bottom
bottom

UNIVERSITÉ DE BLIDA 1
Faculté des Sciences
Département d'Informatique

THÈSE DE DOCTORAT

Option : Sciences Informatiques et de Données.

PROPOSITION DE STRUCTURE DE DONNÉES SCALABLE POUR L'OPTIMISATION DE RECOMMANDATION DE REQUÊTES

Par
Chaba Mouna Mustapha

devant le jury composé de :

H. Abed	**Professeur, U. de Blida 1**	**Présidente**
N. Boustia	**Professeur, U. de Blida 1**	**Directrice de thèse**
L. Bellatrache	**Professeur, ISAE-ENSMA,Poitiers,France**	**Co-Directeur de thèse**
F. Boumahdi	**Maître de conférences, U. de Blida 1**	**Examinatrice**
K. Boukhalfa	**Professeur, ENSIA, Alger**	**Examinateur**

Blida, le 27 mai 2023

Je dédie mes travaux de thèse :

À mes chers parents : Roukia et Abdelkader

À mes soeurs : Imane, Hayat, Siham, Khadidja et Manel

À Ma chère femme : Nadjah

À Ma belle fille : Manar

. . .

REMERCIEMENTS

"La reconnaissance est la mémoire du cœur."

En premier lieu, je rends louange à Allah le tout Puissant de m'avoir donné le courage, la volonté et de m'avoir éclairé le chemin du savoir ainsi que de m'avoir entouré par des personnes positives qui m'ont beaucoup aidé afin d'accomplir mon travail.

Je tiens à exprimer ma gratitude et mes plus vifs remerciements à ma directrice de thèse Mme Boustia Narhimene, Professeur en informatique à USD-Blida 1, pour son aide, son écoute, sa disponibilité et ses précieux conseils qu'elle m'a prodigué tout au long de ce parcours de recherche. L'autonomie qu'elle m'a laissé, m'ont permis de mener à bien mon travail.

Mes remerciements s'adresseront aussi à ma co-directeur de thèse Mme Ladjel Bellatreche, Professeur en informatique à ENSMA-France, pour toute sa disponibilité, sa patience, son écoute, son esprit critique, enthousiasme et confiance dont il m'a gratifié et ses précieux conseils avisés qui ont nourri mes réflexions tout au long de l'élaboration de mon travail. Qu'il trouve dans ces lignes si courtes l'expression de ma très grande gratitude.

Je tiens à remercier Pr. H.Abed, pour m'avoir fait l'honneur d'accepter de présider mon jury de thèse.

J'exprime toute ma gratitude aux membres du jury Dr. F.Boumahdi et Pr. K. Boukhalfa qui nous feront l'honneur d'évaluer mon travail.

J'exprime toute ma reconnaissance à mes parents Abdelkader et Roukia, à mes soeurs : Imane, Hayat, Siham, Khadidja et Manel. A Ma femme Nadjah et ma petite fille Manar. Grâce à vos encouragements, votre soutien que j'ai pu avancer.

Mes vifs remerciements à mon ami Dr. Riali ishak qui m'a soutenu tout au long de la réalisation de mon projet.

Mes gratitudes s'adressent aussi à mes enseignants du département d'informatique qui n'ont pas lésiné à nous transmettre leur savoir et leur savoir-faire. Sans oublier l'équipe chargé du post-graduation et de la recherche scientifique : Yasmine, Meriem et Aicha.

Résumé

L'émergence du Big Data a apporté de nombreux aspects et de bonnes pratiques qui ont mis l'accent sur plusieurs technologies de l'information telles que les entrepôts de données (EDs), qui font face à de nouveaux défis pour développer leurs services en adéquation avec les nouvelles exigences des systèmes décisionnels. Aujourd'hui, les grandes entreprises doivent analyser leurs données agrégées et stockées au moyen de requêtes d'analyse complexes. Pour satisfaire les exigences du décideur, une compréhension approfondie des propriétés de ces requêtes est nécessaire. En plus de leur nombre élevé, ces requêtes sont dynamiques et fortement interagies. Deux requêtes sont considérées comme interactives si elles partagent des sous-expressions communes. L'exploitation des sous expressions communes est l'un des problèmes les plus importants de la base de données, largement étudié au début des années 80 sous le nom de problème d'optimisation multi-requêtes (PMQO). La littérature a rapporté que le PMQO a largement contribué à résoudre plusieurs instances de la conception physique des EDs, en particulier dans la sélection des vues matérialisées. La sélection du meilleur ensemble de vues matérialisées pour optimiser les performances des requêtes est une tâche difficile. Compte tenu de leur importance et de la complexité de leur sélection, plusieurs efforts de recherche tant du milieu universitaire que de l'industrie ont été menés. Malgré leur importance, les études à ce jour n'ont pas réussi à gérer simultanément les trois propriétés des requêtes analytiques. Dans cette thèse, nous proposons un hypergraphe dynamique comme structure de données pour gérer l'ensemble des trois propriétés mentionnées ci-dessus, et nous montrons sa grande contribution dans la corrélation et le traitement des deux problèmes d'optimisation multi-requêtes et de sélection des vues matérialisées. Cette structure de données exploite les modèles de coût que nous avons développés pour capturer les sous-expressions communes des requête et matérialiser les plus bénéfiques. Notre approche s'accompagne d'une stratégie proactive, qui oriente les premières requêtes en fonction d'un seuil donné δ vers la phase hors ligne qui sélectionne leurs vues matérialisées appropriées. La phase en ligne exploite le pool de vues obtenu par la première phase pour optimiser les nouvelles requêtes entrantes, et sélectionne de nouvelles vues en supprimant les moins avantageuses. Pour valider notre approche, nous avons mené des expérimentations extensives pour évaluer l'efficacité de notre proposition ainsi que son intégration rentable dans un SGBD commercial.

Mots clés :
Entrepôts de données, Vues matérialisées, Hypergraphes, Interaction des requêtes, Requêtes à grande échelle.

Abstract

The emergence of Big Data has brought many aspects and good practices that have highlighted several information technologies such as data warehouses (DWs), which face new challenges to develop their services in adequacy with the new requirements of decision making systems. Today, large companies need to analyze their aggregated and stored data through complex analytical queries. To satisfy the decision-maker's requirements, a deep understanding of the properties of these queries is necessary. In addition to their high number, these queries are dynamic and highly interacted. Two queries are considered as interactive if they share common sub expressions. The exploitation of common sub expressions is one of the most important problems in the database, widely studied at the beginning of 80's under the name of the problem of multi-query optimization (PMQO). The literature has reported that the PMQO has largely contributed to solving several instances of the physical design of DWs, especially in the selection of materialized views. Selecting the best set of materialized views to optimize query performance is a difficult task. Given their importance and the complexity of their selection, several research efforts from both academia and industry have been conducted. Despite their importance, studies to date fail to consider the complete set of the three properties of queries. In this thesis, we propose a dynamic hypergraph as a data structure to manage the three above properties and we show its great contribution in the correlation and the treatment of the two problems of the multi-query optimization and materialized views selection. This data structure leverages our cost models towards capturing the common query sub expressions and materializing the most beneficial ones. Our approach is accompanied by a proactive strategy, which orients the first queries according to a given threshold δ towards the offline phase which selects their appropriate materialized views. The online phase exploits the pool of views obtained by the first phase to optimize the new incoming queries, and selects new views by dropping the less beneficial ones. To validate our approach, we conducted extensive experiments to assess the effectiveness of our proposal as well as its cost-effective integration in a commercial DBMS.

Keywords :
Data Warehouses, Materialized Views, Hypergraphs, Query Interaction, Large Scale of Queries.

ملخص

أدى ظهور البيانات الضخمة إلى جلب العديد من الجوانب والممارسات الجيدة التي سلطت الضوء على العديد من تكنولوجيات المعلومات مثل مستودعات البيانات ، و التي أصبحت تواجه تحديات جديدة في تطوير خدماتها بما يتماشى مع المتطلبات الجديدة لنظم صنع القرار الحديثة. تحتاج الشركات الكبيرة اليوم إلى تحليل بياناتها المجمعة و المخزنة من خلال الاستعلامات التحليلية المعقدة. لتلبية متطلبات صناع القرار، يستوجب علينا فهما شاملا لخصائص هذه الاستعلامات. بالإضافة إلى عددها الكبير، فإن هذه الاستعلامات تعتبر ديناميكية و متفاعلة للغاية. يعتبر استعلامان تفاعليان إذا كانا يشتركان في عبارات فرعية مشتركة. يعد استغلال التعبيرات الفرعية المشتركة أحد أهم المشاكل في مجال قواعد البيانات، والتي تمت دراستها على نطاق واسع في أوائل الثمانينات تحت اسم مشكلة تحسين الاستعلامات المتعددة. أفادت الأدبيات أن مشكلة تحسين الاستعلامات المتعددة قد ساهمت إلى حد كبير في حل العديد من مشاكل التصميم المادي لمستودع البيانات، لا سيما في اختيار الرؤى المادية. يعد اختيار أفضل مجموعة من الرؤى المادية لتحسين أداء الاستعلامات مهمة صعبة ومعقدة. نظرًا لأهميتها وتعقيد اختيارها، فقد تم بذل العديد من الجهود البحثية من الأوساط الأكاديمية والصناعية. على الرغم من أهميتها، لم تتمكن الدراسات حتى الآن من إدارة الخصائص الثلاثة للاستعلامات التحليلية في وقت واحد. في هذه الأطروحة، نقترح استعمال الرسوم البيانية الديناميكية الفائقة كهيكل بيانات لإدارة مجموعة الخصائص الثلاثة المذكورة أعلاه ، ونعرض مساهمتها الكبيرة في ربط ومعالجة مشكلتي التحسين متعدد الاستعلامات واختيار الرؤى المادية. تستفيد بنية البيانات هذه من نماذج التكلفة التي طورناها لالتقاط التعبيرات الفرعية الشائعة للاستعلامات وتجسيد أكثرها فائدة. نهجنا مصحوب باستراتيجية استباقية ، توجه الطلبات الأولى وفقًا لعتبة معينة نحو مرحلة وضع عدم الاتصال التي تحدد الرؤى المادية المناسبة لها. تستغل المرحلة المتصلة مجموعة الرؤى المادية الناتجة عن المرحلة الأولى بهدف تحسين الطلبات الديناميكية الواردة ، واختيار الرؤى المادية الجديدة عن طريق حذف الرؤى الأقل فائدة. للتحقق من صحة نهجنا ، أجرينا تجارب مكثفة لتقييم فعالية اقتراحنا بالإضافة إلى تكامله الفعال من حيث التكلفة في نظام إدارة قواعد البيانات التجاري.

الكلمات المفتاحية :

مستودعات البيانات ، الرؤى المادية ، الرسوم البيانية الفائقة ، تفاعل الاستعلامات ، العدد الكبير للاستعلامات.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Nomenclature

CQL	**C**assandra **Q**uery **L**anguage
DBA	**D**atabase **A**dministrator
ED	**E**ntrepôt de **D**onnées
ETL	**E**xtract **T**ransform **L**oad
hMeTiS	**M**ultilevel **H**ypergraph **P**artitioning
JSON	**J**ava**S**cript de **O**bject **N**otation
LRU	**L**east **R**ecently **U**sed
MQO	**O**ptimisation **M**ulti Requêtes
MQT	**M**aterialized **Q**uery **T**able
MVPP	**P**lan d'**E**xécution **M**ulti **V**ues
OLAP	**O**n-Line **A**nalytical **P**rocessing
OLTP	**O**n-Line **T**ransaction **P**Processing
OQL	**O**bject **Q**uery **L**anguage
PMQO	**P**roblème d'**O**ptimisation **M**ulti Requêtes
PSS	**P**roblème de **S**élection des **S**ous-expressions Communes
PSV	**P**roblème de **S**élection de **V**ues Matérialisées
QoS	**Q**ualité de **S**ervice
RDF	**R**esource **D**escription **F**ramework
ROLAP	**R**lational **O**n-Line **A**nalytical **P**Processing
SGBD	**S**ystème de **G**estion de **B**ase de **D**onnées
SPJ	**S**élection **P**rojection **J**ointure
SSD	**S**ystèmes de **S**tockage de **D**onnées
SSB	**S**tar **S**chema **B**enchmark
SPARQL	**S**imple **P**rotocol and **R**df **Q**uery **L**anguage
SQL	**S**tructured **Q**uery **L**anguage
UQP	**U**nified **Q**uery **P**lan
XML	e**X**tensible **M**arkup **L**anguage

Chapitre 1

Introduction Générale

"**"L'imagination est plus importante que le savoir..."**"
— —— Albert Einstein

1.1 Contexte et problématique

Aujourd'hui, le succès des entreprises est souvent soutenu par une excellente infrastructure informatique, contenant des données précieuses sur leurs activités et leurs clients. Ces données sont généralement stockées dans des référentiels de données qui soutiennent le processus de la prise de décision tels que les entrepôts de données. Au cours des dernières années, les systèmes d'entreposage de données ont connu une révolution spectaculaire pour améliorer leur qualité de services (QoS), permettant aux entreprises d'augmenter leur pouvoir de décision. L'émergence des Big Data [2] a été l'un des facteurs les plus importants motivant le développement des systèmes des entrepôts de données pour gérer les différents *"V"* apportés par le Big Data (Volume, Variété, Véracité, Vélocité et Valeur). Dans la littérature, Plusieurs études importantes ont été déployées pour augmenter (développer) un entrepôt de données. Ces études concernent les différentes phases de son cycle de vie : sources de données, où de nouvelles ressources comme le Linked Open Data ont été envisagées [3], ingestion de données [4], infrastructures de déploiement [5], ressource de calcul [6], etc.

Les requêtes, qui sont l'une des composantes les plus importantes dans les systèmes d'entreposage de données, ont également été affectées par l'explosion du phénomène des Big Data, où des techniques avancées ont été proposées pour optimiser leurs performances. Les vues matérialisées (VM) et les calculs répétés sur des données redondantes sont considérées comme l'une des techniques les plus intéressantes pour optimiser la performance des requêtes dans les systèmes de stockage des données (SSD) modernes.

Aujourd'hui, les requêtes analytiques sont : (1) très nombreuses (2) dynamiques et (3) partagent des opérations similaires. Compte tenu de l'importance de ces trois propriétés et de leur relation étroite avec de nombreux problèmes importants (e.g. la conception physique), il est devenu nécessaire de les clarifier. À l'égard de nombre élevé de requêtes, de nombreux

exemples concrets de bases de données analytiques et sémantiques peuvent être donnés : (i) l'application Periscope qui gère une vingtaine de millions de requêtes par jour exécutées sur plusieurs entrepôts de données [1]. (ii) les résultats importants d'un papier récent publié à DaWak'2022, traitant du problème de sélection des vues matérialisées (PSV) dans le SGBD PostgreSQL, sont obtenus sur la base de 10 000 requêtes s'exécutant sur un schéma en étoile (SSB) [7]. (iii) Dans ER'2020, un méta-modèle basé sur les ontologies a été proposé pour intégrer le concept de confiance dans l'environnement LOD, en considérant une charge de 43 284 requêtes SPARQL. En ce qui concerne l'aspect dynamique des requêtes OLAP, l'évolution des directives métier et la diversité des processus d'analyse des données rendent les requêtes analytiques plus dynamique [8]. Ces facteurs de dynamisme ont conduit à une augmentation significative des travaux de recherche portant sur les bases de données indépendantes (autonomes) et auto-adaptatives [9, 10].

Ces propriétés des requêtes n'ont pas attiré le même degré d'intérêt de la part de la communauté de recherche, où la propriété de partage des opérations a pris la plus grande part des travaux de recherche. Un papier récent publié à ICDE'2020 rapporte que 18 % de la charge de travail réelle d'Alibaba sont des requêtes redondantes. Deux requêtes sont considérées comme redondantes si elles sont similaires ou si elles partagent des sous-expressions communes. Dans [11], une analyse qualitative de la charge de requêtes SQLShare a identifié 509 de projections communes et 82 prédicats de sélections communs, ce qui exprime une valeur élevé du point de vue du partage. Le phénomène de partage dans les entrepôts de données ne concerne pas seulement les requêtes, mais aussi les données [12], les infrastructures de déploiement [13] et les ressources de calcul [14].

L'identification de sous-expressions les plus bénéfiques pour réduire le coût d'évaluation des requêtes est connue sous le nom de problème de sélection de sous-expressions communes (PSS). Le PSS est l'un des problèmes les plus anciens pour la communauté des bases de données et il est connecté à de nombreux problèmes qui sont gérés à l'intérieur et à l'extérieur d'un SGBD. Dans les années 80, le PSS a suscité un intérêt particulier où un nouveau phénomène a été identifié par Timos Sellis sous le nom d'optimisation multi requêtes (PMQO). Le problème d'optimisation multi-requêtes consiste à trouver la meilleure fusion de plans de requêtes qui optimise l'ensemble de la charge de travail. Le PMQO a généralement été étudié pour des charges de requêtes connues à l'avance (statiques), où plusieurs algorithmes optimaux ont été proposés en tenant compte des charges de travail de volume raisonnable [15]. Pour faire face à de grandes charges de requêtes impliquant d'énormes sous-expressions communes, des algorithmes approximatifs ont été proposés en temps quadratique au nombre de sous-expressions communes [16]. Dans la littérature, l'optimisation multi-requêtes a largement contribué à résoudre de nombreux problèmes de conception physique qui ont été amplifiés par l'arrivée de l'ère des entrepôts de données. Le problème de

1. https://www.sisense.com/press-release/periscope-data-and-learn-it-partner-to-train-data-teams/

sélection des vues matérialisées est l'un des problèmes les plus importants qui a tiré parti du concept d'optimisation multi-requêtes pour optimiser les performances des requêtes analytiques.

La sélection de vues matérialisées est une tâche difficile pour la conception des bases de données avancées. L'idée d'utiliser des vues matérialisées pour satisfaire la qualité de service (QoS) des bases de données ne date pas d'aujourd'hui, mais de quarante ans [17]. L'importance des vues matérialisées a été amplifiée depuis que la conception physique de l'entrepôt de données est devenue plus sophistiquée pour faire face aux requêtes complexes d'aide à la décision [18, 19]. La sélection d'un ensemble de vues matérialisées qui satisferait les exigences fonctionnelles et non fonctionnelles est une tâche complexe [20]. Cela a donné lieu au problème de sélection de vue matérialisée (PSV) qui est prouvé comme NP-difficile [20]. Au cours de la dernière décennie, le PSV a été l'un des thèmes de recherche les plus actifs dans le domaine des entrepôts de données. Par scholar googling "Materialized Views Selection" et en choisissant la période 2012-2022, nous obtenons plus de 21 000 résultats[2]. De nombreuses solutions ont été implémentées dans les principaux SGBD open source et commerciaux. La plupart de ces travaux ne considèrent généralement que quelques dizaines de requêtes statiques [21]. Certains de ces travaux ont étudié le PSV dans un contexte dynamique, où des approches réactives ont été proposées pour sélectionner des vues matérialisées pour une utilisation transitoire, au lieu de s'appuyer sur l'historique des optimisations dédiées aux charges de requêtes précédentes.

En examinant les travaux existants traitant du PMQO et du PSV, nous constatons que les études à ce jour ne parviennent pas à considérer simultanément les trois propriétés des charges de travail récentes, où des charges de requêtes dynamiques à grande échelle impliquant des sous-expressions redondantes sont prises en compte. Un autre point que nous avons constaté lors de l'analyse de ces études est l'absence de l'aspect dynamique dans les travaux qui traitent le PMQO et le PSV de manière conjointe. En effet, la plupart de ces travaux ont considéré le processus de sélection des vues matérialisées comme un processus statique qui ignore la phase d'évolution du système qui permet de gérer l'arrivée de nouvelles requêtes et de mettre à jour les vues existantes. Cette évolution nécessite de faire passer le système d'un état à un autre en proposant des approches d'optimisation en ligne dédiées à l'optimisation des requêtes dynamiques (ad-hoc).

1.2 Objectifs et contributions

L'objectif principal de notre travail de thèse est de proposer une approche proactive pour optimiser les performances des requêtes analytiques s'exécutant sur les applications modernes. De nos jours, les entreprises effectuent des analyses en temps réel sur leurs données

2. https://scholar.google.com/scholar?q=materialized+views+selection&hl=fr&as_sdt=0%2C5&as_ylo=2012&as_yhi=2022

d'entreprise au moyen de requêtes d'analyse complexes, qui présentent trois propriétés principales : (1) très nombreuses (2) dynamiques/ad-hoc et (3) partagent des expressions (opérations) communes. Ces propriétés se rapportent aux différentes phases du cycle de vie de l'entrepôt de données et peuvent affecter de nombreux problèmes complexes et importants tels que l'optimisation multi requêtes et la sélection des vues matérialisées. Pour atteindre nos objectifs, il est nécessaire de présenter une étude de synthèse des différents concepts et sujets connexes. Le premier point que nous abordons concerne les entrepôts de données, en se concentrant principalement sur leur cycle de vie et les défis actuels et futurs. Ensuite, nous abordons les deux problèmes d'optimisation multi-requêtes et de sélection des vues matérialisées, en présentant un état de l'art sur les travaux les plus importants qui traitent ces deux problèmes d'une manière séparée ou conjointement. Une analyse approfondie de cette étude d'état de l'art nous a permis de découvrir l'efficacité de la théorie des graphes dans la gestion des deux problèmes. Par conséquent, nous proposons dans ce travail d'utiliser les hypergraphes comme support pour gérer nos trois propriétés, en définissant des algorithmes intelligents avec moins coût de calcul et haute qualité pour traiter le PMOO et le PSV d'une manière conjointe.

Les contributions principales apportées par ce travail de thèse sont résumé de la façons suivante :

1. **les hypergraphes comme solution pour gérer nos trois propriétés de requêtes** En se basant sur notre étude d'état de l'art, et d'après notre analyse des travaux existants, nous avons constaté leur incapacité à gérer simultanément les propriétés récentes des requêtes analytiques. La deuxième observation que nous avons fait concerne l'utilisation massive des structures de graphes par ces travaux pour surmonter la difficulté des deux problèmes d'optimisation multi requêtes et de sélection des vues. Au cours des dernières années, une approche a été proposé à [22] qui s'appuie sur les structures des hypergraphes pour coupler le PMQO et le PSV en considérant des requêtes interactives à grande échelle. L'utilisation des hypergraphes est motivée par leur capacité à gérer des problèmes complexes à grande échelle et leurs outils de partitionnement (e.g. hMeTis). Des résultats prometteurs ont été obtenus, montrant le grand avantage des hypergraphes dans le traitement conjoint du PMQO et du PSV [22]. Cette étude ne tient pas compte de l'aspect dynamique des requêtes analytiques, ce qui contredit les enjeux et les besoins actuels des entreprises. Pour combler les lacunes de cette approche, nous proposons une approche qui capitalisent l'utilisation des hypergraphes pour coupler le PMQO et le PSV sous un nouvel angle, où des requêtes dynamiques à grande échelle impliquant des sous-expressions redondantes sont prises en compte (voir la figure 1.1).

2. **Une approche proactive pour la sélection des vues matérialisées** Pour faire face à la

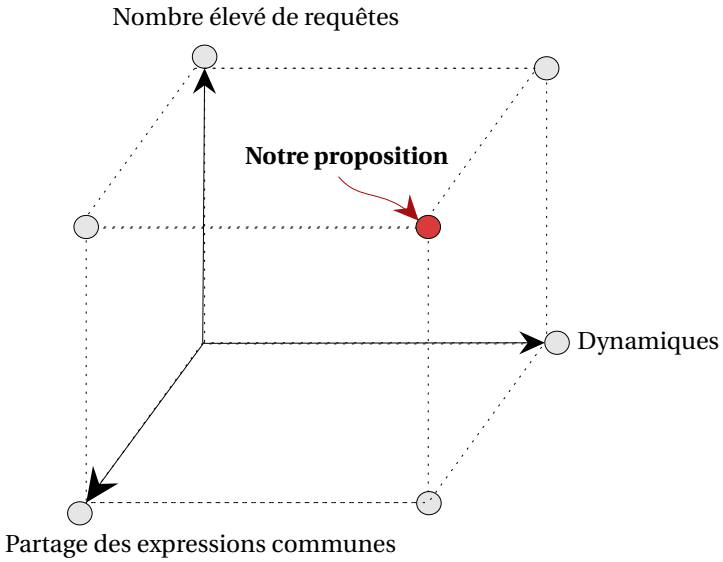


FIGURE 1.1 – Position de notre approche selon l'état de l'art

nature dynamique des requêtes analytiques, plusieurs approches réactives ont été proposées dans la littérature pour traiter le PSV. Ces systèmes ne sont pas conçus pour anticiper les requêtes futures lors de la sélection de vues matérialisées. De ce fait, nous proposons dans ce travail une approche proactive qui vise à sélectionner les vues matérialisées les plus bénéfiques pour les requêtes actuelles et futures. Comme nos requêtes arrivent dynamiquement, il est nécessaire de rendre notre système plus évolutive, en intégrant une phase d'optimisation en ligne dédiée aux requêtes dynamiques. Notre approche est accompagnée d'une stratégie, qui oriente un ensemble de premières requêtes vers la phase hors ligne en se basant sur un seuil donnée δ. Durant la phase hors ligne, les sous-expressions communes des requêtes sont capturées au moyen de notre structure d'hypergraphe. Par la suite, les opérations les plus bénéfiques estimées par notre modèle de coûts (voir l'annexe A) sont considérées comme candidates à la matérialisation. Contrairement à la plupart des études traditionnelles qui supposent que les requêtes sont déjà pré-ordonnées, nos requêtes peuvent être ré-ordonnées afin d'augmenter la réutilisation des vues sélectionnées avant leur suppression. Les autres requêtes à venir sont optimisées par la phase en ligne qui exploite les résultats de la première phase à travers trois actions : (i) la ré-utilisation des vues déjà sélectionnées, (ii) l'élimination de celles qui sont moins avantageuses, et (iii) la sélection de nouvelles vues si nécessaire.

3. **Le développement d'assistant (wizard) ProRes** À la lumière de nos découvertes, nous avons développé un wizard (conseiller) appelé ProRes, inspiré par les assistants les plus importants développés par les éditeurs commerciaux qui aident les DBA et les concepteurs dans leurs activités d'administration lors de la sélection des vues matérialisées. La particularité (l'avantage) de ProRes est son efficacité dans la gestion simultanée des

propriétés récentes des requêtes analytiques d'aujourd'hui.

1.3 Organisation du manuscrit et publications

D'un point de vue organisationnel, le reste du manuscrit est organisé comme suit (voir la figure 1.2) :

- **Le chapitre 2** présente un aperçu sur les entrepôts de données sur lesquelles nos expérimentations sont validées. Nous commençons ce chapitre par introduire les définitions et les notions de base d'entreposage de données. Ensuite, nous passons en revue l'architecture générale d'entrepôt de données et nous discutons les architectures les plus répandues, en montrant les avantages et le cas d'utilisation de chaque architecture. Après cela, nous présentons les différentes phases du cycle de vie de conception des systèmes d'entreposage de données. Nous terminons ce chapitre en discutant les efforts inlassables du milieu de recherche pour développer ces systèmes.

- **Le chapitre 3** est consacré à la présentation des processus de traitement et d'optimisation des requêtes. Nous présentons en premier lieu les différentes étapes de traitement des requêtes dans un SGBD relationnel. Puis, nous aborderons les propriétés récentes des requêtes analytiques imposées par les exigences actuelles des entreprises. En troisième lieu, nous présentons un état de l'art couvrant les travaux les plus importants qui s'appuie sur l'exploitation de sous expressions communes pour optimiser la performance des requêtes. Nous nous focalisons sur les travaux qui traitent le PMQO et le PSV. Sur la base de cette étude, une nouvelle classification de ces travaux a été proposée en intégrant de nouveaux critères : la considération des propriétés récentes des requêtes analytiques, les structures de données utilisées et la considération des techniques d'ordonnancement des requêtes.

- **Le chapitre 4** détaille nos principales contributions qui consistent à utiliser la théorie des hypergraphes pour surmonter la gestion simultanée de nos trois propriétés, lors du traitement des deux problèmes inter-connectés d'optimisation multi-requêtes et de sélection de vues matérialisées. Dans la première partie de ce chapitre, nous montrons la capacité et l'efficacité des hypergraphes à gérer simultanément : (i) le nombre élevé de requêtes (ii) leur dynamique et (iii) leur forte interaction. Ensuite, nous présentons notre approche de sélection proactive de vues matérialisées qui intègre concurremment nos trois propriétés et nous étudions la complexité de nos algorithmes.

- **Le chapitre 5** présente une validation expérimentale de notre approche et montre les différents module de notre outil. Nous commençons notre étude d'efficacité par une description de l'environnement matériel, de jeu de données et de requêtes utilisées dans nos expérimentations. Ensuite, nous menons des expérimentations extensive pour

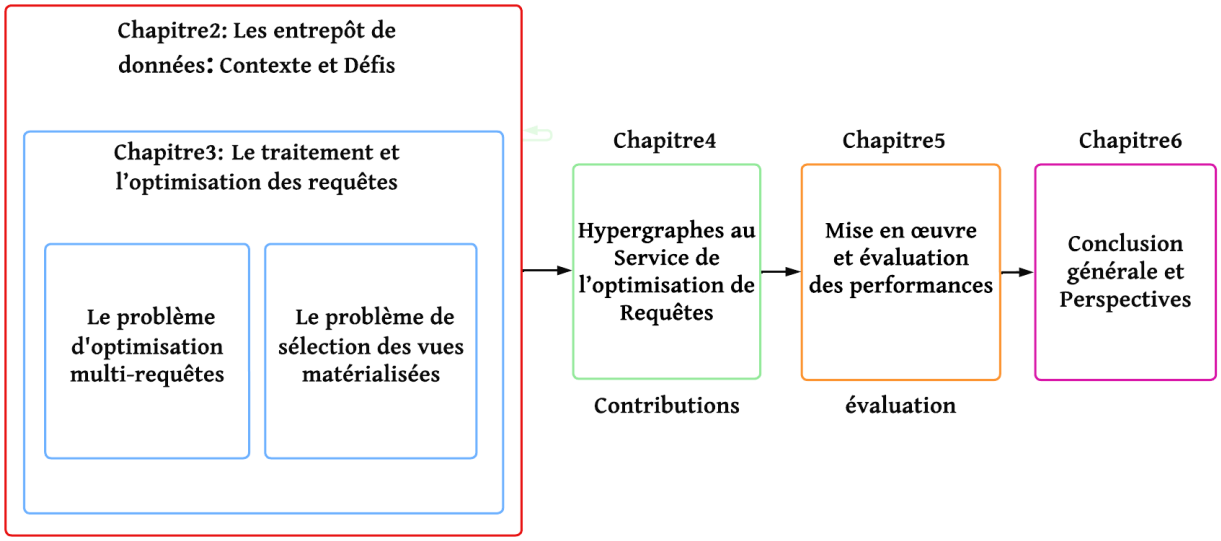


FIGURE 1.2 – Répartition des chapitres de thèse

évaluer l'efficacité de nos algorithmes par rapport aux principales études de l'état de l'art. Finalement, nous présentons notre outil avec ses différents modules.

- **La Conclusion générale** : conclut la thèse en fournissant un résumé et une évaluation des contributions apportées par ce travail. Ce chapitre présente également les perspectives qui ressortent de nos travaux.

Nous listons dans ce qui suit l'ensemble de publications et de communications internationales liées aux contributions apportées par ce travail de thèse.

Publications internationales :

Mustapha Chaba Mouna, Ladjel Bellatreche, Boustia Narhimene : ProRes : Proactive reselection of materialized views. Comput. Sci. Inf. Syst. 19(2) : 735-762 (2022) [23]

Communications internationales :

Mustapha Chaba Mouna, Ladjel Bellatreche, Boustia Narhimene : Selecting Subexpressions to Materialize for Dynamic Large-Scale Workloads. DaWaK 2021 : 39-51 [24]

Mustapha Chaba Mouna, Ladjel Bellatreche, Boustia Narhimene : HYRAQ : optimizing large-scale analytical queries through dynamic hypergraphs. IDEAS 2020 : 17 :1-17 :10 [25]

Mustapha Chaba Mouna, Ladjel Bellatreche, Boustia Narhimene : Optimisation Conjointe et Dynamiques des Requêtes Régulières et Recommandées. EDA 2018 : 195-210 [26]

Chapitre 2

Les Entrepôts de Données : Contexte et Défis

“ Un entrepôt de données ne s’achète pas, il se construit. "
— — Bill Inmon

Sommaire

2.1 Introduction

L’entreposage de données remonte aux années 1980, lorsque les chercheurs Paul Murphy et Barry Devlin d’IBM ont développé l’«entrepôt de données d’entreprise». Le concept d’entrepôt de données de Devlin et Murphy consistait à résoudre les divers problèmes associés à la transformation de flux de données des systèmes opérationnels aux systèmes d’aide à la décision.

Les idées qui ont contribué à l’émergence des entrepôts de données ont commencé dès les années 1960. Tout a commencé dans un projet de recherche conjoint entre General Mills et Dartmouth College, où les termes «dimensions» et «faits» ont été développé pour la première fois. Puis, dans les années 70, lorsque Bill Inmon a commencé à définir le terme «Data Warehouse» et à discuter de ses concepts et principes.

Le succès massif des entrepôts de données au fil des années 1990, a conduit les entreprises à exiger des services plus efficaces et un accès accru aux données afin de prendre des décisions pertinentes et d’économiser de l’argent. La décennie 2000 a été marquée par l’émergence de nombreuses applications qui ont attiré des millions d’utilisateurs. Cela a conduit à une croissance significative du volume de données et de complexité des charges

de requêtes générées par ces applications. Facebook génère près d'un péta-octet de données chaque jour.

L'explosion des données a conduit à l'émergence de nouvelles structures de données flexibles et capables d'exprimer des relations plus complexes, y compris des données semi-structurées telles que XML, JSON et Avro. Étant donné que les entrepôts de données traditionnels n'étaient pas conçus pour gérer ces types de données, de nouveaux systèmes NoSQL sont apparus tels que Hadoop.

Récemment, De nombreux efforts ont été déployés pour développer les systèmes des entrepôt de données par les meilleurs aspects et principes apportés par les Big Data. Ces efforts concernent notamment l'architecture générique et les différentes phases du cycle de vie de l'entrepôt de données.

Nous présenterons dans ce chapitre, un état de l'art portant sur les entrepôt de données, leur architecture, leur cycle de vie et leur évolution. En nous focalisant particulièrement sur les différentes phases du cycle de vie , et les divers efforts déployés pour les développer. .

2.2 Concepts de base

Dans cette section, nous présentons quelques définitions et notions de base liées aux entrepôts de données et nous expliquons leurs principales caractéristiques.

2.2.1 Définitions

plusieurs descriptions ont été proposées dans la littérature pour définir un entrepôt de données. Nous discutons ci-dessous les définitions les plus populaires d'un entrepôt de données.

Définition 2.2.1 *Le terme entrepôt de données a été inventé par Bill Inmon en 1990 qui a défini un entrepôt de données par une collection de données orientée sujet, historisée, intégrée, non volatile et utilisées pour aider à la prise de décision d'une entreprise [27].*

Cette définition est restée précise pendant une dizaine d'années. Par la suite, de nombreux chercheurs ont proposé d'autres définitions en examinant d'autres dimensions.

Définition 2.2.2 *les auteurs dans [28] proposent une autre définition en considérant un entrepôt de données comme une architecture de stockage de données qui permet aux décideurs d'entreprise d'organiser systématiquement, comprendre et utiliser leurs données pour prendre des décisions stratégiques et pertinentes. Cette définition a attiré de nombreux chercheurs comme Matthew Mayo qui a utilisé cette définition dans son article*[1].

1. https://www.kdnuggets.com/2016/08/big-data-key-terms-explained.html/2

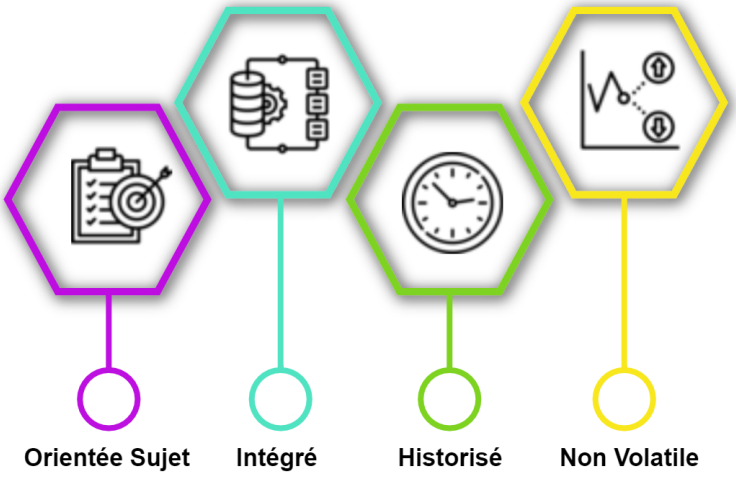


FIGURE 2.1 – Les caractéristiques d'un entrepôt de données

Définition 2.2.3 *par ailleurs, Bin Jiang dans son livre [29], a discuté la validité de la définition donnée par Inmon et a redéfini l'entrepôt de données, en le décrivant comme suit : un entrepôt de données est une infrastructure informatique permet à l'entreprise d'intégrer, de collecter et de préparer régulièrement des données afin de faciliter l'analyse de données.*

Définition 2.2.4 *Ralph Kimball qui est considéré l'un des leaders dans le domaine des entrepôts de données a proposé une définition beaucoup plus simple que celle de Bill Inmon. Il définit un entrepôt de données comme une copie des données de transaction spécifiquement structurées pour l'exploration et l'analyse [30].*

La définition de Kimball offre moins de perspicacité et de profondeur que celle de Inmon, mais elle n'est pas moins précise. Pendant de nombreuses années, les chercheurs ont débattu de l'approche la plus efficace pour les entreprises entre l'approche de Ralph Kimball et celle de Bill Inmon. Dans les sections suivantes nous détaillons les deux approches , ses avantages et ses inconvénients.

2.2.2 Les caractéristiques clés d'un entrepôt de données

Un entrepôt de données est caractérisé par quatre points principaux : orientée sujet, historisé, intégré et non volatile (La figure 2.1). Ci-dessous, Nous expliquons brièvement ces caractéristiques principales.

orientée sujet : Un entrepôt de données est conçu pour aider les entreprises à analyser leurs données définies sur un sujet particulier. Ces sujets peuvent être liés aux ventes, aux marketing, aux distributions, etc.

historisé : Ceci signifie que les données collectées dans un entrepôt de données sont organisées par périodes (hebdomadaires, mensuelles, annuelles, etc.) et fournissent des informations du point de vue historique.

intégré : Les données d'un entrepôt sont rassemblées à partir de diverses sources et fusionnées d'une manière cohérente.

non volatile : cela signifie que les données antérieures ne seront pas supprimées lorsque de nouvelles données y sont ajoutées. Les données historiques sont conservées pour les comparaisons et l'analyse.

2.3 Architectures des entrepôts de données

L'architecture des entrepôts de données fait référence à la conception d'un framework de collection et de stockage des données d'une entreprise. Elle se concentre sur la recherche de la méthode la plus efficace pour extraire des informations d'un ensemble brut de données et de les placer dans une structure facilement digestible dédiée aux processus d'analyse et d'aide à la décision.

Ces dernières années, L'architecture des entrepôt de données a connu une évolution assez spectaculaire en raison de plusieurs facteurs. Ces facteurs sont liés principalement aux objectifs commerciaux de l'entreprise et aux aspects optimales apportés par le Big Data qui continuent d'évoluer d'une façon permanente.

Dans cette section, nous présentons un aperçu sur l'architecture générique d'un entrepôt de données. Ceci est suivi d'une discussion sur les architectures les plus dominantes dans l'histoire des entrepôt de données et les facteurs qui ont contribué à leurs succès.

2.3.1 Architecture générique d'un entrepôt de données

L'architecture générale d'un entrepôt de données est composée de quatre couches [31] comme le montre la figure 2.2.

1. **Les sources de données :** Un système d'entreposage de données offre l'intégration de données provenant de sources hétérogène et nombreuses : des feuilles de calcul , des fichiers XML , des base de données relationnelles,etc. Ces données peuvent être internes et/ou externes à l'entreprise.

2. **La couche d'intégration de données :** Le processus d'intégration des données existantes provenant de sources externes est connu sous le nom de processus d'extraction, de transformation et de chargement (ETL) [32]. En pratique, de nombreuses applications commerciales ont été développées pour effectuer facilement le processus d'ETL comme Xplenty,Talend et Stitch.[33].

