MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et de l'Informatique
Département de Mathématiques et d'Informatique
Filière : Informatique

MEMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : **Ingénierie des Systèmes d'Information**

THEME :

Génération un plan d'exécution global pour optimisation multi- requête dans entrepôt de données.

Etudiant(e) : « Bechelaghem Kenza »
« Boughliem chaimaa »
Encadrant(e) : « Mme Betouati Fatiha »

Année Universitaire 2018-2019

Dédicaces

Nous dédions ce modeste travail à :

Nos parents, qui n'ont jamais cessé de nous encourager et nous soutenir

Tous les membres de la famille Bechelaghem et Boughliem

Nos amies : Lakhdar Hanchour Haoua, Harizi Amira, Kerichich Nesrine et Belghachem Kheira.

Tous ceux qui nous sont chers.

Remerciements

C'est avec grand plaisir que je réserve cette page, en signe de gratitude et de reconnaissance à tous ceux qui nous ont aidés à la réalisation de ce travail.

Le grand merci au Dieu, le clément, le miséricordieux qui nous a donné la force et le courage de réaliser notre travail.

Merci infiniment à nos parents qui nous ont donné le coup de pousse jusqu'à la fin de notre projet.

Nous remercions cordialement ***Mm BETOUATI FATIHA*** *pour son encadrement et ses encouragements.*

Un grand merci à Mm ***Behnas Nacera*** *et Mr* ***Hamadi Abd el kader*** *qui ont été toujours disponibles pour nous apporter son aide.*

Nous remercions également Messieurs ***nabi Ali riyad*** *et surtout son frère* ***Nabi Rachid*** *qui a été toujours disponible pour nous aider.*

On remercie vivement Mesdames et Messieurs les membres du jury d'avoir accepter d'évaluer ce travail.

Enfin Un grand merci à tous nos amis pour leur dévouement et leur amitié sans faille.

Résumé

L'entrepôt de données est une exigence important pour les entreprises Grâce à son utilité de suivre, gérer et améliorer les performances de l'entreprise, il entre dans le cadre de la prise de décision et de l'analyse décisionnelle.

Ce mémoire se concentre sur l'étude du problème de la construction un plan d'exécution global pour optimisation multi- requête dans entrepôt de données basant sur les techniques d'optimisation.

Le plan d'exécution global vise, d'une part, à facilité d'apporter une réponse promptement et activement a une grande échelle et complexe nombre des requêtes décisionnelles et, d'autre part à améliorer la performance de ces requêtes. Et alors améliore la performance de l'entrepôt ainsi l'entreprise.

Ce présent rapport, résumera le déroulement de toutes les étapes du projet.

Mots Clés : l'Online Analytical Processing (OLAP), Entrepôt de données, Vue matérialisée, Fragmentation, Optimisation multi requêtes, MVPP.

Liste des figures

Liste des tableaux

Liste des abréviations

Abréviation	Expression Complète	Page
MVPP	*Multi-View Processing Plan.*	1
ED	Entrepôt de données.	3
OLTP	On-Line Transactional Processing.	3
OLAP	On-Line Analytical Processing	4
ETL	Extract-Transform-Load.	7
ROLAP	relational On-Line Analytical Processing.	9
MOLAP	Multidimensional On-Line Analytical Processing.	11
HOLAP	Hybrid On-Line Analytical Processing.	11
PEG	Plan d'Exécution Global.	19
FHP	Fragmentation horizontale primaire.	28
FHD	Fragmentation horizontale dérivée.	29
FH	Fragmentation horizontale.	30
HA	algorithmes heuristiques.	32

Table des matières

Introduction

Aujourd'hui, les entreprises ont besoin des applications analytiques pour répondre rapidement et efficacement à leur large et complexe nombre des requêtes décisionnelles. Un entrepôt de données (Data Warehouse) est une nécessité pour toute entreprise. Il permet un accès rapide et fiable aux diverses données importantes et aide à la prise de décisions au sein de l'entreprise.

Les entrepôts de données sont élaborés pour donner aux décideurs des systèmes dédiés à l'analyse des données volumineuses. Ils sont connus par leurs requêtes complexes caractérisées par des opérations telles que la jointure, sélections et agrégations… etc. La complexité de ces requêtes détériore de manière significative les performances de l'entrepôt ce qui nécessite de les ***optimiser*** pour faciliter la gestion des données.

