaque structure d'optimisation de manière isolée (rare).
– Conjointe : consiste à traiter simultanément les problèmes liés aux deux structures d'optimisation d'une dépendance de type : Forte-Forte [34, 35, 169]. Le problème majeur de ce genre d'implémentation est la complexité élevée liée à la jointure de plusieurs problèmes $\mathcal{NP}$-complets.
– Séquentielle : cette implémentation s'applique souvent sur des structures d'optimisation ayant une dépendance non mutuelle, autrement dit une dépendance de type Forte-Faible [53, 59, 52]. Elle permet de réduire la complexité de l'implémentation conjointe. Elle s'applique comme une succession de sélection isolée de façon à ce que les sorties d'un problème soient inclues dans les entrées du problème suivant. Dans cette implémentation, l'ordre de traitement des techniques d'optimisation est important [52].

4.2 Cas d'étude : Gestion du Buffer et Ordonnancement des Requêtes

Dans cette section, nous proposons une étude de cas pour la sélection multiple en considérant deux techniques : la gestion du buffer et l'ordonnancement des requêtes. Nous commençons par détailler le principe de chaque technique, et les motivations derrière leur combinaison. Nous présentons ensuite l'ensemble des travaux reliés à chaque technique de façon isolée, ainsi qu'à leur combinaison. Nous terminons la section par une synthèse des travaux existants.

4.2.1 La gestion du buffer

Le buffer [10] de bases des données contient des copies d'une partie des données existant sur une mémoire secondaire [101]. Ces copies sont stockées sous forme de pages et résultent d'un traitement de données par le SGBD. Elles peuvent être constituées des relations, des plans de requêtes ou des données

10. Dans ce travail, nous employons les mots cache et buffer indifféremment pour désigner les mémoires buffer de la base de données. Nous employons également le terme *cacher* pour désigner l'allocation du buffer pour un ensemble de données.

pré-calculées (sous relations, résultats de jointure ...). L'intérêt de mettre des données dans la mémoire cache est de pouvoir les réutiliser par d'autres requêtes en réduisant le nombre d'accès en mémoire secondaire où se logent les données de façon permanente (HDD, Flash, Cloud ...). La réutilisation de ces données permet d'augmenter la performance du système grâce aux caractéristiques physiques de la mémoire centrale qui rendent le coût de lecture et d'écriture négligeables par rapport au temps d'accès aux données sur le disque [101].

Le *Tuning* du buffer est une série de traitements permettant d'adapter les différents paramètres du buffer requis pour maintenir une bonne performance du système. Il décide de la Politique de Gestion du Buffer (PGB) à savoir l'allocation et le remplacement des pages dans le buffer. Il permet aussi de partitionner le buffer de manière équitable entre les serveurs et les utilisateurs dans le contexte d'allocation des ressources physiques. La base de données est divisée en pages de tailles égales (512 à 4096 octets) et le buffer est composé de blocs de pages de tailles égales. Le nombre de blocs est un paramètre interne au SGBD et reste constant durant une instance du système.

Une bonne gestion du buffer permet de réduire au mieux le nombre d'accès disque (e.g. les résultats de requêtes) et certains traitements du SGBD (e.g. le plan d'exécution d'une requête). La gestion du buffer s'effectue selon les étapes suivantes (cf. figure 2.1) :

- Une requête demande la donnée;
- Le gestionnaire parcours les pages du buffer à la recherche de cette donnée. S'il trouve la donnée, il s'agit d'un succès de cache (*cache hit* en anglais) et la donnée est transmise au moteur d'exécution. Dans le cas contraire, il s'agit d'un défaut de cache (*cache miss* en anglais) et le SGBD doit la charger de la mémoire secondaire.
- La donnée est traitée par le SGBD et stockée sur le buffer à la limite de l'espace disponible pour utilisation ultérieure.

Formalisation du problème de la Gestion du Buffer : Nous formalisons le problème de gestion du buffer de la façon suivante :

Étant données les entrées suivantes :

- Un schéma d'entrepôt de données relationnel D;
- Une charge de requêtes Q, représentée par son MVPP de l nœuds :$\{no_1, no_2, \ldots no_l\}$;

Et étant donnée la contrainte suivante :

- La taille du buffer B

Le problème de la gestion du buffer consiste à choisir une stratégie de gestion du buffer SGB permettant l'allocation et le remplacement des données dans le buffer, tel que le coût d'exécution de la charge de requêtes Q soit minimal.

À la différence d'une politique de gestion (Section 4.2.1), une stratégie de gestion du buffer est l'ensemble des traitements constituant le scénario de la mise en buffer.

4.2.1.1 Architecture de la mémoire :

Toutes les composantes du SGBD nécessitant des informations dans le disque, interagissent avec le buffer et le gestionnaire de buffer (cf. figure 2.1). Les informations requises par les différentes composantes sont : les données, les méta-données, les statistiques et les index [101]. Il existe deux types d'architectures pour les gestionnaires de buffer :

- Le gestionnaire effectue un contrôle direct de la mémoire principale, ce qui est le cas dans les SGBD relationnels.

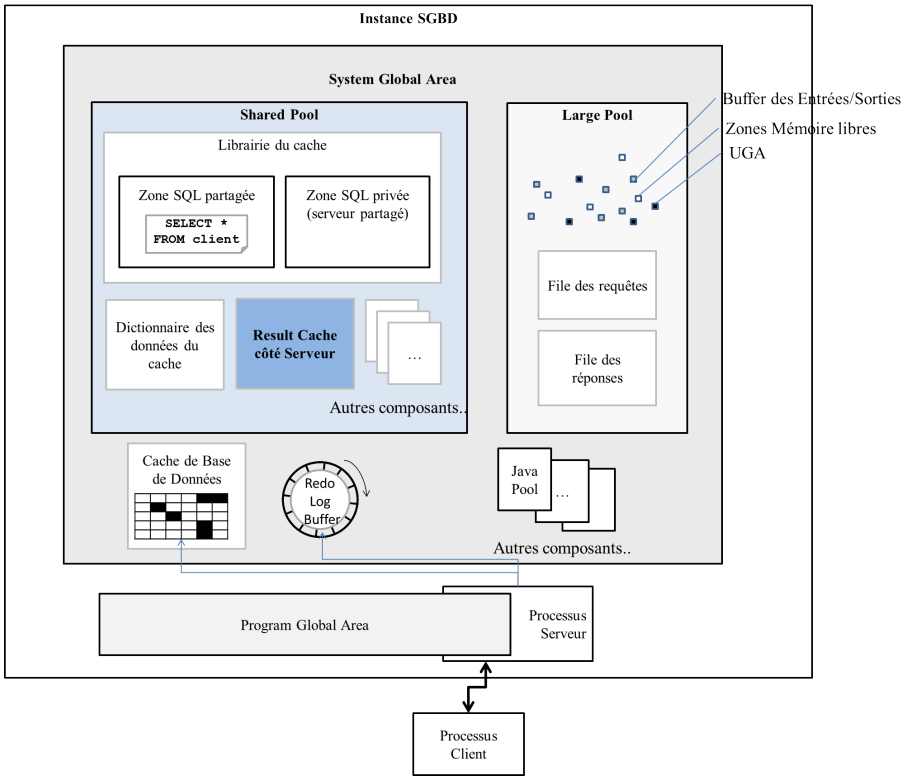

FIGURE 2.16 – Les principales composantes de gestion de la mémoire dans un SGBD

– Le gestionnaire alloue le buffer dans la mémoire virtuelle, ce qui permet au système d'exploitation de décider quelles parties du buffer allouer en mémoire principale, ou dans le *swap* géré par le système d'exploitation. Ce type d'architecture est utilisé dans les SGBD orienté-objets et les SGBD *In-memory*.

Dans les deux types d'architecture, on est confronté au même problème de gestion de la mémoire: le gestionnaire doit décider quelles pages allouer dans l'espace limité, et lesquelles ôter dans le cas où la mémoire buffer est saturée [101].

Dans cette section, nous prenons l'exemple d'un SGBD Oracle afin de détailler les composantes de la mémoire centrale responsables du traitement des requêtes SQL. Les principales structures associées à la base de données sont :

– System Global Area (SGA) : une partie de la mémoire de lecture/écriture partagée par tous les processus d'une instance Oracle.
– Program Global Area (PGA) : une partie mémoire non partagée contenant des données exclusives pour un processus. Un PGA existe pour chaque processus et leur collection forme l'instance totale du PGA.
– User Global Area (UGA) : est une partie mémoire associée à une session utilisateur.
– Autres zones pour les logiciels intégrés.

La SGA contient plusieurs structures désignées chacune à une tâche particulière durant l'exécution de la base de données Oracle [6]. Les paramètres de la SGA sont donnés au démarrage de l'instance et souvent contenus dans le fichier d'initialisation "init.ora". Par exemple, le paramètre SGA_MAX_SIZE

définissant la taille de la mémoire SGA. Si plusieurs utilisateurs se connectent de façon concurrente à la même instance, les données de cette instance sont partagées par ces utilisateurs. Par conséquent, la SGA est parfois appelée *Shared Global Area* ou zone mémoire partagée. Afin d'assurer ses tâches, la SGA contient les structures suivantes :

- Redo Log Buffer : contient les informations concernant les transactions/requêtes qui ont été validées, mais qui n'ont pas encore été écrites sur les données.
- Buffer cache : contient les copies de données brutes ou pré-calculées résultants des traitements des requêtes SQL. Ces copies sont en forme de blocs d'une taille fixe définie dans le paramètre DB_BLOCK_SIZE (entre 2 K et 32 K). La taille totale d'un buffer est stockée dans le paramètre DB_CACHE_SIZE.
- Shared pool : ou la mémoire partagée, contient la trace (les informations) des requêtes SQL ou PL/SQL récemment exécutées. Parmi les données qui y sont cachées, on trouve le dictionnaire de données du cache (Data dictionary cache), le cache des résultats de requêtes (Result Cache) et la librairie du cache.
- Java pool : une composante optionnelle permettant d'utiliser des fonctionnalités Java sur le serveur.
- Large pool : une composante optionnelle permettant de réduire la charge de la mémoire partagée (Shared pool).

4.2.1.2 Mise en cache des résultats de requêtes Le Result Cache est une partie mémoire qui peut être sur le serveur ou sur le client. Elle contient principalement les résultats de requêtes déjà exécutées. Contrairement au cache de BD (cf. figure 2.16), le Result Cache contient des ensembles de résultats de traitement au lieu de blocs de données. Ces résultats proviennent des requêtes SQL et des fonctions PL/SQL. Les premiers se cachent dans le SQL Query Result Cache, et les seconds se cachent dans le PL/SQL Function Result Cache. À la différence d'un Result Cache côté serveur, un Result Cache côté client est configuré au niveau application et se loge dans une mémoire client au lieu de la base de données.

Pour permettre la réutilisation des résultats, Oracle dispose de *hints* permettant de forcer la mise en cache des résultats d'une requête SQL. Le hint employé est : /+ *result_cache* +/.

Exemple. Soit la requête SQL suivante :

```
SELECT /*+ RESULT_CACHE */ ville, AVG(Age)
  FROM client
 GROUP BY ville;
```

Le résultat de cette requête sera mis en cache pour une utilisation ultérieure. Si la même requête est lancée une autre fois, en supposant que le contenu du buffer n'a pas été modifié entre les deux lancements, alors le SGBD lira le résultat déjà calculé au lieu de le recharger à partir du disque :

```
-----------------------------------------------------------------------------------------------
| Id  | Operation           | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                             |     5 |    30 |  1846  (25)| 00:00:23 |
|   1 |  RESULT CACHE       | gk69saf6h3ujx525twvvsnaytd |       |       |            |          |
|   2 |   HASH GROUP BY     |                             |     5 |    30 |  1846  (25)| 00:00:23 |
|   3 |    TABLE ACCESS FULL| CLIENT                      |  1000K|  5859K|  1495   (7)| 00:00:18 |
-----------------------------------------------------------------------------------------------
```

4.2.2 L'ordonnancement des requêtes

Initialement, l'ordonnancement a été défini et étudié dans le contexte d'optimisation dans les entreprises et les compagnies de production pour établir les emplois du temps et d'autres planifications des tâches. Par la suite, il a été introduit dans les systèmes d'exploitation comme un système de gestion de processus. Les SGBD ont adopté cette technique à deux niveaux. Dans le premier niveau, l'ordonnanceur est interne au SGBD, et permet d'ordonnancer ses processus pour assurer qu'ils soient tous exécutés au bout d'un temps fini [11]. Le deuxième niveau concerne l'ordonnanceur de requêtes. Cet ordonnanceur a été considéré comme une technique d'optimisation non redondante qui cherche à réduire le temps d'exécution de la charge.

Comme nous l'avons présenté dans l'étude de la fragmentation horizontale, l'interaction entre les requêtes est très fréquente dans le contexte de bases de données, notamment dans les EDR. Elle peut être exploitée en réutilisant les sous-expressions communes entre les requêtes dans le but de réduire le nombre d'accès au disque. Les résultats de ces sous-expressions peuvent être stockés en mémoire secondaire de façon permanente (vues matérialisées), ou sur la mémoire principale de façon temporaire (objets du buffer). Le rôle de l'ordonnanceur est de planifier l'exécution des requêtes de la charge de façon à exploiter au mieux les données pré-calculées existantes sur les supports de stockage. Ainsi, l'ordonnancement des requêtes dans les bases de données peut tirer profit des données pré-calculées présentes sur les supports de stockage.

En plus de la réduction du coût d'exécution de la charge, l'ordonnanceur peut aussi privilégier certaines requêtes prioritaires en écourtant leur temps d'attente dans la file. Cet aspect entre dans le cadre des travaux sur la qualité des services (*Quality of Service*) [196].

Définition 1

Nous appelons Planning d'Exécution (PE, ou Schedule en anglais), l'ordre des requêtes défini par l'ordonnanceur.

Exemple. Soit une file d'attente φ composée d'un ensemble de trois requêtes. Les requêtes s'exécutent sur une mémoire principale MP (contenant un buffer de base de données) et une mémoire secondaire MS. Dans la figure 2.17, quand la requête Q2 est exécutée avant Q1, son résultat est stocké sur le support de stockage employé pour l'optimisation (soit la MP ou la MS). Ce résultat est lu directement pour répondre à la requête Q1. Cependant, dans le deuxième ordre, où Q3 est lancée avant Q1, le support stocke les résultats de Q3, ce qui réduit la taille mémoire disponible. L'exécution de Q1 permet de générer deux résultats : $(A \bowtie B)$ et $(A \bowtie B) \bowtie C$. L'espace mémoire disponible ne permet pas de stocker les deux ensembles de données. Le stockage du résultat de $(A \bowtie B) \bowtie C$ ne servira pas la requête Q2. Ainsi, l'ordre $Q2 \rightarrow Q1 \rightarrow Q3$ affiche un coût d'exécution plus faible que l'ordre $Q3 \rightarrow Q1 \rightarrow Q2$.

Formalisation du problème d'ordonnancement des requêtes : Nous formalisons le problème d'ordonnancement des requêtes de la façon suivante :

Étant données les entrées suivantes :

– Un schéma d'entrepôt de données relationnel D;

11. Cette propriété est dite : *la garantie d'absence de famine* dans les problèmes d'ordonnancement

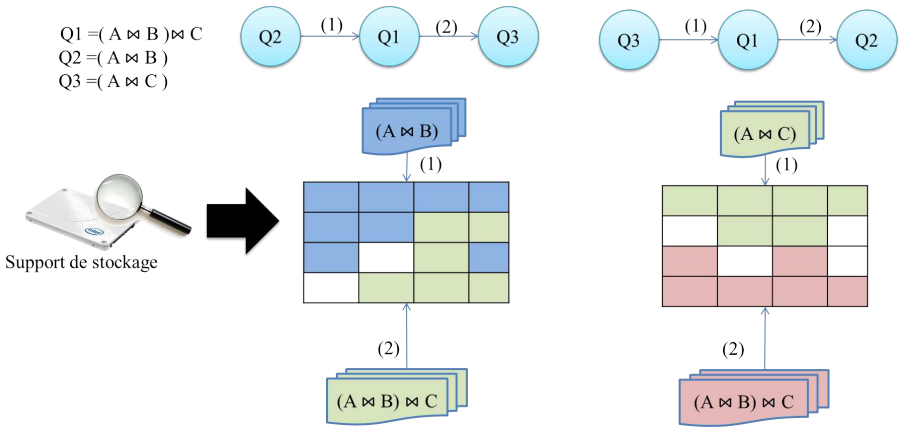

FIGURE 2.17 – Exemple d'ordonnancement de trois requêtes

– Une charge de requêtes Q, représentée par son MVPP de l nœuds : $\{no_1, no_2, \ldots no_l\}$;
– Une politique de gestion PG_S du support de stockage S dans lequel les données réutilisables vont être conservées de façon permanente ou temporaire. Elle permet l'allocation et le remplacement des données sur le support S.

Et soit les contraintes suivantes (optionnelles) :

– Le délai d'attente maximal ou la priorité de certaines requêtes de Q.

Le problème d'ordonanncement des requêtes consiste à produire un planning d'exécution $\mathcal{P}$ définissant l'ordre dans lequel les requêtes doivent être exécutées, tel que le coût d'exécution de la charge Q soit minimal.

Le coût à réduire peut être le nombre d'Entrées/Sorties, le temps de traitement, les opérations CPU ou le coût énergétique.

4.2.3 Combinaison de la gestion du buffer et de l'ordonnancement des requêtes

Les deux problèmes de gestion du buffer et d'ordonnancement des requêtes sont fortement dépendants (Section 4.1). Leur combinaison permet de tirer profit de leurs avantages respectifs. En effet, l'ordonnancement peut exploiter le contenu du buffer pour réduire le nombre d'accès au disque, et par conséquent réduire le temps de réponse et le coût énergétique de la charge. D'autre part, le contenu du buffer dépend de l'ordre dans lequel les requêtes s'exécutent.

4.2.4 Travaux sur la gestion du buffer

La gestion du buffer est l'un des domaines de recherche les plus anciens en base de données. Les premiers travaux ont considéré l'étude de la mémoire avec ces différents types. Mattson et al. [147] ont étudié la hiérarchie des mémoires. Ils présentent une étude comparative entre les différentes politiques d'allocation et de remplacement comme : LRU (*Least Recenty Used*), FIFO (*Firts In First Out*) et Random (algorithme de remplacement de pages aléatoires). Depuis le début des années 80, les chercheurs se sont intéressés à l'exploitation de la mémoire buffer afin d'améliorer les performances du système. Dans le travail de Kaplan [120], plusieurs politiques de gestion de buffer non-LRU ont été simulées sur les

données d'un SGBD INGRES. Ces alternatives ont permis de réduire le taux de défaut de cache (cache miss) de 10 à 15 % par rapport à LRU.

D'autres travaux ont considéré le lien entre la mémoire buffer des systèmes d'exploitation (SE) et celle des SGBD. Le travail de Stonebraker [181] montre le rôle des systèmes d'exploitation comme un support aux SGBD. Les systèmes d'exploitation possèdent leur propre mémoire buffer. Parallèlement, les SGBD ont une mémoire buffer indépendante de celle des SE pour plusieurs raisons qu'ils ont discuté dans [181]. En se basant sur le fait qu'un accès disque provoque 5000 opérations CPU, les auteurs présument qu'un gestionnaire de buffer doit offrir un coût d'accès réduit à quelques centaines d'instructions pour être bénéfique. D'autre part, et malgré l'existence de plusieurs variantes de LRU, cette politique n'est pas considérée comme une politique convenable aux systèmes de bases de données [181, 101].

Effelsberg et Härder [89] ont défini les principes de gestion de la mémoire buffer des bases de données. Les principales tâches d'un gestionnaire de buffer ont été définies par les trois fonctions suivantes :

- La recherche dans la mémoire buffer;
- L'allocation des blocs;
- Le remplacement des pages.

Le but principal de la gestion du buffer est de minimiser les Entrées/Sorties (E/S) physiques car les accès à une page sur le disque sont beaucoup plus coûteux que sur le buffer [89, 101]. Les auteurs détaillent les différentes étapes de traitement d'une lecture/écriture au niveau physique. L'étude a été réalisée sur un système de gestion de base de données CODASYL, qui est un système navigationnel distinct du système hiérarchique proposé par IBM.

Le travail de Chou et DeWitt présente une évaluation des stratégies de gestion de buffer existantes [74]. Les instances actives des différents utilisateurs sont attribuées aux différentes mémoires buffer et gérées par plusieurs stratégies de gestion différentes. Ce travail cherche la *bonne* politique d'allocation pour un plan de requête donné sans considérer l'effet des autres requêtes sur l'optimisation.

Sacco et Scholnick ont étudié la gestion du buffer et ont proposé deux principaux travaux [167, 168]. Dans leur premier travail [167], un *Hot Set Model* a été proposé. Le *Hot Set* définit les pages sur lesquelles il existe des lectures en boucle (*Looping Behavior*) [74]. Le modèle proposé détermine le *hot set* pour chaque requête. Le buffer est alloué de façon à couvrir le *hot set* avant l'exécution de la requête. Cependant, ce type d'optimisation est local et ignore l'ensemble des requêtes. De plus, le contenu du buffer n'est pas considéré dans la politique d'allocation. Dans leur deuxième travail [168], les auteurs ont également noté que l'implémentation de l'approche proposée peut provoquer un temps d'attente infini dans le SGBD.

A la fin des années 80, les chercheurs se sont intéressés à intégrer la gestion du buffer dans le processus d'optimisation des requêtes comme une technique d'optimisation à part entière. Cornel et Yu [81] ont proposé un gestionnaire de buffer intégré au processus d'optimisation. La principale motivation est l'existence de certaines données, dans les plans d'accès des requêtes, qui sont utiles pour le gestionnaire du buffer. D'autre part, les plans d'accès aux requêtes varient sous des tailles de buffer différentes. Ce travail porte des critiques sur les approches existantes basées sur le *Working Set Model* comme LRU. Le *Working Set* est un concept définissant la mémoire requise pas un processus dans un intervalle de temps [88]. Le travail propose une alternative basée sur le *Hot Set Model* [88, 90].

Cornell et Yu [81] défendent l'idée que le *Working Set Model* convient aux systèmes de base de

données réseaux ou hiérarchiques. Cependant, les bases de données relationnelles [77] nécessitent plus d'informations sur les données, ce qui signifie que le *Hot Set Model* est plus adéquat. Ils ont également comparé le *working set model* (par LRU) avec le *hot set model* afin d'intégrer la gestion du buffer dans le processus d'optimisation des requêtes. Dans le but de réduire la taille du problème traité, une méthodologie a été proposée en se basant sur l'occupation du buffer afin de réduire le nombre de plans d'accès à considérer. Les approches conventionnelles d'optimisation estiment le coût de chaque requête indépendamment des autres en utilisant des fonctions de coût d'Entrées/Sorties [175]. L'impact du buffer n'est souvent pas reflétée dans ce type de fonctions d'estimation [81]. L'estimation du coût nécessite non seulement la connaissance des opérations de jointures requises, mais aussi leur ordre d'exécution. La difficulté d'estimation est liée à la multitude de paramètres à considérer dans l'exécution d'une requête. Plusieurs hypothèses ont été définies pour simplifier les fonctions de calcul de coût. Le contenu du buffer est l'un des paramètres importants dans cette estimation. Le travail de Cornel et Yu [81] montre que le plan d'accès optimal pour une requête dépend de l'allocation du buffer et des compromis établis entre les besoins du CPU et des E/S. De plus, les travaux précédents ont considéré des requêtes individuelles, or dans l'optimisation multi-requêtes (MQO), l'espace du buffer doit être alloué pour satisfaire la totalité de la charge. Un Recuit Simulé a été proposé pour intégrer la gestion du buffer dans la phase d'optimisation.

La gestion du buffer a suivi l'évolution des supports de stockage et des systèmes de bases de données. Les travaux récents s'intéressent à l'adaptation des politiques de gestion du buffer aux nouvelles caractéristiques physiques de l'environnement de base de données. Parmi ces travaux, nous citons Ou *et al.* [158] qui ont étudié la gestion du buffer sur les mémoires flash. Sachant que les caractéristiques de la mémoire flash nécessitent une adaptation des politiques de remplacement des pages dans le buffer, les auteurs ont proposé une nouvelle politique de remplacement auto-adaptative dirigée par l'estimation du coût.

En plus des travaux réalisés dans les bases de données traditionnelles [168, 89, 74, 81], d'autres types de bases de données ont été concernés par la gestion du buffer, comme les bases de données sémantiques [200] et les entrepôts de données [172]. Dans la majeure partie des travaux récents, la gestion du buffer est effectuée par les politiques d'allocation utilisées par les systèmes d'exploitation comme LRU, qui ne sont pas adaptées aux systèmes de bases de données [172, 101] car elles ne considèrent pas l'optimisation multi-requêtes. Une nouvelle approche de gestion appelée *Watchman* est proposée par Scheuermann *et al.* [172]. Les auteurs montrent que dans un entrepôt de données, la mise en cache d'un ensemble d'instances résultantes d'une requête est plus bénéfique que la mise en cache des pages individuelles.

Traditionnellement, le tuning est effectué par les experts. Avec l'évolution des supports de stockage et des systèmes de base de données, choisir un compromis entre la capacité des supports et leur prix devient plus difficile, notamment dans les systèmes où les données évoluent et nécessitent une adaptation dynamique de leurs structures d'optimisation. D'autres travaux [93, 182, 183, 187] proposent un tuning auto-adaptatif en considérant l'arrivée des nouvelles charges et des nouvelles statistiques afin d'améliorer le taux de succès de cache (*cache hit ratio*). Tran *et al.* [187] ont proposé une approche de *tuning* automatique et dynamique des buffers de base de données afin de réduire les défauts de pages (*cache miss*). Leur motivation est que les systèmes croissent et deviennent plus sophistiqués, ce qui impose l'utilisation des outils d'aide à la prise de décision (ou *advisors* en anglais). De plus, les charges de requêtes changent constamment (tous les mois, jours, heures, etc.). Ainsi, la taille du buffer et la répartition des ressources partagées doivent s'adapter en conséquence.

SGBD	DB2	SQLServer	Oracle	MySQL	Postgres	Sybase	TERADATA
Ordonnanceur intégré	oui	oui	oui	oui	non	non	non
Version actuelle	9.7	2008 R2	11g	2.2	9	15.5	13.0

TABLE 2.3 – L'intégration des ordonnanceurs dans les SGBD

4.2.5 Travaux sur l'ordonnancement des requêtes

L'ordonnancement a été définit initialement dans les compagnies de production. Il a été intégré par la suite comme une composante dans les systèmes d'exploitation dite *ordonnanceur*. L'ordonnanceur permet de définir l'ordre dans lequel les processus doivent être exécutés. Le rôle principal de l'ordonnanceur est de permettre à tous les processus de s'exécuter tout en optimisant les ressources physiques comme le processeur ou les bus ou encore pour la gestion des sections critiques comme les imprimantes partagées ou les pages en écriture [19].

L'ordonnancement a été intégré aux SGBD afin de permettre une bonne gestion de leurs processus. Parmi les SGBD commerciaux ayant un ordonnanceur intégré, nous citons : DB2, SQLServer et Oracle. (Table 2.3). L'ordonnanceur d'un SGBD permet de supporter la concurrence des utilisateurs et des transactions en assurant que tous les utilisateurs puissent accéder à la donnée dans un intervalle de temps limité. Dans ces ordonnanceurs, le but principal est l'accès à la ressource indépendamment de la performance qui est considérée comme une tâche de l'optimiseur seul. Cependant, l'ordre d'exécution des requêtes permet d'augmenter la performance du système en exploitant les données pré-calculées disponibles à chaque instant. Cette propriété a été étudiée dans la dernière décennie où les chercheurs l'ont considéré comme une technique d'optimisation des requêtes [146, 163, 143].

L'ordonnancement des requêtes a été prouvé corrélé avec d'autres techniques d'optimisation. La corrélation réside dans l'utilisation des données pré-calculées stockées temporairement sur le buffer ou de façon permanente sur le disque. Dans le premier cas, la gestion du buffer est employée [111, 184, 186]. Dans le travail de Gupta *et al.* [111], la gestion du buffer et l'ordonnancement des requêtes ont été considérés dans les entrepôts de données relationnels. Plusieurs solutions ont été proposées pour combiner les deux techniques en considérant l'optimisation multi-requêtes (MQO). Les auteurs proposent une étude de complexité du problème où ils prouvent sa $\mathcal{NP}$-complétude via une preuve de $\mathcal{NP}$-complétude du problème d'ordonnancement. La validation des algorithmes n'a pas été établie dans un SGBD réel pour prouver leur efficacité notamment lors du passage à l'échelle. D'autres travaux ont considéré la combinaison des deux techniques dans des environnements divers comme les bases de données centralisées [186], distribuées et parallèles [146].

Dans le deuxième cas, où les données sont stockées sur la mémoire secondaire (persistantes), les vues matérialisées sont employées [163]. Cependant, l'ordonnancement n'a pas un sens si les données ne subissent aucun remplacement. Autrement dit, si les mêmes données restent sur le support de stockage toute la durée d'exécution de la charge, l'ordre d'exécution des requêtes n'a pas un impact sur la performance. Pour cela, les vues matérialisées sont créés de façon dynamique. Ceci permet de faire avancer les requêtes utilisant le contenu actuel de la mémoire avant qu'il soit remplacé par de nouvelles vues satisfaisant un autre ensemble de requêtes. Dans le travail de Phan *et al.* [163], une approche de création de vues matérialisées dynamiquement a été proposée. L'espace des candidats à la matérialisation est obtenu

	Gestion du buffer	Ordonnancement des requêtes	GBOR
Hors-ligne	[81] [158] [168] [165] [172] [200]	[146]	[111] [184] [186]
Dynamique	[93] [173] [182] [183]	[163]	Aucun
En-ligne	[187]	[143]	Aucun

TABLE 2.4 – Classification des travaux existants dans la GB et l'OR

en utilisant l'un outil d'aide à la prise de décision : DB2 Advisor, intégré au SGBD DB2. La sélection des vues matérialisées permettant d'augmenter la performance du système est obtenue par un algorithme génétique. L'approche montre l'interaction entre les supports de stockage, les requêtes et les techniques d'optimisation mais ce résultat n'est pas explicite dans le travail. Parmi les limites de l'approche, il y a sa dépendance au SGBD employé. Elle n'est donc pas multi-plateforme. De plus, l'algorithme génétique est très gourmand car il appartient à la classe de méta-heuristiques évolutionnaires manipulant des populations. Ce type d'algorithmes ne convient pas lors du passage à l'échelle, ou du traitement en-ligne.

4.2.6 Synthèse des travaux de la gestion du buffer et de l'ordonnancement des requêtes

Beaucoup de travaux ont considéré la gestion du buffer et l'ordonnancement des requêtes de manière isolée ou conjointe. En analysant la littérature, nous identifions trois classes d'études : les études d'optimisation hors-ligne (statiques), dynamique et en-ligne.

Optimisation hors-ligne : Dans ce cas, la charge est supposée connue au préalable. Les deux problèmes isolées et combinées ont été largement considérés dans cette classe.

Optimisation dynamique : Dans ce cas, des efforts ont été établis en adaptant le système (de façon hors-ligne ou en-ligne) en considérant les changements apportés aux données et aux requêtes. Par exemple, le taux de succès du cache (*cache hit ratio* en anglais) peut être adapté dynamiquement suite aux changements de la charge. L'ordonnancement de requêtes dynamique a suscité moins d'intérêt dans la communauté de bases de données comparé au problème de gestion du buffer.

Optimisation en-ligne : Dans ce niveau, le processus d'optimisation se déroule durant l'exécution du système. Peu de travaux ont considéré le cas en-ligne dans chacun des deux problèmes isolés. A notre connaissance, aucun travail n'a été établi sur la combinaison des deux techniques dans le cas en-ligne. Le problème est le manque de connaissances requises pour établir un ordonnancement ou une stratégie de buffer appropriée. La Table 2.4 résume notre discussion en montrant l'intérêt apporté à chaque technique dans les différents modes (isolé et combiné).

4.3 Limites de la sélection multiple

Comme nous l'avons présenté précédemment, les problèmes liés aux structures d'optimisation sont $\mathcal{NP}$-complets. Souvent, la combinaison des structures d'optimisation engendre une augmentation de l'espace de recherche et un degré de complexité plus élevé que le mode isolée. Dans les approches existantes, la combinaison est effectuée en utilisant des approches complexes, et qui exploitent partiellement les requêtes, les supports ou les données. Afin de contourner cette complexité, l'interaction doit être dûment exploitée pour écarter les candidats non pertinents de l'espace de recherche.

5 Bilan et discussion

La conception physique est un processus important dans les bases de données, et en particulier, les entrepôts de données relationnels. Ces derniers possèdent plusieurs caractéristiques liées à leur fonction principale qui est l'analyse et l'aide à la prise de décision. Parmi ces caractéristiques, nous citons la volumétrie des tables, le schéma dénormalisé et la complexité des requêtes OLAP. Comme le processus de prise de décision est critique dans une entreprise, le temps de réponse doit être raisonnable malgré le volume important des données et la complexité des requêtes. La conception physique permet de sélectionner l'ensemble des structures d'optimisation à implémenter avec les données de l'entrepôt de façon à réduire au mieux le temps de réponse de la charge.

La conception physique impose plusieurs choix à l'administrateur [52] :
– Le choix des structures d'optimisation parmi celles supportées par le SGBD;
– Le choix du mode de sélection des structures d'optimisation : Isolée ou Multiple;
– Le choix de l'algorithme de résolution et de ses paramètres.

La principale limite des approches existantes est l'ignorance mutuelle de certains facteurs comme le contenu du cache par la FH, et la répartition des données par la GB.

6 Nouvelle vision

Dans cette section, nous proposons une nouvelle vision qui découle de notre analyse de l'état de l'art. Nous proposons ainsi un nouveau modèle pour la formalisation du problème de la conception physique. De ce modèle, nous allons extraire un certain nombre de propriétés que nous pouvons exploiter pour la résolution des problèmes d'optimisation.

6.1 Vers une explicitation des paramètres de la conception physique

En analysant la formalisation classique du problème de la conception physique, nous observons que le processus prend des requêtes, un schéma de BD et des structures d'optimisation avec leurs contraintes afin de fournir un schéma de structure d'optimisation minimisant le coût de la charge des requêtes.

La sélection du *meilleur* schéma en sortie s'effectue par une métrique d'évaluation dite *Modèle de Coût*. Ce modèle mathématique permet de mesurer la qualité d'un schéma pour aiguiller le processus

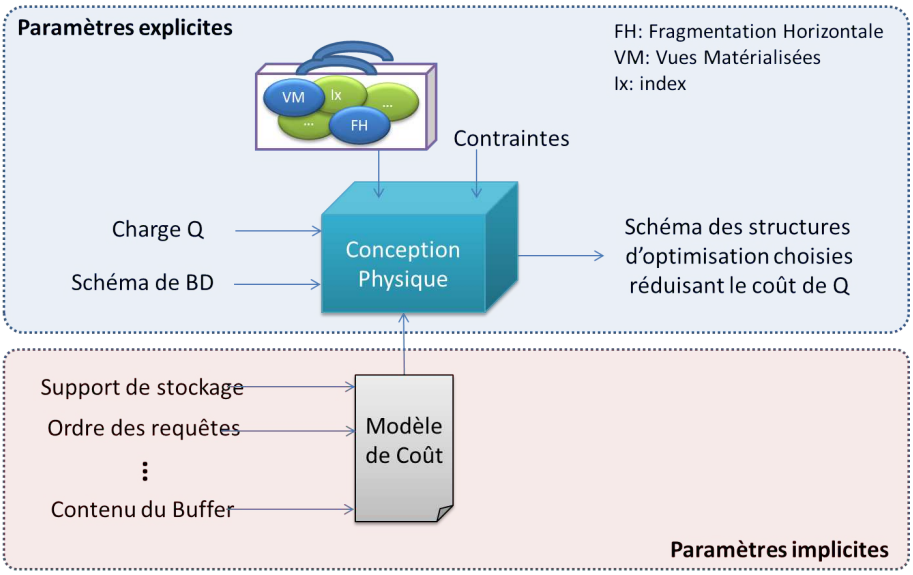

FIGURE 2.18 – Les paramètres implicites et explicites de la conception physique

d'optimisation et d'en sélectionner le meilleur. Pour permettre d'effectuer ces mesures, le modèle de coût utilise un ensemble de paramètres et de méta-données liés à trois composantes :

- Les requêtes : types de requêtes, prédicats, l'ordre, etc.
- La base de données : les attributs, les instances, le schéma, etc.
- L'architecture de l'environnement : le support de stockage, l'architecture de la base, le débit, etc.

Dans la formalisation existante de la conception physique, le support est considéré souvent comme un disque dur (Hard Disc Drive, HDD). Avec l'évolution des supports de stockages, une multitude de supports sont adoptés. Bien qu'il représente un paramètre important dans l'estimation du coût, le support n'est pas explicite dans la formalisation de la conception physique. Il doit donc être intégré dans le modèle de coût par une inspection de l'environnement.

Le support de stockage n'est pas la seule composante qui manque dans la formalisation, mais aussi l'ordre des requêtes et la politique de gestion du buffer. Nous distinguons ainsi deux classes de paramètres liés au processus de la conception physique (cf. figure 2.18):

- **Les paramètres explicites :** ce sont des paramètres directement liés au processus de la conception physique. Ces paramètres sont les requêtes, le schéma de la base de données, les structures d'optimisation et leurs contraintes.
- **Les paramètres implicites :** ce sont les paramètres indirectement liés au processus de la conception physique et souvent cachés dans le modèle de coût. Ces paramètres concernent l'ordre des requêtes, le support de stockage et l'architecture.

Les données implicites posent une ambiguïté dans la formalisation, et imposent un effort supplémentaire dans l'élaboration du modèle de coût. Nous proposons ainsi de revoir la formalisation en explicitant tous les paramètres de la conception physique.

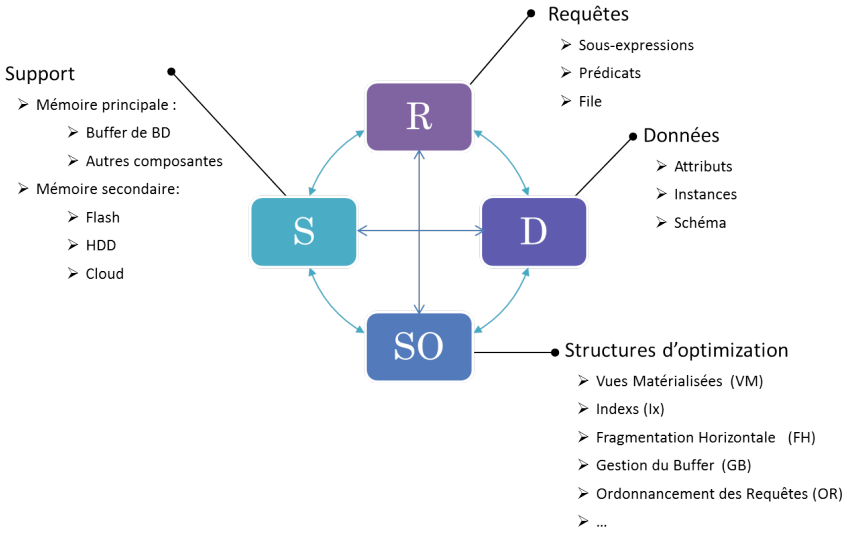

FIGURE 2.19 – Le modèle explicite de la conception physique

6.2 Modèle explicite de la conception physique

Pour remédier aux problèmes des données implicites, nous proposons un nouveau modèle permettant de ressortir les paramètres requis lors de la formalisation. Nous appelons ce modèle, le Modèle Explicite de la Conception Physique (MECP).

L'étude menée sur la formalisation classique dévoile l'existence de quatre principales composantes : les requêtes, le schéma de la base, les structures d'optimisation et le support de stockage. Ces composantes sont les principaux axes dans notre modèle. Chacune des composantes est constituée de plusieurs éléments. La figure 2.19 expose notre modèle avec les différents paramètres explicites de la conception physique.

1. Les requêtes (R) : cette composante représente la charge. Elle contient trois éléments : les sous-expressions, les prédicats et la file.
2. Les données (D) : cette composante représente la base de données. Elle contient : les attributs, les instances (n-uplets) et le schéma de la base.
3. Les supports (S) : contient une mémoire principale avec un buffer et d'autres composantes de gestion et une ou plusieurs mémoires secondaires (e.g. Cloud ou Flash.).
4. Les structures d'optimisation (SO) : c'est l'ensemble des structures supportées par le SGBD utilisées pour réduire le coût de la charge de requêtes.

Considérons notre MECP comme un quadruplet :

$$\mathbb{PCP} =< R, D, S, SO >$$

Où R, D, S et OS représentent respectivement les quatre composantes : Requêtes, Données, Support et Structure d'Optimisation. Dans ce cas, une instance du problème de la conception physique pour la fragmentation horizontale peut être représentée de la façon suivante :

$$Problème\ de\ la\ FH =< \{\emptyset, Predicats, \emptyset\}, \{\emptyset, instances, SchemaEnEtoile\}, \{\emptyset, HDD\}, \{FH\} >$$

Dans cette formalisation, nous pouvons constater l'absence de la mémoire principale dans la définition du problème, ainsi que l'ignorance des sous-expressions et de la file des requêtes.

De même, une instance de la conception physique dans un cas de sélection multiple de GB et d'OR, sera représentée de la façon suivante :

$$Problème\ de\ GBOR =< \{sousexpression, \emptyset, file\}, \{\emptyset, instances, \emptyset\}, \{Buffer, \emptyset\}, \{GB, OR\} >$$

Nous constatons à ce niveau l'ignorance du schéma et des attributs lors de l'optimisation des requêtes par la GBOR. Ceci peut limiter le rendement de n'importe quel algorithme employé pour l'optimisation par GBOR.

6.3 L'omniprésence de l'interaction

Dans le modèle explicite que nous proposons, plusieurs niveaux d'interactions apparaissent. En effet, à l'intérieur de chaque composante, les éléments interagissent entre eux. Par exemple, les structures d'optimisation interagissent entre-elles à cause des dépendances Fortes et Faibles identifiées dans le travail de Zilio *et al.* [201]. Les index et les vues matérialisées, ou encore les index et la fragmentation horizontale [169, 33] sont des exemples de cette interaction. De plus, il existe une interaction entre les éléments de deux composantes différentes. Les vues matérialisées (composante Structure d'Optimisation), par exemple, interagissent avec le support de stockage (composante Support) [33].

Ainsi, le MECP nous permet de distinguer deux types d'interaction dans l'environnement de bases de données:

- Interaction intra-composante : c'est une dépendance entre les éléments d'une même composante. Plus formellement, l'interaction est intra-composante si deux éléments e_i^k et e_j^k d'une même composante k du MECP interagissent, où i peut être égale à j.
- Interaction inter-composante : c'est une dépendance entre les éléments de plusieurs composantes. Plus formellement, l'interaction est inter-composante si deux éléments e_i^k et e_j^l de deux composantes respectivement k et l du MECP interagissent.

6.3.1 Interaction intra-composante

Cette interaction touche aux éléments d'une même composante d'un système de BD. Elle a été établie dans quelques travaux d'optimisation récents [169, 201, 33], montrant l'existence d'interaction entre les structures d'optimisation. Les attributs interagissent également, ce qui permet de définir des hiérarchies dans les schémas en flocon de neige [10], ou les fragments verticaux d'une relation [113]. Cependant, aucun des travaux existant n'a considéré la totalité des éléments de l'ensemble.

Dans la section qui suit, nous donnons quelques exemples de ce type d'interaction :

Interaction des supports de stockage : Afin d'effectuer les différents traitements, les SGBD chargent les données de la mémoire secondaire (où elles sont logées de façon permanente) en mémoire principale

(où elles seront logées temporairement). La mémoire principale est un passage obligatoire pour toutes ses opérations. Plusieurs travaux d'optimisation se focalisent sur la mémoire secondaire sans considérer le buffer de la mémoire principale. L'ignorance d'une telle interaction nuit à l'estimation du coût et ainsi à la qualité de la solution finale.

De plus, l'espace disponible dans les deux mémoires peut être exploitée de façon conjointe afin de sélectionner à la fois des structures d'optimisation persistantes (vues matérialisées) ou temporaires (objets du buffer).

Interaction des requêtes : dans le travail de Sellis [176], l'optimisation multi-requête a été initiée comme une alternative à l'optimisation des requêtes individuelles. La raison derrière cette proposition est que les requêtes partagent des sous-expressions communes, et ainsi des sous-résultats communs qui peuvent être considérés afin d'augmenter les performances du système.

Considérons la relation universelle RU obtenue par le produit cartésien de toutes les relations de la base. Un résultat d'une requête ou d'une sous-expression algébrique est toujours un sous-ensemble d'instances et d'attributs de cette relation universelle.

Dans le cas d'un entrepôt de données, toutes les jointures passent par la table des faits. Ainsi, le résultat final d'une requête de jointure en étoile est une partie de cette table des faits, où les différentes jointures des dimensions et leurs prédicats de sélection constituent des filtres permettant d'éliminer les faits non-pertinents de l'analyse. La probabilité d'avoir des sous-résultats communs entre les requêtes OLAP est importante à cause de son schéma et des requêtes de jointures en étoile.

Interaction des données : Les attributs peuvent avoir des liens de dépendances entre eux. Ces liens sont souvent exploités pour définir les hiérarchies des dimensions dans les schémas en flocon de neige. De plus, les instances des tables, et notamment les instances de la table des faits dans un entrepôt de données, sont des lieux d'interaction où plusieurs opérations ciblent les mêmes ensembles d'instances.

Interaction des structures d'optimisation : Certaines structures d'optimisation portent sur un ensemble d'éléments en ignorant d'autres. Considérons les instances du problème de la conception physique pour la FH et la GBOR, présentées dans un exemple précédant (cf. page 49).

Nous constatons que la fragmentation ignore la mémoire principale et la GBOR ignore la façon dont les données sont partitionnées. D'autres structures produisent en sorties des schémas pouvant être en entrées des autres structures. Par exemple, la fragmentation horizontale produit un schéma de partitionnement sur lequel les requêtes peuvent encore être optimisées par la gestion du buffer et/ou par l'ordonnancement.

Il serait donc intéressant d'exploiter ces structures par une sélection multiple afin de tirer profit de leur complémentarité.

6.3.2 Interaction inter-composante

Plusieurs éléments n'appartenant pas à la même composante peuvent aussi interagir. C'est le cas des structures d'optimisation et des supports de stockage, ou encore les instances. Ce type d'interaction ne concerne pas seulement deux composantes de l'environnement, mais toutes les composantes à la fois. Pour illustrer ce phénomène, nous instancions le MECP dans le cas d'une sélection isolée, et d'une

sélection multiple de structures d'optimisation.

6.3.2.1 Sélection isolée : La FH L'instanciation du problème de fragmentation horizontale dans notre MECP s'effectue comme suit : Au niveau des requêtes, la FH s'intéressent aux prédicats dans les approches existantes [60, 121, 145, 37]. Au niveau des données, elle s'intéresse aux instances. Le support de stockage considéré est la mémoire secondaire. Les approches existantes ignorent la mémoire principale car ils s'intéressent uniquement à la persistance des données. La figure 2.20 montre l'instanciation de la FH dans le modèle explicite proposé.

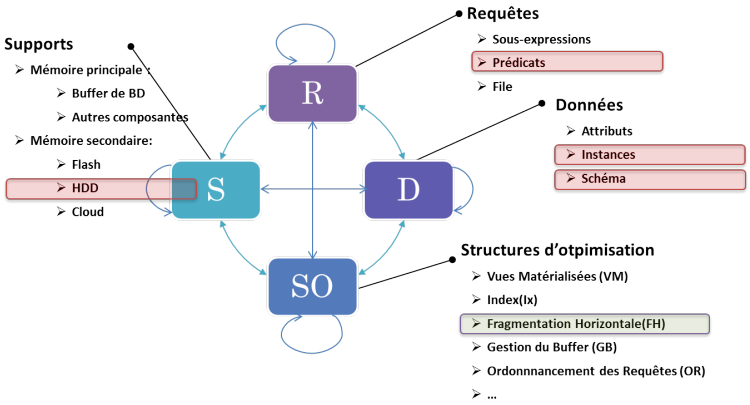

FIGURE 2.20 – Instanciation de la FH dans le modèle explicite de la conception physique

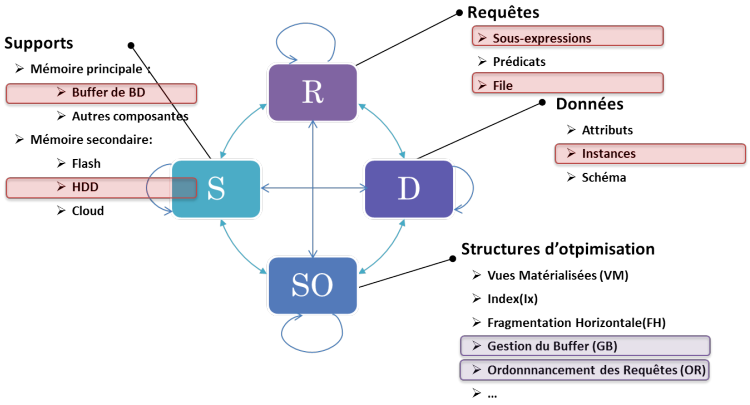

FIGURE 2.21 – Instanciation de la GBOR dans le modèle explicite de la conception physique

6.3.2.2 Sélection multiple : la GB et l'OR La gestion du buffer s'intéresse aux instances, à la mémoire principale (buffer) et aux sous-expressions partagées entre les requêtes. L'ordonnancement s'intéresse à l'ordre des requêtes dans la file, au contenu des supports de stockages (mémoire principale et/ou secondaire) ainsi qu'aux instances. Au niveau structure d'optimisation, la GB et l'OR interagissent car les deux problèmes comportent des similarités et des complémentarités. La principale complémentarité est que les sous-expressions communes gérées par le buffer sont indispensables pour définir l'ordre des

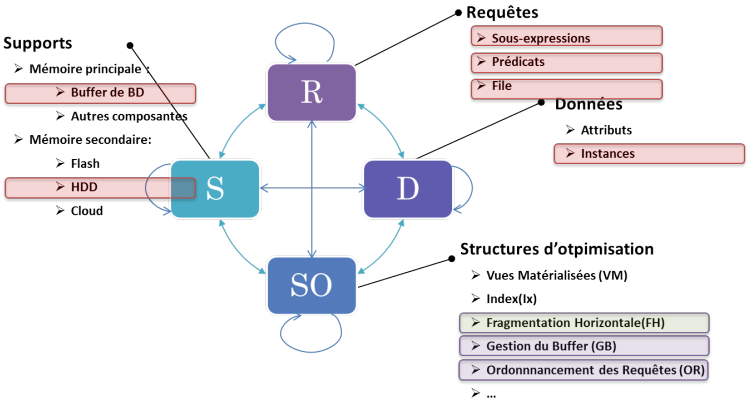

FIGURE 2.22 – Instanciation de la combinaison de la FH, la GB et l'OR dans le MECP

requêtes. D'autre part, l'ordre établi par l'ordonnanceur déterminera le contenu du buffer. La figure 2.21 montre l'instanciation de la GB et de l'OR dans le modèle explicite proposé.