3. **La couche d'entrepôt de données :** Cette couche représente le lieu de stockage centralisé de données sous la forme d'un entrepôt de données ou de plusieurs magasins

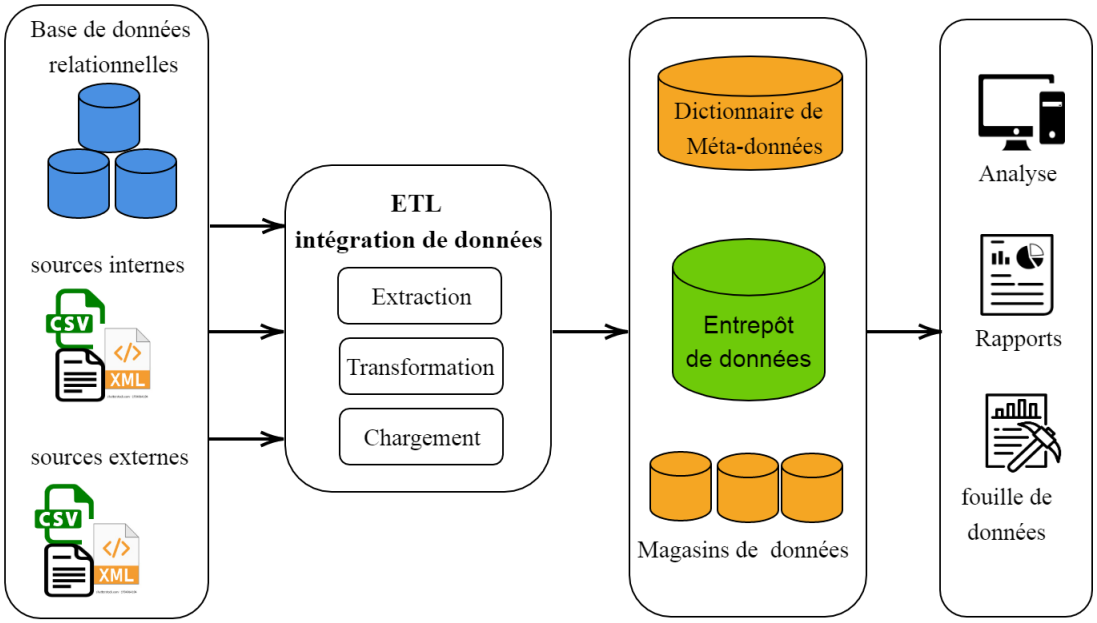


FIGURE 2.2 – L'architecture générale d'un entrepôt de données

de données (data marts). Les magasins de données sont de petites parties de l'entrepôt de données permettant de stocker les données d'un sujet particulier, tandis qu'un entrepôt de données englobe les données de toute l'entreprise. Le dictionnaire de méta-données est Un composant supplémentaire au niveau de cette couche . Il contient généralement des informations sur les structures de données, les règles métier, et des informations sur les données spécifiques de la base de données (i.e. la source et la cible des données, la qualité des données, le stockage et la présentation de données, etc.) [34].

4. **La couche d'exploitation de données :** Cette couche offre des outils permettant d'aider l'utilisateur à visualiser et à exploiter le contenu de l'entrepôt de données. Ces outils couvrent l'ensemble des besoins analytiques de l'entreprise , notamment : (i) l'exploration de données ,(ii) la génération de rapports (iii) les tableaux de bord et (iv) l'analyse OLAP [35].

2.3.2 Classification des architectures : une étude comparative

Dans la littérature scientifique, deux classifications différentes sont couramment adoptées pour l'architecture des entrepôts de données : (i) La première classification est structurelle et dépend du nombre de couches utilisées par l'architecture. (ii) Alors que la deuxième

classification dépend de la façon dont les différentes couches sont utilisées pour construire un entrepôt de données à des fins d'analyse spécifiques [36].

Dans la première classification , on distingue souvent trois types différents d'architecture des entrepôts de données :

1. **Une architecture mono-couche :** L'architecture mono-couche vise à minimiser la quantité de données stockées par l'élimination des redondances. Dans cette architecture, la couche d'entrepôt de données est virtuelle où le système d'entreposage de données et le système transactionnel partagent les mêmes données qui sont stockées dans le même endroit. Le problème de cette architecture réside dans son incapacité de répondre aux exigences pressantes de séparation entre le traitement analytique et le traitement transactionnel. [37].

2. **Une architecture à deux couches :** Afin de répondre aux exigences de séparation des traitements analytiques et transactionnels, l'architecture à deux couches ajoute une nouvelle couche pour que les sources de données soient séparées de la couche des entrepôts de données [38]. La principale limitation de cette architecture est sa faible performance dans les volumes importants de données provenant de sources différentes [39].

3. **Une architecture à trois couches :** Dans cette architecture, une nouvelle couche est ajouté permettant de matérialiser les données opérationnelles obtenues après la phase d'intégration et de nettoyage des sources de données. Cette couche est directement utilisée pour alimenter l'entrepôt de données. Ce qui permet de séparer les problèmes d'extraction et d'intégration de données de ceux d'alimentation d'entrepôt de données [37].

Une deuxième taxonomie des architectures d'entrepôt de données se compose de cinq architectures différentes : magasins de données indépendants, architecture en Bus de magasins de données, architecture en étoile (Hub-and-spoke) , architecture centralisée et architectures fédérées [40]. Chaque architecture correspond à une manière particulière dont les principales couches discutées précédemment sont combinées différemment [41].

Selon Ariyachandra et Watson [40, 42], il y a eu de nombreuses débats et même des controverses sur la meilleure architecture d'entrepôt de données. Les deux sommités du domaine d'entreposage de données Bill Inmon et Ralph Kimball, étaient au cœur de ce débat. Inmon plaide pour l'architecture centralisée , tandis que l'architecture en bus de magasins de données est approuvée par Kimball.

1. **L'architecture centralisée :** L'architecture centralisée d'un entrepôt de données a été recommandé par Bill Inmon, car elle convient aux outils de base de données traditionnels permettant de répondre aux exigences de développement d'un entrepôt de données à l'échelle de l'entreprise [43]. Dans cette architecture, il n'y a pas de magasins de

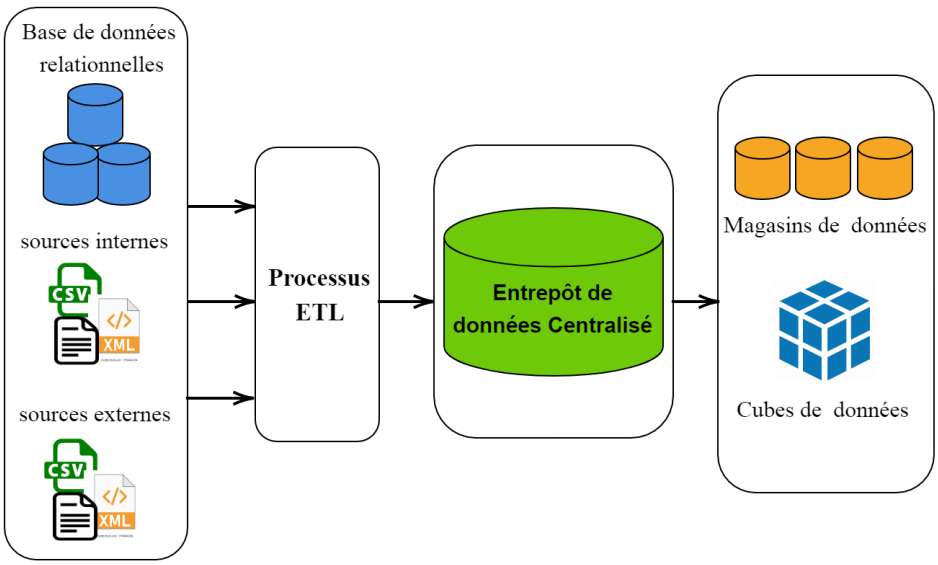


FIGURE 2.3 – l'architecture recommandée par Bill Inmon

données dépendants. Il s'agit plutôt d'un entrepôt de données centralisé contenant des données et des magasins de données intégrées[36]. La figure 2.3 illustre les principaux modules de l'architecture centralisée.

2. **L'architecture en bus de magasins de données :** Cette approche est définie par Ralph Kimball, qui propose une architecture ascendante en utilisant la modélisation dimensionnelle de données [44]. Cette approche suggéré de créer une base de données (ou un magasin de données) par processus métier, Plutôt que de créer une seule base de données à l'échelle de l'entreprise. Le bus de données est la technique inventée par Kimball permettant d'intégrer les différents magasins de données afin d'obtenir l'intégralité de l'entrepôt de données d'entreprise [43]. La figure 2.4 montre les différents modules de cette architecture.

Des expériences intensives sont conduites dans la littérature afin d'évaluer le succès des différentes architectures en se basant sur plusieurs facteurs, comme la qualité de l'information,la qualité du système, etc. [42, 45]. Les résultats obtenus ont montré que l'architecture en bus, et l'architecture centralisée ont obtenu des scores similaires dans les mesures de succès et ont surpassé toutes les autres architectures.

Nous concluons que l'architecture en bus des magasins de données et l'architecture centralisée sont les approches les plus réussies dans le monde des entrepôts de données, et que chacune d'eux a ses propres avantages et conditions de sélection.

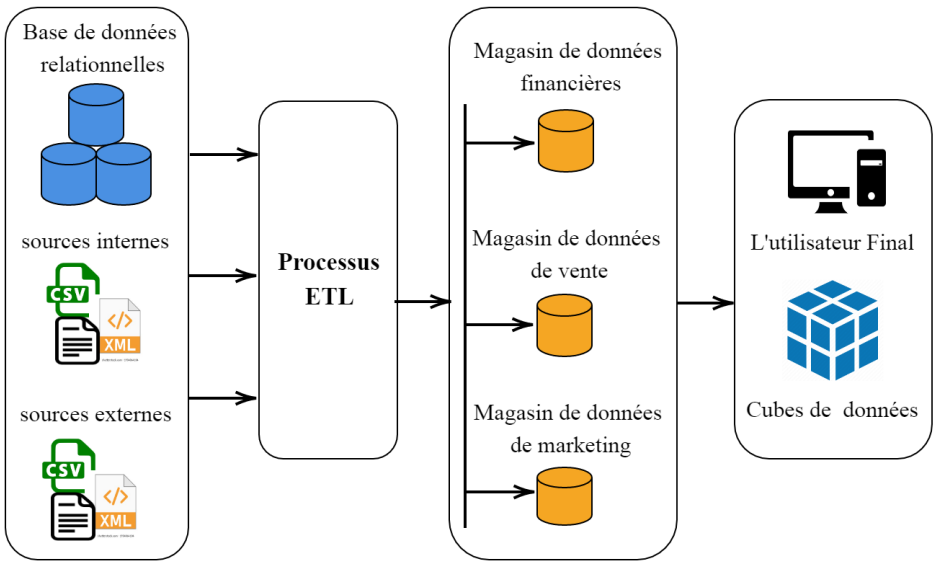


FIGURE 2.4 – l'architecture recommandée par Ralph Kimball

2.4 Conception et cycle de vie d'un entrepôt de données

La conception d'un entrepôt de données est une tâche difficile impliquant de nombreux acteurs (concepteurs, analystes, administrateurs, programmeurs et utilisateurs finaux) , ressources (données, matériels et supports de stockage) et outils d'accès aux données [46]. Afin de faciliter cette tâche, un cycle de vie de conception a été proposé dans la littérature permettant de définir une démarche complète de traçabilité dans les systèmes d'entreposage de données. Le cycle de vie de l'entrepôt de données comprend généralement les phases suivantes : analyse des besoins, modélisation conceptuelle, modélisation logique, la phase d'ETL , modélisation physique et modélisation de déploiement [47]. Les phases décrites dans la figure 2.5 doivent être suivies lors de la conception d'un entrepôt de données.

Le cycle de vie d'un entrepôt de données a connu trois évolutions principales [48] : une évolution verticale, horizontale et interne. L'évolution verticale a permis d'identifier les cinq phases de cycle de vie de conception. Au début, le cycle de vie ne comprenait que trois phases [49] : La modélisation logique, la modélisation physique et la phase d'ETL. Par la suite, le cycle a évolué verticalement par l'ajout de la phase d'analyse des besoins[50] puis de la phase de modélisation conceptuelle [51]. L'évolution horizontale a diversifié chaque phase de conception en ajoutant de nouveaux schémas de stockage (schémas relationnels, sémantiques, Nosql ou newSQL, etc [52] , plate-formes de déploiement (Centralisé, parallèle [53], cloud [54], grille [55]),etc. L'évolution interne a identifié les différentes étapes de chaque phase du cycle de vie [46]. Dans ce qui suite, nous expliquons chacune des phases de cycle de vie de l'entrepôt de données.

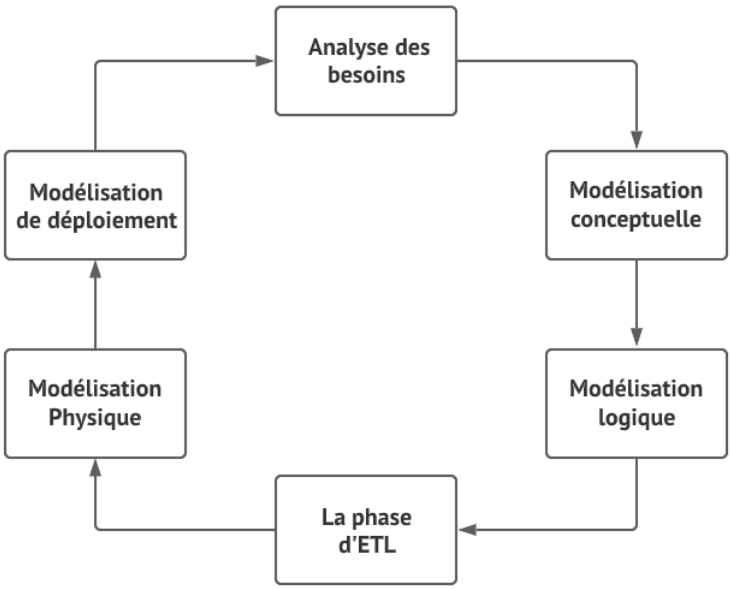


FIGURE 2.5 – Le cycle de vie d'un entrepôt de données.

2.4.1 Analyse des besoins :

la phase d'analyse des besoins est l'une des phases les plus critiques pouvant influencer toutes les autres phases de cycle de vie de l'entrepôt de données. Afin de réaliser cette tâche, les concepteurs collectent, filtrent et documentent les besoins des utilisateurs finaux afin de sélectionner et de représenter les informations pertinentes aux objectifs stratégiques de l'entreprise. L'objectif prédominant de cette phase est d'identifier les objectifs et d'élaborer les besoins qui pourraient mesurer la performance de l'entreprise [56]. A la fin de cette phase, on peut avoir des spécifications pour les faits à modéliser [47] et une meilleur vision sur les problèmes actuels ou potentiels.

La phase d'analyse de besoins a attirée une forte attention dans la littérature, où de nombreuses approches ont été proposées pour faciliter cette tâche. Ces approches sont généralement structurées en quatre étapes principales : 1) l'élicitation des besoins à travers des entretiens avec les décideurs (2) spécification des besoins (3) traduction du modèle formel des besoins en un schéma multidimensionnel d'un ou plusieurs cubes de données; et (4) la mise en œuvre de l'entrepôt de données [57]. Les besoins des entreprises sont généralement divisées en deux types différents : (a) des besoins fonctionnelles et (b) des besoins non fonctionnelles.

Besoins fonctionnelles La définition des besoins fonctionnelles est ce qu'un produit doit faire ou devrait faire. En d'autres mots, les besoins fonctionnelles décrivent l'action et la réalisation et représentent les comportements souhaités et résultants du système [58, 59]. Plusieurs exemples de besoins fonctionnelles peuvent être donnés dans le domaine des bases de données : l'authentification, la gestion des comptes et des privilèges, la gestion des transactions du système, etc.

Besoins non-fonctionnelles La littérature a largement rapporté que les besoins non fonctionnelles ont un rôle crucial dans le succès des systèmes d'entreposage de données [59, 60, 61]. Les besoins non-fonctionnelles se réfèrent généralement aux qualités attendues du système [62]. Les types courants des besoins classées comme non-fonctionnelles incluent : la performance, la sécurité, la fiabilité, l'exactitude, l'énergie et d'autres besoins qui ne décrivent pas nécessairement les actions qui doivent être prises par le système [58, 63].

2.4.2 Modélisation conceptuelle :

Cette phase vise à dériver un schéma conceptuel expressif, claire et indépendant de la mise en œuvre d'entrepôt de données selon un modèle défini [47]. De nombreux travaux de la littérature se sont concentrés sur cette phase, en proposant des solutions méthodologiques capables de satisfaire les besoins des décideurs. Certains approches sont basés sur des extensions de modélisation célèbre (e.g. E/R, UML) et d'autres se sont basés sur des modélisations ad-hoc comme dans les travaux de [64] et [65]. les partisans de la modélisation E/R prétendent que les extensions E/R devraient être adoptées car elles ont été suffisamment testées et se sont avérés puissantes et flexibles pour s'adapter à une variété de domaines d'application [66, 67]. D'un autre côté, les partisans des modèles orientés objet affirment que leurs approches sont plus expressifs et mieux représentatifs des propriétés statiques et dynamiques des systèmes d'information. La modélisation orientée objet est actuellement la tendance dominante dans la modélisation conceptuelle de données, en particulier la conception UML [68, 69, 70, 71]. Selon Matteo Golfarelli [47], les modèles ad-hoc sont plus intuitifs et lisibles par les utilisateurs non experts [64, 65].

2.4.3 Modélisation logique :

Les processus et techniques de la modélisation logique des entrepôts de données relationnelles sont très différents de ceux d'une base de données transactionnelles [49, 50]. La modélisation logique dans le contexte des entrepôts de données est conçue pour simplifier et optimiser les requêtes décisionnelles complexes [72]. Elle prend comme entrée : un schéma conceptuel, des exigences non fonctionnelles (e.g. la performance) et les règles de transformation permettent de construire le schéma logique à partir d'un schéma conceptuelle [73]. Dans la littérature, il existe trois types de modélisation logique : la modélisation Relationnelle ROLAP (Relational OnLine Analytical Processing) , la modélisation multidimensionnel MOLAP (Multidimensional OnLine Analytical Processing) et le modèle hybrid. le modèles ROLAP est considéré comme le choix le plus populaire par les concepteurs [74].

Dans le modèle ROLAP, plusieurs structures de données ont été proposées afin d'optimiser les requêtes complexes [50]. Ces structures représentent principalement trois schémas : le schéma en étoile, le schéma en flocon de neige et le schéma en constellations [50, 75, 76]. Les

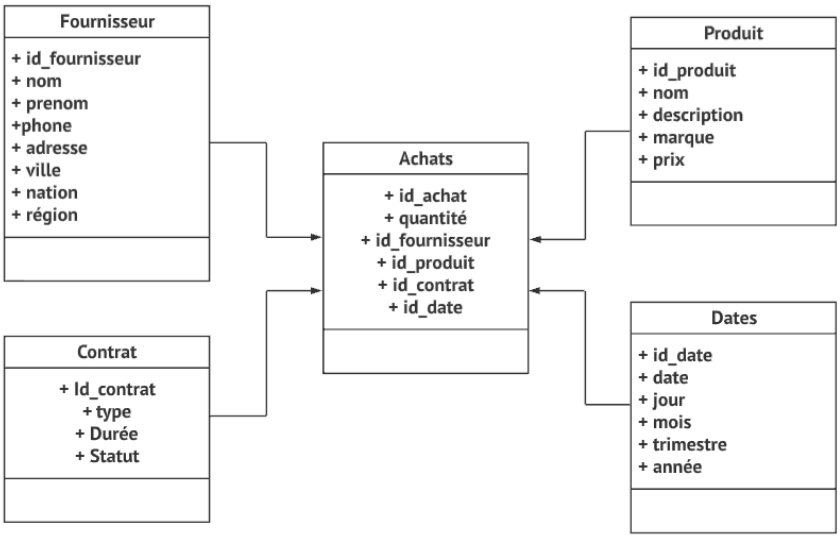


FIGURE 2.6 – Exemple d'un schéma en étoile

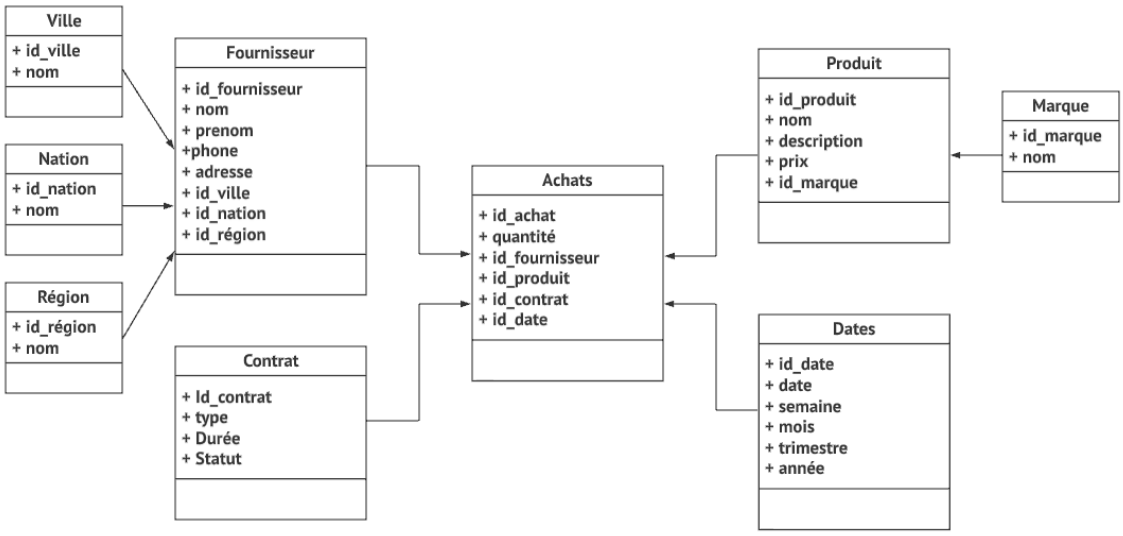


FIGURE 2.7 – Exemple d'un schéma en flocon de neige

figures 2.6, 2.7 et 2.8 représentent respectivement les trois schémas d'un entrepôt de données pour la gestion des achats et des ventes d'une entreprise commerciale.

Le paradigme de modélisation le plus répandu est le schéma en étoile [77]. Ce modèle se compose d'une grande table centrale appelée table des faits et de petites tables de dimension indépendantes entre-elles et liées directement à la table des faits [78]. Le schéma en flocon est une forme d'expansion du schéma en étoile, dans lequel certaines tables de dimension sont normalisées. Le schéma en flocon est généralement utilisé si le schéma en étoile n'est pas capable de décrire la complexité de la base de données [79]. La forme de schéma résultant est similaire à un flocon de neige. Le schéma de constellation de faits est utilisé généralement pour modéliser les applications sophistiquées qui nécessitent plusieurs tables des faits en partageant les mêmes tables de dimension. Ce type de schéma peut être considéré comme une collection des schéma en étoiles [77].

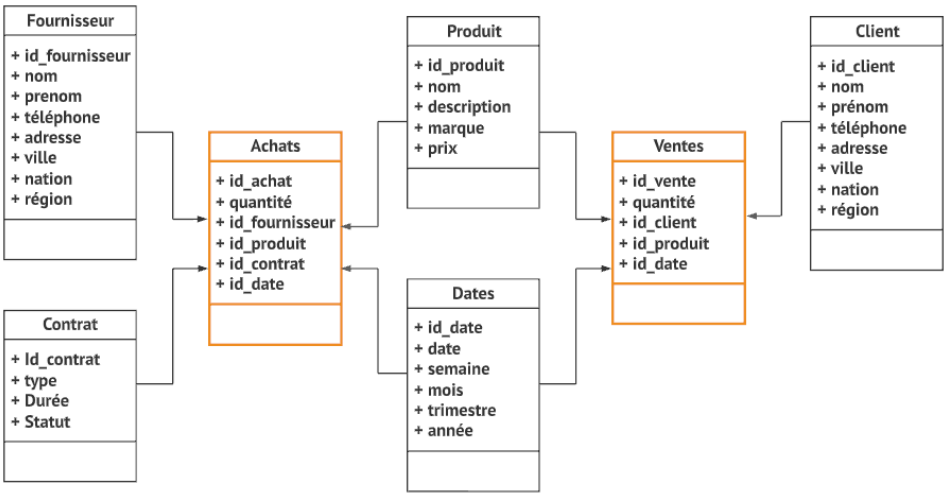


FIGURE 2.8 – Exemple d'un schéma en constellations

Le modèle multidimensionnel MOLAP est un ensemble de données représentées par un tableau multidimensionnel à n dimensions. Le schéma MOLAP est conçu pour permettre le stockage et la récupération efficaces et pratiques de gros volumes de données qui sont étroitement liées, visualisées et analysées sous différents angles [80].

Dans cette thèse, nous utilisons le schéma en étoile SSB composé d'une table des faits et de quatre tables de dimensions [81]

2.4.4 Modélisation physique :

Au cours de la dernière décennie, la modélisation (la conception) physique est devenue l'une des thèmes de recherche les plus actives dans le monde des entrepôts de données. Une petite recherche à Google Scholar sur le sujet "physical design in data warehouses", en choisissant la période 2010-2021, permet d'identifier 59 400 résultats. Cette phase vise à sélectionner un ensemble pertinent de structures d'optimisation (e.g. indexes [82] , vues matérialisées [83, 84], etc.) afin de satisfaire un ensemble des exigences non fonctionnelles (e.g. minimisation du coût d'exécution des requêtes, minimisation du consommation d'énergie, etc) tout en respectant un ensemble de contraintes (e.g. espace de stockage, coût de maintenance, etc).

Dans cette thèse, nous nous focalisons sur l'optimisation des requêtes analytique et la conception physique de l'entrepôt de données,notamment la sélection des vues matérialisées. Pour cette raison, une attention particulière sur la phase de modélisation physiques sera donné dans les sections suivantes.

2.4.5 La phase d'ETL :

La phase d'ETL est le processus responsable de l'extraction des données provenant de diverses sources hétérogène (e.g. un fichier plat, une application ERP, un tableur Excel, etc.),

de leur transformation dans de nouveaux formats selon les besoins de l'entreprise puis de leur chargement dans l'entrepôt de données cible pour être exploité par les différentes requêtes analytiques. Le facteur clé pour la réussite de la réalisation des projets d'entreposage de données dépend généralement de la qualité du conception d'ETL et de sa maintenance [85].

Le processus ETL est une tâches difficile, complexe et consomme la plupart des efforts dans la modélisation et la construction d'un entrepôt de données [86]. Selon certains experts de l'industrie, Le processus ETL prend environ 60 à 80% de l'effort des réalisation d'un projet d'entreposage de données [85, 87]. De nombreux efforts académiques et industriels ont été déployés pour concevoir efficacement le processus d'ETL en se basant sur différentes techniques [88]. Ces études peuvent être classés principalement en trois catégories : modélisation basée sur des expressions de mappage , modélisation basée sur l'environnement UML et des approches basées sur la modélisation conceptuelle [86, 88]. Une étude comparative a été menée dans[88] pour évaluer les trois classes de conception d'ETL. Sur la base de cette étude, les auteurs ont conclu que l'approche basées sur la modélisation conceptuelle d'ETL est meilleure que les deux autres approches en termes de simplicité et d'efficacité. Dans cette thèse, nous supposons que la phase de conception d'ETL est déjà réalisée.

2.4.6 Modélisation de déploiement :

la phase de déploiement joue un rôle important dans le processus d'optimisation des requêtes OLAP, où Le choix de la plateforme de déploiement est un pré-requis pour atteindre la disponibilité et l'évolutivité des données. Il existe plusieurs types de plateformes adaptées au déploiement des entrepôts de données : (1) machines centralisées [89, 90] qui est considérée comme la plateforme la plus connue et omniprésente. (2) architectures parallèles [91, 92], qui sont l'une des solutions de déploiement les plus robustes et pertinentes qui peuvent gérer efficacement le déluge de données. (3) architectures Cloud [93, 94] qui ont récemment gagné en popularité , où l'utilisation du Cloud a connue une croissance spectaculaire dans le monde industriel et académique. Notamment, dans le monde des entrepôts de données où la technologie du Cloud représente la tendance actuelle pour déployer un entrepôt de données.

2.5 Évolutions récentes des entrepôts de données

Les entrepôts de données ont une longue histoire de succès et de progrès, motivés par les demandes croissantes et continues du milieu académique et industriel pour des services plus efficaces en matière de gestion des données massives, d'augmenter le pouvoir de décision et pour capturer de la valeur ajoutée [95]. Dans cette section, nous allons donner un bref aperçu des évolutions récentes des systèmes d'entreposage de données.

Récemment, les systèmes des entrepôts de données ont été confrontés à un défi majeur représenté par la pandémie du Covid-19 qui a mis en évidence plusieurs technologies de l'information pour lutter contre ce virus. On évoque notamment, les entrepôts de données, les big data, l'apprentissage automatique, le traitement d'images et les infrastructures de stockage avancées. Les entrepôts de données ont joué un rôle déterminant dans cette guerre en assistant les gouvernements à analyser les grosses données en temps réel et à prendre les bonnes décisions au bon moment, grâce aux techniques avancées apporté par l'évolution étonnante de l'entrepôt de données ces dernières années.

De grandes entreprises et laboratoires ont travaillé sans relâche pour fournir des solutions Capables d'intégrer et d'analyser les données du virus Covid-19 qui évoluent dans le temps et dans l'espace. L'entreprise américaine *Cerner* a développé une solution d'entreposage de données permettant au centre de santé *Roper St Francis* de coordonner les ressources entre les hôpitaux, les cabinets médicaux et la communauté, et d'identifier les patients dont le test est positif et de retirer l'isolement les patients dont le test est négatif, ce qui permet de rationaliser la consommation d'équipements de protection individuelle [2]. Une autre étude [96] s'est appuyée sur l'utilisation des entrepôts de données pour mieux déterminer l'évolution de la maladie et étudier les stratégies de traitement potentielles pour les patients gravement malades.

L'évolution fulgurante des entrepôts de données résulte de plusieurs facteurs et phénomènes. Le phénomène du Big Data est l'un des facteurs les plus importants influençant cette évolution [2, 97]. Récemment, plusieurs efforts ont été proposés pour augmenter le système d'entreposage de données par les meilleurs aspects apportés par les big Data. Ces efforts couvrent la majorité des phases du cycle de vie de l'entrepôt de données en fournissant des techniques, des services et des structures de données avancées.

Aujourd'hui, les quantités de données stockées dans les entrepôts de données sont de plus en plus énormes. Alors que les entrepôts de données traditionnels ne sont plus en mesure de gérer ces données qui sont constamment générées dans de nombreux domaines et applications tels que les données des réseaux sociaux, les Blogs, les vidéos, les jeux en ligne, etc. Le défi est aggravé par le fait qu'une grande partie de ces données ne sont pas structurées (des données non structurées). Pour faire face à cette situation, de nombreuse études ont proposé d'utiliser les outils avancés du big data qui sont devenus aujourd'hui des ajouts stratégiques à l'environnement d'entreposage de données, car ils peuvent remplir plusieurs rôles.

Le modèle de programmation MapReduce [98] a joué dernièrement un rôle crucial dans la gestion, la collection et l'analyse des données massives dans le contexte des entrepôts de données [99, 100]. MapReduce permet de paralléliser les tâches pour ensuite rassembler les résultats partiels et générer le résultat final, tout cela est pris en charge dans un système de

2. https://www.cerner.com/perspectives/6-ways-data-and-analytics-can-be-powerful-tools-in-the-covid-19-fight

fichiers distribué comme Hadoop [101]. Hadoop est une plateforme open source permettant de capturer, organiser , stocker , partager, analyser, visualiser et de traiter les données provenant de sources disparates (structurées, semi-structurées et non structurées). La capacité de Hadoop à stocker et à analyser de grands ensembles de données en parallèle dans un cluster d'ordinateurs permet de fournir des performances exceptionnelles à un coût très faible.

En se basant sur Hadoop, de nombreux entrepôts de données cloud et géants [94, 102, 103] ont été développés et largement utilisés dans divers domaines [104]. Le rôle de Hadoop dans l'entreposage de données a évolué rapidement. Notamment, dans la phase d'ETL tel qu'il a été largement utilisé comme une plate-forme transitoire pour la phase d'ETL [105], en tenant en compte les étapes suivantes : (1) charger les données dans le système distribué Hadoop (2) exécuter une analyse sur ces données et essayez de trouver une signification (3) nettoyer les données et les transformer en un format structuré en utilisant MapReduce (4) extraire les données de Hadoop et les charger dans l'entrepôt de données.

2.6 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art portant sur les entrepôts de donnée. En nous focalisant plus sur son cycle de vie et les efforts incessants de la communauté pour les développer. Nous avons commencé par donner quelques définitions et notions de base liées aux entrepôts de données. Ensuite, nous avons analysé les architectures les plus dominantes dans le domaine des entrepôts de données, en décrivant les points forts et les critères de sélection de chaque architecture. Ceci est suivi d'une présentation des différentes phases de cycle de vie de la conception d'un entrepôt de données. Finalement, nous avons discuté les efforts les plus intéressants pour développer les EDs, en décrivant les facteurs qui ont conduit à cette évolution.

Dans le prochain chapitre, nous allons présenté les concepts fondamentaux et les tâches liées aux traitement des requêtes dans un système de gestion des bases de données (SGBD) relationnel. Ensuite, nous donnons un état de l'art sur les travaux les plus importants traitant de l'optimisation des requêtes analytiques, en décrivant Les facteurs affectant cette optimisation.

Chapitre 3

Le traitement et l'optimisation des requêtes

"**L'information est le pétrole du 21e siècle et l'analyse est le moteur à combustion**."
— —— Peter Sondergaard

Sommaire

3.1 Introduction

Au cours des deux dernières décennies, le monde a connu une forte augmentation du volume de données constamment générées par divers fournisseurs tel que les entreprises, les administrations publiques, la recherche scientifique, les réseaux sociaux, etc. Ces données doivent être collectées, stockées et analysées afin de prendre des décisions rapides et pertinentes, ce qui nécessite l'exécution rapide d'un nombre important de requêtes analytiques. Pour satisfaire les besoins des décideurs, il est nécessaire d'avoir une compréhension approfondie des propriétés de ces requêtes et une manière efficace d'y traiter en temps opportun.

Le traitement et l'optimisation des requêtes sont l'un des sujets les plus étudiés dans le domaine des bases de données, tel qu'ils jouent un rôle primordial dans la performance de tout SGBD. Les problèmes de traitement et d'optimisation des requêtes sont connus comme des tâches difficiles car ils sont liés aux différentes phases du cycle de vie de conception d'un entrepôt de données. En examinant la littérature, nous recensons une pléthore d'initiatives et de travaux qui se répartissent principalement en trois catégories : (a) les travaux portant sur l'optimisation logique des requêtes, (b) l'optimisation physique des requêtes et (c) les études qui traitent les deux types d'optimisation conjointement.

Dans ce chapitre, nous présentons un état de l'art portant sur les travaux les plus importants traitant de l'optimisation des requêtes OLAP. En premier lieu, nous étudions le processus de traitement de ces requêtes dans un SGBD relationnel. Ensuite, Nous mettons en évidence les propriétés récentes des requêtes analytiques imposées par les exigences actuelles des systèmes décisionnels. Finalement, nous portons une analyse approfondie sur les techniques d'optimisation dédiées aux requêtes OLAP, avec une focalisation particulière sur la réutilisation des sous-expressions communes entre les requêtes et leur rôle dans la résolution des problèmes d'optimisation multi-requêtes et de sélection des vues matérialisées. Au cours de cette étude, nous examinons la capacité des travaux d'état de l'art à gérer simultanément les propriétés actuelles des requêtes analytiques.

3.2 Traitement des requêtes dans un SGBD relationnel

Le traitement des requêtes est l'une des tâches les plus difficiles et les plus importantes dans le succès d'un SGBD. Elle représente l'art scientifique qui permet aux utilisateurs d'obtenir les informations requises d'un SGBD de manière fiable et en temps opportun. Cette tâche est responsable de la traduction d'une requête, généralement écrite dans un langage non procédural de haut niveau (tel que SQL), en un programme d'évaluation efficace pouvant être utilisées au niveau physique du système de fichiers.

Le principal défi auquel est confrontée la tâche de traitement des requêtes est d'améliorer l'efficacité de ses techniques et algorithmes, en particulier ceux liés à l'optimisation des requêtes qui est devenue la tâche la plus complexe avec l'explosion du volume de données, accompagnée du développement des technologies de stockage de données et des plate-formes

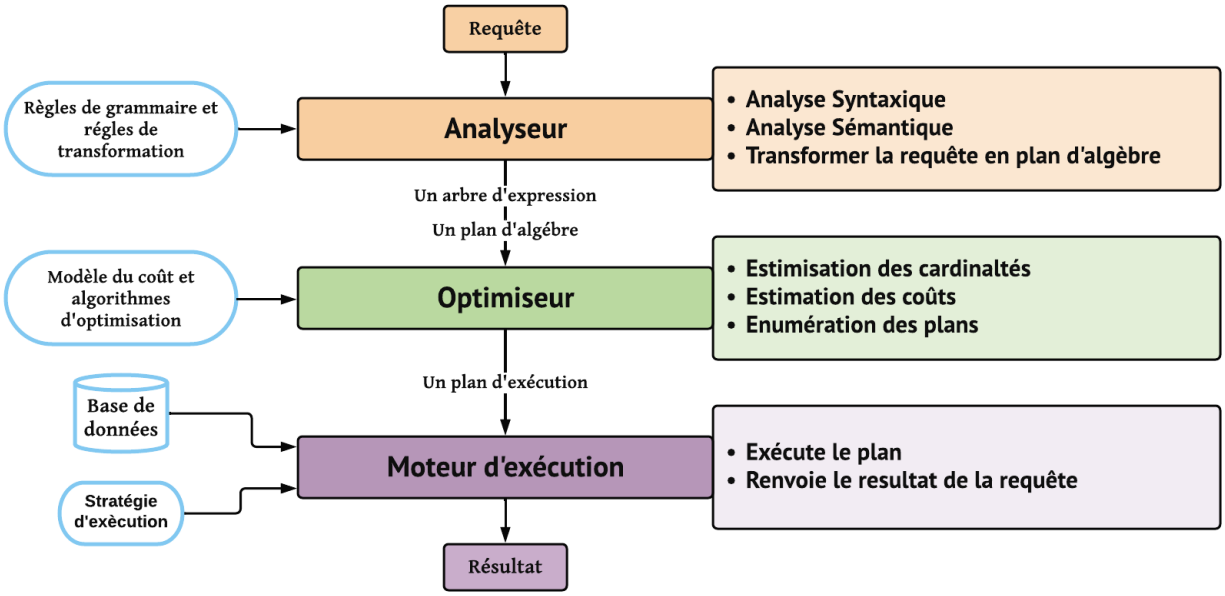


FIGURE 3.1 – Le Processus de traitement des requêtes dans un SGBD relationnel.

de déploiement permettant de gérer les données hétérogènes et massives. Plusieurs initiatives ont été proposé dans littérature pour gérer la tache de traitement des requêtes dans les infrastructures de déploiement avancées tel que : les systèmes multi-bases [106] , les systèmes multi-stores [107, 108], les systèmes hétérogènes CPU-GPU [109], etc. Nous rapportons ici que leurs techniques se sont avérées pratiques et donnent de bonnes performances.

Le traitement des requêtes se décompose en trois étapes principales [110] : (i) l'analyse et la transformation de la requête entrante sous forme d'un arbre d'expression en utilisant les règles d'algèbre relationnelle (ii) l'optimisation de la requête en produisant un bon plan d'exécution et (iii) l'exécution du code de la requête, en mode compilé ou interprété, afin de produire le résultat final. Dans cette section, nous allons détailler les trois étapes du traitement des requêtes en donnant des exemples au besoin. La figure 3.1 donne un aperçu des différentes étapes du processus général de traitement des requêtes.

3.2.1 Analyse et transformation

La première action que le moteur de traitement des requêtes doit effectuer sur une requête soumise à un SGBD, consiste à analyser la requête et de la transformer sous une représentation interne, qui est généralement basée sur l'algèbre relationnelle. Cette étape est accomplie par deux composants principaux [111] :

1. **Un analyseur de requête** : qui construit une arborescence (un arbre d'analyse) à partir de la forme textuelle de la requête. Pour ce faire, une vérification syntaxique est effectuée sur la requête en utilisant un ensemble de règles grammaticales.

2. **Un pré-processeur de requête :** qui effectue des vérifications sémantiques sur la requête. Par exemple, il doit s'assurer que toutes les relations mentionnées par la requête existent réellement dans la base de données, et que les noms des attributs existent véritablement dans les relations impliquées dans la requête. Il permet aussi de transformer l'arbre d'analyse initiale en un arbre d'opérateurs algébriques en utilisant un ensemble de règles de transformations.

3.2.1.1 Analyse Syntaxique de la requête

Cette étape reçoit en entrée un texte écrit dans un langage spécifique tel que SQL et le convertit en un arbre d'analyse. Deux types de nœuds peuvent être distingués dans cet arbre d'analyse [111] :

1. **Les atomes** : qui correspondent aux nœuds feuilles de l'arbre analyse. Les atomes représentent les éléments lexicaux de la requête tels que les mots-clés (e.g. SELECT, WHERE, FROM), les noms des relations , les noms des attributs, les constantes, les opérateurs logiques (AND , OR, etc.) et les opérateurs arithmétiques (< , > , = ,etc.), etc.

2. **les catégories syntaxiques :** représentent les noms des sous-parties de la requête et jouent un rôle similaire à leurs noms descriptifs. Elles sont généralement formées d'un ou plusieurs nœuds atomiques. Les nœuds des catégories syntaxiques sont représentés par des crochets triangulaires autour d'un nom descriptif. Par exemple, la catégorie syntaxique <Requête> sera utilisé pour représenter une requêtes SQL. La catégorie <Condition> représentera toute condition dans la requête SQL (i.e. toute expression qui peut suivre WHERE en SQL).