L'optimisation multi requête est un problème connus dans le domaine de base de données, sa résolution vise à optimiser les performances globales d'une charge de travail de requête par ***la génération un plan d'exécution global*** qui sera exploité pour sélectionner des structures d'optimisation tel que *les vues matérialisées, l'indexation et la fragmentation* au but de minimiser le cout d'exécution de ces requêtes.

La génération un plan d'exécution global consiste à générer un arbre algébrique unifiable optimal pour une charge de requêtes complexe. Il existe cependant beaucoup plus de travaux concernant la construction de ce plan avec différent structure tel que MVPP, AND / OR view graph et Data Cube Lattice (Treillis) pour l'optimisation multi- requête dans entrepôt de données mais ils souffrent de scalabilité. Dans le cadre de ce projet nous nous intéressons à ***Multi-View Processing Plan (MVPP).***

L’objectif de notre travail est de construire plan d’exécution globale et qui sera utilisé comme une structure de donnée pour la sélection des structures d'optimisation *vue matérialisée ou fragmentation horizontale.*

Ce mémoire est organisé en trois chapitres comme suit :

Le premier chapitre sera consacré aux définitions des concepts fondamentaux relatifs de l’entrepôt de données .Nous commençons par un zoom sur la technologie d’entreposage de données, comme nous donnons un aperçu de la conception d'un entrepôt de données.

Le deuxième chapitre : présenter les travaux existants dans le contexte de construction plan d’exécution global. Nous présentons dans un premier temps notre problématique à résoudre comme nous donnons un aperçus sur le processus d’optimisation de requêtes passons par les phases d’optimisation et nous détaillons les deux technique d’optimisation *vues matérialisées* et *fragmentation horizontale*, enfin nous terminons par une synthèse.

Le troisième chapitre : sera consacré à la présentation de notre approche qui permet de résoudre le problème de la construction plan d’exécution global et finalement nous présentons l’implémentation de notre approche.

Chapitre 1

Entrepôt de données

1.1 Introduction

Les entrepôts de données ont émergé comme solution potentielle répondant aux besoins du stockage et de l'analyse de grands volumes de données, Nous allons détailler en cour de ce chapitre les concepts de base de l'entrepôt de données et cycle de vie de la conception d'entrepôt de données et les techniques d'optimisation des requêtes.

1.2 La technologie d'entrepôt de données

Les bases de données sont utilisées dans les entreprises pour gérer un volume d'informations importantes contenues dans leurs systèmes opérationnels. Ces données sont gérées selon des processus transactionnels en ligne (OLTP : "*On-LineTransactionalProcessing"*).

L'exploitation de l'information contenue dans ces systèmes opérationnels est devenue une préoccupation essentielle pour les dirigeants des entreprises qui désirent améliorer leur prise de décision par une meilleure connaissance de leur propre activité, de celle de la concurrence, des employés, des clients et des fournisseurs. Les entreprises sont donc à la recherche de systèmes supportant efficacement les applications d'aide à la décision, ce qui implique la naissance de l'entrepôt de données.

Le concept de l 'Entrepôt de données(ED), tel que connu aujourd'hui, est apparu pour la première fois en 1980 ; l'idée consistait alors à réaliser une base de données destinée exclusivement au processus décisionnel. Les nouveaux besoins de l'entreprise, les quantités

importantes de données produites par les systèmes opérationnels et l'apparition des technologies aptes à sa mise en œuvre ont contribué à l'apparition du concept « Entrepôt de données» comme support aux systèmes décisionnels [8].

1.3 Définitions

Définition 1. L'entrepôt de données (Data Warehouse) est défini en 1990 par Bill Inmon « Un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles et historiées, organisées pour supporter un processus d'aide à la d´décision. ».

L'entrepôt de données stocke des données provenant de différentes sources d'informations hétérogènes et distribuées. L'objectif principal d'un entrepôt de données est de fournir rapidement et de façon fiables informations utiles à la prise de décision [8].

- **Orientées sujet** : un ED rassemble et organise des données associées aux différentes structures fonctionnelles de l'entreprise, pertinentes pour un sujet ou thème et nécessaire aux besoins d'analyse [12].
- **Intégrées :** les données résultent de l'intégration de données provenant de différentes sources pouvant être hétérogènes [12].
- **Historiées :** les données d'un ED représentent l'activité d'une entreprise durant une certaine période (plusieurs années) permettant d'analyser les variations d'une donnée dans le temps [12].
- **Non-volatiles:** les données de l'ED sont essentiellement utilisées en interrogation (consultation) et ne peuvent pas être modifiées (sauf certains cas de rafraîchissement) [12].