6.3.2.3 Sélection multiple : FH, GB et OR Nous pouvons remonter l'interaction entre les structures d'optimisation à un niveau plus haut en considérant plusieurs structures d'optimisation. En considérant la FH, la GB et l'OR, nous pouvons toucher aux différents éléments des composantes : supports de stockage avec une mémoire principale et secondaire, les requêtes avec les sous-expressions et les prédicats, et les données avec leurs instances. Ainsi nous pouvons considérer tous les aspects pour avoir une optimisation *multi-niveaux*. La figure 2.22 illustre cette instanciation.

6.4 Aspects dynamiques et passage en-ligne

Dans les travaux existants, la conception physique a souvent été considérée comme statique. Autrement dit, le processus d'optimisation s'effectue hors-ligne. Dans quelques travaux récents, quelques approches ont été proposées pour rendre certaines techniques d'optimisation dynamiques. Dans ce cadre, la sélection d'un schéma de fragmentation horizontale a été revisitée par Dans Bellatreche *et al.* [26], pour prendre en compte l'aspect dynamique où la charge des requêtes peut changer. Dans le travail de Phan et Li [163], les auteurs ont étudié la sélection d'un ensemble de vues matérialisées de façon dynamique. Ainsi, les vues candidates seront matérialisées et/ou dématérialisées à la demande, selon le gain qu'elles apportent au système.

Cependant, l'adaptation dynamique des données et des requêtes, ou encore le changement de la tailles des structures déjà existantes, n'ont pas été considérés dans la formalisation de la conception physique. La conception physique doit être modélisée d'une façon permettant de gérer ses aspects dynamiques.

Pour cela, la dimension **temps** doit être intégrée afin de permettre de décrire l'état du système à l'instant T_i. Pour illustrer cet aspect, nous pouvons percevoir le système de bases de données comme un automate à mémoire. L'état initial représente l'état du système avant la conception physique (à l'instant T_0). Le passage vers un autre état se déclenche par un changement au niveau des données, des requêtes, des tailles de structures d'optimisation ou des supports. La figure 2.23 illustre le processus de conception

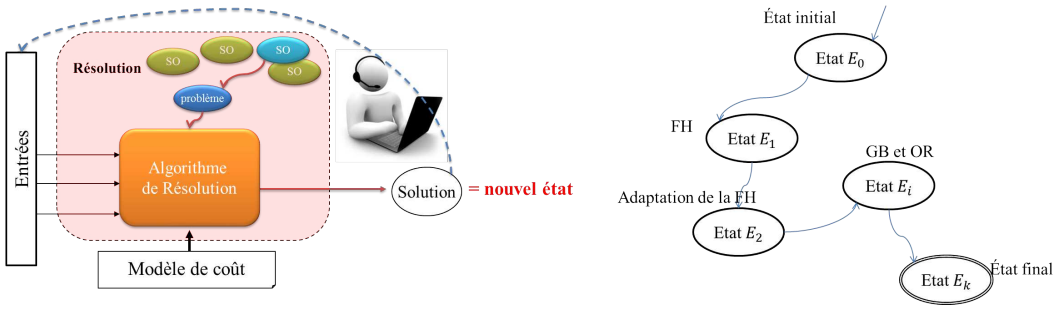

FIGURE 2.23 – Représentation des états du système par un automate

physique sur un système de base de données modélisé par un automate à mémoire.

7 Positionnement et contributions

Dans ce chapitre, nous avons présenté l'état de l'art sur la conception physique dans les bases de données. Dans notre étude, nous avons insisté sur l'importance de l'optimisation qui est au cœur de la conception physique et permet aux systèmes d'atteindre des bons niveaux de performance, afin de faire face aux évolutions et à la sophistication des systèmes de bases de données. Nous avons discuté quelques techniques d'optimisation, en donnant l'ensemble des travaux liés à chacune de ces techniques, dans les deux modes de sélection : isolé et multiple. Les techniques étudiées sont : la gestion du buffer, l'ordonnancement des requêtes et la fragmentation horizontale.

En étudiant la formalisation classique du problème de la conception physique, nous avons identifié deux types de paramètres : *les paramètres implicites et les paramètres explicites*. Alors que les derniers sont directement liés au problème de $\mathbb{CP}$, les premiers sont cachés, et ne sont ressortis que dans l'élaboration du modèle de coût. Cette lacune est commune dans les travaux de conception physique. Les paramètres implicites réduisent la visibilité lors du traitement de n'importe quelle structure d'optimisation. Dans la fragmentation horizontale, par exemple, toutes les approches proposées supposent que les requêtes ont la même probabilité de participer dans le processus de partitionnement. Ce qu'elles considèrent, c'est uniquement l'ensemble des prédicats de sélection, peu importe à quel requête appartiennent-ils, ni au contenu du buffer au moment de leur exécution [127, 38].

Pour la gestion du buffer et l'ordonnancement des requêtes, plusieurs travaux ont été proposés, majoritairement en mode hors-ligne. A notre connaissance, le travail de Phan *et al.*, traitant la matérialisation des vues dynamique avec l'ordonnancement des requêtes, est le seul travail à avoir considéré l'ordonnancement dans la conception physique en combinaison avec des techniques autres que la gestion du buffer. De même, très peu de travaux ont considéré la gestion du buffer avec d'autres structures d'optimisation courantes comme la fragmentation, les index ou encore les vues matérialisées.

Suite à ces constats, nous avons proposé un nouveau modèle de conception physique explicitant tous les paramètres à considérer ainsi que les composantes principales d'un système de base de données regroupant ses paramètres.

Au final, il en ressort de notre état de l'art, que chacune des techniques peut, et doit être, sélectionnée

en mode multiple pour une meilleure performance, car les structures d'optimisation se complémentent les unes les autres. Le manque d'efficacité des approches de conception physique existantes est principalement lié au manque de visibilité sur le système de BD causé par les paramètres implicites dans la formalisation classique. Avec l'explicitation des paramètres d'un système de BD, nous pouvons établir une sélection multiple permettant de considérer les différentes composantes à la fois.

Nous résumons l'ensemble de nos contributions liées à la nouvelle vision de la façon suivante :

- Étude du problème de la conception physique et identification du besoin d'explicitation des paramètres.
- Proposition d'un modèle explicite pour la conception physique.
- Étude d'un ensemble de techniques d'optimisation, en mode isolé et multiple : la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes.
- Considération de l'aspect dynamique de la conception physique.

En plus de ces contributions, nous présentons dans le reste de ce manuscrit, un ensemble d'algorithmes de résolutions pour chaque cas d'étude : la gestion du buffer et l'ordonnancement hors-ligne, la fragmentation horizontale, la combinaison de la fragmentation horizontale avec la gestion du buffer et l'ordonnancement des requêtes, et finalement, la gestion du buffer et l'ordonnancement des requêtes en-ligne. Nous décrivons également notre outil d'aide à la prise de décision que nous avons mis en place dans le but d'effectuer les différentes expérimentations liées à notre étude.

Chapitre 3

Interaction des composantes *Support* et *Requêtes*

Sommaire

1 Introduction

Les bases de données en général, et les entrepôts de données en particulier, sont des environnements qui favorisent l'interaction entre les requêtes. Cette interaction est due à la nature du schéma en étoile souvent utilisé pour concevoir les entrepôts. Le schéma en étoile contient plusieurs tables de dimensions indépendantes entre-elles et liées à une table des faits. Pour cela, les jointures des requêtes passent par la table des faits en augmentant la probabilité d'avoir des sous-expressions communes entre les plans de requêtes. Plusieurs travaux dans la littérature ont étudié ce phénomène pour réutiliser les résultats des sous-expressions communes afin de réduire le coût de certaines requêtes. Les travaux réalisés se sont focalisés sur la matérialisation des vues [199] et la gestion du buffer [89]. Cependant, la réutilisation des données pré-calculées est toujours liée à un autre problème complexe qui est l'ordonnancement des requêtes. L'ordonnancement permet d'exploiter le contenu dynamique du support de stockage, et c'est ainsi que l'interaction entre le support de stockage et les requêtes a été mise en évidence. Phan et Li ont combiné l'ordonnancement des requêtes avec le problème de sélection dynamique des vues matérialisées [163]. D'un autre côté, l'interdépendance entre le problème de l'ordonnancement des requêtes et la gestion du buffer a été considérée et étudiée dans quelques travaux récents [186, 111]. Il en ressort que la combinaison des deux problèmes est $\mathcal{NP}$-complète. Cette complexité est souvent contournée par différentes méthodes, en l'occurrence, les algorithmes génétiques [52], les processus Markoviens [108], etc.

A cause de la complexité élevée, nous étudions le problème combiné hors-ligne, i.e. les requêtes sont connues au départ et le processus d'optimisation est lancé avant le début d'exécution de la charge. Dans ce chapitre, nous formalisons le problème combiné, et donnons une étude de complexité pour prouver sa $\mathcal{NP}$-complétude au sens fort. Pour contourner la complexité du problème, nous donnons trois types d'algorithmes de résolution :

- Le premier type est celui des algorithmes simples qui utilisent un seul critère d'élagage de l'espace de recherche. Cet élagage consiste à écarter les solutions réduisant la performance du système. Dans cette catégorie, nous proposons l'algorithme dirigé par l'Affinité des requêtes.
- Le deuxième type est celui des méta-heuristiques inspirées des phénomènes naturels comme les algorithmes génétiques, le recuit simulé ou le hill climbing. Elles permettent de chercher des solutions optimales en un temps de calcul plus important.
- Le troisième est un compromis entre les deux premiers types. Pour cela, nous proposons une nouvelle approche "diviser pour régner", que nous appelons *Queen-Bee* afin de trouver une solution quasi-optimale en un temps raisonnable.

Ces types d'algorithmes découlent de notre analyse de l'état de l'art des problèmes d'optimisation en général, et de la fragmentation horizontale en particulier. Dans notre étude, nous avons souligné le fait que les approches ont évolué en cherchant à la fois un algorithme efficace et réactif, d'où la proposition des algorithmes dirigés par l'affinité et par un modèle de coût.

Les algorithmes proposés sont décrits en détails le long de ce chapitre. Chaque approche est validée par confrontation à l'expérimentation en utilisant notre outil de simulation qui sera présenté dans le chapitre 6. Les solutions proposées sont validées sous un SGBD Oracle. Nous testons également les performances de nos approches lors du passage à l'échelle, pour étudier leur efficacité dans les bases de données volumineuses.

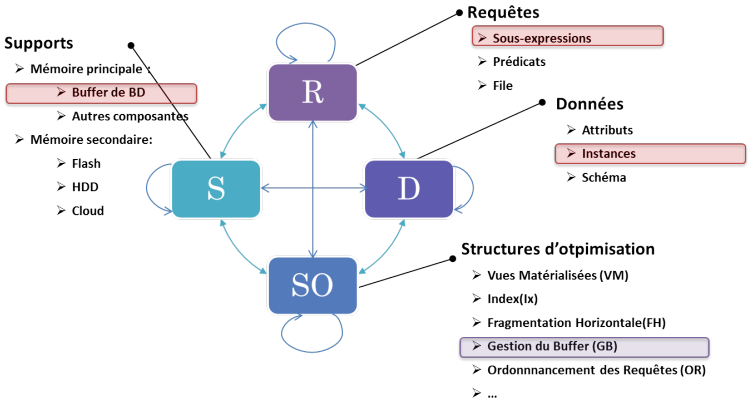

FIGURE 3.1 – Explicitation du problème de la gestion du buffer

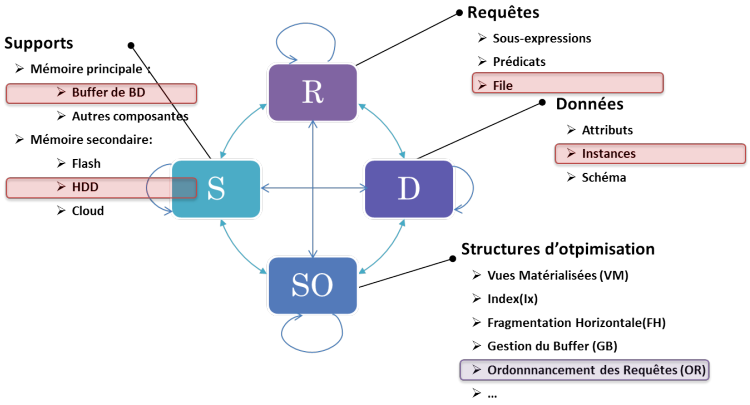

FIGURE 3.2 – Explicitation du problème de l'ordonnancement des requêtes

Comme nous l'avons exposé dans l'état de l'art, il existe plusieurs relations inter-composantes dans les systèmes de base de données. Les composantes que nous avons identifiées dans le modèle explicite pour la conception physique, interagissent deux à deux. Dans ce chapitre, nous étudions l'interaction entre la composante *Support* et la composante *Requêtes*. Pour cette étude, nous choisissons une sélection multiple de structures d'optimisation en combinant la gestion du buffer et l'ordonnancement des requêtes. Ce choix permet de prendre en considération le support de stockage via la gestion du buffer, et l'interaction des requêtes via l'ordonnancement. L'instanciation du problème de gestion du buffer et d'ordonnancement dans notre modèle est donnée respectivement dans les figures 3.1 et 3.2.

Ce chapitre sera organisé en cinq sections. La première section est consacrée à la présentation de la gestion du buffer et de l'ordonnancement des requêtes dans les entrepôts de données. Dans cette section, nous discutons les différents scenarii de combinaison des deux techniques, pour en ressortir le plus adapté. Ce scénario définira notre démarche de traitement des deux techniques, que nous détaillons dans la section suivante. Dans la deuxième section, nous présentons les motivations de notre choix du mode hors-ligne, la phase de préparation des entrées, les étapes de résolution simultanée, ainsi que les hypothèses de travail. La section trois présente le problème de gestion de buffer et d'ordonnancement hors-ligne de façon formelle. Nous présentons la formalisation des problèmes isolés et celle du problème

combiné. Une étude de complexité est menée afin de prouver que la combinaison des deux problèmes en mode hors-ligne est $\mathcal{NP}$-complète au sens fort. Nous proposons également une structure de données permettant de représenter les solutions potentielles, ainsi qu'un nouveau modèle de coût permettant de considérer le contenu du buffer et l'ordre des requêtes. Enfin, nous présentons notre politique de gestion de buffer adoptée. Dans la quatrième section, nous étalons les algorithmes proposés pour contourner le problème joint. L'évaluation des performances de ces algorithmes est présentée dans la cinquième section. Nous concluons ce chapitre par une discussion et un bilan général de l'étude.

2 La gestion du buffer et l'ordonnancement des requêtes dans les EDR

Le choix des EDR dans notre étude est lié principalement aux particularités qu'ils ramènent à savoir le schéma en étoile, le volume élevé et les requêtes de jointures en étoile. Ces particularités rendent les sous-expressions communes plus nombreuses et le besoin d'optimisation plus important.

Il existe plusieurs scenarii envisageables pour combiner la résolution du problème de la gestion du buffer à celle d'ordonnancement des requêtes, dans un contexte hors-ligne. Parmi ces scenarii, nous citons :

Scénario 1 : D'abord, résoudre le problème d'ordonnancement. Le planning d'exécution obtenu sera ensuite analysé par un algorithme pour la gestion du buffer. Cependant, dans ce scénario, l'aspect séquentiel impose l'ignorance du contenu du buffer par l'ordonnanceur, ce qui réduit la qualité du planning d'exécution retourné.

Scénario 2 : Résoudre le problème joint par une heuristique permettant de parcourir aléatoirement l'espace de recherche des deux problèmes à la fois. Dans ce scénario, l'espace de recherche des deux problèmes est très large (Section 4.2). Ceci est dû à l'interaction des différentes solutions entre-elles. La recherche utilisant les heuristiques gourmandes en temps de calcul est le seul moyen d'explorer un tel espace en l'absence de critères d'élagage, ce qui rend la solution non faisable dans la pratique. Or, les décideurs/concepteurs favorisent les techniques réactives, ce qui n'est pas le cas des méta-heuristiques.

Scénario 3 : Une politique de gestion de buffer (PGB) peut être définie au préalable. L'ordonnancement s'appuie sur la PGB, qui induit la politique d'allocation et de remplacement des données en cache. Une fois définie, cette politique permet d'aiguiller le processus d'ordonnancement selon le contenu du buffer. Néanmoins, une telle politique n'est pas suffisante pour fournir une gestion optimale du buffer. En effet, la PGB ne permet pas d'identifier les objets du buffer, élire les candidats, et de privilégier les plus bénéfiques selon certains critères comme : la taille de l'objet, la fréquence d'utilisation, le coût de son chargement à partir du disque ou encore sa dépendance d'un autre objet.

Scénario 4 : En analysant les trois scenarii précédents, nous avons proposé un nouveau scénario qui consiste à proposer une PGB puis à résoudre les deux sous problèmes simultanément. Dans l'algorithme de résolution, nous attribuons des critères d'identification et de choix d'objets candidats au buffer afin de renforcer la PGB et de l'adapter aux besoins de la charge. Contrairement aux scenarii cités précédemment, celui-ci est le plus prometteur car il permet de combler les lacunes liées à l'aspect séquentiel, à l'élagage et aux choix des candidats.

Le scénario 4 définit notre démarche de résolution de la GBOR dans le cas hors-ligne. Cette démarche

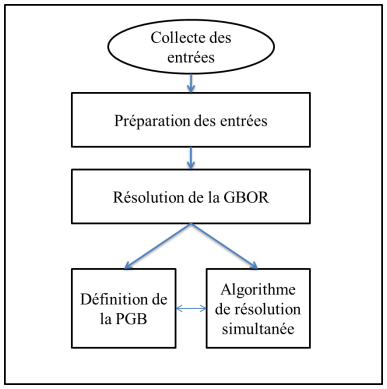

FIGURE 3.3 – Démarche de la résolution de la GBOR hors-ligne

consiste en trois étapes :

1. Collecte des entrées : bases de données, charge de requêtes et taille de buffer disponible;
2. Préparation des entrées : tracé des plans et génération du MVPP;
3. Résolution de la GBOR :
   - Définition de la politique de gestion du buffer;
   - Résolution simultanée avec des critères spécifiques.

Cette démarche sera employée dans la résolution du problème de GBOR. La figure 3.3 illustre chacune des étapes que nous décrivons dans la section suivante.

3 Démarche de la gestion du buffer et d'ordonnancement des requêtes dans les EDR

Dans cette section, nous expliquons la motivation de l'étude de la GBOR hors-ligne. Nous détaillons ensuite chaque étape de notre démarche, à savoir : la préparation des entrées et la résolution du problème.

3.1 Pourquoi le modèle hors-ligne?

La GBOR est un problème d'optimisation qui joint deux problèmes *NP*-complets (cf. Section 4.2). Cette complexité impose le traitement de la GBOR dans le cas statique, ce qui nous permet de ressortir les caractéristiques du problème de base et de ses algorithmes de résolution.

Dans ce chapitre, nous traitons la GBOR hors-ligne en proposant des techniques pouvant s'adapter aux cas les plus complexes. Ainsi, cette étude nous permettra d'analyser le problème statique avant de définir des méthodes prévues pour le passage en-ligne.

3.2 Préparation des entrées

Les entrées du problème de GBOR hors-ligne sont principalement : la base de données D, une charge Q et la taille du buffer B. Les objets du buffer sont tout résultat partiel d'une requête de la charge Q. Pour

illustrer l'ensemble d'objets candidats, nous présentons l'exemple suivant :

Exemple. Soit la requête $Q3.2$ suivante, définie sur l'entrepôt *Star Schema Benchmark* [157] (cf. Annexe 1) :

```
--Q3.2 :
select 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;
```

Supposons que l'optimiseur Oracle trace le plan de la requête $Q3.2$ de la façon suivante : (i) commencer par joindre les tables Lineorder et Supplier, ensuite (ii) le résultat de cette jointure sera joint avec la table Customer, et en fin (iii) la table Date. Ainsi, nous pouvons ressortir les sous-expressions résultant de cet ordre de jointure :

```
--Q3.2 :
select c_city, count(*) as counter
from(
select /*+result_cache*/ *--no3
from(
select /*+result_cache*/ *--no2
from(
select /*+result_cache*/*--no1
from lineorder, supplier
where lo_suppkey = s_suppkey and  s_nation = 'UNITED STATES'
)join customer on
lo_custkey= c_custkey
where c_nation = 'UNITED STATES'
)join date on
lo_orderdate = d_datekey
where d_year >= 1992 and d_year <= 1997
)
group by c_city,
order by counter desc;
```

Dans ce cas, le nœud no_1 du plan de la requête Q_1 représentant la jointure entre *lineorder* et *supplier*, produit un sous-résultat de requête pouvant être réutilisé par d'autre requêtes. Pour cette raison, ce résultat est considéré comme un objet buffer potentiel.

Dans la littérature, le problème d'identification des sous-expressions communes (nœuds) d'une charge de requêtes, pour l'optimisation (MQO) est un problème $\mathcal{NP}$-complet [112, 164, 118, 70]. De ce fait, notre étude ne considère pas le problème de sélection des plans de requêtes permettant de fournir les meilleures sous-expressions. Pour représenter la charge par un MVPP, nous exploitons les travaux établis au sein de notre équipe dans le cadre de la thèse d'Ahcene BOUKORCA [55], proposant une technique de construction de MVPP optimal par les algorithmes de construction de circuits électroniques. Cette approche a été communiquée et validée et son efficacité est considérable dans le cadre des requêtes de jointure en étoile.

La fixation du plan d'exécution pour chaque requête parmi tous ses plans possibles, permet d'effectuer un premier élagage de l'espace des objets candidats pour le buffer (cf. Section 4.2).

Nous proposons de tracer les plans d'évaluation des requêtes pour en construire le graphe MVPP [199]. Dans notre étude, nous raisonnons en terme de résultats intermédiaires représentés par des nœuds du MVPP, et non pas par des pages comme dans certains travaux de gestion du buffer, ce qui est plus bénéfique pour la charge globale [172]. Ainsi, la charge de requêtes en entrée du problème de la GBOR sera représentée par son MVPP.

Pour illustrer ce principe, nous considérons l'exemple initial (cf. page 2.6). Les nœuds partagés entre les plans (e.g. la jointure j_1 entre la table *Ventes* et *Temps*) sont des candidats au buffer.

3.3 Principales étapes du problème joint

L'établissement d'une gestion du buffer et d'un ordonnancement des requêtes nécessite plusieurs choix, à savoir : (1) La première requête à exécuter, (2) la politique de gestion du buffer, (3) les candidats à la mise en buffer et (4) la prochaine requête. Dans ce qui suit, nous allons décrire chacun de ces choix.

3.3.1 Choix de la première requête à exécuter

La première requête a un impact considérable sur la performance et le rendement de l'optimiseur car c'est elle qui initialise le contenue de la mémoire cache. Si la requête Q_i est choisie pour être exécutée en premier dans le Planning d'Exécution, la requête Q_j suivante sera choisie selon le contenu du buffer résultant de l'exécution de Q_i, ainsi qu'à d'autres critères de choix liés à l'algorithme de résolution. La séquence d'ordonnancement est donc dépendante de Q_i.

3.3.2 Choix de la politique de gestion du buffer

La politique de gestion du buffer comporte la politique d'allocation et de remplacement des données dans le buffer. Ces deux politiques sont définies en même temps pour permettre à la fois de :

- Choisir les pages à mettre dans le buffer et leurs emplacements (allocation);
- Choisir quelles pages enlever du buffer à chaque fois que le buffer est saturé (remplacement).

Le choix de la PGB permettra ainsi de définir le contenu du cache à chaque exécution. Notre PGB adoptée est décrite dans la Section 4.5.

3.3.3 Élection des candidats à la mise en buffer

L'élection nécessite l'identification des nœuds (résultats des sous-expressions) présents dans les plans des requêtes qui peuvent être mis dans le buffer. Parmi ces candidats, il faut identifier ceux qui sont requis par la charge à chaque exécution d'une requête.

Pour notre étude, les nœuds considérés sont en théorie tout sous-ensemble d'instances (n-uplets) de la Relation Universelle (RU) obtenue par la jointure de toutes les tables de la base. Cependant, dans notre étude, les nœuds candidats sont ceux représentant les résultats de jointures des sous-requêtes. Ce choix est lié au coût élevé de l'opération de jointure correspondant à la sous-expression et à sa fréquence notamment dans les entrepôts de données relationnels (EDR). Contrairement aux sélections, la jointure engendre plusieurs tables volumineuses (table des faits et plusieurs dimensions) ce qui rend son traitement plus lourd qu'une sélection. D'autre part, les jointures sont plus souvent accédées par les plans de requêtes dans un MVPP que les nœuds finaux (résultat final), ce qui les rend plus bénéfiques en cache.

Par exemple, les nœuds S_1 (sélection) et J_1 (jointure) dans le MVPP de notre exemple (cf. page 23) sont deux nœuds considérés, mais seul J_1 est candidat au buffer, car nous considérons seulement les jointures comme candidats dans notre étude.

Dans les travaux futurs, nous proposons d'étendre notre approche pour couvrir tout type d'opération algébrique dans l'élection des candidats.

3.3.4 Choix des prochaines requêtes

Selon l'ensemble des requêtes restantes, et le contenu du buffer, la prochaine requête à exécuter est déterminée par la stratégie d'ordonnancement choisie (propre à l'algorithme d'optimisation). Elle peut être sélectionnée de façon aléatoire (par une heuristique) ou par une technique prédéfinie (algorithme déterministe).

3.4 Hypothèses de travail

Dans notre étude, et à cause de la complexité du problème traité, nous limitons les possibilités liées à l'environnement. La multitude d'environnements (e.g. centralisé, distribué et parallèle), des requêtes (e.g. sélection et mise à jour.) ou encore des opérations (e.g. jointure par index, par hachage ou boucles imbriquées) rend la démarche plus complexe et nuit à la compréhension de l'approche générique.

Dans notre travail, nous considérons les hypothèses suivantes :

1. Les données de l'entrepôt sont représentées par un schéma en étoile comportant une table des faits F, et n tables de dimension : $\{D_1, D_2, \ldots, D_n\}$.
2. La charge considérée comporte des requêtes de jointures en étoile de la forme :

```
Select [att1, att2, .. attn] , AGR1, AGR2, .. AGRm
From F, D1, .. D_k
Where clé1=F.clé1 and clé2=F.clé2 and ... clék=F.clék
and P1 and ... Pp
[Group by att1, att2, .. attn]
```

```
[Order by att1, att2, .. attn][Dsc][Asc]
```

3. Les opérations de sélection sont poussées en bas du plan (*Push-Down Selection*) pour réduire la taille des résultats intermédiaires (*Rule Based Approach*).
4. Toutes les requêtes considérées ont la même priorité d'exécution.

4 Problème de la gestion du buffer et d'ordonnancement des requêtes hors-ligne

Dans cette section, nous présentons la formalisation du problème combiné. Nous présentons également une étude de complexité, dont nous établissons une preuve de $\mathcal{NP}$-complétude au sens fort du problème. Ensuite, nous proposons un codage des individus permettant de représenter les solutions potentielles dans nos approches. Le modèle de coût utilisé dans la résolution est décrit en détail, suivi de la politique de gestion du buffer adoptée.

4.1 Formalisation

En plus de la formalisation classique du problème de $\mathbb{CP}$, nous donnons la formalisation explicite selon le modèle proposé dans la Section 6 du chapitre 2. Afin de formaliser le problème combiné de la GBOR, nous procédons d'abord à une formalisation de chaque problème de manière isolée.

Les deux figures 3.1 et 3.2 donnent respectivement les instanciations de la gestion du buffer et l'ordonnancement des requêtes.

Le problème de gestion du buffer considère le buffer de la mémoire principale, indépendamment de la nature de la mémoire secondaire. D'autre part, la composante *données* est ignorée car le gestionnaire de buffer alloue la mémoire pour les objets candidats au cache indépendamment de leur nature (e.g. données, méta-données et index). Dans le cas de l'optimisation des requêtes, les objets concernés sont souvent les résultats des sous-expressions des requêtes. La file d'attente n'est pas une entrée pour le problème classique de GB. Toutefois, elle peut influencer le contenu du buffer à chaque instant.

Le problème d'ordonnancement des requêtes ignore les éléments de la composante *données* : le schéma et les attributs. L'ordre des requêtes dépend de la charge de requêtes et de la façon dont elles se présente dans la file d'attente. L'ordonnancement cherche à exploiter le contenu actuel des supports de stockage avant qu'il ne soit modifié. Cependant, les sous-expressions communes entre les plans d'exécutions des requêtes sont principale raison derrière le changement du contenu des supports. Ainsi, il est possible de prédire ce contenu en considérant ces sous-expressions.

Chacune des deux techniques peut s'exprimer de la façon suivante :

$$\textit{Problème de GB} =< \{sousexpression, \emptyset, \emptyset\}, \{\emptyset, instances, \emptyset\}, \{Buffer, \emptyset\}, \{GB\} >$$

$$\textit{Problème de OR} =< \{\emptyset, \emptyset, file\}, \{\emptyset, instances, \emptyset\}, \{Buffer, HDD\}, \{OR\} >$$

Cette formalisation montre non seulement les éléments requis pour chaque problème, mais aussi la complémentarité des deux techniques ce qui a motivé le traitement des deux problèmes de façon

conjointe. A partir des deux formalisations précédentes d'une part, et des scenarii étudiés dans la Section 2 d'autre part, nous formalisons le problème joint dans le cas hors-ligne de la façon suivante :

Étant données les entrées suivantes :
– Un schéma d'entrepôt de données relationnel D;
– Une charge de requêtes Q, représentée par son MVPP de l nœuds :$\{no_1, no_2, \ldots no_l\}$;
– Une politique de gestion du buffer PG_{buffer}.

Et soit la contrainte suivante :
– La taille maximale du buffer B disponible pour satisfaire la charge Q

Le problème de la gestion du buffer et d'ordonnancement des requêtes (GBOR) consiste à trouver une stratégie de gestion du buffer SGB et un planning d'exécution $\mathcal{P}$ tels que le le coût d'exécution de la charge de requêtes Q soit minimal.

4.2 Complexité

Dans cette section, nous prouvons que le problème de décision de GB est $\mathcal{NP}$-complet au sens fort, ceci implique qu'il n'existe pas d'algorithme polynomial ou pseudo-polynomial permettant de le résoudre, à moins que $\mathcal{P}$=$\mathcal{NP}$. La preuve de complexité est basée sur une transformation depuis le problème d'*Interval packing* (i.e., INTVPACK-WEIGHT) connu pour être $\mathcal{NP}$-complet au sens fort. Dans ce problème d'intervalles, l'objectif est de sélectionner le maximum d'intervalles pondérés, de telle façon qu'à aucun moment, le poids total des intervalles choisis ne dépasse un seuil donné [75]. Ce problème a été utilisé pour établir la preuve de complexité de plusieurs problèmes de mise en cache pour pages de tailles multiples. Dans ce qui suit, nous donnons une définition formelle de ce problème pivot.

Considérons N intervalles (s_i, t_i) $1 \leq i \leq N$ avec leurs poids w_i. Pour tout sous-ensemble $S \subseteq \{1, \cdots, N\}$, le poids de S est $w(S) = \sum_{i \in S} w_i$ et une coupure à l'instant γ est un ensemble d'intervalles qui contient γ: $cut_\gamma(S) = \{i : s_i < \gamma < t_i\}$. Le problème d'INTVPACK-WEIGHT consiste à choisir le sous-ensemble d'intervalles ayant un poids maximal qui satisfait la contrainte : $w(cut_\gamma(S)) \leq W$.

– *Problème:* INTVPACK-WEIGHT [75]
– *Instance:* Un ensemble de N intervalles ouverts (s_i, t_i) pour $i = 0, \cdots, N-1$, où chaque intervalle i a un poids $w_i \geq 0$. Soient les deux entiers positifs W et L.
– *Question:* Existe-t-il un sous-ensemble S d'intervalles avec $w(S) \geq L$ qui satisfait : $w(cut_\gamma(S)) \leq W$ pour tout nombre réel γ?

Le problème INTVPACK-WEIGHT a été prouvé $\mathcal{NP}$-Complet au sens fort en utilisant une transformation du problème de couverture de sommets (i.e. VERTEX COVER) (voir [104]).

Théorème 1

[75] INTVPACK-WEIGHT est $\mathcal{NP}$-Complet au sens fort.

Nous considérons maintenant le résultat de complexité du problème de Gestion du Buffer. La réduction suit les principes utilisés dans [75] pour prouver la $\mathcal{NP}$-Completude du problème de cache pour des pages de tailles multiples.

Nous considérons le problème de décision correspondant à la GB. A des fins de clarté, nous supposons (pour l'étude de complexité seulement) que le coût de chargement d'un nœud non caché est égal à

sa taille.

- *Problème :* GB : Problème de Gestion du Buffer;
- *Instance :* Un ensemble de requêtes $\{Q_1, Q_2, \dots Q_m\}$ avec un ensemble $N = \{r_1, r_2 \dots r_l\}$ de nœuds dans leur plan d'exécution avec un coût et une taille $Cout(n_i) = Taille(n_i)$; $(1 \leq i \leq l)$, une séquence d'opérations pour un plan d'exécution $r_1, r_2, \dots r_m \in N$, une taille du buffer B et un seuil de coût F.
- *Question:* Existe-t-il une politique de remplacement qui serve $r_1, r_2, \dots r_m$ avec une taille buffer de B et un coût d'au plus F?

Théorème 2

GB est un problème $\mathcal{NP}$-Complet au sens fort.

Démonstration : GB est dans la classe $\mathcal{NP}$, car un algorithme non-deterministique (i.e., *Non-Deterministic Turing Machine*) nécessite uniquement de connaitre le sous-ensemble de nœuds à cacher et ensuite de vérifier en un temps polynomial si le coût total est au plus F. Nous construisons une instance de GB à partir de l'instance générique d'*INTVPACK-WEIGHT*. Pour tout intervalle i, dénoté (s_i, t_i), nous définissons un nœud n_i, $0 \leq i \leq N-1$ avec une taille $Taille(n_i) = w_i$ et un coût $Cout(n_i) = Taille(n_i)$. La séquence d'opérations générée par les requêtes consiste en $2N$ demandes :$r_1, \cdots, r_{2N}$. Chaque nœud n_i est demandé deux fois: la première fois à l'instant s_i et la seconde à l'instant t_i. La taille du cache est $B = W$ et le seuil du coût est $F = 2\sum_{i=0}^{N-1} w_i - L$.

La transformation précédente est polynomial car elle est calculée en $O(N)$. Pour compléter la preuve, nous avons besoin d'établir que : *INTVPACK-WEIGHT* possède une solution si, et seulement si, GB a une solution de la construction précédente.

(Si) Supposons que INTVPACK-WEIGHT a une solution S. Nous prouvons maintenant que la solution correspondante est valide pour GB. Durant la séquence d'opérations, seul le nœud associé à la solution S sera chargé dans le cache : n_i est chargé à l'instant s_i et ôté à l'instant t_i après sa deuxième et dernière utilisation. La condition $w(cut_\gamma(S)) \leq W$ pour la solution S d'INTVPACK-WEIGHT assure que les nœuds chargés ne dépassent jamais la capacité du cache $B = W$. Le coût total est défini par : chaque nœud dans S est chargé une seule fois dans le cache ajoutant un coût $w(S)$, tandis que les nœuds qui ne sont pas dans S, notés $\bar{S}$ sont chargés deux fois en mémoire secondaire en ajoutant un coût $w(\bar{S})$. Comme S est une solution d'INTVPACK-WEIGHT, nous avons $w(S) \geq L$. Du fait que $N = S \cup \bar{S}$, le coût total est majoré par $F = w(S \cup \bar{S}) \leq 2w(N) - L = 2\sum_{i=0}^{N-1} w_i - L$. Ainsi, une solution valide au problème de GB est retrouvée.

(Seulement Si) Supposons que GB a une solution. Alors, nous prouvons qu'une solution valide peut être définie pour le problème d'INTVPACK-WEIGHT. Chaque nœud est requis deux fois. Supposons que S est définie par un ensemble de nœuds qui ne génèrent pas de défaut de page quand ils sont demandés la deuxième fois. De tels nœuds sont forcément dans le cache. Supposons que S définit un sous-ensemble de N intervalles correspondant à ces nœuds. Du fait que le coût total est majoré par $F = 2\sum_{i=0}^{N-1} w_i - L$, ceci implique que $w(S) \geq L$ et par conséquent la première contrainte d'INVPACK-WEIGHT est satisfaite. Deuxièmement, les nœuds $i \in S$ sont stockés dans le cache toute la durée de l'intervalle ouvert (s_i, t_i) sans générer un défaut de page. Ainsi, la capacité du cache C n'est jamais dépassée. Comme $B = W$, le poids de la coupure à tout instant γ satisfait la contrainte : $w(cut_\gamma(S)) \leq W$. Alors, nous avons défini une solution valide pour le problème décisionnel d'INTVPACK-WEIGHT. ■

Théorème 3

Le problème de Gestion du Buffer et d'Ordonnancement de Requêtes hors-ligne et en-ligne est un problème $\mathcal{NP}$-Complet au sens fort.

Notons que le problème d'ordonnancement des requêtes hors-ligne a été prouvé $\mathcal{NP}$-complet dans [111].

4.3 Représentation des solutions potentielles

Le traitement de n'importe quel problème d'optimisation nécessite une représentation des solutions potentielles afin de pouvoir les manipuler dans les algorithmes. Le codage est donc une étape primordiale pour passer de l'espace des solutions réelles à l'espace des individus manipulables par un algorithme. Il permet la représentation des solutions de façon compréhensible par le langage machine d'une part, et l'application de transitions pour l'amélioration sur cette solution d'autre part.

Dans le problème de la gestion du buffer et d'ordonnancement des requêtes, la solution en question est composée de deux parties en interaction : l'ordre des requêtes, et l'état du cache à chaque fois que l'une des requêtes est exécutée. L'ordre des requêtes définit le planning d'exécution de la charge, et les états du cache constituent la stratégie du buffer associée au Planning.

Pour cela nous avons proposé une représentation combinée de deux structures en interaction :

1. Un vecteur $\mathcal{P}$ représentant l'ensemble des n requêtes de la charge;
2. Chaque cellule du vecteur $\mathcal{P}$, contenant une requête Q_i, est reliée à un vecteur de valeurs $j \in \{0, 1\}$ dit *bitmap* de longueur l, où l est le nombre de nœuds candidats de la charge contenue dans le MVPP. Ces bitmaps sont indexés par les nœuds du MVPP. Les valeurs 0 et 1 dans la cellule du bitmap indiquent respectivement l'absence ou la présence du nœud dans le buffer après l'exécution de la requête Q_i correspondante. Une solution finale pour le scénario du buffer est représentée par juxtaposition de ces bitmaps, notée BM.

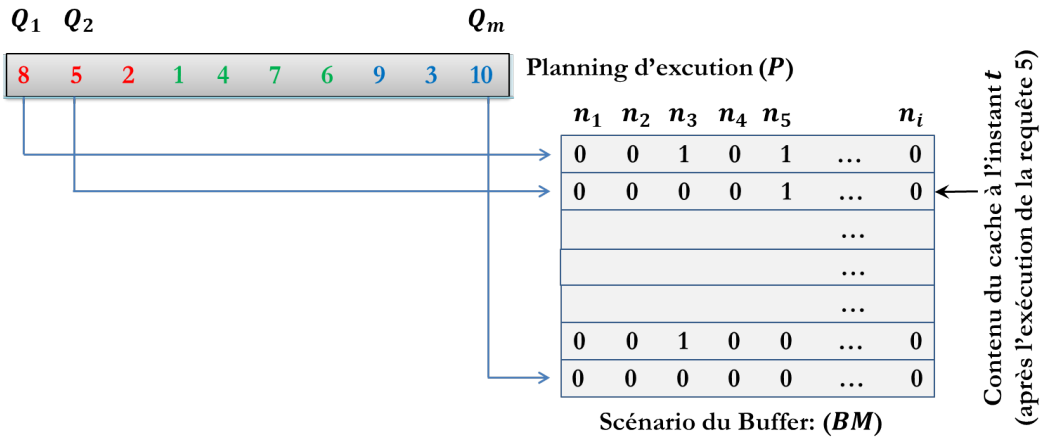

FIGURE 3.4 – Codage des solutions potentielles de la GBOR

Ainsi, une solution de la GBOR est représentée sous forme d'un couple : $(\mathcal{P}, BM)$.

La figure 3.4 présente un exemple de la GBOR représentée par notre codage. La sémantique de cette représentation est la suivante : le vecteur à gauche est le planning d'exécution $\mathcal{P}$ de la file d'attente de

la charge. La matrice de droite est l'état du buffer BM après chaque exécution. Les requêtes sont alors exécutées dans l'ordre spécifié par le vecteur $\mathcal{P}$. À chaque fois qu'une requête est lancée, elle induit un changement du contenu du buffer, en enlevant des nœuds et en cachant d'autres.

Exemple. Considérons un buffer initialement vide.

Dans le codage précèdent (cf. figure 3.4), l'exécution de la première requête Q_1 (8 dans le MVPP) déclenche la mise en cache de deux résultats de calcul : le résultat du nœud n_3 et celui du nœud n_5. La requête Q_2 (5) s'exécute après et utilise les nœuds pertinents existant en cache. La requête Q_2 (5 dans le MVPP) référence d'autres nœuds utiles pour le reste de la charge, mais qui ne peuvent être mis en cache faute d'espace. Pour contourner le problème de saturation du cache, le gestionnaire du buffer remplace les données en cache en enlevant le nœud n_3. La requête suivante est exécutée de la même manière, et ainsi de suite jusqu'à la fin de la file.

L'exécution de la charge dans un ordre donné en file d'attente provoque un changement dynamique de l'état du cache depuis la première requête jusqu'à la dernière, et par conséquent, des performances variables selon l'ordonnanceur et la stratégie du cache.

L'ensemble des algorithmes que nous proposons pour la gestion du buffer et l'ordonnancement, utilise ce codage pour la représentation et la manipulation des solutions potentielles.

4.4 Modèle de coût pour la gestion du buffer et l'ordonnancement des requêtes

Un modèle de coût est une fonction permettant d'évaluer la qualité d'une solution à un problème donné. Une telle métrique est indispensable pour évaluer la performance d'une solution donnée, sans avoir à la déployer sur un système de base de données réel.

à partir de notre analyse du problème de la fragmentation horizontale et de l'évolution de ses approches de résolution, nous avons constaté l'importance que les travaux récents ont accordé au modèle de coût. Les solutions basées sur le modèle de coût offrent des performances élevées. L'évaluation de la qualité d'une solution pour notre problème d'optimisation, a pour but de déterminer le coût d'une charge de requêtes en termes d'Entrées/Sorties, de consommation énergétique, d'opération CPU ou d'espace de stockage. Dans les systèmes existants, les supports de stockage ont évolué et se sont diversifiés. Les dernières générations présentent des coûts (prix) moins élevé et des capacités de stockage plus importantes. De ce fait, les entreprises s'intéressent moins au coût de stockage, mais elles misent sur la vitesse de traitement et aux nombres d'entrées/sorties.

Nous proposons un modèle de coût permettant d'estimer le nombre d'Entrées/Sorties (E/S) requises pour l'exécution de la charge en tenant compte de l'ordre d'exécution des requêtes et du contenu du buffer. La Table 3.1 énumère les paramètres employés pour le calcul de coût, avec leur description.

4.4.1 Estimation basique du nombre d'entrées/sorties

Le coût total de la charge est donné par la somme des coûts des requêtes exécutées :

$$Cout\_Total(\mathbf{Q}) = \sum_{i=1}^{n} Cout(Q_i) \tag{1}$$

Paramètre	Description
F	Table des faits
D_i	Table de dimension
PJ	Première jointure dans le plan d'exécution de la requête
JI	Jointure intermédiaire dans le plan de la requête
AG	Opération finale regroupant les agrégats, les tris et les projections
Res_i	Le i-ième résultat intermédiaire dans le plan de la requête
Ref_k	Le nombre d'instances chargées de la relation k/ ($k \in \{F, D_i, Res_i\}$)
$Share_i$	Share de la dimension D_i avec la table F
Fs^i_j	Facteur de sélectivité de l'attribut att_i pour le prédicat P_j
noeud	Nœud dans le plan de la requête, représentant une jointure (PJ, JI) ou une opération finale (AG)
racine	Nœud dans le plan de la requête, représentant l'opération finale (AG)
jointure_precedente	Opération de jointure précédant l'opération en cours

TABLE 3.1 – Les paramètres employés dans le modèle de coût

Le coût d'exécution de la i-ième requête Q_i dépend des opérations qu'elles effectuent, de la taille des pages qu'elles chargent et du contenu du buffer durant son exécution. Comme nous l'avons déjà mentionné (Section 3.4), les requêtes considérées sont des requêtes de jointure en étoile de la forme :

```
Select [att1, att2, .. attn,] AGR1, AGR2, .. AGRm
From F, D1, .. D_k
Where clé1=F.clé1 and clé2=F.clé2 and ... clék=F.clék
and P1 and ... Pp
[Group by att1, att2, .. attn,]
[Order by att1, att2, .. attn ][Dsc][Asc]
```

La requête de jointure en étoile nécessite l'exécution de plusieurs opérations consécutives. Nous distinguons les 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.

L'ordre des opérations est défini par le plan d'évaluation optimal de la requête. Le meilleur plan d'évaluation consiste à effectuer en premier les opérations ayant le résultat intermédiaire de plus petite taille [52]. Ainsi, l'ordre des jointures est défini selon la taille des résultats intermédiaires. D'autre part, dans le contexte d'entrepôt de données, toutes les jointures (et en particulier la première) impliquent la table centrale du schéma en étoile (table des faits). Voici l'ordre d'exécution des jointures pour la requête Q_i :

$$F \bowtie D_1 \bowtie D_2 \bowtie \cdots \bowtie D_n \tag{2}$$

Où D_j est la j-ième table de dimension et n est le nombre de jointures dans le plan d'évaluation de Q_i. Notons que n est inférieur ou égal au nombre de tables de dimension de la base.

En analysant la structure des requêtes de jointure en étoile, nous constatons que le coût d'exécution de Q_i est obtenu par la somme de trois coûts élémentaires C_{PJ}, C_{JI} et C_{AG} [52]. Dans notre étude, nous supposons que la jointure est effectuée par hachage, et que les index sont ignorés. Ces coûts de jointures sont calculés ainsi [101] :

$$C_{PJ}(noeud) = 3(\psi(F) + \psi(D_1)) \tag{3}$$

$$C_{JI_i}(noeud) = 3(\psi(Res_i) + \psi(D_{i+1})) \tag{4}$$

$$C_{AG}(noeud) = 2 \times (AGR + GB)(|Res_n|) \tag{5}$$

Où AGR (respectivement GB) est égal à 1 si une agrégation (resp. groupement) est effectuée, et 0 sinon. Soit R une relation ou une sous-relation résultant d'un ensemble d'opérations antérieures. La fonction $\psi(R)$ fournit les E/S requises pour le chargement de R. Nous utilisons la fonction *Yao* [195] qui permet d'estimer le nombre de pages contenant une instance pertinente pour une opération algébrique donnée, en utilisant les lois de probabilité pour déterminer si une instance pertinente pour une opération, est présente dans une page donnée. Cette fonction utilise trois paramètres pour chaque opération : le nombre de pages de la relation R concernée par la lecture, le nombre d'instances total dans R et le nombre d'instances pertinentes de la relation R. La fonction permet de fournir une estimation du nombre de pages à charger, contenant toutes les instances pertinentes.

Définition 2

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. Ce Share représente le nombre d'instances de la table des faits en relation avec une instance de cette dimension. Notons que dans le cas d'une répartition uniforme des données dans les tables, on a :

$$Share_{D_i} = \frac{||F||}{||D_i||}$$

La notion de Share permet d'expliciter la valeur de $\psi(R)$:

$$\psi(R) = \begin{cases} yao(||F||, |F|, Share_{D1} \times Ref_{D1}) & \textit{Si R est la tables des faits} \\ yao(||D_i||, |D_i|, Ref_{Di}) & \textit{Si R est une table de dimension } D_i \\ yao(||Res_i||, |Res_i|, Ref_{Res_i}) & \text{Si R est un résultat intermédiaire } Res_i \end{cases} \tag{6}$$

Le nombre d'instances chargées d'une table de dimension lors d'une sélection est donné par le produit des facteurs de sélectivité des prédicats et la cardinalité de la dimension :

$$Ref_{D_i} = ||D_i|| \times \prod_{j>0} Fs_j^i \tag{7}$$

Où Fs_i^j est le facteur de sélectivité de l'attribut j de la table D_i. D'autre part, le nombre d'instances chargées à partir d'un résultat intermédiaire dépend des opérations précédentes dans le plan de la requête.

Pour calculer la taille du résultat intermédiaire, nous utilisons le *Share* de la dimension et la proportion des faits retournés par l'opération précédente.

$$Ref_{Res_i} = \lceil \frac{Share_{Di+1} \times Ref_{Di+1} \times Ref_{Di}}{||F||} \rceil \tag{8}$$

4.4.2 Impact de la mémoire cache dans le calcul du coût

Nous prenons maintenant en considération le contenu du cache. Le coût se calcule récursivement sur les nœuds. Le calcul commence à partir du nœud racine du plan qui représente l'opération finale. Le coût de chaque nœud dépend des nœuds fils représentant les opérations algébriques précédant ce nœud. Le calcul récursif s'arrête aux nœuds feuilles (les tables de la base). Plus formellement :

$$Cout(Q_i) = C(racine) \tag{9}$$

La fonction $C(no_i)$ estime le coût en termes d'E/S requises pour l'exécution du nœud no_i dans le plan d'exécution. Le contenu du buffer est vérifié chaque fois qu'une opération est sur le point d'être exécutée pour savoir si les données doivent (ou non) être chargées à partir du disque.

$$C(racine) = \begin{cases} 0 & \textit{si la racine est présente dans le buffer} \\ \theta(racine) + C(jointure\_precedante) & \textit{sinon} \end{cases} \tag{10}$$

Finalement, $\theta(no_i)$ fournit le coût de traitement du nœud no_i en cours en utilisant les formules préalablement décrites dans ce modèle. Plus formellement :

$$\theta(noeud) = \begin{cases} C_{PJ}(noeud) & si\ noeud = PJ \\ C_{JI}(noeud) & si\ noeud = JI \\ C_{AG}(noeud) & si\ noeud = AG \end{cases} \tag{11}$$

Notons que le nœud racine correspond à l'opération finale : $C(racine) = C_{AG}(racine)$.

Le différentiel par rapport au contexte précédent (estimation basique), réside dans la vérification du contenu du buffer avant l'exécution d'une opération algébrique. Ainsi, l'opération de jointure (PJ ou JI) n'ajoute aucun coût supplémentaire si le résultat du nœud correspondant est présent dans le cache.

4.5 Politique de gestion du buffer adoptée

Comme nous avons vu dans l'état de l'art, les stratégies de gestion du buffer souvent utilisées comme LRU ne sont pas adaptées aux bases de données en général, et aux entrepôts de données en particulier [101]. Pour cela, nous proposons une gestion du buffer qui permette de satisfaire un plus grand nombre de requêtes avec une taille de buffer limitée, en considérant la charge globale.