3.2.1.1.1 Une grammaire pour les catégories syntaxiques de base Pour illustrer le processus d'analyse syntaxique d'une requête SQL, nous présentons ci-dessous quelques règles grammaticales qui décrivent les catégories syntaxiques de base. Pour plus de détails, veuillez vous référer au travail [111].

une requête SQL est représentée par la catégorie syntaxique **<Requête>**, nous lui donnons la règle générique suivante :

$$<Requête> ::= SELECT <ColumnList> FROM <Tables> WHERE <Condition>$$

Les catégories syntaxiques <ColumnList> et <tables> représentent respectivement, les listes qui peuvent suivre SELECT et FROM dans le texte sql, tandis que la catégorie <Condition> représente les conditions SQL (des expressions qui peuvent être vraies ou fausses). Nous donnerons dans ce qui suit quelques règles simplifiées pour ces catégorie syntaxiques.

La catégorie syntaxique <ColumnList> représente la liste des attributs sélectionnés par la requêtes SQL. Cette liste peut être constituée d'un seul attribut, ou de plusieurs attributs séparés par des virgules. La catégorie <ColumnList> est définie comme suit :

$$<ColumnList> ::= <Attribut>,<ColumnList>$$
$$<ColumnList> ::= <Attribut>$$

La catégorie syntaxique <tables> représente la liste des relations qui sont mentionnées dans la clause FROM de la requête SQL. Cette catégorie est définie par les règles suivantes :

$$<Tables> ::= <Relation>,<Tables>$$
$$<Tables> ::= <Relation>$$

Ces deux règles stipulent qu'une liste de relations peut inclure une ou plusieurs relations séparées par des virgules.

La catégorie syntaxique <Condition> représente la liste des conditions SQL qui apparaissent après la clause WHERE dans le text SQL. Nous présentons ci-dessous un ensemble de quelques règles simplifiées. Nous notons également que cet ensemble de règles n'est pas exhaustif en raison de la présence d'un grand nombre de règles liées à cette catégorie.

$$<Condition> ::= <Condition> AND <Condition>$$
$$<Condition> ::= <Attribut> \ IN \ (<Requête>)$$
$$<Condition> ::= <Attribut> = <Attribut>$$
$$<Condition> ::= <Attribut> != <Attribut>$$
$$<Condition> ::= <Attribut> \ LIKE \ \%Motif\%$$

Les catégories syntaxiques <Relation>, <Attribut> et <Motif> ne sont pas définies par des règles grammaticales, car elles peuvent être remplacées par n'importe quelle chaîne de caractères [111].

Exemple 3.2.1 *Pour mieux comprendre le processus d'analyse syntaxique d'une requête SQL, nous considérons la requête suivante Q_i définie sur le schéma en étoile d'un entrepôt de données pour la gestion des achats d'une entreprise commerciale. Le schéma de cet entrepôt de données est illustré dans la figure 2.6. Il est constitué d'une table de faits Achats et quatre tables de dimension : Fournisseur, Contrat, Dates et Produit. Supposons que les gestionnaires de l'entreprise veulent savoir « les marques des produits achetés dont le prix unitaire dépasse* 200 *€» . Nous répondons à ce besoin en formulant la requête SQL suivante :*

La Requête Q_i

```
SELECT  marque
FROM  Produit, Achats
WHERE  Achats.id_produit = Produit.id_produit
AND    prix > '200'
```

La figure 3.2 montre l'arbre d'analyse qui a été généré pour la requête Q_i en utilisant les règles de grammaire présentées ci-dessus. .

3.2.1.2 Analyse Sémantique de la requête

Après avoir terminé l'analyse syntaxique, et dans le cas où la requête est syntaxiquement valide, la prochaine étape consiste à valider sémantiquement la requête. Ceci est fait par le

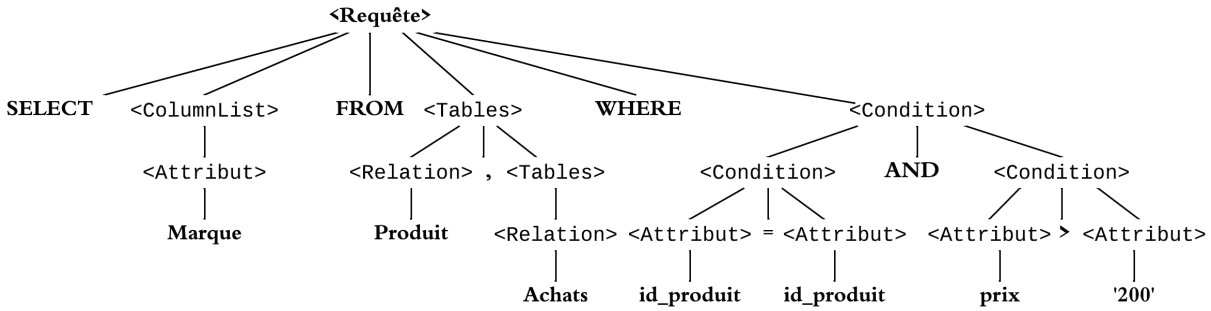


FIGURE 3.2 – L'arbre d'analyse de la requête Q_i .

pré-processeur de requêtes qui a la responsabilité de vérifier plusieurs règles sémantiques. Nous citons par exemple :

1. **La vérification des relations :** vérifier que toutes les relations mentionnées après la clause <FROM> font référence à des relations ou à des vues dans le schéma courante de la base de données.

2. **La vérification des attributs :** Chaque attribut mentionné dans les clauses SELECT ou WHERE doit être un attribut de l'une des relations mentionnées après la clause FROM de la requête actuelle.

3. **La vérification des types :** Chaque attribut doit avoir un type approprié pour son usage. Par exemple, l'attribut prix est utilisé dans la requête Q_i dans une comparaison <, cela nécessite que le type de cet attribut soit un nombre , une date ou une chaîne (la comparaison se fait par ordre alphabétique).

3.2.1.3 Transformation de la requête

Dans cette sous section, nous présentons une énumération des règles algébriques qui permettent de transformer un arbre d'analyse en un arbre d'expression équivalent, qui peut avoir par la suite un impact significatif dans la construction d'un plan physique plus efficace. Le résultat de cette étape de transformation est un plan logique de la requête.

La génération du plan logique d'une requête représenté sous la forme d'un arbre d'analyse, passe par deux étapes principales : (1) la transformation de l'arbre d'analyse en un plan logique de requête et (2) l'amélioration du plan initiale en utilisant les règles algébriques. Le processus de transformation d'un arbre d'analyse en un plan logique s'effectue en respectant les règles suivantes :

- ☐ Les relations sont extraites à partir de <Tables> de l'arbre d'analyse.
- ☐ Les prédicats de sélections σ_{cond} correspondent à chaque condition capturé à partir des <Condition> de l'arbre d'analyse.

- ☐ Les projections π_{liste} correspondent à chaque attribut présent dans la liste "$liste$" capturée à partir de <ColumnList>.
- ☐ le plan logique initial correspond au produit cartésien des relations, deux à deux, en second lieu une sélection σ_{cond} , et enfin une projection π_{liste}.

3.2.1.3.1 Règles de transformation et d'amélioration du plan Pour générer un plan de requête plus efficace et équivalent à l'arbre d'analyse initial, l'optimiseur de requêtes doit savoir quelles règles de transformation préservent cette équivalence. Pour cela, nous discutons dans ce qui suit quelques règles de transformation algébriques [112]. Nous nous concentrons ici sur les opérations de projection, de sélection et de la jointure. Pour une énumération exhaustive sur ces règles de transformation et une description détaillée sur le processus de conversion d'un arbre d'analyse en algèbre relationnelle, veuillez vous référer aux travaux [111, 112].

1. **Décomposition en cascade de** σ : Une condition de sélection conjonctive (composée d'une conjonction de plusieurs sélections élémentaire) peut être décomposée en une séquence d'opérations individuelles :

   $\sigma_{\ cond1\ AND\ cond2\ AND...\ condn}(R) \equiv \sigma_{\ cond1}(\sigma_{\ cond2}(...(\sigma_{\ condn}(R))...))$

   Où, R : est une relation.

   $cond_i$: est la i^{eme} condition de sélection

2. **Commutativité de sélection** : l'opérateur de sélection σ est commutative.

   $\sigma_{\ cond1}(\sigma_{\ cond2}(R)) \equiv \sigma_{\ cond2}(\sigma_{\ cond1}(R))$

3. **Décomposition de l'opérateur de projection** π : dans une séquence d'opérations de projection π, toutes les opérations de projection peuvent être ignorées sauf la dernière.

   $\pi_{\ liste1}(\pi_{\ liste2}(...\pi_{\ (listen}(R))... \equiv \pi_{\ liste1}(R)$

   Où, $liste_i$: est la i^{eme} liste des attributs sélectionnés.

4. **Commutativité de** σ **avec** π **:** on peut commuter les deux opérateur σ et π, si la condition de sélection $cond$ n'est liée que aux attributs $Att_1, ...Att_n$ dans la liste de projection.

   $\pi_{\ Att_1,\ Att_2,...\ Att_n}(\sigma_{\ cond}(R)) \equiv \sigma_{\ cond}(\pi_{\ Att_1,\ Att_2,...\ Att_n}(R))$

5. **Commutativité de produit cartésien :** le produit cartésien de deux relations $R1$ et $R2$ est commutative.

   $R1 \text{ X } R2 \equiv R2 \text{ X } R1$

6. **Commutativité de la jointure :** l'opération de jointure de deux relations $R1$ et $R2$ est commutative.

   $R1 \bowtie R2 \equiv R2 \bowtie R1$

7. **Commutativité de sélection σ avec la jointure $\bowtie$:** ont dit qu'une opération de sélection est σ est commutative avec une jointure $\bowtie$ si l'ensemble des attributs de la condition de sélection $cond$ n'est lié qu'aux attributs d'une des relations jointes (nous supposons ici que cet ensemble est lié aux attributs de la relation R1).

   $\sigma_{cond}(R1 \bowtie R2) \equiv (\sigma_{cond}(R1)) \bowtie R2$

8. **Commutativité de projection π avec la jointure $\bowtie$:** considérons une liste de projection L composée de plusieurs attributs $L = \{A_1, A_2, ...A_n, B_1, B_2, ...B_m\}$, où $\{A_1, A_2, ...A_n\}$ sont les attributs de R1 et $\{B_1, B_2, ...B_m\}$ sont les attributs de R2. On dit que la projection π est commutative avec la jointure $\bowtie$, si la condition de jointure $cond$ n'implique que les attributs de la liste L

   $\pi_L(R1 \bowtie_{cond} R2) \equiv (\pi_{A_1,A_2,...A_n}(R1)) \bowtie_{cond} (\pi_{B_1,B_2,...B_m}(R2))$

Pour améliorer les plans logiques des requêtes , les optimiseurs appliquent un ensemble de règles. Nous présentons ci-dessous les règles couramment utilisées [111] :

- ☐ L'une des techniques d'amélioration les plus efficaces, consiste à exécuter l'opérateur σ_{cond} dès que possible. En termes de plan logique de requête , cela signifie que l'opération de sélection σ_{cond} est poussée vers le bas le plus loin possible.
- ☐ La projection peut être poussée vers le bas comme la sélection afin de minimiser la taille de la relation en réduisant la taille des tuples, puis en minimisant le coût du traitement des requêtes.
- ☐ Les paires sélection/produit cartésien peuvent être parfois combinés et remplacés par une opération de jointure, ce qui est généralement plus rapide que de faire un produit suivi d'une sélection sur le résultat très large du produit.
- ☐ Trouver l'ordre de jointure optimal pour un ensemble de jointures $\bowtie$ est l'un des principaux objectifs des optimiseurs de requêtes. Étant donné qu'un ordre différent dans l'exécution des opérations de jointure $\bowtie$ peut entraîner une grande différence dans la taille des résultats intermédiaires.

Exemple 3.2.2 *Nous donnons ici un exemple d'utilisation des règles d'équivalence et d'optimisation décrites ci-dessus. Pour ce faire , nous considérons la requête Q_i définie sur un entrepôt de données pour la gestion des achats (exemple3.2.1) et son arbre d'analyse (voire figure 3.2). Un plan de requête initial a été construit et illustré dans la figure 3.3a. Nous voulons dire par δ l'opérateur d'élimination des doublons. Le plan initial peut être transformer en un plan équivalant mais plus efficace en appliquant quelques règles de transformation sur la requête Q_i (voir figure 3.3b).*

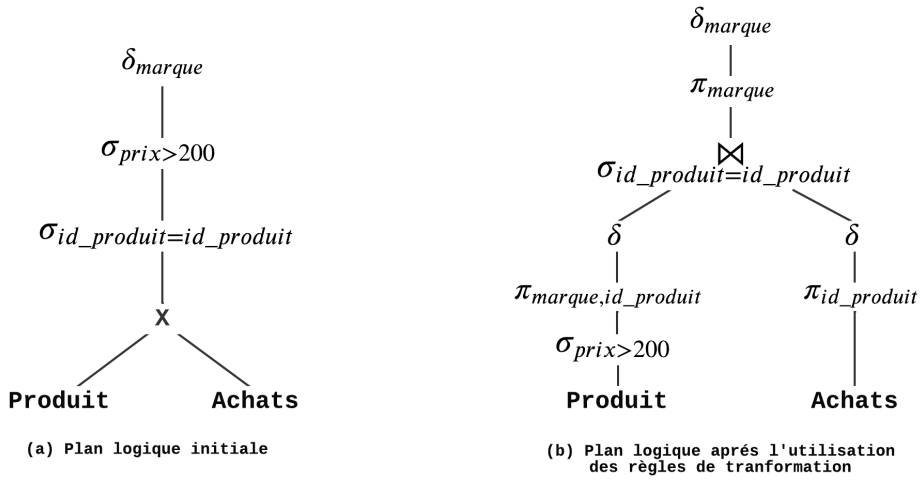


FIGURE 3.3 – Transformation d'un plan logique d'une requête.

3.2.2 Génération du plan physique

Après avoir analysé une requête et la transformée en un plan de requête logique, nous devons ensuite transformer le plan logique en un plan physique de requête. Cela est fait en sélectionnant les algorithmes appropriés pour implémenter chaque opérateur du plan logique, et en sélectionnant un ordre d'exécution pour ces opérateurs. Généralement, plusieurs plans physiques peuvent être dérivés du plan logique de requête. Ensuite, un processus de sélection est effectué sur l'ensemble des plans physiques pour choisir le meilleur plan en termes de coût de traitement, ce qu'on appelle un processus d'énumération des plans basée sur les coûts [113]. L'estimation du coût d'un plan physique implique l'estimation du coût d'exécution de chaque opérateur et les combiner à la fin pour calculer le coût global de la requête. Le coût de traitement des requêtes est estimé en se basant sur deux paramètres principaux : (1) Le coût d'entrée/sortie qui représente le temps de chargement des données depuis les périphériques de stockage (e.g. Disque dur) et (2) Le coût CPU pour traiter chaque opérateur [114].

La génération d'un plan physique optimale pour une requêtes données est l'une des tâches les plus difficiles pour un optimiseur de requêtes. Cette tâche est liée à deux problèmes importants et connexes : l'énumération des plans physiques possibles et l'estimation du coût de chaque plan d'exécution [115].

3.2.2.1 Approches d'énumération des plans

Énumérer tous les plans possibles pour les requêtes complexes est un grand défi pour l'optimiseur de requêtes. Cette tâche peut être coûteuse car une quantité énorme de plans physiques potentiels peut être générée, ce qui complique les tâches futures telles que l'évaluation des plans et le choix du meilleur en termes de coût. Par conséquent, les approches

d'énumération des plans de requête tentent d'éliminer rapidement les plans qui ne peuvent pas conduire aux meilleurs coûts estimés. En principe, il existe deux stratégies principales pour énumérer les plans physique d'une requête [116] :

1. **Stratégie Descendantes** : l'énumération des plans dans cette stratégie est appliquée de manière récursive. En partant de la racine du plan logique de la requête, chaque possibilité de calculer les opérateurs est considérée en appliquant les règles d'équivalence, les coûts de chaque possibilité sont calculés, et la meilleure est choisie. cette stratégie est utilisée par Microsoft SQL Server, Cascades, Volcano et Tandem [117].

2. **Stratégie ascendante** : en commençant par les feuilles d'un plan logique de requête vers la racine. Pour chaque sous-expression S_{exp} du plan logique, toutes les possibilités de calcul de la sous-expression S_{exp} sont énumérées en fonction des règles d'équivalence et leurs coûts sont déterminés en combinant les plans de requête possibles déjà déterminés pour la sous-expressions de S_{exp}. Cette stratégie est adopté par Oracle, DB2, System R et Informix [117].

Les deux types de stratégies énumèrent tous les plans possibles d'une requête. Cependant, les stratégies ascendantes sont plus populaires, car elles peuvent écarter les plans à coûts élevés plus tôt [116]. Dans ce qui suite, nous décrivons les approches les plus importantes pour l'énumération des plans [111, 116].

- **Stratégie de séparation et évaluation (Branch and Bound)** Dans cette approche, les plans possibles d'une requête sont énumérés à l'aide de la stratégie ascendante. Si les coûts estimés d'une sous-expression énumérée sont déjà supérieurs à la limite supérieure déterminée pour les coûts, cette sous-expression peut être ignorée car tous les plans de requête contenant cette sous-expression auront des coûts plus élevés que le meilleur actuellement. Si nous trouvons un plan de requête avec des coûts inférieurs, alors ce plan de requête devient le meilleur actuellement et ces coûts inférieurs deviennent la nouvelle limite supérieure. L'avantage de cette stratégie est que l'optimiseur de requêtes peut s'arrêter une fois qu'un plan à très faible coût a été trouvé, de sorte que le temps consacré à l'optimisation des requêtes ne domine pas les coûts globaux de traitement des requêtes.
- **Stratégie d'escalade (Hill Climbing)** Cette approche calcule également d'abord un plan de requête en utilisant une bonne heuristique. Ensuite, l'approche tente de réduire les coûts de ce plan de requête en apportant de petites modifications au plan de requête. Si de petites modifications ne réduisent pas les coûts, le plan de requête actuel est choisi. De cette façon, les bons candidats pour les plans de requête à faible coût sont choisis rapidement. À l'instar de l'approche d'énumération des plans de séparation et évaluation, l'escalade peut être arrêtée lorsqu'un plan de requête à très faible coût est déjà trouvé. Cependant, comme de grands changements ne sont pas apportés au plan de

requête, tous les plans de requête possibles ne sont pas énumérés, et l'escalade peut donc manquer des plans de requête optimaux.

- ☐ **Stratégie de programmation dynamique** La programmation dynamique est une variante de la stratégie ascendante générale. Pour chaque sous-expression initiale, tous les plans possibles sont énumères, en gardant le plan le moins coûteux puis on monte dans l'arbre jusqu'à le nœud racine. Le principal avantage de cette approche est qu'elle élague dynamiquement l'espace de recherche.

3.2.2.2 Estimation du coût

Le coût réel du traitement d'une requête est estimé généralement en unité de temps (par exemple, en seconde) qui correspond à la consommation de temps depuis le début du traitement de la requête jusqu'à ce que le résultat soit donné par le SGBD. L'estimation du coût de traitement des requêtes implique l'estimation du coût d'exécution de chaque opérateur, qui à son tour implique l'estimation de cardinalité de ces opérateurs. La cardinalité d'un résultat intermédiaire ou d'une relation représente le nombre de ses tuples. La qualité des fonctions d'estimation de cardinalité a un impact direct sur la qualité d'un modèle de coût. De nombreux travaux ont montré qu'une mauvaise estimation de cardinalité implique le choix d'un mauvais plan d'exécution qui peut être des milliers de fois plus lent [118, 119].

L'estimation de cardinalité d'un résultat intermédiaire nécessite la disponibilité des statistiques sur la distribution des données dans les relations de base. Ainsi, pour estimer la cardinalité d'un résultat intermédiaire, tout optimiseur de requête utilise des outils pour obtenir et mettre à jour les statistiques de distribution des données. Ensuite, il applique des fonctions sur ces statistiques en fonction des paramètres de requête. La qualité d'un optimiseur de requête est étroitement liée à la qualité du modèle de coût, qui est, à son tour, basée sur l'estimation de la cardinalité. En conséquence, de mauvaises statistiques peuvent facilement altérer le plan d'exécution, et la disponibilité de bonnes statistiques peut facilement améliorer la qualité du plan [120]. Aujourd'hui, la plupart des SGBD existants disposent de catalogues statistiques [121, 122]. Les statistiques sont calculées juste après la mise à jour des relations de base ou périodiquement par le SGBD. Ces statistiques peuvent être classées en trois catégories :

- la distribution des valeurs d'un attribut, comme le nombre de tuples, la taille moyenne de la colonne ou de tuple, les valeurs distinctes et les valeurs maximales et minimales d'un attribut. Afin de réduire la complexité du problème de calcul des statistiques, de nombreux SGBD supposent qu'il existe une distribution uniforme des valeurs [123, 124]. Pour ce faire, un SGBD peut utiliser les histogrammes afin de sauvegarder la distribution des valeurs d'un attribut. Les types d'histogrammes les plus courants [125] sont les histogrammes de largeur égale, de profondeur égale et de valeurs les plus fréquentes. La figure 3.4 illustre des exemples d'histogrammes de distribution des valeurs de l'attribut : prix de la table produit.

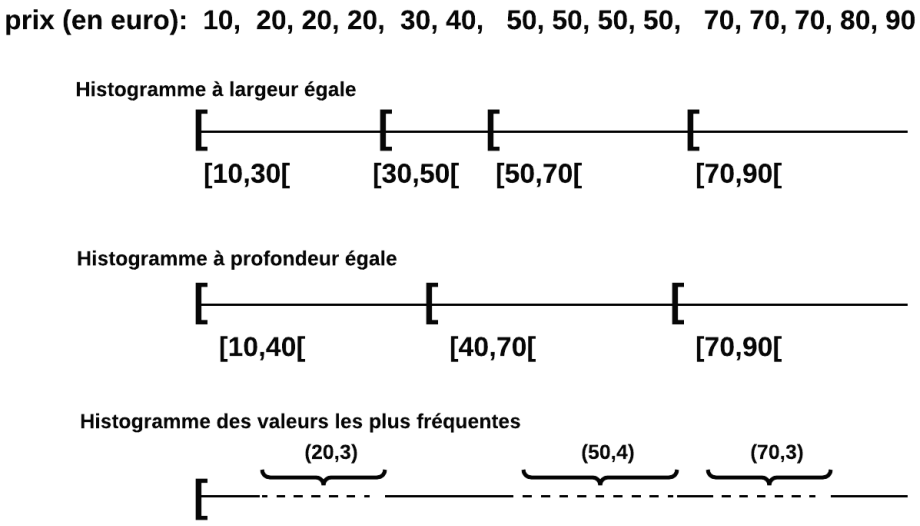


FIGURE 3.4 – Différents types d'histogramme.

- Les paramètres matériels qui incluent le réseau, les caractéristiques du disque dur (par exemple, la bande passante et le temps de lecture d'une page).
- Les statistiques d'utilisation des méthodes d'accès (index, vues matérialisées, etc.), comme le temps passé à parcourir un index.

3.2.2.3 L'ordre de jointure

Le problème d'ordre de jointure joue un rôle crucial dans la performance du traitement des requêtes, notamment dans l'optimisation des requêtes de jointure en étoile impliquant des tables de dimension et une table de faits. L'ordre de jointure détermine la taille des résultats intermédiaires [126] ce qui constitue un facteur important qui influence directement les performances du traitement des requêtes. Par conséquent, un bon plan d'exécution dépend fortement de l'ordre des jointures. Plusieurs structures ont été proposé dans la littérature pour représenter un arbre d'une requête impliquant plus de deux opérations de jointures. Les structures suivantes sont les structures couramment utilisées : arbre de jointure linéaire gauche [127], arbre de jointure linéaire droite [128], arbre de jointure linéaire en zigzag [129] et l'arbre de jointure ramifié (bushy) [130].

Le problème d'ordre de jointure dans sa forme générale est prouvé NP-complet [131], en raison de la croissance exponentiel de l'espace de recherche par rapport au nombre des relations de la requête de jointure. Par exemple, 5 ou 6 relations de jointure pourrait créer un espace de recherche dont la taille dépasse 1000 plans de requête différents. Le problème d'ordre de jointure a été largement étudié par la communauté des base de données en proposant des . Ces étude sont classifié en quatre classes principales d'algorithmes [132] : (i)

algorithmes déterministes (ii) algorithmes randomisés (iii) algorithme génétique et (iv) algorithmes hybrides qui combinent les éléments de deux ou plusieurs algorithmes. Pour plus de détail sur ces algorithmes referez-vous à la documentation [132] qui donne une bonne classification et description sur les algorithmes d'ordre de jointure. Au cours des dernières années, les techniques d'apprentissage automatique ont été largement utilisées dans le cadre de la sélection du meilleur ordre de jointure d'une requête analytique de jointure [133, 134].

3.2.3 Exécution du plan

Cette étape est réalisée par le moteur d'exécution qui se charge d'exécuter chaque opérations du plan physique choisi. Le moteur d'exécution interagit avec la plupart des composants du SGBD. Les données sont placées dans des tampon mémoire afin d'être manipuler par le moteur d'exécution, qui doit interagir avec le planificateur pour éviter d'accéder aux données verrouillées. Il doit également interagir avec le gestionnaire de journaux (les logs), pour s'assurer que toutes les modifications de la base de données sont correctement enregistrées [111]. L'exécution d'un opérateur du plan physique peut être effectuée en mode matérialisé ou parallèle.

3.2.3.1 Mode matérialisé

Dans le mode d'exécution matérialisée, le résultat d'une opération est stocké sous la forme d'une relation temporaire (c'est-à-dire que le résultat est physiquement matérialisé). Par exemple, une opération de jointure peut être calculée et le résultat entier sera stocké sous forme d'une relation temporaire et sera utilisée à l'évaluation des opérations de niveau suivant. [112]. Le coût de ce mode d'exécution est toujours plus élevé, car il faut du temps pour évaluer et écrire la relation temporaire, puis pour récupérer les résultats de cette relation.

3.2.3.2 Mode parallélisé

Le concept général de parallélisme vise à diviser une grande tâche en tâches plus petites, où chaque petite tâche est affectée à une machine pour accomplir une partie de la tâche principale. L'exécution d'un plan d'une requêtée en mode parallèle, peut être réalisé en exécutant plusieurs requêtes sur différents sites ou en divisant une requête en plusieurs sous-requêtes qui s'exécutent en parallèle, cela contribue à améliorer les performances [112]. Le parallélisme d'une requête peut être réalisé en deux modes : le parallélisme inter-requête et le parallélisme intra-requête [135].

3.2.3.2.1 Parallélisme intra-requête C'est la forme de parallélisme où une seule requête est exécuté en parallèle sur plusieurs processeurs. le principal avantage de ce type de parallélisme est d'accélérer l'exécution des requêtes complexes. Ce modèle de parallélisme est scindé en deux types : parallélisme intra-opérateur et parallélisme inter-opérateur.

1. **parallélisme intra-opérateur** Le parallélisme intra-opérateur est basé sur la décomposition d'un opérateur en un ensemble de sous-opérateurs indépendants, appelés instances d'opérateurs. Cette décomposition se fait à l'aide du partitionnement des relations. Chaque instance d'opérateur traitera alors une partition de relation. La décomposition de l'opérateur bénéficie fréquemment du partitionnement initial des données (par exemple, les données sont partitionnées sur l'attribut de jointure) [135].

2. **parallélisme inter-opérateur** Le parallélisme inter-opérateurs, signifie que plusieurs opérateurs au sein d'une requête peuvent être exécutés en parallèle. Cela peut être réalisé soit par l'exécution parallèle d'opérations indépendantes, soit en pipeline [136]. Dans le parallélisme indépendant, les opérations indépendantes d'une requête peuvent être exécutées en parallèle. Ce parallélisme est très utile dans le cas du degré de parallélisme inférieur. Dans le parallélisme en pipeline, il est possible d'exécuter deux opérations simultanément sur des processeurs différents, de sorte qu'une opération consomme des tuples en parallèle avec une autre opération, les réduisant. ce type de parallélisme est utile pour le petit nombre de processeurs et évite l'écriture des résultats intermédiaires sur le disque.

3.2.3.2.2 Parallélisme inter-requête Lorsque plusieurs requêtes sont soumises à un SGBD, le parallélisme inter-requêtes permet de les exécuter en parallèle sur plusieurs processeurs pour améliorer la performance globale des requêtes [137]. L'inter-requête est très efficace pour la prise en charge des applications de traitement transactionnel (OLTP), où un grand nombre de requêtes oltp peuvent être exécutées en quelques secondes [138]. Le principal défi pour le parallélisme inter-requêtes est l'allocation des processeurs, où chaque requête peut impliquer plusieurs jointures et peut prendre plusieurs processeurs pour s'exécuter [139]. Pour résoudre ce problème, plusieurs initiatives ont été proposé dans la littérature [140, 141]. Oracle et DB2 sont deux exemples sur les SGBDs qui supportent le parallélisme inter-requêtes.

3.3 Optimisation des requêtes analytiques : exploitation des sous-expressions communes

Dans les situations critiques, la prise de décisions rapides et précises nécessite une exécution rapide d'un grand nombre de requêtes de navigation et d'exploration de données qui sont généralement collectées et stockées dans des entrepôts de données. Pour satisfaire l'exigence du décideur, il est nécessaire d'étudier en profondeur les différentes propriétés de ces enquêtes.

Le partage de sous-expressions communes est l'une des propriétés importantes des requêtes analytiques. L'identification des sous-expressions les plus bénéfique pour un ensemble de requêtes dans le but de les optimiser est connue sous le nom de problème de sélection de sous-expressions.[142]. Le problème de la sélection des sous-expressions est fortement

connectés à divers problèmes importants que nous pouvons diviser en deux catégories principales : (i) les problèmes gérés à l'intérieur d'un SGBD tels que la réécriture de requêtes [9] , le problème d'optimisation multi-requêtes (PMQO)[143], la sélection des vues matérialisées [20] et (ii) les problèmes gérés à l'extérieur d'un SGBD comme l'optimisation des tâches massives de données [144] et les tâches de cloud computing [145].

Les problèmes d'optimisation multi-requêtes et de sélection des vues matérialisées sont les problèmes les plus importants qui utilisent des sous-expressions de requêtes. Dans cette section, nous présentons une études sur les propriétés des requêtes analytiques qui s'exécutent sur les applications modernes. Ensuite, nous passons en revue les principales études traitant de l'optimisation multi-requêtes et de sélection des vues matérialisées de manière isolée et conjointe et leur rôle dans la résolution d'autres problèmes importants.

3.3.1 Propriétés des requêtes analytiques

De nos jours, les applications modernes nécessitent des solutions efficaces pour traiter un grand nombre de requêtes d'analyse. Pour illustrer ce point, quatre exemples sont donnés : (i) la plateforme snowflake traite plus de 30 millions de requêtes par jour de sa clientèle [1] (ii) le système SQLShare – une expérience SQL en tant que service pluriannuelle, dans laquelle 24 275 requêtes existent [146]. (iii) les résultats obtenus d'un papier récent publié à VLDB'2020, traitant du problème de sélection des vues matérialisées dans le SGBD Oracle, sur la base de 650 requêtes s'exécutant sur un schéma en étoile [9], et (iv) les journaux de requêtes SPARQL exécutés sur les données scientifiques de DBpedia contiennent 43 284 requêtes [147].

Les efforts d'augmentation des entrepôts de données ne doivent pas ignorer le nombre énorme de requêtes qui impacte fortement l'optimisation logique et physique des requêtes. En plus de leur nombre énorme, les requêtes analytiques sont également connus par les deux propriétés suivantes : (2) dynamiques et (3) partagent des sous-expressions communes. Par une analyse rapide de ces propriétés, nous constatons que les deux premières propriétés sont factuelles alors que la dernière est comportementale (voir la figure 3.5).

Ces trois propriétés ne reçoivent pas le même degré d'attention de la part de la communauté des chercheurs. Le partage des sous-expressions communes est une propriété directrice et en même temps impactée par les deux autres propriétés. Le partage des opérations entre plusieurs requêtes simultanées a été étudié pour la première fois par Sellis en 1988 dans le contexte de l'optimisation multi-requêtes (PMQO) [143]. En 2018, ce principe a été reproduit dans le cadre des bases de données Cloud sous le nom de « Pay One, Get Hundreds for Free » [148]. Ce phénomène de partage ne cesse de croître, nous aimerions ici souligner une découverte importante rapportée dans [21] selon laquelle plus de 18 % des requêtes de charges réelles de certains projets du Cloud d'Alibaba sont redondantes. Cela représente un nombre élevé du point de vue du partage.

1. https://diginomica.com/snowflake-ceo-insists-his-company-can-take-heat

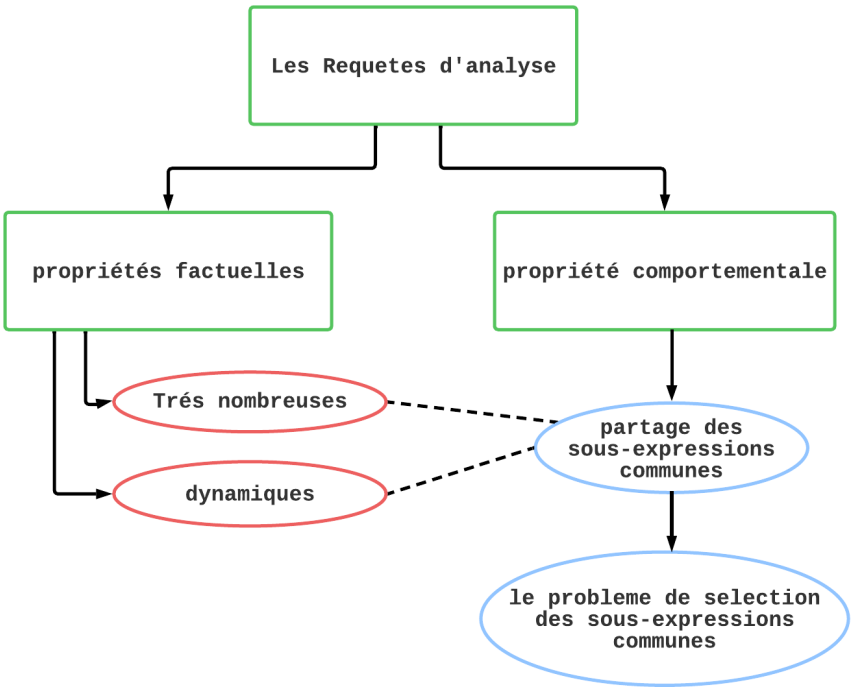


FIGURE 3.5 – Les propriétés des requêtes analytiques d'aujourd'hui

En ce qui concerne l'aspect dynamique des requêtes analytiques (OLAP), la diversité de l'analyse des données et l'évolution des directives métier rendent la charge des requêtes plus dynamique [149]. Ce dynamisme a contribué à multiplier les travaux de recherche sur les bases de données auto-adaptatives et autonomes [9, 10, 150]. Plusieurs approches réactives telles que DynaMat [151], WATCHMAN [152] et le conseiller (MQT) ont été proposés pour la sélection dynamique de vues matérialisées.

3.3.2 Problème d'optimisation multi-requêtes (PMQO)

les requêtes peuvent être exécutées séparément ou conjointement. Dans ce dernier cas, l'identification des sous-expressions communes des requêtes est l'enjeu clé pour la performance d'optimisation multi-requêtes . Le PMQO est l'un des principaux sujets étudiés par la communauté des bases de données [153]. Un chapitre spécifique sur le PMQO a été réservé dans l'encyclopédie des systèmes de bases de données éditée par Ling Liu et Tamer Ozsu [154]. Historiquement, le PMQO a été largement étudié dans les années 80 [143, 155], où Mr.sellis a identifié un nouveau phénomène connu par interaction de requêtes et a donné naissance à un nouveau problème appelé optimisation multi-requêtes (MQO). Le PMQO a été initialement étudie dans le cadre des bases de données relationnelles [156], puis dans toutes les générations de bases de données sans aucune exception. Le PMQO peut être formalisée comme suit : étant donné une charge de requêtes $Q1, Q2, ..., Qn$ à optimiser, où chaque requête Qi possède un ensemble de plans individuels possibles, le PMQO consiste à trouver

un plan global de requête en fusionnant les plans individuels, de sorte que le coût de traitement de toutes les requêtes est minimisé.

Le PMQO a été prouvé NP-difficile dans [15] où sa formulation d'espace de recherche a été donnée. Un algorithme A^* avec des fonctions de délimitation et une expansion intelligente des états (basée sur l'ordre des requêtes), pour éliminer rapidement les états peu prometteurs a été proposé par Timos Sellis à [143] et il a garantit une solution optimale, pour une petite charge de requêtes. Plusieurs variantes de l'algorithme de Sellis ont été proposées [15] pour élaguer l'espace de recherche du PMQO. La génétique, le recuit simulé et les algorithmes par séparation et évaluation ont également été analysés expérimentalement pour traiter le PMQO en considérant des charges de requêtes plus grandes [157, 158]. D'autres algorithmes sont basés sur la théorie des jeux [159]. En 2018, les auteurs de [153] ont abordé le problème de l'extension des optimiseurs de requêtes basés sur les coûts pour prendre en charge l'optimisation multi-requêtes, en proposant des heuristiques gourmandes et des techniques d'optimisations. Leurs techniques se sont révélées pratiques et donnaient de bons résultats.

L'adaptation de ces résultats a été reproduite dans les différentes générations de bases de données à travers leurs langages de requêtes : les bases de données orientées objet (OQL) [160], les bases de données sémantiques (SPARQL) [161],les bases de données XML (XPATH) [162], les bases de données distribuées [163, 164], les bases de données de flux (CQL) [165], les bases de données de graphes (Sparql 1.1) [166], data mining [167] et les systèmes SQL-on-Hadoop [168].

Dans la littérature, le PMQO a été combiné avec plusieurs problèmes importants dans le domaine des bases de données tels que : la mise en cache [169], la sélection des vues matérialisées [170], la réutilisation [20], l'indexation [171], le partitionnement des données [172], l'optimisation des requêtes exploratoires [163]. Dans le contexte des entrepôts de données relationnelles, le PMQO a été amplifié en raison de la nature entrelacée des requêtes OLAP qui ont une **forte interaction** entre elles. La théorie des graphes a largement contribué à la résolution du PMQO à travers ses structures de données avancées qui ont été utilisée pour représenter le plan unifié des requêtes (UQP) en utilisant les graphes de connexion (les graphes de requêtes) [155], ou pour traiter le PMQO à grande échelle comme les hypergraphes qui ont utilisée pour capturer l'interaction entre un grand nombre de requêtes tout en assurant l'évolutivité (le passage à l'échelle) du système [92, 173]. La figure 3.6 donne un aperçu des problèmes les plus importants liés à PMQO et montre également les algorithmes et les structures de données utilisées pour résoudre ce problème.