Définition 2. Traitement analytique en ligne OLAP effectue une analyse multidimensionnelle des données d'entreprise et offre la possibilité d'effectuer des calculs complexes, une analyse des tendances et une modélisation de données sophistiquée [8].

Définition 3. Le modèle multidimensionnel est une représentation de données dans un espace n-dimensionnel. Habituellement appelé cube de données ou hyper cube. Il est utilisé

comme une structure de données pour sauvegarder les données et pour faciliter l'optimisation des requêtes OLAP. Un modèle multidimensionnel est composé d'un ensemble de dimensions et de faits [6].

- **Un fait :** il représente le sujet analysé, il est formé de mesures correspondantes aux informations de l'activité étudiée. Ces mesures sont calculables à partir d'un grand nombre d'enregistrements [7].
- **Une dimension :** elle représente l'axe d'analyse de l'activité. Elle regroupe des paramètres et des informations qui peuvent influencer sur les mesures d'activités du fait. Les dimensions possèdent des hiérarchies permettant de faire l'analyse d'un niveau faible de détail vers un niveau plus détaillé [7].

Figure 1 montre un exemple de cube représentant les activités de vente. Il a trois dimensions : Produit, Le temps et le client.

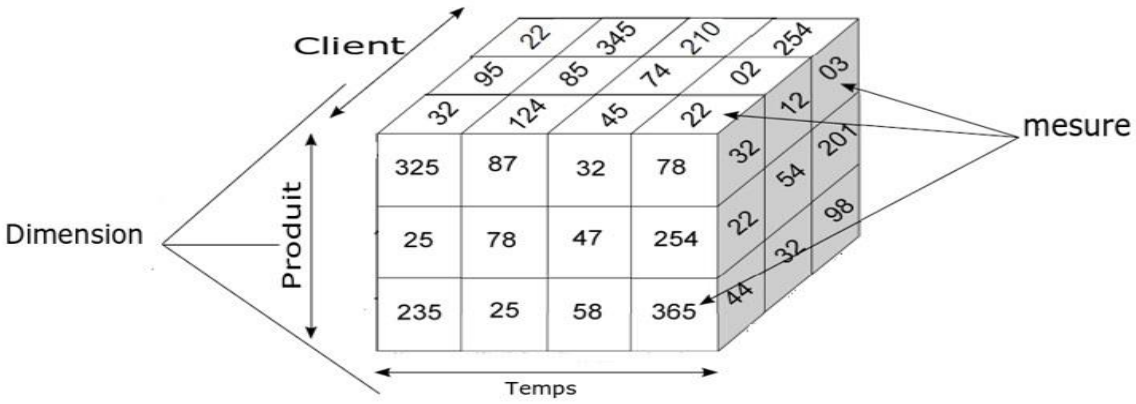


Figure 2– Un exemple de cube de données.

Définition 4. Magasin de données (Data marts) est un sous-ensemble d'un entrepôt de données destiné à fournir des données aux utilisateurs, et souvent spécialisé vers un groupe ou un type d'affaire [8].

1.4 Architecture d'un entrepôt de données [2]

L'objectif principal d'un entrepôt de données est de fournir rapidement et de façon fiable les informations utiles à la prise de décision, cela nécessite la reconstitution des

informations afin de les rendre plus facilement compréhensibles et utilisables. Afin d'atteindre cet objectif, l'architecture type d'un entrepôt de données, comme illustrée dans la figure 1, est structurée en quatre axes.

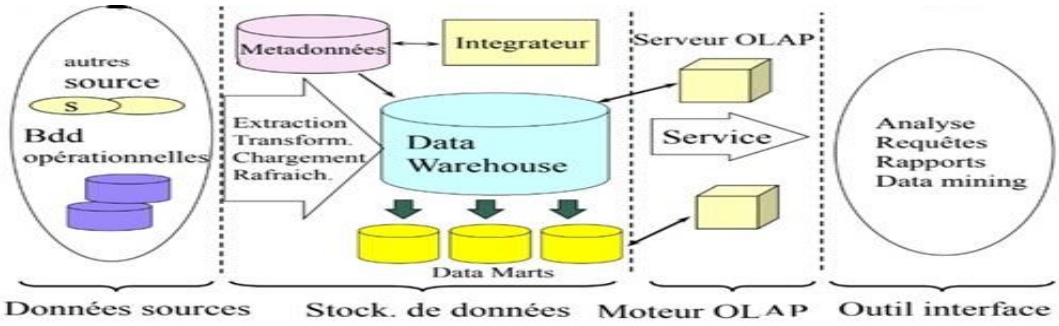


Figure 3– Architecture d'entrepôt de données [9].