Tout gestionnaire de buffer a besoin de définir une politique d'allocation et une de remplacement des données [6, 182]. Il existe plusieurs modes d'allocation et de remplacement. Dans notre travail, et de par la complexité de nos problèmes traités, nous raisonnons par résultats intermédiaires (nœuds), et nous

nous intéressons aux nœuds à conserver dans le buffer, et à quel moment les retirer. Nous proposons une politique dite *Dynamic Buffer Manager* (DBM) permettant à la fois l'allocation et le remplacement des données dans le buffer selon l'ensemble des requêtes à optimiser.

a. Politique d'allocation

L'allocation de données dans le cache consiste à trouver de l'espace pour y loger des pages. Le gestionnaire vérifie pour chaque opération sur le point d'être exécutée, si son résultat existe déjà en cache. Dans le cas contraire, et si l'espace disponible le permet, le gestionnaire met ce résultat en cache.

Nous supposons dans ce travail qu'un nœud ne peut être caché qu'une seule fois durant l'exécution de la charge.

Le choix d'allocation est défini par trois facteurs :

1. L'ordre d'arrivée des requêtes, qui est fourni par l'ordonnanceur;
2. L'ensemble des nœuds candidats au cache, qui est déterminé par l'algorithme de GBOR, et en dehors de la politique d'allocation;
3. La taille du buffer disponible.

Ainsi, le choix qui reste à faire par la politique d'allocation est de décider dans quel ordre allouer le buffer pour plusieurs candidats d'une même requête. Dans notre travail, nous proposons de traiter les nœuds selon l'ordre des opérations correspondantes dans le plan de la requête.

b. Politique de remplacement

Si l'espace disponible ne permet pas de cacher le résultat intermédiaire, un ou plusieurs résultats doivent être retirés afin de laisser la place pour d'autres données. Le remplacement est une opération plus délicate que l'allocation, car le choix des pages à ôter est plus difficile selon l'utilité et le nombre de nœuds occupant le buffer.

Nous utilisons un critère appelé *Utilité*, qui représente la participation d'un nœud en cache dans les requêtes restantes en file d'attente.

Soit une charge constituée d'un ensemble de plans d'exécution contenant chacun plusieurs nœuds.

Définition 3

Nous appelons Utilité d'un nœud no_i, noté $U(no_i)$, le nombre de requêtes à venir accédant no_i.

Définition 4

Soit la relation d'ordre stricte $\subset_*$ entre deux nœuds, appelée : Relation d'inclusion planaire. On dit que :

$$no_i \subset_* no_j$$

si et seulement si la sous-expression correspondant au nœud no_j contient strictement la sous-expression du nœud no_i. On dit également que no_j couvre no_i.

Démonstration : La relation d'inclusion planaire, notée $\subset_*$, est une relation binaire définie sur l'ensemble des nœuds $\mathcal{N}$ du MVPP.

- $\forall no_i \in \mathcal{N} : no_i \not\subset_* no_i$ car une sous expression ne peut être strictement incluse en elle-même.
- $\forall (no_i, no_j, no_k) \in \mathcal{N}^3 : (no_i \subset_* no_j) \wedge (no_j \subset_* no_k) \Rightarrow (no_i \subset_* no_k)$ car si une sous-expression i couvre une autre, alors elle couvre également toutes les sous-expressions de contenues dans cette

dernière.
La relation $\subset_*$ est donc irréflexive et transitive, d'où $\subset_*$ est une relation d'ordre strict. ■

Définition 5

Un nœud no_i est dit postérieur à un nœud no_j si et seulement si : $no_j \subset_* no_i$

Définition 6

Un nœud no_i est dit antérieur à un nœud no_j si et seulement si : $no_i \subset_* no_j$

Nous notons $UR(no_i)$, l'ensemble des requêtes dont le nœud no_i apparaît dans l'arbre algébrique.

Définition 7

Nous notons $Buffer(no_i)$ la propriété :

"le nœud no_i est présent dans le buffer"

. Un nœud no_i est dans le buffer si toutes ses pages y sont.

L'*Utilité* d'un nœud avant l'exécution des requête de la charge est égale au nombre de requêtes qui l'utilisent. A chaque utilisation, cette valeur est décrémentée de 1. Un nœud devient *inutile* dès lors que :

$$(Buffer(no_i) = vrai) \wedge (U(no_i) = 0)$$

Une fois l'*Utilité* s'annule pour un nœud no_i ($U(no_i) = 0$), ce nœud est retiré du cache car il ne servira aucune requête plus tard, en libérant ainsi de l'espace dans le buffer. Notons que tout nœud no_j *postérieur* à no_i dans le MVPP aura une utilité nulle. Plus formellement :

$$(U(no_i) = 0 \wedge \exists no_j \in \mathcal{N} : no_i \subset_* no_j) \rightarrow U(no_j) = 0$$

5 Algorithmes de résolution hors-ligne

En raison de la complexité du problème joint de la gestion du buffer et de l'ordonnancement des requêtes, et en analysant les travaux de la conception physique, deux classes d'algorithmes sont envisageables :

- Algorithmes simples, ayant une complexité algorithmique réduite et une efficacité limitée;
- Algorithmes lourds, qui sont efficaces mais avec un temps de traitement élevé.

Dans le but d'étudier les caractéristiques des algorithmes de résolution, et de parvenir à un compromis entre l'efficacité et la complexité algorithmique, nous proposons des algorithmes dans chaque classe.

Dans la catégorie des algorithmes simples, les approches proposées se basent sur un critère donné pour élaguer l'espace de recherche, tout en réduisant la complexité algorithmique. Dans cette catégorie, nous proposons un nouvel algorithme dirigé par l'affinité entre les requêtes. Cette affinité, définie dans la section qui suit, représente la corrélation entre les requêtes.

Dans la deuxième catégorie, des méta-heuristiques sont utilisées dans les problèmes d'optimisation combinatoires afin d'explorer un espace de recherche plus grand que les algorithmes simples. Ces algorithmes sont majoritairement inspirés de phénomènes naturels. On peut distinguer des algorithmes basés

sur les populations comme les algorithmes génétiques (AG), ou sur le parcours comme le Hill Climbing (HC), ou entre les deux comme le recuit simulé (RS). Parmi ces heuristiques, nous utilisons le Hill Climbing et un Algorithme Génétique qui ont déjà prouvé leur efficacité dans ce genre de problèmes, notamment les problèmes d'optimisation combinatoires [52].

Dans notre étude, nous avons visé ces deux classes d'algorithmes, pour tirer profit de leurs caractéristiques dans le but de trouver un compromis entre l'efficacité et la complexité algorithmique. L'étude de ces algorithmes nous a permis de proposer un nouvel algorithme, dit *Queen-Bee* (QB). Cet algorithme permet d'atteindre un haut niveau d'efficacité en un temps raisonnable grâce à sa complexité réduite.

Dans ce qui suit, nous allons détailler chaque algorithme, et faire une évaluation théorique pour montrer leur efficacité et leur complexité. Pour représenter et manipuler les solutions potentielles, les algorithmes proposés utilisent la structure de données décrite en Section 4.3. Le buffer est supposé vide initialement.

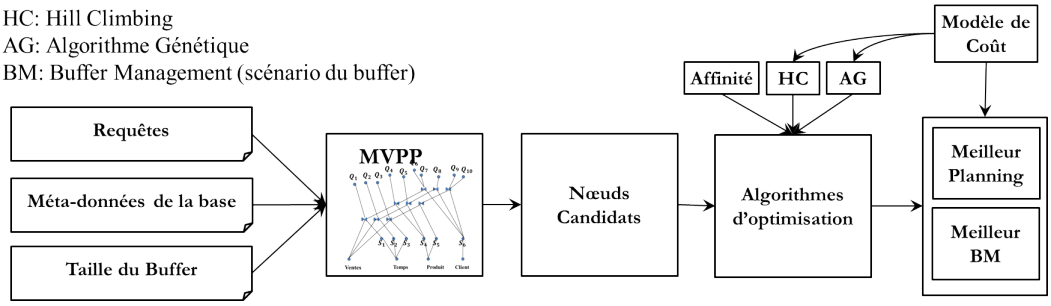

FIGURE 3.5 – La méthodologie de résolution de la GBOR hors-ligne

5.1 Algorithme d'Affinité des requêtes

Dans la classe des algorithmes simples de la GBOR, un algorithme est basé sur un critère d'élagage de l'espace de recherche. Ce critère est nécessaire pour piloter le processus d'optimisation, mais il n'est pas suffisant en général. L'avantage de ce type d'algorithmes, est la complexité algorithmique réduite.

Le principal critère à considérer dans la GBOR est l'interaction entre les requêtes. En effet, comme nous l'avons exposé dans l'état de l'art (cf. chapitre 2), les requêtes dans un EDR présentent beaucoup de sous-expressions communes. La réutilisation de ces sous-expressions permet d'éviter la reprise du calcul déjà réalisé, et donc de gagner en performance. L'affinité entre les requêtes permet d'ordonner les requêtes de façon à maximiser cette réutilisation. C'est pourquoi, nous proposons un algorithme dirigé par l'affinité entre les requêtes afin d'établir un planning d'exécution.

5.1.1 Principe de l'algorithme d'Affinité

Afin de décrire le principe de notre algorithme d'Affinité, nous commençons par donner les définitions suivantes :

Définition 8

Nous notons $Plan_i$ la une succession de nœuds appartenant au plan d'exécution de la requête Q_i sur

le SGBD hôte.

$$Plan_i = \{no_1, no_2, \ldots no_k\}/k \leq l$$

Soit une charge de n requêtes $\mathcal{Q} = \{Q_1, Q_2, \ldots Q_n\}$. Chaque requête Q_i est représentée par son plan d'exécution $Plan_i$.

Définition 9

L'affinité entre deux requêtes Q_i et Q_j, notée $\mathcal{AFF}(Q_i, Q_j)$, représente le nombre de sous-expressions communes entre leurs plans respectifs $Plan_i$ et $Plan_j$.

$$\mathcal{AFFR}(Q_i, Q_j) = \|Plan_i \cap Plan_j\|$$

Définition 10

L'affinité réflexive d'une requête est le nombre de ses sous-expressions en commun avec d'autres requêtes de la charge.

$$\mathcal{AFFR}(Q_i) = Max(\mathcal{AFF}(Q_i, Q_k)_{k\in[1,n],k\neq i})$$

L'algorithme d'Affinité commence par construire la matrice des affinités entre les requêtes. Dans cette matrice, le coefficient (i, j) contient le taux d'affinité entre les requêtes Q_i et Q_j. Cette matrice est donc symétrique. En fonction de cette matrice, la requête ayant l'affinité réflexive maximale est choisie pour être la première au Planning d'Exécution. Une fois cette requête élue, l'ordonnanceur choisit les requêtes restantes en fonction de l'affinité et du contenu du buffer, géré par la politique de gestion décrite en Section 4.5.

La résolution du problème joint en utilisant l'algorithme d'affinité, est effectuée en deux étapes : dans la première étape, la matrice d'affinité est construite pour fournir une solution initiale pour l'ordonnanceur en triant les requêtes par affinité réflexive décroissante. La requête avec l'affinité réflexive maximale est élue pour être la première à s'exécuter. La deuxième étape consiste à améliorer l'ordre des requêtes restantes en utilisant la politique de gestion du buffer. Cette étape comporte quatre opérations qui s'effectuent itérativement : la requête élue est exécutée et ses nœuds candidats sont mis dans le buffer, dans la limite de l'espace disponible. La politique de gestion du buffer adoptée s'occupe de l'allocation et du remplacement des données. L'ordonnanceur réajuste l'ordre des requêtes restantes en fonction du contenu du buffer. Il élit la requête ayant le nombre maximal de nœuds dans le buffer. Si deux requêtes ont le même nombre, il choisira celle qui a l'affinité réflexive la plus élevée. Avant de passer à l'exécution de la prochaine requête élue, la matrice BM est mise à jour par les nœuds présents dans le buffer. Le processus d'ordonnancement et de gestion du buffer réitère jusqu'au traitement de la dernière requête. La figure 3.6 montre la méthodologie de résolution dirigée par l'affinité. Dans les sections suivantes, nous décrivons le détail de chaque phase.

Pour illustrer ce principe, nous décrivons l'application de chaque étape sur le MVPP de notre exemple initial (cf. page 23).

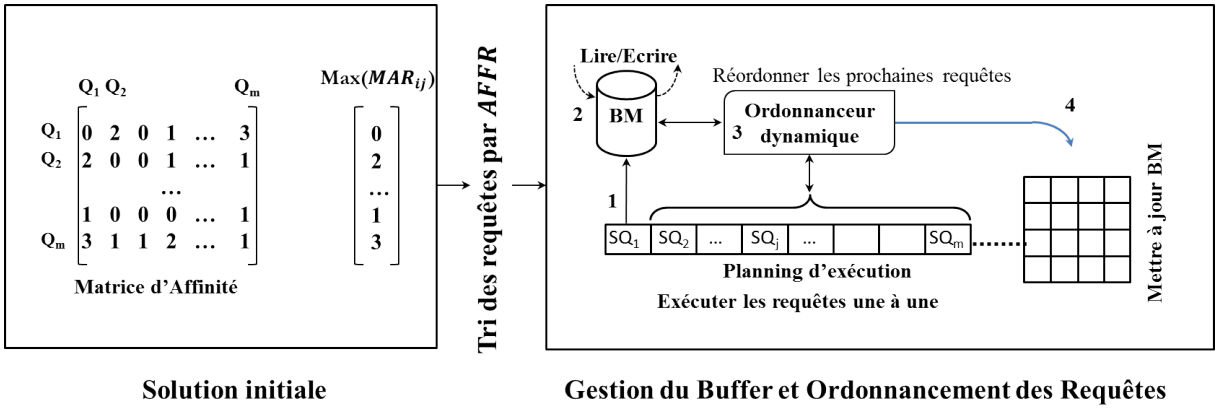

FIGURE 3.6 – Principe de l'algorithme dirigé par l'affinité

Algorithme 1: Algorithme d'Affinité

```
1: Algorithme Affinite():
2: init_buffer(); {initialisation du buffer}
3: MAR(); {construction de la matrice MAR}
4: for all q_i ∈ Q do
5:    AFFR(q_i); {calcul de l'affinité réflexive}
6: end for
7: Q_rest := Q
8: q^elue := max_AFFR(Q_rest)
9: while Q_rest ≠ ∅ do
10:   for all no_j ∈ plan(q^elue) do
11:      Gestion_Buffer(no_j);
12:   end for
13:   Q_rest := Q_rest − {q^elue};
14:   max := utilisation_buffer(Q_rest);
15:   if ( max ≠ 0) then
16:      q^elue := requete(max);{élire l'une des requêtes utilisant le maximum de nœuds en cache.}
17:      maj_P(q^elue); {réajuster le planning d'exécution.}
18:   else
19:      {si aucun nœud en cache n'est requis par les requêtes restantes.}
20:      q^elue := max_AFFR(Q_rest);{choisir la requête ayant l'affinité réflexive maximale.}
21:   end if
22: end while
```

5.1.2 Construction de la Matrice d'Affinité des Requêtes

A partir du MVPP en entrée, l'algorithme commence par construire la Matrice d'Affinité des Requêtes, nommée MAR : les lignes et les colonnes représentent les requêtes de la charge. Chaque cellule $MAR[i, j]$ de la matrice contient la valeur d'affinité $\mathcal{AFF}(Q_i, Q_j)$ correspondante. La matrice obtenue dans notre exemple sera celle représentée dans la figure 3.7, ainsi que les valeurs d'affinité reflexive, et le tri initial des requêtes.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Q1	0	0	0	1	0	1	0	0	0	0
Q2	0	0	0	0	1	0	0	1	0	0
Q3	0	0	0	0	0	0	0	0	1	1
Q4	1	0	0	0	0	2	2	0	0	0
Q5	0	1	0	0	0	0	0	2	0	0
Q6	1	0	0	2	0	0	3	0	0	0
Q7	1	0	0	2	0	3	0	0	0	0
Q8	0	1	0	0	2	0	0	0	0	0
Q9	0	0	1	0	0	0	0	0	0	3
Q10	0	0	1	0	0	0	0	0	3	0

	AFFR
Q1	1
Q2	1
Q3	1
Q4	2
Q5	2
Q6	3
Q7	3
Q8	3
Q9	3
Q10	3

Ordre
Q6
Q7
Q8
Q9
Q10
Q4
Q5
Q1
Q2
Q3

FIGURE 3.7 – La matrice d'affinité des requêtes obtenue du MVPP

5.1.3 Choix de la première requête

Après la construction de la matrice d'affinité, la valeur d'affinité réflexive $\mathcal{AFFR}(Q_i)$ est calculée pour chaque ligne (requête) de la matrice. La requête ayant la valeur maximale de $\mathcal{AFFR}$ est choisie pour être la première dans le planning d'exécution, car elle permet de fournir des nœuds en cache pour un plus grand nombre de requêtes. Le reste du planning contient les requêtes restantes dans un ordre aléatoire.

5.1.4 Ordonnancement et gestion du buffer

Après l'identification de la première requête du planning d'exécution, deux phases de traitement sont requises :

Phase 1 : Le gestionnaire du buffer associé à l'ordonnanceur met les nœuds candidats de cette requête dans le buffer tant que la taille disponible le permet. Par exemple, la requête Q_6 met dans le buffer un ensemble de nœuds. Cet ensemble servira l'une des requêtes en interaction avec Q_6 comme le montre la matrice d'affinité. *Phase 2 :* Après avoir mis en cache les objets de la requête en cours, l'ordonnanceur choisit la prochaine requête dans le planning comme celle ayant le nombre maximal de nœuds pertinents dans le cache.

Quand la deuxième requête est choisie, les deux phases précédentes sont appliquées de la même manière jusqu'à la dernière requête dans le planning. Si aucune requête ne contient de nœud pertinent dans le cache, l'algorithme élit celle ayant le maximum d'affinité réflexive $\mathcal{AFFR}$ parmi les requêtes restantes.

5.1.5 Bilan et discussion

Cet algorithme exploite principalement l'affinité entre les requêtes. Il permet de maximiser la réactivité du moteur de BD, puisqu'il maximise la proximité temporelle des requêtes ayant des affinités importantes, et ainsi des nœuds candidats qui peuvent satisfaire plusieurs requêtes. Commencer par ces requêtes permet de mettre leurs nœuds en cache avant qu'il ne soit saturé. Le choix de la prochaine requête utilisant le plus de nœuds cachés permet de remplacer ces nœuds régulièrement, car à chaque utilisation l'*Utilité* ($U(no_i)$) est décrémentée.

L'affinité n'est pas un critère suffisant pour décider de l'ordre des requêtes. Il peut y avoir deux requêtes ayant la même valeur d'affinité réflexive, mais donnant chacune une performance différente. Donc l'algorithme d'affinité, bien que simple et rapide, ne permet pas d'atteindre une efficacité optimale. Cette constatation nous amène à étudier la contribution de méta-heuristiques en terme de performance.

5.2 Hill Climbing

L'algorithme de Hill Climbing (HC) est une méthode de voisinage utilisée pour trouver une solution proche de l'optimum [166]. Dans la GBOR, le Hill Climbing permet d'explorer l'espace de recherche en donnant la chance à plusieurs solutions possibles pour être évalués, et ainsi à d'autres nœuds à être mis en cache.

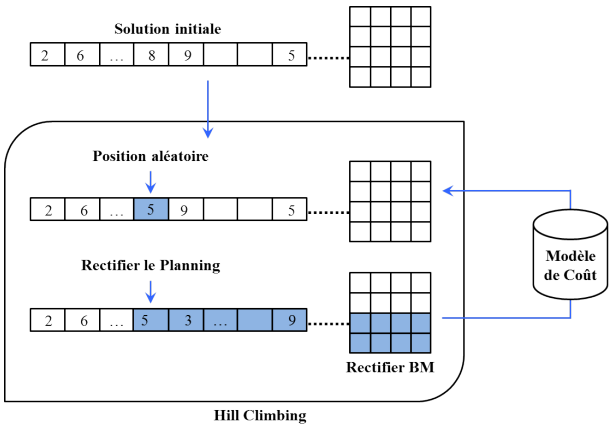

FIGURE 3.8 – Principe du Hill Climbing

5.2.1 Principe

Le principe du Hill Climbing proposé est de choisir une solution initiale et d'effectuer itérativement des changements locaux. Ces changements permettent de chercher dans son voisinage une autre solution réduisant le coût d'exécution de la charge. L'algorithme s'arrête lorsqu'il atteint la condition d'arrêt (le nombre d'itérations maximal). L'évaluation de la qualité des solutions est effectuée à l'aide du modèle de coût proposé en Section 4.4. La figure 3.8 résume le principe de l'algorithme.

5.2.2 Solution initiale

La solution initiale constitue le point de départ du processus d'exploration de l'espace de recherche. Le choix de cette solution influe le déroulement de l'algorithme, et par conséquent, la qualité de la solution finale. La solution initiale peut être choisie deux manières différentes : aléatoirement, ou issue d'un autre algorithme. La première alternative est plus rapide mais ne garantit pas une meilleure performance de l'algorithme. Pour cela, nous allons exploiter le résultat de l'algorithme d'Affinité pour être un état initial dans le Hill Climbing. Ce qui permet de démarrer d'une solution plus efficace exploitant l'affinité entre les requêtes [52].

5.2.3 Mouvements effectués

Le passage de la solution courante vers un voisinage qui présente de meilleures performances, est effectué par une fonction de transformation appelée : *Transform*. Cette fonction effectue une transformation de la solution courante dans une position aléatoire i dans le vecteur $\mathcal{P}$.

$$Transform(P, BM, i) = (\mathcal{P}', BM')$$

Comme le vecteur $\mathcal{P}$ représente l'ensemble des requêtes ordonnées, le changement d'une valeur de ce vecteur entraînera une multi-occurrence d'une requête dans la solution. La figure 3.8 montre un exemple de cette transformation.

En plus de la multi-occurrence, le changement de la position d'une requête entraînera le changement des nœuds cachés. Par conséquent, toute la séquence des requêtes qui suit cette position doit être adaptée au contenu du buffer issu de cette transformation.

Pour éviter cette incohérence, nous proposons une fonction *Rectify* qui rectifie le contenu du buffer correspondant à partir de la position i du vecteur $\mathcal{P}$, ainsi que l'ordre des requêtes suivantes. Pour effectuer ces modifications, la fonction Rectify utilise le même principe d'ordonnancement et de gestion du buffer employé dans l'algorithme d'affinité sur le sous-ensemble de requêtes restantes, à partir de la position i.

$$Rectify(\mathcal{P}_k, BM_k, i) = (\mathcal{P}_k^r, BM_k^r)$$

Où $\mathcal{P}_k^r$ et BM_k^r représentent respectivement le nouveau Planning d'Exécution et son scénario du buffer rectifiés après l'application de transformation.

Algorithme 2: Fonction de transformation

```
1: Fonction Transform():
2: INPUT: (P, BM, i)
3: q := requete_aleatoire(P, BM, i);
4: P[i] := q;
5: retourner (P, BM);
```

La transformation est guidée par le modèle de coût qui évalue la performance des solutions. Une nouvelle solution est retenue si elle améliore la performance (réduit le coût) par rapport à la solution

Algorithme 3: Fonction de Rectification

```
1: Fonction Rectify():
2: INPUT: (𝒫, BM, pos)
3: for all i = pos jusqu'à l do
4:    𝒫[i] := elire_max_usage();
5:    for all noeud ∈ 𝒫 do
6:       verifier_buffer(noeud) ;
7:       maj_BM();
8:    end for
9: end for
10: retourner (𝒫, BM);
```

courante. L'algorithme la transforme en une solution voisine de meilleure qualité. Le processus réitère jusqu'à atteindre le nombre total d'itérations.

Algorithme 4: Algorithme de Hill Climbing

```
1: Algorithme HC():
2: while (nombre d'itération maximale non atteint) do
3:    (𝒫_sol, BM_sol) := Affinity();
4:    i := position_aleatoire();
5:    (𝒫_new, BM_new) := Transform(𝒫_sol, BM_sol, i);
6:    (𝒫_new, BM_new) := Rectify(𝒫_new, BM_new, i);
7:    if (Cout(𝒫_new, BM_new < Cout(𝒫_sol, BM_sol)) then
8:       (𝒫_sol, BM_sol) := (𝒫_new, BM_new);
9:    end if
10: end while
11: retourner (𝒫_sol, BM_sol);
```

5.3 Algorithme génétique

Les algorithmes génétiques sont largement utilisés dans les problèmes d'optimisation combinatoire en bases de données. Contrairement au Hill Climbing, et grâce à leur opérateur de *mutation*, les algorithmes génétiques peuvent sortir des optima locaux. Dans cette section nous allons étudier en détail cet algorithme appliqué à notre problème de GBOR.

5.3.1 Principe

Les algorithmes génétiques sont des méthodes évolutionnaires inspirées des phénomènes biologiques et basées sur le concept de survie des meilleurs individus aux cours des générations [115]. L'algorithme démarre d'une population initiale, et effectue des opérations génétiques sur les individus. Les meilleurs individus sont conservés et croisés pour obtenir une nouvelle génération. Ainsi, à partir d'un nombre

fini de générations, les individus acquièrent de meilleures performances. Un algorithme génétique est principalement défini par cinq éléments :

1. Le codage des chromosomes (individus);
2. La génération de population initiale;
3. La fonction objectif : fitness;
4. Les opérateurs génétiques : sélections, croisement, mutation;
5. Paramétrage de l'algorithme : taille d'une population, taux de croisement, taux de mutation, nombre de générations maximal et condition d'arrêt.

5.3.2 Codage et fonction objectif

Afin de pouvoir appliquer les opérateurs génétiques, les chromosomes sont représentés par notre codage proposé en Section 4.3. Un chromosome est constitué de deux structures : le vecteur des requêtes $\mathcal{P}$, et la matrice du scénario de buffer BM. La fonction objectif permet d'évaluer la qualité des individus. Dans la GBOR, le meilleur individu est celui qui a un coût d'exécution réduit de la charge globale. Ainsi, nous proposons la fonction objectif F à partir du modèle de coût comme suit :

$$F(\mathcal{P}, BM) = -Cout(\mathcal{P}, BM)$$

5.3.3 Génération de population initiale

Les individus de la première génération sont engendrés aléatoirement pour avoir une population homogène avec des performances variables. La diversité de la première génération permet d'élargir le champ de recherche de l'algorithme.

Algorithme 5: Algorithme Génétique

```
1: Algorithme Génétique:
2: génération de population initiale Pop_0;
3: evaluer(Pop_0);
4: while condition d'arrêt non atteinte do
5:     Pop_{i+1} := selection(Pop_i);
6:     Pop_{i+1} := croisement(Pop_i);
7:     Pop_{i+1} := mutation(Pop_i);
8:     evaluer(Pop_{i+1});
9: end while
```

5.3.4 Opérateurs génétiques

Afin d'effectuer des changements sur les individus d'une population, les opérateurs génétiques sont utilisés avec des probabilités prédéfinies (paramètres de l'AG). Ces opérateurs sont : la sélection, le croisement et la mutation.

Algorithme 6: Fonction de Croisement

```
1: Fonction croisement:
2: i := position_aleatoire();
3: b := debut_class(i, fils1);
4: e := fin_class(i, fils2);
5: for all j = b to e do
6:     temp := fils1[j];
7:     fils1[j] := fils2[j];
8:     fils2[j] := temp;
9: end for
10: Rectify(BM, b);
11: retourner (P, BM);
```

La sélection : Consiste à élire des parents pour être croisés, en choisissant les meilleurs individus de la population en termes de performance. Le taux de sélection est un paramètre de l'algorithme et il permet de fixer le nombre d'individu à élire pour obtenir la prochaine génération.

Définition 11

Nous appelons Classe de requêtes tout sous-ensemble de requêtes de la même charge ayant au moins un nœud candidat en commun.

Exemple. Dans la figure 3.9, les cellules du planning $\mathcal{P}_i$ sont regroupées en trois classes. Par exemple, les requêtes Q_1 (notée '1') et Q_7 (notée '7') sont de la même classe car elles partagent une jointure en commun. Quant à la requête Q_5 (notée '5'), elle appartient à une autre classe car elle ne partage aucune jointure avec Q_1 et Q_7. Le MVPP (cf. page 23) permet de ressortir les jointures partagées entre les plans de requêtes, sur lesquelles nous nous somme basés pour répartir les classes.

Le croisement : Les parents s'échangent leur patrimoine génétique afin de produire de nouveaux individus. Il existe plusieurs types de croisement : *bit, boundary,* etc. Le croisement consiste à choisir une position ou deux, selon le type, pour faire une coupure dans les gènes et effectuer l'échange des séquences génétiques. Or, ces types ne sont pas adaptés au problème de la GBOR, car ils provoquent la multi-occurrence des requêtes. Pour cela, nous avons défini un croisement adapté à notre problème comme le montre la figure 3.9 :

Pour favoriser l'interaction entre les requêtes, le croisement est effectué en échangeant les séquences génétiques par classes de requêtes, au lieu de les croiser aléatoirement. Ceci permet de conserver le regroupement obtenu par l'algorithme d'Affinité (solution initiale).

La mutation : Permet de diversifier la population en effectuant un changement aléatoire dans un gène d'un individu. La probabilité de mutation doit être très faible (inférieure à 5% par exemple). Cet opérateur permet de sortir des optima locaux inhérents aux autres heuristiques comme dans le cas du Hill Climbing. Ainsi, une solution peut contenir des séquences de requêtes qui ne sont pas forcément de la même classe que les requêtes voisines dans le planning. Pour imposer qu'une requête soit toujours dans sa classe, nous proposons d'effectuer la mutation de la façon suivante :

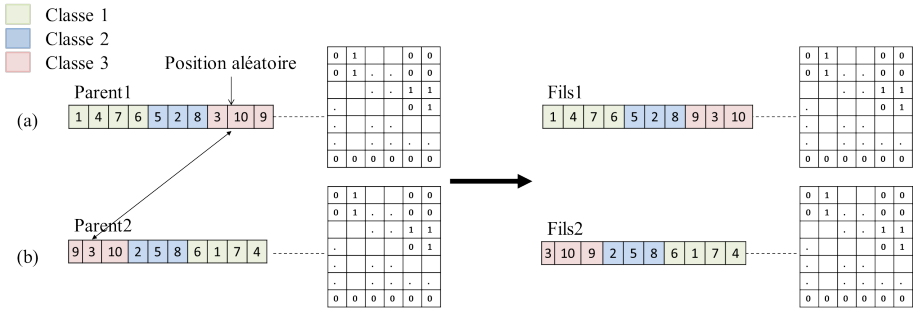

FIGURE 3.9 – Principe de croisement des parents

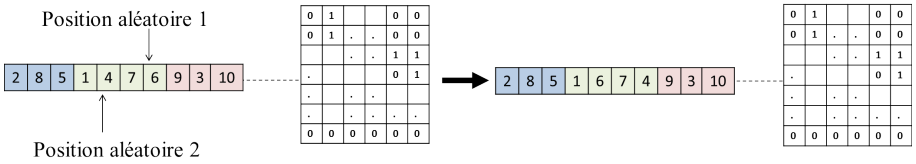

FIGURE 3.10 – Principe de mutation d'un individu

Chaque classe est délimitée par une position de début d et de fin f. Deux positions aléatoires sont choisies dans l'intervalle $[d, f]$, et leurs requêtes sont échangées. La figure 3.10 illustre cette opération.

5.3.5 Bilan et discussion

Les méta-heuristiques permettent d'explorer de grands espaces de recherche et de trouver des solutions optimales à certains problèmes $\mathcal{NP}$-complets. Dans la GBOR, ces méta-heuristiques permettent de produire des solutions ayant une efficacité quasi-optimale contrairement à l'algorithme d'Affinité. Or, ces algorithmes sont très gourmands en temps de calcul et nécessitent une durée de traitement beaucoup plus élevée.

L'algorithme génétique utilise les opérateurs en respectant l'Affinité. Cependant, le hill climbing effectue des mouvements aléatoires. L'utilisation de l'affinité entre les requêtes rend notablement plus efficace le processus d'ordonnancement et de gestion de buffer. Théoriquement, chaque fois qu'une requête est élue, toutes les requêtes qui partagent un nœud avec elle sont exécutées juste après, avant que le buffer soit entièrement vidé. L'arrivée d'une nouvelle requête après ce groupe provoquera le même comportement. La GBOR nécessite toutefois une métrique plus pointue pour décider de l'ordre comme le modèle de coût utilisé dans les méta-heuristiques. Les techniques d'optimisation dirigées par modèle de coût (Cost Based Approaches) ont montré leur efficacité dans les problèmes d'optimisation combinatoire [52], comparées aux approches dirigées par affinité [60] ou les techniques de data mining [145].

Ces faits aussi bien que la recherche d'un compromis entre la complexité et l'efficacité de l'algorithme, nous ont poussé à proposer un nouvel algorithme basé sur le concept : "*diviser pour régner*" qui exploite à la fois l'affinité entre les requêtes et le modèle de coût. Cet algorithme, que nous appelons : "*Queen-Bee*", est détaillé dans la section suivante.

5.4 Queen-Bee

Dans le but d'avoir une approche intermédiaire proposant un compromis entre la complexité de l'algorithme et son efficacité (qualité de la solution), nous proposons un nouvel algorithme : "*Queen-Bee*". Cet algorithme exploite l'affinité entre les requêtes, ainsi que le coût d'exécution, pour chercher localement les meilleurs ordres des sous-ensembles de requêtes.

Avant de décrire l'algorithme, considérons l'exemple suivant :

Exemple. Considérons la charge de 10 de notre MVPP. En réorganisant les plans de façon à regrouper les requêtes partageant au moins une jointure, nous obtenons le MVPP de la figure 3.11-b. Les groupes r_1, r_2 et r_3 sont appelées des *"ruches"*. Quand la requête Q_1 est exécutée, la jointure j_1 est mise en buffer (la taille du résultat de J_1 est supposée inférieure à la taille du buffer disponible). La mise en buffer de J_1 servira à la fois à Q_4, Q_6 et Q_7. Cependant, elle n'est pas bénéfique pour les requêtes de r_2 ni de r_3. L'exécution de Q_4, Q_6 ou Q_7 après Q_1 fournit une meilleure performance car elle permet de lancer les requêtes utilisant le contenu courant du buffer (au moment de l'exécution). Une fois toutes les requêtes de r_1 sont lancées, l'*Utilité* des nœuds du buffer s'annule, et ces nœuds sont ôtés pour laisser la place à d'autres. De cette manière, les ruches sont exécutées l'une après l'autre, et le buffer n'est occupé que par les nœuds pertinents pour la ruche en cours.

5.4.1 Principe

Pour réduire la complexité des premiers algorithmes de résolution, nous avons proposé une approche inspirée de la vie des abeilles. Le principe est de raisonner par affinité entre les requêtes, et de décomposer la charge globale en sous-ensembles de requêtes ayant une interaction entre-elles (un nœud candidat en commun). Un exemple de décomposition est illustré dans la figure 3.11.

Cette décomposition permet de traiter le problème d'ordonnancement de façon locale. Suivant notre politique de gestion de buffer, aucun nœud ne restera en cache après l'exécution des requêtes contenues dans l'un des sous-ensembles obtenus. Ceci est dû au fait qu'aucun de ces nœuds ne sera utilisé par les prochaines requêtes. Son *Utilité* sera nulle, i.e. $U(no_i) = 0$, ce qui implique son retrait du buffer.

Nous avons donc fait l'hypothèse qu'utiliser au maximum les nœuds présents dans le cache, permettrait de maximiser le rendement du processus d'optimisation des requêtes. L'approche Queen-Bee consiste à ordonnancer les requêtes en une succession de *ruches*. De chaque ruche, une requête est élue pour être la première à s'exécuter parmi ses homologues (requêtes de la même ruche). Cette requête est dite *la queen-bee*, et son exécution impose le chargement d'un nouveau contexte dans la mémoire cache. Les requêtes suivantes dans la ruche, exploitent les nœuds déjà chargés par la requête queen-bee.

L'ordonnancement et la gestion du buffer sont effectués de la façon suivante dans l'algorithme *Queen-Bee* :

1. Construction du Graphe de Requêtes en Composantes Connexes (GRCC);
2. Tri des composantes connexes (optionnel);
3. Ordonnancement local des composantes connexes.

Dans ce qui suit, nous allons détailler chaque phase de l'algorithme.

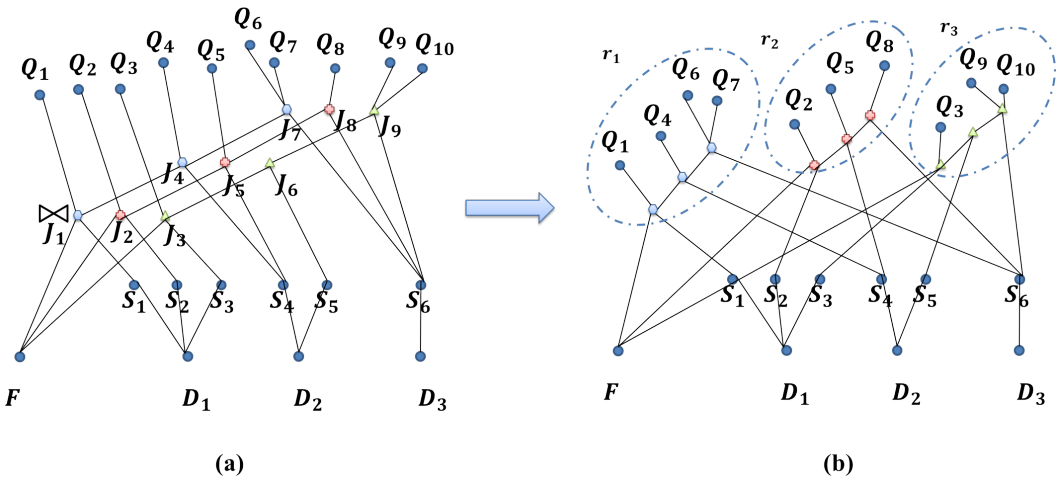

FIGURE 3.11 – Regroupement de requêtes par interaction

5.4.2 Graphe de Requêtes en Composantes Connexes

Pour la construction des ruches à partir du MVPP, nous utilisons des techniques de la théorie des graphes. Le graphe représentant les groupes de requêtes en interaction est appelé : *Graphe de Requêtes en Composantes Connexes* (GRCC). Ce graphe est construit de la façon suivante :

- Les sommets représentent les requêtes de la charge globale.
- Une arête relie deux sommets si les deux requêtes correspondantes ont au moins un nœud candidat en commun.
- Les sommets sont étiquetés par le coût d'exécution initial de la requête correspondante. Les arêtes sont étiquetés par le nombre de nœuds partagés par les requêtes reliées.

Un exemple de GRCC est présenté sur la figure 3.12. Ce graphe est construit à partir de notre MVPP.

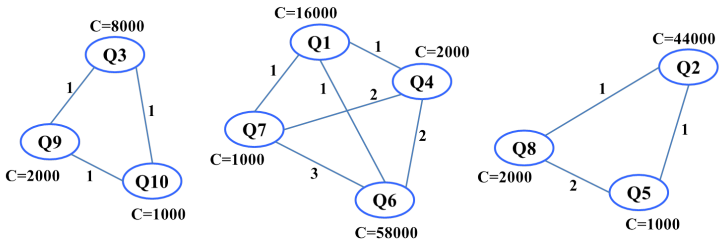

FIGURE 3.12 – Construction du graphe de requêtes à composantes connexes

5.4.3 Tri des composantes connexes

Cette phase permet de trier les ruches selon les recommandations de l'administrateur, car ce dernier peut proposer des priorités d'exécution des requêtes. Par conséquent, les ruches prioritaires sont traitées en premiers afin de réduire le temps d'attente des requêtes prioritaires. Si l'administrateur n'impose pas de contraintes pour le temps d'attente ou de priorité de certaines requêtes, l'ordre dans lequel les ruches sont traitées (les composantes connexes) n'a aucune importance.

Par exemple, les deux ordres différents des ruches suivants :

$$\{(Q1, Q7, Q4, Q6); (Q3, Q9, Q10); (Q2, Q8, Q5)\} \text{ et } \{(Q3, Q9, Q10); (Q1, Q7, Q4, Q6); (Q2, Q8, Q5)\}$$

ont le même coût d'exécution. Ceci revient au fait que les nœuds requis par les requêtes de deux composantes sont disjoints. Pour cela, nous ne considérons pas cette phase dans notre étude. Cependant, dans des travaux futurs, elle sera exploitée pour écourter le temps d'attente des requêtes prioritaires.

5.4.4 Ordonnancement local des requêtes

Afin d'ordonner les requêtes à l'intérieur des ruches, l'ordonnanceur local prend chaque composante connexe dans le GRCC généré et ordonne les requêtes en effectuant trois étapes :

1. Identifier la Queen-Bee selon un critère donné;
2. Explorer les sommets restant dans la composante en utilisant le même critère pour le choix des prochaines requêtes, en mettant à jour le coût de chaque sommet suivant le contenu du buffer;
3. Fournir le sous-ensemble de requêtes ordonné obtenu, une fois la composante connexe est entièrement parcourue.

Les requêtes de la même ruche, choisies après la requête queen-bee, exploitent le contenu du buffer car elles ont au moins un nœud candidat en commun. Par contre, la queen-bee n'a aucune réduction du coût car elle est la première de sa composante à s'exécuter et au moment de son exécution le buffer ne contient aucun nœud pouvant lui être utile (buffer vide).

Définition 12

Le fan-out d'une requête est la somme de ces participations en termes de nœuds partagés dans la charge. Plus formellement :

$$fanout(Q_i) = \sum_{i \neq j} \mathcal{AFFR}(Q_i, Q_j)$$

Les critères d'élection de la requête queen-bee pouvant avoir un impact le processus d'ordonnancement sont : le coût d'exécution des nœuds, ou le *fan-out* des requêtes.

La figure 3.13 donne un exemple de tri basé sur le coût comme critère de choix. Quand nous traversons la composante, en commençant par la requête de coût minimal, le coût des sommets non encore visités est mis à jour selon le contenu du cache. A chaque étape, le sommet de coût minimal est élu. L'algorithme de tri réitère l'opération tant que la composante n'est pas entièrement parcourue.

La requête de moindre coût est exécutée en premier : comme elle a la charge de peupler la mémoire cache, la choisir minimise le coût global de la séquence d'ordonnancement. Cette première requête n'a aucun gain du contenu du buffer, car les requêtes ayant des nœuds communs avec elle n'ont pas encore été exécutées. Mais au même temps, elle apportera un gain à toutes les requêtes de sa composante en cachant ces nœuds partagés, avant que le buffer ne soit saturé.

Le fan-out peut aussi être un critère intéressant pour le choix de la queen-bee. L'idée consiste à commencer par la requête apportant le maximum de nœuds communs aux autres requêtes. La figure 3.14 donne un exemple de tri par fan-out. On peut observer que théoriquement, le tri par coût minimal donne des performances différentes que celles obtenues par fan-out maximal. Les résultats de la pratique sont présentés et discutés dans la partie expérimentale.

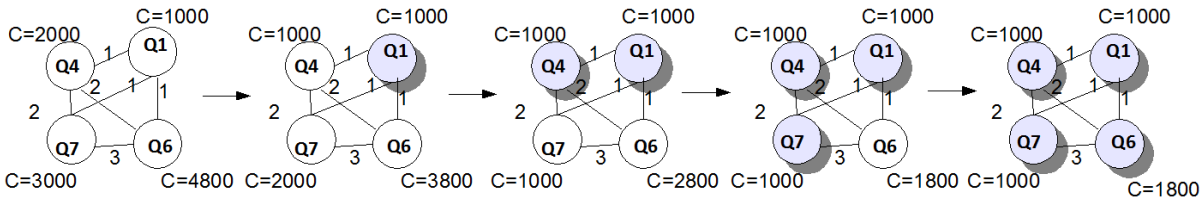

FIGURE 3.13 – Ordonnancement local par coût minimal

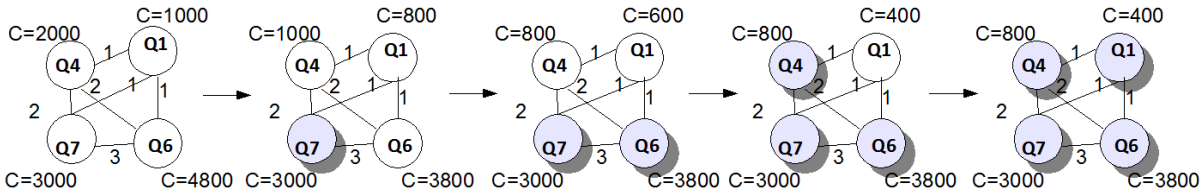

FIGURE 3.14 – Exemple d'ordonnancement local par fan-out maximal

Exemple. Le coût total [12] obtenu dans la figure 3.13 du tri par coût minimal est égal à 1000 + 600 + 600 + 600 + 4000 = 6200 I/O, et celui obtenu dans la figure 3.14 par fan-out maximal est égal à 3000 + 3800 + 800 + 400 = 8000I/O. Les deux critères ne donnent pas le même ordre. Pour étudier l'impact de ces critères dans la pratique, nous avons établi une étude expérimentale sur chacun (Section 6).

5.5 Bilan et discussion

Dans cette partie nous avons proposé un algorithme appelé Queen-Bee. Cet algorithme est inspiré de la vie des abeilles, et permet de diviser l'espace de recherche afin de réduire la complexité algorithmique du traitement de la GBOR. La complexité algorithmique pour la Queen-Bee est déterminée de la façon suivante :

12. Les coûts fournis dans l'exemple sont donnés à titre indicatif, ils ne correspondent pas aux coûts réels.

L'algorithme Queen-Bee s'organise en trois principales étapes. Dans la première, le GRCC est construit pour les n requêtes. Soit l le nombre de nœuds dans le MVPP pour tous les plans de requêtes. Pour chaque requête, les nœuds existants dans le plan sont explorés et comparés avec les autres $n-1$ plans de requêtes. La génération du GRCC est effectuée en $O(n^2 \times l)$. La phase d'ordonnancement des composantes connexes est optionnelle. La dernière phase est l'optimisation locale. Elle se produit au niveau de chaque composante connexe en effectuant les opérations suivantes pour les n_i requêtes internes :

Explorer la composante en cours, et pour chaque requête :
a. Vérifier le contenu du buffer;
b. Mettre à jour le contenu du buffer;
c. Évaluer la requête;
d. Trier des requêtes restantes dans la composante;
e. Choisir la prochaine requête;

L'estimation du coût d'exécution nécessite l'exploration du plan d'exécution de la requête en cours, et pour chaque nœud, le contenu du buffer est vérifié. L'estimation du coût est effectuée en $O(l)$. Le tri des requêtes restantes est effectué en $O(log(n_i))$. Le coût des autres opérations (a,b,e) est constant, ainsi l'ordonnancement de toutes les composantes est fait en $O(n \times l + n \times log(n))$.

La complexité globale peut être simplifiée en ne conservant que l'élément le plus important : $O(n^2 l)$, car on ne conserve que la complexité maximum des parties d'algorithme exécutées en séquence. En effet lorsque n et l sont grands, alors $n^2 \times l$ est plus importante que $n \times l$ et $n \times log(n)$. Autrement dit : $(n^2 \times l >> n \times l) \wedge (n^2 \times l >> n \times log(n))$. Donc en pratique, le terme : $O(n^2 \times l)$, est le seul à conserver pour définir la complexité de l'algorithme. L'algorithme est donc effectué en $O(n^2 \times l)$.

La complexité est donc polynomiale, contrairement aux méta-heuristiques connues par leur complexité exponentielle. L'algorithme Queen-Bee partitionne l'ensemble des requêtes selon leur affinité en sous-ensembles que nous appelons : *ruches*. Puis, pour chaque ruche, il procède par élire une requête d'initialisation selon un critère donné. Ce critère et l'évolution du contenu du buffer, sont utilisés pour ordonner localement les autres requêtes de cette ruche. Les critères de choix que nous avons étudiés sont : le coût et le fan-out. Leur efficacité est comparée par expérimentation dans la Section 6.

6 Évaluation des performances des algorithmes

Dans cette partie, nous confrontons à l'expérimentation les algorithmes que nous proposons. Pour pouvoir réaliser une grande série de tests sans être contraints par le SGBD et la volumétrie des tables (les temps de traitement sont potentiellement élevés), nous avons développé un simulateur. Ce simulateur, détaillé dans le Chapite 6 se base sur un modèle de coût et fournit des résultats théoriques pour les différentes configurations de test. Ces résultats sont par la suite déployés sur un jeu de données réel sous Oracle11g pour être validés.

Nous présentons d'abord notre jeu de données, puis les résultats de notre étude organisés en deux parties : les expérimentations théoriques basées sur un modèle de coût, puis la validation sous Oracle11g.

Table	Type	Cardinalité
Lineorder	Faits	$6.000.000 \times SF$
Supplier	Dimension	$2.000 \times SF$
Customer	Dimension	$30.000 \times SF$
Part	Dimension	$200.000 \times (1 + log_2 SF)$
Dates	Dimension	2556

TABLE 3.2 – Cardinalités des tables de l'entrepôt SSB

6.1 Jeu de données

Nos expérimentations ont été réalisées sur le benchmark SSB (*Star Schema Benchmark* [157]). Ce benchmark contient un entrepôt de données dont le schéma en étoile comporte quatre tables de dimensions autour d'une table des faits.

La Table 3.2 présente les différentes tables du schéma en étoile du SSB. Les cardinalités sont exprimées en fonction d'un facteur d'échelle SF (*Scale Factor* en anglais). Pour notre étude, nous avons utilisé un facteur SF=100 afin de générer un entrepôt de taille de 100*Go* de données. La charge utilisée contient 30 requêtes SSB de jointure en étoile.

6.2 Simulations

Nous commençons par étudier la performance des algorithmes en comparant l'algorithme d'affinité avec les heuristiques : AG et HC. Pour l'AG, nous fixons le nombre de populations à 400, le taux de mutation à 0.05 et le nombre d'individus par population à 20. A chaque génération, les meilleurs 10 individus (parents) de la population en cours, sont sélectionnés pour être croisés. Pour l'algorithme de Hill Climbing, le nombre d'itérations est fixé à 400, et la solution initiale est obtenue par l'algorithme d'affinité.

Les résultats d'expérimentations menées avec ces trois algorithmes, sont présentés dans la figure 3.15. On constate que toutes les heuristiques sont plus efficaces que l'algorithme d'affinité. Ils montrent aussi que l'algorithme génétique donne la meilleure optimisation de la charge. Les heuristiques ont la capacité de parcourir un espace de recherche beaucoup plus large que les algorithmes simples (Affinité) ce qui explique le gain obtenu en optimisant la charge par le HC et l'AG. L'algorithme d'Affinité permet de commencer le planning par la requête ayant le plus d'affinité avec les autres. Le tri des requêtes est guidé par le contenu du buffer pour adapter le planning. Cependant, ce tri par affinité n'est pas suffisant pour garder les données pertinentes en cache et satisfaire la totalité de la charge.