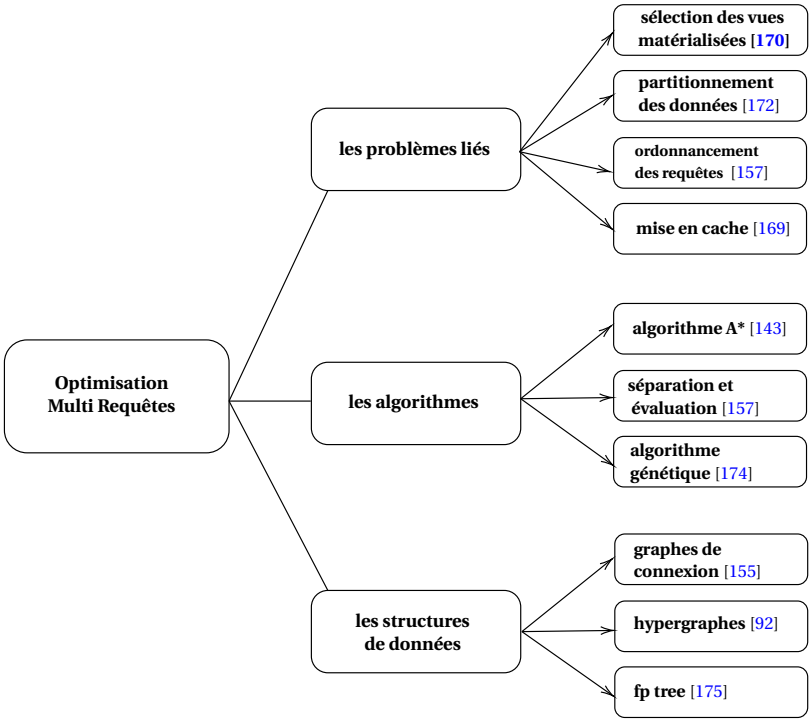


FIGURE 3.6 – Vue d'ensemble des facteurs liés au PMQO

Exemple 3.3.3 *Pour illustrer les avantages de l'utilisation des techniques d'optimisation multi-requêtes, nous considérons les deux requêtes suivantes exprimées en SQL :*

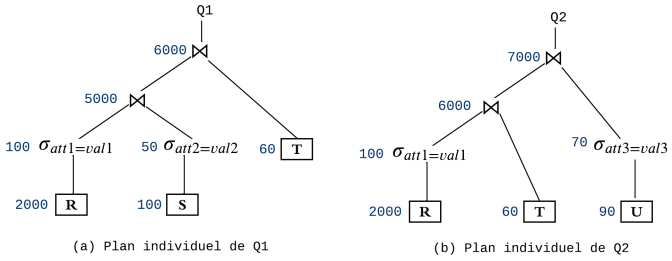
La Requête Q_1

```
SELECT  *
FROM  R, S, T
WHERE R.key1=S.key1
 AND  R.key2=T.key2
 AND  R.att1=val1
 AND  S.att2=val2
```

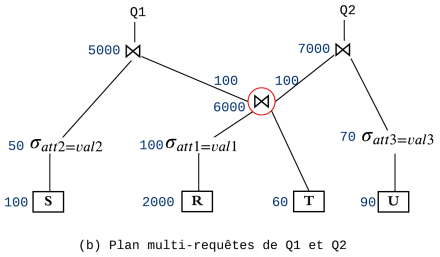
La Requête Q_2

```
SELECT  *
FROM  R, T, U
WHERE R.key2=T.key2
 AND  R.key3=U.key3
 AND  R.att1=val1
 AND  U.att3=val3
```

Nous supposons que les relations de base R, S, T et U ont respectivement un coût de balayage de 2000, 100, 60 et 90 unités et nous supposons l'absence de toute structure d'optimisation. Un système traditionnel exécutera chacune de ces requêtes indépendamment en utilisant les meilleurs plans individuels suggérés par l'optimiseur de requêtes. Nous supposons que les meilleurs plans de Q1 *et* Q2 *sont représentés par la figure 3.7a .*



(A) les plans individuels de Q1 et Q2



(B) le plan global de Q1 et Q2

FIGURE 3.7 – Three simple graphs

Les meilleurs plans pour les deux requêtes Q1 et Q2 sont représentés respectivement par les expressions algébriques suivantes : $(\sigma_{att1=val1}(R) \bowtie \sigma_{att2=val2}(S)) \bowtie T$ *et* $(\sigma_{att1=val1}(R) \bowtie T) \bowtie \sigma_{att3=val3}(U)$ *, le coût de chaque plan est calculé par la somme des coûts de chaque opération.*

coût(Q1)= $\sum$ *coût(op) = 2000+100+100+50+60+5000+6000= 13310 pages (en supposant que la taille de résultat intermédiaire de la jointure* $(\sigma_{att1=val1}(R) \bowtie \sigma_{att2=val2}(S))$ *= 100 pages)*

coût(Q2)= $\sum$ *coût(op) = 2000+60+90+100+70+6000+7000= 15320 pages (en supposant que la taille de résultat intermédiaire de la jointure* $(\sigma_{att1=val1}(R) \bowtie T)$ *= 100 pages)*

Le coût total des deux requêtes avec l'optimisation individuelle est égale à 28630 pages. Cependant, on remarque que le résultat intermédiaire $(\sigma_{att1=val1}(R) \bowtie T)$ *peut être partagé par les deux requêtes comme le montre la figure 3.7b. En utilisant les techniques d'optimisation multi-requêtes le coût total du plan global des requêtes est égale à 20670 pages, ce qui réduit le coût d'exécution totale d'environ 28 %.*

3.3.3 Problème de sélection des vues matérialisées (PSV)

Au cours des dernières années, il y a eu un intérêt croissant pour la résolution des problèmes de conception physique des entrepôts de données, en particulier dans la sélection de vues matérialisées, qui sont considérées comme l'une des techniques les plus intéressantes pour optimiser les requêtes OLAP. Le problème de sélection des vues matérialisée est l'un des problèmes les plus importants qui utilise les sous-expressions communes des requêtes.

Il vise à sélectionner les sous-expressions qui offrent un bénéfice élevé relativement aux fonctions objectifs utilisées pour une charge de requêtes, tout en respectant certaines contraintes telles que le budget de stockage [173], le coût de maintenance [20] et le coût de réécriture des requêtes [9] . Le PSV est considéré comme un problème NP-difficile [20]. Une large panoplie d'algorithmes a été proposée pour traiter ce problème , couvrant les technologies SQL et NoSQL ainsi que les différentes infrastructures de déploiement (centralisées, parallèles, cloud). Avant d'aborder ces travaux, nous donnons une brève définition des vues matérialisées. Ensuite, nous présentons la formalisation générique de PSV.

Définition 3.3.5 *Les vues matérialisées sont des requêtes dont les résultats sont stockés et maintenus afin de faciliter l'accès aux données dans leurs tables de base sous-jacentes [22, 176].*

Dans la littérature, plusieurs formalisation existent pour le problème de sélection des vues matérialisées (PSV). La formalisation générique d'un PSV est présentée comme suit [20] : étant donné un ensemble de requêtes $\{Q_1, \ldots, Q_n\}$ et un ensemble de contraintes C (par exemple, coût de stockage, coût de maintenance, coût de traitement), le PSV consiste à sélectionner un ensemble de vues matérialisées qui satisfont les exigences non fonctionnelles telles que la performance des requêtes, la consommation d'énergie, etc. et respecte l'ensemble C. Par la suite, plusieurs variantes ont été dérivées de la formalisation générique de PSV : (i) minimiser le coût de traitement des requêtes sous le contrainte d'espace de stockage [177] (ii) minimiser le coût de traitement de la requête et le coût de maintenance sous le contrainte d'espace de stockage [178] (iii) minimiser le coût de traitement des requêtes et la consommation d'énergie sous le contrainte de stockage [179].

Plusieurs approches ont été proposé pour traiter ces variantes. Une classification complète de ces approches a été proposée à [170], qui a identifié les dimensions de base pour la classification des différents algorithmes de sélection des vues matérialisées comme (i) les contraintes non fonctionnelles qu'ils prennent en compte lors du processus de sélection des vues et (ii) les heuristiques qu'ils utilisent pour obtenir les vues candidates. Cette étude divise les algorithmes de PSV en quatre catégories principales : algorithmes déterministes, algorithmes aléatoires, programmation par contraintes et algorithmes hybrides.

1. **algorithmes déterministes :** De nombreux travaux de recherche sur la sélection de vues utilisent des stratégies déterministes pour résoudre le problème de sélection de vues. La plupart de ces travaux utilisent la recherche exhaustive [180, 181] ou des heuristiques comme les algorithmes gloutonne [182, 183] .

2. **algorithmes aléatoires :** Les algorithmes aléatoires sont génétiques [184] ou utilisent un recuit simulé [185]. Les algorithmes génétiques génèrent des solutions à l'aide de techniques inspirées du processus d'évolution naturelle telles que la sélection, la mutation et le croisement. La stratégie de recherche de ces algorithmes est très similaire

à l'évolution biologique. Les algorithmes aléatoires peuvent être appliqués à des problèmes complexes traitant de grands espaces de recherche. Cependant, la qualité de la solution dépend de la configuration de l'algorithme ainsi que du réglage extrêmement difficile de l'algorithme qui doit être effectué au cours de nombreux tests.

3. **algorithmes hybrides :** Les algorithmes hybrides combinent les stratégies des algorithmes déterministes et randomisés dans leur recherche afin de fournir de meilleures performances en termes de qualité de solution. Les solutions obtenues par des algorithmes déterministes sont utilisées comme configuration initiale pour les algorithmes de recuit simulé ou comme population initiale pour les algorithmes génétiques comme dans [186].

4. **programmation par contraintes :** La programmation par contraintes est un descendant de la programmation déclarative. Cette technique de programmation a été exploitée dans de nombreuses applications pour résoudre des problèmes combinatoires [187]. Une approche basée sur la programmation par contraintes a été présentée à [188] pour résoudre le problème de sélection de vues. Les auteurs ont prouvé expérimentalement qu'une approche basée sur la programmation par contraintes fournit de meilleures performances par rapport à une méthode aléatoire en termes de coût.

Le processus de sélection des vues matérialisées passe par l'identification des vues candidates. Le nombre des vues candidates peut être très important [20]. Devant cette situation, Plusieurs efforts de recherche ont été menées à l'intérieur et à l'extérieur du SGBD pour représenter efficacement l'espace de recherche du PSV, en se basant sur l'utilisation des structures de données qui permettent de capturer les sous-expressions communes des requêtes. Dans la littérature, trois types de structures de données sont couramment utilisés :

1. **Graphe de vues ET-OU :** un graphe de vues ET-OU représente l'union de tous les plans possibles de chaque requête [189]. Le graphe de vues ET-OU a été défini à [190] par un graphe orienté acyclique composé de deux types de nœuds : les nœuds d'opération et les nœuds d'équivalence. Chaque nœud d'opération représente une expression algébrique (Sélection-Projection-Jointure). Un nœud d'équivalence représente un ensemble d'expressions logiques équivalentes. Le graphe de vues ET-OU a été utilisé par plusieurs approches dans la littérature [20, 191, 192].

2. **Plan d'exécution multi-vues (MVPP) :** Le plan MVPP a été proposé par yang à [193]. Il est représenté par un graphe acyclique, qui est composé de quatre niveaux : au niveau 0, on retrouve les nœuds feuilles représentant les tables de base de l'entrepôt de données. Au niveau 1, on trouve des nœuds qui représentent les résultats d'opérations algébriques unaires telles que la sélection et la projection. Au niveau 2, on trouve des

nœuds représentant des opérations binaires telles que la jointure, l'union, etc. Le dernier niveau représente les résultats de chaque requête. Chaque nœud intermédiaire du graphe est étiqueté avec le coût de chaque opération et son coût de maintenance.

3. **Cube de données en treillis :** Ce type de graphe a été proposé par Harinarayan à [183] qui a modélisé les données en plusieurs dimensions. Le cube de données en treillis est un graphe orienté acyclique dont les nœuds représentent les vues (les requêtes) qui agrègent les données par les attributs indiqués. Les arêtes dénotent la relation de dérivabilité entre les vues.

En raison de l'importance et de la complexité de PSV, de nombreux efforts de recherche du milieu académique et industriel ont été menés. Les résultats obtenus par ces efforts ont été implémentés dans des SGBD commerciaux et open source tels que Data Tuning Advisor pour SQL Server [194], Design Advisor pour DB2 [195], SQL Access Advisor pour Oracle et Parinda pour PostgreSQL [196]. En examinant les principales solutions de PSV, nous constatons qu'elles ne considèrent généralement que quelques dizaines de requêtes statiques. Cette situation est inadaptée aux exigences actuelles des applications analytiques, où des charges de requêtes dynamiques à grande échelle impliquant des sous-expressions redondantes sont prises en compte. La diversité des analyses de données et l'évolution des directives métier rendent la charge de requêtes plus dynamique [8]. Ce dynamisme a contribué à multiplier les travaux de recherche sur les bases de données auto-adaptatives et autonomes [9, 150]. Plusieurs approches réactives telles que DynaMat [151], WATCHMAN [152] et le conseiller MQT [197] pour la sélection dynamique de vues matérialisées ont été proposées. La principale caractéristique de ces approches est qu'elles réagissent à l'utilisation transitoire, Au lieu de simplement s'appuyer sur les charges précédentes de requêtes [198]. Ces solutions doivent résoudre de multiples problèmes [199] : quelles vues doivent être matérialiser pour servir les requêtes actuelles et futures et quand expulser les vues matérialisées (cache LRU) [197]. Récemment, des approches pro-actives de sélection des vues matérialisées ont été proposées, qui se sont avérées efficaces pour optimiser les requêtes actuelles et futures et pour gérer le pool des vues matérialisées [23, 200].

Sur la base de cette discussion, la classification traditionnelle proposée à [188] peut être enrichie par l'inclusion de nouvelles dimensions. En plus des dimensions traditionnelles telle que les contraintes utilisées et le type d'algorithmes de sélection, on peut inclure d'autres dimensions telles que : les structures de données utilisées pour représenter l'espace de recherche, la nature de la sélection (statique/dynamique), le volume des charges de requêtes considérées et la stratégie de sélection des vues (réactive/pro-active). La figure 3.8 résume notre classification des travaux qui traitent du PSV.

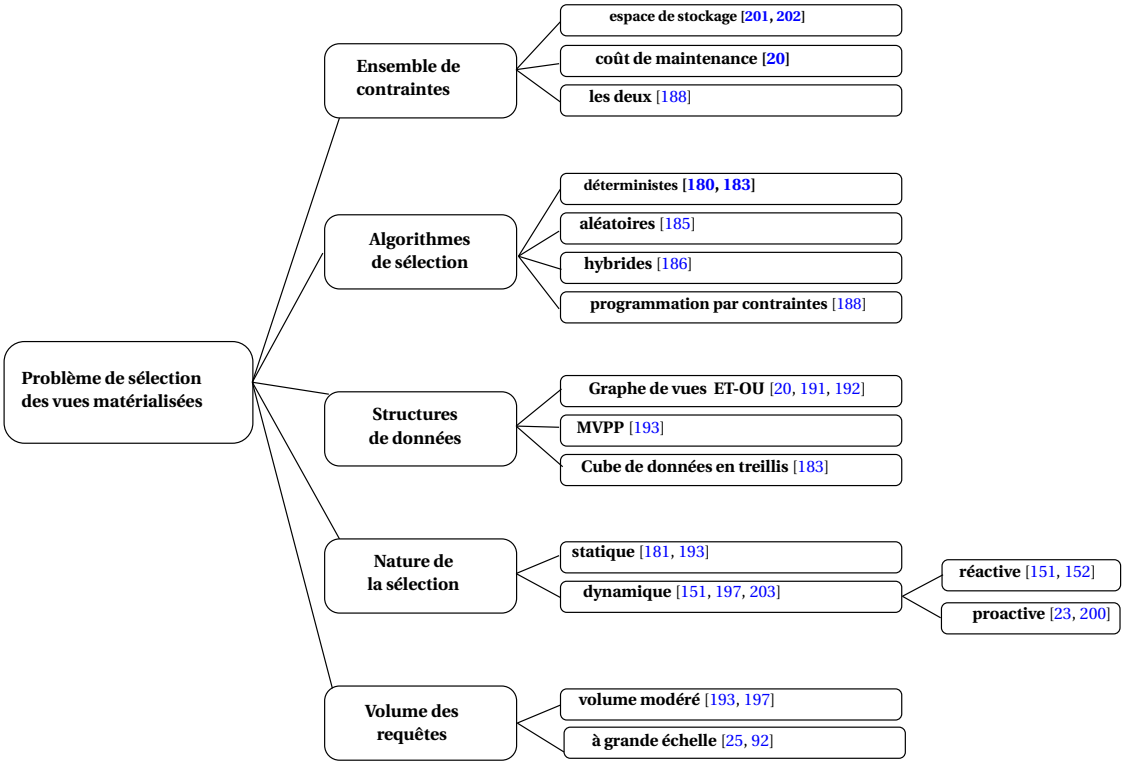


FIGURE 3.8 – Classification des travaux du PSV

3.3.4 Traitement simultané d'optimisation multi requêtes et de sélection des vues matérialisées

La sélection de vues matérialisées est un facteur clé pour concevoir et améliorer la qualité de service (QoS) des applications de bases de données avancées telles que les bases de données analytiques [20], les bases de données autonomes [9, 204], les bases de données sémantiques [205, 206], les bases de données cloud [207] et les bases de données NoSQL [208]. En analysant les études de PSV, on se rend compte qu'elles ne sont pas reliées au PMQO. Certains ne citent même pas l'article pionnier de PMQO [143].

Le travail proposé par Yang à [193] est le pionnier dans le contexte des entrepôts de données qui a montré la forte dépendance entre PMQO et PSV. Il vise à construire un plan de requête unifié pour un ensemble donné de requêtes . Au début, ils sélectionnent le plan de jointure individuel pour chaque requête. Par la suite, ces plans sont fusionnés dans un plan de requête unifié appelé MVPP représenté par un graphe orienté acyclique et dédié au processus de sélection des vues matérialisées. Deux algorithmes sont proposés pour sélectionner le meilleur MVPP qui a le coût minimum. Le premier algorithme, appelé solution réalisable, il génère tous les MVPP possibles et le plan avec le coût minimum sera choisi. Cet algorithme est coûteux en termes de calcul. Pour simplifier l'algorithme précédent, un deuxième algorithme est proposé basé sur la programmation linéaire en entier. Cet algorithme est exécuté en deux étapes : (1) la génération des vues candidates qui ont un bénéfice positif entre le traitement de la requête et la maintenance de la vue. Ce bénéfice correspond à la somme du

traitement des requêtes utilisant la vue moins le coût de maintenance de cette vue, (2) la sélection des vues, où seuls les nœuds candidats à bénéfice positif sont sélectionnés pour être matérialisés.

Le principal inconvénient du travail de Yang [193] concerne la scalabilité de ses algorithmes de construction de MVPP. Pour contrer ce problème, les hypergraphes ont été proposés dans [209] pour coupler PMQO et PSV en considérant un nombre élevé de requêtes. Les auteurs ont utilisé la structure d'hypergraphe pour l'identification de sous-expressions communes entre un très grand nombre de requêtes afin d'étudier leurs contributions à la sélection des vues matérialisées appropriées. En fait, la prise en compte d'un grand nombre de requêtes produit un nombre massif de vues à matérialiser, ce qui viole la contrainte d'espace de stockage. En effet, il est impossible de matérialiser toutes les vues candidates devant cette situation. Pour traiter ce problème, les auteurs [209] ont mis en place une stratégie qui vise à maximiser le bénéfice de l'utilisation des vues matérialisées avant leur suppression. Pour ce faire, ils ont proposé une nouvelle politique d'ordonnancement des requêtes qui permet de trouver le meilleur ordre des requêtes qui produit le plus grand bénéfice de l'ensemble de vues matérialisées.

Une analyse approfondie des principales études portant sur PMQO et PSV permet d'identifier plusieurs points communs. Le principal point commun entre PMQO et PSV est l'utilisation de la théorie des graphes et de leur structure de données soit pour élaguer leurs espaces de recherche à grande échelle, soit pour identifier les sous-expressions communes entre les requêtes. Un autre point partagé par PSV et PMQO est l'utilisation des politiques d'ordonnancement des requêtes pour améliorer leurs algorithmes.

Pour illustrer ce point, plusieurs bons exemples peuvent être donnés. Nous citons par exemple, les travail de Cosar [157], où l'ordonnancement des requêtes a joué un rôle crucial pour améliorer l'algorithme d'optimisation multi-requêtes. De plus, plusieurs travaux ont étudié PSV sous le contrainte d'ordonnancement de requêtes. Dans [197], une formalisation dynamique de PSV est donnée en considérant une politique d'ordonnancement des requêtes basée sur un algorithme génétique. Les deux principales limitations de l'algorithme de Phan [197] sont (i) leur dépendance vis-à-vis de DB2 advisor et (ii) leur algorithme génétique gourmand pour réordonner les requêtes qui ne conviennent pas lors du passage à l'échelle. Pour surmonter ces limitations, une approche évolutive appelée SLEMAS est proposée dans [209]. La scalabilité de cette approche est assurée au moyen d'une structure d'hypergraphe qui est utilisée pour identifier le partage d'opérations entre un nombre important de requêtes. Par la suite, les opérations les plus partagées sont considérées comme candidates à la matérialisation. Les vues matérialisées dans SLEMAS sont sélectionnées dynamiquement en tenant compte des contraintes d'ordonnancement pour une charge de requêtes connue a priori. Le principal inconvénient de cette approche est leur hypothèse d'un ensemble statique de requêtes, ce qui contredit la nature dynamique des requêtes analytiques.

3.4 Discussion

La riche revue des deux problématiques inter-connectées et bien connues dans le monde des bases de données qui sont : l'optimisation multi-requêtes (PMQO) et la sélection de vues matérialisées (PSV), nous a permis de tirer plusieurs conclusions : (i) l'importance des solutions proposées et ses incapacités dans la gestion simultanée des 3-propriétés récentes des requêtes analytique (nombre élevé de requêtes, dynamiques et interactives) (ii) la contribution exceptionnelle de la théorie des graphes et ses structures de données dans la résolution des PMQO et PSV (iii) le grand impact de l'ordonnancement des requêtes sur la résolution des deux problèmes. Le tableau 1.1 résume notre discussion en montrant les avantages et les inconvénients des principales études traitant de PMQO et PSV de manière isolée ou conjointement.

TABLE 3.1 – Classification des études traitant PMQO et PSV séparément ou conjointement

Problèmes	**Travaux**	**Propriétés considérées**	**Structure de données**	**ordonnancement des requêtes**	**inconvénient**
PMQO	[143]	interaction des requêtes	Graphes de requêtes	Non	concerne la scalabilité
	[157]	interaction des requêtes	Graphes de requêtes	Oui	concerne la scalabilité
PSV	[151]	Dynamique	Non	Non	concerne la scalabilité
	[197]	Aucune propriété	Non	Oui	scalabilité et dépendance à DB2
PMQO & PSV	[193]	interaction des requêtes	MVPP	Non	scalabilité
	[209]	Nombre élevé de requêtes & interaction des requêtess	Hypergraphe	Oui	une charge statique de requêtes

3.5 Conclusion

De nos jours, les applications modernes nécessitent des solutions puissantes pour effectuer plusieurs analyses en temps réel via des requêtes analytiques complexes. Pour répondre à ces exigences, il est d'abord nécessaire de comprendre les techniques de traitement de ces

requêtes et leurs caractéristiques récentes. Pour cela, nous avons commencé ce chapitre par une étude du processus de traitement des requêtes et ses différentes étapes : l'analyse et la transformation de la requête, la génération du plan physique et l'exécution du plan. Ces requêtes doivent être rapidement optimisées pour accélérer le processus décisionnel. De nombreux efforts académiques et industriels ont été faits pour optimiser ces requêtes, où l'exploitation des sous-expressions communes a joué un rôle important dans la production de résultats prometteurs. La maîtrise du problème de la sélection des sous-expressions communes a contribué dans la résolution de plusieurs problèmes, en particulier dans la sélection des vues matérialisées et l'optimisation multi-requêtes, sur lesquels nous nous concentrons dans ce travail. Nous présentons dans la deuxième partie de ce chapitre, les principales études traitant de PMQO et PSV de manière isolée et conjointe.

Aujourd'hui, les requêtes analytiques sont connues par trois propriétés principales : (1) très nombreuses, (2) dynamiques, et (3) partagent des opérations similaires. Une analyse approfondie des travaux de l'état de l'art nous a permis de déduire que les solutions traditionnelles ne peuvent pas être reproduites directement pour optimiser ces requêtes soit au niveau logique, soit au niveau physique (e.g. en sélectionnant des vues matérialisées). Nous supposons que l'optimisation de ces requêtes en utilisant les techniques d'optimisation physiques nécessite des structures de données évolutives et flexibles comme ***les hypergraphes*** qui ont déjà montré leurs contributions dans la gestion du PMQO et PSV en considérant un nombre énorme de requêtes statiques. Compte tenu de la nature dynamique des requêtes analytiques, nous supposons que l'approche de sélection des vues matérialisées doit être ***proactive*** afin de pouvoir optimiser les requêtes actuelles et futures.

Chapitre 4

Hypergraphes au Service de L'optimisation de Requêtes : Gestion des 3-Propriétés

"**Exécuter 100 fonctions sur une structure de données est mieux que d'exécuter 10 fonctions sur 10 structures de données.**"
— —— Alan Perlis

Sommaire

4.1 Introduction

Aujourd'hui, les charges de requêtes analytiques incluent des requêtes très nombreuses, dynamiques et interactives. De nombreuses entreprises utilisent toujours des SGBD traditionnels qui doivent gérer et optimiser ces charges de requêtes. Les 3-propriétés de requêtes impactent de nombreux problèmes importants tels que l'optimisation multi-requêtes (PMQO) et la conception physique, en particulier le problème de sélection de vues matérialisées (PSV). Les solutions traditionnelles d'état de l'art (voir le Chapitre 3) proposent des approches limitées pour la gestion des 3-propriétés de requêtes, où ils ont été partiellement considérés. La prise en compte simultanée des trois propriétés de requêtes est très important pour répondre aux exigences analytiques actuelles. Nous supposons que le traitement simultané de ces trois propriétés nécessite l'utilisation d'une structure de données flexible et évolutive comme les hypergraphes qui ont déjà montré leurs contributions à l'optimisation des requêtes statiques à grande échelle.

Dans ce chapitre, nous présentons nos contributions qui portent sur l'utilisation des hypergraphes pour coupler le PMQO et le PSV sous un nouvel angle, en considérant simultanément les trois propriétés des requêtes analytiques. Nous commençons ce chapitre par la présentation des concepts de bases et les définitions liées aux hypergraphes, en montrant les motivations derrière l'utilisation des hypergraphes pour traiter notre problème. Ensuite, nous décrivons les notions clés qui ouvrent la voie à notre approche, en montrant le processus de représentation des requêtes par un hypergraphe et en mettant en évidence l'efficacité des hypergraphes dans la gestion de nos 3-propriétés. Après avoir réuni tous les ingrédients, nous présentons notre approche proactive de sélection de vues matérialisées qui repose sur l'exploitation des hypergraphes pour traiter conjointement le PMQO et le PSV, en considérant concurremment nos trois propriétés de requêtes.

4.2 Motivation

La théorie des hypergraphes a une longue histoire de résolution des problèmes à grande échelle grâce à sa capacité à modéliser toutes les relations et son efficacité à diviser les espaces de recherche pour les problèmes difficiles en plusieurs sous-espaces de recherche grâce à ses outils de partitionnement rapides. Les hypergraphes sont plus expressifs que les graphes traditionnels, tel qu'ils peuvent capturer n'importe quelle relation entre un groupe d'objets, alors que les graphes traditionnels ne peuvent capturer que des relations binaires [210]. Cette spécificité donne plus de flexibilité dans la formulation précise de plusieurs problèmes importants en calcul scientifique combinatoire [211]. Depuis 20 ans, la théorie des hypergraphes contribue à modéliser et à résoudre plusieurs problèmes dans divers domaines tels que les problèmes de localisation [212], les problèmes combinatoires [213], en chimie [214],

en télécommunications [212], en traitement d'images [215, 216], en réseaux sociaux [217], etc.

Dans le monde des bases de données, les hypergraphes ont été utilisés lors des phases logiques et physiques (par exemple, dans la détection des dépendances fonctionnelles [218], le partitionnement des données pour optimiser les charges de requêtes OLTP [219], la sélection de vues matérialisées pour optimiser les charges de requêtes OLAP à grande échelle [22], etc). Dans ce qui suit, nous présentons quelques applications pratiques de la théorie des hypergraphes dans le domaine des bases de données.

- **Modélisation des schémas de bases de données :** une base de données peut être vue comme un ensemble d'attributs et un ensemble de relations entre ces attributs. Cette propriété a motivé certains travaux à introduire la théorie des hypergraphes pour modéliser les schémas de bases de données relationnelles [220]. Les sommets de l'hypergraphe sont l'ensemble des attributs, et l'ensemble des hyperarêtes est l'ensemble des relations entre ces attributs.

- **Partitionnement et réplication des données :** En [219] , les auteurs ont présenté une approche de partitionnement des bases de données OLTP en se basant sur l'utilisation des hypergraphes. Cette approche représente une base de données et sa charge de requêtes OLTP à l'aide d'un hypergraphe, où les tuples sont représentés par des sommets et les transactions sont représentées par des arêtes reliant les tuples utilisés dans la transaction. Ensuite, ils proposent d'utiliser les algorithmes de partitionnement de l'hypergraphe pour trouver des partitions équilibrées, qui minimisent approximativement le nombre de transactions multi-sites.

- **Optimisation des requêtes SPARQL :** Dans les base de données sémantique, une étude a été menée à [221] sur l'utilisation des hypergraphes pour l'optimisation des requêtes SPARQL pour les données RDF. Ce travail se concentrent sur le stockage des données RDF sous forme d'un hypergraphe, puis sur le traitement des requêtes SPARQL dans l'hypergraphe résultant. Après avoir analysé les performances concernant le temps de traitement des requêtes avec d'autres systèmes, l'approche proposée a donné de très bons résultats en moins de temps.

- **Optimisation des requêtes OLAP :** En 2015, un effort de recherche [22] a été porté en proposant des hypergraphes pour traiter l'identification de sous-expressions communes pour un large ensemble de requêtes et pour étudier leurs contributions à la sélection des vues matérialisées. Les hypergraphes ont prouvé leur efficacité pour traiter conjointement le PSV et le PMQO, en considérant un nombre important de requêtes analytique connues à l'avance.

Dans de nombreux domaines du monde réel, les hypergraphes se sont avérés très efficaces pour gérer rapidement d'énormes quantités de données et de requêtes. Par exemple,

en réseaux sociaux, le nombre d'utilisateurs est estimé d'environ 4,62 milliards en 2022 [1], soit environ 58 % de la population totale de la terre, où les hypergraphes ont été largement utilisés pour capturer l'interaction et les comportements des utilisateurs. Ces dernières années, des résultats encourageants ont été obtenus démontrant l'utilité et l'efficacité des hypergraphes dans le traitement conjoint des deux problèmes de sélection des vues matérialisées et d'optimisation multi-requêtes pour un large ensemble de requêtes analytiques, sans la prise en compte de l'aspect dynamique de ces requêtes. L'historique de succès des hypergraphes dans la résolution de ces problème, nous motive à les utiliser dans notre contexte de couplage du PMQO et PSV, en considérant concurremment nos 3 propriétés de requêtes analytiques.

4.3 Hypergraphes comme solution pour gérer les 3 propriétés de requêtes

Dans cette section, nous présentons les notions et les définitions fondamentales liées aux hypergraphes et leur capacité à gérer les trois propriétés des requêtes analytiques.

Définition 4.3.6 *Un hypergraphe $H = (V, E)$, est défini comme un ensemble de sommets (nœuds) V et un ensemble d'hyper-arêtes E, où chaque hyper-arête connecte un sous-ensemble non vide de nœuds [222]. Notez que lorsque $|e_i| = 2$ ($\forall\ e_i \in E$), l'hypergraphe est un graphe standard.*

Définition 4.3.7 *Le degré d'un sommet $v_i \in V$, noté $d(v_i)$ représente le nombre d'hyper-arêtes distinctes dans E qui relient v_i.*

Définition 4.3.8 *La taille (la longueur) d'un hyper-arête $e_i \in E$, notée $|e_i|$ est définie comme sa cardinalité (le nombre des sommets connectés par l'hyper-arête e_i).*

Définition 4.3.9 *Un nœud pivot d'un hypergraphe est la première jointure partagée par toutes les requêtes.*

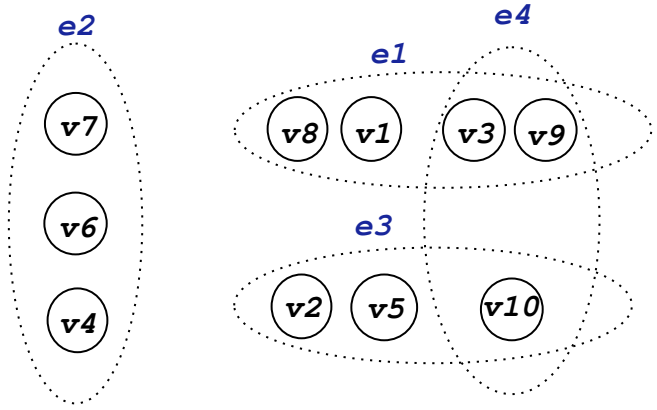
La matrice d'incidence d'un hypergraphe permet de compter la connexion entre les hyper-arêtes et les sommets, où les lignes et les colonnes représentent respectivement les sommets et les hyperarêtes. La valeur (i, j) dans la matrice, notée par IM_{ij}, est égale à 1 si le sommet v_i est connecté dans l'hyperarête e_j, et 0 sinon.

$$IM_{ij} = \begin{cases} 1, \text{ Si le } j^{\text{ème}} \text{ hyper-arête contient le } i^{\text{ème}} \text{ sommet} \\ 0, \text{ Sinon.} \end{cases}$$

4.3.1 Représentation d'une requête par un hypergraphe

Dans cette section, nous présentons notre processus qui permet de passer d'une requête OLAP à un hypergraphe. Pour donner une hypothèse réaliste, nous considérons comme cas

1. https://datareportal.com/social-media-users



(A) L'Hypergraphe H_1

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

(B) La Matrice d'incidence de H_1

FIGURE 4.1 – Exemple d'un hypergraphe

d'étude la classe de requêtes SPJ (Sélection-Projection-Jointure), qui représentent les requêtes les plus couramment étudiées en théorie des bases de données. Nous nous concentrons particulièrement sur les opérations de sélection et de jointure, qui sont connues par les opérations les plus coûteuses. Une requête SQL est généralement représentée sous forme d'un arbre de requête.

Définition 4.3.10 *Un arbre de requête est une structure de données arborescente représentant une expression d'algèbre relationnelle. Les tables de la requête sont représentées sous forme de nœuds feuilles. Les opérations d'algèbre relationnelle sont représentées par les nœuds internes. La racine de l'arbre représente la requête dans son ensemble.*

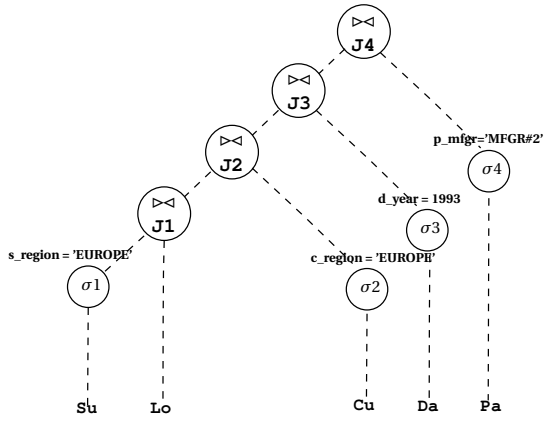
Dans notre étude, le plan d'une requête donnée est obtenu par un arbre de profondeur gauche [223], où toutes les sélections sont poussées aussi loin que possible dans son arbre (graphe) de requête. Dans ce qui suit, nous montrons comment un arbre de requête est transformé en hypergraphe.

Exemple 4.3.4 *Pour illustrer comment représenter une requête OLAP par un hypergraphe, supposons la requête suivante Q_i définie sur le benchmark de schéma en étoile (SSB)*[2] *qui contient une table de faits Lineorder et quatre tables de dimension Customer, Supplier, Part et Dates.*

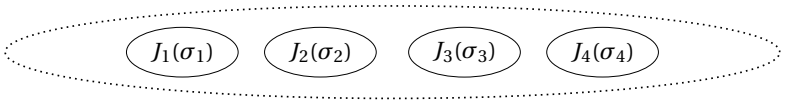
La Requête Q_i

```
SELECT   d_year, s_nation, p_category,
         sum(lo_revenue - lo_supplycost) as profit
FROM   DATES, CUSTOMER, SUPPLIER, PART, Lineorder
WHERE  lo_custkey = c_custkey    (J2)
AND    lo_suppkey = s_suppkey    (J1)
AND    lo_PARTkey = p_PARTkey    (J4)
```

2. https://www.cs.umb.edu/~poneil/StarSchemaB.pdf



(A) L'arbre de la requête Q_i



(B) L'Hypergraphe de la requête Q_i

	e
J_1	1
J_2	1
J_3	1
J_4	1

(C) Sa Matrice d'incidence

FIGURE 4.2 – La représentation de la requête Q_i par un hypergraphe

```
AND     lo_orderdate = d_datekey  (J3)
AND     c_region = 'EUROPE'
AND     s_region = 'EUROPE'
AND     d_year = 1993
AND     p_mfgr = 'MFGR#2'
group by    d_year, s_nation, p_category
order by    d_year, s_nation, p_category;
```

L'arbre de la requête Q_i est donné sur la figure 4.2a, où quatre jointures et sélections sont bien représentées. Cet arbre peut être facilement transformé en un hypergraphe à quatre sommets représentant les quatre jointures $J1$, $J2$, $J3$, $J4$ et une hyper-arête e_j correspondant à notre requête Q_i. Notez que chaque nœud de jointure est défini par son prédicat de jointure et ses sélections associées.

À partir de l'hypergraphe de la requête , nous pouvons également générer l'arbre algébrique de la requête si le type d'arbre de traitement de jointures (e.g. arbre de profondeur gauche , de profondeur droite ou ramifié) et l'ordre de jointure sont connus à l'avance. L'arbre de requête de la figure 4.2a est obtenu en considérant la stratégie de profondeur gauche et l'ordre de jointure suivant ($SU \bowtie LO \bowtie CU \bowtie DA \bowtie PA$).

Il est important de noter que l'ordre de jointure joue un rôle crucial dans l'optimisation des requêtes de jointure en étoile impliquant des tables de dimension et une table de faits. Dans notre étude, la table de faits d'une requête analytique est toujours jointe à des tables

de dimension suivant leur taille (de la plus petite à la grande, en considérant leur facteurs de sélectivité). Des techniques récentes basées sur l'apprentissage automatique et profond pour résoudre le problème de l'ordre de jointure peuvent être facilement incorporées dans notre approche [224, 225].

4.3.2 Hypergraphe pour capturer l'interaction des requêtes

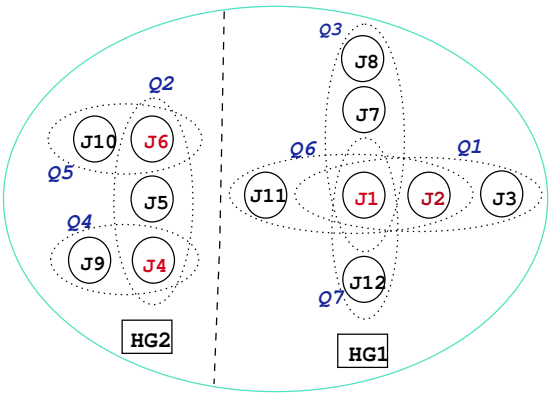
La théorie des hypergraphes [226] permettait à une hyperarête de connecter un nombre arbitraire de sommets au lieu de deux sommets dans les graphes réguliers, ce qui permet de mieux comprendre de nombreux systèmes réels. Les hypergraphes ont montré une efficacité supérieure pour capturer l'interaction entre les objets étudiés dans de nombreux domaines tels que l'exploration de données, la récupération de texte/image, la bio-informatique, l'exploration sociale et l'apprentissage automatique.

Dans cette section, nous montrons comment les hypergraphes peuvent facilement capturer l'interaction de requêtes analytiques. Pour une charge de requêtes donnée W nous définissons un hypergraphe global GH avec un ensemble de sommets GH_V et un ensemble d'hyperarêtes GH_E. Chaque sommet $v_i \in GH_V$ correspond à un nœud de jointure J_i, tandis que chaque hyperarête $e_j \in GH_E$ correspond à une requête $Q_j \in W$. L'ensemble des sommets d'une hyperarête e_j représente l'ensemble des jointures qui participent au traitement de la requête Q_j.

Exemple 4.3.5 *Pour illustrer le processus de construction de l'hypergraphe global pour une charge de requêtes donnée, nous considérons les sept (7) requêtes OLAP ($\{Q_1, Q_2, Q_3, \ldots, Q_7\}$) définies en annexe B et générées aléatoirement à l'aide du générateur de requêtes de benchmark SSB. La figure 4.3 montre l'hypergraphe global obtenu et sa matrice d'incidence. La matrice d'incidence peut facilement donner des indices sur les jointures les plus partagées en ajoutant une colonne représentant la fréquence d'utilisation de chaque opération de jointure. La fréquence d'utilisation d'une opération de jointure $Join_i$ est calculée comme suit :*

$$Freq(Join_i) = \sum_{j=1}^{n} IM_{ij} \qquad \{n \text{ est le nombre de requêtes}\} \tag{4.1}$$

Étant donné que les opérations sélections sont effectuées avant les jointures, notre hypergraphe global peut contenir plusieurs composants disjoints. En effet, les opérations de sélection réduisent le partage des requêtes. Dans notre exemple, l'hypergraphe contient les deux composantes suivantes $HG1$: ($Q1$, $Q3$, $Q6$, $Q7$) et $HG2$: ($Q2$, $Q4$, $Q5$). Nous observons que toutes les requêtes du premier composant $HG1$ partagent la même opération de jointure identifiée par le nœud $J1$. Comme le montre l'hypergraphe et sa matrice d'incidence, les quatre nœuds identifiés par $J1$, $J2$, $J4$ et $J6$ sont les opérations de jointure les plus partagées et elles seront considérées comme candidates à la matérialisation. Les expressions SQL des résultats intermédiaires les plus partagés par les requêtes sont décrites dans le tableau 4.1.



(A) L'Hypergraphe Global

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	FRQ
j_1	1	0	1	0	0	1	1	4
j_2	1	0	0	0	0	1	0	2
j_3	1	0	0	0	0	0	0	1
j_4	0	1	0	1	0	0	0	2
j_5	0	1	0	0	0	0	0	1
j_6	0	1	0	0	1	0	0	2
j_7	0	0	1	0	0	0	0	1
j_8	0	0	1	0	0	0	0	1
j_9	0	0	0	1	0	0	0	1
j_{10}	0	0	0	0	1	0	0	1
j_{11}	0	0	0	0	0	1	0	1
j_{12}	0	0	0	0	0	0	1	1

(B) La Matrice d'incidence

FIGURE 4.3 – L'hypergraphe global des 7 requêtes et sa matrice d'incidence

Le nœud de jointure dans l'hypergraphe	L'expression SQL de la jointure
Le nœud J1	Select * from Lineorder, Dates where lo_orderdate = d_datekey and d_year >= 1992 and d_year <= 1997
Le nœud J2	Select * from Lineorder, Customer where lo_custkey = c_custkey and c_region = 'AFRICA'
Le nœud J4	Select * from Lineorder, Dates where lo_orderdate = d_datekey and d_year >= 1993 and d_year <= 1998
Le nœud J6	Select * from Lineorder,Supplier where lo_suppkey = s_suppkey and s_nation = 'VIETNAM'

TABLE 4.1 – Les expressions SQL des résultats intermédiaires les plus partagés.