Pour réduire la complexité algorithmique tout en augmentant l'efficacité, nous avons proposé l'algorithme Queen-Bee basé sur l'affinité et l'optimisation locale.

Afin de tester l'impact des différents facteurs pouvant influencer le choix des nœuds à mettre en cache, nous avons lancé l'algorithme Queen-Bee en variant le critère de choix de la requête queen-bee. Les résultats dans la figure 3.17 montrent que l'élection de la Queen-Bee par coût minimal est meilleure que le choix par coût maximal. De même, le résultat d'ordonnancement par fan-out minimal, est meilleur que le choix par fan-out maximal. Ce résultat s'explique par le fait qu'une requête ayant un

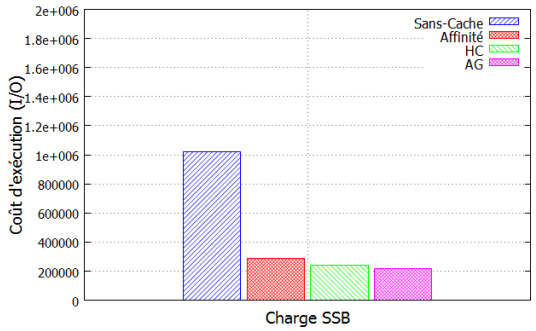

FIGURE 3.15 – Performances des algorithmes : Affinité, AG et HC

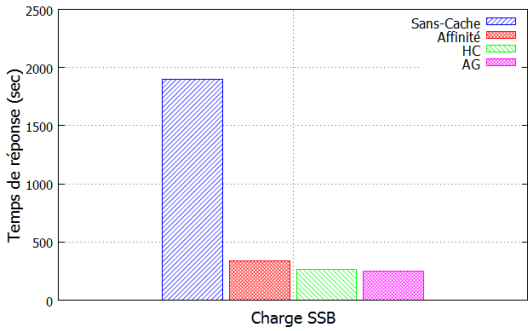

FIGURE 3.16 – Performance effective des algorithmes : Affinité, AG et HC

fan-out minimal est une requête ayant un nombre réduit de nœuds, qui est souvent une requête avec un coût réduit. Le coût minimal donne une meilleure performance à l'algorithme Queen-Bee car il permet de sacrifier la requête la moins coûteuse, et de fournir des nœuds du bas du plan de requêtes. Ce type de nœuds correspond aux premières jointures. La taille du buffer étant limitée, les nœuds les plus bas occuperont les premiers le buffer avant qu'il ne soit saturé. Dans le cas contraire, où une requête avec un coût maximal est exécutée, le buffer sera occupé par des nœuds d'un niveau plus haut dans le plan d'exécution. Ces nœuds là, ne servirons pas toutes les requêtes de la ruche contrairement aux nœuds de la première jointure. Bien que la différence en performance est étroite dans notre étude, elle peut être plus significative lors du passage à l'échelle. Pour le reste de nos expérimentations, nous avons retenu le critère d'élection par coût minimal.

Afin de montrer la performance de l'algorithme Queen-Bee, nous l'avons comparé aux cas suivants :

- Sans optimisation;
- Avec une gestion du buffer par LRU, sans ordonnancement;
- Avec une gestion du buffer par LRU et ordonnancement par l'ordonnanceur dynamique;
- Avec une gestion du buffer par le gestionnaire dynamique (DBM), sans ordonnancement;
- Avec une gestion du buffer par le gestionnaire dynamique et ordonnancement par l'ordonnanceur dynamique (intégrés aux méta-heuristiques).

La taille du buffer est variée (entre 0 Giga-octets et 32 Giga-octets) pour montrer son impact sur le rendement de chaque stratégie d'optimisation. Les résultats dans la figure 3.18 montrent que l'ordonnancement est un complément de performance aux politiques de gestion du buffer adoptées. Autrement dit, l'ordonnancement des requêtes permet d'améliorer la performance obtenue par la gestion du buffer. De plus, la politique de gestion de buffer dynamique est plus appropriée que LRU dans le contexte de base de données [101], car elle prend en considération l'interaction entre les requêtes. L'expérience montre que l'algorithme Queen-Bee donne une performance comparable à celle obtenue par heuristiques (l'AG).

Nous observons aussi qu'à partir d'une certaine taille de buffer, l'ordonnancement n'a plus aucun impact sur la performance car tous les nœuds peuvent se loger dans le buffer. C'est pour cette raison que l'ordonnancement donne la même performance que pour l'algorithme génétique (DBM-DQS) et Queen-Bee pour un espace buffer = $32Go$.

Pour mettre en évidence la performance de l'algorithme Queen-Bee lors du passage à l'échelle, nous augmentons le volume des données en faisant varier le SF de 1 à 120. Nous observons que le gain

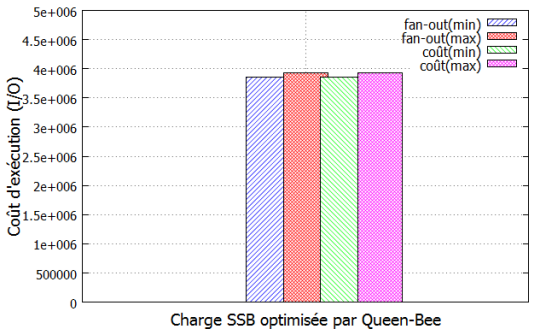

FIGURE 3.17 – Performance de la Queen-Bee par la variation des facteurs pris en compte par l'ordonnanceur

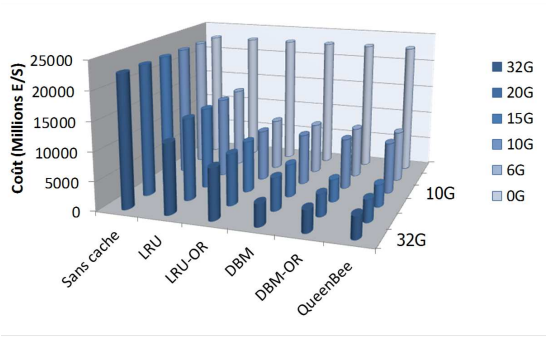

FIGURE 3.18 – Performance des différents algorithmes obtenue par la variation de la taille de la mémoire buffer

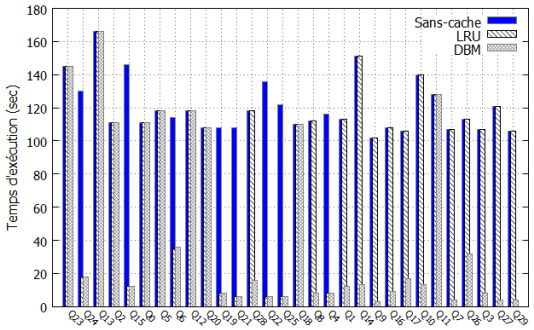

FIGURE 3.19 – Performance effective par requête en utilisant LRU

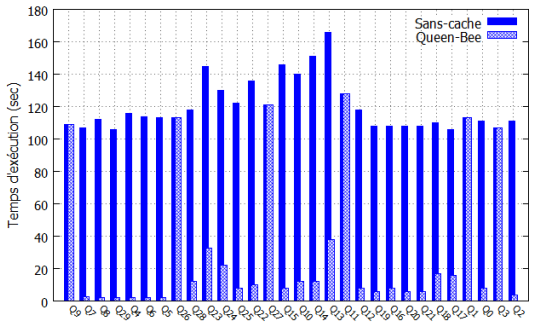

FIGURE 3.20 – Performance effective de l'algorithme Queen-Bee et LRU par requête

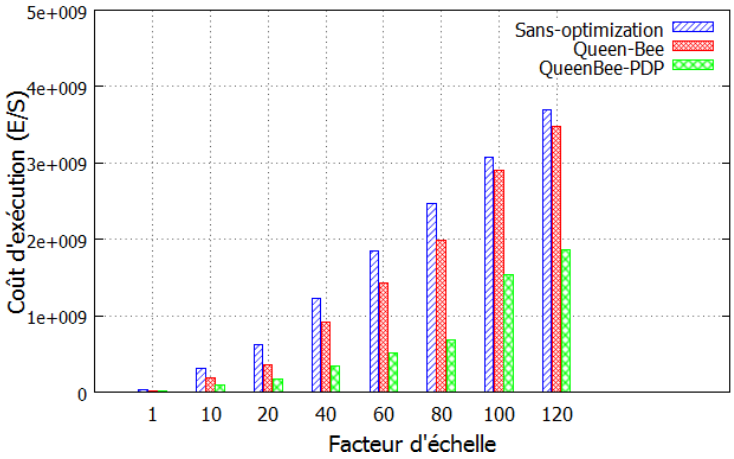

FIGURE 3.21 – Passage à l'échelle avec l'algorithme Queen-Bee

apporté reste limité, car la taille des objets à mettre dans le buffer devient plus importante, et le buffer est saturé plus rapidement. Le fait de pousser les sélections en bas du plan d'exécution n'est pas suffisant pour réduire la taille des nœuds intermédiaires quand le volume devient important. Pour remédier à ce problème, il faut optimiser la taille des résultats intermédiaires en effectuant une projection des attributs requis au lieu d'utiliser tous les attributs. Parmi les règles qui existent pour l'optimisation des plans de requêtes, il y a la descente des projections, aussi appelée : *Push-Down Projection* (PDP). Elle permet de projeter l'ensemble des attributs requis pour chaque opération algébrique, ce qui permet de réduire considérablement la taille des nœuds. En intégrant le PDP au processus d'optimisation dans la phase de préparation des entrées, et plus précisément la construction du MVPP, nous observons que le rendement de la Queen-Bee devient meilleur lors du passage à l'échelle comme le montre la figure 3.21.

L'une des motivations de notre proposition de l'algorithme Queen-Bee est l'élagage de l'espace de recherche pour réduire la complexité algorithmique.

La réactivité des algorithmes proposés a été évaluée. La figure 3.22 présente une comparaison entre les temps de réponse des algorithmes avec le même jeu de données. L'algorithme Queen-Bee fournit une réactivité comparable à celle de l'algorithme d'affinité, qui est plus beaucoup plus importante que celle des méta-heuristiques ($\approx 50\times$ plus rapide). Ces résultats confirment l'étude théorique et les hypothèses relatives à la complexité algorithmique.

6.3 Bilan de la simulation

Les résultats théoriques confirment les hypothèses discutées dans les sections précédentes. L'algorithme d'affinité est le plus rapide des algorithmes proposés. Cependant, son efficacité est limitée comparée aux heuristiques. Un bon compromis rapidité-efficacité est l'algorithme Queen-Bee, car il donne bien une meilleure performance dans un temps réduit.

L'algorithme Queen-Bee offre une efficacité considérable lors du passage à l'échelle, et propose de bonnes performances avec des bases de données volumineuses. Notons que la phase de préparation des entrées joue aussi un rôle important dans le processus d'optimisation.

6.4 Configuration et paramétrage

Afin de valider les résultats de simulation sous Oracle11g, nous adoptons le même jeu de données que celui utilisé dans le simulateur (SSB de SF=100 et 30 requêtes). Les requêtes sont exécutées suivant les recommandations dans chaque solution $(\mathcal{P}, BM)$ retournée par l'un des algorithmes. Pour effectuer cette validation, les requêtes doivent être réécrites afin d'assurer la prise en compte des stratégies de mise en buffer. Les paramètres du SGBD sont réglés afin de préparer le buffer.

Une fois une requête est réécrite, le buffer doit être alloué pour les nœuds pertinents correspondant à la requête avant d'être évaluée par l'optimiseur Oracle. Pour cela, plusieurs commandes SQL sont requises :

1. tuning du buffer (*alter system set db_cache size = 800 Mo;*);
2. effacement du buffer pour enlever tous les objets (*execute dbms_result_cache.flush;*);
3. mise en buffer d'un nœud (*...select/*+result cache*/ * from ...*);

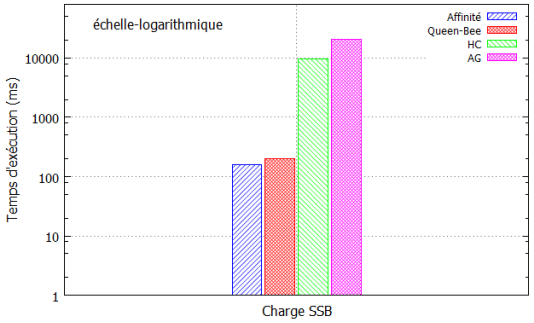

FIGURE 3.22 – Réactivité des algorithmes

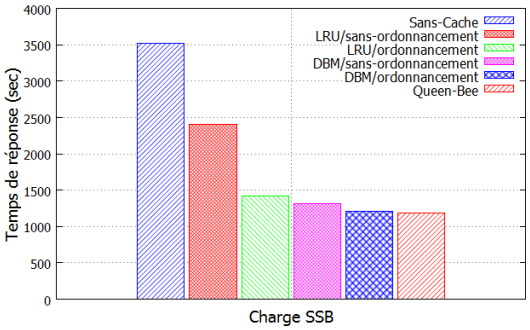

FIGURE 3.23 – Performances sous un SGBD réel

4. élimination d'un nœud du buffer (*exec dbms_result_cache.invalidate object(3)*).

Plus de détails concernant les processus de réécriture et de déploiement du script de gestion de buffer, sont présentés dans le chapitre 6.

6.5 Résultats réels

Pour comparer les deux politiques LRU et DBM, nous prenons la même charge de requête dans un ordre statique sous Oracle11g avec 6*Go* d'espace buffer. Les requêtes sont exécutées en utilisant : (1) aucune politique de gestion de buffer, (2) LRU, et (3) notre politique DBM. Dans la figure 3.19, nous observons que LRU donne une meilleure performance pour quelques requêtes mais elle n'est pas suffisante pour couvrir la totalité de la charge comme le DBM.

La performance des requêtes par Queen-Bee est présentée dans la figure 3.20. L'expérience montre que les requêtes élues comme queen-bee n'ont aucun gain, mais elles donnent une réduction importante de coût aux autres requêtes.

6.6 Bilan de la validation

En observant les résultats dans la figure 3.18 et figure 3.23, nous pouvons constater la grande similarité entre les résultats de simulation et celles de validation. Cela valide notre modèle de coût, puisqu'il semble à même de prévoir de manière assez fine le comportement d'un ensemble de requêtes en environnement réel. Nous constatons que l'interaction des requêtes est une propriété déterminante dans le processus d'optimisation par GBOR. Elle permet de guider l'optimiseur, de réduire la complexité algorithmique et d'augmenter la performance.

7 Conclusion

Le problème de GBOR est une combinaison de deux sous-problèmes *NP*-Complets. C'est pourquoi, nous avons choisi de l'aborder dans le cas hors-ligne pour étudier ses caractéristiques avant de l'adapter au cas plus complexes (en-ligne).

La combinaison de la Gestion du Buffer et d'Ordonnancement des Requêtes est l'un des problèmes classés NP-durs dans l'optimisation combinatoire (Section 4.2). La traiter nécessite non seulement de prendre en compte de l'interaction entre les requêtes, mais aussi d'utiliser des critères pointus pour évaluer différentes solutions potentielles dans un espace de recherche très large. Compte tenu de la taille de l'espace de recherche, deux phases d'élagage ont été proposées : une première dans la préparation des entrées, en représentant la charge par un MVPP, et une deuxième dont le rôle est l'élagage propres aux algorithmes de ce type.

La combinaison admet plusieurs scénarii envisageables. Nous en avons proposé quatre, dont trois présentent des limites en performance. Pour combler les lacunes de ses approches, un quatrième scénario a été proposé qui découle de l'analyse des trois précédents. Il constitue notre démarche de résolution de la GBOR dans le cas hors-ligne.

Le dilemme auquel on s'est confronté est qu'un algorithme basé sur l'affinité seule est rapide, simple mais peu efficace. Ceux basés sur un modèle de coût (e.g. Hill Climbing et Algorithme Génétique) sont plus efficaces mais très gourmands en temps de calcul. Nous avons donc recherché et identifié une solution intermédiaire pour remédier au manque d'efficacité des uns et à la gloutonnerie (en temps de calcul) des autres. C'est l'algorithme Queen-Bee qui constitue ce compromis. Il exploite à la fois l'affinité et le modèle de coût, et permet de diviser l'espace de recherche et de mener la recherche localement. Il permet ainsi d'avoir une complexité algorithmique réduite et une efficacité comparable à celles des algorithmes lourds.

Nous avons conduit une étude théorique de la complexité, qui a été consolidée par une étude expérimentale comprenant une simulation et une validation sur un SGBD réel.

Dans un système de base de données, les requêtes proviennent de différents utilisateurs. Dans un contexte en-ligne, l'arrivée de certaines requêtes peut coïncider avec le traitement d'autres requêtes déjà en file d'attente, ou en cours d'exécution. L'ordonnancement statique que nous avons traité dans ce chapitre ne permet pas de gérer l'aspect en-ligne, ce qui impose l'élaboration de nouvelles techniques adaptées pour la GBOR en-ligne. Ce problème fera l'objet d'une étude détaillée dans le chapitre 5.

Chapitre 4

Contribution de l'interaction sur la fragmentation horizontale

Sommaire

1 Introduction

Les systèmes de bases de données sont des environnements typiques pour l'interaction où les différentes composantes interagissent entre-elles. Certains travaux récents ont étudié l'impact mutuel des techniques comme la fragmentation horizontale avec les Index, ou les vues matérialisées [170, 201]. La sélection d'un ensemble de structures d'optimisation pour la conception physique exploite plusieurs composantes comme : les requêtes, les données, et les supports. La figure 4.1 illustre cette interaction liée au choix des structures d'optimisation.

D'autre part, dans l'étude des techniques d'optimisation menée dans le chapitre précédent, nous avons montré que la gestion du buffer et l'ordonnancement des requêtes exploitent l'interaction entre le support de stockage et les requêtes. Nous avons également montré, dans l'état de l'art, que la fragmentation horizontale exploite les requêtes sans prendre en considération leur ordre, ni le support de stockage. De ce fait, il serait intéressant d'étudier la combinaison des trois techniques afin de prendre en considération les différentes composantes à la fois, et d'étudier l'interaction des techniques entre-elles (cf. figure 4.1).

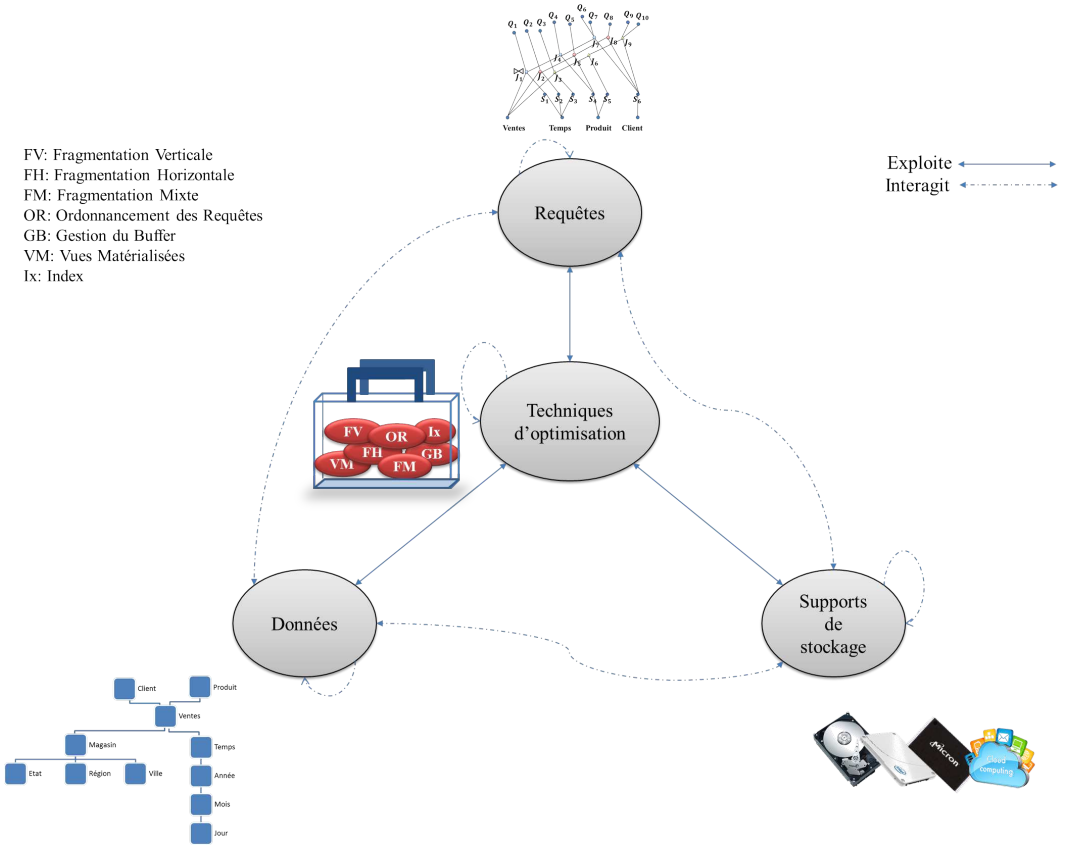

FIGURE 4.1 – Choix des techniques d'optimisation en fonction des interactions existant dans l'environnement

Ces trois techniques sont liées à des problèmes d'optimisation complexes ($\mathcal{NP}$-complets). De plus, leur combinaison introduit de nouveaux challenges que nous allons étudier dans ce chapitre.

Nous présentons dans ce chapitre une formalisation du problème joint et nous étudions les particularités et les difficultés liées à cette combinaison. Une démarche est proposée dont nous étudions les avantages et les inconvénients. A cause de la complexité élevée de la combinaison, et dans le but d'alléger les algorithmes, nous faisons un effort non seulement sur la partie algorithmique, mais aussi sur la structure de données induite par notre approche [62]. Les expérimentations donnent des résultats encourageants, ce qui ouvre de nouvelles perspectives pour l'exploitation de l'interaction.

D'un point de vue organisationnel, ce chapitre comporte cinq sections. La première section est consacrée au problème de la fragmentation horizontale, dont nous présentons la formalisation, la complexité, la représentation des schémas de fragmentation et le modèle de coût. Dans la deuxième section, nous proposons un algorithme de fragmentation horizontale par exploitation de l'interaction des requêtes. Ensuite, nous étudions la combinaison de la fragmentation avec la gestion du buffer et l'ordonnancement des requêtes. Cette étude nous permettra de définir une démarche pour la combinaison, que nous détaillons dans la section quatre. La dernière section présente les résultats de l'étude expérimentale.

2 Problème de la fragmentation horizontale dans les entrepôts de données

Dans cette section, nous présentons la complexité du problème de la fragmentation horizontale (formalisé dans le chapitre 2). Nous proposons ensuite une représentation dynamique des schémas de fragmentation horizontale. Nous terminons par présenter le modèle de coût adapté pour la fragmentation horizontale.

2.1 Complexité

La complexité du problème de la fragmentation horizontale a été établie dans le travail de Boukhalfa *et al.* [52]. Les auteurs prouvent que le problème de sélection de schéma de fragmentation horizontale primaire et dérivée pour un entrepôt de données est $\mathcal{NP}$-complet.

La preuve est établie par réduction au problème de 3-Partition. Pour cela, les auteurs ont défini un problème de décision simplifié à partir du problème de Fragmentation Horizontale de la façon suivante : une seule table de dimension D_1 est considérée comme ayant un seul attribut A. Le domaine de l'attribut A est décomposé en k sous-domaines $\{sd_1, sd_2, \ldots sd_k\}$ pour partitionner la table D_1. La fragmentation primaire de la table D_1 est propagée sur la table des faits F par une fragmentation dérivée. Le problème décisionnel est dit : *Problème de Fragmentation Horizontale à un Seul Domaine* (PFHSD), il consiste à partitionner la table des faits en plusieurs partitions telles que le nombre total de fragments ne dépasse pas une constante, et que le coût de la charge soit minimal. Le problème de Fragmentation Horizontale se ramène de cette façon au problème 3-Partition, qui est prouvé $\mathcal{NP}$-complet au sens fort. Les auteurs ont montré qu'une instance du problème 3-Partition admet une solution si et seulement si une instance du PFHSD admet une solution.

Les auteurs montrent aussi que le nombre de sous-domaines utilisés pour fragmenter l'entrepôt croît de façon exponentielle avec l'augmentation des attributs participant à la fragmentation. L'espace de recherche devient très grand et il s'avère quasi-impossible d'énumérer toutes les solutions potentielles. Il est donc impossible de trouver un algorithme pour résoudre le problème de FH en un temps polynomial

ou pseudo-polynomial [29].

2.2 Représentation des solutions potentielles

Les schémas de FH sont souvent représentés en utilisant un codage fixe [52] qui juxtapose des vecteurs représentant les attributs dans le niveau des sélections, où chaque vecteur représente la décomposition des domaines de chaque attribut de FH. La valeur de chaque cellule d'un vecteur donné représentant un attribut A_i^k appartient à $[1 \cdots n_i]$, où n_i représente le nombre de sous-domaines de l'attribut A_i^k. A partir de cette représentation, un schéma de FH de chaque table de dimension D_j est engendré de la façon suivante :

- Toutes les cellules d'un attribut de fragmentation A_i^k de D_j ont des valeurs différentes : ce qui signifie que tous les sous-domaines seront utilisés pour partitionner D_j.
- Toutes les cellules d'un attribut de fragmentation A_i^k ont la même valeur : ceci signifie que l'attribut ne participera pas dans le processus de fragmentation.
- Quelques cellules ont la même valeur : leurs sous-domaines seront fusionnés.

Pour le déroulement d'un exemple, nous adaptons le MVPP de l'exemple initial en modifiant les plans d'exécution de façon à avoir un ensemble de prédicats plus représentatifs pour les différents cas particuliers de notre étude.

Exemple. Le schéma dans la figure 4.2-b représente un codage pour les solutions potentielles. Les attributs sont *Quantité, Saison, Couleur, Type* et *Genre*. Les vecteurs sont décomposés en plusieurs cellules chacun pour couvrir les différents sous-domaines de chaque attribut. Par exemple, l'attribut *Type* comporte les sous-domaines *T1, T2* et un champs *else* pour les autres valeurs restantes. A partir de ce codage, un schéma peut être exprimé en respectant les règles de codage précédentes. Un exemple de schéma de FH est donné dans la figure 4.2-b. Dans ce schéma, la table *Temps* est partitionnée par une fragmentation horizontale primaire en trois fragments :

- $Temps_1 = \sigma_{\text{Saison='Automne'} \vee \text{Saison='Printemps'}}(Temps)$
- $Temps_2 = \sigma_{\text{Saison='Hiver'}}(Temps)$
- $Temps_3 = \sigma_{\text{Saison='Eté'}}(Temps)$

La table Produit est partitionnée en fonction des deux attributs *Couleur* et *Type*. Les fragments obtenus sont :

- $Produit_1 = \sigma_{\text{Couleur=couleur1} \wedge \text{Type} \in \{T1,T2\}}(Produit)$
- $Produit_2 = \sigma_{\text{Couleur} \neq \text{couleur1} \wedge \text{Type} \in \{T1,T2\}}(Produit)$
- $Produit_3 = \sigma_{\text{Couleur=couleur1} \wedge \text{Type} \notin \{T1,T2\}}(Produit)$
- $Produit_4 = \sigma_{\text{Couleur} \neq \text{couleur1} \wedge \text{Type} \notin \{T1,T2\}}(Produit)$

La table *Client* ne participe pas à la FH car l'attribut *Genre* impliqué dans le codage a la même valeur dans toute ses cellules.

La table des faits est dans un premier temps partitionnée en deux fragments par le biais d'une fragmentation horizontale primaire. Bien que la FHP soit principalement destinée aux tables de dimension, elle peut servir dans certains cas à découper la tables des faits lorsque où cette table comporte des mesures employées dans les prédicats de sélection des requêtes (cf. figure 4.7). Le découpage produit les deux fragments suivants :

Quantité	< 100	$\leq 100 ; < 1000$	else	
Saison	Automne	Hiver	Printemps	Eté
Couleur	Couleur1	else		
Type	T1	T2	else	
Genre	Féminin	Masculin		

(a)

Quantité	1	2	2	
Saison	1	2	1	3
Couleur	1	2		
Type	1	1	2	
Genre	1	1		

(b)

FIGURE 4.2 – Exemple de schéma de codage (b) obtenu à partir des attributs de sélection et leurs sous-domaines (a)

- $Ventes_1 = \sigma_{\text{Quantité}<100}(Ventes)$
- $Ventes_2 = \sigma_{\text{Quantité}\geq 100}(Ventes)$

Dans un deuxième temps, la fragmentation horizontale dérivée est appliquée sur la table des faits, et plus précisément aux deux fragments $Ventes_1$ et $Ventes_2$:

- $Ventes_{1,1} = (Ventes_1 \ltimes Produit_1 \ltimes Temps_1 \ltimes Client)$
- $Ventes_{1,2} = (Ventes_1 \ltimes Produit_1 \ltimes Temps_2 \ltimes Client)$
- $Ventes_{1,3} = (Ventes_1 \ltimes Produit_1 \ltimes Temps_3 \ltimes Client)$
- $Ventes_{1,4} = (Ventes_1 \ltimes Produit_2 \ltimes Temps_1 \ltimes Client)$
- $Ventes_{1,5} = (Ventes_1 \ltimes Produit_2 \ltimes Temps_2 \ltimes Client)$
- $Ventes_{1,6} = (Ventes_1 \ltimes Produit_2 \ltimes Temps_3 \ltimes Client)$
- ...
- $Ventes_{2,12} = (Ventes_2 \ltimes Produit_4 \ltimes Temps_3 \ltimes Client)$

Les prédicats représentant chaque sous-schéma sont :

- ss_1 = ((Quantité < 100) $\wedge$ (Saison = Automne $\vee$ Saison = Printemps) $\wedge$ (Couleur = couleur1 $\wedge$ Type $\in \{T1, T2\}$))
- ss_2 = ((Quantité < 100) $\wedge$ (Saison = Automne $\vee$ Saison = Printemps) $\wedge$ (Couleur $\leq$ couleur1 $\wedge$ Type $\in \{T1, T2\}$))
- ss_3 = ((Quantité < 100) $\wedge$ (Saison = Hive) $\wedge$ (Couleur = couleur1 $\wedge$ Type $\in \{T1, T2\}$))
- ...
- ss_{24} = ((Quantité ≥ 100) $\wedge$ (Saison = Eté) $\wedge$ (Couleur $\neq$ couleur1 $\wedge$ Type $\notin \{T1, T2\}$))

Cependant, ce codage est statique et ne permet pas d'étendre un schéma de fragmentation pour l'adapter aux nouvelles requêtes. De plus, il impose plusieurs étapes complexes de préparation pour extraire les prédicats requis (cf. chapitre 2).

Pour éviter le codage statique, nous proposons un codage incrémental conçu à l'arrivée des requêtes du MVPP. Avant de proposer ce codage, nous commençons par présenter l'algèbre sous-jacente, qui a un double rôle : (1) fournir un schéma de codage, et (2) produire un schéma de fragmentation.

2.2.1 Schéma de codage

Dans la majeure partie des SGBD, la FH est supportée par deux principales primitives pour gérer les fragments : *Merge* et *Split*. La première consiste à fusionner deux partitions en une seule, et la deuxième permet d'éclater une partition en deux (cf. figure 4.3). Ces opérations sont effectuées au niveau physique.

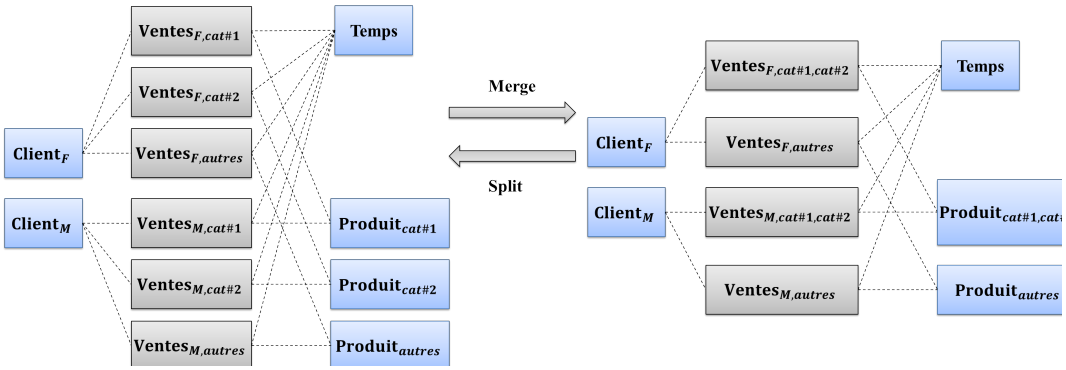

FIGURE 4.3 – Le principe d'éclatement et de fusion des fragments lors de la conception physique par Split/Merge

Il serait intéressant de les propager au niveau structurel afin de fournir un schéma de codage incrémental. Nous utilisons donc ses fonctions afin de construire le codage, et de manipuler les schémas de fragmentation.

Le codage proposé est construit à partir d'un ensemble de requêtes élues, en insérant au fur et à mesure les nouveaux prédicats trouvés dans le schéma de codage. Pour intégrer les nouveaux prédicats, il faut insérer à la fois les attributs et les sous-domaines participant à la FH dans le schéma de codage. Pour cela, nous proposons les deux fonctions suivantes :

a) Split Horizontal : permet d'insérer un nouvel attribut trouvé dans les prédicats des requêtes :

$$Split\_Horizontal(AS, a_i) = AS'/AS' = AS \cup \{a_i\}$$

Où a_i représente le nouvel attribut trouvé qui sera inséré dans l'ensemble d'attributs de sélection AS. Par exemple, dans la figure 4.4, un Split Horizontal est appliqué en passant de la sélection S_1 à S_2 de la requête Q_1. Le nouvel attribut trouvé $Saison$ est inséré avec les sous-domaines trouvés et un champ *else* supplémentaire pour assurer la complétude.

b) Split Vertical : permet d'insérer un nouveau sous-domaine en éclatant (fragmentant) le champ *else* dans le vecteur de l'attribut concerné.

$$Split\_Vertical(SD_{ij}, a_i) = SD_{ij} \cup \{else\}$$

Où SD_{ij} est l'ensemble de nouveaux sous-domaines trouvés par la requête Q_j qui seront insérés dans le champs *else*. Le champs sera éclaté en plusieurs cellules pour couvrir les sous-domaines contenu dans SD_{ij} et en rajoutant un champs *else*. Dans la figure 4.4, un Split Vertical est appliqué en passant de la sélection S_1 à S_2 de la requête Q_2. L'attribut *Saison* contiendra un nouvel ensemble de sous-domaines qui sont : *Hiver* et *Printemps*.

Afin d'élaguer l'espace de recherche par les attributs et les sous-domaines requis par les requêtes, il serait intéressant d'utiliser uniquement les attributs de certaines requêtes qui figurent dans le MVPP. Pour cela, nous proposons dans la Section 3.2 une approche permettant d'élire un ensemble de requêtes pour la FH. À partir de l'ensemble des requêtes élues, la construction du codage est faite de manière incrémentale en explorant les plans de requêtes élues du MVPP un à un, et en adaptant le codage avec

les nouveaux attributs et sous-domaines. L'extraction de l'ensemble des attributs de sélection (S) se fait à partir du MVPP, et le codage se construit à l'aide des deux fonctions *Split Vertical* et *Split Horizontal*.

Le processus d'encodage commence par un ensemble de requêtes choisies par l'algorithme de fragmentation horizontale dites *requêtes élues*. Ces requêtes sont représentées par un MVPP. Rappelons que les sélections sont poussées en bas des plans pour réduire la taille des résultats intermédiaires. Le niveau des sélections est considéré dans le MVPP pour obtenir l'ensemble des prédicats, et ces requêtes élues sont considérées une à une pour construire dynamiquement le schéma de codage.

Initialement, le domaine de l'attribut trouvé est composé de deux sous-domaines : le premier est décrit par l'opération de sélection et le second pour le champ *else* afin d'assurer la complétude. Le champ *else* peut être scindé quand de nouveaux prédicats sont trouvés (cf. figure 4.4) en utilisant la fonction *Split Vertical*.

Le principe de ce codage est de commencer par un ensemble d'attributs vide, et d'ajouter les attributs de sélection de chaque requête en créant de nouveaux vecteurs, chacun contenant un seul champ. Chaque fois qu'une sélection est trouvée dans la requête en cours, l'une des trois opérations est effectuée :

1. Si l'attribut n'existe pas dans le schéma, appliquer un Split Vertical pour étendre le schéma en construisant un nouvel attribut. Le champ est scindé en plusieurs parties pour couvrir les sous-domaines trouvés. Un champ *else* est ajouté afin d'assurer la complétude si les sous-domaines trouvés ne couvrent pas toutes les valeurs possibles du domaine de l'attribut. La figure 4.4 illustre l'ajout d'attributs dans l'ensemble initialement vide S par la requête Q_1. L'attribut *Quantité* est trouvé dans la première sélection. Il est donc inséré dans le schéma initialement vide. La deuxième sélection retrouve le nouvel attribut *Saison*. Il est donc inséré de la même manière, ainsi de suite.

2. Si l'attribut existe déjà, appliquer le Split Horizontal sur le champ *else* pour rajouter les nouveaux sous-domaines. Si les sous-domaines trouvés ne couvrent pas toutes les valeurs possibles du domaine de l'attribut, un champ *else* est inséré pour la complétude. La figure 4.4 montre un exemple de sous-domaines ajoutés par la requête Q_2 pour l'attribut *Saison* qui existe déjà dans le codage. Le champs *else* est éclaté en trois nouveaux champs : *Hiver*, *Printemps* et *Eté*. Comme les valeurs trouvées sont les seules valeurs dans le domaine de l'attribut, alors le champ *else* n'est pas inséré.

3. Finalement, si l'administrateur connaît la valeur restante dans les champs *else*, il la remplace par cette valeur. Dans la figure 4.4, le *else* est remplacé par *Masculin* pour l'attribut *Genre*, et par *Eté* pour l'attribut *Saison*.

A la fin de cette phase, un schéma de codage est produit qui contient l'ensemble des attributs de sélection et les sous-domaines associés. Pour appliquer la procédure de codage sur notre exemple, nous commençons par un ensemble vide d'attributs. Quand la première requête élue Q_1 arrive, deux nouveaux attributs et leurs sous-domaines dans Q_1 sont ajoutés. La requête Q_2 ne contient pas de nouveaux attributs mais de nouveaux prédicats qui déclenchent un Split Horizontal sur le schéma existant. Finalement, quand la dernière requête est traitée, le schéma est ajusté avec les valeurs connues remplaçant les *else* (cf. figure 4.4).

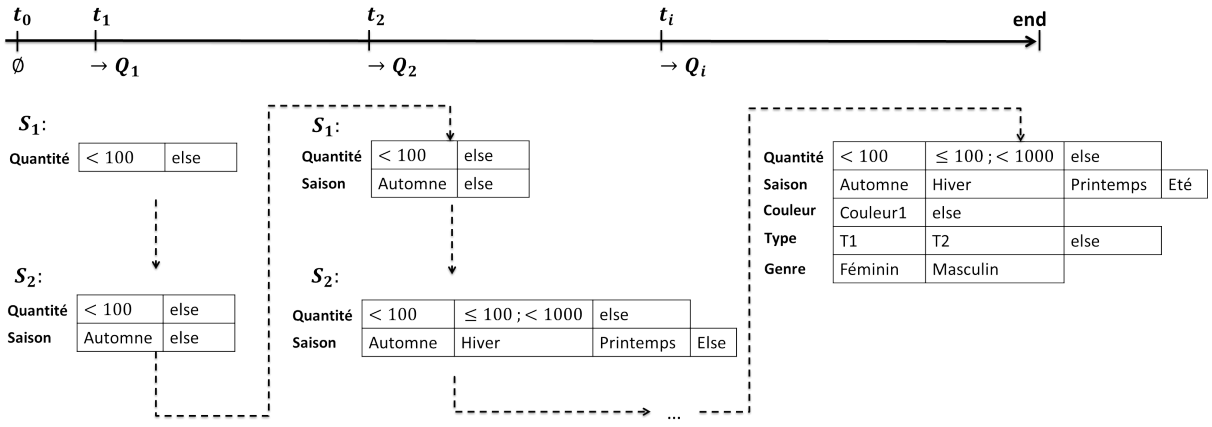

FIGURE 4.4 – Codage Incrémental par Split Horizontal et Vertical sur les sous domaines et les attributs

2.2.2 Sélection du schéma de Fragmentation Horizontale

Le codage obtenu, bien qu'il couvre tous les prédicats requis par la charge, mais ne représente pas le meilleur sous-schéma de Fragmentation Horizontale car il produit un nombre de fragments élevé qui dépasse souvent le seuil W ce qui viole la contrainte de maintenance. Pour remédier à cela, il est nécessaire d'appliquer des fusions et des éclatements sur le schéma de codage pour sélectionner un meilleur schéma de Fragmentation.

Pour améliorer le schéma de fragmentation à partir du codage obtenu, nous utilisons les deux fonctions Split et Merge :

a) La fonction Merge : si le schéma courant dépasse le nombre de fragments autorisé, la fonction Merge est appliquée pour fusionner deux fragments en un seul. Plus formellement :

$$Merge(sd_i^k, sd_j^k, A^k, PS) \rightarrow PS'$$

Où la fonction *Merge* est appliquée sur deux sous-domaines sd_i^k et sd_j^k de l'attribut A^k pour les fusionner en une même partition.

b) La fonction Split : si le schéma courant ne dépasse pas le nombre de fragments autorisé, la fonction Split est appliquée en éclatant un fragment en deux. Plus formellement :

$$Split(sd_i^k, A^k, PS) \rightarrow PS'$$

Où la fonction *Split* est appliquée sur le sous-domaine sd_i^k pour le scinder en deux sous-domaines si et seulement s'il couvre au moins deux autres sous-domaines de l'attribut A^k.

2.2.3 Avantages du codage proposé

En plus de la représentation des schémas de Fragmentation Horizontale, notre codage dynamique offre de nouveaux avantages :

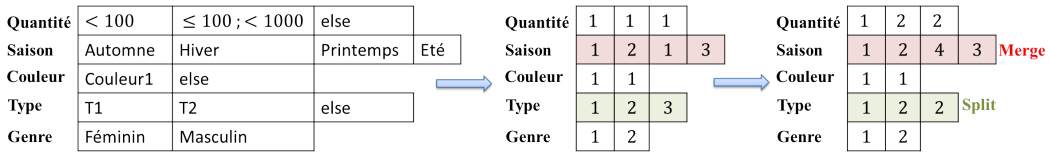

FIGURE 4.5 – Sélection d'un nouveau schéma de FH par mouvements de Split et de Merge

– **Le passage à l'échelle :**
L'arrivée de nouvelles requêtes dans la charge impose l'adaptation du schéma de Fragmentation Horizontale. L'augmentation du nombre de requêtes (Big Data) nécessite l'adaptation rapide du schéma. Cette adaptation peut être assurée par notre structure de données flexible qui, grâce à son algèbre associée, permet d'insérer de nouveaux attributs et sous-domaines des nouveaux prédicats trouvés.

– **L'élagage de l'espace de recherche :**
Le principe de construction de la structure de données dynamique se base sur l'interaction. Les requêtes élues pour guider le processus de fragmentation sont utilisées pour construire le codage avec les prédicats requis. Ces prédicats élaguent l'espace de recherche en réduisant le nombre de schémas potentiels.

– **Autres avantages : (Génération de Profils)**
La structure de données dynamique est exploitée plus loin dans ce chapitre (Section 4) pour la génération de différents profils à savoir les profils de requêtes, de sous-schémas et des facteurs de sélectivité. Ces profils serviront lors de l'évaluation des solutions par le modèle de coût à l'identification des sous-schémas valides pour les requêtes ainsi que l'ordre des jointures.

2.3 Modèle de coût pour la FH

Le partitionnement horizontal décompose le schéma en étoile en N sous-schémas en étoile ($N > 0$). L'évaluation de la qualité d'un schéma de partitionnement obtenu par un algorithme s'effectue par un modèle de coût mathématique. La métrique estime le nombre d'Entrées/Sorties (E/S) requis pour l'exécution d'une requête sur le nouveau schéma fragmenté. Le coût global de la charge est obtenu par la somme des coûts de toutes les requêtes sur le nouveau schéma.

$$Cout\_Total = \sum_{i=0}^{n} cout(Q_i)$$

L'exécution d'une requête sur un schéma fragmenté s'effectue selon les étapes suivantes comme le montre la figure 4.6 :

1. Identification des sous-schémas valides (pertinents)
2. Exécution des requêtes sur chaque sous-schéma valide
3. Union des résultats partiels

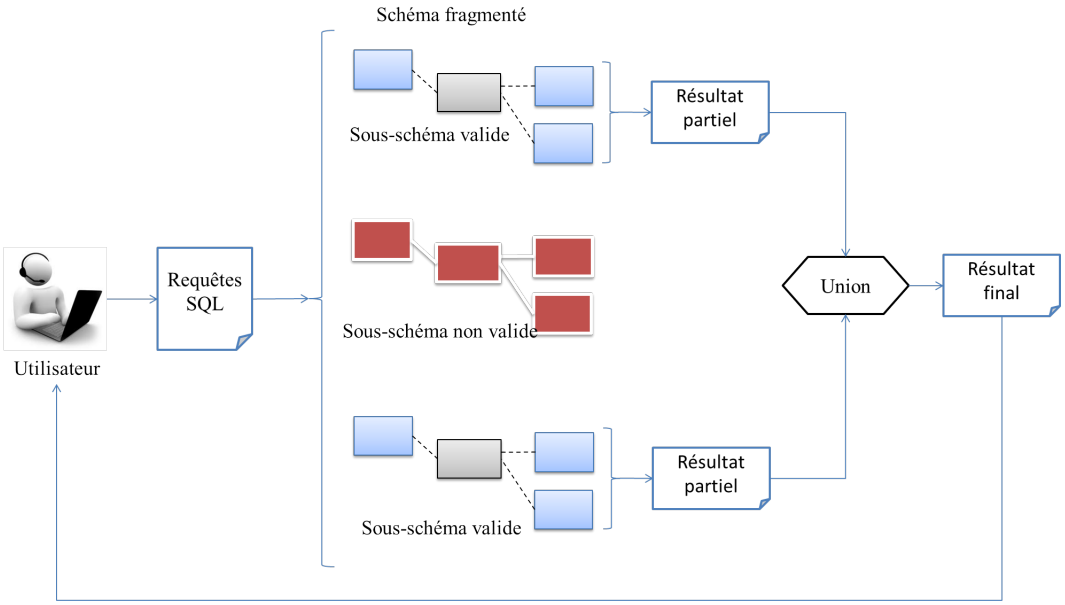

FIGURE 4.6 – Identification des sous-schémas valides et exécution d'une requête sur un schéma fragmenté

2.3.1 Identification des sous-schémas valides

L'exécution d'une requête sur un schéma fragmenté horizontalement doit s'effectuer uniquement sur les sous-schémas valides, car les sous-schémas non valides ne contiennent aucune instance pertinente pour la requête. L'identification des sous-schémas valides permet d'éliminer les sous-schémas inutiles du plan d'exécution de la requête et ainsi de réduire les Entrées/Sorties (cf. figure 4.6).

Nous définissons la fonction booléenne $V(Q_i, sc_j)$ qui détermine si le sous-schéma ss_j est valide ou non pour la requête Q_i.

L'identification d'un sous-schéma ss_j valide pour une requête Q_i suit deux étapes : (1) identification des fragments de dimensions valides pour Q_i, et (2) identification du fragment de faits correspondant.

Les fragments de dimension sont identifiés en comparant les prédicats définis sur Q_i et ceux participant dans la clause du fragment de dimension. Le fragment de faits correspondant est construit à partir des fragments de dimension (cf. page 26).

2.3.2 Exécution des requêtes sur les sous-schémas valides

Une fois les sous-schémas valides identifiés pour une requête Q_i, l'exécution de cette requête nécessitera (1) l'identification du type de correspondance entre le fragment et la requête, et (2) l'identification des jointures à effectuer.

Soit Q_i une requête et F_j un fragment de faits dans un sous-schéma ss_j valide pour Q_i. Soient p et q représentant respectivement la conjonction de prédicats de la requête Q_i et le fragment de faits F_j. Il existe trois types de correspondance définis dans la littérature [52] : correspondance totale, correspondance partielle et pas de correspondance.

- Il existe une correspondance partielle entre F_j et Q_i si et seulement si ss_j est valide pour Q_i et il existe au moins une instance dans F_j participant à Q_i.
- Il existe une correspondance totale entre F_j et Q_i si et seulement si toutes les instances de F_j participent à la réponse à Q_i. Autrement dit : $p \rightarrow q$.
- Il existe aucune correspondance entre F_j et Q_i si et seulement si aucun instance de F_j ne participe à Q_i. Formellement, $p \wedge q$ est ne peut être satisfait. Dans ce cas, le sous-schéma ss_j n'est pas valide pour Q_i.

Après l'identification de correspondance, les jointures à effectuer sont déterminées.

Dans les requêtes de jointure en étoile, les tables de dimensions impliquées dans la jointure sont celles ayant un attribut référencé dans (1) la Clause SELECT ou (2) la Clause WHERE. Dans le premier cas, la table est automatiquement jointe avec le fragment de faits du sous-schéma valide. Pour le deuxième cas, la table n'est pas automatiquement jointe, et nécessite une jointure avec le fragment de faits selon le type de correspondance entre la requête Q_i et le sous-schéma ss_j :

- Pas de correspondance, alors le fragment n'est pas valide et Q_i ne s'exécutera pas sur ss_j.
- Correspondance totale, dans ce cas toutes les jointures ont été pré-calculées et aucune dimension ne sera jointe avec le fragment de faits. Cependant, dans le cas où un attribut de dimension est référencé dans la Clause SELECT alors la table de dimension doit être jointe avec le fragment de faits.
- Correspondance partielle, dans ce cas les tables de dimension liées au fragment de faits ou ayant un attribut référencé dans la Clause SELECT sont jointes.

Le modèle de coût défini dans le cas non fragmenté dans le chapitre 3 est employé pour l'estimation du coût des opérations de jointures sur tous les sous-schémas en étoile tel que le fragment de fait représente la table des faits initiale et les fragments de dimension qui lui sont liés représentent les tables de dimension. Ce modèle de coût reste applicable car la fragmentation horizontale permet de conserver le schéma de l'entrepôt (en étoile).

2.3.3 Union des résultats partiels

La dernière opération à effectuer est l'union de tous les résultats partiels obtenus en exécutant la requête Q_i sur chaque sous-schéma valide. Ainsi, le coût d'exécution de la requête obtenu à partir des sous-schémas valides ($Sub\_C(Q_i, ss_j)$) et le coût de l'union ($Union\_C$) de ses résultats partiels :

$$cout(Q_i) = \sum_{j=0}^{Nsc} (Sub\_C(Q_i, ss_j) \times V(Q_i, ss_j)) + Union\_C(Q_i)$$

3 Fragmentation horizontale par exploitation de l'interaction des requêtes

Dans cette section, nous donnons la motivation qui nous a menés à la proposition de l'algorithme d'élection des requêtes. Nous présentons le principe de l'algorithme et la méthodologie d'intégration de ce processus dans la sélection d'un schéma de fragmentation horizontale.