4.3.3 Hypergraphe pour gérer un nombre élevé de requêtes

Les hypergraphes ont fait preuve d'efficacité dans la gestion de plusieurs applications traitant une énorme quantité de données et de transactions. Plusieurs exemples réels peuvent être donnés : Comme dans la conception VLSI, dans laquelle des centaines de millions de portes sont nécessaires pour concevoir des circuits logiques à travers l'hypergraphe. De plus, dans les applications de réseaux sociaux, les hypergraphes ont été exploités pour capturer l'interaction et les comportements des milliards d'utilisateurs [217]. L'utilisation des hypergraphes est motivée par leur capacité à gérer des problèmes complexes grâce à leurs outils de partitionnement avancés.

Le partitionnement hypergraphique est couramment utilisé pour diviser l'espace de recherche de problèmes combinatoires à grande échelle en plusieurs sous-espaces de recherche, ce qui réduit considérablement la complexité des problèmes étudiés. Cette fonctionnalité est assurée grâce aux outils sophistiqués de partitionnement qui garantissent l'évolutivité. Nous présentons dans cette section, quelques définitions fondamentales et notions de bases pour comprendre le processus de partitionnement d'un hypergraphes.

Définition 4.3.11 *Le problème de partitionnement d'un hypergraphe H (V, E) consiste à diviser l'ensemble des sommets de l'hypergraphe H en un nombre fixe de k partitions disjointes de taille bornée* $\prod = \{V_1, V_2, ..., V_k\}$*, en minimisant une fonction objectif donnée [227]. Un algorithme de partitionnement d'un hypergraphe doit vérifier les propriétés suivantes :*

1. *chaque partition* V_i *est un sous-ensemble non vide de sommets, i.e.,* $V_i \neq \emptyset$ *, pour* $1 \leq i \leq k$ *;*
2. *toutes les partitions sont disjointes deux à deux, c'est-à-dire* $V_i \cap V_j = \emptyset$ *, pour* $1 \leq i < j \leq k$ *;*
3. *l'union de toutes les partition est égale à V. c'est-à-dire* $\cup_{v_i} = V$ *, pour* $1 \leq i \leq k$ *;*

Où, le nombre de partitions $|\prod| = k$ *et* $k > 2$*, le partitionnement* $\prod$ *est appelée un partitionnement k-way. Sinon, lorsque* $k = 2$*, on appelle* $\prod$ *un bi-partitionnement hypergraphique (ou bien bi-sectionnement).*

Définition 4.3.12 *Une Hyperarête coupée* $e \in E$ *est coupée si elle relie plus d'une partie (i.e.,* $|V_e| > 1$*), et non coupée sinon (i.e.,* $|V_e| = 1$*).*

De nombreuses approches existent pour résoudre le problème de partitionnement d'un hypergraphe. Ces algorithmes peuvent être classées en quatre catégories :

- Énumération exhaustive qui produit tous les partitionnements possibles. A la fin, il sélectionne le partitionnement au coût optimal défini par les concepteurs. Cette méthode donne la solution optimale, mais elle a une complexité exponentielle, en raison de la taille d'espace de recherche [228].

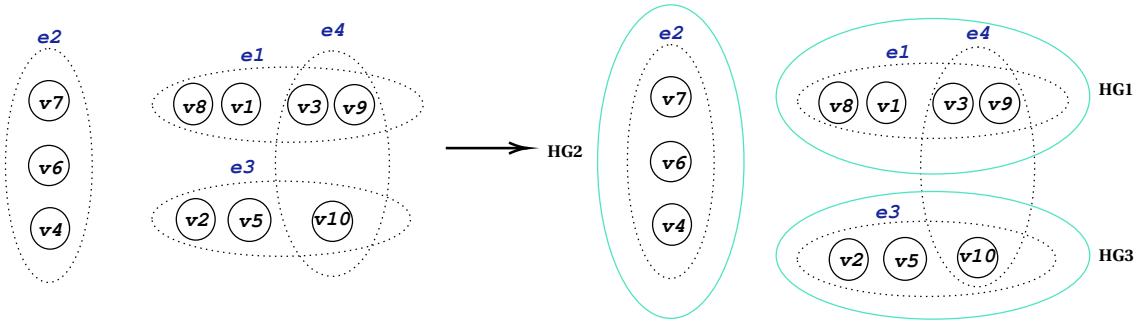


FIGURE 4.4 – Un exemple de partitionnement hypergraphique en utilisant hMetis

- Algorithme par séparation et évaluation (Branch and Bound) qui est proposé pour améliorer l'approche exhaustive en implémentant un arbre de profondeur gauche de partitionnement partiel. Cet arbre permet d'obtenir une solution sous-optimale en vérifiant les contraintes d'équilibre dans chaque nœud. Malgré ses performances, il peut avoir une complexité exponentielle [228].
- L'approche de Fiduccia-Mattheyses qui est proposée pour mettre à l'échelle les algorithmes de partitionnement hypergraphique [229]. Le partitionnement de Fiduccia-Mattheyses est une heuristique linéaire permet d' améliorer le partitionnement en utilisant des passes itératives dans lesquelles chaque sommet est déplacé exactement une fois. Les passes sont généralement appliquées jusqu'à ce qu'il ne reste que peu ou pas d'amélioration. Les solutions initiales sont souvent produites à l'aide d'un simple algorithme randomisé.
- Le framework Fiduccia-Mattheyses à plusieurs niveaux qui se compose de trois composants principaux : le clustering, le partitionnement de niveau supérieur et le composant de raffinement.

Il existe aussi d'autres méthodes pour résoudre le problème de partitionnement d'un hypergraphe telles que le recuit simulé et la recherche Tabou [230], les techniques spectrales [231], les algorithmes pilotés par le flux de réseau [232] et le solveur de flux incident [233]. hMeTis [234], et PaToH [235] sont deux exemples des techniques les plus importantes pour le partitionnement hypergraphique , qui sont initialement développées dans le domaine VLSI. La figure 4.4 montre un exemple de partitionnement de l'hypergraphe donné à la figure 4.1a en trois partitions en utilisant l'algorithme hMeTis.

4.3.4 Hypergraphe pour gérer l'aspect dynamique de requêtes

L'exemple4.3.5 a montré la construction d'un hypergraphe pour une charge de requêtes donnée et son utilisation dans la sélection de vues matérialisées candidates de manière statique. Cette situation est inadéquate au regard des exigences des applications analytiques d'aujourd'hui, où des charges de requêtes dynamiques à grande échelle sont envisagées. L'aspect dynamique de ces requêtes nécessite une construction incrémentale de l'hypergraphe global,

une sélection dynamique des vues matérialisées et une bonne gestion du pool de vues qui doivent optimiser les requêtes actuelles et futures. Pour ce faire, nous proposons dans ce qui suit une approche proactive de sélection de vues matérialisées, qui intègre concurremment nos trois propriétés de requêtes (voir la figure 3.5).

4.4 Re-sélection proactive des vues matérialisées

À ce stade, nous avons tous les ingrédients pour présenter notre approche proactive pour capturer les sous-expressions communes des requêtes pour des fins de matérialisation. Avant de détailler notre proposition et ses différentes étapes, nous formalisons notre problématique.

4.4.1 Formalisation du problème

Étant donné : (i) un entrepôt de données avec une table de faits F et n tables de dimensions $\{D_1,D_2,...,D_n\}$, (ii) un ensemble de requêtes connues à l'avance (requêtes en lots), (iii) un ensemble de requêtes ad-hoc qui arrivent dynamiquement, et (iv) une contrainte de stockage. Notre problème consiste à sélectionner un ensemble de vues matérialisées accélérant la performance de toutes les requêtes et satisfaisant la contrainte d'espace de stockage.

Afin de gérer les deux types de requêtes (les requêtes connues à l'avance et les requête ad-hoc) nous proposons une approche composée de deux phases : (i) une phase hors ligne qui gère un ensemble de premières requêtes Q_{first} basées sur un seuil donné δ, en sélectionnant les vues matérialisées les plus intéressantes grâce à l'utilisation des hypergraphes et (ii) une phase en ligne qui traite l'arrivée des requêtes ad-hoc, en les optimisant par la réutilisation des vues déjà sélectionnées , en enrichissant la structure de l'hypergraphe, et en sélectionnant de nouvelles vues en éliminant les moins intéressantes. L'architecture globale de notre proposition est décrite dans la figure 4.5.

4.4.2 La phase hors ligne

En fonction d'un seuil donné δ, les δ premières requêtes sont acheminées vers la phase hors ligne qui utilise la structure d'hypergraphe pour sélectionner les sous-expressions communes les plus bénéfiques, qui seront considérées comme candidates à la matérialisation. La phase hors ligne repose sur les principaux modules suivants :

1. **Analyseur de requêtes** : ce module prend en entrée un texte écrit en langage SQL et le convertit en un arbre de requête (voir l'exemple 4.3.4). Cette tâche permet d'analyser l'ensemble des requêtes Q_{first} afin d'identifier leurs opérations logiques (Sélection, Projection, et Jointure).

2. **Construction de l'hypergraphe** : après avoir analysé chaque requête dans l'ensemble de requêtes Q_{first}, nous utilisons les mêmes règles décrites dans l'exemple 4.3.5 pour

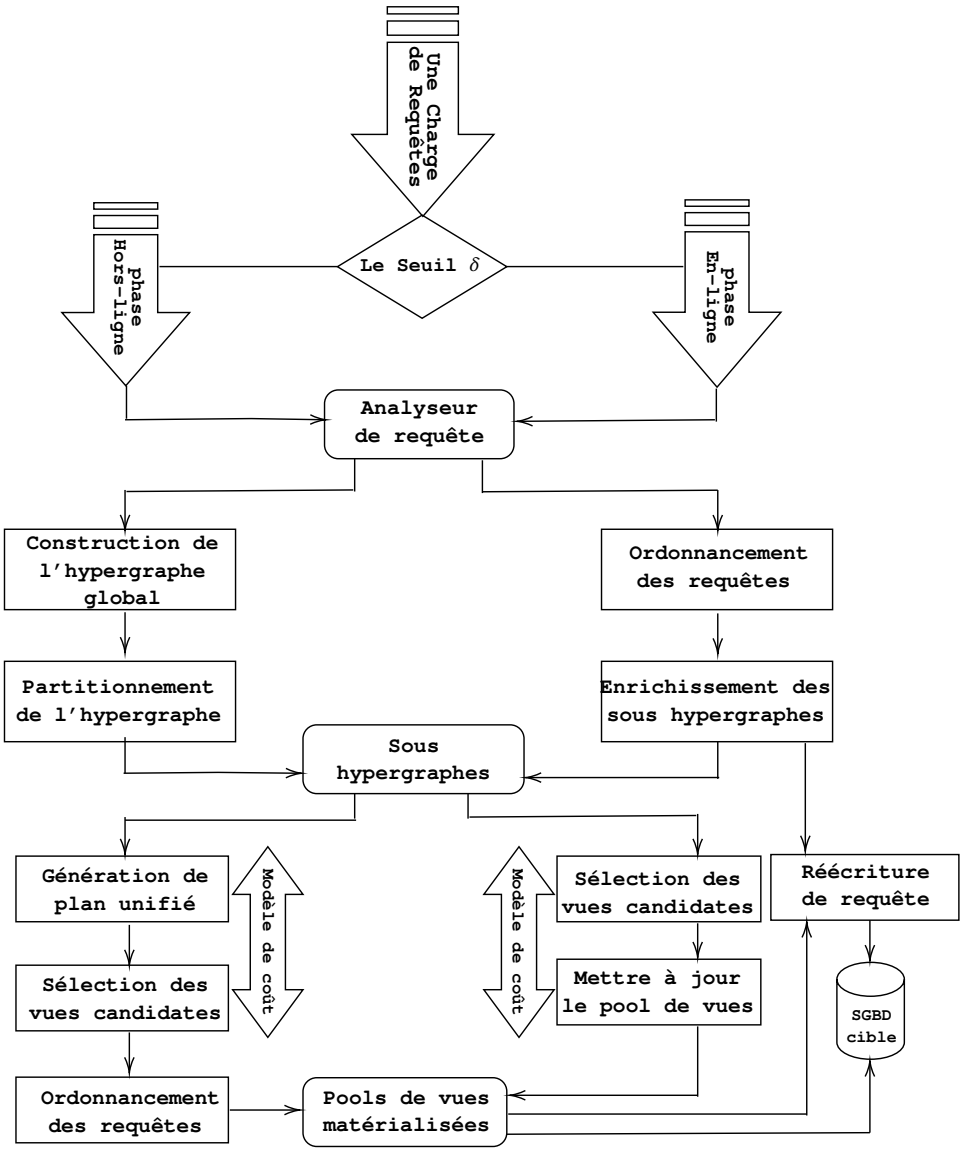


FIGURE 4.5 – L'architecture globale de notre approche

construire notre hypergraphe global initial en utilisant deux primitives principales : $ajouter-n\text{œ}ud()$ et $ajouter-hyperar\hat{e}tes()$. Comme nous l'avons mentionné précédemment, les sommets de l'hypergraphe définissent les jointures (et leur sélections) et les hyper-arêtes représentent l'ensemble des requêtes de Q_{first}. Une hyper-arête e_j connectant un ensemble de sommets correspond aux nœuds de jointure qui participent à l'exécution de la requête $Qj \in Q_{first}$. Le tableau 4.2 résume la correspondance entre la vision *graphe* et la vision *requête*.

3. **Partitionnement de l'hypergraphe :** L'objectif principal de notre approche est de sélectionner les opérations de jointures les plus partagées entre les requêtes pour qu'elles soient candidates à la matérialisation. Pour atteindre cet objectif , il est nécessaire de faciliter la gestion de notre hypergraphe global, d'autant plus que nous traitons un très grand nombre de requêtes . Pour ce faire, nous proposons une méthode basée sur le

Vision d'hypergraphe	Vision de requête
Ensemble de sommets	Ensemble de nœuds de jointure
Hyper-arête	Requête
Sous-hypergraphe	Composant connecté
Graphe orienté	Plan d'exécution

TABLE 4.2 – Analogie du graphe avec Requête.

principe : "diviser pour régner". Cette méthode vise à partitionner l'hypergraphe initial en composants (sous-hypergraphes) de requêtes en fonction de leur interaction, où l'interaction entre les requête est maximale à l'intérieur de chaque composant et minimale entre les composants. Pour assurer la scalabilité de notre approche, nous adaptons l'algorithme hMeTiS [226].

Contrairement aux codes originaux de hMeTiS, où le nombre de partitions est connu à l'avance, dans notre cas, le nombre de composants à construire est inconnu. Pour partitionner notre hypergraphe, nous adaptons un algorithme existant dérivé de la théorie des graphes pour agréger les nœuds de jointure en petits composants connectés. Le processus de partitionnement est appliqué sur l'hypergraphe initial H (V, E) et le résultat de la partitionnement de l'hypergraphe est k sous-hypergraphes, où chacun est un hypergraphe : $H_i(V_i, E_i)$, où $|E_i| \leq |E|$, pour $1 \leq i \leq k$; . Notre algorithme de partitionnement suit la même heuristique détaillée dans [236] selon la démarche suivante :

Tout d'abord, nous adaptons le code du partitionnement multi-niveaux d'hypergraphes (hMeTiS) pour découper notre hypergraphe en k partitions de sorte que le nombre d'hyper-arêtes coupées soit minimal. Dans notre contexte, le nombre exact de partitions à construire est inconnu. En même temps, nous voulons obtenir toutes les partitions disjointes possibles (appelées composants connectés). Pour ce faire, nous adaptons l'algorithme original à notre problème en bipartitionnant jusqu'à ce qu'aucune partition ne puisse être repartitionnée sans couper les hyper-arêtes. Plus précisément, l'algorithme se comporte comme suit : (i) l'ensemble des sommets sera divisé si et seulement si le nombre d'hyper-arêtes à couper est nul. (ii) Chaque résultat de bissection du partitionnement de l'hypergraphe sera divisé de la même manière jusqu'à ce qu'il n'y ait plus d'hypergraphe divisible.

4. **Génération du plan de requêtes unifié** : Cette étape vise à générer le plan de requête unifié (UQP) pour chaque composant connexe (sous-hypergraphe) en transformant chaque sous-hypergraphe en un graphe orienté. Cette génération est pilotée par un modèle de coûts qui permettent d'ordonner les nœuds. Cette étape est nécessaire pour ordonner les nœuds de jointure dans chaque composant. Ajouter un arc au graphe orienté correspond à mettre un ordre entre deux nœuds de jointure. Le processus de transformation comporte trois étapes principales :

(a) choisir le nœud pivot , le nœud pivot (défini à 4.3.9) correspond au nœud qui offre le meilleur bénéfice possible s'il est matérialisé. Le bénéfice de chaque nœud ni est calculé sur la base de notre modèle de coût (décrit en annexe A) en utilisant l'équation suivante :

$$benefice(n_i) = (nbr\_utilisation - 1) \times coût\_traitement(n_i) - coût\_constr(n_i) \tag{4.2}$$

où nbr_utilisation , coût_traitement(ni) et coût_constr(ni) représentent respectivement le nombre de requêtes utilisant le nœud de jointure n_i, le coût de traitement de n_i et le coût de construction de n_i. Dans notre étude, nous supposons que la jointure par hachage est utilisée pour traiter les opérations de jointure. Le coût d'une jointure impliquant deux tables T_1 et T_2 est donné par la formule suivante : $3 \times (|T_1| + |T_2|)$, où $|T_1|$ représente le nombre de pages de la table T_1 et $|T_2|$ représente le nombre de pages de la table T_2. Le coût de construction d'un nœud n_i est défini comme la somme du coût de sa génération et de son stockage.

(b) Transformer le nœud pivot de l'hypergraphe au graphe orienté.

(c) Supprimez le nœud pivot de l'hypergraphe. Nous mentionnons que nous nous sommes inspirés du travail proposé par [92] pour générer le plan de requêtes unifié qui a assuré la scalabilité de notre approche. La figure 4.6 montre un exemple pour l'étape de transformation de l'hypergraphe en un graphe orienté. Au final, les nœuds de jointure ayant un bénéfice positif sont sélectionnés comme candidats à la matérialisation. Dans cet exemple, les deux nœuds J_1 et J_2 sont sélectionnés comme vues candidates.

5. **ordonnancement de requêtes** Pour augmenter le bénéfice et la réutilisation des vues matérialisées avant leur suppression, nous proposons de re-ordonner les requêtes de l'ensemble Q_{first}. Nous formalisons notre problème d'ordonnancement de requêtes comme suit : pour un plan de requêtes unifié (UQP) donné , ce module prend en entrée l'ensemble des requêtes de ce composant et leurs nœuds de jointures déjà sélectionnés comme candidats à la matérialisation. Notre module d'ordonnancement vise à donner un nouvel ordre pour les requêtes de ce composant, en maximisant le bénéfice net de l'utilisation de vues matérialisées et en réduisant le coût de traitement global des requêtes. En fait, nous nous sommes inspirés des travaux proposés dans [209] montrant l'efficacité de leurs algorithmes d'ordonnancement pour maximiser le bénéfice des vues matérialisées et optimiser le coût de traitement des requêtes. Contrairement à cet algorithme dans lequel les vues matérialisées sont toutes supprimées après leur utilisation par les requêtes appropriées. Dans notre proposition, nous gardons le nœud

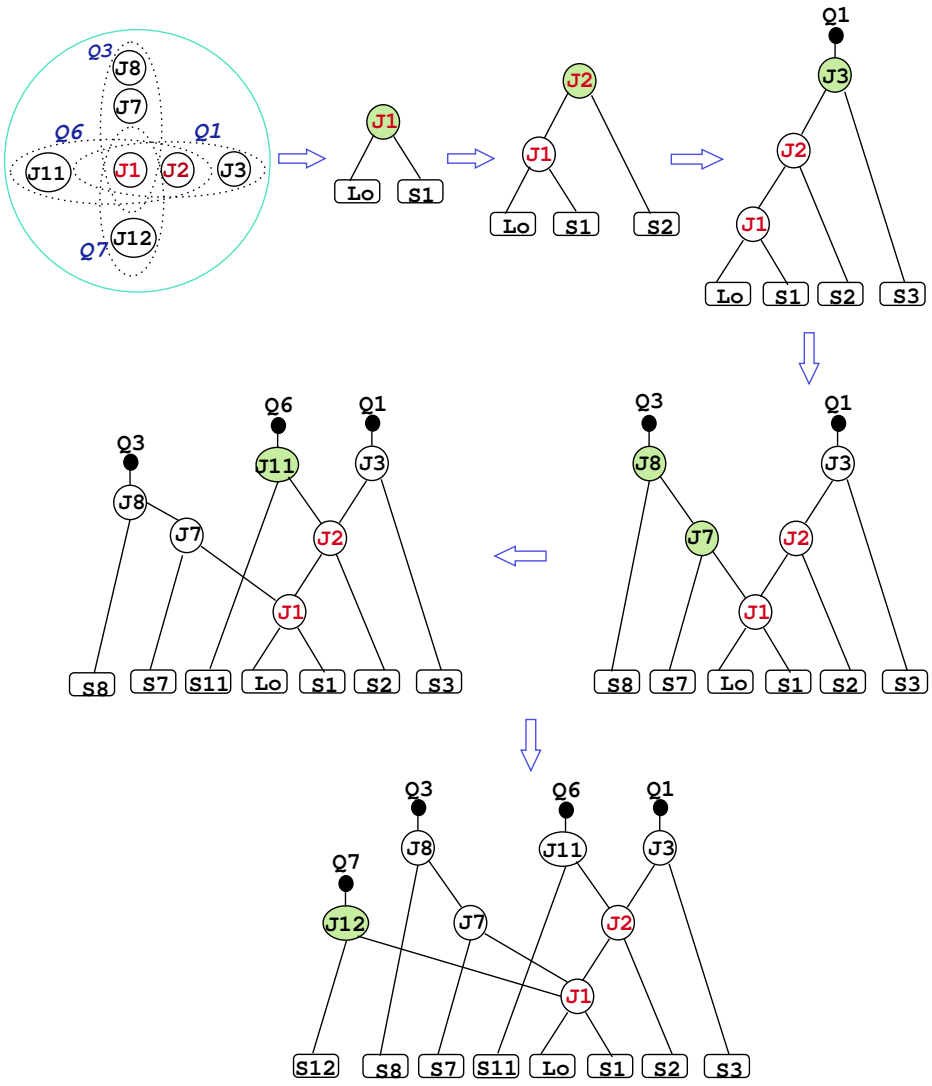


FIGURE 4.6 – Transformation d'un hypergraphe à un plan de requêtes unifié

Pivot de chaque composant pour servir les futures requêtes entrantes. Les nœuds pivots représentent les vues ayant le bénéfice maximal pour chaque composant. Pour clarifier notre processus d'ordonnancement, considérons le composant hypergraphe illustré sur la figure 4.7 dans lequel 4 requêtes sont impliquées. Dans cet exemple, deux nœuds de jointure sont sélectionnés comme candidats pour la matérialisation : {J1 , J2 } qui sont écrits en rouge. La première étape de notre processus consiste à ordonner les nœuds de jointure en fonction de leurs bénéfices. Deuxièmement, chaque requête est affectée à un poids calculé en additionnant le bénéfice de ses nœuds reine utilisés. Enfin, nous ordonnons les requêtes en fonction de leur poids.

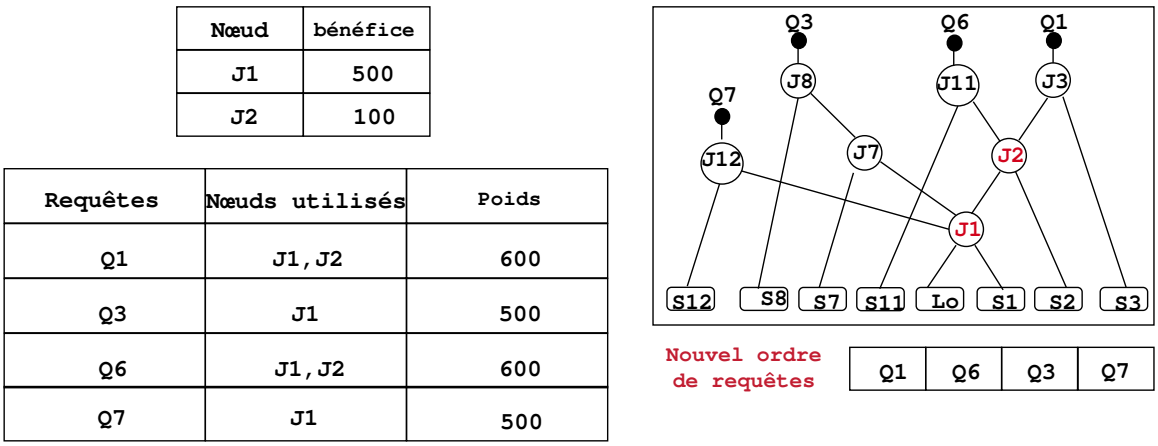


FIGURE 4.7 – Processus d'ordonnancement des requêtes

4.4.3 La phase en ligne

Dans cette section, nous discutons l'exploitation de l'hypergraphe généré par la phase hors ligne pour gérer/optimiser les requêtes ad-hoc qui arrivent dynamiquement. Pour gérer l'arrivée dynamique des requêtes ad-hoc, un ensemble de primitives est proposé pour augmenter de façon incrémentale l'hypergraphe global construit dans la première phase (hors ligne) comme : ajouter-nœud(), ajouter-hyperarête(), supprimer-nœud(), supprimer-hyperarête (), etc. Lors de l'arrivée de ces requêtes, un ensemble de vues matérialisées est sélectionné dynamiquement en fonction des sous-expressions (Nous nous intéressons aux opérations de jointures) partagées identifiées via notre hypergraphe et une fonction de bénéfice prenant en compte les contraintes de stockage et de maintenance. A chaque instant t de l'arrivée de ces requêtes, le contenu et la taille de l'hypergraphe global et du pool de vues matérialisées changeront dynamiquement. Deux modules principaux caractérisent la phase en ligne.

4.4.3.1 Enrichissement de l'hypergraphe :

Pour assurer un enrichissement efficace de l'hypergraphe global, nous proposons une stratégie qui consiste à placer les requêtes entrantes dans le composant (sous-hypergraphe) approprié et à matérialiser les jointures les plus partagées identifiées dynamiquement par notre hypergraphe, dans le but de les réutiliser par de futures requêtes. Notre stratégie pour placer ces requêtes est basée sur deux critères définis entre la requête entrante Q^t et les sous-hypergraphes existants. Cela se fait en respectant les principes suivants :

1. **Priorité 1 :** Afin d'augmenter la réutilisation des vues matérialisées déjà sélectionnées et stockées dans le pool de chaque sous-hypergraphe, nous avons défini une métrique entre la requête entrante Q^t et un sous-hypergraphe HG_I^t. Cette métrique est appelée $Nbr\_Vues\_partagées$ et est définie comme suit :

$$Nbr\_Vues\_partagées(Q^t, HG_i^t) := |Jointures(Q^t) \bigcap Pool^{HG_i^t}| \quad (4.3)$$

La métrique ci-dessus représente le cardinal de l'intersection entre l'ensemble des nœuds de jointure de la requête entrante Q^t et l'ensemble des nœuds de jointure **déjà matérialisés** pour le $i^{ème}$ sous-hypergraphe HG_I^t. Après avoir calculé cette métrique entre Q^t et chaque sous-hypergraphe. La requête Q^t doit être placée dans le composant (sous-hypergraphe) qui maximise cette métrique. Après avoir placer la requête Q^t dans le sous-hypergraphe approprié, nous réécrivons la requête de manière transparente en ré-utilisant la meilleur vue (parmi les nœuds de Q^t qui sont déjà matérialisées) qui entraîne le meilleur coût pour Q^t (on estime le coût de la requête en utilisant notre modèle de coût défini en annexe A).

2. **Priorité 2 :** Si la requête Q^t ne partage aucune opération de jointures déjà matérialisée avec les sous-hypergraphes existants, Q^t est alors placé dans le composant qui partage avec elle le maximum de nœuds de jointure (non pas encore matérialisées). Pour ce faire, nous avons défini une nouvelle métrique appelée Poids_jointures_partagées (Q^t,HG_I^t) (implémentée dans l'algorithme2). Cette métrique est calculée entre la requête Q^t et chaque sous-hypergraphe HG_I^t. La métrique Poids_jointures_partagées est liée au degré de chaque nœud de jointure appartenant à Q^t dans la matrice d'incidence d'un sous-hypergraphe donné. Le degré du nœud est renvoyé à partir de la dernière colonne de chaque matrice d'incidence que nous étiquetons la colonne FRQ. Pour définir la métrique Poids_jointures_partagées , nous devons d'abord définir un algorithme qui calcule les degrés de chaque nœud de jointure de la requête Q^tt dans la matrice d'incidence de la $i^{ème}$ sous-hypergraphe en utilisant l'algorithme1.

Algorithm 1: Calculer le degré

Inputs un nœud de jointure : V_i^t, la matrice d'incidence : $M^{HG_i^t}$;
Outputs le degré calculé : $Degree$;
if $V_i^t \notin nœuds(HG_i^t)$ **then**
    $Degree$=0;
    /* si le sommet V_i^t n'appartient pas à l'ensemble des sommets du sous-hypergraphe HG_i^t, alors son degré est nul */
else
    $Degree = \sum_{j=1}^{Number\_Columns-1} M^{HG_i^t}(i,j)$;
    /* Nous supposons que le nœud de jointure V_i^t est positionné dans la $i^{ème}$ ligne */
end

En utilisant l'algorithme ci-dessus (Algorithme1), nous avons défini l' algorithme qui calcule le poids des jointures partagées de la requête entrante Q^t dans la matrice d'incidence du $i^{ème}$ composant d'hypergraphe HG_I^t.

Algorithm 2: Calculer poids des jointures partagées

Inputs La requête entrante : Q^t, le $i^{ème}$ sous-hypergraphe : HG_i^t;
Outputs le poids calculé : $Poids\_Jointures\_partagées$;
$Poids\_Jointures\_partagées \leftarrow 0$;
foreach $nœud \in noeuds\_de\_jointure(Q^t)$ **do**
    $Degree \leftarrow Calculer\ le\ degré(nœud, M^{HG_i^t})$;
    if $Degree \neq 0$ **then**
        $Poids\_Jointures\_partagées \leftarrow Poids\_Jointures\_partagées + 1$;
    end
end

Les changements dynamiques dans les composants hypergraphiques conduisent à une mise à jour dynamique de leurs matrices d'incidence. Pour mettre à jour une matrice d'incidence d'un sous-hypergraphe donné, nous avons défini l'algorithme Mettre à jour une matrice d'incidence (Q^t, $M^{HG_i^t}$) qui permet d'ajouter des colonnes, des lignes et de mettre à jour la colonne FRQ qui représente le degré de nœuds dans la matrice d'incidence. Nous effectuons cette tâche lorsqu'une nouvelle requête est placée dans le $i^{ème}$ composant de l'hypergraphe globale. Si la requête Q^t ne partage aucune jointure avec les composants existants, un nouveau composant hypergraphe est construit et associé à cette requête.

Algorithm 3: Mettre à Jour la Matrice d'Incidence

Inputs La requête entrante : Q^t , la matrice d'incidence : $M^{HG_i^t}$;
Outputs La matrice d'incidence mise à jour : $M^{HG_i^t}$;
ajouter la colonne Q^t à la matrice $M^{HG_i^t}$;
foreach $nœud \notin nœuds(Q^t)$ **do**
    if $nœud \notin nœuds(HG_i^t)$ **then**
        ajouter une nouvelle ligne à la matrice $M^{HG_i^t}$;
        $Degree(nœud, M^{HG_i^t}) = 1$;
    else
        $Degree(nœud, M^{HG_i^t}) = Degree(nœud, M^{HG_i^t}) + 1$;
    end
end

4.4.3.2 Re-sélection dynamique des vues matérialisées :

L'arrivée dynamique et continue des requêtes entraîne un nombre massif de vues candidates à la matérialisation ce qui viole la contrainte d'espace de stockage. En fait, il est impossible de matérialiser toutes les vues candidates dans une telle situation. Pour faire face à ce problème, nous proposons une stratégie qui consiste à matérialiser les opérations de jointures les plus partagées pouvant être utilisées par les requêtes entrantes à l'avenir. Pour ce faire, on réitère le processus suivant à chaque arrivée d'une nouvelle requête à un composant (sous-hypergraphe) jusqu'à la saturation de l'espace de stockage (fixé à l'avance) :

À l'arrivée d'une nouvelle requête Q^t au $i^{ème}$ sous-hypergraphe HG_i^t, on calcule le bénéfice de chaque opération de jointure appartenant à la requête Q^t en utilisant notre fonction de calcul des bénéfices (voir équation 4.2). Ensuite, nous vérifions s'il existe des jointures ayant un bénéfice positif. Dans ce cas, nous appelons l'algorithme $Mettre$ à $Jour\ les\ Vues\ Matérialisées$ pour mettre à jour le pool du $i^{ème}$ composant HG_i^t en matérialisant les jointures qui ont un bénéfice positif. Si l'espace de stockage est saturé, on supprime les vues matérialisées ayant le moins bénéfice dans le pool de ce composant et nous les remplaçons par la matérialisation des jointures ayant un bénéfice positif de la requête courante. Nous présentons dans l'algorithme ci-dessous (algorithme 5) notre processus général de construction incrémentale de l'hypergraphe et de sélection dynamique des vues matérialisées durant la phase en ligne.

Algorithm 4: Mettre à Jour les Vues Matérialisées

Inputs la requête : Q^t ; le composant hypergraphique : Fi ;
le pool de vues associé au composant Fi : $Pool^{F_i}$; espace disque : DS;
Output le pool mis à jour du composant Fi : $Pool^{F_i}$;
$joins \leftarrow Retourner\_jointures(Q^t)$;
Calculer_benefices (Joins);
$L \leftarrow Retourner\_Jointures\_ayant\_Benefice\_positif(Joins)$;
Tri_descendant(L) ;
/* selon leurs bénéfices */
foreach $nœud \in L$ **do**
  if $nœud \notin Pool^{F_i}$ And $taille(Pool^{F_i}) + taille(nœud) < DS$ **then**
    Matérialiser (nœud);
    $Pool^{F_i}.ajouter(nœud)$;
    $taille(Pool^{F_i}) \leftarrow taille(Pool^{F_i}) + taille(nœud)$;
  else
    if $nœud \notin Pool^{F_i}$ And $taille(Pool^{F_i}) + taille(nœud) > DS$ **then**
      Tri_croissant($Pool^{F_i}$) ;
      /* selon leurs bénéfices */
      $idx \leftarrow 0$;
      repeat
        if $benefice(nœud) > benefice(Pool[idx])$ **then**
          Dématérialiser (Pool [idx]) ;
          $taille(Pool^{F_i}) \leftarrow taille(Pool^{F_i}) - taille(nœud)$;
          $idx \leftarrow idx + 1$;
        end
      until $taille(Pool^{compi}) + taille(nœud) < DS$ OR
       $benefice(nœud) < benefice(Pool[idx])$;
      if $taille(Pool^{compi}) + taille(nœud) < DS$ **then**
        Matérialiser (nœud);
        $Pool^{F_i}.ajouter(nœud)$;
        $taille(Pool^{F_i}) \leftarrow taille(Pool^{F_i}) + taille(nœud)$;
      end
    end
  end
end

Algorithm 5: Sélection dynamique des vues matérialisées sur la base d'hypergraphe

Inputs : la requête entrante Q^t à l'instant t ; Espace de stockage Sp;
Outputs : une liste de composants (F) de HG ; pool de vues pour chaque composant;
loop;
$t := 1$;
Analyseur_requête(Q^t);
if $|F^t| = 0$ **then**
    $F_i^t = Construire\_nouveau\_composant()$;
    $ajouter\_hyperarête(F_i^t, Q^t)$;
    $M^{F_i^t} := Calculer\_matrice\_incidence(F_i^t)$;
    $F.ajouter(F_i^t)$;
else
    foreach $F_i^t \in GH^t$ **do**
        $Nbr\_vues\_partagés^t := |Jointures(Q^t) \bigcap Pool^{F_i^t}|$;
        $List_1^t.ajouter(Nbr\_vues\_partagés^t)$;
        $Nbr\_Jointure\_partg^t := Calculer\ poids\ de\ jointures\ partagés(Q^t, F_i^t)$;
        $List_2^t.ajouter(Nbr\_Jointure\_partg^t)$;
    end
    $Maximum_1^t := Maximum(List_1^t)$;
    if $Maximum_1^t = 0$ **then**
        $Maximum_2^t := Maximum(List_2^t)$;
        if $Maximum_2^t = 0$ **then**
            $Construire\_nouveau\_composant(F_i^t)$;
            $ajouter\_hyperarête(F_i^t, Q^t)$;
            $M^{F_i^t} := Calculer\_matrice\_incidence(F_i^t)$;
            $F.ajouter(F_i^t)$;
        else
            $pos^t := Return\_pos(Maximum_2^t, List_2^t)$;
            $ajouter\_hyperarête(F.get(pos^t), Q^t)$;
            $Mettre$ à $Jour\ la\ Matrice\ d'Incidence(Q^t, M^{F.get(pos)^t}$;
            $Mettre$ à $Jour\ Vues\ Matérialisées(Q^t, F.get(Pos^t), Pool^{F.get(Pos^t)}, Sp)$;
        end
    else
        $Pos^t := Retourner\_position(Maximum_1^t, List_1^t)$;
        $ajouter\_hyperarête(F.get(pos^t), Q^t)$;
        $Récrire(Q^t, Pool^{F.get(Pos)^t})$;
        $Mettre$ à $Jour\ la\ Matrice\ d'Incidence(Q^t, M^{F.get(pos)^t})$;
        $Mettre$ à $Jour\ les\ Vues\ Matérialisées\ (Q^t, F.get(Pos^t), Pool^{F.get(Pos^t)}, Sp)$;
    end
end
$t := t + 1$;
END LOOP

Exemple 4.4.6 *Pour illustrer notre stratégie dynamique de gestion des requêtes ad-hoc, nous considérons l'hypergraphe montré à la figure 4.3 qui a été construit pour une charge de sept requêtes (définie à l'annexe B), et nous supposons l'arrivée de la requête Q^t définie par le code SQL suivant :*

La Requête Q^t

```
SELECT c_city, s_city, d_year, sum(lo_revenue) as revenue
FROM   CUSTOMER, LINEORDER, SUPPLIER, DATES
WHERE  lo_custkey = c_custkey    (J7)
AND    lo_suppkey = s_suppkey    (J13)
AND    lo_orderdate = d_datekey  (J1)
AND    (c_city='IRAQ     4' or c_city='JORDAN   6')
AND    s_region = 'AUSTRALIE'
AND    d_year >= 1992 and d_year <= 1997
group by    c_city, s_city, d_year
order by    c_city, s_city, d_year;
```

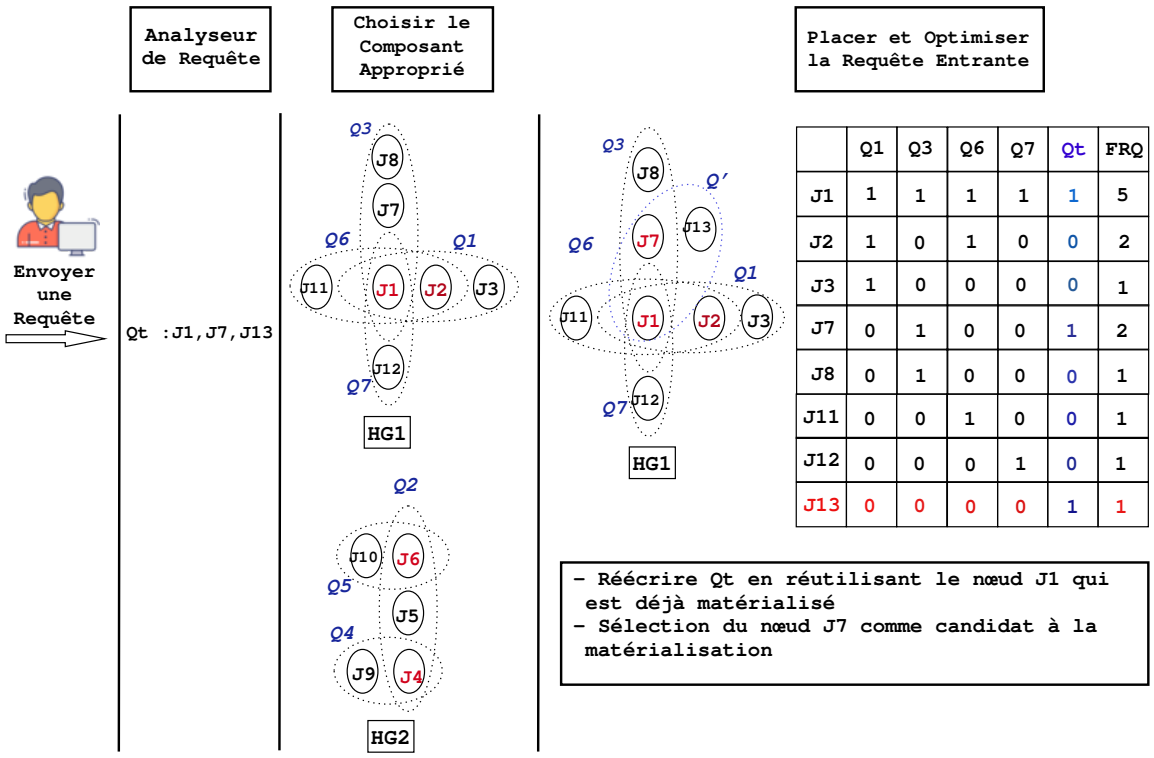
La figure 4.8 montre notre stratégie de traiter la requête Q^t , où la requête a été placé dans le composant (sous-hypergraphe) $HG1$ qui partage avec elle le nœud de jointure $J1$ déjà matérialisée et le nœud de jointure $J7$ qui ne s'est pas encore matérialisée. La requête Q^t a été optimisée par la ré-utilisation de la vue matérialisée de la jointure $J1$. On note aussi que l'arrivée de Q^t a conduit à la matérialisation de la jointure $J7$ où son bénéfice est devenu positif.

4.5 Complexité de nos algorithmes

Comme nous l'avons expliqué dans les sections précédentes, notre approche comporte deux principales phases. La complexité des algorithmes de chaque phase est donnée séparément dans ce qui suit.

la phase hors ligne comporte cinq étapes séquentielles principales. Nous donnons dans ce qui suit la complexité de chaque étape :

- La première étape consiste à analyser les requêtes pour identifier les opérations logiques de chaque requête afin de construire l'hypergraphe global. Cette tâche a une complexité linéaire égale à $O(N*M)$ où N et M représentent respectivement le nombre de requêtes et le nombre maximal d'opérations utilisées par une requête donnée. Dans le contexte des entrepôts de données, si on a D tables de dimension, on aura au maximum $(D+1)$ sélections (D sélections sur les tables de dimensions et une sur la table de faits), D jointures, D projections et une agrégation, ce qui donne $M = D+1+D+D+1 = 3*D+1$.



	Q1	Q3	Q6	Q7	Qt	FRQ
J1	1	1	1	1	1	5
J2	1	0	1	0	0	2
J3	1	0	0	0	0	1
J7	0	1	0	0	1	2
J8	0	1	0	0	0	1
J11	0	0	1	0	0	1
J12	0	0	0	1	0	1
J13	0	0	0	0	1	1

FIGURE 4.8 – Le processus d'optimisation d'une requête ad-hoc

- La deuxième étape consiste à construire l'hypergraphe global. Cette tâche a une complexité linéaire égale à $O(N * D)$, où N et D représentent respectivement le nombre de requêtes et le nombre maximal d'opérations de jointures utilisées par une requête donnée. Cette tâche nécessite l'ajout des hyper-arêtes avec ses différents sommets à l'hypergraphe global, où les hyper-arêtes et les sommets d'un hypergraphe représentent respectivement l'ensemble des requêtes et les opération de jointures impliquées dans chaque requête.

- La troisième étape concerne le processus de partitionnement hypergraphique effectué par l'algorithme hMeTiS. La complexité de hMeTiS a été étudiée à [237], où les auteurs ont montré que pour un partitionnement K-way la complexité est égale à $O((V + N) * log(K))$, où V est le nombre de sommets (dans notre cas, V est le nombre d'opérations de jointures impliqués dans la charge de requêtes) et N est le nombre d'hyper-arêtes (nombre de requêtes de la charge).

- La quatrième étape est la transformation de chaque sous-hypergraphe en un graphe orienté. Au cours de cette étape, nous devons identifier successivement le nœud pivot (pivot) puis le supprimer du sous-hypergraphe jusqu'à ce que tous les nœuds aient été supprimés. La complexité de cette étape a été démontré à [236] par $O(V^2)$, où V est le nombre de sommets à l'intérieur du composant. Chaque composant a un nombre limité de requêtes et peut être transformé indépendamment, ce qui permet à notre approche d'évoluer.

- La dernière étape est l'ordonnancement des requêtes de chaque composant. Cette tâche comporte deux étapes principale : la première étape consiste à calculer le poids de chaque requête du composant. Ensuite, ordonner les requêtes de chaque composant selon leurs poids calculés. La complexité de la première étape est égale à $O(N * P)$, où N est le nombre de requête du composant est P est le nombre de noeuds reines du composant (qui ont un bénéfice positif). La complexité de deuxième étape est égale à $O(N * log(N))$, tel qu'on utilise le tri par fusion pour ordonner les requêtes par ordre descendant selon leur poids (N est le nombre de requêtes du composant).

Dans ce qui suit , nous évaluons la complexité de divers algorithmes liés à la construction de l'algorithme5 qui gère le processus d'optimisation des requêtes ad-hoc dans la phase en ligne, où la complexité de chaque algorithme est donnée d'une manière séparée.

- La complexité de la fonction qui calcule le degré d'un nœud (voir algorithme1) est une complexité linéaire égale à $O(|E|)$ où $|E|$ est le nombre d'hyper-arêtes dans l'hypergraphe représenté par cette matrice d'incidence.

- La complexité de la fonction qui calcule le nombre des nœuds de jointure partagés entre une requête entrante Q_{incom} et un hypergraphe donné (voir algorithme2) est une complexité linéaire estimée par : $O(|nœud(Q_{incom})| * |E|)$, tel que $|nodes(Q_{incom})| = D$. Si on a D tables de dimension, on aura au maximum D opérations de jointures impliquées dans la requête Q_{incom}, et $|E|$ est le nombre d'hyper-arêtes dans l'hypergraphe impliqué dans cette fonction ($|E|$ est la complexité de l'algorithme1 appelé par cet algorithme).

- La complexité de l'algorithme3 qui met à jour la matrice d'incidence d'un hypergraphe donné est linéaire. Cette complexité est définie par : $|nœud(Q_{incom})| = D$, où D est le maximum de nœuds de jointures impliqué dans la requête Q_{incom} (si on a D tables de dimensions).

- La complexité de l'algorithme4 qui met à jour le pool des vues matérialisées d'un sous-hypergraphe donnée est une complexité quasi-linéaire définie par : $O(|L| * (log(|L|) + |Pool^{F_i}| * log(|pool^{F_i}|)))$, tel que L est la liste des nœuds de jointures de la requête Q_{incom} ayant un bénéfice positif et $Pool^{F_i}$ représente le pool des vues du composant (sous-hypergraphe) F_i. On note ici que nous utilisons la méthode de tri par fusion pour trier la liste L et le pool des vues $Pool^{F_i}$ selon les bénéfices des nœuds.

Dans notre approche, les requêtes et leurs opérations de jointures sont présentées par un hypergraphe. Une hyper-arêtes que représente une requête peut connecter des milliers de nœuds , ce qui permet à notre approche d'évoluer avec le nombre de tables de dimension de l'entrepôt de données.

4.6 Conclusion

Dans les chapitre précédents, nous avons identifié les limites des travaux existants traitant de l'optimisation multi-requêtes et de sélection des vues matérialisées dans la gestion simultanée des propriétés récentes des charges de requêtes analytiques. Ces charges de requêtes impliquent de très grand nombre de requêtes dynamiques et très interactives. Dans ce chapitre, nous proposons une approche pro-active qui utilise des hypergraphes pour la gestion de nos trois propriétés. Ces structures exploitent des modèles de coût pour capturer les sous-expressions communes des requêtes et matérialiser les plus bénéfiques.

Dans ce travail, nous nous sommes inspirés des aspects de la nouvelle BI (BI next generation), où deux types de requêtes sont considérés : des requêtes OLAP connues à l'avance et des requêtes ad-hoc. Pour traiter les deux types de requêtes, nous avons accompagné notre approche d'une stratégie qui oriente les δ premières requêtes de la charge connue vers la phase hors ligne qui sélectionne leurs vues appropriées, et ré-ordonne cet ensemble de requêtes afin d'augmenter le bénéfice et le partage des vues sélectionnées. La phase en ligne gère le pool de vues obtenu par la première phase en ajoutant/supprimant des vues pour optimiser les nouvelles requêtes entrantes (les requêtes ad-hoc). La phase hors ligne et la phase en ligne partagent l'utilisation de la même structure d'hypergraphe pour optimiser les deux types de requêtes.

Dans le prochain chapitre, nous mènerons des expériences approfondies en utilisant un benchmark du schéma en étoile pour évaluer l'efficacité et l'efficience de notre approche par rapport aux principales études d'état de l'art. Ensuite, nous présenterons notre conseiller (outil) appelé ProRes et ses différents modules qui aident l'administrateur de base de données (DBA) à effectuer la tâche de conception physique de la base de données.

Chapitre 5

Mise en œuvre et évaluation des performances

"**La sagesse commence dans l'émerveillement...**"
— —— Socrate

Sommaire

5.1 Introduction

Sur la base de nos contributions qui portent sur l'utilisation des hypergraphes pour gérer les caractéristiques récentes des requêtes analytique, et sur l'exploitation de ces structures pour la sélection dynamique des vues matérialisées. Nous présentons dans ce chapitre,

une étude de performance afin de valider et de confronter notre proposition aux principales études de l'état de l'art. Ensuite, nous introduisons notre outil inspiré par les conseillers commerciaux bien connus.

Avant de décrire et de commenter nos expériences, nous présentons l'environnement, le jeu de données et les requêtes que nous avons utilisé pour réaliser ces expériences. Ensuite, nous abordons nos expérimentations en examinant l'évolutivité (la scalabilité) de notre approche dans la génération du plan de requêtes unifié par rapport aux études connexes.Par la suite, nous menons des expérimentations intensives pour évaluer la qualité des vues matérialisées sélectionnées par notre approche par rapport à ceux choisis par les études les plus importantes de l'état de l'art, en considérant plusieurs stratégies. Nous étudions ensuite l'effet du variation du seuil δ de notre approche sur la performance des requêtes. Nos résultats ont été validés théoriquement à l'aide de notre modèle de coût et réellement dans un SGBD commercial. Finalement, nous présentons notre outil permettant de simuler tous les processus dont nous avons discuté dans cette thèse grâce à nos algorithmes et à notre modèle de coûts.

5.2 Étude de performance

Dans cette section, nous présentons une validation expérimentale de notre approche en utilisant l'environnement suivant : un serveur avec un processeur Intel Xeon E5-2690V2 de 3 GHz, 24 Go de mémoire principale (RAM) et 1 To de disque dur. Notre approche a été comparée aux deux approches suivantes :

1. l'approche proposée dans [193] (que nous appelons $YANG$) qui est considérée comme l'étude pionnière en entrepôt de données qui met en évidence la forte dépendance entre le problème d'optimisation multi-requêtes et le problème de sélection des vues matérialisées. Pour réaliser cette comparaison, nous avons développé un module pour implémenter l'approche proposée par Yang et al [193], en considérant leurs deux algorithmes : (a) un algorithme naïf qui génère tous les Mvpp (traitement de plan multi-vue) possibles et choisit le plan avec le coût minimum (b) un algorithme basé sur la programmation en nombres entiers qui est plus rapide que le premier. Le Mvpp généré par les deux méthodes (algorithmes) est utilisé pour sélectionner les vues matérialisées ayant un bénéfice positif entre le coût de maintenance des vues et le coût global de traitement des requêtes.

2. l'approche proposé dans [197] qui est considéré comme l'un des travaux les plus importants qui a mis en évidence le rôle crucial que joue l'ordonnancement des requêtes dans la gestion efficace de la sélection dynamique des vues matérialisées. Nous référençons ce travail dans nos expériences par $PHAN$. Pour PHAN, nous avons développé les fonctions suivantes : (a) un algorithme génétique qui vise à trouver l'ordre optimal

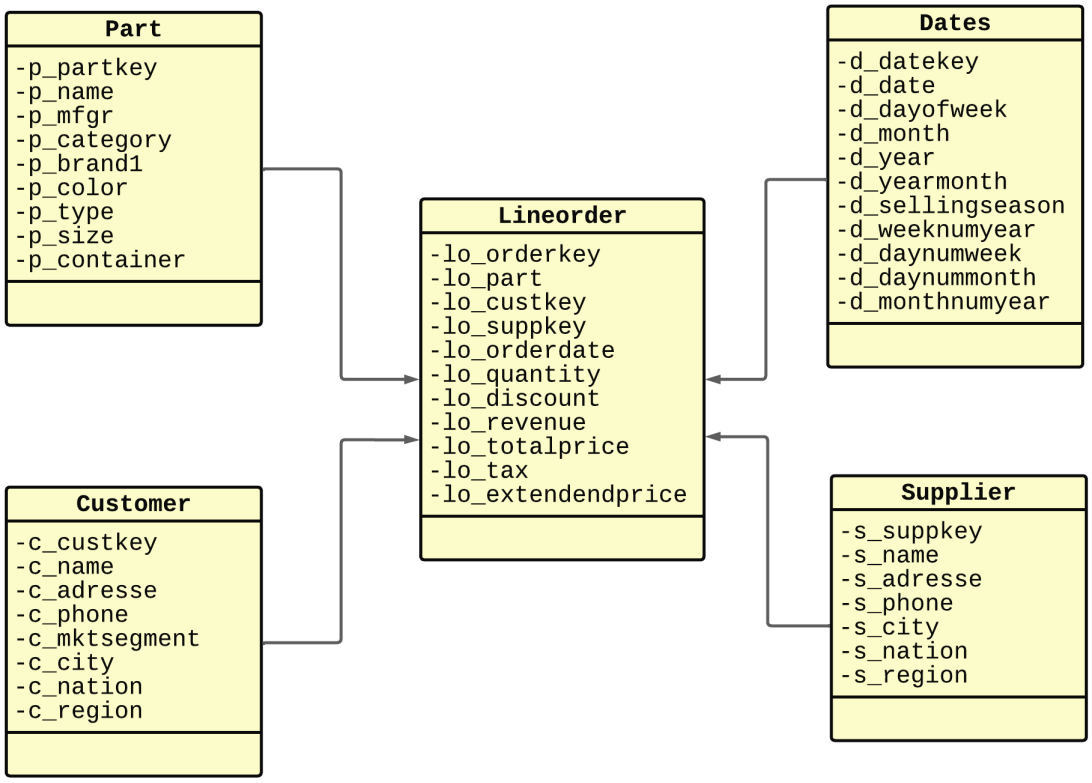


FIGURE 5.1 – schéma en étoile du benchmark SSB [1]

des requêtes en utilisant la sélection naturelle issue de la théorie de Darwin à 1000 générations. (b) un algorithme pour sélectionner les nœuds ayant le plus grand bénéfice en tant que candidats à la matérialisation (dans [197], ces nœuds sont sélectionnés à l'aide de conseiller DB2) (c) un algorithme pour élaguer l'ensemble des nœuds candidats en fonction de leur bénéfice (d) un autre algorithme pour évaluer le bénéfice net de l'ensemble élagué de vues candidates dans l'optimisation d'une charge de requêtes donnée. (e) et enfin, un algorithme de gestion du cache suivant les règles *LRU*.

5.2.1 Jeu de données

Nous utilisons dans nos expérimentations, l'entrepôt de données SSB (Benchmark du schéma en étoile) [81] composé d'une table de faits *Lineorder* et quatre tables de dimension : *Part*, *Customer*, *Supplier* et *Dates*. La figure 5.1 présente le schéma en étoile de l'entrepôt de données SSB [1]. Afin de réaliser nos expérimentations, nous avons déployé l'entrepôt de données SSB dans le SGBD Oracle 12c avec une taille de 30 Go, tout en respectant les étapes suivantes :

- Après la création d'une sessions (une connexion) Oracle (renommée *ssb*), nous créons la structure de la table de faits et de chaque table de dimension de SSB, en respectant les structures décrites en [1]. Le code suivant est un exemple du code utilisé pour créer la table de dimension *Customer*

La table *Customer*

```
CREATE TABLE Customer (
c_custkey       INTEGER NOT NULL,
c_name          VARCHAR2(25) NOT NULL,
c_address       VARCHAR2(25) NOT NULL,
c_city          CHAR(10) NOT NULL,
c_nation        CHAR(15) NOT NULL,
c_region        CHAR(12) NOT NULL,
c_phone         CHAR(15) NOT NULL,
c_mktsegment    CHAR(10) NOT NULL );
```

- La deuxième étape consiste à générer les données de l'entrepôt SSB, en spécifiant la taille souhaitée de l'entrepôt. Pour faire cela, nous utilisons l'outil *dbgen* du SSB permettant de générer les données de chaque table (Pour plus de détails, voir [1]).
- Après la génération de données, on passe à l'étape d'alimentation des tables de l'entrepôt SSB. On fait cela par l'utilisation de l'outil *sqlloader* fournit par le SGBD Oracle.

Pour tester l'efficacité et la scalabilité de notre approche, il est nécessaire de l'évaluer avec plusieurs charges de requêtes. Pour faire cela, nous avons utilisé un générateur aléatoire de requêtes SSB (appelé *qgen*) pour générer des charges de requêtes de tailles variées. Les requêtés générées couvrent la plupart des types de requêtes OLAP.

5.2.2 Scalabilité de notre approche

Cette expérimentation vise à étudier la scalabilité de notre approche par rapport à l'approche YANG [193] et à l'approche PHAN [197] dans la génération du plan de requêtes unifié (UQP) et de sélection des vues appropriées, en fonction du nombre de requêtes reçues. Pour ce faire, nous varions la taille de nos charges de requêtes de 1000 à 5000 requêtes. Les résultats obtenus sont représentés sur la Fig 5.2. Ils montrent la scalabilité de notre approche et la limitation de l'algorithme YANG dans la gestion de charges de requêtes à grande échelle, où l'algorithme de Yang [193] nécessite d'environ 14 heures pour générer un UQP pour une charge de 5000 requêtes. L'utilisation des hypergraphes et l'adaptation du partitionnement de l'algorithme hMeTis sont les facteurs clés de cette évolutivité. L'algorithme génétique proposé par [197] explore l'espace de recherche de N! représentant le nombre de permutations pour trouver le meilleur ordonnancement de requêtes. Les résultats obtenus par l'approche PHAN montrent la difficulté de leur algorithme génétique à gérer des charges de requêtes à grande échelle, alors que notre approche d'ordonnancement traite un nombre raisonnable de requêtes par composants, grâce à notre approche de partitionnement de l'hypergraphe en plusieurs composants et à notre algorithme d'ordonnancement de complexité linéaire.

1. https://clickhouse.com/docs/en/getting-started/example-datasets/star-schema/

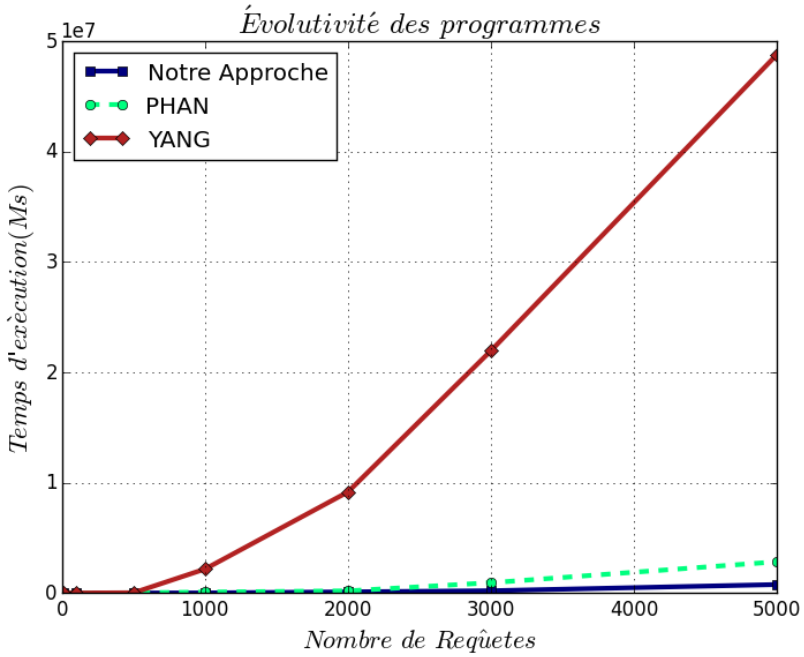


FIGURE 5.2 – Comparaison d'évolutivité

5.2.3 Nombre de vues matérialisées sélectionnées et leurs bénéfices

La sélection d'un petit ensemble de vues matérialisées optimisant l'ensemble de la charge de requêtes est l'une des mesures de qualité importantes récemment mises en évidence par un SGBD leader dans le domaine des bases de données [9]. Par conséquent, nous avons mené des expériences pour évaluer cette métrique en considérant deux charges différentes avec 100 et 1000 requêtes générées aléatoirement. La principale leçon de cette expérimentation (illustrée dans la figure 5.3) est que notre approche sélectionne moins de vues et optimise plus de requêtes par rapport aux deux autres approches. Cela est dû à son fonctionnement qui est basé sur la matérialisation de sous-expressions communes ayant un partage et un bénéfice élevés. Ces résultats sont obtenus grâce à la suppression des vues inutiles et à l'ordonnancement des requêtes de la phase hors ligne.

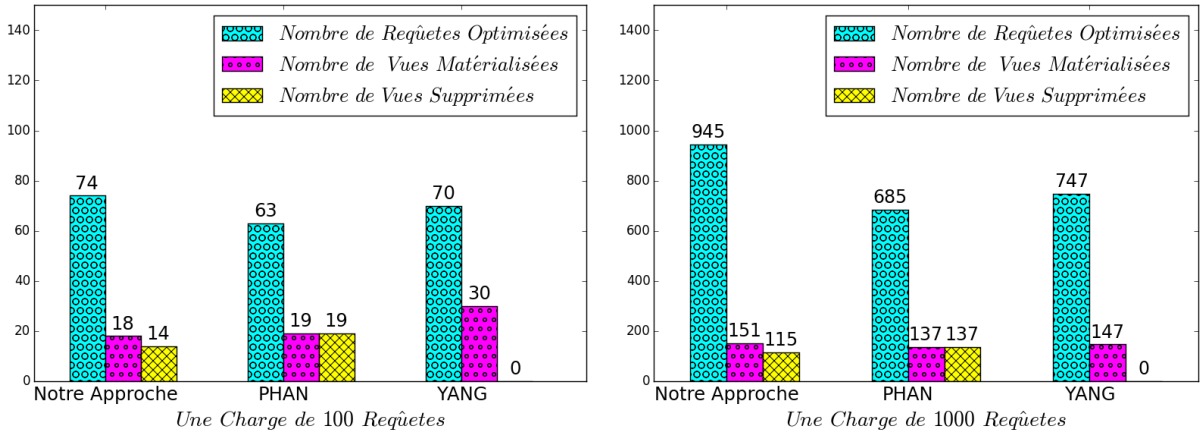


FIGURE 5.3 – Comparaison entre notre approche et les algorithmes de Phan et Yang

5.2.4 Qualité de notre modèle de coût

Notre proposition est basée sur un composant très important représentant un modèle de coût mathématique lié au coût de traitement des requêtes avec et sans l'utilisation des vues, au coût de construction des vues matérialisées et aux contraintes de stockage. Ce modèle de coût se trouvent en annexe B. Nous étudions dans cette expérimentation les contributions des vues matérialisées sélectionnées par notre approche et les autres algorithmes sur le coût global de traitement des requêtes et le coûts de matérialisation des vues sélectionnées. Pour ce faire, nous avons évalué théoriquement les différents coûts à l'aide de notre modèle de coûts mathématique . Les résultats obtenus sont reportés sur la figure 5.4. Les vues sélectionnées par notre approche sont plus avantageuses que celles générées par les autres algorithmes. Cela est dû à notre stratégie de matérialisation qui sélectionne les candidats les plus avantageux et à notre politique d'ordonnancement qui permet d'augmenter (maximiser) le bénéfice des vues sélectionnées.

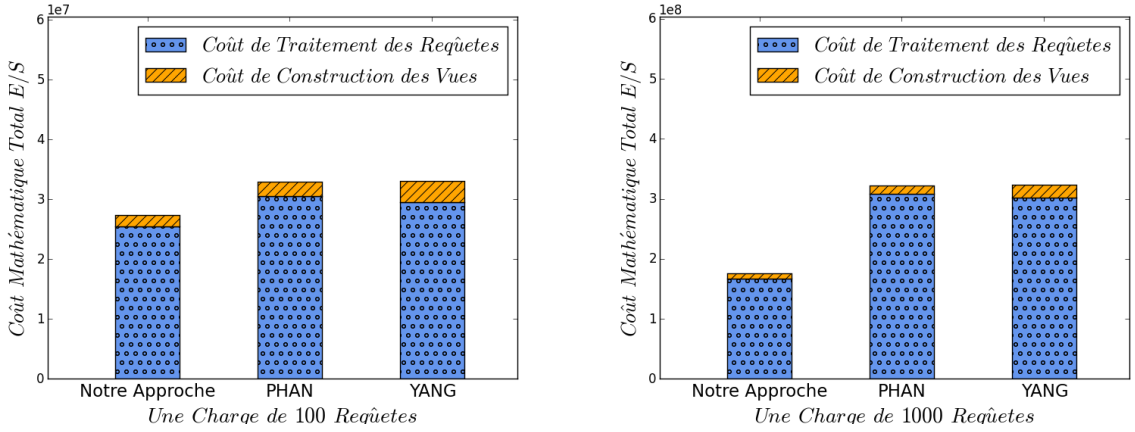


FIGURE 5.4 – Comparaison entre les trois approches en termes de coûts de traitement de requêtes/construction des vues

5.2.5 Validation Oracle

Le but de cette expérimentation est de valider les résultats obtenus théoriquement dans les expérimentations précédentes. Pour ce faire, nous considérons une charge de 100 requêtes exécutées sur un entrepôt de données avec 30 Go déployés dans le SGBD Oracle 12c. L'espace de stockage des vues matérialisées est défini sur 60 Go. Les résultats obtenus en termes de temps d'exécution (en secondes) sont reportés sur la Figure 5.5. Nous observons que notre approche surpasse les autres algorithmes. Les résultats obtenus coïncident avec ceux obtenus théoriquement, ce qui confirme l'efficacité et la supériorité de notre approche.

5.2.6 Importance du dynamisme et d'ordonnancement sur l'optimisation des requêtes

Pour tester l'efficacité du dynamisme et de l'ordonnancement des requêtes et afin d'analyser le comportement des différentes algorithmes (notre algorithme et les algorithmes de PHAN et YANG), nous envisageons les deux scénarios suivantes :

1. **Matérialisation statique** : Dans cette expérimentation, nous considérons un scénario naïf dans lequel les nœuds de jointure sélectionnés par chaque algorithme sont matérialisés jusqu'à la saturation de l'espace de stockage (i.e., sans supprimer les vues qui sont matérialisées et déjà utilisées). Pour réaliser cette expérimentation, nous avons considéré un entrepôt de données avec 30 Go et une charge de 100 requêtes utilisant le benchmark du schéma en étoile SSB . En analysant la figure 5.6a, on remarque que notre approche surpasse les algorithmes de Yang et PHAN, ce qui prouve que notre approche n'évite pas la sélection des meilleurs vues à matérialiser même dans le cas statique. Cela prouve aussi l'efficacité de notre algorithme d'ordonnancement de requêtes, où chaque vue matérialisé est utilisée par le maximum de requêtes.

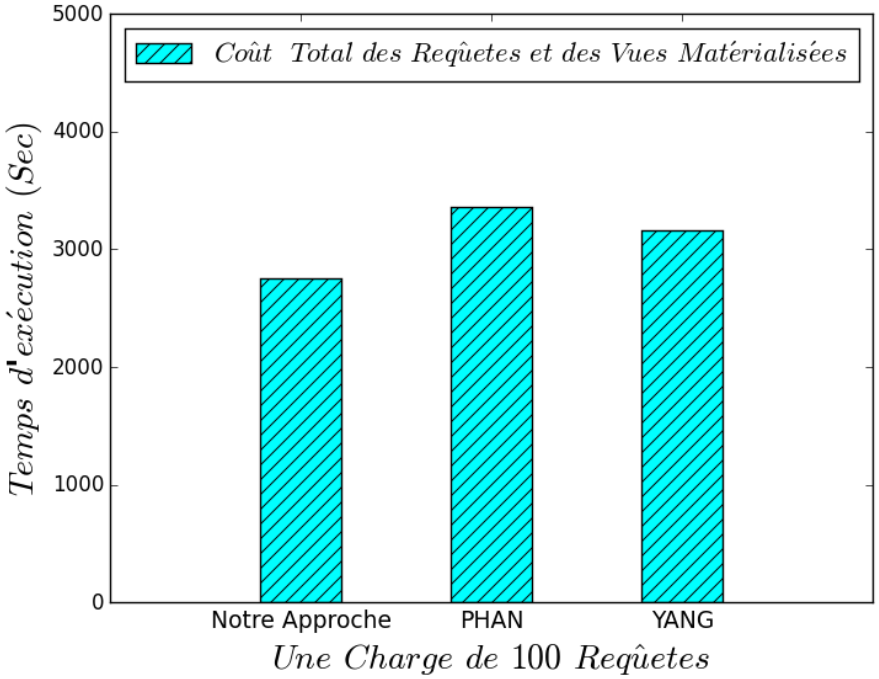
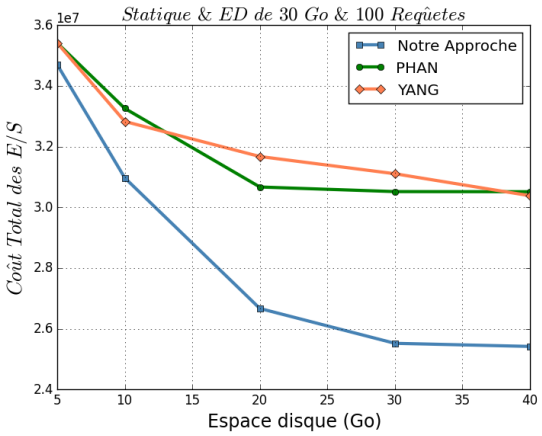


FIGURE 5.5 – Validation Sur Oracle

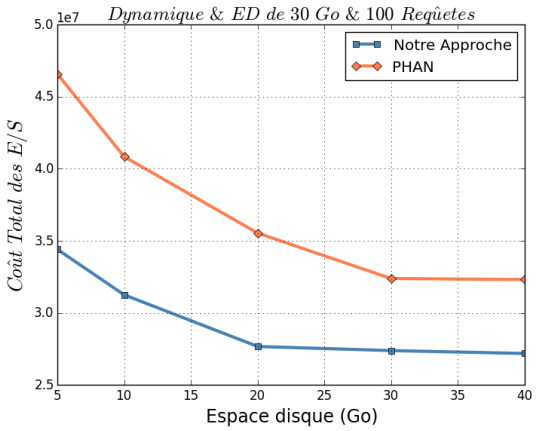
2. **Matérialisation dynamique avec ordonnancement des requêtes** : pour réaliser cette expérimentation, nous avons utilisé les mêmes données ci-dessus en considérant un benchmark de données SSB avec 30 Go et une charge de 100 requêtes générées de manière aléatoire à l'aide du générateur SSB. Comme le montre la figure 5.6b, notre approche surpasse largement l'algorithme de Phan. Cela est dû au nombre minimal de suppressions de vues dans notre approche par rapport à celle de Phan. De plus, les vues matérialisées sélectionnées par notre approche sont utilisées au maximum pour optimiser les requêtes appropriées avant qu'elles soient supprimées, cela est dû à notre politique d'ordonnancement de requêtes qui augmente les bénéfices des vues matérialisées sélectionnées.

5.2.7 Impact du seuil δ sur le coût de traitement des requêtes

Dans cette expérimentation, nous évaluons d'abord théoriquement (à l'aide de notre modèle de coût) puis dans Oracle 12c le coût global de traitement d'une charge donnée de 100 requêtes en faisant varier la valeur du seuil δ . Les résultats obtenus sont illustrés en 5.7. Nous observons l'effet évident du seuil δ sur le coût de traitement des requêtes, où il existe une relation inverse entre les valeurs de seuil δ et le coût de traitement des requêtes. A chaque augmentation de la valeur seuil, le coût de traitement des requêtes diminue jusqu'à ce que la valeur seuil atteigne 70 qui représente le point de stabilité de l'optimisation du coût de traitement des requêtes.

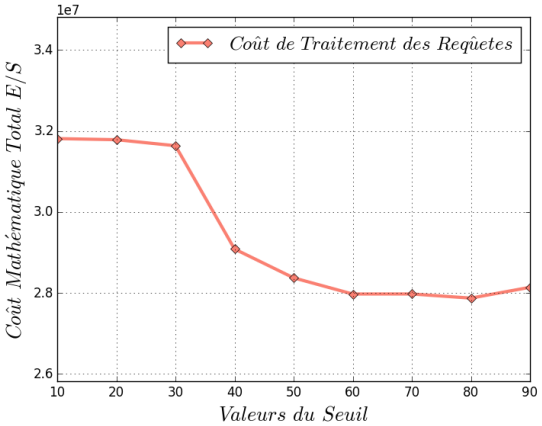


(A) Scénarios Statiques

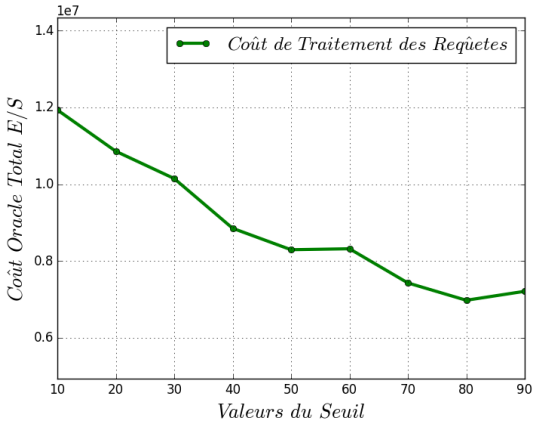


(B) Scénarios Dynamiques

FIGURE 5.6 – Performance de notre approche dans des scénarios statiques et dynamiques



(A) Du point de vue théorique



(B) Validation Oracle

FIGURE 5.7 – L'effet du seuil sur le coût de traitement de requêtes

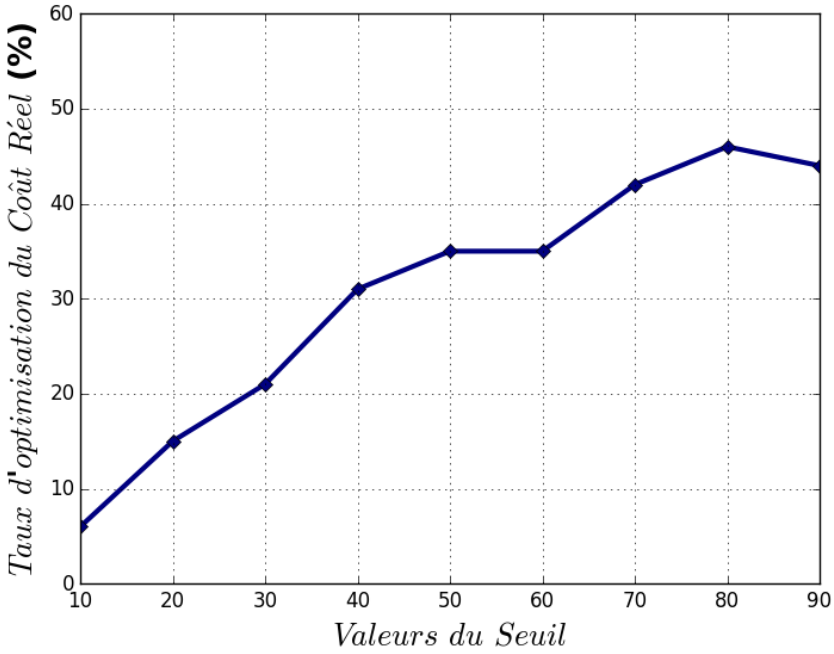


FIGURE 5.8 – Le taux de réduction du coût d'Oracle

5.2.8 Le taux de réduction du coût de traitement en fonction des valeurs de seuil δ

Dans cette expérimentation, nous suivons le même scénario ci-dessus en considérant la même charge de requêtes (de 100 requêtes) et en variant les valeurs de seuil δ. L'objectif de cette expérimentation est d'évaluer l'impact des valeurs seuils et de la qualité des vues matérialisées sélectionnées par notre stratégie proactive sur le coût global de traitement des requêtes. Pour ce faire, nous estimons dans un premier temps le coût global réel (coût Oracle) de traitement de la charge des requêtes sans utiliser notre approche ($Cost_{without}$). Ensuite, nous estimons le coût de traitement de cette charge en utilisant notre approche (($Cost_{with}$)) en faisant varier les valeurs de seuil δ. Enfin, nous calculons le taux de réduction des coûts comme suit :

$$1 - \frac{query\ cost\ with\ views}{query\ cost\ without\ views}$$

La figure 5.8 montre les résultats obtenus implémentés dans le SGBD Oracle 12c. Les résultats obtenus confirment les résultats précédents et prouvent que notre approche devient plus intéressante lorsque le seuil hors ligne augmente jusqu'à la valeur du point de stabilité $\delta \geq 70$, où le taux de réduction des coûts est compris entre 42% et 47%. Cela est dû à l'expansion de notre pool par les vues matérialisées les plus bénéfiques.

5.2.9 Le nombre de requêtes optimisées en fonction des valeurs seuil δ

Dans la dernière expérimentation, nous essayons d'évaluer notre stratégie proactive en termes de nombre de requêtes optimisées en fonction des valeurs du seuil δ . Pour ce faire, nous considérons les mêmes scénarios ci-dessus et la même charge de requêtes. Les résultats

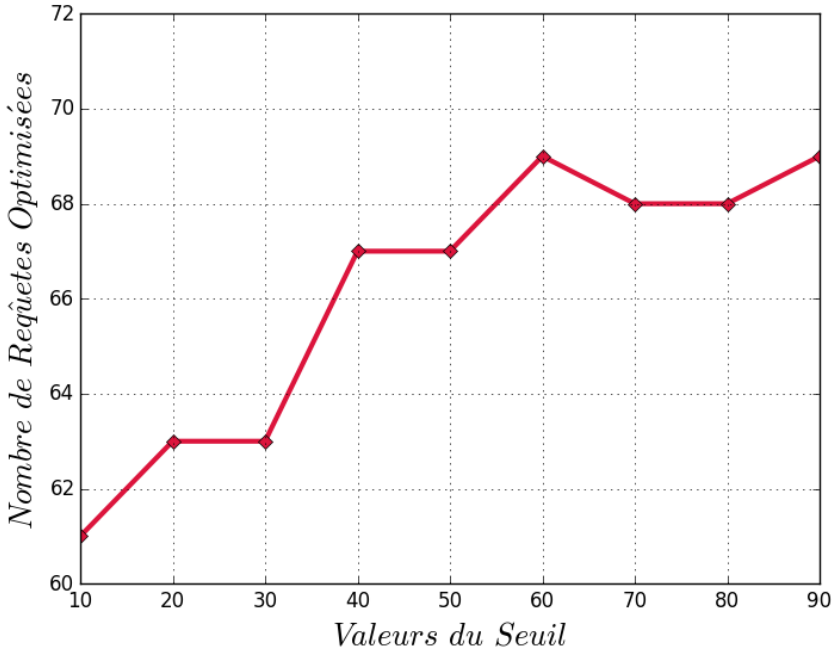


FIGURE 5.9 – Le Nombre de requêtes optimisées en fonction du seuil

obtenus sont décrits dans la figure 5.9. Ils ont montré la relation proportionnelle entre le nombre de requêtes optimisées et les valeurs de seuil δ jusqu'à la valeur $\delta \geq 60$, où il y a une stabilité dans le nombre de requêtes optimisées, où 69 requêtes ont été optimisées sur les 100 requêtes reçues.