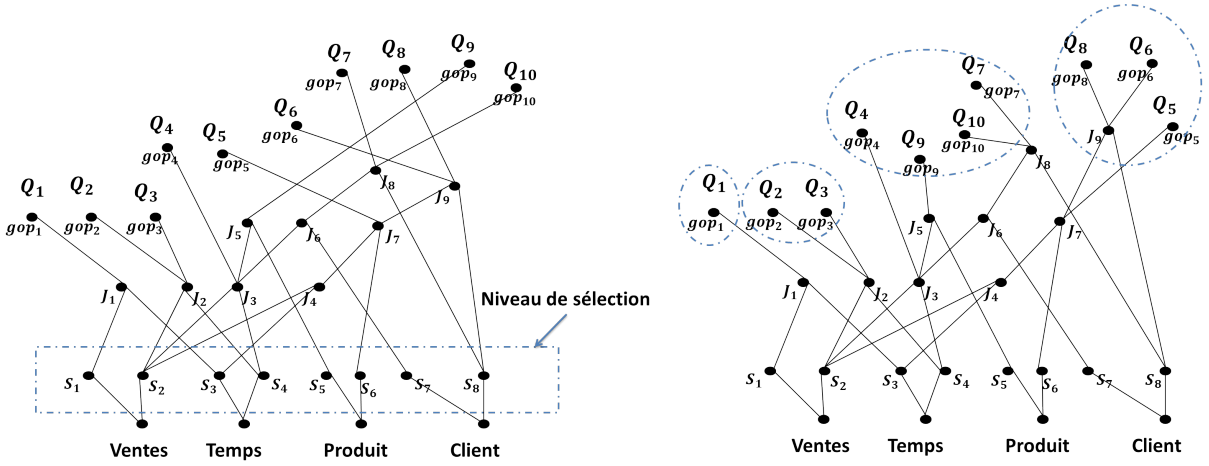

FIGURE 4.7 – Exemple de MVPP avec exploitation de l'interaction

3.1 Motivation

Pour mieux illustrer l'idée à la base de notre proposition, nous considérons l'exemple du MVPP présenté en figure 4.7. Rappelons que nous appliquons une approche *Rule-Based* pour faire descendre les opérations de sélection sont en bas des plans lors de la construction du MVPP. Ce graphe contient quatre principaux niveaux : **(a)** les nœuds feuilles représentant les tables, **(b)** les nœuds de sélection pouvant participer à la fragmentation ((e.g., $\{s_1, s_2 \dots s_8\}$)), **(c)** les opérations binaires (e.g., $\{j_1; j_2 \dots; j_9\}$) et **(d)** les groupements, tri et projections (e.g., $\{gop_1; gop_2; \dots gop_{10}\}$).

La jointure j_3 des tables *Ventes* et *Temps* est utilisée simultanément par les requêtes Q_4, Q_7, Q_9 et Q_{10}. Pour satisfaire la requête Q_4 en optimisant les sélections $\{s_2, s_4\}$ impliquées dans la jointure j_3, le coût d'exécution de j_3 doit être considérablement réduit. Ceci a des conséquences non seulement sur la requête Q_4, mais aussi sur toutes les requêtes qui lui sont corrélées dans ce nœud : Q_7, Q_9 et Q_{10}. Parmi les techniques d'optimisation existantes, celle qui s'adapte le mieux avec ce niveau de nœuds, i.e. les sélections, est la fragmentation horizontale.

Propager le gain en performance tout au long des plans de requêtes, tout en respectant le nombre maximal de fragments finaux est une tâche difficile. Pour ce faire, nous exploitons l'interaction entre les requêtes, pour grouper les requêtes en quatre sous-ensembles disjoints selon les jointures communes : $\{Q_1\}; \{Q_2; Q_3\}; \{Q_4; Q_7; Q_9; Q_{10}\}; \{Q_5; Q_6; Q_8\}$.

On observe que le nombre de groupes obtenu est égal au nombre de jointures directes avec la table des faits (quatre dans l'exemple). La raison est que chaque paire de requêtes dans le même groupe est corrélée au moins dans la première jointure, qui est très coûteuse. Si cette première jointure est optimisée par la FH sur ses sélections, toutes les requêtes du même groupe vont bénéficier de cette sélection optimisée. L'observation de ces faits nous permet de déduire l'impact de telles propriétés sur le déroulement du processus de FH. Dans ce sens, nous proposons une nouvelle approche pour la FH en exploitant l'interaction des requêtes dans une démarche *diviser pour régner*.

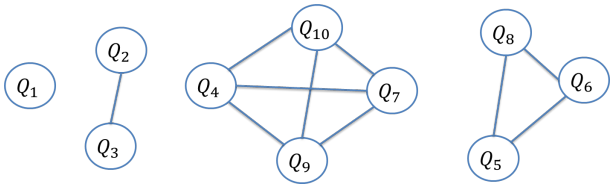

FIGURE 4.8 – Groupes de requêtes obtenus à partir du MVPP

3.2 Principe de la FH par interaction

Dans cette section, nous détaillons notre proposition en plusieurs niveaux. Nous commençons par décrire le codage utilisé et la façon dont nous traitons les attributs de sélection. Puis, nous présentons comment exploiter l'interaction des requêtes dans le processus de FH en privilégiant les nœuds de jointure communs. Pour traiter ces nœuds, nous proposons une approche *diviser pour régner* dite : *Algorithme de Requête Elue* (ARE) dont le principe est d'élaguer l'espace des attributs pertinents et leurs sous-domaines. Nous décrivons enfin comment l'algorithme ARE dirige la FH et nous donnons ses étapes.

3.3 Privilégier les nœuds de jointures avec grande interaction

A partir de l'ensemble des attributs candidats obtenu, un schéma de fragmentation doit être produit afin de réduire au mieux le coût d'exécution. Comme nous l'avons déjà mentionné, les solutions existantes sont soit (1) simples comme les algorithmes dirigés par l'affinité, soit (2) utilisant des méta-heuristiques basées sur des modèles de coût comme le Recuit Simulé, le Hill Climbing ou encore l'Algorithme Génétique [52]. Notre étude précédente (cf. chapitre 3) nous a montré que l'exploitation de l'interaction permet de produire un autre type d'algorithme à la fois efficace et réactif. Nous proposons de rajouter un nouveau critère pour conduire le processus de FH, pour parvenir à un compromis entre les deux approches, avec une efficacité meilleure et une complexité algorithmique réduite.

Comme nous avons déjà vu dans l'exemple de motivation, l'opération de jointure est très coûteuse. Notons toutefois que les requêtes en interaction partagent au moins la première jointure. La considération du nœud de première jointure est cruciale car il servira toutes les requêtes partageant ce nœud. Pour ressortir et exploiter l'interaction des requêtes et réduire la complexité des algorithmes de résolution, nous proposons l'*Algorithme de Requête Elue* (ARE) pour ressortir l'interaction entre les requêtes.

A partir du MVPP, l'ARE fournit l'ensemble des groupes de requêtes corrélées, tel que chaque couple de requêtes ayant au moins un nœud de jointure commun est dans le même groupe. La figure 4.8 présente l'ensemble des groupes obtenus à partir du MVPP de l'exemple. Dans chaque groupe, l'algorithme élit une requête qui permettra d'aiguiller le processus de FH. Cette requête, que l'on appelle *Requête Elue* (RE) est considérée comme la plus importante dans son groupe. Le choix de la requête élue RE_i du groupe i est effectué selon le critère du coût d'exécution. Dans chaque groupe, la requête ayant le coût minimal est élue. Généralement, la requête ayant un coût minimal contient moins d'opérations de jointure et de sélections. En conséquence, elle peut contribuer à l'élagage des prédicats de sélection. Le nombre de nœuds communs n'est pas pris en compte dans le choix de la requête élue car il existe au moins un nœud partagé entre cette requête et les requêtes du même groupe.

3.4 Effet des requêtes élues sur le Codage : Élagage de l'Espace de Recherche

L'Algorithme de Requêtes Élues vise à fournir un sous-ensemble d'attributs de sélection et de sous-domaines pertinents pour piloter la FH. L'ensemble de requêtes élues fourni permet d'élaguer l'espace de recherche et d'éliminer les possibilités les moins efficaces comme les sélections et les jointures les moins fréquentes.

Les attributs obtenus à partir du MVPP de la phase précédente (par Algorithme de Requêtes Élues), seront élagués encore une fois en considérant dans un premier temps, seulement les attributs requis par les requêtes élues. Dans le processus de fragmentation, l'algorithme satisfait les requêtes élues au lieu de choisir aléatoirement les prédicats. Cette phase d'élagage réduit la complexité de l'algorithme en réduisant la taille de l'espace de recherche. Les détails de l'algorithme de FH en utilisant ARE sont présentés dans la section suivante.

3.5 Exploiter les requêtes élues dans la Fragmentation Horizontale

L'algorithme de FH que nous proposons, dit *Requêtes Elues pour la Fragmentation Horizontale* (REFH), s'appuie sur le MVPP, et effectue l'élagage de l'ensemble des attributs de sélection selon les requêtes apportant le plus de gain en utilisant ARE.

L'ARE regroupe les requêtes en sous-ensembles disjoints, puis trie les requêtes à l'intérieur par coût minimal pour retourner l'ensemble de requêtes élues RE_i de chaque groupe i. L'ensemble des requêtes élues (RE) obtenu est ensuite trié par ordre de coûts décroissants. Ceci nous mène à traiter les requêtes les plus coûteuses avant d'épuiser le nombre de fragments possible W. L'ARE traite le premier ensemble de requêtes élues et effectue l'élagage selon les besoins de ses requêtes, i.e. seuls les attributs requis par les requêtes élues sont pris, les autres sont ignorés. Chaque sous-domaine est indexé par le nombre de requêtes élues qui l'utilise.

Soit u_{ij} le nombre de requêtes élues utilisant le sous-domaine sd_j de l'attribut a_i, et k_i la valeur maximale de u_{ij} pour l'attribut a_i. L'ensemble d'attributs est trié par usage décroissant (valeur de k_i) afin de commencer le partitionnement par les attributs les plus utilisés avant que W ne soit épuisé. Après avoir trié les attributs, chaque attribut a_i subit des opérations de fusion/éclatement (Merge/Split) de la manière suivante :

1. Les sous-domaines qui utilisés par aucune requête élue ($u_{ij} = 0$) sont regroupés en une même partition P_0;

2. Les sous-domaines les plus utilisés, ayant k_i requêtes élues y accédant, sont regroupés en une seule partition P_k (k_i est la valeur maximale d'usage de l'attribut a_i);

3. Finalement, si $N > W$ ou $k_i \neq 0$ alors, les sous-domaines restant sont fusionnés avec la partition P_0; sinon, les sous-domaines accédés par $(k_i - 1)$ requêtes élues sont regroupés dans une nouvelle partition. La même opération est répétée jusqu'à ce que $k_i = 0$ ou $N > W$. Ceci permet de créer des partitions en se basant sur les besoins des requêtes les plus importantes.

Si la fragmentation est toujours possible ($N < W$), les partitions obtenues sont éclatées selon la corrélation entre les requêtes accédant chaque sous-domaine de la partition : Soit deux sous-domaines sd_1 et sd_2 d'un même attribut a_i référencés respectivement par deux sous-ensembles de requêtes $Q^1 \subset Q$

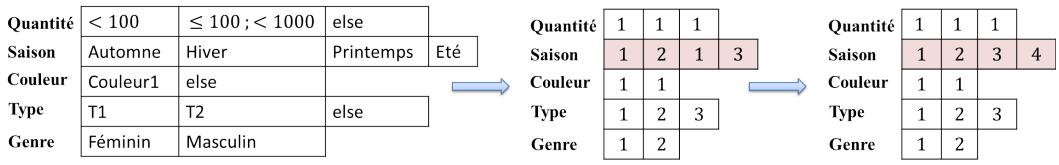

FIGURE 4.9 – Eclattement des partitions par corrélation des requêtes

et $Q^2 \subset Q$. Si $Q^1 \cap Q^2 = \emptyset$ alors les deux sous-domaines ne sont pas corrélés et doivent appartenir à deux fragments différents. Ainsi, s'il existe un fragment Fg_k contenant les deux sous-domaines à la fois, i.e. les cellules correspondantes ont la même valeur, alors les deux sous-domaines sont éclatés pour fournir deux nouveaux fragments.

Exemple. Si dans le schéma de la figure 4.9 les sous-domaines *Hiver* et *Automne* sont référencés respectivement par les sous-ensembles de requêtes $\{Q_2, Q_6, Q_8\}$ et $\{Q_4, Q_5, Q_7\}$ alors un *Split* est appliqué à l'attribut *Saison* pour obtenir *Hiver* et *Printemps* dans deux fragments différents (cf. figure 4.9).

Si $N < W$ après ces opérations de fusion et d'éclatement en considérant seulement les requêtes élues, alors la fragmentation reste possible. Pour cette raison, le processus d'optimisation enchaîne en rajoutant un nouvel ensemble de requêtes élues à satisfaire.

L'ensemble suivant des requêtes est celui des successeurs des requêtes élues courantes. Si au moins un groupe contient encore un successeur (parmi les requêtes triées par coût minimal), alors un nouvel ensemble de requêtes élues est généré par les successeurs trouvés de tous les groupes. Le même processus est appliqué pour étendre le schéma de codage aux nouveaux attributs et sous-domaines requis de façon incrémentale. Le processus est itéré jusqu'à ce que $N < W$ ou qu'aucune requête à élire ne reste.

3.6 Bilan

Les approches existantes pour la Fragmentation Horizontale font généralement l'effort sur le niveau algorithmique. Bien que ces approches aient amélioré la première génération d'algorithmes de résolution, elles souffrent d'une lacune majeure qui est l'ignorance de l'interaction entre les requêtes. Pour cela, nous avons revisité le problème de Fragmentation Horizontale pour étudier ce critère et son impact sur le processus de partitionnement.

Le problème de FH est prouvé $\mathcal{NP}$-complet [29], ce qui rend tous les algorithmes qui s'adresse à ce problème lourds et gourmands en temps de calcul. Pour remédier à cela, nous avons proposé une technique d'élagage de l'espace de recherche de type *diviser pour régner*, et nous avons allégé l'algorithme de résolution en faisant un effort sur la structure de données en proposant un codage dynamique supportant la FH guidée par l'interaction.

4 Combinaison de la FH avec la GBOR

Afin de motiver le choix de la combinaison de ces techniques d'optimisation, nous considérons la formalisation de chaque technique de façon isolée : la fragmentation horizontale, la gestion du buffer et

l'ordonnancement des requêtes établies dans le chapitre état de l'art (cf. chapitre 3).

A partir de ces formalisations, nous observons les similarités et les complémentarités entre les trois techniques. Les similarités résident dans le schéma de base de données utilisé, la charge de requêtes et la fonction objectif (minimisant le coût). Les complémentarités consistent dans le fait que chaque technique traite un niveau différent dans l'environnement de base de données (Données, Requêtes et Support).

Par exemple, la fragmentation horizontale ne considère ni le buffer ni l'ordre des requêtes. De plus, le problème de gestion du buffer fournit une stratégie d'allocation et de remplacement des données dans la mémoire cache qui est une entrée du problème d'ordonnancement. Ces particularités nous ont poussé à considérer le problème joint appelé problème de *Partitionnement-Bufferization-Ordonnancement* (PBO). La formalisation du problème combiné est la suivante :

Étant donnés un entrepôt de données D, une charge de requêtes Q, un buffer de taille B et un seuil W, le problème PBO consiste à partitionner le schéma de l'entrepôt D de façon à satisfaire la contrainte du seuil W tel que : le coût d'exécution de Q dans un planning donné et un scénario de gestion du buffer de taille B soit minimal.

4.1 Étude de complexité

En plus de la complexité spécifique à chacun des problèmes, la sélection multiple rajoute de nouvelles difficultés à savoir :

- La détermination de l'ordre de lancement des algorithmes d'optimisation pour chaque technique;
- La prise en compte de la nature des objets du buffer candidats après le partitionnement horizontal;
- La réécriture et l'ordonnancement des requêtes selon le contenu du buffer et le schéma de fragmentation.

4.1.1 L'ordre des algorithmes de résolution

La sélection multiple impose un ordre dans lequel les techniques d'optimisation doivent être appliquées. Dans [52], trois implémentations de la combinaison ont été distingués : indépendante, conjointe ou séquentielle (voir la section 4.1 dans le chapitre 2).

Nous avons montré dans l'état de l'art la grande dépendance entre le problème de gestion du buffer et celui d'ordonnancement des requêtes. Le changement d'une solution de gestion du buffer influence la solution d'ordonnancement, et vice versa. La dépendance entre la GB et l'OR est de type *Forte-Forte*. De ce fait, ces deux techniques sont bien adaptées pour une implémentation conjointe.

D'autre part, la fragmentation horizontale modifie le schéma de la base [161], ce qui a un impact sur les plans d'exécution des requêtes ainsi que sur les objets candidats au buffer. Ainsi, la fragmentation ne peut être effectuée après la gestion du buffer ou l'ordonnancement des requêtes. Les deux approches possibles sont : (1) effectuer la FH avant la GBOR, ou (2) déployer les trois techniques simultanément.

La dépendance entre la FH et la GBOR est de type *Forte-Faible*, ce qui favorise une implémentation séquentielle en commençant par la FH suivie de la GBOR.

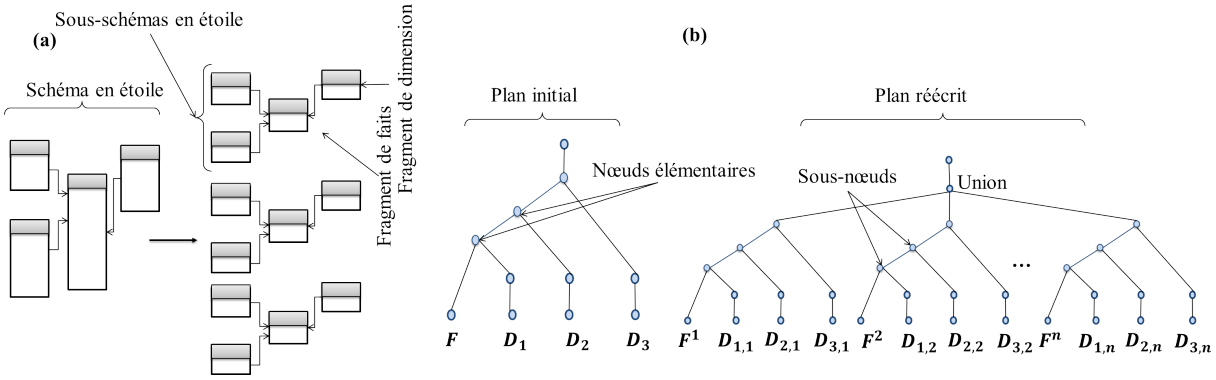

FIGURE 4.10 – Sous-nœuds obtenus par Fragmentation Horizontale d'un schéma en étoile

4.1.2 Les nouveaux objets du buffer

Théoriquement, les objets du buffer sont tout sous-ensemble d'instances (tuples) de la relation universelle *RU* obtenue par la jointure de toutes les tables de la base. Les travaux existant ont considéré le cas non partitionné. Dans ces travaux, les objets considérés sont les résultats ou sous-résultats obtenus par les opérations algébriques dans le plan de requêtes. Nous appelons ces objets : *nœuds élémentaires*.

Dans le cas partitionné, les objets du buffer ne sont pas des *nœuds élémentaires*, car les requêtes sont réécrites pour couvrir les différents sous-schémas en utilisant des jointures entre fragments suivies par l'union de résultats partiels. Dans ce cas, les *nœuds élémentaires* sont cassés en plusieurs opérations à l'intérieur des sous-schémas valides. Nous appelons ces nœuds : *les sous-nœuds*.

Ainsi, dans un schéma fragmenté horizontalement, les objets du buffer sont des sous-nœuds. Le nombre de sous-nœuds est plus grand que le nombre de nœuds élémentaires, mais leurs tailles sont plus petites à cause des sélections optimisées par la FH (cf. figure 4.10).

4.1.3 La réécriture des requêtes

L'un des problèmes de la sélection multiple avec la FH est la réécriture des requêtes sur le schéma fragmenté. Des difficultés supplémentaires apparaissent lors de la considération simultanée de la gestion du buffer et de l'ordonnancement des requêtes.

Quand une base de données est fragmentée horizontalement, son schéma en étoile se transforme en plusieurs sous-schémas en étoile comme le montre la figure 4.10. Le principe de réécriture des requêtes est détaillé dans la Section 2.3. Le plan de requête est projeté sur les sous-schémas valides où les jointures sont effectuées entre les fragments internes au sous-schéma, et l'union est effectuée entre les résultats de chaque sous-schéma (cf. figure 4.10) [52].

Pour effectuer les jointures internes, il est important d'identifier les fragments requis ainsi que l'ordre des jointures car ils affecteront les performances de la charge et les objets du cache.

L'ordre des jointures par sélectivité minimale est souvent utilisé pour réduire le coût des jointures. Ce choix est dû au fait que la sélectivité croissante (minimale) est simple à appliquer et prends un temps de calcul réduit comparée aux autres techniques [52]. Cependant, les objets obtenus en utilisant l'ordre de

jointure OJ_1 donnent une performance différente des objets obtenus par l'ordre OJ_2. Ces objets peuvent donner des plannings d'exécution différents à la charge et par conséquent des coûts variables.

4.2 Modèle de coût générique

Comme nous l'avons montré dans la Section 4.1.3, la fragmentation horizontale permet d'éclater le schéma en étoile de l'entrepôt en N sous-schémas en étoile, tel que N est le nombre de fragments de faits obtenu par le processus de fragmentation dérivée.

Pour estimer le coût d'exécution des requêtes sur le schéma partitionné, nous utilisons le modèle de coût décrit dans la Section 2.3.

A l'intérieur de chaque sous-schéma valide, les jointures entre fragments doivent être ordonnées par sélectivité minimale en utilisant les facteurs de sélectivité. Le coût d'exécution d'une requête sur le sous-schéma est obtenu par la somme des E/S requises pour effectuer toutes les opérations sur le sous-schéma ss_j. La formule d'estimation reste la même que dans le cas non-partitionné.

Après la fragmentation, le modèle de coût vérifie le contenu du buffer après chaque exécution de requête. Soit Op_k^{Qi} la sous-expression de la requête Q_i correspondant au sous-nœud no_k. Le coût d'exécution de Op_k^{Qi} est ignoré (=0) si no_k est caché, ou s'il existe un autre nœud no_j parmi les descendants de no_i dans le nouveau plan de la requête et que no_j est caché.

$$Cout(Op_k^{Q_i}) = \begin{cases} 0 & si\ Buffer(no_k) = 1 \vee (\exists no_j \in \mathcal{N} : no_i \subset_* no_j \wedge Buffer(no_j) = vrai) \\ C(no_k) & \text{sinon} \end{cases} \quad (1)$$

Au final, ce modèle permet de donner le total d'entrées/sorties requises par les requêtes réécrites sur le schéma fragmenté, tout en considérant le contenu du cache à chaque exécution.

5 Démarche de résolution du problème combiné

Dans cette section, nous détaillons la démarche proposée pour la résolution simultanée des problèmes de fragmentation horizontale, de gestion du buffer et d'ordonnancement des requêtes. Nous commençons par présenter la méthodologie de résolution globale. Nous décrivons ensuite le processus de préparation des entrées, suivi par les algorithmes employés pour chaque étape de notre démarche, à savoir la FH et la GBOR.

5.1 Méthodologie de résolution

A partir de l'analyse effectuée dans la section précédente, nous proposons de traiter les trois problèmes en considérant les deux étapes qui suivent dans l'ordre :

1. Effectuer la Fragmentation Horizontale (FH)

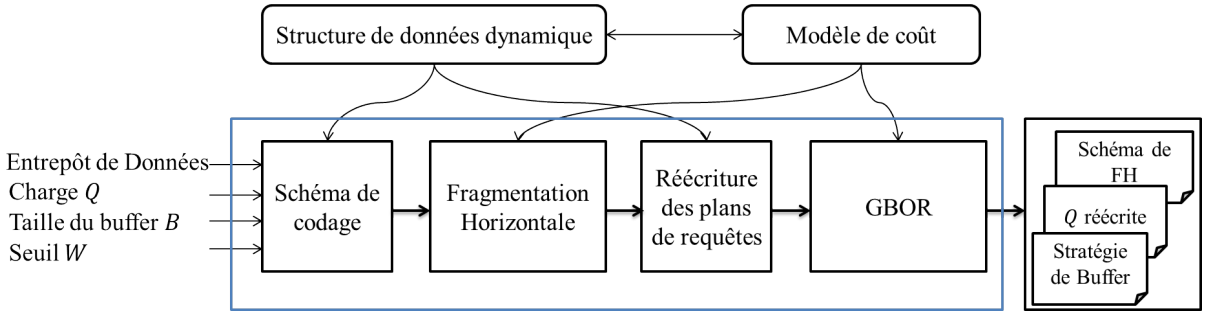

FIGURE 4.11 – Méthodologie de résolution de la combinaison de la FH et la GBOR

2. Effectuer simultanément une gestion du buffer et un ordonnancement des requêtes (*GBOR*) sur le schéma partitionné.

La motivation est que (1) la fragmentation horizontale modifie le schéma, (2) la gestion du buffer et l'ordonnancement des requêtes sont fortement corrélés et (3) la considération des trois problèmes simultanément est plus complexe (voir Section 4.1.1).

Les problèmes isolés sont des problèmes difficiles, et les techniques existantes sont soit de faible complexité et une efficacité limitée, ou efficace avec une grande complexité algorithmique (méta-heuristiques). Pour avoir un compromis entre l'efficacité et la complexité algorithmique, nous proposons de résoudre chaque problème de façon locale guidée par l'interaction des requêtes. Afin d'alléger les algorithmes, nous faisons également un effort sur la structure de données qui manipule les solutions potentielles. La structure de données dynamique proposée pour la FH est employée dans notre démarche afin d'alléger le processus de réécriture des requêtes.

La figure 4.11 résume notre méthodologie. Les entrées sont analysées et traitées afin de générer un codage pour le schéma en utilisant la structure de données dynamique. Ce schéma permet de représenter les fragments manipulés. L'algorithme de FH s'exécute sur ce codage afin de trouver un schéma de partitionnement minimisant le coût d'exécution. Une fois un schéma de fragmentation final est retourné, les plans de requêtes doivent être tracés sur les nouveaux sous-schémas. La structure de données dynamique est exploitée pour détecter les sous-schémas valides pour chaque requête et avoir les nouveaux plans. Ces nouveaux plans sont traités par l'algorithme de GB et d'OR pour produire un planning d'exécution des requêtes et une stratégie de mise en buffer correspondante. Le modèle de coût générique proposé dans la Section 4.2 est utilisé par les deux algorithmes pour mesurer la qualité des solutions. Les détails de chaque phase de la méthodologie sont donnés le long de ce chapitre.

5.2 Traitement des entrées et encodage

Dans cette section, nous détaillons les traitements effectués sur les entrées ainsi que la structure de données dynamique qui considère l'interaction des requêtes. A partir des entrées, cette structure permet de représenter les solutions et d'obtenir les nouveaux plans de requêtes. Le MVPP est obtenu par une approche d'optimisation indépendante. Pour cela, nous employons le MVPP obtenu par l'optimisation effectuée par l'algorithme de Boukorca *et al.* [55].

5.2.1 Représentation par MVPP

La charge de requêtes est une entrée principale du problème combiné. Afin de tirer profit de l'interaction entre les plans de requêtes, la charge est représentée par son MVPP. Cette représentation permet d'exploiter l'interaction pour la phase de fragmentation. Elle permet aussi de faire la réécriture des requêtes en projetant les plans initiaux de la charge sur le nouveau schéma partitionné. Ainsi, les sous-nœuds peuvent être identifiés pour construire l'ensemble des objets candidats au buffer.

5.2.2 Représentation des solutions

Pour établir les différentes étapes d'optimisation de notre sélection multiple, deux étapes de codage sont effectuées :

(1) Le codage employé pour les schémas de partitionnement est celui de FH que nous avons proposé et détaillé dans la Section 2.2.1. Le résultat de cette phase est un schéma d'encodage représentant l'ensemble des attributs de sélection et les sous-domaines qui leur sont associés.

(2) Pour représenter le planning d'exécution de la charge et son scénario de gestion du buffer, nous employons le codage proposé dans le chapitre 3.

5.2.3 Génération de profiles

Les requêtes sont réécrites par projection de leur plan sur le nouveau schéma. Cette phase de réécriture peut être effectuée en utilisant les profils obtenus à partir de la structure de données dynamique, qui offre trois fonctions :

1. identification des sous schémas valides;
2. ordre des jointures à l'intérieur des sous schémas;
3. estimation du coût d'exécution.

Pour obtenir les profils de requêtes, nous utilisons le schéma de codage et de remplir chaque cellule par 1 si le sous-domaine est requis par la requête, et 0 sinon. Les cellules sont aussi indexées par la sélectivité de chaque sous-domaine pour avoir le profil des facteurs de sélectivité. Pour représenter les sous-schémas, les valeurs des cellules sont égales à 1 dans les partitions pertinentes (requises), et 0 ailleurs.

La figure 4.12 présente un exemple de schéma de partitionnement, un profil de requêtes pour Q_2, un profil de facteurs de sélectivité de notre MVPP et un exemple de profil de sous-schéma.

Cette représentation permet au modèle de coût d'identifier les sous-schémas valides de chaque requête par la correspondance entre la requête et les profils de sous schémas. Si nous considérons que le meilleur ordre de jointure est celui obtenu par sélectivité minimale, cet ordre peut être fourni en utilisant le profil de facteurs de sélectivité pour estimer la taille des résultats intermédiaires. De plus, ces profils permettent l'estimation des coûts d'exécution de la charge en estimant la taille des résultats intermédiaires à l'aide des sélectivités.

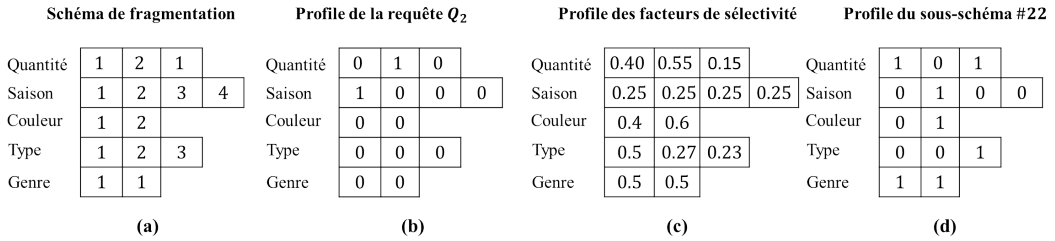

FIGURE 4.12 – Profils obtenus par la structure de données dynamique

5.3 Fragmentation horizontale basée sur les requêtes élues

Pour faire face à la complexité du problème de FH, nous proposons d'utiliser l'algorithme de REFH dirigé par l'interaction des requêtes, qui permet d'obtenir un schéma de Fragmentation Horizontale ayant une performance élevée en un temps de traitement réduit. L'algorithme sélectionne le schéma de FH en se laissant guider par l'interaction et par le modèle de coût. Le schéma fourni a une qualité comparable à celle obtenue en utilisant des méta-heuristiques. Lors du passage à l'échelle, l'algorithme s'exécute en un temps beaucoup plus faible que les autres algorithmes (cf. Section 6.2).

5.4 Réécriture des requêtes et identification des objets du buffer

Afin de pouvoir appliquer un algorithme de GBOR sur le schéma de FH obtenu, les requêtes doivent être réécrites pour chaque nouveau sous-schéma, afin de construire le nouveau MVPP. La projection des plans de requêtes est effectuée selon les étapes suivantes : (1) identification des sous-schémas valides et leur type de correspondance, (2) extraction de l'ensemble des prédicats et des fragments de dimension à joindre, (3) réécriture de la sous-requête correspondante à l'exécution de la requête Q_i sur le sous-schéma valide ss_j, et enfin (4) l'union (UNION ALL) des résultats de toutes les sous-requêtes pour obtenir le résultat final.

La réécriture utilise les profils de requêtes pour déterminer les sous-schémas valides sur lesquels les requêtes s'exécutent.

5.5 Gestion du Buffer et Ordonnancement des Requêtes

Pour résoudre le problème de GBOR, nous utilisons l'algorithme Queen-Bee qui a une faible complexité algorithmique et une grande efficacité, ce qui permet d'alléger le traitement du problème combiné. Cet algorithme est appliqué au le schéma de fragmentation résultant de l'algorithme de REFH. Les entrées du problème de GBOR sont projetées sur le schéma de FH obtenu. Ainsi, le schéma de l'entrepôt est constitué de l'ensemble des nouveaux sous-schémas en étoile, et sa charge est représentée par un nouveau MVPP obtenu par projection des plans initiaux sur les sous-schémas en étoile. Les objets candidats au buffer considérés sont les nouveaux sous-nœuds dans le MVPP obtenu par réécriture sur le schéma de FH retourné.

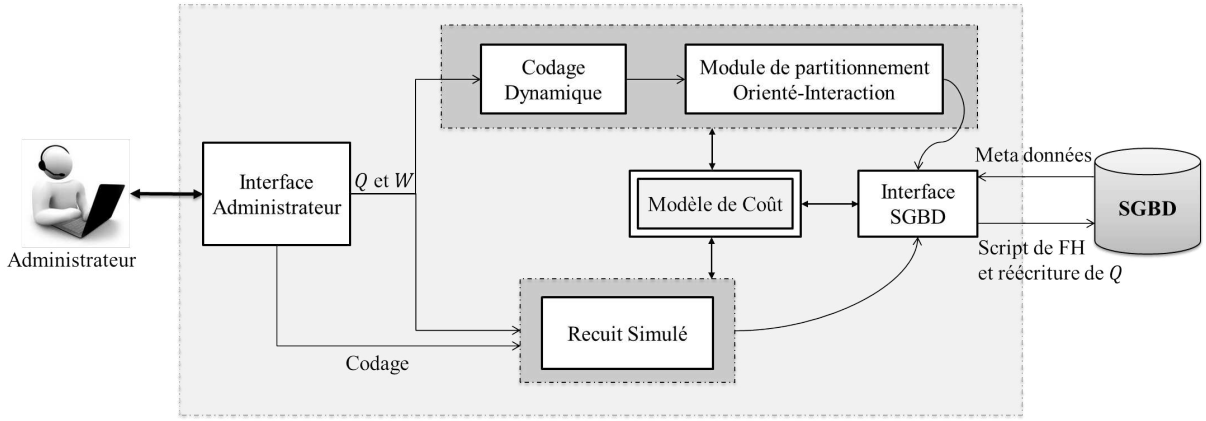

FIGURE 4.13 – Résolution du problème de la Fragmentation Horizontale

6 Évaluation des performances des algorithmes

Dans cette section, nous établissons une étude expérimentale en deux parties : la FH seule, et la FH combinée à la GBOR. Pour chaque partie, nous effectuons une série de simulations et de validation. Nous commençons par décrire le jeu de données employé pour les différents tests. Nous discutons ensuite les résultats d'expériences obtenus pour la FH, qui sont ensuite validés sous Oracle11g. Enfin, nous présentons les performances obtenues en combinant la FH avec la GBOR.

6.1 Jeu de données

Dans l'étude expérimentale, nous travaillons sur le benchmark SSB [157] avec un facteur d'échelle de 100, afin d'obtenir 100 GB de données. La base est stockée sous Oracle11g sur un serveur de 32GB de RAM et un processeur Intel® Xeon® CPU de 2x2.40GHz. La charge principale contient 22 requêtes de jointures en étoile SSB. Une deuxième charge est aussi utilisée, qui contient 12 requêtes de jointure en étoile SSB sans interaction entre-elles. Cela est destiné à étudier l'impact de l'interaction des requêtes sur le déroulement de la fragmentation.

6.2 Évaluation de la fragmentation horizontale

Dans cette section, nous conduisons une étude expérimentale intensive validant notre proposition. Nous commençons par décrire le processus de simulation. Les résultats obtenus sont discutés et justifiés pour révéler les différentes particularités de notre approche.

6.2.1 Simulations

Pour effectuer les différents tests, nous avons doté notre simulateur (cf. chapitre 3) d'un module supplémentaire de partitionnement. Ses entrées sont l'entrepôt et sa charge de requêtes. Nous faisons varier le seuil W afin de montrer son impact sur la performance. Notre modèle de coût fournit le nombre d'Entrées/Sorties requis par chaque requête pour mesurer la qualité des schémas de FH. Les algorithmes

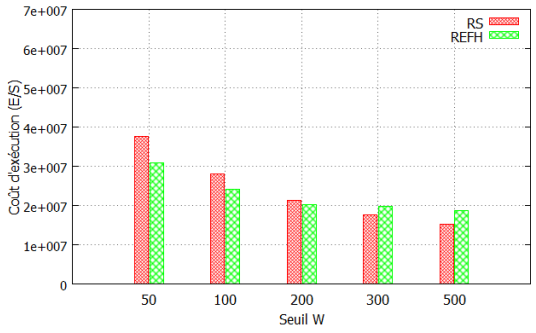

FIGURE 4.14 – Comparaison de performance des algorithmes en faisant varier le Seuil

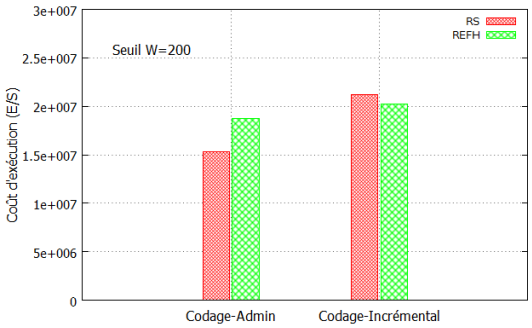

FIGURE 4.15 – Sensibilité des deux algorithmes au codage

utilisés sont : REFH et le Recuit Simulé (RS). Le choix de cet algorithme repose sur le fait qu'il est largement utilisé dans ce genre de problèmes d'optimisation [52]. De plus, il est moins gourmand qu'un algorithme génétique et plus performant qu'un Hill Climbing [29]. Pour paramétrer notre RS, nous avons fixé le nombre d'itération à 500 et la température à 300. Comme l'algorithme est non déterministe, il est exécuté plusieurs fois (entre 5 et 10 selon la variance) de façon à déterminer sa performance moyenne. Contrairement à notre algorithme qui construit son propre codage, le RS nécessite un codage préétabli comme entrée additionnelle (cf. figure 4.13).

6.2.1.1 Coût d'exécution pour des seuils W variables Dans la première expérience, nous comparons les schémas de FH sélectionnés par le RS à ceux sélectionnés par le REFH. Dans la figure 4.14, le coût d'exécution de la charge est donné en terme de nombre d'E/S. Les résultats montrent que l'algorithme REFH est plus efficace que le RS sauf pour les valeurs de seuil très grandes ($W \geq 300$). Les différences observées sont toutefois très proches. L'efficacité de REFH revient aux opérations Split/Merge qui s'effectuent à la demande, i.e. l'algorithme vise à satisfaire le plus grand nombre de requêtes parmi les plus bénéfiques (élues) qui contribuent considérablement à l'amélioration de performance de la charge.

Quand $W < 300$, le RS effectue un nombre d'itérations assez grand en cherchant aléatoirement la meilleure solution, alors que le REFH détecte et effectue systématiquement les mouvements nécessaires (Split/Merge) pour chercher le meilleur schéma de fragmentation.

Si W est très grand (≥ 300), le REFH s'arrête lorsque toutes les requêtes élues et leurs prédicats sont satisfaits. Cependant, plusieurs opérations de Split/Merge restent encore possibles pour réduire le coût de la charge. Ces opérations additionnelles ne peuvent être effectuées par le REFH, mais le RS peut les atteindre par un nombre élevé de transformations (mouvements aléatoires).

Nous rappelons que W est le nombre de fragments de faits, qui représente le nombre de sous-schémas générés. Le nombre total de fragments dans le schéma est plus grand que W.

Nous concluons de cette expérience que le REFH peut atteindre un niveau élevé d'efficacité comparé aux heuristiques. Cependant, pour les valeurs de Seuil très grandes W, le RS est plus efficace.

6.2.1.2 Sensibilité au codage : Un autre avantage majeur de notre approche est que l'algorithme construit son propre codage. Cela signifie que la phase de codage est une partie entière de l'algorithme

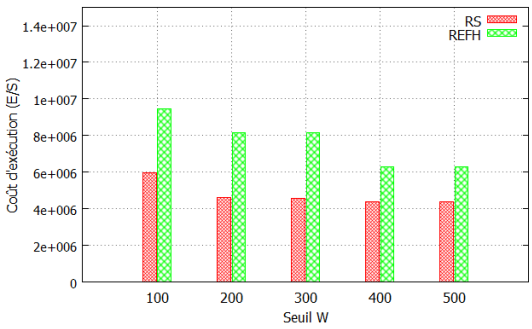

FIGURE 4.16 – Rendement du RS et REFH pour une charge sans jointures communes

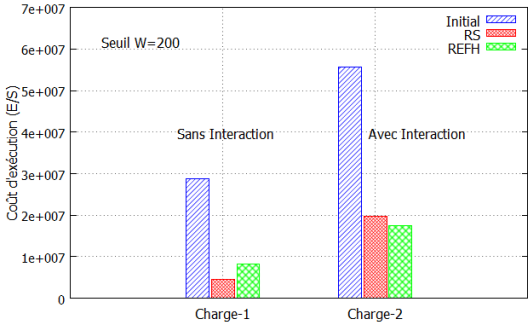

FIGURE 4.17 – L'impact de l'interaction entre les requêtes sur la performance

et fournit un schéma de codage dépendant de la charge, contrairement au RS qui nécessite un codage préétabli pour son déroulement. Le schéma de codage attribué au RS est généralement fourni par l'administrateur, et peut contenir les répartitions de sous-domaines meilleures pour les attributs de sélection.

Afin d'étudier la sensibilité des algorithmes de fragmentation au codage, nous avons effectué deux expériences : (1) La première utilise le processus de codage incrémental de REFH. Le codage obtenu est attribué au RS, et (2) la seconde en utilisant le codage de l'administrateur dans les deux algorithmes. La figure 4.15 montre que l'utilisation du codage de l'administrateur rend le RS plus efficace, car il fournit une décomposition des sous-domaines plus appropriée s'il connaît sa charge et ses données. Cependant, ce codage pénalise le REFH car la décomposition peut ne pas convenir aux besoins de certaines requêtes lors de l'élection des requêtes et de l'amélioration du schéma.

A partir de cette expérience, nous concluons que les algorithmes de résolution de la fragmentation horizontale sont sensibles au codage. Le RS ne peut construire son propre codage et fait appel à l'administrateur pour le lui fournir. Cependant, dans les faits, l'administrateur peut ne pas connaître la meilleure décomposition des sous-domaines, ce qui rend le REFH plus efficace de par son autonomie.

6.2.1.3 Impact de l'interaction des requêtes Afin d'étudier l'impact de l'interaction dans notre approche, nous utilisons une nouvelle charge de requêtes n'ayant aucune jointure commune entre-elles. La figure 4.16 présente le coût d'exécution de la nouvelle charge traitée par les deux algorithmes. Les résultats montrent que le REFH n'est pas suffisamment efficace comparé au RS, dans le cas où il n'y a pas d'interaction entre les requêtes. Ceci est dû au fait que seul un sous-ensemble de requêtes est optimisé, mais le gain n'a pu se propager parmi les autres.

Nous comparons maintenant les performances de REFH avec et sans interaction. La figure 4.17 montre que le REFH est meilleur sur une charge corrélée (en interaction), mais son efficacité est limitée sans interaction entre les requêtes.

A partir de ces expérimentations, nous concluons que notre approche est plus efficace dans le cas où la charge est corrélée, car le processus d'optimisation est guidé principalement par l'interaction.

6.2.1.4 Réactivité Nous étudions maintenant la réactivité des algorithmes. La figure 4.18 présente le temps de réponse des algorithmes. En faisant varier le Seuil W, le temps de réponse du RS augmente

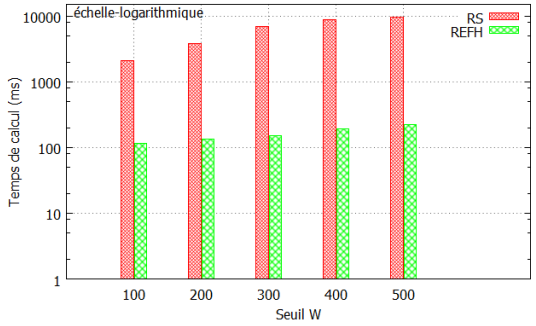

FIGURE 4.18 – Comparaison de la réactivité du RS et de l'algorithme de REFH

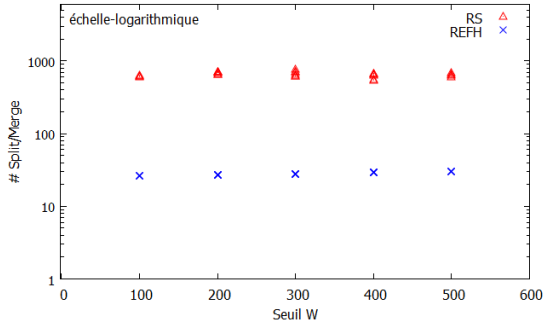

FIGURE 4.19 – Mouvements Split et Merge effectués par chaque algorithme

considérablement pour des valeurs élevées de W, contrairement au REFH qui reste relativement faible.

Ce temps de réponse élevé est courant dans cette famille d'algorithmes, contrairement à celui de notre algorithme qui est très rapide. Ceci est dû aux phases d'élagage de l'espace de recherche et au processus de fragmentation guidé par l'interaction. L'ensemble de requêtes élues permet de restreindre l'ensemble des prédicats à considérer, et de déterminer le poids de chaque prédicat dans la fragmentation. Le tri effectué par l'algorithme REFH permet donc de considérer les prédicats les plus bénéfiques avant l'épuisement de W.

Dans l'expérience qui suit, nous allons encore étudier la rapidité des algorithmes, mais en termes de mouvements et d'opérations de Split et/ou Merge. La figure 4.19 montre le nombre d'opérations Split/Merge effectuées par les algorithmes à différentes valeurs de W.

Comme le RS est non déterministe, il est exécuté à mainte reprises, et le nombre d'opérations (split/merge) varie pour chaque valeur de W. D'autre part, le REFH est déterministe et une solution unique est fournie pour une valeur W donnée, ainsi un nombre fixe de mouvements (split/merge) est effectué. Comme l'illustre la figure 4.19, le RS est très gourmand en terme de mouvements et d'opération d'éclatement et de fusion par rapport au REFH pour les mêmes raisons que dans l'expérience précédente; i.e. notre approche est orientée-interaction contrairement aux heuristiques qui explorent un espace très large de façon aléatoire.

À partir de cette dernière expérience, nous montrons que REFH reste plus rapide que le RS notamment quand le W est très grand ce qui rend le temps d'exécution de l'algorithme en croissance considérable. Ainsi, l'algorithme de REFH est plus adapté pour le passage à l'échelle.

6.2.1.5 La sélectivité des prédicats Quelques partitions pertinentes pour la REFH ne conviennent pas au processus de fragmentation car leurs facteurs de sélectivité sont soit trop élevés soit trop faibles. De telles sélectivités produisent des partitions non homogènes (trop larges ou trop petites) ce qui pénalise certaines requêtes. Pour éviter cette situation, l'intervalle de sélectivité peut être ajouté aux algorithmes de FH afin de les étendre avec un nouveau critère d'élagage.

Nous étudions les facteurs de sélectivité pour notre charge en faisant varier leurs valeurs minimales et maximales. Les résultats sont présentés dans la figure 4.20. En effet, les valeurs élevées (≈ 1) ou faibles (≈ 0) pénalisent la charge. La meilleure performance est obtenue dans l'intervalle $[0.15; 0.30]$ qui repré-

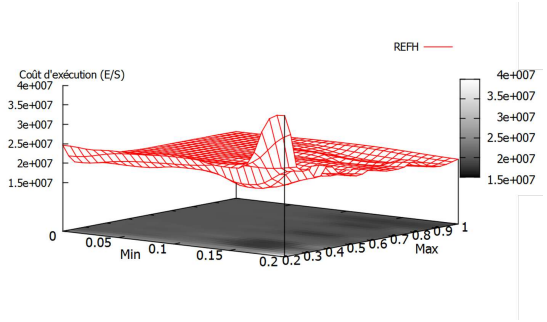

FIGURE 4.20 – Les intervalles de Facteurs de Sélectivité pour le fragmentation

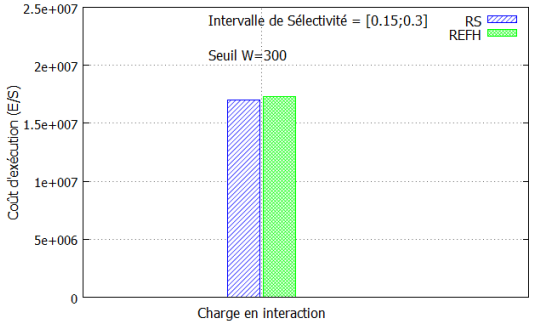

FIGURE 4.21 – Amélioration du REFH en définissant les intervalles de sélectivité

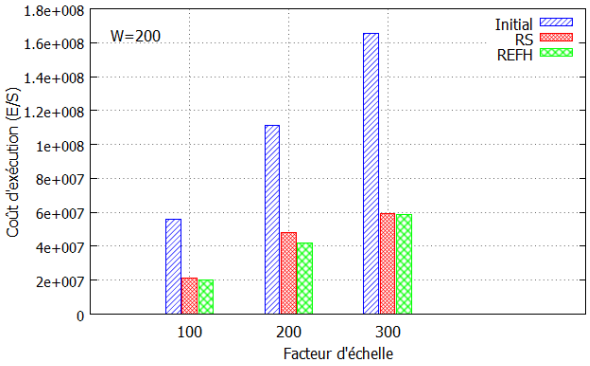

FIGURE 4.22 – Efficacité de la FH lors du passage à l'échelle

sente les facteurs de sélectivité des prédicats les plus adaptés pour la fragmentation. Nous comparons le REFH amélioré avec le RS dans l'expérience illustrée dans la figure 4.21, qui montre que le rendement du REFH est amélioré par ce nouveau critère (i.e. intervalle de sélectivité) même avec des valeurs plus grandes de Seuil ($W = 300$).

6.2.1.6 Passage à l'échelle : Pour montrer l'efficacité de la FH dirigée par l'interaction lors du passage à l'échelle, nous avons conduit une autre expérience en augmentant la taille de l'entrepôt SSB. La variation du facteur SF du benchmark SSB produit des bases de données de tailles comprises entre 100 et 300 Giga octets. L'application de la fragmentation horizontale sur ces bases de données suggère que l'approche dirigée par l'interaction reste efficace même dans les bases de données volumineuses (cf. figure 4.22).

6.2.2 Validation sous Oracle11g

Dans le but de valider nos résultats de simulation, nous déployons les solutions obtenues sur un SGBD Oracle11g. Le même jeu de données est utilisé : schéma SSB (100G) avec 22 requêtes sur notre serveur (décrit dans le chapitre 3).

Le déploiement des solutions obtenues par les algorithmes s'effectue en respectant le principe de

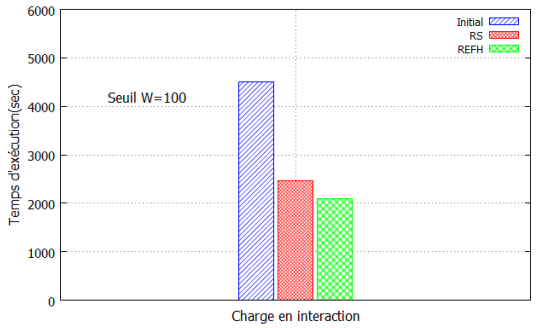

FIGURE 4.23 – Validation des résultats de simulation sous Oracle11g

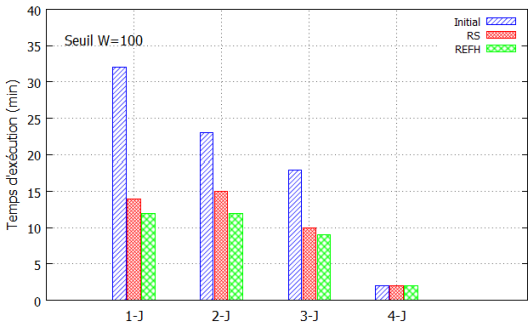

FIGURE 4.24 – Gain en performance par classes de requêtes sous Oracle11g

déploiement détaillé dans [52] :

- Création du schéma partitionné: Dans notre étude, la base de données employée est celle du Benchmark SSB. La FH Primaire des tables de dimension est effectuée en rajoutant une colonne supplémentaire Col_i à chaque table de dimension D_i pour éviter le problème de fragmentation de la table des faits avec plusieurs tables de dimension connu dans les versions antérieurs de certains SGBD comme Oracle. Ainsi le mode de fragmentation employé est le mode *LIST*. Suivant le schéma de fragmentation sélectionné par l'algorithme de résolution, un script SQL est lancé afin de remplir automatiquement les valeurs de la colonne Col_i insérée par le numéro du fragment qui doit contenir l'instance correspondante. Après le remplissage, la table D_i est fragmentée par *LIST* sur la colonne Col_i. En utilisant un script SQL, la fragmentation de la table des faits est effectuée en insérant une colonne Col_F à celle-ci. Le remplissage de cette colonne s'effectue par concaténation des valeurs des colonnes Col_i de chaque dimension dans les instances correspondantes. La fragmentation de la table des faits est finalement employée par *LIST* sur la colonne Col_F.
- Réécriture des requêtes sur le schéma partitionné : La fragmentation horizontale et l'insertion de nouvelles colonnes dans les tables empêchent d'exécuter les requêtes sur les fragments valides car les prédicats des requêtes ne sont pas définis sur ces nouvelles colonnes. Pour que les requêtes tirent profit de la fragmentation, chacune doit être réécrite sur le nouveau schéma suivant les étapes décrites dans la Section 5.4.
- Exécution des requêtes sur le schéma partitionné : Le script de requêtes réécrites est exécuté sur le schéma fragmenté afin d'obtenir les réponses aux requêtes avec un coût d'exécution réduit.

Les résultats présentés dans la figure 4.23 confirment les résultats obtenus par simulation. Les résultats suggèrent que l'algorithme REFH est plus efficace que le RS avec un seuil W=100 dans un système réel. Dans une autre expérience, nous regroupons les requêtes par nombre de jointures. La figure 4.24 montre le temps d'exécution requis par classe (n-J signifie que la requête contient n jointures). Nous constatons que notre algorithme apporte moins de gain aux requêtes ayant un grand nombre de jointures (e.g. 4-J), car il commence par satisfaire les requêtes les moins coûteuses, ce qui est généralement le cas des requêtes ayant peu de jointures. D'autre part, nous constatons aussi que notre approche permet de propager le plus de gains à travers la charge.

6.3 Évaluation de la sélection multiple

Pour montrer l'efficacité de notre approche, nous avons effectué une série de tests sur la performance et la réactivité de nos algorithmes isolés et de l'approche combinant les trois techniques. La sélection du schéma de FH, du planning d'exécution et la stratégie du buffer s'effectue à l'aide de notre simulateur et du modèle de coût intégré. Cependant, la validation sous un SGBD réel nécessite les phases suivantes :

- le déploiement du schéma de FH retourné par l'algorithme de REFH;
- la réécriture des requêtes sur le nouveau schéma de FH à l'aide des profils générés par la structure de données dynamique;
- le tuning du buffer avec les paramètres requis (e.g. réglage de la taille du buffer)
- la réécriture des requêtes obtenues lors de la phase précédente en considérant la stratégie du buffer retournée. Cette phase nécessite la considération des sous-nœuds dans les N sous-schémas en étoile obtenus.
- le lancement des requêtes réécrites en respectant le planning retourné.

Dans l'état actuel du travail, le déploiement de ces expériences a été conduit sur le simulateur. A cause de la lourdeur de la phase de validation, et par ce que l'automatisation de ce traitement est en cours de réalisation, nous avons seulement effectué des simulations. Toutefois, notre simulateur a déjà prouvé son efficacité dans l'évaluation et la sélection de solutions pour les deux problèmes : FH (Section 6.2) et GBOR (cf. chapitre 3). Les détails du script de déploiement sont fournies dans le chapitre 6.

6.3.1 Simulation

Nous comparons notre approche (PBO) avec plusieurs cas d'études : FH basée sur l'algorithme de REFH, FH basée sur l'algorithme du Recuit Simulé proposé en [52], GB et OR basés sur l'algorithme Queen-Bee et finalement, une combinaison des trois techniques en utilisant l'algorithme de RS. Le RS est choisi dans notre étude car il a été largement utilisé dans des problèmes d'optimisation combinatoires, et a prouvé son efficacité comparé au Génétique qui est plus lent et au Hill Climbing qui est moins efficace [52]. Pour paramétrer notre RS, nous fixons le nombre maximal d'itérations à 500 et la température initiale à 300.

La figure 4.25 montre que l'algorithme de requête élue donne une meilleure performance que le RS pour une charge ayant des requêtes en interaction. La raison est que cet algorithme est guidé par l'interaction des requêtes tandis que le RS explore aléatoirement l'espace de recherche. La Queen-Bee optimise la charge 1, contrairement à la charge 2 où aucun objet n'est caché (pas d'interaction). Les trois techniques (FH-GB-OR) sont combinées en utilisant le RS, qui donne une meilleure efficacité que la sélection isolée. L'approche PBO est plus efficace que le RS dans la charge 2. La raison est que le PBO est guidé par l'interaction des requêtes, qui le rend plus efficace qu'un algorithme de recherche aléatoire.

Nous observons aussi que la gestion du buffer après la FH permet de réduire le coût d'exécution de la charge même si les requêtes n'ont initialement (avant fragmentation) aucune interaction entre-elles, car le partitionnement permet de changer les plans de requêtes en créant de nouveaux objets candidats. Le nombre de nœuds de la charge 1 passe de 67 nœuds élémentaires à 1393 sous-nœuds, et la charge 2 passe de 77 à 2580. Comme les sous nœuds sont de plus petites tailles que les nœuds élémentaires, un nombre plus important de nœuds peut se loger dans le buffer, ce qui augmente le succès de cache (*cache hit ratio*).

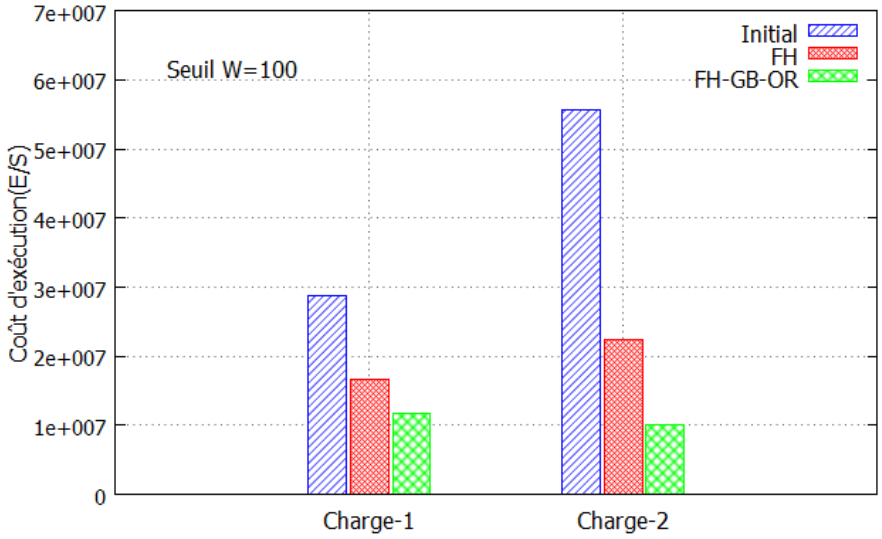

FIGURE 4.25 – Comparaison de performance des techniques isolées et combinées

6.3.2 Réactivité

Le temps de réponse de l'optimiseur est un critère déterminant dans le processus de sélection d'une solution optimale à un problème donné, car le rapport temps/efficacité constitue une mesure objective de la qualité d'un algorithme ou d'une approche. Pour évaluer la réactivité dans notre approche, nous avons effectué une expérience pour comparer le temps de déroulement de chaque algorithme dans le cas d'une sélection isolée, ainsi que dans la sélection multiple. La figure 4.26 présente la réactivité des algorithmes (échelle logarithmique). L'algorithme de requête élue et l'algorithme Queen-Bee sont beaucoup plus rapides que le RS, ce qui rend leur combinaison dans PBO plus intéressante que les heuristiques.

Dans ces expérimentations, nous montrons que la sélection multiple donne un compromis entre la complexité algorithmique et l'efficacité en exploitant l'interaction des requêtes.

7 Conclusion

Dans ce chapitre, nous discutons la nécessité de développer des solutions de conception physique, en considérant l'interaction entre les différentes composantes d'un système de base de données. Pour mener cette étude, nous avons commencé par revoir le problème de fragmentation horizontale en proposant une nouvelle approche inspirée de notre travail sur la gestion du buffer et l'ordonnancement des requêtes. Cette approche est dirigée par l'interaction, ce qui n'a jamais été considérée auparavant dans l'élaboration d'une solution au problème de partitionnement. Nous avons également proposé une structure de données dynamique permettant d'alléger l'algorithme et de supporter le processus de réécriture et d'estimation de coût. Nous avons ensuite proposé une approche employant trois techniques : la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes. La motivation derrière ce choix est que chaque technique considère un ensemble de composantes différent que l'autre. Nous avons également étudié les caractéristiques et les difficultés liées à la combinaison. Pour remédier aux

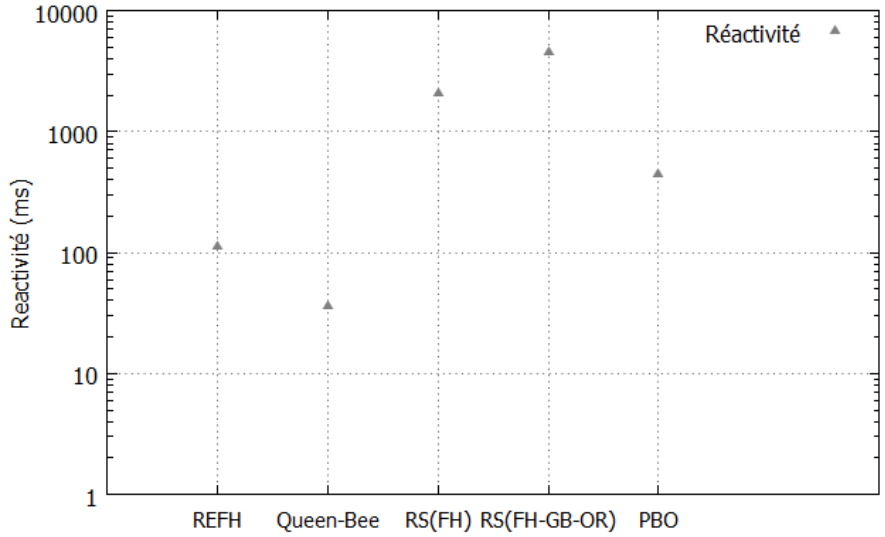

FIGURE 4.26 – Comparaison de la réactivité de chaque technique

problèmes de cette combinaison, principalement la complexité et l'adaptation des entrées, nous avons proposé respectivement l'utilisation d'algorithmes de type "diviser pour régner" et d'une structure de données dynamique.

Les résultats sont encourageants. Ils suggèrent assurément de considérer l'interaction à tous les niveaux dans un système de base de données pour améliorer ses performances.

Chapitre 5

L'aspect dynamique de la conception physique

Sommaire

1 Introduction

L'environnement des bases de données est soumis à des accès concurrents de plusieurs utilisateurs, ainsi que de constants changements au niveau des requêtes et des données, notamment avec l'arrivée du *Big Data*. Dans notre synthèse établie dans l'état de l'art, la GBOR par exemple a été traitée en considérant le cas hors-ligne dans la majorité des travaux, où la charge est statique et connue dès le départ par l'administrateur. Cependant, le contexte d'utilisation d'une BD n'est pas fermé : non seulement les données évoluent durant la vie du système, mais l'ensemble des requêtes peut aussi évoluer, au gré des utilisateurs.

De ce fait, la GBOR hors-ligne doit s'adapter aux changements des charges de requêtes et à la concurrence des utilisateurs.

La GBOR doit être auto-adaptative pour prendre en considération les changements qui se produisent sur les données et les requêtes. Dans l'état de l'art, nous avons distingué deux types d'optimisation par GBOR : statique et auto-adaptative. Dans le dernier type, nous distinguons deux cas de GBOR auto-adaptative : dynamique et en-ligne (cf. figure 5.1).

1. La GBOR *en-ligne* : le processus d'optimisation prend en compte l'aspect concurrent lors de l'exécution de la charge. Dans l'optimisation en-ligne, on décide sur la base de la connaissance du passé et du présent, contrairement au cas hors-ligne où l'on connaît aussi l'avenir.
2. La GBOR *dynamique* : le processus d'optimisation tient compte de l'évolution de la charge et des données. Par exemple, l'ajout de nouvelles requêtes impose l'adaptation de la solution courante. Ce type de GBOR peut être hors-ligne ou en-ligne.

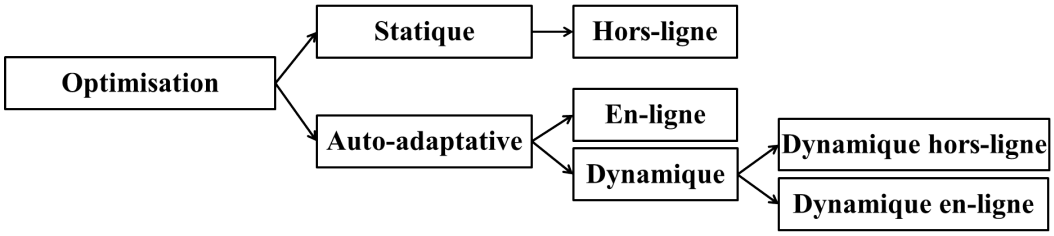

FIGURE 5.1 – Classification des modes d'optimisation

Le cas hors-ligne est traité dans le chapitre 3. Nous étudions ici le cas en-ligne. Nous présentons les avantages de la GBOR dynamique et ses particularités. Nous montrons aussi que la GBOR dynamique n'est qu'un cas particulier du traitement en-ligne. L'étude détaillée du problème en-ligne est menée, suivie d'une proposition d'un algorithme déterministe. L'étude expérimentale inclut des résultats de simulations théoriques ainsi qu'une validation sous Oracle11g. Pour conclure le chapitre, nous proposons un résumé et un bilan sur l'étude de la GBOR auto-adaptative.

2 L'optimisation en-ligne dans les entrepôts de données relationnels

Dans cette section, nous présentons le problème d'optimisation par GBOR en-ligne dans les entrepôts de données relationnels. Nous commençons par l'étude des particularités et des besoins des utilisateurs.

Nous présentons également les différents scenarii possibles pour le traitement d'une charge de requêtes en-ligne. Nous cherchons par la suite les ressemblances entre les deux modes en terme de démarche de résolution.

2.1 Particularités de l'environnement et des besoins des utilisateurs

Les EDR servent principalement à historiser les données [116]. Leur mise à jour n'est pas aussi fréquente que celle des bases de données traditionnelles. Par exemple, les données d'un entrepôt de ventes peuvent être rafraîchies une fois par mois voire une fois par trimestre. L'évolution des données peut engendrer l'évolution des requêtes. Ainsi, une solution $(\mathcal{P}_0, BM_0)$ pour l'environnement initial peut s'avérer moins efficace après l'évolution du système et nécessite le passage vers une nouvelle solution $(\mathcal{P}_1, BM_1)$ plus efficace. Ce passage permet une adaptation de l'ordonnancement des requêtes et de la stratégie de mise en buffer en fonction des nouvelles données et des nouvelles requêtes.

Dans le cas en-ligne, les techniques d'optimisations sont confrontées à un problème : le temps nécessaire pour optimiser la charge. En effet, le processus d'optimisation lui-même peut prendre un temps de traitement important, ce qui freine la performance du système.

Dans le cas hors-ligne (statique), la charge est optimisée une seule fois au début sans qu'il y ait besoin de faire des adaptations durant la vie du système. Dans le cas en-ligne, la réactivité de l'algorithme d'optimisation est cruciale car elle permet de réduire considérablement les délais d'attentes des requêtes.

Ainsi, la GBOR en-ligne est confrontée au dilemme de réactivité-efficacité pour que le processus d'optimisation ne s'effectue pas au détriment de la réactivité.

2.2 La gestion du buffer et l'ordonnancement des requêtes en-ligne

Avec l'évolution des systèmes de bases de données, l'information est consultée par un grand nombre d'utilisateurs qui interrogent simultanément la base. Les techniques d'optimisation existantes sont majoritairement adaptées au cas hors-ligne. Ces techniques ignorent l'aspect concurrent. Cependant, dans la vie pratique, on est souvent confronté à l'exécution en-ligne des requêtes.

L'arrivée d'un premier ensemble de requêtes E_0 à l'instant t_0 permet de déclencher un processus de GBOR statique répondant aux besoins de la charge en cours. Durant l'exécution des requêtes de E_0, d'autres peuvent arriver à l'instant t_1. Plusieurs cas peuvent être distingués :

- Cas 1 : une requête arrive dans le système alors que la file d'attente est vide, avant toute exécution. Dans ce cas, la requête est exécutée et ses objets candidats au buffer sont cachés dans la limite de l'espace disponible. Aucun calcul lié à la GBOR n'est requis.
- Cas 2 : N requêtes arrivent alors que la file d'attente est vide ($N > 1$), avant toute exécution. Un ordonnancement est requis pour planifier l'exécution de l'ensemble des requêtes selon leur interaction et selon les objets mis dans le buffer.
- Cas 3 : une ou plusieurs requêtes arrivent alors que des traitements sont en cours (dans la file ou en cours d'exécution). Les requêtes qui viennent d'arriver rejoignent celles en attente (dans la file) pour être analysées comme une seule charge.
- Cas 4 : l'un des cas précédents survient, alors que d'autres requêtes ont déjà été exécutées. Une

trace des optimisations déjà établies peut servir pour réduire le temps de traitement par la GBOR. Par exemple, les plans d'évaluation, l'affinité des requêtes et les objets candidats au buffer sont utiles pour optimiser, sans avoir à traiter la totalité des requêtes, notamment lors d'une réexécution d'une requête déjà analysée.

2.3 La gestion du buffer et l'ordonnancement des requêtes dynamiques

Durant la vie du système, les données et les requêtes peuvent évoluer. La séquence d'ordonnancement calculée sur la base de l'état initial de la charge peut n'être plus efficace après une ou plusieurs évolutions. C'est pourquoi l'adaptation de la solution initiale est requise pour maintenir une bonne performance du système. La GBOR dynamique peut être considérée hors-ligne ou en-ligne.

2.3.1 L'optimisation dynamique hors-ligne

Cette technique d'optimisation est nécessaire dans les cas suivants :

1. Les requêtes existantes changent de plan d'évaluation;
2. Les données changent;
3. De nouvelles requêtes arrivent.

Dans les deux premiers cas, les objets potentiels du buffer changent, ce qui impose la reconstruction du MVPP, point d'entrée de la GBOR. Dans le dernier cas, la solution de GBOR initiale peut être adaptée aux nouvelles requêtes. Leurs nouveaux objets du buffer pour alimenter l'ancien ensemble d'objets candidats.

2.3.2 L'optimisation dynamique en-ligne

A la différence de la GBOR hors-ligne, l'optimisation en-ligne nécessite une réactivité élevée (temps d'optimisation réduit) comme nous l'avons discuté en Section 2.1. Les scenarii considérés sont :

- Cas 1 : une requête arrive dans le système et la file d'attente est vide, avant toute exécution.
- Cas 2 : N requêtes arrivent et la file d'attente est vide ($N > 1$), avant toute exécution.
- Cas 3 : une ou plusieurs requêtes arrivent et d'autres sont en cours de traitement (dans la file ou en cours d'exécution), avant toute exécution.
- Cas 4 : l'un des cas précédents survient, alors que d'autres requêtes ont déjà été exécutées.

En matière de traitement, les trois premiers sont semblables au cas en-ligne. Dans le quatrième cas, l'adaptation des structures d'optimisation préalablement obtenues ne peut avoir lieu si les données ou les requêtes ont évolué. Ainsi, un nouveau MVPP doit être construit, et qui induira de nouveaux plans, un contenu de buffer, et par conséquent un nouvel ordonnancement.

2.4 Bilan

Dans la GBOR auto-adaptative, deux cas sont distingués : le *dynamique* et le *en-ligne*. La description des deux problèmes montre que la réactivité est cruciale dans ce contexte. De plus, la GBOR dynamique

peut être considérée hors-ligne ou en-ligne. En analysant sa particularité, nous avons conclu que le traitement de la GBOR dynamique peut être considéré comme un cas particulier de GBOR en-ligne.

Dans la suite de ce chapitre, nous étudions le problème de la GBOR en-ligne, et nous proposons une approche de résolution également applicable dans un contexte dynamique.

3 Problème de la gestion du buffer et d'ordonnancement des requêtes en-ligne dans les EDR

Comme nous l'avons vu dans le chapitre 3, les entrepôts de données relationnelles sont des environnement où l'interaction des requêtes est plus importante en raison de leur schéma en étoile. Le grand ensemble d'opérations communes entre les requêtes, favorise l'optimisation par gestion du buffer et ordonnancement des requêtes.

Dans cette section, nous formalisons le problème de la GBOR en-ligne dans les EDR et nous adaptons notre politique de gestion du buffer proposée dans le chapitre 3. Enfin, nous décrivons un modèle de coût spécifiquement conçu pour le cas de l'exécution en-ligne.

3.1 Formalisation

Soit une charge de N requêtes issues de plusieurs utilisateurs et stockées dans une file d'attente. Nous formalisons le problème joint relatif à la gestion de buffer et à l'ordonnancement, de la façon suivante :

Étant données les entrées suivantes :
- Un schéma d'entrepôt de données relationnel D;
- Un ensemble de requêtes $Q = \{Q_1, Q_2, ..., Q_n\}$ représentées par un MVPP;
- Une politique de gestion du buffer PGB;
- Une file d'attente.

Et soit la contrainte suivante :
- La taille totale du buffer disponible B.

Le problème de GBOR en-ligne consiste à trouver une solution $(\mathcal{P}, BM)$ qui réduit le coût en exploitant la taille disponible B du buffer, tel que $\mathcal{P}$ est le meilleur planning d'exécution de Q et BM est sa stratégie de gestion du buffer associée.

Nous rappelons que dans notre modèle explicite, nous représentons le problème en-ligne de la façon suivante (cf. page 49):

$$\textit{Problème de GBOR en-ligne} =< \{sousexpression, \emptyset, file\}, \{\emptyset, instances, \emptyset\}, \{Buffer, \emptyset\}, \{GB, OR\} >$$

3.2 Gestion du buffer sans connaissance préalable des requêtes

Contrairement à la GBOR hors-ligne dans laquelle la charge doit être connue, dans le contexte en-ligne, l'administrateur peut ne pas avoir de connaissances (données) sur les requêtes à venir. Dans ce cas, la gestion du buffer est contrainte par ces données et la décision de remplacement des nœuds ne peut être établie à partir des besoins des prochaines requêtes.

C'est pourquoi, nous proposons d'associer la politique de gestion de buffer *LRU* à notre ordonnanceur. Cette politique est l'une des plus utilisées dans les bases de données. Bien qu'elle ne tient pas compte des besoins et de l'interaction de la charge, *LRU* permet d'apporter un certain niveau de performance à la charge [101].

3.3 Gestion du buffer avec connaissance préalable des requêtes

L'administrateur peut avoir une idée sur les requêtes à venir à l'aide des probabilités, des estimations ou des données réelles. Ces données peuvent être injectées dans le gestionnaire de buffer pour lui permettre de décider lesquels des objets mettre ou enlever du buffer à chaque instant.

Notre gestionnaire de buffer requiert des fréquences des requêtes pour déterminer si une requête à venir utilise un nœud dans un temps donné (par heure, jour, mois, etc.). Le gestionnaire de buffer est semblable à celui utilisé dans la GBOR hors-ligne (voir Section 4.5 du chapitre 3).

3.4 Modèle de coût pour la GBOR en-ligne

L'évaluation des solutions de la GBOR en-ligne diffère du cas hors-ligne. En plus de l'estimation basique et la considération du buffer, l'évaluation nécessite une troisième composante pour la considération de l'exécution en-ligne.

Les deux premières composantes émargent déjà au modèle de coût proposé dans 4.4. Nous étendons ce modèle ici en y ajoutant la troisième composante, à savoir les paramètres d'exécution en-ligne.

3.4.1 Considération de l'exécution en-ligne

Quand des requêtes sont lancées simultanément, l'arrivée d'un nouvel ensemble de requêtes déclenche le calcul d'un nouveau planning d'exécution et d'une nouvelle stratégie de gestion du buffer. Ce processus peut prendre une durée variable selon les entrées et l'algorithme d'optimisation lui-même. A chaque arrivée, les nouvelles requêtes se joignent à celles en file d'attente. Le déclenchement du processus d'ordonnancement peut causer un retard d'exécution en augmentant le temps de réponse des requêtes. Ce retard dépend de la réactivité de l'algorithme. Le coût/temps d'ordonnancement des requêtes à l'arrivée est ajouté comme un coût supplémentaire qui augmentera le temps de réponse. Nous notons ce temps supplémentaire $Delai(Q')$, où $Q' \subseteq Q$. Un écouteur de requête permet de capturer l'ensemble des requêtes à l'arrivée de la file d'attente QA. L'écouteur déclenche l'optimiseur quand un nouvel ensemble de requêtes arrive. Le temps écoulé pour leur traitement est ajouté en terme de secondes au temps d'exécution de la charge ($Delai(QA) > 0$). Le coût équivalent à ce temps en termes d'E/S est noté *cout_suppl*. Il est ajouté au coût d'exécution total à chaque optimisation.

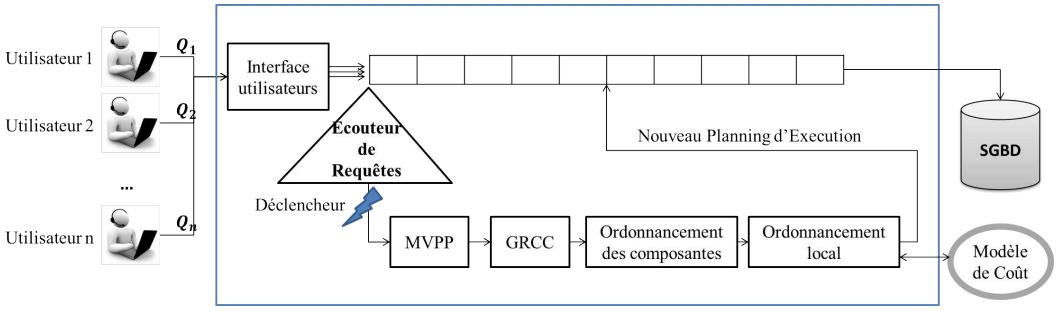

FIGURE 5.2 – Méthodologie de résolution par la Queen-Bee En-ligne

3.4.2 Coût total

Le coût total d'exécution de la charge est la somme des coûts de toutes les requêtes individuelles majoré du coût requis pour l'optimiseur :

$$Cout\_Total = \sum_{i=1}^{n} Cout(op_{l_i}^{Q_i}) + cout\_suppl(QA_i)$$

Le temps d'exécution correspondant est obtenu par la somme du temps d'exécution de chaque requête et du temps écoulé à chaque optimisation :

$$Temps\_Total = \sum_{i=1}^{n} Temps(Cout(op_{l_i}^{Q_i})) + Delai(AQ_i)$$

4 Algorithme d'optimisation en-ligne

Pour gérer l'équilibre efficacité-réactivité, nous traitons le problème de GBOR en-ligne en adaptant l'algorithme *Queen-Bee* proposé dans le cas hors-ligne. Le choix de cet algorithme est principalement motivé par :

1. l'efficacité dans la GBOR statique, comparable à celle des algorithmes lourds comme le Hill Climbing et l'Algorithme Génétique;
2. la réactivité due à une complexité algorithmique réduite;
3. l'adaptabilité de l'algorithme Queen-Bee au cas en-ligne.

En effet, l'algorithme *Queen-Bee* s'adapte au cas en-ligne en prenant en considération la file d'attente comme une entrée supplémentaire. La file d'attente permet d'ajouter des requêtes déjà traitées ou en attente pour qu'elles soient considérées dans le même ensemble que les requêtes nouvellement arrivées. Nous appelons notre nouvel algorithme : *Queen-Bee En-ligne* (QBE).

4.1 Principe

Afin de réduire la complexité algorithmique et les délais d'attentes provoqués par la GBOR en-ligne, nous utilisons une adaptation de l'algorithme *Queen-Bee*. L'idée est de créer le MVPP de façon

dynamique, en prenant à chaque instant les requêtes arrivant sur la file. Ces requêtes sont capturées par un écouteur qui déclenche le processus d'analyse. Les requêtes du MVPP sont regroupées par affinité pour construire les *ruches*. Pour chaque *ruche*, l'algorithme élit une requête (*queen-bee*) destinée à être la première à s'exécuter dans sa *ruche*. Les objets mis dans le cache par la *queen-bee* permettent de satisfaire les requêtes de la même *ruche*. La figure 5.2 illustre le principe de résolution par la Queen-Bee En-ligne.

D'après les entrées du problème de GBOR en-ligne (Section 4.1), l'algorithme nécessite de considérer la file d'attente en plus des entrées du cas statique. Un module supplémentaire est intégré par rapport à la Queen-Bee hors-ligne pour permettre de capturer les nouvelles requêtes. Ce module est l'écouteur de requêtes. Ainsi, les quatre modules qui définissent notre Queen-Bee En-ligne sont :

1. L'écouteur permettant de capturer les nouvelles requêtes à l'arrivée de la file d'attente;
2. Le générateur de GRCC en-ligne;
3. L'ordonnanceur de composantes connexes (OCC) qui est optionnel selon l'existence de priorité entre les requêtes ou si l'administrateur limite le temps d'attente de quelques-unes;
4. L'ordonnanceur local à l'intérieur de chaque composante (OL).

Nous associons une politique de gestion du buffer à notre algorithme selon les connaissances de l'administrateur :

– si l'administrateur n'a aucune connaissance préalable de la charge, alors nous utilisons la politique de gestion la plus souvent utilisée dans les systèmes de bases de données, qui est LRU (*Least Recently Used*).
– Sinon, et si l'administrateur connaît au moins la fréquence des requêtes, alors nous proposons d'utiliser la politique de gestion du buffer DBM proposée dans la GBOR hors-ligne.

4.2 Écouteur des requêtes

Ce module est à l'écoute des requêtes en provenance des différents utilisateurs. Il permet de capturer ces requêtes pour déclencher le processus de GBOR dès leur arrivée à la file. Le MVPP est tracé de façon incrémentale en fusionnant les plans des nouvelles requêtes. Ce MVPP (incluant les anciennes et nouvelles requêtes) est associé aux entrées (EDR, Q et B). L'ensemble Q contient les nouvelles requêtes QA, et celles dans la file d'attente.

Nous supprimons les doublons de requêtes dans la file. Si plusieurs utilisateurs lancent la même requête au même temps, celle-ci est traitée une seule fois car nous considérons uniquement les requêtes de sélection (pas de mise à jour).

4.3 Génération du graphe de requêtes en composantes connexes

A partir du MVPP, le graphe de requêtes en composantes connexes est construit en adaptant l'approche hors-ligne. Deux cas sont distingués :

1. Si le GRCC est construit pour la première fois (premier ensemble QA), on procède de la même manière que dans le cas hors-ligne (cf. section 5.4.2 du chapitre 3).

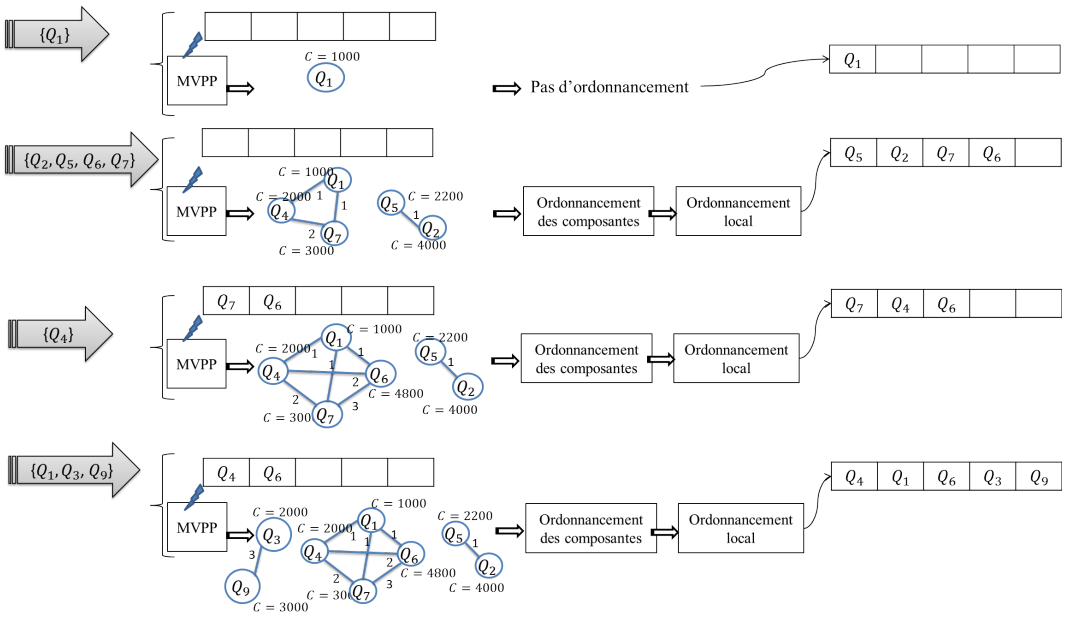

FIGURE 5.3 – Exemples d'exécution de Queen-Bee En-ligne

2. Si le GRCC existe déjà, il est mis à jour en ajoutant des nouveaux sommets aux ruches existantes. Afin d'identifier la ruche à laquelle appartient une nouvelle requête, on s'appuie sur sa première jointure. Une requête doit partager au moins la première jointure avec les requêtes de même ruche qu'elle. De nouvelles ruches peuvent éventuellement être créées si des requêtes n'appartiennent à aucune ruche existante.

4.4 Ordonnancement des composantes connexes

L'ordonnancement des composantes connexes permet de privilégier certaines ruches contenant les requêtes prioritaires. Il permet aussi d'écourter les délais d'attente moyens des requêtes. Dans notre étude, nous considérons que les requêtes ont la même priorité. Comme les objets du cache requis pour deux ruches différentes sont disjoints, alors :

$$\{(Q1, Q7, Q4, Q6); (Q3, Q9, Q10); (Q2, Q8, Q5)\} \text{et} \{(Q3, Q9, Q10); (Q1, Q7, Q4, Q6); (Q2, Q8, Q5)\}$$

ont le même coût d'exécution, cela étant déjà discuté dans le chapitre 3.

Toutefois, si l'administrateur choisit de limiter le délai d'attente pour certaines requêtes, une contrainte doit être ajouté pour exiger que ces requêtes soient planifiées pour exécution avant que ce délai ne soit dépassé. Si les requêtes possèdent des délais d'attente, le problème de GBOR devient plus difficile, car en plus de chercher l'ordonnancement et la mise en buffer optimaux, il faut respecter ces délais.

Nous ne considérons pas la contrainte des délais. Les algorithmes temps-réels sont plus appropriés pour ce genre de contraintes, mais leur intégration dans les noyaux de bases de données constitue un travail en lui-même. Ce travail sera un prolongement de cette thèse.

Algorithme 7: Algorithme de Queen-Bee En-ligne()

1: $file \leftarrow Ecouteur()$;
2: **if** ($GRCC$ is empty) **then**
3: $Q_{attente} = file$;
4: $GRCC(Q_{wait})$;{créer le GRCC pour les nouvelles requêtes}
5: $OCC(Q_{attente}, \emptyset)$;{définir les priorités entre les ruches}
6: $OL(Q_{attente}, \emptyset)$;{ordonnancement local à l'intérieur des ruches}
7: **else**
8: $Q_{lancees} = requetes\_precedantes$;
9: $Q_{attente} = file$;
10: $maj\_GRCC(Q_{attente}, Q_{lancees})$; {mettre à jour le GRCC avec les requêtes en attente}
11: $OCC(Q_{attente}, Q_{lancees})$; {mettre à jour les priorité entre les ruches}
12: $OL(Q_{attente}, Q_{lancees})$; {ordonnancement local à l'intérieur des ruches modifiées}
13: **end if**
14: $ajout\_retard()$;{ajout du temps écoulé pour l'ordonnancement}
15: $exec(Q_{attente})$;
16: $evaluer(Q_{attente})$;
17: $maj\_file()$;{mettre à jour la file d'attente}

4.5 Ordonnancement local des requêtes

Les requêtes appartenant à une même ruche doivent être ordonnées. Pour cette raison, un *Ordonnancement Local* (OL) est requis. Celui que nous utilisons ici fonctionne selon les étapes suivantes :

1. Identifier la requête *queen-bee* selon un critère Cr donné (e.g. coût minimal ou fan-out maximal. Voir chapitre 3);
2. Simuler l'exécution de la requête élue, qui charge dans le buffer les nœuds candidats selon la PGB adoptée;
3. Explorer les sommets restants dans la composante en cours, en sélectionnant la requête suivante selon le critère Cr. Si deux requêtes ont la même valeur pour ce critère (même coût par exemple), on privilégie celle qui utilise le plus grand nombre de résultats contenus dans le buffer.
4. Itérer le procédé jusqu'à épuisement des requêtes.
5. Le résultat est la suite des requêtes dans l'ordre sélectionné par l'ordonnanceur local.

Le résultat retourné de chaque ruche est un ordonnancement partiel de la charge globale.

4.6 Solution finale

La solution finale pour le problème de GBOR en-ligne consiste à concaténer tous les ordonnancements des ruches (ordres partiaux), et leur mise en buffer. Plus formellement, une solution $(\mathcal{P}, BM)$ est obtenue par les étapes suivantes :

1. Tri des composantes connexes par l'algorithme. Si ce tri n'est pas effectué, les composantes seront traitées dans un ordre quelconque. Soit OC_1,OC_2,... OC_k l'ordre retourné pour les k composantes connexes générées.
2. Ordonnancement local de chaque composante OC_i
3. Concaténation des résultats d'ordonnancement de toutes les composantes, dans l'ordre OC_1,OC_2,... OC_k. Soit $\mathcal{P}'$ l'ordonnancement obtenu.
4. Concaténation des stratégies de mise en buffer selon l'ordonnancement $\mathcal{P}'$, pour construire la matrice de gestion de buffer BM'.
5. Retourner l'ordonnancement final $\mathcal{P}'$, et la stratégie de gestion du buffer BM' de la charge globale.

Exemple. La figure 5.3 illustre quelques exemples de déroulement dans plusieurs situations selon les requêtes arrivant et le contenu de la file d'attente.

Dans le cas (a), Q_1 est ajoutée au GRCC initialement vide. Aucun ordonnancement n'est requis.

Dans le cas (b), quatre requêtes arrivent et aucune requête n'est présente dans la file. Le GRCC précédent est mis à jour et l'ensemble des nouvelles requêtes QA est ordonnancé.

Dans les cas (c) et (d), les nouvelles requêtes qui arrivent mettent à jour le GRCC et l'ordonnancement les traite avec celles déjà présentes dans la file pour ajuster l'ordonnancement précédemment établi.

5 Évaluation des performances des algorithmes

Notre expérimentation est effectuée en deux niveaux : une phase de simulation, et une phase de validation. La phase de simulation permet d'obtenir des résultats théoriques à partir de notre algorithme dans des contextes différents. Ces résultats sont validés, en les déployant sur un SGBD réel pour montrer la performance effective.

5.1 Jeu de données

La phase de simulation est effectuée en utilisant notre modèle de coût, un schéma en étoile SSB de 100Go et 30 requêtes s'exécutant de façon concurrente (en provenance de plusieurs utilisateurs). La taille du buffer utilisée est de 20Go. Après avoir obtenu le couple $(\mathcal{P}, BM)$ correspondant au planning d'exécution et sa stratégie de gestion du buffer, nous passons à la phase de validation. La validation des résultats théoriques est effectuée sur une base de données réelle avec le même jeu de données déployé sur notre serveur.

5.2 Résultats de simulation

Dans la figure 5.4, plusieurs requêtes sont lancées de façon concurrente dans quatre intervalles de temps. Notons T_i l'événement d'avoir de nouvelles requêtes à l'instant t_i. Ainsi, T_1, T_2, T_3 et T_4 représentent quatre nouveaux ensembles de requêtes qui arrivent durant la vie du système. L'exécution des requêtes qui arrivent est relancée dans quatre contextes distincts :

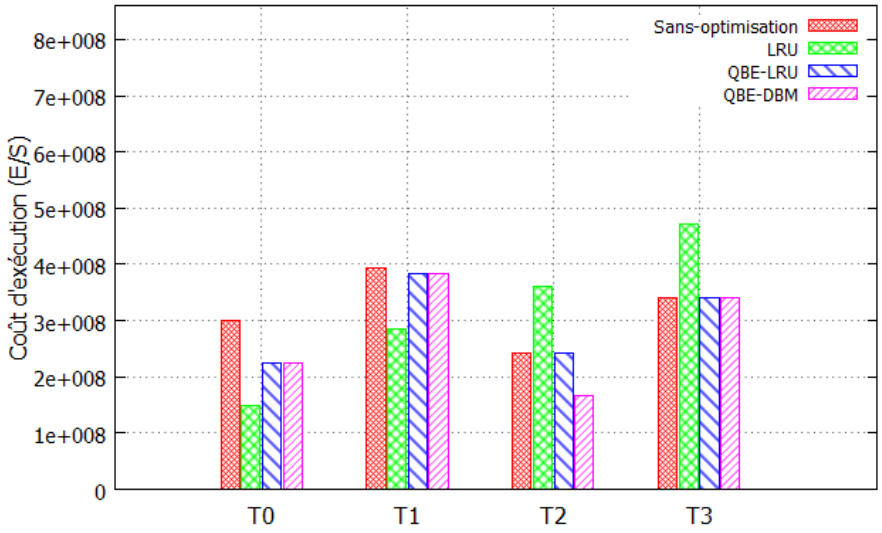

FIGURE 5.4 – Exécution des sous-ensembles concurrents de requêtes

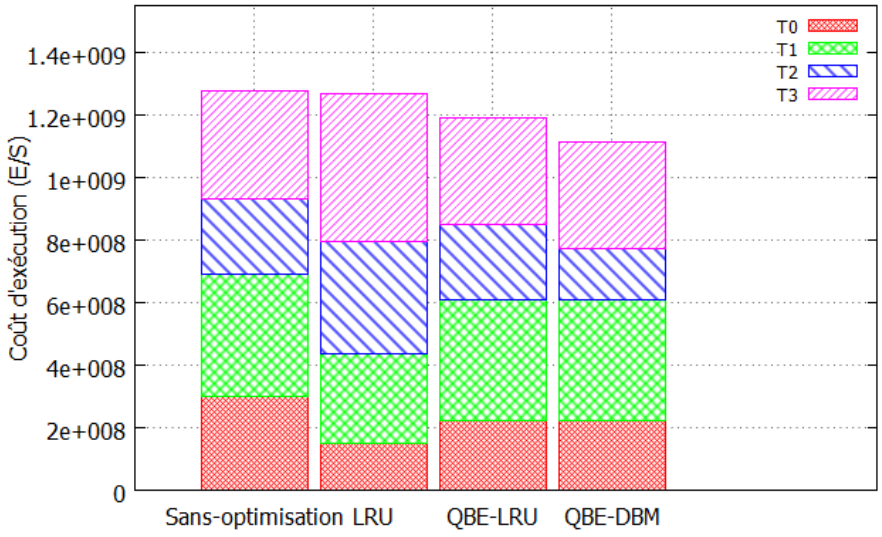

FIGURE 5.5 – Coût d'exécution total obtenu par chaque technique d'optimisation

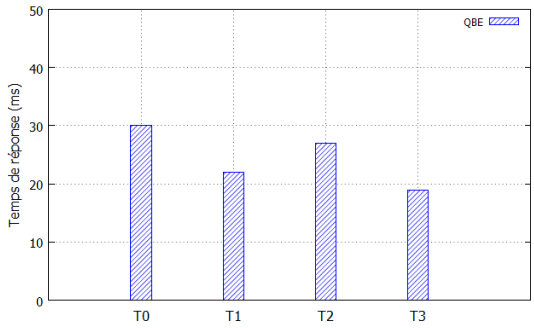

FIGURE 5.6 – Temps écoulé pour l'exécution de l'ordonnanceur queen-bee en-ligne

FIGURE 5.7 – Temps d'exécution total sous Oracle11g

- Aucune optimisation, où des requêtes dans la file d'attente sont traitées sans ordonnancement ni bufferisation.
- LRU pour la gestion du buffer sans ordonnancement.
- Queen-Bee En-ligne en utilisant LRU, où l'utilisateur ne connaît pas la charge.
- Queen-Bee En-ligne en utilisant la politique DBM, où l'utilisateur a une connaissance préalable des fréquences des requêtes.

Les résultats montrent que les requêtes après chaque arrivée T_i sont traitées d'une façon différente selon le contexte. La variation du planning d'exécution ou de la stratégie du buffer a un impact significatif sur la performance. Le coût d'exécution total (cf. figure 5.5) montre l'impact de la mise en buffer sur la performance. Il montre aussi l'interdépendance entre l'ordonnancement et la gestion du buffer comme nous l'avons déjà montré dans le cas hors-ligne (statique). Les résultats montrent également que le DBM est plus efficace que LRU, toutefois il nécessite une connaissance préalable de la charge. Nous notons aussi que le DBM donne une meilleure performance pour la Queen-Bee En-ligne comparé à LRU.

Sur la figure 5.6, nous observons le temps écoulé pour le traitement des nouvelles requêtes (en millisecondes). Le processus prend en compte le nouvel ensemble de requêtes, le contenu du buffer et les requêtes déjà en attente qui ne sont pas encore exécutées. Les résultats montrent que l'algorithme se déroule en un temps négligeable par rapport au temps d'exécution de la charge optimisée (cf. figure 5.7).

5.3 Validation sur Oracle11g

Pour valider les résultats de simulation, nous déployons les solutions $(\mathcal{P}, BM)$ produites sur notre SGBD. Le planning d'exécution obtenu en utilisant notre algorithme (QBE-DBM) est exécuté sur une base de données réelle (SSB de $100Go$) localisée sur le serveur en utilisant le SGBD Oracle11g. La taille du buffer est de $20Go$ comme dans les simulations.

La figure 5.8 montre qu'il existe un sous-ensemble de requêtes qui ne bénéficient d'aucun gain du processus d'optimisation. Ces requêtes sont soit : (1) sans aucune interaction avec les autres, (2) sans aucun nœud pertinent dans le buffer (*cache miss*) ou (3) sont choisies pour être une queen-bee. Les autres requêtes ont une réduction considérable dans leur coût en utilisant les objets cachés auparavant par les queen-bee, qui sont gardées dans le buffer aussi longtemps que possible par le DBM.

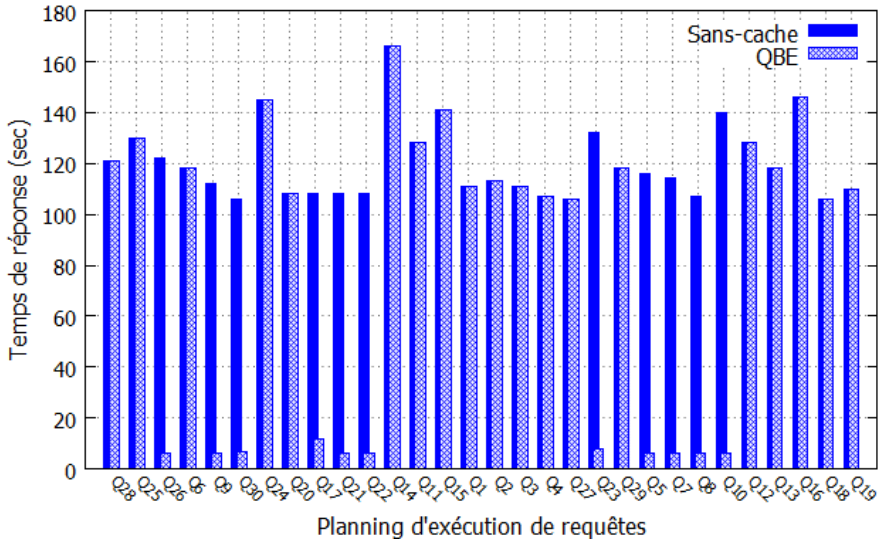

FIGURE 5.8 – Temps d'exécution de chaque requête dans le planning obtenu par la Queen-Bee En-ligne

6 Conclusion

Dans ce chapitre, nous avons présenté une approche originale pour traiter la gestion du buffer et l'ordonnancement des requêtes dans un contexte en-ligne. Notre approche est une adaptation de la proposition détaillée dans le chapitre 3 concernant la GBOR hors-ligne. L'interaction des requêtes a été exploitée pour améliorer le coût d'exécution en considérant les aspects en-ligne. Un modèle de coût avancé est proposé pour évaluer les requêtes en considérant à la fois le contenu du buffer, l'ordre des requêtes et les coûts supplémentaires liés à l'exécution en-ligne de l'ordonnanceur.

L'algorithme proposé est déterministe avec une complexité algorithmique polynomiale qui permet d'effectuer l'ordonnancement plus rapidement qu'une méta-heuristique. La politique de gestion du buffer associée à l'optimiseur dépend des connaissances de l'administrateur : si la charge n'est pas connue, alors la politique LRU est employée, sinon, nous employons notre DBM proposée pour la GBOR hors-ligne.