5.3 ProRes : Un Framework pour la sélection dynamique des vues matérialisées

La conception physique est l'une des phases les plus importantes du cycle de vie d'un entrepôt de données . Cela est dû à son rôle important dans la sélection des structures d'optimisation telles que les vues matérialisées et les index pour accélérer les performances des requêtes complexes. Cette phase a été amplifiée par les besoins continus de stockage et de gestion efficace du déluge de données dans les systèmes de stockage. Cette situation motive les chercheurs à proposer des outils (appelés advisors) pour assister les administrateurs et les concepteurs de bases de données dans leurs tâches lors de la sélection de leurs structures d'optimisation pertinentes pour un entrepôt de données et une charge de requêtes donnée. Plusieurs outils issus à la fois du milieu universitaire et de l'industrie ont été proposés pour assister les concepteurs et les DBA dans leurs tâches. On peut citer, par exemple, l'assistant du paramétrage des données pour SQL Server [238], l'assistant de conception pour DB2 [195], l'assistant d'accès SQL pour Oracle [239] et Parinda pour PostgreSQL [196]. Ces outils utilisent des modèles de coûts pour recommander des structures d'optimisation appropriées, mais ils ne sont pas conçus pour traiter à la fois les 3 propriétés de requêtes analytiques (dynamisme de requêtes, leurs volume et l'interaction entre eux). Pour dépasser cette limite, nous avons

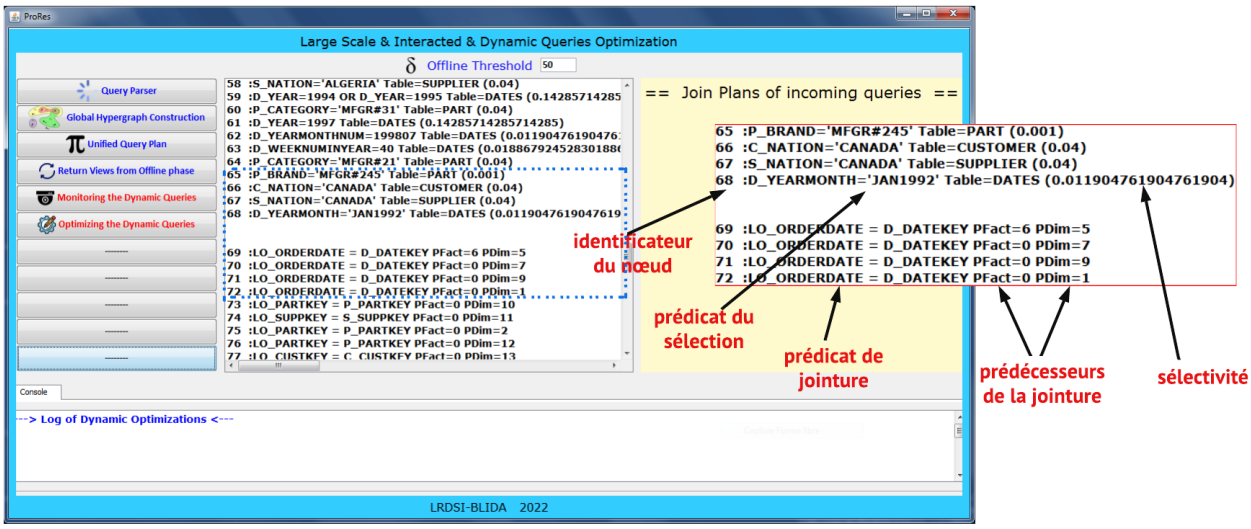


FIGURE 5.10 – Exemple de résultat d'analyseur de requête

développé un outil, appelé ProRes inspiré des assistants commerciaux bien connus permettant d'assister les DBA dans leurs tâches lors de la sélection de vues matérialisées avec un fort avantage dans la gestion des trois propriétés des requêtes analytiques. L'assistant ProRes est développé en Java et intègre toutes les phases et modules de notre approche.

5.3.1 Modules du système

L'assistant ProRes est composé d'un ensemble de modules indépendants qui intègrent tous les algorithmes de nos deux phases (i.e., la phase hors ligne et la phase en ligne). Notre outil permet d'assister les administrateurs de bases de données (DBA) dans leurs tâches de sélection de vues matérialisées permettant d'optimiser les requêtes connues à l'avance et les requêtes ad-hoc en donnant la valeur du seuil δ, qui est fixé par le DBA. Les différents modules de notre outil seront abordés dans les prochaines sections.

5.3.1.1 analyseur de requête

Dans ce module, les utilisateurs peuvent donner une seule requête SQL ou une charge de requêtes SQL à exécuter. L'analyseur prend en entrée un texte écrit dans un langage SQL et le convertit en un arbre d'analyse. La figure 5.10 montre un exemple de l'arbre d'analyse. L'arbre d'analyse d'une requête donnée est réalisé par une analyse syntaxique qui donne les éléments grammaticaux de base de la requête textuelle sous la forme d'un arbre d'analyse et une vérification sémantique de cet arbre en vérifiant les relations, les attributs utilisés, les clauses de la requête, etc.

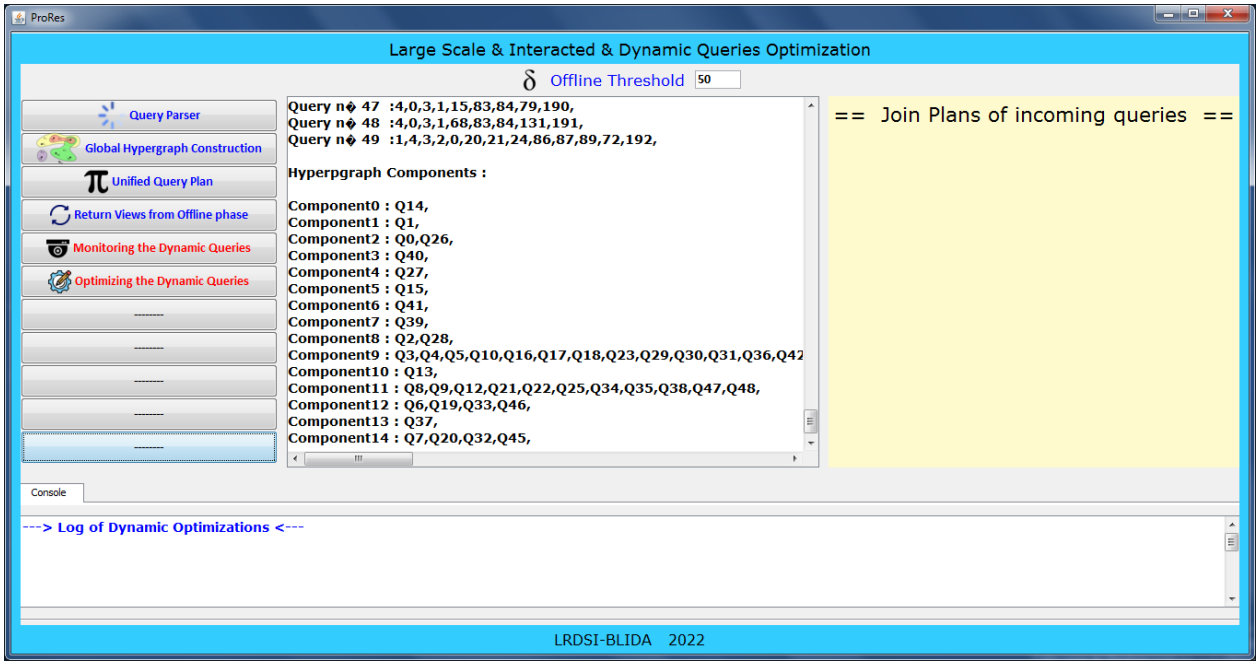


FIGURE 5.11 – Exemple de résultat de génération d'hypergraphe

5.3.1.2 Module d'hypergraphe

La génération de l'hypergraphe global est l'une des étapes les plus importantes de la phase hors ligne de notre approche. Le module d'hypergraphe est chargé de générer un hypergraphe global à partir d'une charge donnée de requêtes en fonction du seuil δ. De plus, il permet de partitionner l'hypergraphe global en plusieurs composants (sous-hypergraphe) au moyen de l'algorithme de partitionnement des hypergraphes hMeTis. La figure 5.11 montre un exemple de résultat de construction d'un hypergraphe global et de son partitionnement pour une charge de 50 requêtes (fixé par le DBA). Après avoir donné la valeur du seuil δ=50, le module d'hypergraphe a permis de générer l'hypergraphe globale des premières 50 requêtes et de le partitionner en 14 composants, où l'interaction entre les requêtes est maximale à l'intérieur de chaque composant.

5.3.1.3 Module de génération d'UQP et de sélection de vues matérialisées

Après avoir générer l'hypergraphe global qui permet de capturer l'interaction des requêtes et après avoir divisé le problème initial en petits sous-problèmes distincts (pouvant être exécutés en parallèle) grâce à notre stratégie de partitionnement d'hypergraphe. La génération du plan de requête unifié (UQP) devient une simple transformation de chaque sous hypergraphe en un graphe orienté et enfin de fusionner les graphes résultants. Ce module vise à connecter le PMQO et le PSV en considérant un nombre importants de requêtes interactive. Ceci est fait par l'intégration des connaissances liées aux vues matérialisées lors de la génération d'UQP. Pour cela, nous avons défini une fonction de bénéfice qui reflète les besoins non fonctionnels liés au PSV (minimisation du coût de traitement des requêtes, du coût de

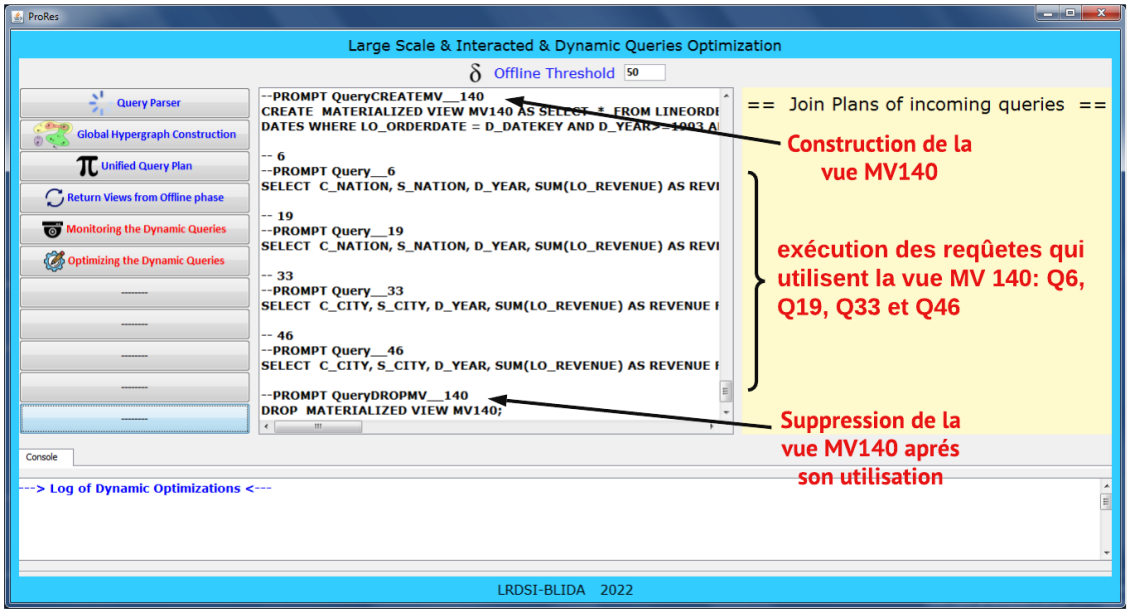


FIGURE 5.12 – Exemple de résultat de génération d'UQP et de sélection des vues

stockage, etc.). Étant donné que nos requêtes sont très nombreuses, il serait impossible de matérialiser toutes les vues candidates sous la contrainte d'espace de stockage. Pour faire face à ce problème, nous avons proposé une solution flexible caractérisée par la présence temporaire des vues matérialisées sur le disque afin d'exploiter toutes les vues candidates. Afin d'éviter la suppression massive des vues matérialisées, nous avons proposé une politique d'ordonnancement des requêtes qui permet de trouver le meilleur ordre de requête qui maximise le bénéfice des vues sélectionnées. La figure 5.12 montre le résultat de cette tâche.

Après le traitement des premières δ requêtes par la phase hors ligne. Les autres requêtes à venir sont gérées par la phase en ligne qui exploite les vues matérialisées sélectionnées par la phase hors ligne. Dans notre proposition, nous ne gardons que le nœud Pivot de chaque composant pour servir les futures requêtes entrantes. Les nœuds pivots représentent les vues ayant le maximum de bénéfice dans chaque composant. La figure 5.13 montre le script SQL des nœuds pivots (déjà matérialisées) qui sont gardés dans le disque après la fin d'exécution de la phase hors ligne.

5.3.1.4 Module de traitement en ligne des requêtes

Dans ce module, nous gérons l'arrivée dynamique des requêtes ad-hoc. Notre assistant ProRes surveille en permanence les requêtes à venir et il est connecté au SGBD Oracle. A l'arrivée d'une requête, son plan individuel est généré par un arbre profond à gauche [223], où tous les prédicats de sélections sont poussées aussi loin que possible dans son graphe de requête (arbre), et son coût d'exécution est estimé théoriquement par notre modèle du coût et réellement par le SGBD Oracle. La figure 5.14 montre ces différentes tâches.

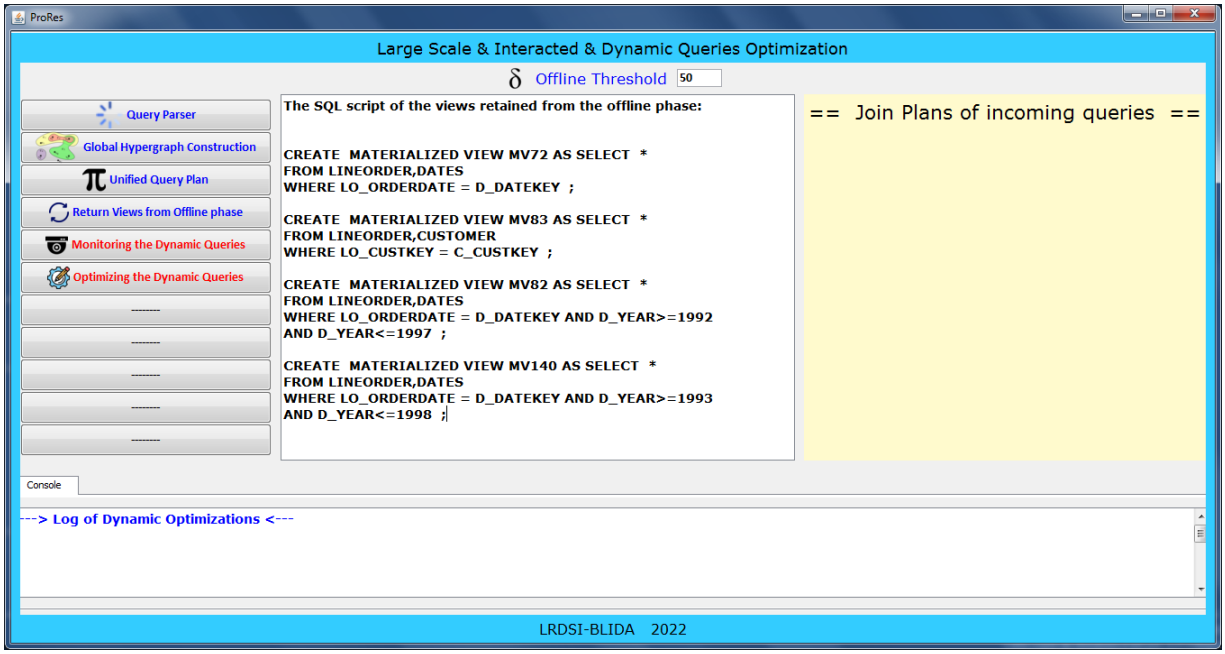


FIGURE 5.13 – Le script SQL des nœuds pivots

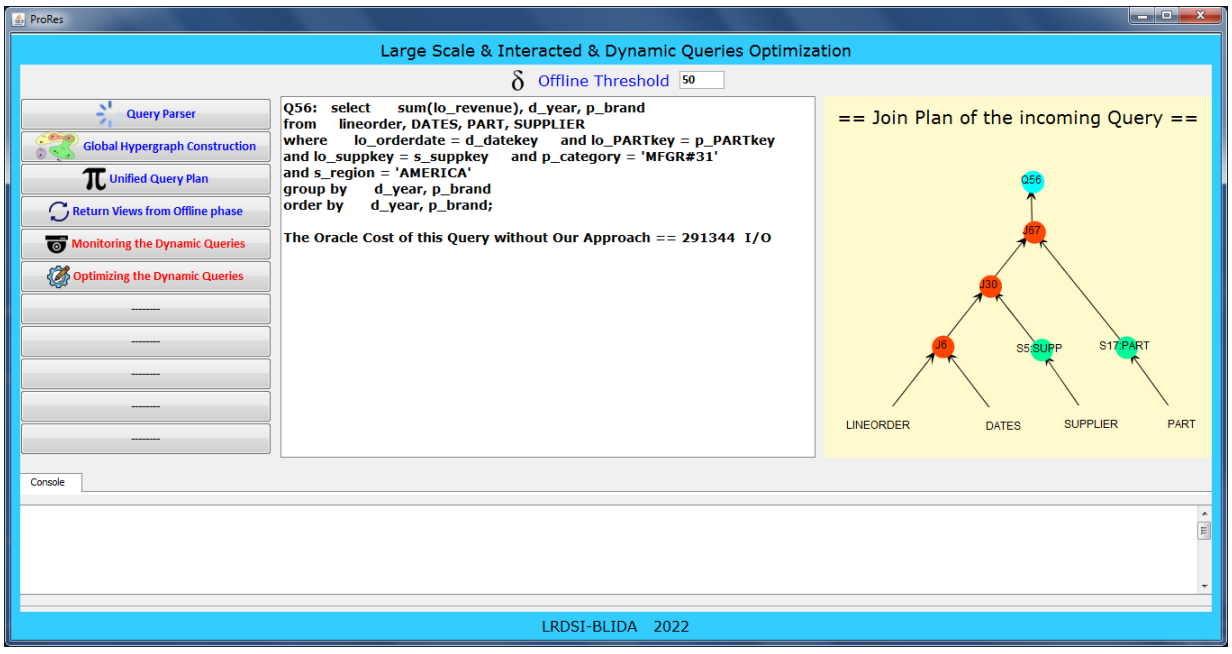


FIGURE 5.14 – Analyse et Génération du plan de la requête entrante

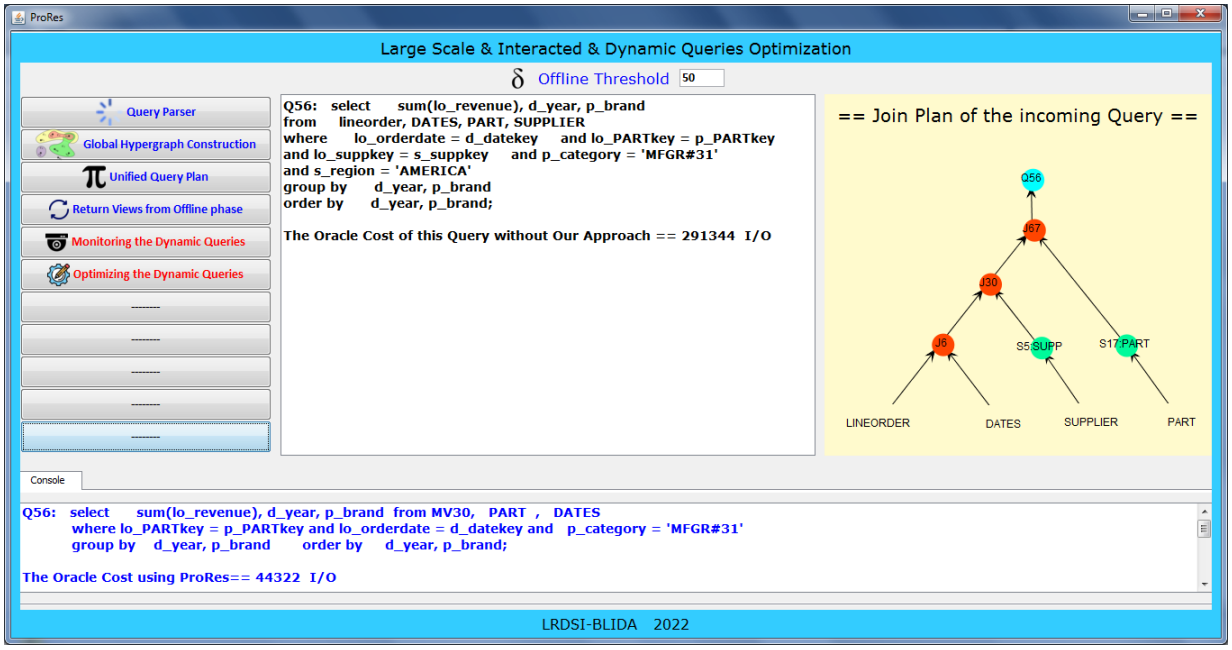


FIGURE 5.15 – Optimisation de la requête entrante

Après avoir analysé la requête entrante et après le placement de cette requête dans le sous hypergraphe approprié , la requête entrante est optimisée par la ré-utilisation des vues matérialisées du pool de ce composant. La figure 5.15 montre un exemple d'optimisation de la requête entrante Q_{56} , où son coût d'exécution réel est optimisé à 85 % de son coût initial.

5.3.2 Implémentation

L'assistant *ProRes* a été développé dans un environnement *Java*. Notre choix du langage de programmation *Java* est dû à ses fonctionnalités admirables. Sun MicroSystems a décrit Java avec la liste des fonctionnalités suivantes [2] : Simple et familier, compilé et interprété, indépendant de la plate-forme, portable, neutre sur le plan architectural, orienté objet, robuste, sécurisé, distribué, multithread et interactif, haute performance, dynamique et extensible. L'assistant ProRes est développé dans les environnements Windows et Linux et il est doté d'une passerelle de connexion au SGBD Oracle 12c. La figure 5.16 montre l'interface de la passerelle de connexion qui permet la connexion à l'entrepôt de données.

5.4 Conclusion

Dans ce chapitre, nous avons mené une étude d'efficacité afin de valider et de comparer notre proposition avec les principales études de l'état de l'art. Nos expérimentations ont été réalisées autour d'un benchmark réel SSB modélisé par un schéma en étoile. Les requêtes utilisées (issues du SSB) couvrent la plupart des types de requêtes OLAP. Notre approche est comparée aux travaux pionniers qui étudiait les problèmes d'optimisation multi requêtes et

2. https://techvidvan.com/tutorials/features-of-java-programming-language/

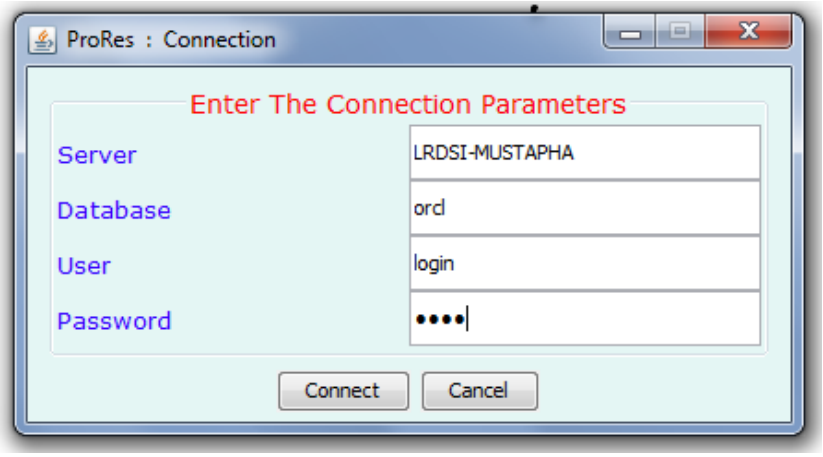


FIGURE 5.16 – La Passerelle de connexion

de sélection des vues matérialisées, où plusieurs scénarios ont été considérés. Les expérimentations sont validés théoriquement à l'aide du notre modèle du coût et réellement dans un SGBD commercial. Les résultats obtenus sont prometteurs et montrent l'efficience et l'efficacité de notre approche.

Notre approche est supportée par un assistant appelé ProRes qui peut être connecté à un SGBD commercial et qui aident les administrateurs de base de données dans leurs tâches lors de la sélection des vues matérialisées. Dans la deuxième partie de ce chapitre, nous avons présenté notre assistant ProRes et ses différents modules. ProRes tire parti de la théorie des hypergraphes pour gérer simultanément les trois propriétés des requêtes analytiques et pour sélectionner les vues matérialisées appropriées. Actuellement, nous intégrons d'autres techniques d'optimisation des requêtes telles que les index et le partitionnement horizontal des données.

Chapitre 6

Conclusion générale et Perspectives

"**La science n'est rien de plus qu'un raffinement de la pensée quotidienne...**"
— —— Albert Einstein

Sommaire

6.1 Conclusion

Aujourd'hui, nous vivons à l'ère du déluge de données, où de grandes quantités de données sont générées en continu par plusieurs fournisseurs de données. De grandes entreprises telles qu'Alibaba, Google, Twitter, Baidu et Huawei travaillent sans relâche pour fournir des solutions efficaces et rapides pour traiter ce volume de données, où la disponibilité des bonnes données au bon moment est devenue un enjeu impératif pour prendre des décisions pertinentes et économes. Ces données sont généralement exploitées par les chercheurs et les décideurs afin d'effectuer des analyses en temps réel via des requêtes analytiques complexes. De nos jours, les requêtes analytiques sont caractérisées par trois propriétés importantes : (i) très nombreuses (ii) dynamiques et (iii) partagent des sous-expressions communes. Ces propriétés peuvent toucher plusieurs problèmes bien étudiés et connus tel que le problème d'optimisation multi-requêtes (PMQO) et de la sélection des vues matérialisées (PSV). Une large panoplie d'études ont été menées pour traiter ces deux problèmes, alors que ces travaux ne parviennent pas à gérer simultanément les trois propriétés des requêtes analytiques. Par conséquent, la question du traitement simultané des trois propriétés des requêtes n'est plus de savoir pourquoi nous le faisons, mais comment nous pouvons y parvenir de manière efficace ?

Dans cette thèse, nous revisitons les deux problèmes corrélés et bien connus dans le monde des bases de données que sont : l'optimisation multi-requêtes et la sélection des vues matérialisées sous un nouvel angle, en considérant les trois propriétés des requêtes analytiques d'une façon simultanée. Nous devinons que l'optimisation de ces requêtes à l'aide des techniques d'optimisation physiques telle que les vues matérialisées nécessite l'utilisation des structures de données évolutives comme les hypergraphes qui ont déjà montré leurs contributions pour les charges statiques de requêtes. Notre méthodologie consiste à utiliser les hypergraphes et de les adapter afin de gérer concurremment les trois propriétés de requêtes. Ensuite , à évaluer son efficacité dans la résolution des deux problèmes d'optimisation multi-requêtes et de sélection des vues matérialisées.

Ce travail couvre un large éventail de sujets importants : entrepôts de données, optimisations multi-requêtes, sélection des vues matérialisées, hypergraphes, modèles de coûts, etc. Pour faciliter la présentation du notre approche et la description de nos résultats, nous avons proposé une démarche qui porte principalement sur les points suivants : (i) présenter un aperçu des concepts et techniques liées à nos problèmes étudiés (ii) élaborer un état de l'art sur les études les plus importantes portant sur le PMQO et le PSV de manière isolée et conjointe, et identifier ses limites dans la gestion simultanée des trois propriétés de requêtes d'analyse. (iii) présenter les notions fondamentales et les facteurs clés qui nous ont motivés à compter sur les hypergraphes pour traiter notre problématique (iv) mettre en évidence les techniques utilisées qui permettent aux hypergraphes de gérer très efficacement nos trois propriétés. (v) proposer et présenter une approche proactive de sélection des vues

matérialisées en se basant sur l'utilisation des hypergraphes. (vi) mener des expérimentations intensives pour évaluer la qualité de nos découvertes et présenter notre outil ProRes qui capitalise nos découvertes et assiste les concepteurs dans leurs tâches de sélection de vues matérialisées.

6.1.1 Étude et synthèse des travaux d'état de l'art

Notre travail de thèse a été guidé par une étude approfondie des travaux existants qui portent sur le traitement et l'optimisation des requêtes, avec une focalisation particulière sur les études traitant du PMQO et du PSV séparément ou conjointement. Cette étude nous a permis d'identifier les lacunes de ces travaux, où nous avons constaté l'absence d'un travail complet et compréhensif étudiant le rôle de la sélection de sous-expressions communes dans la résolution du PSV, tout en considérant simultanément les propriétés récentes des requêtes analytiques. Cette étude nous a également permis de découvrir l'utilisation massive des structures de graphes et leur efficacité dans la gestion des deux problèmes. À la lumière de cette étude, nous avons proposé notre approche qui capitalise l'utilisation des hypergraphes pour traiter notre problème.

6.1.2 Hypergraphes pour gérer les propriétés récentes des requêtes analytiques

La première contribution de cette thèse consiste à découvrir la capacité et l'efficacité des structures de données des hypergraphes dans la gestion simultané des propriétés récentes des requêtes d'analyse, où les requêtes d'aujourd'hui sont caractérisées par trois points principaux : (1) très nombreuses, (2) dynamique (ad-hoc) et (3) partagent des expressions communes. En général, les charges de requêtes OLAP se caractérisent par le partage élevé des opérations communes (expressions communes) entre eux. Notre structure d'hypergraphe nous a permis de capturer facilement l'interaction de ces requêtes grâce à sa flexibilité et sa capacité à représenter des relations complexes et non appariées. Les hypergraphes ont une longue histoire dans la gestion de problèmes à grande échelle grâce à leur efficacité à diviser l'espace de recherche des problèmes difficiles en plusieurs sous-espaces de recherche grâce à leurs outils de partitionnement avancés. Pour faciliter la gestion du nombre élevé de requêtes, nous avons utilisé une heuristique de partitionnement hypergraphique (hMeTiS) qui a assuré la scalabilité de notre approche. En outre, la diversité des analyses de données et l'évolution des directives métier rendent les charges de requêtes OLAP plus dynamique. Pour gérer l'arrivée dynamique de requêtes ad-hoc, un ensemble de primitives est proposé pour augmenter progressivement notre hypergraphe. Notre étude expérimentale a montré l'efficacité des hypergraphes pour traiter les 3 propriétés des requêtes.

6.1.3 Sélection proactive des vues matérialisées

La deuxième contribution de notre thèse concerne la proposition d'une approche proactive de sélection des vues matérialisées qui intègre concurremment nos trois propriétés de requêtes (voir la figure 3.5). De nos jours, La nouvelle Business Intelligence (BI Next Generation) implique des requêtes connues à l'avance et des requêtes ad-hoc/dynamiques. Inspiré par ces aspects et pour traiter les deux types de requêtes, nous avons proposé une approche composée de deux phases : une phase hors ligne pour optimiser un ensemble de requêtes connues à l'avance et une phase en ligne dédiée à l'optimisation des requêtes ad-hoc en fonction d'un seuil donné δ. La phase hors ligne et la phase en ligne partagent l'utilisation de la même structure d'hypergraphe pour capturer l'interaction entre les requêtes et sélectionner les vues matérialisées les plus avantageuses. Notre approche est comparée aux études majeures qui abordent le même problème. Les expérimentations menées montrent l'efficience et l'efficacité de notre approche à traiter conjointement le PMQO et le PSV et à gérer simultanément les propriétés récentes des requêtes analytiques.

6.1.4 Développement d'un outil d'aide à la sélection des vues matérialisées

Sur la base de nos découvertes, nous avons développé un outil, appelé ProRes inspiré des conseillers académiques et commerciaux bien connus. Notre outil permet d'assister les DBA dans leurs tâches lors de la sélection de vues matérialisées pour les charges de requêtes dynamiques à grande échelle. ProRes est développé en Java et intègre toutes les phases et modules de notre approche. La principale particularité de notre assistant est sa capacité à gérer les trois propriétés des requêtes d'analyse.

6.2 Perspectives

De nombreuses perspectives et pistes ouvertes peuvent être envisagées pour améliorer les travaux initiés dans cette thèse. Nous esquissons dans cette section, les perspectives à court et à long terme qui nous paraissent être les plus intéressantes pour l'amélioration de nos découvertes.

6.2.1 Envisager d'autres techniques d'optimisation

Dans ce travail, nous nous sommes focalisés sur le problème de la sélection des vues matérialisées pour optimiser les performances des requêtes OLAP. Notez qu'il existe une large panoplie de structures d'optimisation dédiées à l'optimisation des charges de requêtes analytiques, telles que les index, le partitionnement vertical, le partitionnement horizontale, etc. Notre approche doit être étendue pour intégrer d'autres techniques d'optimisation. Pour ce faire, nous devons rendre notre assistant plus générique pour permettre aux DBA de sélectionner d'autres structures d'optimisation.

6.2.2 Intégration des techniques avancées d'apprentissage automatique

Récemment, les techniques d'apprentissage automatique ont été largement étudiées et exploitées dans le cadre d'optimisation des requêtes. Ces techniques ont attiré une forte attention des chercheurs pour résoudre plusieurs problèmes importants, tels que l'estimation du coût pour les charges de requêtes à grande échelle [240], l'estimation et la prédiction du consommation énergétique [241], la sélection des meilleurs vues matérialisées [204], etc. Très récemment, Plusieurs approches ont été proposées pour résoudre le problème d'ordre de jointure en utilisant les techniques avancées d'apprentissage automatiques [134], où les résultats obtenus sont promoteurs. Dans notre étude, nous avons utilisé l'un des algorithmes populaires, dans lequel l'ordre de jointure est effectué en fonction de la taille des différentes tables de dimension impliquées dans cette requête de manière ascendante. Par conséquent, il semble très intéressant d'exploiter certaines techniques d'apprentissage automatique pour la sélection d'ordre de jointure des requêtes, ce qui peut améliorer leurs plans d'exécution.

6.2.3 Reproduire notre approche pour traiter les requêtes SPARQL

Dans ce travail, nous concentrons nos efforts sur la connexion et la résolution du PMQO et du PSV pour l'optimisation des charges de requêtes analytiques. Ces deux problèmes ont été abordés dans tous les langages de requête associés à toutes les générations de bases de données, et notamment dans les bases de données sémantiques, où un large éventail de recherches a été effectué sur le PMQO et le PSV. Dans [205, 242], Le PMQO et le PSV ont été liés pour optimiser la performance des requêtes SPARQL, en considérant des charges de requêtes de taille moyenne. Nous pensons que l'exploitation de nos découvertes dans l'utilisation des hypergraphe pour gérer les charges de requêtes OLAP à grande échelle est une grande opportunité de gérer les charges de requêtes SPARQL dans les bases de données sémantique.

6.2.4 Envisager des plates-formes de déploiement avancées

Le choix de la bonne plate-forme de déploiement est l'un des facteurs les plus importants ayant un impact direct sur l'optimisation logique et physique des requêtes. Au cours de ce travail, nous avons étudié les problèmes d'optimisation multi requêtes et de sélection des vues matérialisées, en considérant des charges de requêtes OLAP exécutées dans un entrepôt de données **centralisé**. Notre problème a été abordé sans considérer les plates-formes de déploiement avancées comme les architectures parallèles et distribuées. Par conséquent, il semble important de considérer d'autres architectures de déploiement afin d'améliorer l'efficacité et la scalabilité de nos algorithmes.

Annexe A

Modèle de coût

Nous utilisons dans ce travail de thèse, un modèle de coût intuitif pour l'estimation des coûts d'exécution (en termes de nombre d'entrées sorties (E/S)) des différentes opérations algébriques et du coût de construction des vues matérialisées. Ce modèle s'inspire des modèles de coûts implémentés par les SGBD commerciaux [243]. Notre modèle de coût est basé sur les paramètres définis dans le tableau A.1. R_i est une relation qui peut être une table de faits ou de dimensions, tandis que A et B sont des attributs.

Si nous revenons à l'arbre de requête illustré dans 4.3.4 , nous observons que chaque nœud intermédiaire est étiqueté avec un opérateur algébrique, alors que les nœuds feuilles correspondent à des tables de base. Pour chaque nœud intermédiaire ni, nous pouvons attribuer deux valeurs arithmétiques : (a) Le coût de traitement du nœud ni noté $Process\_Cost(ni)$ et (b) La taille du résultat intermédiaire ni résultant du traitement de l'opération algébrique qui correspond à ni, notée $Size(ni)$.

A.0.1 Taille des résultats intermédiaires

1. Taille de la projection : la taille du résultat d'une projection sur la relation R_i est, en principe, la taille de R_i elle-même :

$\|\|R_i\|\|$	nombre de tuples de R_i
$\|R_i\|$	nombre de pages dans lesquelles nous stockons R_i
Valeurs_distinctes (A)	nombre de valeurs distinctes de l'attribut A
Valeur_Max(A)	Valeur maximale de l'attribut A
Valeur_Min(A)	Valeur minimale de l'attribut A
Sélectivité($A = value$)	$\frac{1}{Valeurs\_distinctes(A)}$
Sélectivité($A = B$)	$\frac{1}{max(Valeurs\_distinctes(A), Valeurs\_distinctes(B))}$

TABLE A.1 – Principaux paramètres de notre modèle de coût

$$Size(\prod_{A}(R)) = ||R|| \tag{A.1}$$

2. Sélections simples : Si le résultat intermédiaire n_i représente une sélection simple (ayant la forme : attribut op valeur) , où attribut ∈ à la relation R, op ∈ {=, <, >, $like$, ...} et valeur ∈ Domaine(attribut) , sa taille est estimée en utilisant la formule suivante :

$$Size(attribut\ op\ valeur) = Sélectivité(attribut\ op\ valeur) \times ||R|| \tag{A.2}$$

3. Sélections complexes : Si le résultat intermédiaire n_i est une sélection complexe de la forme $clause_R = (attr_1\ op\ val_1$ et $attr_2\ op\ val_2$ et ... et $attr_n\ op\ val_m)$. La taille du résultat intermédiaire est estimée comme suit :

$$Size(clause_R) = Sélectivité(clause_R) \times ||R|| \tag{A.3}$$

où :
- $Sélectivité(clause_R) = Sélectivité(attr_1$ op $val_1) \times Sélectivité(attr_2$ op $val_2) \times$ ... $\times\ Sélectivité(attr_n$ op $val_m)$

4. Taille des jointures : Si nous revenons à l'exemple 4.3.4 qui illustre un arbre de requête composé de quatre opérations de jointure ayant l'ordre : $J1$, $J2$, $J3$ et $J4$. Nous estimons la taille des quatre jointures par les formules suivantes :

$$Size(J_1) = ||Lo|| \times ||Su|| \times Sélectivité(Lo.key_1 = Su.Key_1) \tag{A.4}$$

$$Size(J_2) = ||J_1|| \times ||Cu|| \times Sélectivité(Lo.key_2 = Cu.Key_2) \tag{A.5}$$

$$Size(J_3) = ||J_2|| \times ||Da|| \times Sélectivité(Lo.key_3 = Da.Key_3) \tag{A.6}$$

$$Size(J_4) = ||J_3|| \times ||Pa|| \times Sélectivité(Lo.key_4 = Pa.Key_4) \tag{A.7}$$

En général ,

$$Size(J_i) = ||J_{i-1}|| \times ||D_i|| \times Sélectivité(F.key_i = D_i.key_i) \tag{A.8}$$

Où , F est la table de faits et D est une table de dimension

A.0.2 Coûts de traitement des opérateurs algébriques

1. Coût de traitement de la projection $\prod_A$(R)

$$Cost_{\prod_A} = |R| \tag{A.9}$$

Où : - |R| : est la taille en termes de pages de la relation R

2. Coût de la sélection $\sigma_{prédicat}(R)$

   (a) Prédicats simples : les prédicats simples de sélection ont la forme suivante : attribut op valeur , où attribut $\in R$, op $\in \{=, <, >, like, ...\}$ et valeur $\in$ Domaine(attribut)

$$Cost\,(\sigma_{predicate}(R)) = |R| \tag{A.10}$$

   (b) Prédicats complexes : les prédicats complexes représentent une conjonction de prédicats simples de sélection désignés par $clause_R$, où $clause_R = (attr_1\ op\ val_1$ et $attr_2\ op\ val_2$ et ... et $attr_n\ op\ val_m)$

$$Cost\,(clause_R) = |R| \tag{A.11}$$

3. Coût de la jointure $R \bowtie S$

   Dans notre étude, nous utilisons l'algorithme de jointure par hachage, qui est l'implémentation la plus efficace lorsque l'une des deux tables est petite, ce qui est cohérent avec les requêtes de jointure en étoile dans les entrepôts de données. Nous supposons ici que la table R est la plus petite.

$$Cost\,(R \bowtie S) = |R| + |S| + 2 \times (|R| + |S|) \times (1 - q) \tag{A.12}$$

   où, q est la fraction de |R| dont la table de hachage tient en mémoire. On distingue ici les deux cas suivants :

   (a) Le cas où q = 1 , la table de hachage tient en mémoire : la jointure est une simple jointure de hachage. Une jointure de hachage est plus rentable lorsque la table de hachage de la plus petite table tient en mémoire. Dans ce cas, le coût est limité à une seule lecture sur les deux relations impliquées dans la jointure.

$$Cost\,(R \bowtie S) = |R| + |S| \tag{A.13}$$

   (b) Le cas où toutes les tables de hachage ne tiennent pas en mémoire

$$Cost\,(R \bowtie S) = 3|R| + 3|S| \tag{A.14}$$

Annexe B

La charge des sept requêtes

Nous avons mentionné dans ce qui précède que nous avons utilisé un générateur aléatoire (appelé *qgen*), pour générer aléatoirement des charges de requêtes OLAP, en utilisant une template composée par 13 requêtes. Pour plus de détails sur cette template de requête, nous recommandons aux lecteurs de se référer à la thèse de [236] qui donne une bonne description de cette template de requêtes. Nous présentons dans cette annexe, la charge des sept requêtes permettant de générer l'hypergraphe illustré dans l'exemple 4.3.5.

```
Requête Q1:
select  c_nation, s_nation, d_year, sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_region = 'AFRICA'
    and s_region = 'AFRICA'
    and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;

Requête Q2:
select  c_city, s_city, d_year, sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where    lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_nation = 'VIETNAM'
    and s_nation = 'VIETNAM'
    and d_year >= 1993 and d_year <= 1998
```

```
group by  c_city, s_city, d_year
order by d_year asc, revenue desc;

Requête Q3:
select c_city, s_city, d_year, sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and (c_city='IRAQ     4' or c_city='JORDAN   6')
    and (s_city='IRAQ     4' or s_city='JORDAN   6')
    and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc;

Requête Q4
select   c_city, d_year, sum(lo_revenue) as revenue
from   CUSTOMER, lineorder, DATES
where  lo_custkey = c_custkey
    and lo_orderdate = d_datekey
    and c_nation = 'GERMANY'
    and d_year >= 1993 and d_year <= 1998
group by c_city, d_year
order by d_year asc, revenue desc;

Requête Q5
select s_city, p_brand, sum(lo_revenue - lo_supplycost) as profit
from      SUPPLIER,lineorder, PART
where lo_suppkey = s_suppkey
    and lo_PARTkey = p_PARTkey
    and s_nation = 'VIETNAM'
    and p_category = 'MFGR#34'
group by  s_city, p_brand
order by  s_city, p_brand;

Requête Q6:
select c_nation, s_nation, d_year,sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, SUPPLIER, DATES
where lo_custkey = c_custkey
```

```
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_region = 'AFRICA'
    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;

Requête Q7:
select c_city, d_year, sum(lo_revenue) as revenue
from     CUSTOMER, lineorder, DATES
where lo_custkey = c_custkey-
    and lo_orderdate = d_datekey
    and c_nation = 'BRAZIL'
    and d_year >= 1992 and d_year <= 1997
group by  c_city, d_year
order by  d_year asc, revenue desc;
```

Bibliographie

[1] Pat O'Neil, Eizabeth O'Neil, and Xuedong Chen. Star schema benchmark-revision 3. *edn*, 2009.