L'étude expérimentale est conduite en deux temps : (1) théoriquement en utilisant notre modèle de coût et notre simulateur Java, et (2) une validation des résultats de simulation sur une base de données réelle de 100Go avec 20Go de buffer. Les résultats montrent que notre approche permet d'atteindre 25% de gain dans la charge utilisée, qui est meilleur que le gain apporté par les techniques isolées ou LRU.

Il ressort de notre étude, que la conception physique peut être considérée en-ligne. La gestion du buffer et l'ordonnancement sont l'une des techniques d'optimisation qui permettent de considérer l'aspect dynamique. Leur exploitation dans la conception physique fournit des performances considérables pour les requêtes dans un environnement dynamique.

Chapitre 6

Outil d'aide à la conception physique

Sommaire

1 Introduction

La phase de conception physique impose à l'administrateur d'effectuer une bonne sélection de structure d'optimisation, afin d'augmenter les performances du système. Chacune des structures d'optimisation supportées par un SGBD, présente des avantages et des limites. Vue l'interaction existante entre elles, il n'existe pas de solution aboutissant à tous les coups à la meilleure combinaison de ces structures. De plus, l'administrateur doit choisir l'ensemble d'algorithmes pour chaque problème lié aux techniques d'optimisation existantes.

L'évolution et les changements fréquents que subit la base de données rendent, à chaque instant, le choix de l'administrateur plus complexe. Ce choix est crucial pour maintenir une bonne performance du système, notamment depuis l'arrivée du Big Data (collections de données volumineuses et un nombre important de requêtes). L'étude menée par Fabio et al. [155], montre l'effort considérable que doit fournir l'administrateur pour la conception physique.

Ces difficultés ont motivé des recherches poussées dans le milieu industriel. Plusieurs SGBD commerciaux proposent des solutions d'assistance à la conception physique. C'est le cas de *Oracle SQL Acces Advisor* [6] et *DB2 Design Advisor* [201], que nous avons déjà cité dans le chapitre 2. Ces outils permettent de proposer à l'administrateur des recommandations concernant un ensemble de structures d'optimisation comme la fragmentation horizontale, les index ou les vues matérialisées. La robustesse de ces outils a été étudiée dans [91], et l'étude révèle les limites de ces outils face aux requêtes non représentatives, ou aux hypothèses non valides dans un environnement réel. De plus, les outils existants négligent les aspects suivants :

- L'interaction entre les requêtes, les supports, les structures et les données;
- Le buffer de base de données et sa politique de gestion;
- L'ordre d'exécution des requêtes;
- La réactivité des algorithmes d'optimisation;
- L'aspect en-ligne de l'optimisation.

Dans ce chapitre, nous présentons un outil d'aide à la conception physique, appelé : *Physical Design Advisor*. Cet outil implémente l'ensemble des algorithmes étudiés le long de ce travail. Il permet de proposer des recommandations pour un ensemble de structures d'optimisation, à savoir la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes. Il supporte donc la sélection isolée et multiple, ainsi que les modes hors-ligne et en-ligne.

D'un point de vue organisationnel, ce chapitre s'articule sur six sections. La Section 2 concerne l'étude des difficultés liées à la conception physique. La Section 3 décrit l'architecture fonctionnelle selon laquelle l'outil a été conçu. Nous détaillons par la suite la simulation dans la Section 4, qui montre les cas d'utilisation de l'outil, et l'interprétation des recommandations qu'il fournies. Dans la Section 5, nous décrivons le processus de validation requis pour différents cas d'utilisation. Nous concluons le chapitre par un bilan et un ensemble de perspectives.

2 Difficultés et besoins des administrateurs

La performance des requêtes est au cœur de la conception physique. Avec l'augmentation du volume des bases de données, et la complexité croissante des requêtes, la performance devient une exigence primordiale pour les concepteurs et les administrateurs. La tâche de l'administrateur est de plus en plus complexe, car il est amené à faire des choix liés à plusieurs aspects : (1) les structures d'optimisation, (2) le mode de sélection, (3) le mode d'optimisation et (4) les algorithmes de sélection et leurs paramètres. Ces choix sont déterminants en matière de performance et de réactivité.

2.1 Choix des structures d'optimisation

Il existe un grand nombre de structures d'optimisation supportées par les SGBD. Nous les avons classées dans le chapitre 2 en deux catégories : les structures non redondantes et les structures redondantes. Dans la première catégorie, les schémas d'optimisation ne nécessitent pas un coût de stockage et de maintenance supplémentaire. Entrent dans cette catégorie la fragmentation horizontale, l'ordonnancement et le traitement parallèle. La deuxième catégorie impose un coût de stockage ou de maintenance supplémentaire. Nous citons la fragmentation verticale, les index, et les vues matérialisées.

Face à cette multitude de choix, la sélection des structures pertinentes s'avère très difficile pour l'administrateur.

2.2 Choix du mode de sélection

En plus de la multitude des structures, l'administrateur doit décider du mode de sélection. Une sélection isolée consiste à choisir une seule structure pour la conception physique. Une sélection multiple consiste à en choisir plusieurs. La sélection multiple souvent plus adaptée pour l'optimisation des bases de données, car chaque structure permet de rajouter un gain supplémentaire en performance. Cependant, le choix de la meilleure combinaison est également une tâche difficile. Nous avons montré que les structures d'optimisation interagissent entre-elles. Cela doit être pris en compte pour établir une bonne conception. Or, l'administrateur est souvent incapable d'analyser les interactions entre les différentes composantes de la base de données. Sans cette analyse, il est difficile d'établir les dépendances permettant de choisir la meilleure combinaison de structures d'optimisation.

2.3 Choix du mode d'optimisation

Un système de base de données peut recevoir des requêtes en provenance d'un grand nombre d'utilisateurs. Les techniques d'optimisation, comme la fragmentation horizontale et les index, sont majoritairement implantées en mode hors-ligne. Leurs algorithmes ne peuvent être appliqués en mode en-ligne. Or, pour gérer l'aspect concurrent et l'évolution des requêtes, des données et des structures d'optimisation, l'administrateur a besoin de techniques d'optimisation en-ligne, i.e. qui peuvent fournir des recommandations de manière interactive. Les outils existants ne proposent pas de fonctionnalités de ce type.

2.4 Choix des algorithmes et de leurs paramètres

Il existe plusieurs algorithmes permettant de traiter chaque problème lié à une structure d'optimisation. Ces algorithmes peuvent être classés en deux catégories : les algorithmes simples, et les algorithmes lourds. La première catégorie permet de trouver des solutions dont la performance est limitée, dans un temps très court. La deuxième catégorie propose des solutions plus efficaces, mais sous un délai plus long (plusieurs dizaines de minutes pour une charge de 30 requêtes sur une BD de 100Go, par exemple). Ainsi, l'administrateur est confronté au dilemme efficacité-réactivité, que nous avons discuté dans le chapitre 3. Les algorithmes permettant de faire ce compromis, dont notre algorithme Queen-Bee (cf. chapitre 3) et Requête-Elue (cf. chapitre 4), sont rares et ne prennent pas en compte toutes les structures existantes (e.g. les index ou les vues matérialisées).

3 Architecture fonctionnelle de l'outil de simulation

L'outil *Physical Design Advisor* a été conçu pour permettre de contourner les difficultés liés aux choix de l'administrateur lors de la conception physique. Cet outil est développé dans un environnement Java. L'outil est doté d'une passerelle permettant de se connecter à une base de données Oracle11g. L'étude des besoins des administrateurs/concepteurs a permis d'identifier les tâches que l'outil doit considérer. La première phase est la collecte des méta-données requises pour le processus d'optimisation. Deux cas sont possibles :

- Si la base de données existe, l'administrateur s'y connecte et introduit l'ensemble des requêtes qui lui sont associées. L'outil extrait les méta-données à partir de ses entrées, et crée une méta-base pour les stocker.
- Si la base de données n'existe pas, l'administrateur doit fournir les profils et les statistiques requises pour remplir la méta-base directement.

Après l'extraction des méta-données, l'administrateur peut faire les différents choix liés aux structures d'optimisation, aux algorithmes et aux modes de sélections. Un tableau de bord est mis à sa disposition pour effectuer les différents paramétrages avancés. Ensemble avec les paramètres instanciés, les méta-données permettent de préparer les entrées du processus de la conception physique et alimenter le modèle de coût.

La conception physique est menée selon les choix établis par l'administrateur. L'outil permet d'aiguiller la conception vers trois principaux chemins possibles :

1. la gestion du buffer et l'ordonnancement des requêtes;
2. la fragmentation horizontale;
3. la combinaison de la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes.

Dans chaque cas, nous employons les approches détaillées dans les chapitres 3 et 4. L'outil intègre l'ensemble des algorithmes étudiés, sur lesquels il se base pour fournir des recommandations. Si les recommandations sont satisfaisantes pour l'administrateur, il peut les déployer avec les scripts correspondants, sinon le processus recommence avec de nouveaux paramètres. La figure 6.1 représente l'architecture fonctionnelle de *Physical Design Advisor*.

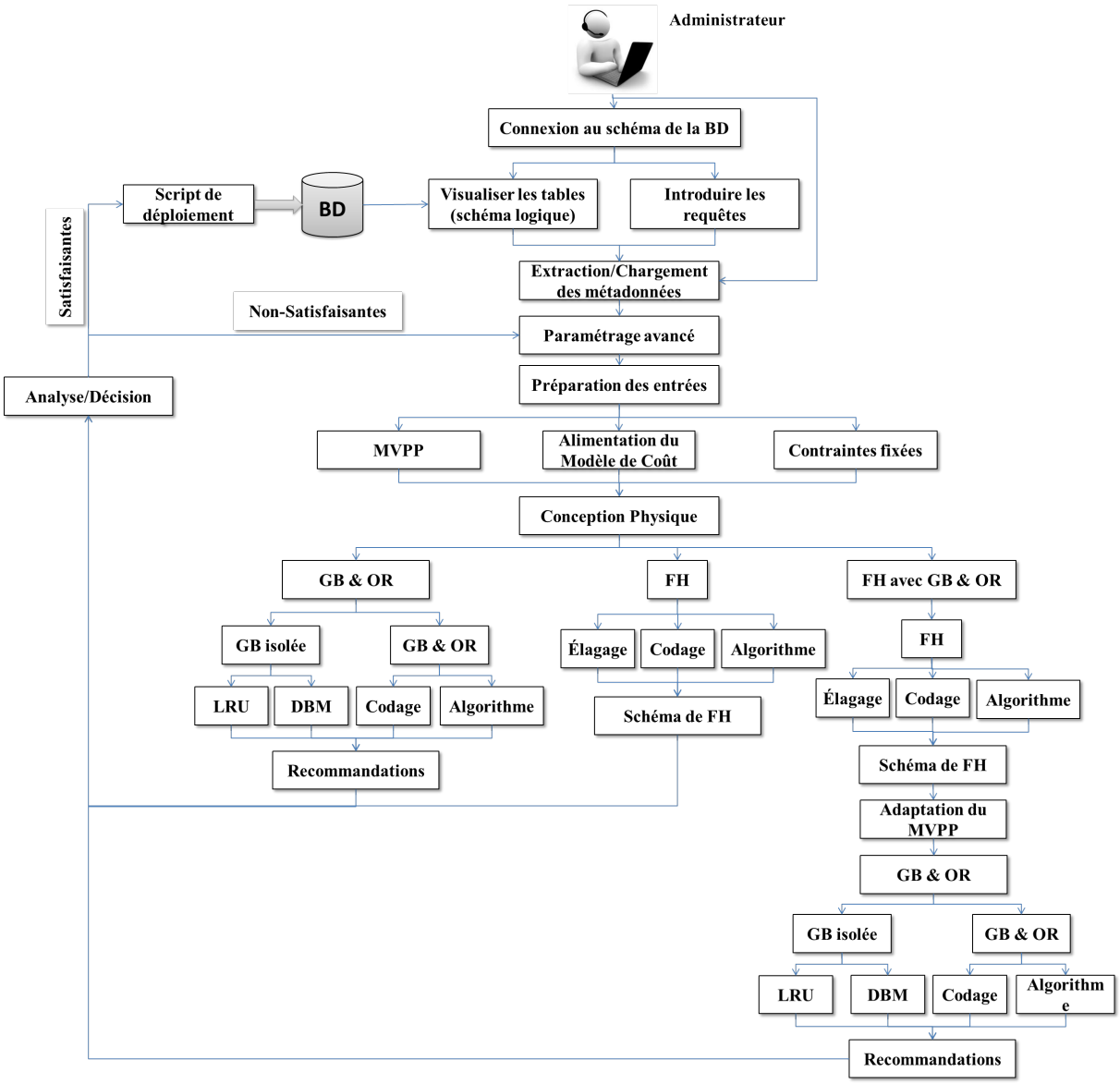

FIGURE 6.1 – Architecture fonctionnelle de l'outil d'aide à la prise de décision : *Physical Design Advisor*

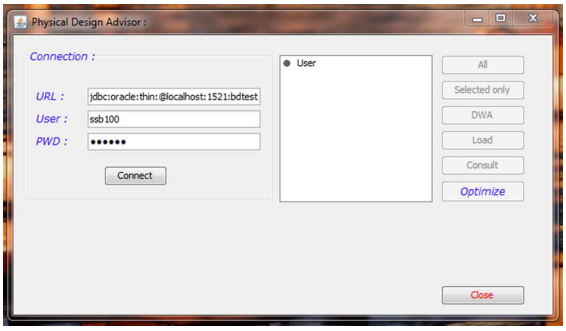

FIGURE 6.2 – Interface principale de l'outil

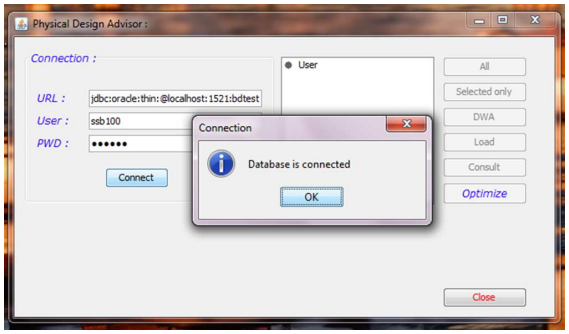

FIGURE 6.3 – Connection à la base de données

4 Simulations

Dans cette section, nous décrivons en détail les phases d'exploitation de l'outil. Nous commençons par présenter la phase de préparation des entrées, ainsi que les consignes permettant d'exprimer les besoins de l'administrateur. Nous expliquons également quels paramétrages sont requis pour obtenir des recommandations selon les exigences des utilisateurs. Finalement, nous donnons les éléments permettant d'interpréter les résultats fournis par l'outil dans différents scenarii.

4.1 Préparation de la simulation et expression des besoins

Au démarrage de la simulation, il existe deux scenarii selon les besoins de l'administrateur, comme nous l'avons décrit dans l'architecture fonctionnelle :

a Si la base de données à optimiser existe déjà, il suffit de s'y connecter sur le module de connexion comme le montre les figures 6.2 et 6.3) pour procéder à la simulation.

b Sinon, l'administrateur doit fournir le schéma logique des tables, ainsi que les statistiques relatives aux données et aux requêtes. L'outil demande à l'administrateur de se connecter au schéma de base de données sur lequel il veut effectuer l'optimisation. Les statistiques permettent de construire la méta-base contenant les méta-données. L'administrateur peut ainsi passer directement à l'étape d'optimisation que nous allons décrire plus loin. Ce scénario est un cas particulier du scénario (a).

La connexion à la base de données permet de visualiser les tables existantes dans le schéma. Toutes les bases de données seront présentées avec leurs tables dans une arborescence. La figure 6.4 montre un cas d'exécution sur une base de données contenant un seul schéma : il s'agit de l'entrepôt du Benchmark SSB contenant une table des faits (lineorder) et quatre tables de dimensions. L'administrateur peut sélectionner une partie ou la totalité des tables participant au processus d'optimisation. D'autres part, l'administrateur peut utiliser le module *DWA* (*Data and Workload Administration*) afin d'introduire les informations sur les requêtes (cf. figure 6.5), ainsi que les clés étrangères reliant les tables (cf. figure 6.6). Une fois les tables sont sélectionnées et les données des requêtes sont introduites, les méta-données peuvent être extraites et chargées dans une méta-base créée automatiquement par l'outil (cf. figure 6.7). L'extraction de ses données s'effectue en analysant les propriétés des tables et des requêtes. La méta-base créée permet de fournir toutes les données nécessaires pour alimenter le modèle de coût, le codage (structures de données) et les algorithmes.

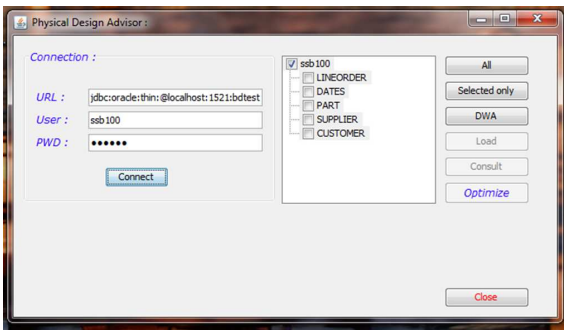

FIGURE 6.4 – Visualisation des données

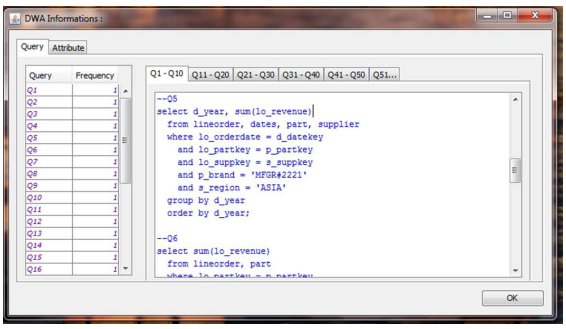

FIGURE 6.5 – Introduction des requêtes

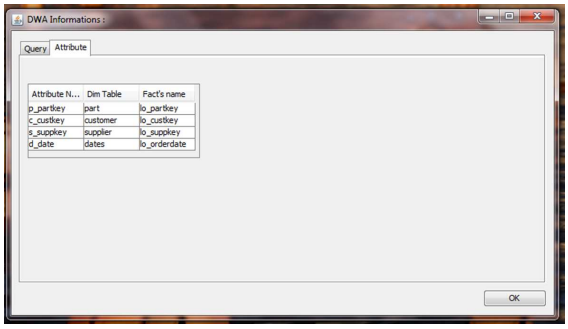

FIGURE 6.6 – Définition des clés étrangères

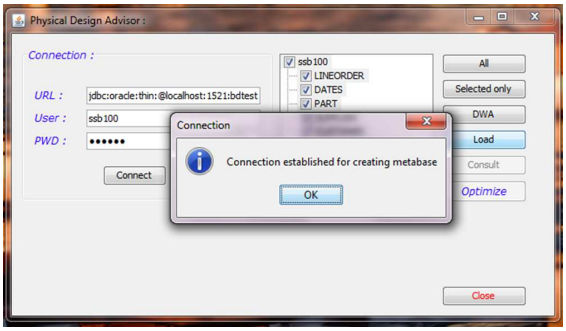

FIGURE 6.7 – Extraction des méta-données

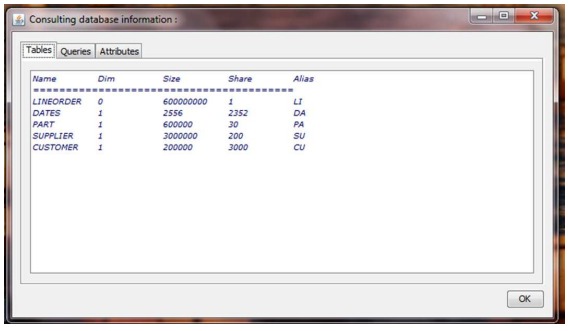

FIGURE 6.8 – Méta-données des données chargées

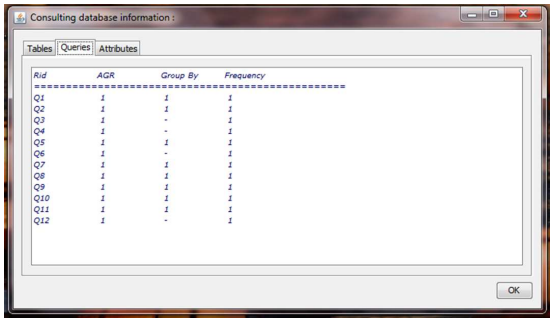

FIGURE 6.9 – Méta-données des requêtes

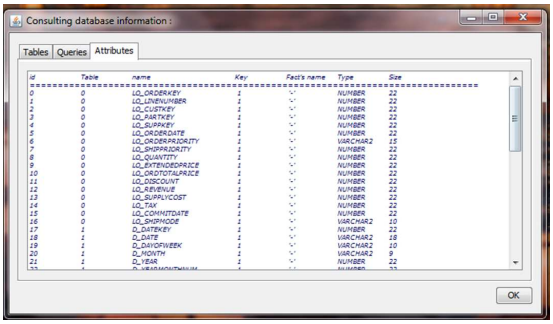

FIGURE 6.10 – Méta-données des attributs

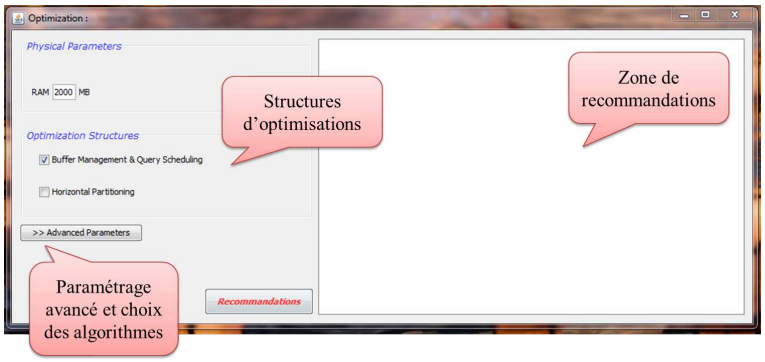

FIGURE 6.11 – Interface des choix d'optimisation et des recommandations

L'administrateur peut visualiser la méta-base créée, en utilisant le module *Consult*, qui permet de présenter les informations extraites des trois éléments : (1) les tables, (2) les requêtes et (3) les attributs. Dans notre jeu de données, le contenu de la méta-base est présenté dans les trois figures 6.8, 6.9 et 6.10. Le modèle de coût s'appuie sur des informations comme les cardinalités des tables, les types des attributs ou les opérations des requêtes, afin de fournir le coût d'exécution en termes d'Entrées/Sorties pour chaque requête.

Une fois les méta-données extraites, la phase d'optimisation peut s'effectuer en utilisant le module *Optimization*. La phase d'optimisation nécessite une préparation des entrées et plusieurs choix liés aux besoins de l'administrateur (cf. figure 6.11).

Rappelons que la conception physique supporte deux modes : Sélection Isolée et Sélection Multiple. Ces deux modes, détaillés dans le chapitre 2, dépendent de l'ensemble de structures d'optimisation choisies. Si l'ensemble contient une seule structure, il s'agit d'une sélection isolée. Dans le cas contraire, il s'agit d'une sélection multiple. Dans notre outil, l'ensemble des structures supportées comporte la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes.

La sélection isolée est possible dans notre outil pour les deux scenarii suivants :
- La Fragmentation Horizontale seule.
- La Gestion du Buffer seule.

La sélection multiple est possible dans notre outil pour les trois scenarii suivants :

- La Gestion du Buffer et Ordonnancement des Requêtes

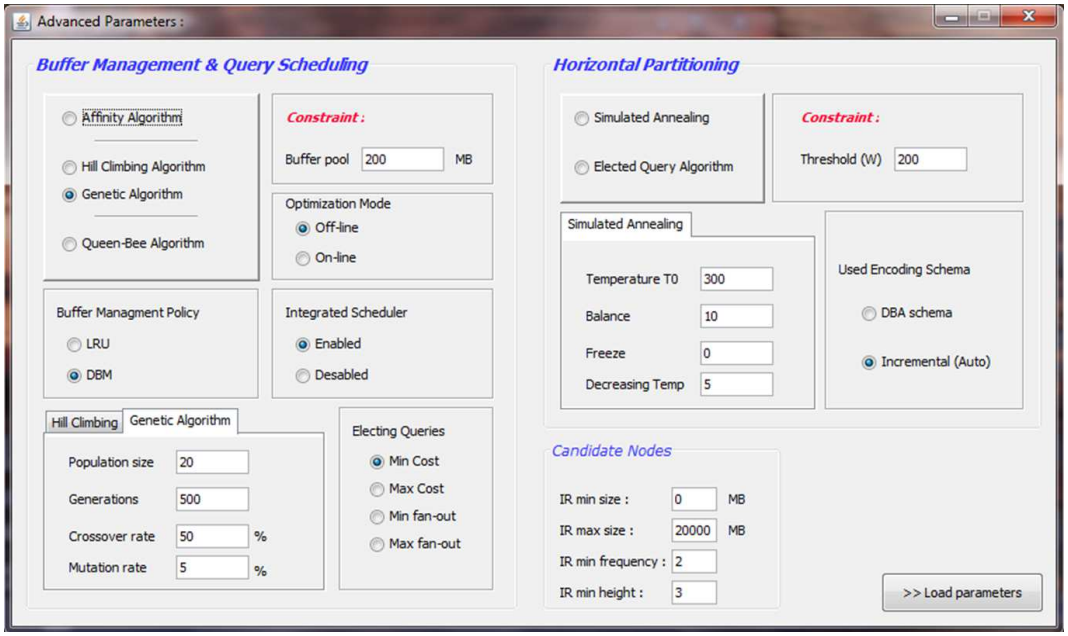

FIGURE 6.12 – Tableau de bord de la conception physique

– La Fragmentation Horizontale avec Gestion du Buffer
– La Fragmentation Horizontale avec Gestion du Buffer et Ordonnancement des Requêtes.

Chaque scénario nécessite un paramétrage avancé selon les besoins et les exigences de l'administrateur. A ce niveau, l'administrateur doit exprimer les différents choix relatifs aux algorithmes, aux contraintes et à l'architecture physique. Le module *Advanced Parameters* contient une panoplie d'algorithmes liés à chaque structure d'optimisation, chacun d'eux nécessitant un paramétrage spécifique. La figure 6.12 montre le tableau de bord de la conception physique proposé par l'outil *Physical Design Advisor*.

Le module *Advanced Parameters* contient deux principales parties :
– Paramétrage pour la Gestion du Buffer et l'Ordonnancement des Requêtes.
– Paramétrage pour la Fragmentation Horizontale.

L'administrateur est amené à définir les paramètres pour toutes les structures impliquées dans sa sélection (isolée ou multiple). Par exemple, si la sélection adoptée est une sélection multiple combinant à la fois la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes, l'administrateur doit régler tous les paramètres liés à ces structures. Cependant, si la fragmentation est la seule structure adoptée, les paramètres de la gestion du buffer et l'ordonnancement sont ignorés.

La partie Gestion du Buffer et Ordonnancement des Requêtes fournit les éléments suivants :

– Les algorithmes intégrés : il en existe trois classes d'algorithmes, comme nous l'avons décrit dans le chapitre 3. La première classe comporte l'algorithme d'Affinité, la deuxième comporte les deux méta-heuristiques : l'Algorithme Génétique et le Hill Climbing, la dernière classe comporte l'algorithme Queen-Bee. Un de ces algorithmes peut être employé pour le processus d'optimisation.
– Les contraintes : la seule contrainte est liée à la gestion du buffer, c'est la taille de la mémoire cache.

- Les modes d'optimisation : la gestion du buffer et l'ordonnancement peuvent s'effectuer hors-ligne (en connaissant la charge), ou en-ligne (avec ou sans connaissance de la charge).
- L'ordonnanceur : pour une sélection isolée de la gestion du buffer, l'ordonnanceur intégré peut être désactivé.
- La politique de gestion du buffer : la gestion du buffer peut être basée sur la politique LRU, ou sur la nouvelle politique DBM, que nous proposons. Contrairement à LRU, La politique DBM prend en considération l'interaction entre les requêtes, mais elle nécessite une connaissance préalable de la charge.
- Les paramètres des méta-heuristiques : l'Algorithme Génétique et le Hill Climbing nécessitent la définition des paramètres requis pour leur exécution. Les paramètres sont décrits dans le chapitre 3. Le bon choix de ces paramètres permet un meilleur rendement de l'optimiseur.
- Les paramètres de l'algorithme Queen-Bee : l'identification des requêtes *queen-bee*, nécessite de définir le critère d'élection. Nous avons identifié deux critères susceptibles d'impacter le choix de la *queen-bee* : le coût d'exécution, et le fan-out (cf. chapitre 3).

La partie Fragmentation Horizontale fournit les éléments suivants :

- Les algorithmes intégrés : l'optimiseur permet de choisir entre deux algorithmes : un Recuit Simulé et notre algorithme de Requêtes Elue.
- Les contraintes : la seule contrainte liée à la fragmentation horizontale est le seuil des fragments W, selon les exigences de l'administrateur.
- Les paramètres du Recuit Simulé : l'algorithme RS nécessite la définition des paramètres suivants : La température initiale, le gel du système (condition d'arrêt), l'équilibre, et le palier de décroissance de température à chaque itération.
- Le codage : la structure de données dynamique peut être une alternative au codage proposé par l'administrateur.

L'élagage des nœuds candidats dans le MVPP a pour but de réduire l'espace de recherche et ainsi renforcer les algorithmes d'optimisation. Le stockage en mémoire cache des nœuds peut être soumis à des conditions préalables afin d'écarter certains candidats :

- Taille minimale de résultats intermédiaires : définit la taille minimale des nœuds du MVPP à considérer. Elle permet de favoriser les nœuds ayant une taille moins importante pour permettre d'allouer le cache pour un plus grand nombre de nœud, avant qu'il ne soit saturé.
- Taille maximale de résultats intermédiaires : définit la taille maximale des nœuds du MVPP à considérer. Elle permet de favoriser les nœuds de tailles importantes pour permettre d'optimiser les requêtes les plus coûteuses, par exemple.
- Fréquence minimale de résultats intermédiaires : définit la fréquence minimale des nœuds du MVPP à considérer. Elle permet de favoriser les nœuds utilisés par un plus grand nombre de requêtes dans la charge.
- Hauteur minimale de résultats intermédiaires : définit la hauteur minimale des nœuds dans les plans d'exécution. La hauteur 0 correspond aux tables, la hauteur 1 correspond aux sélections, la hauteur 3 correspond aux premières jointures, etc. Ceci permet de définir le type de sous-expressions prises en compte.

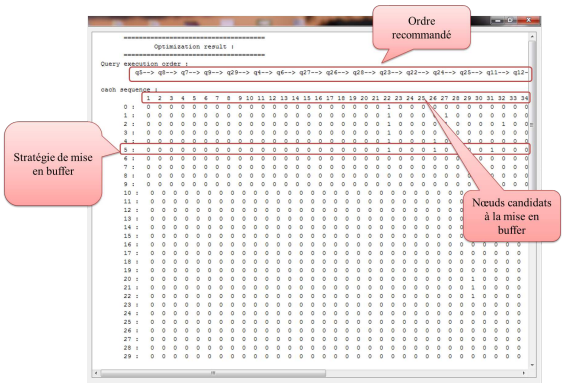

FIGURE 6.13 – Recommandations pour la GBOR

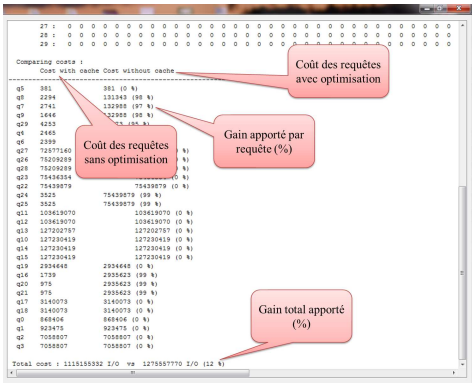

FIGURE 6.14 – Gain apporté par la GB et l'OR

4.2 Interprétation des recommandations pour la gestion du buffer et l'ordonnancement des requêtes

Dans cette section nous exposons un exemple de recommandations obtenues pour la gestion du buffer et d'ordonnancement hors-ligne. La figure 6.13 montre une partie des résultats contenant le planning d'exécution recommandé par l'outil, ainsi que la stratégie de gestion du buffer correspondante. La lecture des résultats est la suivante : la charge de requête optimisée doit s'effectuer dans l'ordre proposé. Autrement dit, la requête Q_5 sera la première, suivie par Q_8, Q_7, etc. jusqu'à la fin du planning.

La stratégie de gestion du buffer correspondant à ce planning est exprimée de la façon suivante : les colonnes sont étiquetées par les nœuds du MVPP. A chaque exécution d'une requête dans le planning $\mathcal{P}$ retourné, cette requête déclenche une allocation d'un ou plusieurs nœuds, et un remplacement d'autres nœuds. Ainsi, l'état du buffer après l'exécution de chaque requête est représenté par la ligne correspondante de la matrice de gestion du buffer. La valeur 0 représente l'absence du nœud dans le buffer, et 1 représente sa présence. Dans la figure 6.13, après exécution de la sixième requête du planning (Q_4), le buffer est alloué pour le nœud 31. Les autres nœuds 22 et 26 y étaient déjà.

Dans la suite des recommandations (cf. figure 6.14), l'outil fournit le détail des coûts d'exécution pour chaque requête, avec et sans la gestion du buffer. Les coûts sont exprimés en termes d'Entrées/Sorties. Le coût de la charge globale est fourni également pour montrer la performance apportée à la charge lors d'une optimisation hors-ligne.

Lors d'une optimisation en-ligne, en plus du planning et de la stratégie de la gestion du buffer, l'outil fournit le temps écoulé pour le traitement de chaque sous-ensemble de requêtes arrivant à un instant donné. Le temps est exprimé en millisecondes. L'outil détaille le gain apporté à la charge après chaque arrivée d'un sous-ensemble de requêtes. La figure 6.15, illustre un exemple de recommandation pour une optimisation en-ligne.

4.3 Interprétation des recommandations pour la fragmentation horizontale

La fragmentation s'effectue sur le codage établi par l'administrateur, ou engendré par l'algorithme lui-même, selon le choix. Le résultat final donne la décomposition des attributs participants dans le

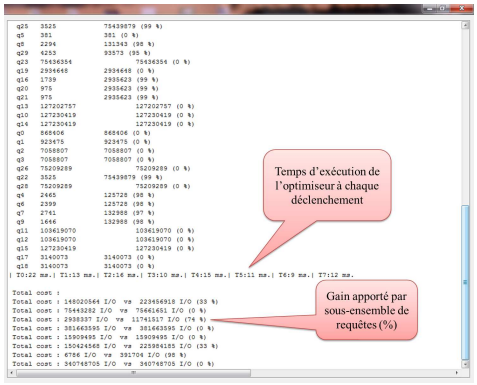

FIGURE 6.15 – Recommandations pour le mode en-ligne

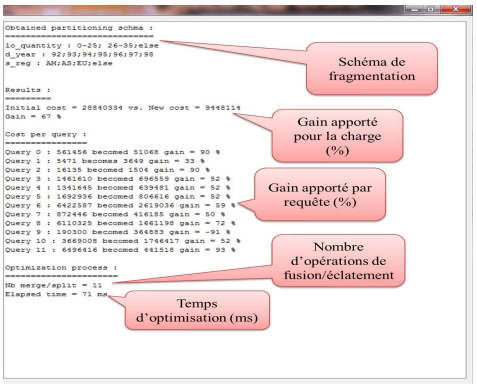

FIGURE 6.16 – Recommandations pour la fragmentation horizontale

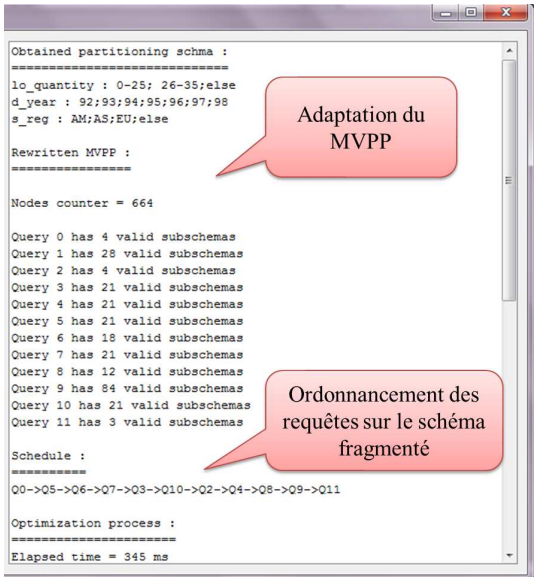

FIGURE 6.17 – Recommandations pour la combinaison des trois techniques

processus de partitionnement.

Le script de fragmentation se base sur cette décomposition pour créer le schéma partitionné de la base de données. La base de données fragmentée est construite à l'aide des scripts de partitionnement supportés par le SGBD hôte. Si la base à optimiser est déjà fragmentée horizontalement, une adaptation est requise pour passer au nouveau schéma. Cette adaptation est assurée par les deux primitives d'éclatement et de fusion (*Split* et *Merge* dans le cas d'un SGBD Oracle par exemple). Les détails de ce processus sont fournis dans le travail de Boukhalfa, 2009 [52].

4.4 Interprétation des recommandations pour la sélection multiple des trois techniques

Si une sélection multiple implique les trois structures, la simulation de la conception physique fournit trois parties de recommandations :

- La recommandation pour la création du schéma de partitionnement;
- La réécriture du MVPP correspondant, en projetant les anciens plans de requêtes sur le nouveau schéma partitionné;
- La GB et OR du MVPP réécrit.

L'outil fournit les coûts d'exécutions relatifs à chaque structure, en comparant le coût de la charge initial, le coût après partitionnement, et le coût après gestion du buffer.

5 Validation des résultats sous Oracle11g

Les recommandations retenues par l'administrateur sont déployées sur le SGBD afin d'obtenir les performances estimées par le simulateur. Dans notre étude, nous avons utilisé un banc d'essai SSB sur un SGBD Oracle11g. Le déploiement de chaque solution nécessite un script particulier, étant donné que les structures choisies sont supportées par le SGBD. Dans cette section, nous détaillons le processus de déploiement des solutions pour la gestion du buffer, l'ordonnancement des requêtes, et la fragmentation horizontale.

5.1 Déploiement de la gestion du buffer et l'ordonnancement

Le script de déploiement comporte trois éléments : l'entête pour le tuning du buffer, la charge optimisée et les instructions systèmes.

L'entête du script : elle contient l'ensemble des instructions permettant de définir les paramètres physiques, et de préparer le buffer pour le processus d'optimisation. La figure 6.22 montre un exemple d'entête. Il existe un grand nombre de paramètres système qui concerne la mémoire cache. Dans notre étude, nous exposons un ensemble restreint de paramètres nécessaires pour le bon fonctionnement de la gestion du buffer. Parmi ces paramètres, nous citons les trois variables suivantes :

- La variable *db_cache_size* permet de définir la taille du cache réservé par le SGBD.
- La variable *result_cache_max_size* représente la taille maximale admise pour qu'un résultat d'une requête soit mis dans le cache.
- La variable *db_cache_size* représente le pourcentage d'occupation admis pour qu'un résultat d'une requête soit mis dans le cache.

La mémoire cache des résultats de requêtes peut être vidée de différentes manières, par exemple en utilisant la commande suivante : *execute dbms_result_cache.flush ;*. Il est fortement recommandé d'utiliser cette commande avant le lancement des requêtes pour assurer une utilisation maximale de la mémoire disponible. Dans la figure 6.18, nous présentons le résultat d'exécution de l'entête pour un jeu de données.

La charge optimisée : L'ensemble des requêtes est réécrit selon les recommandations de l'outil. La réécriture permet de ressortir les nœuds candidats pour le buffer. Les requêtes se présentent dans le script sous forme réécrite. L'ordre d'apparition des requêtes dans le script respecte le planning d'exécution retourné par l'outil. Dans la figure 6.22, les requêtes sont réécrites et exécutées dans l'ordre recommandé. Chaque requête est décomposée en un ensemble de sous-expressions imbriquées. Chaque sous-expression représente un objet potentiel du buffer. Ainsi, les nœuds choisis pour être mis dans le

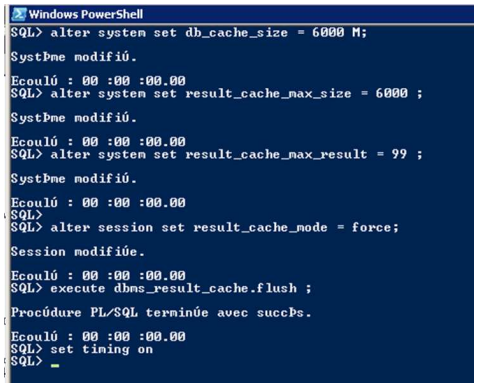

FIGURE 6.18 – Préparation de la mémoire cache du SGBD Oracle11g

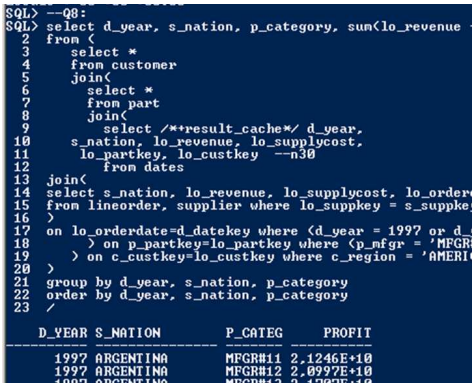

FIGURE 6.19 – Exécution de la charge dans le script des recommandations

buffer, seront munis d'un *hint* (une clause SQL permettant de donner des recommandation au moteur d'exécution du SGBD) [6] de la façon suivante :

```
select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit
 from (
    select *
    from customer
    join(
        select *
        from part
        join(
            select /*+result_cache*/ *  --n30
            from dates
            join(
                select /*+result_cache*/ *  --n25
                from lineorder, supplier
                where lo_suppkey = s_suppkey
                  and s_region = 'AMERICA'
            ) on lo_orderdate=d_datekey
            where (d_year = 1997 or d_year = 1998)
        ) on p_partkey=lo_partkey
        where (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
    ) on c_custkey=lo_custkey
    where c_region = 'AMERICA')
group by d_year, s_nation, p_category
order by d_year, s_nation, p_category;
```

Comme nous l'avons présenté dans le chapitre 3, les nœuds candidats peuvent avoir une taille élevée si la taille des tables augmente. Pour réduire cette taille, nous avons proposé un algorithme permettant de définir l'ensemble minimal d'attributs pour chaque sous-expression. Ainsi, la requête peut être améliorée en projetant les attributs requis uniquement. Voici un exemple de projection permettant de réduire la taille des nœuds :

```
select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit
from (
   select *   from customer
   join(
```

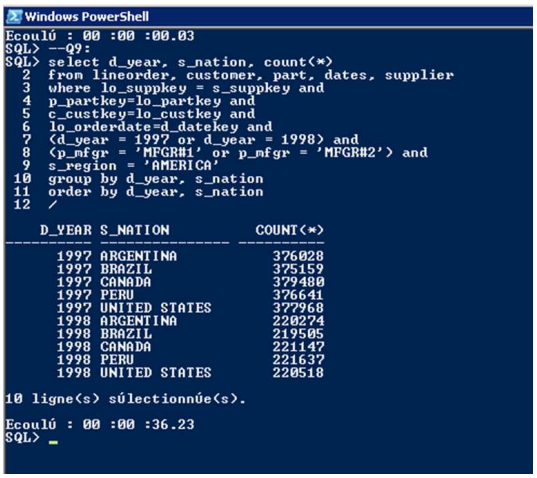

FIGURE 6.20 – Temps de réponse sans optimisation

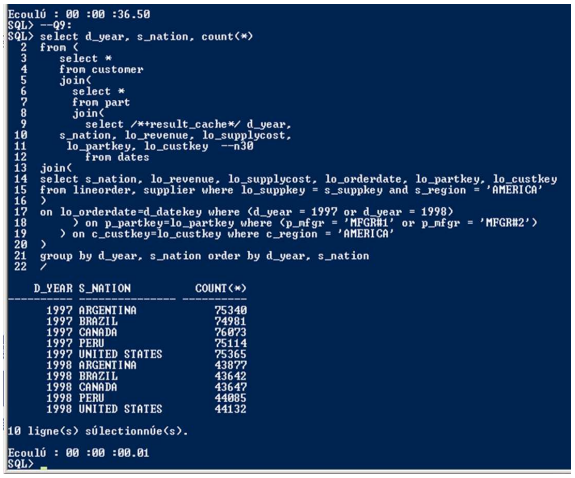

FIGURE 6.21 – Temps de réponse avec optimisation

```
     select *   from part
     join(
       select /*+result_cache*/ d_year, s_nation, lo_revenue, lo_supplycost, lo_partkey, lo_custkey  --n30
       from dates
join(
       select s_nation, lo_revenue, lo_supplycost, lo_orderdate, lo_partkey, lo_custkey
       from lineorder, supplier where lo_suppkey = s_suppkey and s_region = 'AMERICA'
        ) on lo_orderdate=d_datekey where (d_year = 1997 or d_year = 1998)
     ) on p_partkey=lo_partkey where (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
   ) on c_custkey=lo_custkey where c_region = 'AMERICA'
)
group by d_year, s_nation, p_category
order by d_year, s_nation, p_category
/
```

Pour montrer le déroulement de l'exécution avec et sans optimisation, nous exécutons la requête Q_9 de SSB sur l'entrepôt. Dans la figure 6.20, nous montrons le temps de réponse de la requête sans optimisation. Dans la figure 6.21, nous montrons le temps de réponse obtenu par le script implémentant les recommandations. La comparaison des deux cas d'exécution montre la réduction considérable du temps de réponse de Q_9, fournie par le script.

Les instructions systèmes : Les nœuds munis d'un *hint* apparaissent dans la vue système :

```
v$$result_cache_objects
```

Chaque nœud a un identifiant. L'allocation d'un nœud se fait une fois la requête contenant ce nœud, est exécutée. Cependant, l'effacement d'un nœud nécessite une instruction supplémentaire. Oracle11g permet de supprimer un objet dans la mémoire cache des résultats de requêtes, à l'aide de la commande suivante :

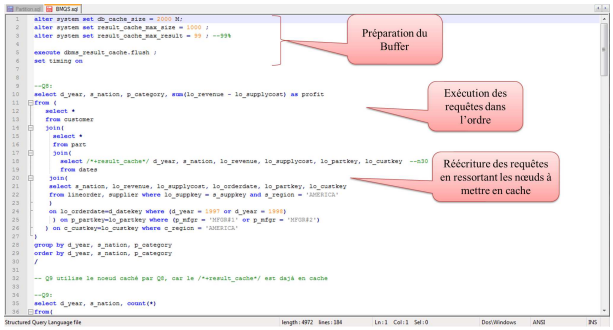

FIGURE 6.22 – Libération des objets du cache

```
exec dbms_result_cache.invalidate_object(3);
```

Dans cette instruction, l'appel de l'objet à ôter est effectué soit par le nom de l'objet dans la vue *v$ result_cache_objects*, soit par son identifiant. La figure 6.22 montre les instructions systèmes insérées pour libérer un résultat de la mémoire cache (nœud no_{21}). Cette commande permet de libérer de l'espace mémoire afin de l'allouer pour d'autres nœuds recommandés (no_{26} et no_{22}).

Dans notre étude, nous générons ce script manuellement. Pour la suite de ce travail (perspectives), nous prévoyons d'étendre notre outil afin d'automatiser la génération de ce script. Cela facilitera la tâche de l'administrateur lors de la conception physique.

5.2 Déploiement de la fragmentation horizontale

Comme nous l'avons décrit dans le chapitre 4, le déploiement de la fragmentation horizontale nécessite trois étapes : la création du schéma partitionné, l'insertion des données du schéma et la réécriture des données. Ce processus est inspiré du principe de déploiement proposé dans le travail de Boukhalfa [52].

Création du schéma partitionné : Le banc d'essai employé est un entrepôt de données en étoile. La création d'un schéma de fragmentation d'un entrepôt passe par deux étapes : la fragmentation horizontale primaire, et la fragmentation horizontale dérivée.

Pour assurer la fragmentation dérivée, la fragmentation primaire s'effectue en insérant une nouvelle colonne Col_i à chaque table de dimension. Cette colonne est destinée à identifier les fragments de chaque table. Par exemple, la table *Dates* est partitionnée selon la décomposition des sous-domaines de l'attribut *d_year*. La figure 6.23 montre l'ajout d'une nouvelle colonne dans le schéma de la table *Dates*. Pour obtenir la table de dimension fragmentée, une nouvelle table est créée ayant les mêmes attributs que la table initiale. En plus des attributs, la table est définie avec des primitives de partitionnement. Pour illustrer cette étape, la figure 6.24 montre un exemple de fragmentation primaire sur l'attribut *d_year* pour obtenir la nouvelle table de dimension *dates1*.

Une fois les tables peuplées, elles doivent être renommées pour assurer la transparence lors de l'exécution des requêtes (cf. figure 6.27).

La fragmentation dérivée est effectuée sur la table des faits selon la combinaison de valeurs des

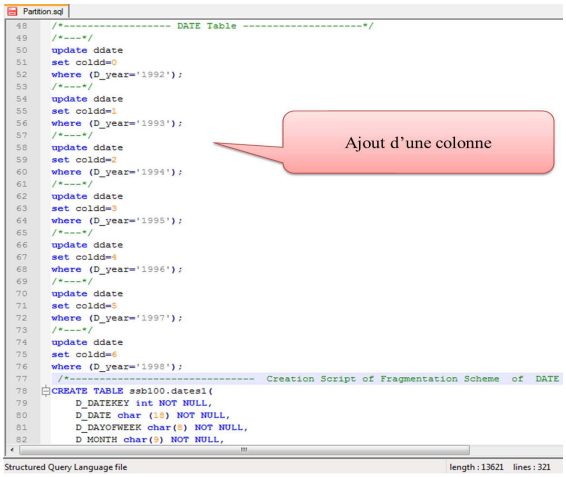

FIGURE 6.23 – Ajout d'une colonne pour la fragmentation

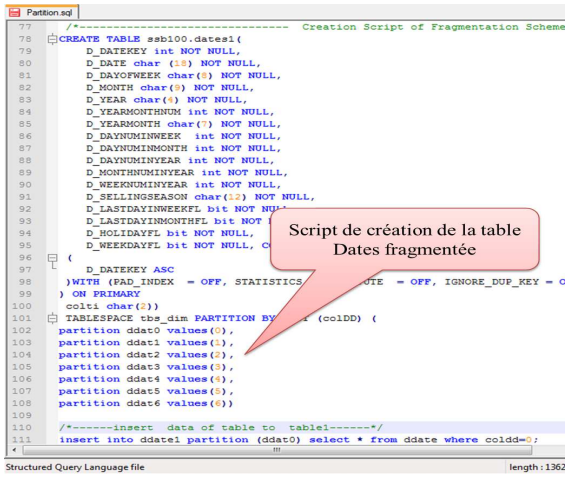

FIGURE 6.24 – Script de partitionnement de la table *Dates*

FIGURE 6.25 – Fragmentation dérivée de la table des faits

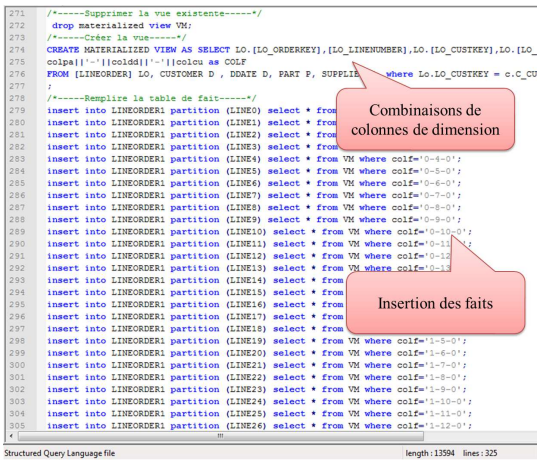

FIGURE 6.26 – Insertion des faits après la fragmentation

colonnes insérées dans les tables de dimension. Pour cela, une colonne est insérée parmi les attributs de la table des faits. La nouvelle table est partitionnée selon cette nouvelle colonne *colf*. Chaque valeur de cette colonne est une combinaison des valeurs des autres colonnes insérées dans les tables de dimension : $Col_1 - Col_2 - \ldots Col_k$. Une vue matérialisée peut servir d'intermédiaire pour construire l'ensemble des valeurs de *colf* à partir des combinaisons des colonnes de dimension. La figure 6.25 illustre ce principe. Une fois la table des faits est fragmentée, les faits sont insérés de la même façon que pour les dimensions, en se basant sur la valeur de la colonne *colf* (Voir la figure 6.26). Les fragments de faits sont donc peuplés par un partitionnement en mode LIST sur la colonne *colf*.

Enfin, la table des faits partitionnée est renommée pour assurer la transparence.

Réécriture et exécution des requêtes sur le schéma partitionné : Pour permettre aux utilisateurs d'exécuter leurs requêtes habituelles sur le schéma fragmenté, en toute transparence, la charge doit être réécrite sur le nouveau schéma suivant les étapes décrites dans la Section 5.4 du chapitre 4. Un outil

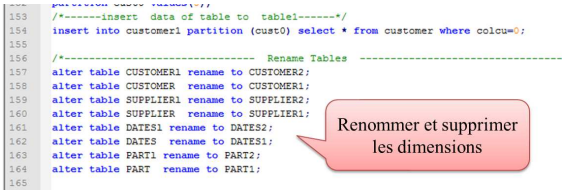

FIGURE 6.27 – Renommage les tables partitionnées

d'administration a été développé au sein de notre équipe, dans le cadre du travail de Boukhalfa, 2009 [52]. Cet outil, appelé *ParAdmin*, permet d'assurer la fragmentation, la génération des scripts partitionnement, ainsi que la réécritures automatiques des requêtes. L'outil *ParAdmin* nous a permis d'obtenir la nouvelle charge de requêtes, et ainsi d'effectuer la validation des résultats sur Oracle11g, en tirant profit de la fragmentation.

6 Conclusion

Dans ce chapitre, nous avons décrit l'outil que nous avons implémenté pour effectuer l'étude des approches proposées. Cet outil permet de supporter l'ensemble de structures d'optimisation étudiées dans ce travail, ainsi que les différents modes de sélection. L'outil *Physical Design Advisor* se base sur un modèle de coût et un ensemble d'algorithmes pour proposer des recommandations à l'administrateur selon ses besoins et ses exigences. Nous avons présenté l'architecture fonctionnelle de l'outil et la façon d'interpréter les résultats qu'il fournit. Plusieurs scenarii d'exécution sont utilisés pour illustrer les différents cas d'étude.

Les recommandations satisfaisantes peuvent être déployées sur le système en utilisant les scripts de déploiement appropriés. Nous avons donné des exemples détaillés de ces scripts, pour la fragmentation horizontale, la gestion du buffer et l'ordonnancement des requêtes. Toutefois, nous n'avons pas finalisé l'automatisation de la génération des scripts. Pour nos études expérimentales le long des chapitres précédents, nous avons généré ces scripts manuellement. Pour les travaux futurs, nous envisageons de finaliser l'implémentation des générateurs de script, permettant de déployer les recommandations automatiquement sur le système de base de données.

Chapitre 7

Conclusion générale et Perspectives

Sommaire

1 Conclusion

Dans cette thèse nous nous sommes intéressés au problème de la conception physique, afin d'améliorer les performances des systèmes de bases de données, et plus particulièrement les entrepôts de données. Les principales contributions de notre travail sont les suivantes :

Nouvelle vision du problème de la conception physique : Lors de notre étude de l'état de l'art sur la conception physique, un constat important s'est imposé à nous : les approches existantes d'optimisation des requêtes n'offrent pas une performance satisfaisante. Cela est dû au fait que ces approches se basent sur une formalisation classique de la conception physique. Cette formalisation néglige d'expliciter un certain nombre de paramètres intervenant dans chaque structure d'optimisation. Par conséquent, ces paramètres ne sont pas pris en compte lors de la conception physique. Nous distinguons deux classes de paramètres : les paramètres explicites et les paramètres implicites. La première classe est directement liée au problème de conception physique. Elle comprend généralement une charge de requêtes à optimiser et un schéma de BD sur lequel cette charge est définie. Les paramètres de la deuxième classe sont utilisés par les modèles de coût qui quantifient la qualité des structures d'optimisation sélectionnées. On peut citer ainsi la nature du support, le contenu du buffer ou l'ordre des requêtes. Or, en arrivant à la phase d'évaluation de la solution par modèle de coût, l'approche de résolution est déjà définie, et indépendamment de ces paramètres. À la lumière de ce constat, nous avons déduit que les paramètres implicites sont à l'origine des lacunes des approches proposées pour l'optimisation dans les BD.

Dans cette optique, nous avons commencé par une explicitation des paramètres en proposant un nouveau modèle pour la conception physique. Ce modèle lève l'ambiguïté liée aux composantes des systèmes de BD concernées par l'optimisation, en exhibant les quatre principales composantes : les requêtes, les données, les supports de stockage et les structures d'optimisation.

En étudiant ces composantes, nous avons constaté l'omniprésence de l'interaction qui se présente de deux manières : l'interaction inter-composantes, et l'interaction intra-composante. Dans la première, un élément d'une composante interagit avec un élément d'une autre. Dans la seconde, les éléments de la même composante interagissent entre-eux. Afin d'étudier plus en détail chaque type de liaison entre les composantes, nous avons choisi des cas d'étude avec un ensemble de structures d'optimisation qui touche aux différentes composantes.

Dans les contributions suivantes, nous allons détailler les cas d'étude, suivis de l'outil proposé pour l'aide à la conception physique.

Étude de l'interaction entre les composantes *support* et les *requêtes* : L'interaction entre les requêtes et les supports a été mise en évidence dans ce travail par une étude du problème joint de la gestion du buffer et de l'ordonnancement des requêtes. La projection des deux problèmes sur le modèle explicite a révélé la similarité et la complémentarité des deux techniques. Dans notre étude, nous avons exposé et critiqué les différents scenarii pour la combinaison des deux problèmes. Nous avons également prouvé que la combinaison est liée à un problème $\mathcal{NP}$-complet au sens fort. L'étude des approches proposées montre que l'intégration de l'interaction des requêtes augmente à la fois la performance et la réactivité des algorithmes. Cette propriété nous a permis de faire face au dilemme réactivité-efficacité des algorithmes existants : les approches dirigées par l'affinité, et les méta-heuristiques dirigées par modèle de coût. L'algorithme Queen-Bee tire profit de cette propriété pour fournir un bon compromis. L'algorithme

Queen-Bee doit son efficacité à l'exploitation intelligente de l'interaction et au modèle de coût utilisé.

L'impact de l'interaction sur la fragmentation horizontale : La fragmentation horizontale est considérée comme une approche logique et physique de conception de BD. En considérant le problème de fragmentation horizontale, nous avons constaté que l'interaction entre les requêtes est négligée dans toutes les approches existantes. Nous avons ainsi proposé un algorithme dirigé par l'interaction permettant de fournir un schéma de fragmentation horizontal en un temps réduit. En plus de l'impact des requêtes, nous avons considéré l'impact de la sélection multiple sur ce problème. Pour cela, nous avons adopté des techniques complémentaires permettant de considérer les aspects ignorés par la fragmentation, à savoir la gestion du buffer et l'ordonnancement des requêtes (GBOR). La sélection multiple augmente encore les performances par rapport aux techniques isolées. Toutefois, la sélection multiple apporte des difficultés supplémentaires et des nouveaux défis au mode de combinaison (indépendant, conjoint ou séquentiel), à l'évolution de l'espace des candidats (objets du buffer) et à l'adaptation de la charge (réécriture des requêtes). Nous avons proposé une démarche permettant de contourner ces difficultés, à l'aide d'une structure de données dynamique et un ensemble d'algorithmes dirigés par l'interaction des requêtes. Ces algorithmes sont respectivement l'algorithme de Requête Élue, et l'algorithme Queen-Bee pour la fragmentation horizontale et la GBOR. La démarche proposée présente une réactivité considérable comparée aux méta-heuristiques dans le cas isolé. Pour le cas multiple, la résolution par méta-heuristiques devient infaisable à cause de la lourdeur des processus d'optimisation imbriqués. La démarche dirigée par l'interaction que nous proposons s'avère plus adaptée et offre des gains considérables en performance pour la charge globale.

Considération de l'aspect dynamique de la conception physique : Un système de base de données est destiné à évoluer en permanence, aussi bien au niveau des données, des requêtes, des tailles de structures d'optimisation que des supports de stockage. Cet aspect dynamique impose que ces solutions existantes puissent s'adapter selon l'état du système. La réactivité joue alors un rôle déterminant dans la performance. Dans le cas d'une optimisation en-ligne, où les requêtes proviennent de plusieurs utilisateurs au même temps, la gestion du buffer et l'ordonnancement des requêtes sont affectés par cette évolution de la charge, et à la réactivité du processus également. Nous avons donc analysé le problème de GBOR en-ligne pour mettre en évidence ses particularités, et pour en proposer une adaptation de l'algorithme Queen-Bee au mode en-ligne. Les politiques de gestion comme LRU sont souvent employées, car la charge ne peut être toujours connues à priori, ou à cause du manque de techniques convenables pour la gestion en-ligne. L'étude expérimentale montre que l'algorithme offre une meilleure efficacité au système comparé à LRU.

Développement d'un outil d'aide à la conception physique : Pour effectuer les différentes simulations, nous avons développé un outil d'aide à la conception physique appelé : *Physical Design Advisor*. L'outil implémente nos algorithmes, et offre un ensemble de techniques d'optimisation pour une sélection simple ou multiple, en mode en-ligne ou hors-ligne.

Nous avons détaillé les fonctionnalités et les modules de notre simulateur. L'outil manque d'un générateur automatique des scripts de déploiement. Toutefois, nous avons prouvé son efficacité en comparant les résultats de simulation avec les résultats de validation réelle sous Oracle11g.

2 Perspectives

Les travaux initiés dans cette thèse peuvent se poursuivre dans de nombreuses directions. Nous esquissons ici quelques pistes de perspectives à court et à long terme.

Étude de Robustesse : L'une des perspectives que nous sommes en train d'étudier est la robustesse de nos algorithmes afin d'évaluer leur passage à l'échelle. Cela peut être réalisé par la considération d'une base de données ayant un nombre important de tables de volume important (plusieurs Téraoctets). L'augmentation du nombre de tables augmentent le nombre de nœuds candidats de notre algorithme Queen Bee et le volume peut impacter notre méthode d'allocation buffer.

Extension du modèle explicite : Dans ce travail, nous avons étudié trois techniques d'optimisation parmi une large panoplie. Les structures d'optimisation les plus souvent utilisées, en plus de la fragmentation horizontale, sont les index et les vues matérialisées. Nous proposons d'étendre notre étude pour intégrer ces deux structures afin d'analyser l'impact de l'interaction sur celles-ci. Nous étudions également l'adaptabilité de nos algorithmes sur ces deux structures en mode isolé ou multiple.

Par ailleurs, nous proposons d'étendre notre modèle pour intégrer de nouveaux éléments et composantes que nous avons écartés dans nos hypothèses de travail. Notre modèle ne tient pas compte de certains éléments tels que : la priorité des requêtes, leur fréquence, le modèle de stockage (R-store et C-store [13]) ainsi que la nature de l'architecture de déploiement. En effet, dans ce travail, les requêtes sont supposées sans priorité lors de leur ordonnancement. Nous allons donc relâcher cette hypothèse afin de répondre au mieux aux exigences des concepteurs. De plus, le seul modèle de données que nous avons considéré est le R-store, ou le stockage par tuple. Or, un nouveau modèle de stockage a été proposé comme une alternative à ce dernier, qui est le modèle C-Store, ou le stockage par colonne. Nous proposons d'étendre notre modèle pour couvrir les deux types de modèles pour plus de généricité.

Préparation de la charge de requêtes : L'entrée principale du problème, qui est la charge de requête, est représentée par un MVPP figé, à cause de la complexité du problème de sélection des plans ($\mathcal{NP}$-complet). Dans des travaux futurs, nous chercherons à améliorer les entrées en intégrant les règles employées par les optimiseurs classiques (*Rule Based Approaches*). En effet, lors de notre étude expérimentale sur l'algorithme Queen-Bee, nous avons intégré la descente des projections (*Push-Down Projection*), ce qui permet de réduire considérablement la taille des résultats intermédiaires. L'intégration d'autres règles est la poursuite naturelle de ce travail.

D'autres part, la charge de requêtes n'est pas toujours connues par l'administrateur à priori, notamment lors de l'optimisation en-ligne. Dans ce cas, les approches existantes cèdent leur place aux techniques basées sur les modèles probabilistes et les méthodes Markoviennes [108, 107]. Il serait donc intéressant d'intégrer ces approches pour prédire les entrées, dans un environnement où l'administrateur n'a pas une connaissance préalable de celles-ci.

Établissement des relations de dépendances dans la conception physique : En plus des dépendances établies par Zilio *et al.*, nous envisageons de formaliser les dépendances entre les autres éléments des composantes du système de base de données (requêtes, données, supports). Ceci permettra de faciliter le choix de la solution par l'administrateur, en pilotant le processus de sélection des structures d'optimisation et de leur combinaison par les dépendances. Nous proposons également de raffiner l'algorithme

13. http://db.csail.mit.edu/projects/cstore/

Queen-Bee par un élagage des candidats guidé par les dépendances entre les nœuds du MVPP.

Bien que notre approche ajoute un degré de complexité lié à la considération des différentes composantes, cette complexité peut être contournée par l'exploitation des dépendances afin d'aider le concepteur à choisir ses structures d'optimisation.

Amélioration de l'outil d'aide à la conception physique : D'un point de vue technique, nous proposons d'améliorer notre simulateur en automatisant le processus de génération des scripts pour le déploiement des recommandations sur un SGBD quelconque. Nous proposons également d'intégrer les nouvelles structures d'optimisation comme les index et les vues matérialisées pour fournir une sélection multiple plus large.

Peut-on faire abstraction du SGBD ? L'ensemble des composantes que nous avons mis en évidence permet de modéliser le système de BD avec les différents paramètres requis. À long terme, et après avoir (1) testé la robustesse de nos algorithmes à grande échelle, (2) intégré d'autres structures d'optimisations et (3) étendu le modèle explicite, nous allons étudier la faisabilité de substituer un SGBD par le modèle explicite pour effectuer les différents traitements sur les données. Chaque composante sera perçue comme un service. Le modèle doit permettre d'inhiber certaines composantes à chaque fois que le concepteur veut faire abstraction de celles-ci. Ainsi, les recommandations peuvent fournir une sélection de structures d'optimisation qui couvre uniquement les composantes voulues afin de réduire la complexité du traitement de la totalité des composantes et de répondre au mieux aux exigences de l'administrateur.

3 Épilogue

Au fil des quatre dernières décennies, la formalisation des problèmes d'optimisation est devenue le prologue d'un "*rituel*" pour la conception physique des bases de données. Initialement, elle avait pour but de simplifier l'analyse des problèmes $\mathcal{NP}$-complets. Aujourd'hui avec l'évolution des systèmes de bases de données, la formalisation classique passe à côté de certains aspects considérés comme détails, mais qui s'avèrent déterminantes pour une *bonne* conception physique.

Ces aspects que nous avons mis en évidence, dévoilent l'omniprésence de l'interaction entre les composantes d'un système de base de données. Certes, la prise en compte de ces caractéristiques nécessite un effort d'adaptation des approches existantes, mais leur exploitation améliore à la fois l'efficacité et la réactivité des approches d'optimisation.

La technologie des bases de données n'a donc toujours pas donné le meilleur d'elle-même.

Bibliographie

[1] Cassandra project. *Available at http://cassandra.apache.org/*.

[2] Corrad project. *Available at http://database.cs.brown.edu/projects/coradd/*.

[3] Couchdb. *Available at http://couchdb.apache.org/*.

[4] Hbase appache project. *Available at http://hbase.apache.org/*.

[5] Mongodb. *Available at http://www.postgresql.org/docs/8.1/static/ddl-partitioning.html*.

[6] Oracle Database Concepts 11g Release. Memory architecture. *Available at* `http://docs.oracle.com/cd/B28359_01/server.111/b28318/memory.htm#i10221`, 2011.

[7] K. Aberer and G. Fisher. Semantic query optimization for methods in object-oriented database systems. *In Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 70–79, March 1995.

[8] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active xml: Peer-to-peer data and web services integration. *Proceedings of the International Conference on Very Large Databases*, pages 1087–1090, 2002.

[9] F. N. Afrati, M. Balazinska, A. D. Sarma, B. Howe, S. Salihoglu, and J. D. Ullman. Designing good algorithms for mapreduce and beyond. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 26:1–26:2, New York, NY, USA, 2012. ACM.

[10] A. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. Technical report research, IBM, 1997.

[11] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Interaction-aware scheduling of report-generation workloads. *The VLDB Journal*, 20(4):589–615, 2011.

[12] M. Atkinson, F. Bancilhon, D. J. DeWitt, K. Dettrich, D. Maier, and S. Zdonik. The object database system manifesto. *In Proceeding of the first International Conference on Deductive, Object-Oriented Databases*, pages 223–240, December 1989.

[13] C. Bachman. The programmer as navigator. *ACM*, 16(1), November 1973.

[14] M. Balazinska, B. Howe, and D. Suciu. Data markets in the cloud: An opportunity for the database community. *PVLDB*, 4(12):1482–1485, 2011.

[15] J. Banerjee, W. Kim, H. J. Kim, and H. F. Korth. Semantics and implementation of schema evolution in object-oriented databases. *In Proceedings of the ACM SIGMOD International Conference on Management of Data(SIGMOD'87)*, pages 311–322, May 1987.

[16] J. Banerjee, W. Kim, and K. C. Kim. Queries in object oriented databases. *In*

Proceedings of the IEEE Data Engineering Conference(ICDE'88), pages 31–38, February 1988.

[17] E. Baralis, S. Paraboschi, and E Teniente. Materialized view selection in a multidimensional database. *Proceedings of the International Conference on Very Large Databases*, pages 156–165, August 1997.

[18] L. Bauer and W. Lehner. The cube-query-languages (cql) for multidimensional statistical and scientific database systems. *In Proceedings of the Fifth International Conference on Database Systems for Advanced Applications(DASFAA'97)*, pages 263–272, April 1997.

[19] J. Beauquier and B. Bérard. *Systèmes d'exploitation: concepts et algorithmes*. Collection Informatique. McGraw-Hill Ryerson, Limited, 1990.

[20] L. Bellatreche. Logical and physical design in data warehousing environments. *In proceedings of International Conference on Extending Database Technology (EDBT) PhD Workshop*, March 2000.

[21] L. Bellatreche. Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un entrepôts de données. Phd. thesis, Blaise Pascal University, France, December 2000.

[22] L. Bellatreche. Bitmap join indexes vs. data partitioning. In *Database Technologies: Concepts, Methodologies, Tools, and Applications*, pages 2292–2300. IGI Global, 2009.

[23] L. Bellatreche. Contributions à la conception et l'exploitation de systèmes d'intégration de données. Mémoire d'habilitation à diriger la recherche, Université de Poitiers, 2009.

[24] L. Bellatreche and S. Benkrid. A joint design approach of partitioning and allocation in parallel data warehouses. In *11th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'09)*, pages 99–110, 2009.

[25] L. Bellatreche, S. Benkrid, A. Ghazal, A. Crolotte, and A. Cuzzocrea. Verification of partitioning & allocation techniques on teradata dbms. In *The 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'2011)*, pages 158–169, Melbourne, Australia, October 2011.

[26] L. Bellatreche, R. Bouchakri, A. Cuzzocrea, and S. Maabout. Horizontal partitioning of very-large data warehouses under dynamically-changing query workloads via incremental algorithms. In *SAC*, pages 208–210, 2013.

[27] L. Bellatreche, K. Boukhalfa, and H. I. Abdalla. Saga: A combination of genetic and simulated annealing algorithms for physical data warehouse design. In *23rd British National Conference on Databases (BNCOD'06)*, pages 212–219, July 2006.

[28] L. Bellatreche, K. Boukhalfa, and P. Richard. Horizontal partitioning in data warehouses: Hardness study, heuristics and oracle validation. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2008)*, pages 87–96, 2008.

[29] L. Bellatreche, K. Boukhalfa, and P. Richard. Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining*, 5(4):1–23, 2009.

[30] L. Bellatreche, S. Breß, A. Kerkad, A. Boukorca, and C. Salmi. In IEEE, editor, *Proceedings of the 31th IEEE International Convention MIPRO (Mipro2013)*, 2013.

[31] L. Bellatreche, A. Cuzzocrea, and S. Ben-

krid. Query optimization over parallel relational data warehouses in distributed environments by simultaneous fragmentation and allocation. In *The 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, pages 124–135, Busan Korea, May 2010.

[32] L. Bellatreche, K. Karlapalem, and Q. Li. Complex methods and class allocation in distributed object-oriented database systems. In *OOIS*, pages 239–256, 1998.

[33] L. Bellatreche, K. Karlapalem, and M. Schneider. Interaction between index and view selection in data warehousing environments. *Submitted to Information Systems Journal*, 2000.

[34] L. Bellatreche, K. Karlapalem, and M. Schneider. On efficient storage space distribution among materialized views and indices in data warehousing environments. In *Proceedings of the International Conference on Information and Knowledge Management (ACM CIKM'2000)*, pages 397–404, 2000.

[35] L. Bellatreche, K. Karlapalem, and M. Schneider. On efficient storage space distribution among materialized views and indices in data warehousing environments. *Submitted to a Conference*, 2000.

[36] L. Bellatreche, K. Karlapalem, and A. Simonet. Horizontal class partitioning in object-oriented databases. *In 8th International Conference on Database and Expert Systems Applications (DEXA'97), Toulouse, Lecture Notes in Computer Science 1308*, pages 58–67, September 1997.

[37] 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.

[38] L. Bellatreche, A. Kerkad, S. Breß, and D. Geniet. Roupar: Routinely and mixed query-driven approach for data partitioning. In *OTM Conferences*, pages 309–326, 2013.

[39] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. Selection and pruning algorithms for bitmap index selection problem using data mining. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2007)*, pages 221–230, September 2007.

[40] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. A data mining approach for selecting bitmap join indices. *Journal of Computing Science and Engineering*, 2(1):206–223, January 2008.

[41] L. Bellatreche, G. Pierra, D. Nguyen-Xuan, and H. Dehainsala. An automated information integration technique using an ontology-based database approach. *In Proceedings of 10th ISPE International Conference on Concurrent Engineering: Research and Applications (ce'03) : Special Track on Data Integration in Engineering*, pages 217–224, July 2003.

[42] L. Bellatreche, C. Salmi, S. Breß, A. Kerkad, A. Boukhorca, and J. Boukhobza. How to exploit the device diversity and database interaction to propose a generic cost model. In *IDEAS, 17th International Database Engineering and Applications Symposium, Barcelona, Spain*, pages 142–147, 2013.

[43] L. Bellatreche, M. Schneider, H. Lorinquer, and M. Mohania. Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2004)*, pages 15–25, September 2004.

[44] L. Bellatreche, A. Simonet, and M. Simonet. An algorithm for vertical fragmentation in distributed object database systems

with complex attributes and methods. *In International Workshop on Database and Expert Systems Applications (DEXA'96), Zurich*, pages 15–21, September 1996.

[45] L. Bellatreche and K. Y. Woameno. Dimension table driven approach to referential partition relational data warehouses. In *to appear in ACM Twelfth International Workshop on Data Warehousing and OLAP (DOLAP09)*, 2009.

[46] S. Benkrid, L. Bellatreche, and H. Drias. A combined selection of fragmentation and allocation schemes in parallel data warehouses. In *4th International Workshop on Data Management in Global Data Repositories (GREP'08)*, pages 370–374, 2008.

[47] P. A. Bernstein and D-M. W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, January 1981.

[48] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-oriented query languages: The notion and the issues. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):223–237, 1992.

[49] R. Blankinship, A. R. Hevner, and S. B Yao. An iterative method for distributed database design. *17th International Conference on Very Large Data Bases, VLDB'91*, pages 389–400, September 1991.

[50] J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, 2013.

[51] R. Bouchakri and L. Bellatreche. On simplifying integrated physical database design. In *15th East European Conference on Advances in Databases and Information Systems (ADBIS'2011)*, pages 333–346, September 2011.

[52] K. Boukhalfa. Administration et tuning des entrepôts de données. Ph.d. thesis, Poitiers University, July 2009.

[53] K. Boukhalfa, L. Bellatreche, and Z. Alimazighi. Hp&bji: A combined selection of data partitioning and join indexes for improving olap performance. *Annals of Information Systems, Special Issue on new trends in data warehousing and data analysis, Springer*, 3:179–2001, November 2008.

[54] K. Boukhalfa, L. Bellatreche, and P. Richard. Fragmentation primaire et dérivée: étude de complexité, algorithmes de sélection et validation sous oracle10g. Techreport http://www.lisi.ensma.fr/members/bellatreche, LISI/ENSMA, 2008.

[55] A. Boukorca, L. Bellatreche, S-A. Benali Senouci, and Z. Faget. Sonic: Scalable multi-query optimization through integrated circuits. In *DEXA (1)*, pages 278–292, 2013.

[56] M. Bouzeghoub. Using semantics to improve schema and data integration. *Proceedings of the International Workshop on Information Integration on the Web (Invited Talk)*, April 2001.

[57] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. The haloop approach to large-scale iterative data analysis. *The VLDB Journal*, 21:169–190, 2012.

[58] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: a maximal frequent itemset algorithm for transactional databases. pages 443–452, April 2001.

[59] G. Canahuate, T. Apaydin, A. Sacan, and H. Ferhatosmanoglu. Secondary bitmap indexes with vertical and horizontal partitioning. In *12th International Conference on Extending Database Technology (EDBT'09)*, pages 600–611, 2009.

[60] S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. In *SIGMOD*, pages 128–136. ACM, 1982.

[61] S. Ceri, B. Pernici, and G. Wiederhold. Optimization problems and solution methods in

the design of data distribution. *Information Systems*, 14(3):261–272, 1989.

[62] S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S. B. Navathe. An objective function for vertically partitioning relations in distributed databases and its analysis. *In Distributed and Parallel Databases Journal*, 2(2):183–207, April 1994.

[63] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *To appear in: OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Novermber 2006.

[64] J. Chang. A heuristic approach to distributed query processing. *Proceedings of the International Conference on Very Large Databases*, pages 54–61, September 1982.

[65] S. Chaudhuri. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1313–1323, November 2004.

[66] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *Sigmod Record*, 26(1):65–74, March 1997.

[67] S. Chaudhuri and V. Narasayya. Self-tuning database systems: a decade of progress. In *VLDB*, pages 3–14. VLDB Endowment, 2007.

[68] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, Mars 1994.

[69] C. Chee-Yong. Indexing techniques in decision support systems. Ph.d. thesis, University of Wisconsin - Madison, 1999.

[70] F. C. F. Chen and M. H. Dunham. Common subexpression processing in multiple-query processing. *Knowledge and Data Engineering, IEEE Transactions on*, 10:493–499, 1998.

[71] P. Chen. The entity-relationship model toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[72] O. Chipara, C. Lu, and G-C. Roman. Real-time query scheduling for wireless sensor networks. In *RTSS*, pages 389–399, 2007.

[73] S. Choenni, H. M. Blanken, and T. Chang. On the selection of secondary indices in relational databases. *Data Knowledge Engineering*, 11(3):207–238, 1993.

[74] H-T. Chou and D. J. DeWitt. An evaluation of buffer management strategies for relational database systems. In *VLDB*, pages 127–141, 1985.

[75] M. Chrobak, G. J. Woeginger, K. Makino, and H. Xu. Caching is hard - even in the fault model. *Algorithmica*, 63(4):781–794, August 2012.

[76] W. W. Chu. Optimal file allocation in multiple computer system. *IEEE Transactions on Computers*, C.18(10):885–889, October 1969.

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

[78] E. F. Codd. Extending the relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):394–434, 1979.

[79] D. Comer. The dificulty of optimum index selection. *ACM Transactions on Database Systems (TODS)*, 3(4):440–445, march 1978.

[80] D. W. Cornell and P. S. Yu. A vertical partitioning algorithm for relational databases. *Proceedings of the Third In-*

ternational Conference on Data Engineering(ICDE'87), pages 30–35, 1987.

[81] D. W. Cornell and P. S. Yu. Integration of buffer management and query optimization in relational database environment. In *VLDB*, pages 247–255, 1989.

[82] Oracle Corp. Oracle warehouse: Unleash the power of information. *White Paper, http://www.oracle.com/tools/datawarehouse*, November 1998.

[83] Informix Corporation. Data warehousing for the telecommunications industry. *White Paper, http://www.informix.com*, 1998.

[84] Microsoft Corporation. Partitioned tables and indexes in sql server 2005. *Retrieved from http://msdn.microsoft.com/en-us/library/ ms345146.aspx*, 2005.

[85] A. Cosar, E-P. Lim, and J. Srivastava. Multiple query optimization with depth-first branch-and-bound and dynamic query ordering. In *CIKM*, pages 433–438, 1993.

[86] S. Dimov D. Pham and C. Nguyen. An incremental k-means algorithm. *Journal of Mechanical Engineering Science*, 218(7):783–795, 2004.

[87] A. Datta, K. Ramamritham, and H. Thomas. Curio: A novel solution for efficient storage and indexing in data warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 730–733, September 1999.

[88] P. J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, 1968.

[89] W. Effelsberg and T. Härder. Principles of database buffer management. *ACM Trans. Database Syst.*, 9(4):560–595, 1984.

[90] W. Effelsberg and M. E. S. Loomis. Logical, internal, and physical reference behavior in codasyl database systems. *ACM Trans. Database Syst.*, 9(2):187–213, 1984.

[91] 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.

[92] C. I. Ezeife and K. Barker. A comprehensive approach to horizontal class fragmentation in distributed object based system. *International Journal of Distributed and Parallel Databases*, 3(3):247–272, 1995.

[93] C. Faloutsos, R. Ng, and T. Sellis. Flexible and adaptable buffer management techniques for database management systems. *IEEE Transactions on Computers*, 44:546–560, 1995.

[94] S. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Transactions on Database Systems*, 13(1):91–128, March 1988.

[95] J-M. Franco. *Le Data Warehouse, Le Data Mining*. Eyrolles, 1997.

[96] M. R. Frank, E. Omiecinski, and S. B. Navathe. Adaptive and automated index selection in rdbms. In *3rd International Conference on Extending Database Technology (EDBT'92)*, pages 277–292, 1992.

[97] 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.

[98] 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.

[99] P. Furtado. Experimental evidence on partitioning in parallel data warehouses. In *DOLAP*, pages 23–30, 2004.

[100] M. Gao, K. Liu, and Z. Wu. Personalisation in web computing and informatics: Theories, techniques, applications, and future research. *Information Systems Frontiers*, 12:607–629, 2010.

[101] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[102] G. Gardarin, J.-R. Gruser, and Z.-H. Tang. A cost model for clustered object-oriented databases. *VLDB*, pages 323–334, 1995.

[103] G. Gardarin and Gardarin O. *Le Client-Serveur*. Edition Eyrolles, 1996.

[104] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[105] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Journal of Distributed Computing, Special Issue for the Twenty Years of Distributed Computing Research*, 2003.

[106] James N. Gray. Database systems: A textbook case of research paying off. Technical report, 1995.

[107] J-Y. Greff, L. Idoumghar, and R. Schott. Application of markov decision processes to the frequency assignment problem. *Journal on Applied Artificial Intelligence*, 18(8):761–773, 2004.

[108] C. M. Grinstead and J. L. Snell. *Introduction to Probability*. American Math. Society, Second Edition, 1997.

[109] Codasyl Database Task Group. Dbtg april 71 report. *ACM*, 1971.

[110] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2):3–18, June 1995.

[111] A. Gupta, S. Sudarshan, and S. Viswanathan. Query scheduling in multi query optimization. In *IDEAS*, pages 11–19, 2001.

[112] P. A. V. Hall. Common subexpression identification in general algebraic systems. *Technical Rep. UKSC 0060, IBM United Kingdom Scientific Centre*, 1974.

[113] M. Hammer and B. Niamir. A heuristic approach to attribute partitioning. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 93–101, 1979.

[114] J. Heflin. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment.* PhD thesis, University of Maryland, College Park., 2001.

[115] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[116] W. H. Inmon. *Building the Data Warehouse*. John Wiley, 1992.

[117] W. H. Inmon. A history of database evolution. *Retrieved from http://www.b-eye-network.com/newsletters/inmon/1589*, 2005.

[118] M. Jarke. Common subexpression isolation in multiple query optimization. In *Query Processing in Database Systems*, pages 191–205. Springer, 1985.

[119] T. Johnson and D. Shasha. Some approaches to index design for cube forests. *IEEE Data Engineering Bulletin*, 20(1):27–35, March 1997.

[120] S. J. Kaplan. Appropriate responses to inappropriate questions. In *Formal Aspects of Language and Discourse*, pages 127–144. Cambridge University Press, 1980.

[121] K. Karlapalem. Redesign of distributed relational databases. Phd. thesis, Georgia Institute of Technology, November 1996.

[122] K. Karlapalem, S. B. Navathe, and M. Ammar. Optimal redesign policies to support

dynamic processing of applications on a distributed database system. *Information Systems*, 21(4):353–367, 1996.

[123] A. Kerkad, L. Bellatreche, and D. Geniet. Exploitation de l'interaction des requêtes olap pour la gestion de cache et l'ordonnancement de traitements. In *EDA*, pages 154–163, 2012.

[124] A. Kerkad, L. Bellatreche, and D. Geniet. Queen-bee: Query interaction-aware for buffer allocation and scheduling problem. In *Data Warehousing and Knowledge Discovery*, pages 156–167, 2012.

[125] A. Kerkad, L. Bellatreche, and D. Geniet. Simultaneous resolution of buffer allocation and query scheduling problems. In *Proceedings of the 6th International Conference on Complex, Intelligent, and Software Intensive Systems(CISIS)*, pages 122–129, 2012.

[126] A. Kerkad, L. Bellatreche, and D. Geniet. Database technology: A world of interaction. In *DEXA (1)*, pages 454–461, 2013.

[127] A. Kerkad, L. Bellatreche, and D. Geniet. La fragmentation horizontale revisitée: Prise en compte de l'interaction de requêtes. In *RNTI-B-9*, pages 124–140, 2013.

[128] A. Kerkad, L. Bellatreche, P. Richard, C. Ordonez, and D. Geniet. A query beehive algorithm for data warehouse buffer management and query scheduling. *International Journal of Data Warehousing and Mining (IJDWM)*, to appear on 2014.

[129] W. Kim. A model of queries for object-oriented databases. *In Proceedings of the 15th. International Conference on Very Large Databases (VLDB'89)*, pages 423–432, August 1989.

[130] W. Kim, E. Bertino, and J. F Garza. Composite objects revisited. *ACM SIGMOD*, pages 337–347, 1989.

[131] R. Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, 1996.

[132] R. Kimball. A dimensional modeling manifesto. *DBMS Magazine*, August 1997.

[133] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Correlation maps: a compressed access method for exploiting soft functional dependencies. *Proc. VLDB Endow.*, 2(1):1222–1233, August 2009.

[134] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Coradd: correlation aware database designer for materialized views and indexes. *Proc. VLDB Endow.*, 3:1103–1113, September 2010.

[135] W. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. *Proceedings of the International Conference on Data Engineering (ICDE)*, 1997.

[136] W. Le, A. Kementsietsidis, S. Duan, and F. Li. Scalable multi-query optimization for sparql. In *ICDE*, pages 666–677. IEEE, 2012.

[137] C. Lecluse, P. Richard, and F. Velez. o_2, an object-oriented data model. *ACM SIGMOD Conference*, 1988.

[138] W. Lehner, T. Ruf, and M. Teschke. Crossdb: A feature-extended multidimensional data model for statistical and scientific databases. *In the 5th International Conference on Information and Knowledge Management (CIKM'96)*, pages 253–260, 1996.

[139] R. Li, S. Wang, and K. C-C. Chang. Multiple location profiling for users and relationships from social network and content. *Proc. VLDB Endow.*, 5:1603–1614, 2012.

[140] S. J. Lim and Y. K. Ng. A formal approach for horizontal fragmentation in distributed deductive database design. *in the 7th International Conferences on Database and Expert Systems Applications (DEXA'96), Lecture Notes in Computer Science 1134, Zurich*, pages 234–243, September 1996.

[141] D. A. B. Lindberg, J. C. Toole, P. R. Young, J. S. Cavallini, L. B. Holcomb, R. J. Linn Jr,

G. R. Cotter, H. M. Wood, J. H. Novak, and A. T. Poliakoff. The high performance computing and communication (hpcc) program: technologies for the national information infrastructure (panel). In *SC*, page 279, 1994.

[142] M. G. Lohman, D. Daniels, L. M. Haas, R. Kistler, and P. G. Selinger. Optimization of nested queries in a distributed relational database. *Proceedings of the International Conference on Very Large Databases*, pages 403–415, August 1984.

[143] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 621–630, New York, NY, USA, 2012. ACM.

[144] H. Mahboubi and J. Darmont. Data mining-based fragmentation of xml data warehouses. In *ACM 11th International Workshop on Data Warehousing and OLAP (DOLAP'08)*, pages 9–16, 2008.

[145] H. Mahboubi and J. Darmont. Enhancing xml data warehouse query performance by fragmentation. In *SAC*, pages 1555–1562. ACM, 2009.

[146] H. Märtens, E. Rahm, and T. Stöhr. Dynamic query scheduling in parallel data warehouses. In *Euro-Par*, pages 321–331, 2002.

[147] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[148] P. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *ICDE*, pages 227–238, 2003.

[149] M. Mohania, S. Konomi, and Y. Kambayashi. Incremental maintenance of materialized views. *In the 8th International Conferences on Database and Expert Systems Applications (DEXA'97), Lecture Notes in Computer Science 1308*, pages 551–560, September 1997.

[150] M. Mohania, S. Samtani, J. F. Roddick, and Y. Kambayashi. Advances and research directions in data warehousing technology. *To appear in the Australian Journal of Information Systems*, 2000.

[151] 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.

[152] S. B. Navathe and M. Ra. Vertical partitioning for database design: a graphical algorithm. In *SIGMOD*, pages 440–450. ACM, 1989.

[153] S.B. Navathe, K. Karlapalem, and M. Ra. A mixed partitioning methodology for distributed database design. *Journal of Computer and Software Engineering*, 3(4):395–426, 1995.

[154] A. Y. Noaman and K. Barker. A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse. *In the 8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 154–161, November 1999.

[155] 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, Berkeley, CA, USA, 2006. USENIX Association.

[156] P. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24(3):8–11, September 1995.

[157] P. O'Neil, B. O'Neil, and X. Chen. Star schema benchmark. *Available at*

http://www.cs.umb.edu/~poneil/StarSchemaB.PDF, 2009.

[158] Y. Ou, T. Härder, and P. Jin. Cfdc: A flash-aware buffer management algorithm for database systems. In *ADBIS*, pages 435–449, 2010.

[159] M. T. Özsu, Dayal, and P. Valduriez. An introduction to distributed object management. *International Workshop on Distributed Object Management*, pages 1–24, August 1992.

[160] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[161] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems : Second Edition*. Prentice Hall, 1999.

[162] S. Papadomanolakis and A. Ailamaki. Autopart: Automating schema design for large scientific databases using data partitioning. In *SSDBM*, pages 383–392. IEEE, 2004.

[163] T. Phan and W-S. Li. Dynamic materialization of query views for data warehouse workloads. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 436–445, 2008.

[164] D. J. Rosenkrantz and H. B. Hunt III. Processing conjunctive predicates and queries. In *Proceedings of the sixth international conference on Very Large Data Bases - Volume 6*, pages 64–72. VLDB Endowment, 1980.

[165] P. Roy, K. Ramamritham, S. Seshadri, P. Shenoy, and S. Sudarshan. Don't trash your intermediate results, cache 'em. *CoRR*.

[166] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[167] G. M. Sacco and M. Schkolnick. A mechanism for managing the buffer pool in a relational database system using the hot set model. In *Proceedings of the 8th International Conference on Very Large Data Bases*, VLDB '82, pages 257–262, San Francisco, CA, USA, 1982. Morgan Kaufmann Publishers Inc.

[168] G. M. Sacco and M. Schkolnick. Buffer management in relational database systems. *ACM Transactions on Database Systems*, 11:473–498, 1986.

[169] A. Sanjay, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in microsoft sql server. In *Proceedings of the International Conference on Very Large Databases*, pages 496–505, September 2000.

[170] A. Sanjay, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, 2004.

[171] S. Sarawagi. Indexing olap data. *IEEE Data Engineering Bulletin*, 20(1):36–43, March 1997.

[172] P. Scheuermann, J. Shim, and R. Vingralek. Watchman : A data warehouse intelligent cache manager. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 51–62, 1996.

[173] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *SIGMOD Conference*, pages 793–795. ACM, 2006.

[174] A. Segev and J. Park. Maintaining materialized views in distributed databases. *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 262–270, February 1989.

[175] 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.

[176] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, March 1988.

[177] T. Shashank. *Professional NoSQL*. John Wiley & Sons, 2011.

[178] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[179] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: achievements and opportunities. Technical report, New York, NY, USA, 1991.

[180] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database research: Achievements and opportunities into the 21st century. Technical report, Stanford, CA, USA, 1996.

[181] M. Stonebraker. Operating system support for database management. *Commun. ACM*, 24(7):412–418, 1981.

[182] A. J. Storm and C. Garcia-arellano. Adaptive self-tuning memory in db2. In *In Proc. of the 2006 Intl. Conf. on Very Large Data Bases*, 2006.

[183] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic partitioning of shared cache memory. *Journal of Supercomputing*, 28(1):2004, 2002.

[184] K-L. Tan and H. Lu. Workload scheduling for multiple query processing. *Information Processing Letters*, 55(5):251–257, september 1995.

[185] R. W. Taylor and R. L. Franck. Codasyl database management systems. *ACM Computer Surveys*, 8(1), march 1976.

[186] D. Thomas, A. A. Diwan, and S. Sudarshan. Scheduling and caching in multiquery optimization. In *COMAD*, pages 150–153, 2006.

[187] 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.

[188] D. C. Tsichritzis and F. H. Lochvsky. Hierarchical database management. *ACM Computer Surveys*, 8(1), march 1976.

[189] J. D. Ullman. Information integration using logical views. *Proceedings of the International Conference on Database Theory (ICDT), Lecture Notes in Computer Science*, 1186:19–40, January 1997.

[190] P. Upadhyaya, M. Balazinska, and D. Suciu. How to price shared optimizations in the cloud. *PVLDB*, 5:562–573, 2012.

[191] P. Valduriez. Join indices. *ACM Trans. Database Syst.*, 12(2):218–246, 1987.

[192] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *ICDE'00*, pages 101–110, 2000.

[193] Y. Wang, J. C.L. Liu, Du, and J. Hsieh. Video file allocation over disk arrays for video-on-demand. *In Proceedings of the International Conference on Multimedia Computing and Systems(ICMCS'96)*, pages 160–173, June 1996.

[194] P. Wehrle, M. Miquel, and A. Tchounikine. A model for distributing and querying a data warehouse on a computing grid. *11th International Conference on Parallel and Distributed Systems (ICPADS 2005)*, pages 203–209, July 2005.

[195] K-Y. Whang, G. Wiederhold, and D. Sagalowicz. Estimating block accesses in database organizations: a closed noniterative formula. *Commun. ACM*, 26:940–944, 1983.

[196] J. Wu, K-L. Tan, and Y. Zhou. Qos-oriented multi-query scheduling over data streams. In *DASFAA*, pages 215–229, 2009.

[197] K. Wu, E. Otoo, and A. Shoshani. An efficient compression scheme for bitmap indices. In *In Proceedings of the ACM Transactions on Database Systems (TODS)*, 2006.

[198] M-C. Wu and A. P. Buchmann. Encoded bitmap indexing for data warehouses. In *Proceedings of the Fourteenth International Conference on Data Engineering (ICDE'98)*, pages 220–230, 1998.

[199] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the International Conference on Very Large Databases*, pages 136–145, August 1997.

[200] M. Yang and G. Wu. Caching intermediate result of sparql queries. In *WWW (Companion Volume)*, pages 159–160, 2011.

[201] D. C. Zilio, J. Rao, S. Lightstone, G. M Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *Proceedings of the International Conference on Very Large Databases*, pages 1087–1097, August 2004.

[202] E. Ziyati, L. Bellatreche, and D. Aboutajdine. Un algorithme génétique pour la sélection d'un schéma de fragmentation mixte dans les entrepôts de données. *Ateliers des Systèmes décisionnels (ASD06)*, December 2006.

Annexe

Les requêtes suivantes sont des requêtes typiques du banc d'essai Star Schema Benchmark. Dans nos études expérimentales, nous avons construit plusieurs ensembles de requêtes à partir de celles-ci, afin d'effectuer les différentes évaluations.

```
--Q1.1 :
select 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;

--Q1.2 :
select sum(lo_extendedprice*lo_discount) as revenue
 from lineorder, date
 where lo_orderdate = d_datekey
  and d_yearmonthnum = 199401
  and lo_discount between4 and 6
  and lo_quantity between 26 and 35;

--Q1.3 :
select 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;

--Q2.1:
select 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;

--Q2.3 :
```

```
select 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;

--Q3.1 :
select 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;

--Q3.2 :
select 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;

--Q3.3 :
select 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;


--Q3.4 :
select 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;

--Q4.1 :
select d_year, c_nation,  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 (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
    group by d_year, c_nation
    order by d_year, c_nation

--Q4.2 :
select 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

--Q4.3 :
select 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
```

Table des figures

Liste des tableaux

Glossaire

AG : Algorithme Génétique
BD : Base de données
DBM : Dynamic Buffer Manager
ED : Entrepôt de données
EDR : Entrepôt de données Relationnel
FH : Fragmentation Horizontale
FHP : Fragmentation Horizontale Primaire
FHD : Fragmentation Horizontale Dérivée
GB : Gestion du Buffer
GBOR : Gestion du Buffer et Ordonnancement des Requêtes
HC : Hill Climbing
LIAS : Laboratoire d'Informatique et d'Automatique pour les Systèmes
LRU : Least Recently Used
MC : Modèle de Coût
OR : Ordonnancement des Requêtes
OODB : Object-Oriented Databases
PBO : Partitionnement-Bufferisation-Ordonnancement
PE : Planning d'exécution
QB : Queen-Bee
QBE : Queen-Bee En-ligne
QoS : Quality of Service
RE : Requête Élue
REFH : Requête Élue pour la Fragmentation Horizontale
RS : Recuit Simulé
SGBD : Système de Gestion de Base de données
SO : Structure d'Optimisation

L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels.

Présentée par :

Amira KERKAD

Directeurs de Thèse :

Ladjel BELLATRECHE et **Dominique GENIET**

Résumé. La technologie de base de données est un environnement adéquat pour l'interaction. Elle peut concerner plusieurs composantes du SGBD : (a) les données, (b) les requêtes, (c) les techniques d'optimisation et (d) les supports de stockage. Au niveau des données, les corrélations entre les attributs sont très communes dans les données du monde réel, et ont été exploitées pour définir les vues matérialisées et les index. Au niveau requêtes, l'interaction a été massivement étudiée sous le problème d'optimisation multi-requêtes. Les entrepôts de données avec leurs jointures en étoile augmentent le taux d'interaction. L'interaction des requêtes a été employée pour la sélection des techniques d'optimisation comme les index. L'interaction contribue également dans la sélection multiple des techniques d'optimisation comme les vues matérialisées, les index, le partitionnement et le clustering. Dans les études existantes, l'interaction concerne une seule composante. Dans cette thèse, nous considérons l'interaction multi-composante, avec trois techniques d'optimisation, où chacune concerne une composante : l'ordonnancement des requêtes (niveau requêtes), la fragmentation horizontale (niveau données) et la gestion du buffer (niveau support de stockage). L'ordonnancement des requêtes (OR) consiste à définir un ordre d'exécution optimal pour les requêtes pour permettre à quelques requêtes de bénéficier des données pré-calculées. La fragmentation horizontale (FH) divise les instances de chaque relation en sous-ensembles disjoints. La gestion du buffer (GB) consiste à allouer et remplacer les données dans l'espace buffer disponible pour réduire le coût de la charge. Habituellement, ces problèmes sont traités soit de façon isolée ou par paire comme la GB et l'OR. Cependant, ces problèmes sont similaires et complémentaires. Une formalisation profonde pour le scénario hors-ligne et en-ligne des problèmes est fournie et un ensemble d'algorithmes avancés inspirés du comportement naturel des *abeilles* sont proposés. Nos propositions sont validées en utilisant un simulateur et un SGBD réel (Oracle) avec le banc d'essai *star schema benchmark* à grande échelle.

Mots-clés : Bases de données avancées, Entrepôt de données, Optimisation des requêtes, Interaction, Support de stockage, Conception Physique.

Abstract. The database technology is an adequate environment for the interaction. It may concern several components of the DBMS: (a) *the data*, (b) *the queries*, (c) *the optimization techniques* and (d) the *devices*. At the data level, correlations between attributes are extremely common in the real world relational data, and have been exploited to define materialized views and indexes. At the query level, interaction has been massively studied under the problem of multi-query optimization. The data warehouses with their star join queries increase the rate of the interaction. The query interaction has been used for selecting optimization techniques such as indexes. The interaction also contributes in selecting multiple optimization techniques such as materialized views, indexes, data partitioning and the clustering. In existing studies, the interaction concerns only one component. In this thesis, we consider the multi-component interaction, with three optimization techniques, where each one concerns one component: the query scheduling (query level), the horizontal data partitioning (data level) and the buffer management (device level). The query scheduling (QS) consists in defining an optimal order of executing queries to allow some queries to get benefit from already processed data. The horizontal data partitioning (HDP) divides the instances of each relation into disjoint subsets. The buffer management (BM) consists in allocating and replacing data in the buffer pool to lower the cost of queries. Usually, these problems are treated either in isolation or pairwise such as BM and QS. However, these problems are similar and complementary. A deep formalization for off-line and online scenario of these problems is given and advanced algorithms inspired from natural bees behavior are proposed. Our proposal has been validated using a simulator and real DBMS (Oracle) using a large scale of star schema benchmark.

Keywords: Advanced databases, Data warehousing, Query optimization, Interaction, Storage device, Physical Design.