[2] Nenad Jukic, Abhishek Sharma, Svetlozar Nestorov, and Boris Jukic. Augmenting data warehouses with big data. *Information Systems Management,*, 2015.

[3] Nabila Berkani, Ladjel Bellatreche, Selma Khouri, and Carlos Ordonez. The contribution of linked open data to augment a traditional data warehouse. *Journal of Intelligent Information Systems*, 55(3) :397–421, 2020.

[4] John Meehan, Cansu Aslantas, Stan Zdonik, Nesime Tatbul, and Jiang Du. Data ingestion for the connected world. In *CIDR*, 2017.

[5] Jinjun Chen and Honggang Wang. Guest editorial : Big data infrastructure i. *IEEE Transactions on Big Data*, 4(2) :148–149, 2018.

[6] Steffen Zeuch, Bonaventura Del Monte, Jeyhun Karimov, Clemens Lutz, Manuel Renz, Jonas Traub, Sebastian Breß, Tilmann Rabl, and Volker Markl. Analyzing efficient stream processing on modern hardware. *Proceedings of the VLDB Endowment*, 12(5) :516–530, 2019.

[7] Mohamed Kechar and Ladjel Bellatreche. Safeness : Suffix arrays driven materialized view selection framework for large-scale workloads. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 74–86. Springer, 2022.

[8] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. Adaptive learning of aggregate analytics under dynamic workloads. *Future Gener. Comput. Syst.*, 109 :317–330, 2020.

[9] Rafi Ahmed, Randall Bello, Andrew Witkowski, and Praveen Kumar. Automated generation of materialized views in oracle. *Proceedings of the VLDB Endowment*, 13(12) :3046–3058, 2020.

[10] Rogério Luís de C Costa, José Moreira, Paulo Pintor, Veronica dos Santos, and Sérgio Lifschitz. A survey on data-driven performance tuning for big data analytics platforms. *Big Data Research*, 25 :100206, 2021.

[11] Verónika Peralta, Willeme Verdeaux, Yann Raimont, and Patrick Marcel. Qualitative analysis of the sqlshare workload for session segmentation. In *DOLAP*, 2019.

[12] Katrin Weller and Katharina E. Kinder-Kurlanda :. A manifesto for data sharing in social media research. *WebSci2016*, pages 166–172, 2016.

[13] Michail Flouris, Renaud Lachaize, and Angelos Bilas. Orchestra : Extensible block-level support for resource and data sharing in networked storage systems. *ICPADS2008*, pages 237–244, 2008.

[14] Sebastian Breß, Bastian Köcher, Henning Funke, Steffen Zeuch, Tilmann Rabl, and Volker Markl. Generating custom code for efficient query execution on heterogeneous processors. *VLDB*, pages 797–822, 2018.

[15] Subrata Ghosh : Timos K. Sellis. On the multiple query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, pages 262–266, 1990.

[16] Tarun Kathuria and S Sudarshan. Efficient and provable multi-query optimization. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 53–67, 2017.

[17] Per-Åke Larson and H. Z. Yang. Computing queries from derived relations. In *The International Conference on Very Large Data Bases (VLDB)*, pages 259–269, 1985.

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

[19] Ladjel Bellatreche, Kamalakar Karlapalem, and Michel Schneider. On efficient storage space distribution among materialized views and indices in data warehousing environments. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 397–404, 2000.

[20] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize under a maintenance cost constraint. In *The International Conference on Database Theory (ICDT)*, pages 453–470, 1999.

[21] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. Automatic view generation with deep learning and reinforcement learning. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1501–1512. IEEE, 2020.

[22] Ahcène Boukorca, Ladjel Bellatreche, Sid-Ahmed Benali Senouci, and Zoé Faget. Coupling materialized view selection to multi query optimization : Hyper graph approach. *IJDWM*, 11(2) :62–84, 2015.

[23] Mustapha Chaba Mouna, Ladjel Bellatreche, and Narhimene Boustia. Prores : Proactive re-selection of materialized views. *Computer Science and Information Systems*, (00) :3–3, 2022.

[24] Mustapha Chaba Mouna, Ladjel Bellatreche, and Narhimene Boustia. Selecting subexpressions to materialize for dynamic large-scale workloads. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 39–51. Springer, 2021.

[25] Mustapha Chaba Mouna, Ladjel Bellatreche, and Narhimene Boustia. Hyraq : optimizing large-scale analytical queries through dynamic hypergraphs. In *Proceedings of the 24th Symposium on International Database Engineering & Applications*, pages 1–10, 2020.

[26] Mustapha Chaba Mouna, Ladjel Bellatreche, and Narhimane Boustia. Optimisation conjointe et dynamiques des requêtes régulières et recommandées. *Business Intelligence & Big Data*, 2018.

[27] William H Inmon. *Building the Data Warehouse*. John Wiley Sons, New York,United States, 1992.

[28] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining : Concepts and Techniques*. Morgan Kaufmann, 2011.

[29] Bin Jiang. *Constructing Generic Data Warehouses with Metadata-driven Generic Operators*. CreateSpace Independent Publishing Platform ; 2nd edition, 2015.

[30] Ralph Kimball and Margy Ross. *The data warehouse toolkit : the complete guide to dimensional modeling, 2nd Edition*. Wiley, 2002.

[31] Muhammad Bilal Shahid, Umber Sheikh, Basit Raza, Munam Ali Shah, Ahmad Kamran, and Adeel Anjum. Application of data warehouse in real life : State-ofthe-art survey from user preferences' perspective. *International Journal of Advanced Computer Science and Applications*, 7(4), 2016.

[32] J P A Runtuwene, I R H T Tangkawarow, C T M Manoppo, and R J Salak. A comparative analysis of extract, transformation and loading (etl) process. In *IOP Conference Series : Materials Science and Engineering*, 2018.

[33] Sreemathy J, Naveen Durai K, Lakshmi Priya E, Deebika R, Suganthi K, and Aisshwarya PT. Data integration and etl : A theoretical perspective. In *2021 7th International Conference on Advanced Computing Communication Systems (ICACCS)*, 2021.

[34] Mohammad Rifaie, Keivan Kianmehr, Reda Alhajj, and Mick J. Ridley. Data warehouse architecture and design. In *IRI*, pages 58–63. IEEE Systems, Man, and Cybernetics Society, 2008.

[35] Azwa Abdul aziz, Julaily Aida Jusoh, Hasni hassan, and et al. a framework for educational data warehouse (edw) architecture using business intelligence (bi) technologies. *Journal of Theoretical and Applied Information Technology*, 69(1), 2014.

[36] G. Blazic, Patrizia Poscic, and Danijela Jaksic. Data warehouse architecture classification. In *MIPRO*, pages 1491–1495. IEEE, 2017.

[37] M. Golfarelli and S. Rizzi, editors. *Data Warehouse Design : Modern Principles and Methodologies*. McGraw Hill, 2009.

[38] Qishan Yang, Mouzhi Ge, and Markus Helfert. Analysis of data warehouse architectures : Modeling and classification. In *ICEIS (2)*, pages 604–611. SciTePress, 2019.

[39] Barry Devlin. *Data warehouse : from architecture to implementation*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[40] Thilini Ariyachandra and Hugh J. Watson. Key organizational factors in data warehouse architecture selection. *Decis. Support Syst.*, 49(2) :200–212, 2010.

[41] Stefano Rizzi. Data warehouse. In *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc., 2008.

[42] Thilini Ariyachandra and Hugh J. Watson. Which data warehouse architecture is most successful? *Business Intelligence Journal.*, 11(1) :4–6, 2006.

[43] Mary Breslin. Data warehousing battle of the giants. *Business intelligence journal*, 7 :6–20, 2004.

[44] Pubudika Kumari Mawilmada. *Impact of a data warehouse model for improved decision-making process in healthcare*. PhD thesis, Queensland University of Technology, 2011.

[45] Kamal Matouk, Mieczysław L Owoc, et al. A survey of data warehouse architectures—preliminary results. In *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1121–1126. IEEE, 2012.

[46] Selma Khouri and Ladjel Bellatreche. Design life-cycle-driven approach for data warehouse systems configurability. *Journal on Data Semantics*, 6(2) :83–111, 2017.

[47] Matteo Golfarelli. From user requirements to conceptual design in warehouse design : a survey. In *Data warehousing design and advanced engineering applications : methods for complex construction*, pages 1–16. IGI Global, 2010.

[48] Selma Khouri and Ladjel Bellatreche. Towards a configurable database design : a case of semantic data warehouses. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 760–767. Springer, 2014.

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

[50] Ralph Kimball. *The data warehouse toolkit : practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., 1996.

[51] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model : A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(02n03) :215–247, 1998.

[52] Oksana Grabova, Jerome Darmont, Jean-Hugues Chauchat, and Iryna Zolotaryova. Business intelligence for small and middle-sized entreprises. *ACM SIGMOD Record*, 39(2) :39–50, 2010.

[53] Thomas Stöhr, Holger Märtens, and Erhard Rahm. Multi-dimensional database allocation for parallel data warehouses. 2000.

[54] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing : new wine or just new bottles ? *Proceedings of the VLDB Endowment*, 3(1-2) :1647–1648, 2010.

[55] Pascal Wehrle, Maryvonne Miquel, and Anne Tchounikine. A model for distributing and querying a data warehouse on a computing grid. In *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, volume 1, pages 203–209. IEEE, 2005.

[56] Nur Hani Zulkifli Abai, Jamaiah H Yahaya, and Aziz Deraman. User requirement analysis in data warehouse design : a review. *Procedia Technology*, 11 :801–806, 2013.

[57] Sandro Bimonte, Leandro Antonelli, and Stefano Rizzi. Requirements-driven data warehouse design based on enhanced pivot tables. *Requirements Engineering*, 26(1) :43–65, 2021.

[58] Prabhu Shankar, Beshoy Morkos, Darshan Yadav, and Joshua D Summers. Towards the formalization of non-functional requirements in conceptual design. *Research in Engineering Design*, 31(4) :449–469, 2020.

[59] Martin Glinz. On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)*, pages 21–26. IEEE, 2007.

[60] Janet Burge and David C Brown. Nfrs : Fact or fiction. *Computer Science Technical Report, Worcester Polytechnic University, WPI-CS-TR-02-01*, 2002.

[61] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Nonfunctional requirements : From elicitation to conceptual models. *IEEE transactions on Software engineering*, 30(5) :328–350, 2004.

[62] Naveen Prakash and Anjana Gosain. An approach to engineering the requirements of data warehouses. *Requirements Engineering*, 13(1) :49–72, 2008.

[63] Manfred Broy. Rethinking nonfunctional software requirements. *Computer*, 48(05) :96–99, 2015.

[64] Matteo Golfarelli. Dfm as a conceptual model for data warehouse. In *Encyclopedia of Data Warehousing and Mining, Second Edition*, pages 638–645. IGI Global, 2009.

[65] Aris Tsois, Nikos Karayannidis, and Timos K Sellis. Mac : Conceptual data modeling for olap. In *DMDW*, volume 39, page 5, 2001.

[66] Carsten Sapia, Markus Blaschka, Gabriele Höfling, and Barbara Dinter. Extending the e/r model for the multidimensional paradigm. In *International Conference on Conceptual Modeling*, pages 105–116. Springer, 1998.

[67] Enrico Franconi and Anand Kamblet. A data warehouse conceptual data model. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 435–436. IEEE, 2004.

[68] Alberto Abelló, José Samos, and Fèlix Saltor. Yam2 : a multidimensional conceptual model extending uml. *Information Systems*, 31(6) :541–567, 2006.

[69] Sergio Luján-Mora, Juan Trujillo, and Il-Yeol Song. A uml profile for multidimensional modeling in data warehouses. *Data & knowledge engineering*, 59(3) :725–769, 2006.

[70] Sandro Bimonte, Omar Boussaid, Michel Schneider, and Fabien Ruelle. Design and implementation of active stream data warehouses. In *Research Anthology on Decision Support Systems and Decision Management in Healthcare, Business, and Engineering*, pages 288–311. IGI Global, 2021.

[71] Fatimaez-Zahra Dahaoui, Lamiae Demraoui, Mohammed Reda Chbihi Louhdi, and Hicham Behja. Toward data warehouse modeling in the context of big data. In *Advances on Smart and Soft Computing*, pages 235–245. Springer, 2021.

[72] Chris Adamson and Mike Venerable. *Data warehouse design solutions*. John Wiley & Sons, Inc., 1998.

[73] Verónika Peralta, Alvaro Illarze, and Raúl Ruggia. On the applicability of rules to automate data warehouse logical design. In *CAiSE Workshops*, 2003.

[74] Matteo Golfarelli and Stefano Rizzi. Designing the data warehouse : Key steps and crucial issues. *Journal of computer science and Information Management*, 2(3) :88–100, 1999.

[75] DL Moody and MAR Kortink. From enterprise models to dimensional models. *Department of Information Systems, University of Melbourne, Parkville, Australia*, 2000.

[76] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*, volume 2. Springer, 1999.

[77] Fahd Sabry Esmail. A survey of real-time data warehouse and etl. *Management*, 9(3) :3–9, 2014.

[78] Sweta Suman, Pallavi Khajuria, and Siddhaling Urolagin. Star schema-based data warehouse model for education system using mondrian and pentaho. In *International conference on Modelling, Simulation and Intelligent Computing*, pages 30–39. Springer, 2020.

[79] Paulraj Ponniah. *Data warehousing fundamentals : a comprehensive guide for IT professionals*. John Wiley & Sons, 2004.

[80] Nilufar Easmin, KM Azharul Hasan, and Nilufa Parvin. An efficient implementation scheme for multidimensional online analytical processing. In *2008 11th International Conference on Computer and Information Technology*, pages 536–541. IEEE, 2008.

[81] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. The star schema benchmark and augmented fact table indexing. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 237–252. Springer, 2009.

[82] Lyazid Toumi and Ahmet Ugur. Static and incremental dynamic approaches for multi-objective bitmap join indexes selection in data warehouses. *The Journal of Supercomputing*, 77(4) :3933–3958, 2021.

[83] Jay Prakash and TV Vijay Kumar. Multi-objective materialized view selection using moga. *International Journal of System Assurance Engineering and Management*, pages 1–12, 2020.

[84] Partha Ghosh, Takaaki Goto, Jyotsna Kumar Mandal, and Soumya Sen. Materialized view driven architecture over lattice of cuboids in data warehouse. In *Proceedings of the The 8th International Virtual Conference on Applied Computing & Information Technology*, pages 27–31, 2021.

[85] Elzbieta Malinowski and Esteban Zimányi. A conceptual model for temporal data warehouses and its transformation to the er and the object-relational models. *Data & Knowledge Engineering*, 64(1) :101–133, 2008.

[86] Shaker H Ali El-Sappagh, Abdeltawab M Ahmed Hendawi, and Ali Hamed El Bastawissy. A proposed model for data warehouse etl processes. *Journal of King Saud University-Computer and Information Sciences*, 23(2) :91–104, 2011.

[87] Vishal Gour, SS Sarangdevot, Govind Singh Tanwar, and Anand Sharma. Improve performance of extract, transform and load (etl) in data warehouse. *International Journal on Computer Science and Engineering*, 2(3) :786–789, 2010.

[88] Sonal Sharma and Rajni Jain. Modeling etl process for data warehouse : an exploratory study. In *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, pages 271–276. IEEE, 2014.

[89] Ladjel Bellatreche. *Utilisation des vues materialisees, des index et de la fragmentation dans la conception logique et physique d'un entrepot de donnees*. PhD thesis, Clermont-Ferrand 2, 2000.

[90] Shamkant B Navathe and Mingyoung Ra. Vertical partitioning for database design : a graphical algorithm. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 440–450, 1989.

[91] Soumia Benkrid. *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données : application aux plateformes parallèles*. PhD thesis, Chasseneuil-du-Poitou, Ecole nationale supérieure de mécanique et d . . ., 2014.

[92] Ahcene Boukorca, Ladjel Bellatreche, and Soumia Benkrid. Hypad : hyper-graph-driven approach for parallel data warehouse design. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 770–783. Springer, 2015.

[93] Michael G Kahn, Joyce Y Mui, Michael J Ames, Anoop K Yamsani, Nikita Pozdeyev, Nicholas Rafaels, and Ian M Brooks. Migrating a research data warehouse to a public cloud : challenges and opportunities. *Journal of the American Medical Informatics Association*, 2021.

[94] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, pages 215–226, 2016.

[95] Nabila Berkani, Ladjel Bellatreche, Selma Khouri, and Carlos Ordonez. Value-driven approach for designing extended data warehouses. 2019.

[96] Lucas M Fleuren, Tariq A Dam, Michele Tonutti, Daan P de Bruin, Robbert CA Lalisang, Diederik Gommers, Olaf L Cremer, Rob J Bosman, Sander Rigter, Evert-Jan Wils, et al. The dutch data warehouse, a multicenter and full-admission electronic health records database for critically ill covid-19 patients. *Critical Care*, 25(1) :1–12, 2021.

[97] Ralph Kimball. The evolving role of the enterprise data warehouse in the era of big data analytics. *Kimball Gr*, 2011.

[98] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : simplified data processing on large clusters. *Communications of the ACM*, 51(1) :107–113, 2008.

[99] Mohammadhossein Barkhordari and Mahdi Niamanesh. Chabok : a map-reduce based method to solve data warehouse problems. *Journal of Big Data*, 5(1) :1–25, 2018.

[100] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive-a petabyte scale data warehouse using hadoop. In *2010 IEEE 26th international conference on data engineering (ICDE 2010)*, pages 996–1005. IEEE, 2010.

[101] James G Shanahan and Laing Dai. Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2323–2324, 2015.

[102] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive : a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2) :1626–1629, 2009.

[103] Luca Leonardi, Salvatore Orlando, Alessandra Raffaetà, Alessandro Roncato, Claudio Silvestri, Gennady Andrienko, and Natalia Andrienko. A general framework for trajectory data warehousing and visual olap. *GeoInformatica*, 18(2) :273–312, 2014.

[104] Jie Song, Chaopeng Guo, Zhi Wang, Yichan Zhang, Ge Yu, and Jean-Marc Pierson. Haolap : A hadoop based olap system for big data. *Journal of Systems and Software*, 102 :167–181, 2015.

[105] M.Bala, O.Boussaid, and Z.Alimazighi. A fine-grained distribution approach for etl processes in big data environments. *Data Knowl. Eng.*, pages 114–136, 2017.

[106] Hongjun Lu, Beng-Chin Ooi, and Cheng-Hian Goh. Multidatabase query optimization : Issues and solutions. In *Proceedings RIDE-IMS93 : Third International Workshop on Research Issues in Data Engineering : Interoperability in Multidatabase Systems*, pages 137–143. IEEE, 1993.

[107] Carlyna Bondiombouy. *Query Processing in Multistore Systems.* PhD thesis, Université Montpellier, 2017.

[108] Hani Ramadhan, Fitri Indra Indikawati, Joonho Kwon, and Bonyong Koo. Musq : A multi-store query system for iot data using a datalog-like language. *IEEE Access*, 8 :58032–58056, 2020.

[109] Viktor Rosenfeld, Sebastian Breß, and Volker Markl. Query processing on heterogeneous cpu/gpu systems. *ACM Computing Surveys (CSUR)*, 55(1) :1–38, 2022.

[110] Abraham Silberschatz, Henry F Korth, Shashank Sudarshan, et al. *Database system concepts*, volume 4. McGraw-Hill New York, 1997.

[111] Hector Garcia-Molina. *Database systems : the complete book*. Pearson Education India, 2008.

[112] Ramez Elmasri. Fundamentals of database systems seventh edition. 2021.

[113] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. Access path selection in a relational database management system. In *Readings in Artificial Intelligence and Databases*, pages 511–522. Elsevier, 1989.

[114] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys (CSUR)*, 25(2) :73–169, 1993.

[115] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[116] Sven Groppe. *Data management and query processing in semantic web databases*. Springer Science & Business Media, 2011.

[117] Guy M Lohman. The db2 universal database optimizer. *IBM Research Division, IBM Almaden Research Center*, 2007.

[118] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1) :3–27, 2013.

[119] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment*, 2(1) :982–993, 2009.

[120] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1) :65–74, 1997.

[121] Robert L Cannon, Jitendra V Dave, and James C Bezdek. Efficient implementation of the fuzzy c-means clustering algorithms. *IEEE transactions on pattern analysis and machine intelligence*, (2) :248–255, 1986.

[122] Nicolas Bruno and Surajit Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 263–274, 2002.

[123] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy Lohman. Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 558–569, 2002.

[124] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid : Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15(3) :200–222, 2001.

[125] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. *ACM Sigmod Record*, 14(2) :256–276, 1984.

[126] David DeHaan and Frank Wm. Tompa. Optimal top-down join enumeration. In *SIGMOD Conference*, pages 785–796. ACM, 2007.

[127] Sophie Cluet and Guido Moerkotte. On the complexity of generating optimal left-deep processing trees with cross products. In *International Conference on Database Theory*, pages 54–67. Springer, 1995.

[128] Ming-Syan Chen, Ming-Ling Lo, Philip S. Yu, and Honesty C Young. Using segmented right-deep trees for the execution of pipelined hash joins. In *VLDB*, pages 15–26, 1992.

[129] Mikal Ziane, Mohamed Zaït, and Pascale Borla-Salamet. Parallel query processing with zigzag trees. *The VLDB Journal*, 2(3) :277–301, 1993.

[130] Guido Moerkotte and Thomas Neumann. Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In *Proceedings of the 32nd international conference on Very large data bases*, pages 930–941. Citeseer, 2006.

[131] Toshihide Ibaraki and Tiko Kameda. On the optimal nesting order for computing n-relational joins. *ACM Transactions on Database Systems (TODS)*, 9(3) :482–502, 1984.

[132] S Vellev. Review of algorithms for the join ordering problems in database query optimization. *Information Technologies and control*, 1 :32–40, 2009.

[133] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–4, 2018.

[134] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. Reinforcement learning with tree-lstm for join order selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1297–1308. IEEE, 2020.

[135] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, 4th Edition.* Springer, 2020.

[136] Philip S. Yu, M-S Chen, Joel L. Wolf, and John Turek. Parallel query processing. In *Advanced Database Systems*, pages 229–258. Springer, 1993.

[137] David DeWitt and Jim Gray. Parallel database systems : The future of high performance database systems. *Communications of the ACM*, 35(6) :85–98, 1992.

[138] Bernardo Miranda, Alexandre AB Lima, Patrick Valduriez, and Marta Mattoso. Apuama : combining intra-query and inter-query parallelism in a database cluster. In *International Conference on Extending Database Technology*, pages 649–661. Springer, 2006.

[139] S Yu Philip, Ming-Syan Chen, Joel L Wolf, and John Turek. Parallel query processing. In *Advanced Database Systems*, pages 229–258. Citeseer, 1993.

[140] Erhard Rahm and Robert Marek. Analysis of dynamic load balancing strategies for parallel shared nothing database systems. In *VLDB*, pages 182–193, 1993.

[141] Erhard Rahm and Robert Marek. Dynamic multi-resource load balancing in parallel database systems. In *VLDB*, volume 95, pages 11–15. Citeseer, 1995.

[142] Alekh Jindal, Konstantinos Karanasos, Sriram Rao, and Hiren Patel. Selecting subexpressions to materialize at datacenter scale. *Proceedings of the VLDB Endowment*, 11(7) :800–812, 2018.

[143] Timos K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1) :23–52, 1988.

[144] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. Scope : easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment*, 1(2) :1265–1276, 2008.

[145] Yasin N Silva, Paul-Ake Larson, and Jingren Zhou. Exploiting common subexpressions for cloud query processing. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1337–1348. IEEE, 2012.

[146] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. Sqlshare : Results from a multi-year sql-as-a-service experiment. In *The ACM Special Interest Group on Management of Data (ACM-SIGMOD)*, pages 281–293, 2016.

[147] Dihia Lanasri, Selma Khouri, and Ladjel Bellatreche. Trust-aware curation of linked open data logs. In *International Conference on Conceptual Modeling*, pages 604–614. Springer, 2020.

[148] Renato Marroquín, Ingo Müller, Darko Makreshanski, and Gustavo Alonso. Pay one, get hundreds for free : Reducing cloud costs through shared query execution. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 439–450, 2018.

[149] Deepika Saxena and Ashutosh Kumar Singh. Auto-adaptive learning-based workload forecasting in dynamic cloud environment. *International Journal of Computers and Applications*, pages 1–11, 2020.

[150] Andrew Pavlo, Matthew Butrovich, Ananya Joshi, Lin Ma, Prashanth Menon, Dana Van Aken, Lisa Lee, and Ruslan Salakhutdinov. External vs. internal : an essay on machine learning agents for autonomous database management systems. *IEEE bulletin*, 42(2), 2019.

[151] Yannis Kotidis and Nick Roussopoulos. Dynamat : A dynamic view management system for data warehouses. In *The ACM Special Interest Group on Management of Data (ACM-SIGMOD)*, pages 371–382, 1999.

[152] Peter Scheuermann, Junho Shim, and Radek Vingralek. WATCHMAN : A data warehouse intelligent cache manager. In *The International Conference on Very Large Data Bases (VLDB)*, pages 51–62, 1996.

[153] Prasan Roy and S. Sudarshan. Multi-query optimization. In *In [154]*. 2018.

[154] Ling Liu and M. Tamer Özsu, editors. *Encyclopedia of Database Systems, 2nd Edition*. Springer, 2018.

[155] Upen S. Chakravarthy and Jack Minker. Multiple query processing in deductive databases using query graphs. In *VLDB*, pages 384–391, 1986.

[156] Robin Rehrmann, Carsten Binnig, Alexander Böhm, Kihong Kim, Wolfgang Lehner, and Amr Rizk. Oltpshare : The case for sharing in OLTP workloads. *Proc. VLDB Endow.*, 11(12) :1769–1780, 2018.

[157] Ahmet Cosar, Ee-Peng Lim, and Jaideep Srivastava. Multiple query optimization with depth-first branch-and-bound and dynamic query ordering. In *International Conference on Information and Knowledge Management(ACM-CIKM)*, pages 433–438, 1993.

[158] Kyuseok Shim, Timos K. Sellis, and Dana S. Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data Knowl. Eng.*, 12(2) :197–222, 1994.

[159] Hossein Azgomi and Mohammad Karim Sohrabi. A game theory based framework for materialized view selection in data warehouses. *Engineering Applications of Artificial Intelligence*, pages 125–137, 2018.

[160] Stanley B. Zdonik and David Maier, editors. *Readings in Object-Oriented Database Systems*. Morgan Kaufmann, 1990.

[161] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. Scalable multi-query optimization for sparql. In *The International Conference on Data Engineering (ICDE)*, pages 666–677, 2012.

[162] Wenfei Fan, Jeffrey Xu Yu, Jianzhong Li, Bolin Ding, and Lu Qin. Query translation from xpath to SQL in the presence of recursive dtds. *VLDB Journal*, 18(4) :857–883, 2009.

[163] Anastasios Kementsietsidis, Frank Neven, Dieter Van de Craen, and Stijn Vansummeren. Scalable multi-query optimization for exploratory queries over federated scientific databases. *PVLDB*, pages 16–27, 2008.

[164] Pietro Michiardi, Damiano Carra, and Sara Migliorini. Cache-based multi-query optimization for data-intensive scalable computing frameworks. *Inf. Syst. Frontiers*, 23(1) :35–51, 2021.

[165] Alin Dobra, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Sketch-based multi-query processing over data streams. *Data Stream Management*, pages 241–261, 2016.

[166] Zahid Abul-Basher. Multiple-query optimization of regular path queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1426–1430. IEEE, 2017.

[167] Marek Wojciechowski Monika Rokosik. Efficient processing of streams of frequent itemset queries. In *The European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 15–26, 2014.

[168] T. Chen and K. Narita. Multiple query optimization in sql-on-hadoop systems. *US Patent 10,572,478*, 2020.

[169] Kevin O'Gorman, Divyakant Agrawal, and Amr El Abbadi. Multiple query optimization by cache-aware middleware using query teamwork. In *The International Conference on Data Engineering (ICDE)*, page 274, 2002.

[170] Imene Mami and Zohra Bellahsene. A survey of view selection methods. *SIGMOD Rec.*, 41(1) :20–29, 2012.

[171] Chun Jin and Jaime G. Carbonell. Predicate indexing for incremental multi-query optimization. In *The International Symposium on Methodologies for Intelligent Systems (ISMIS)* , pages 339–350, 2008.

[172] Ladjel Bellatreche and Amira Kerkad. Query interaction based approach for horizontal data partitioning. *IJDWM*, pages 44–61, 2015.

[173] Amine Roukh, Ladjel Bellatreche, Soumia Bouarar, and Ahacen Boukorca. Eco-physic : Eco-physical design initiative for very large databases. *Information Systems*, pages 44–63, 2017.

[174] Murat Ali Bayir, Ismail H Toroslu, and Ahmet Cosar. Genetic algorithm for the multiple-query optimization problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(1) :147–153, 2006.

[175] Marek Wojciechowski, Krzysztof Galecki, and Krzysztof Gawronek. Three strategies for concurrent processing of frequent itemset queries using fp-growth. In *KDID*, pages 240–258, 2006.

[176] Rada Chirkova, Jun Yang, et al. Materialized views. *Foundations and Trends® in Databases*, 4(4) :295–405, 2012.

[177] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize in a data warehouse. *IEEE Transactions on Knowledge and Data Engineering*, 17(1) :24–43, 2005.

[178] Michael Lawrence and Andrew Rau-Chaplin. Dynamic view selection for olap. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 33–44. Springer, 2006.

[179] Amine Roukh, Ladjel Bellatreche, Ahcène Boukorca, and Selma Bouarar. Eco-dmw : Eco-design methodology for data warehouses. In *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, pages 1–10, 2015.

[180] Wilburt Juan Labio, Dallan Quass, and Brad Adelberg. Physical database design for data warehouses. In *Proceedings 13th International Conference on Data Engineering*, pages 277–288. IEEE, 1997.

[181] Kenneth A Ross, Divesh Srivastava, and S Sudarshan. Materialized view maintenance and integrity constraint checking : Trading space for time. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 447–458, 1996.

[182] Himanshu Gupta. Selection of views to materialize in a data warehouse. In *International Conference on Database Theory*, pages 98–112. Springer, 1997.

[183] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *The ACM Special Interest Group on Management of Data (ACM-SIGMOD)*, pages 205–216, 1996.

[184] David E Goldberg. Genetic algorithms in search, optimization, and machine learning. addison. *Reading*, 1989.

[185] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing : Theory and applications*, pages 7–15. Springer, 1987.

[186] Chuan Zhang, Xin Yao, and Jian Yang. An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(3) :282–294, 2001.

[187] Mark Wallace. Practical applications of constraint programming. *Constraints*, 1(1) :139–168, 1996.

[188] Imene Mami, Remi Coletta, and Zohra Bellahsene. Modeling view selection as a constraint satisfaction problem. In *International Conference on Database and Expert Systems Applications*, pages 396–410. Springer, 2011.

[189] Nick Roussopoulos. The logical access path schema of a database. *IEEE Transactions on Software Engineering*, (6) :563–573, 1982.

[190] Prasan Roy, Srinivasan Seshadri, S Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 249–260, 2000.

[191] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *The ACM Special Interest Group on Management of Data (ACM-SIGMOD)*, pages 307–318, 2001.

[192] Xavier Baril and Zohra Bellahsene. Selection of materialized views : A cost-based approach. In *International Conference on Advanced Information Systems Engineering*, pages 665–680. Springer, 2003.

[193] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *The International Conference on Very Large Data Bases (VLDB)*, pages 136–145, 1997.

[194] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollar, Arun Marathe, Vivek Narasayya, and Manoj Syamala. Database tuning advisor for microsoft sql server 2005. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 930–932, 2005.

[195] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam J. Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 design advisor : Integrated automatic physical database design. In *The International Conference on Very Large Data Bases (VLDB)*, pages 1087–1097, 2004.

[196] Cristina Maier, Debabrata Dash, Ioannis Alagiannis, Anastasia Ailamaki, and Thomas Heinis. PARINDA : an interactive physical designer for postgresql. In *The International Conference on Extending Database Technology (EDBT)*, pages 701–704.

[197] Thomas Phan and Wen-Syan Li. Dynamic materialization of query views for data warehouse workloads. In *The International Conference on Data Engineering (ICDE)*, pages 436–445, 2008.

[198] Luis Leopoldo Perez and Christopher M. Jermaine. History-aware query optimization with materialized intermediate views. In *The International Conference on Data Engineering (ICDE)*, pages 520–531, 2014.

[199] Xi Liang, Aaron J. Elmore, and Sanjay Krishnan. Opportunistic view materialization with deep reinforcement learning. *CoRR*, abs/1903.01363, 2019.

[200] Qiufen Xia, Lizhen Zhou, Wenhao Ren, and Yi Wang. Proactive and intelligent evaluation of big data queries in edge clouds with materialized views. *Computer Networks*, 203 :108664, 2022.

[201] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in sql databases. *VLDB*, pages 496–505, 2000.

[202] Dimitri Theodoratos, Spyros Ligoudistianos, and Timos Sellis. View selection for designing the global data warehouse. *Data & Knowledge Engineering*, 39(3) :219–240, 2001.

[203] Negin Daneshpour and Ahmad Abdollahzadeh Barfourosh. Dynamic view management system for query prediction to view materialization. *International Journal of Data Warehousing and Mining (IJDWM)*, 7(2) :67–96, 2011.

[204] Yue Han, Guoliang Li, Haitao Yuan, and Ji Sun. Autoview : An autonomous materialized view management system with encoder-reducer. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[205] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *PVLDB*, pages 97–108, 2011.

[206] Bery Mbaiossoum, Ladjel Bellatreche, and Stéphane Jean. Materialized view selection considering the diversity of semantic web databases. In *East European Conference on Advances in Databases and Information Systems*, pages 163–176. Springer, 2014.

[207] Nicolas Bruno, Sapna Jain, and Jingren Zhou. Continuous cloud-scale query optimization and processing. *Proc. VLDB Endow.*, 6(11) :961–972, 2013.

[208] Ashish Tapdiya, Yuan Xue, and Daniel Fabbri. A comparative analysis of materialized views selection and concurrency control mechanisms in nosql databases. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 384–388, 2017.

[209] Ahcène Boukorca, Ladjel Bellatreche, and Alfredo Cuzzocrea. SLEMAS : an approach for selecting materialized views under query scheduling constraints. In *International Conference on Management of Data (COMAD)* , pages 66–73, 2014.

[210] Ü. V. Çatalyürek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. *Parallel Algorithms for Irregularly Structured Problems*, 1996.

[211] A. Yzelman and R. H. Bisseling. Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods. *SIAM Journal on Scientfic Computing*, pages 3128–3154, 2009.

[212] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS computational biology*, 5(5) :e1000385, 2009.

[213] Claude Berge. *Hypergraphs : combinatorics of finite sets*, volume 45. Elsevier, 1984.

[214] Elena V Konstantinova and Vladimir A Skorobogatov. Application of hypergraph theory in chemistry. *Discrete Mathematics*, 235(1-3) :365–383, 2001.

[215] Alain Bretto and Luc Gillibert. Hypergraph-based image representation. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 1–11. Springer, 2005.

[216] Aurélien Ducournau, Alain Bretto, Soufiane Rital, and Bernard Laget. A reductive approach to hypergraph clustering : An application to image segmentation. *Pattern Recognition*, 45(7) :2788–2803, 2012.

[217] Vinko Zlatic, Gourab Ghoshal, and Guido Caldarelli. Hypergraph topological quantities for tagged social networks. *CoRR*, abs/0905.0976, 2009.

[218] Joel Fuentes, Pablo Sáez, Gilberto Gutierrez, and Isaac D. Scherson. A method to find functional dependencies through refutations and duality of hypergraphs. *Computer Journal*, 58(5) :1186–1198, 2015.

[219] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism : a workload-driven approach to database replication and partitioning. *PVLDB*, 3(1) :48–57, 2010.

[220] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM (JACM)*, 30(3) :514–550, 1983.

[221] Sangeeta Sen, Anisha Agrawal, Ankit Rathi, Animesh Dutta, and Biswanath Dutta. An analytical approach for query optimization based on hypergraph. In *2015 12th International conference on electrical engineering/electronics, computer, telecommunications and information technology (ECTI-CON)*, pages 1–6. IEEE, 2015.

[222] Alain Bretto. *Hypergraph Theory : An Introduction*. Springer, 2013.

[223] Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees : An analysis of strategy spaces and its implications for query optimization. In *The ACM Special Interest Group on Management of Data (ACM-SIGMOD)*, pages 168–177, 1991.

[224] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. Reinforcement learning with tree-lstm for join order selection. In *The International Conference on Data Engineering (ICDE)*, pages 1297–1308, 2020.

[225] Yifan Qiao, Shengjie Wei, Ruiwei Gao, Nan Han, Shaojie Qiao, and Haiquan Song. A deep reinforcement learning model with plan value network for join order selection. *International Journal of Wireless and Mobile Computing*, 21(4) :365–374, 2021.

[226] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning : applications in vlsi domain. *IEEE Trans. Very Large Scale Integr. Syst.*, 7(1) :69–79, 1999.

[227] Sebastian Schlag. *High-Quality Hypergraph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020.

[228] Andrew E Caldwell, Andrew B Kahng, and Igor L Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(11) :1304–1313, 2000.

[229] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th design automation conference*, pages 175–181. IEEE, 1982.

[230] Petru Eles, Zebo Peng, Krzysztof Kuchcinski, and Alexa Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Design automation for embedded systems*, 2(1) :5–32, 1997.

[231] Charles J Alpert and Andrew B Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. In *31st Design Automation Conference*, pages 652–657. IEEE, 1994.

[232] Honghua Hannah Yang and DF Wong. Efficient network flow based min-cut balanced partitioning. In *The Best of ICCAD*, pages 521–534. Springer, 2003.

[233] Sung-Woo Hur and John Lillis. Relaxation and clustering in a local search framework : application to linear placement. *VLSI Design*, 14(2) :143–154, 2002.

[234] Ladjel Bellatreche, Kamalakar Karlapalem, Mukesh Mohania, and Michel Schneider. What can partitioning do for your data warehouses and data marts? In *Proceedings 2000 International Database Engineering and Applications Symposium (Cat. No. PR00789)*, pages 437–445. IEEE, 2000.

[235] Charles J Alpert and Andrew B Kahng. Recent directions in netlist partitioning : A survey. *Integration*, 19(1-2) :1–81, 1995.

[236] Ahcène Boukorca. *Hypergraphs in the Service of Very Large Scale Query Optimization. Application.* Phd thesis, École nationale supérieure de mécanique et d'aérotechnique, Chasseneuil-du-Poitou, Poitiers, 2016.

[237] Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. Clustering in a high-dimensional space using hypergraph models. 1997.

[238] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. Database tuning advisor for microsoft SQL server 2005. In *VLDB*, pages 1110–1121, 2004.

[239] Richard Strohm, Lance Ashdown, Mark Bauer, Michele Cyran, Steve Fogel, Janis Greenberg, Sumit Jeloka, Paul Lane, Diana Lorentz, Jack Melnick, et al. Oracle database concepts, 11g release 1 (11.1) b28318-05.

[240] Johan Kok Zhi Kang, Sien Yi Tan, Feng Cheng, Shixuan Sun, and Bingsheng He. Efficient deep learning pipelines for accurate cost estimations over large scale query workload. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1014–1022, 2021.

[241] Simon Pierre Dembele, Ladjel Bellatreche, Carlos Ordonez, and Amine Roukh. Think big, start small : a good initiative to design green query optimizers. *Cluster Computing*, 23(3) :2323–2345, 2020.

[242] Bery Leouro Mbaiossoum. *Conception physique des bases de données à base ontologique : le cas des vues matérialisées.* PhD thesis, ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique-Poitiers, 2014.

[243] Jonathan Lewis and Thomas Kyte. *Cost-based Oracle fundamentals.* Springer, 2006.