- **Découverte des données** : Identification des données à importer dans l'entrepôt de donnée qui se trouve dans les systèmes sources.
- **Extraction des données** : Collection des données importantes dans les systèmes de production.
- **Transformation des données** : Rendre les données cohérentes avec la structure d'un entrepôt de données c'est-à-dire les données des systèmes de production doivent être agrégées ou calculées avant leur chargement.
- **Chargement des données** : Insertion des données au sein de l'entrepôt de données.

1.5 Cycle de vie d'entrepôts de données

Dans cette section, nous présentons les différentes phases du cycle de vie de sa conception héritées de l'architecture traditionnelle de bases de données Habituellement, il comprend les phases suivantes : la planification, la conception et l'implémentation, la maintenance et la gestion de l'évolution.

Le schéma ci-dessous représente la succession des tâches nécessaires à la mise en place des entrepôts de données efficaces [6]. La deuxième phase de conception comporte actuellement cinq principales phases, constituant le cycle de conception de l'entrepôt de données : l'analyse de besoins, la modélisation conceptuelle, la modélisation logique, le processus d'extraction-transformation-chargement (ETL) et une phase de déploiement et modélisation physique.

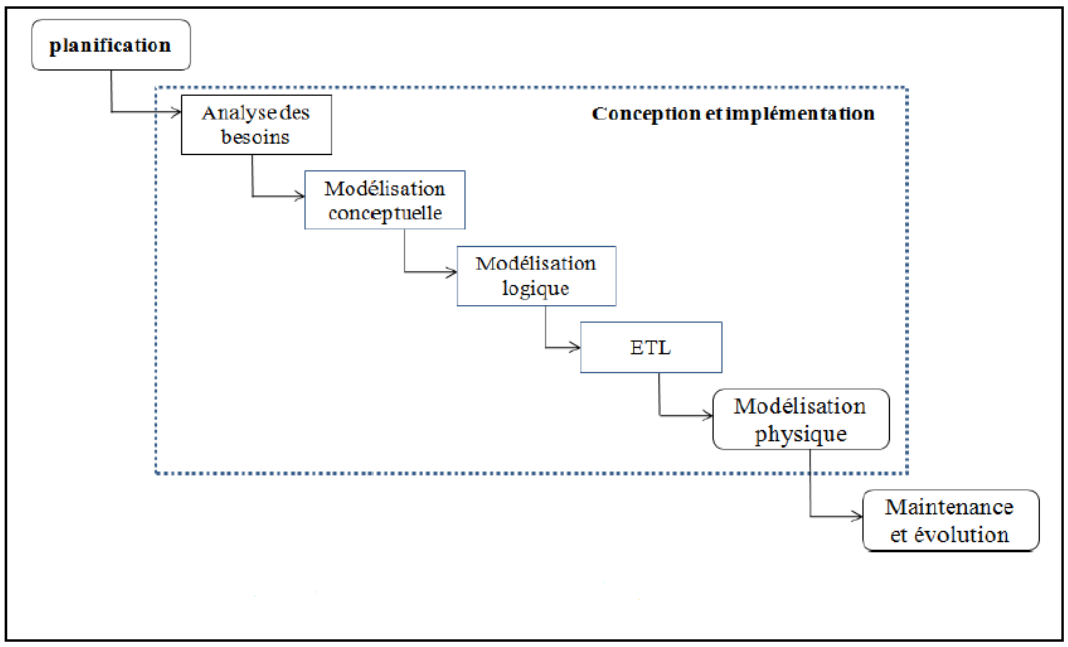


Figure 4– Cycle de vie d'entrepôt de données [1].

1.5.1 La planification

Cette phase consiste à déterminer l'étendue du projet ainsi que les buts et objectifs de l'entrepôt à développer, évaluer la faisabilité technique et économique de l'entrepôt et identifier les futurs utilisateurs de l'entrepôt [6].

1.5.2 La conception et l'implémentation

Cette phase consiste à développer le schéma de l'entrepôt et à mettre en place toutes les ressources nécessaires à son implémentation et à son déploiement [6].

1.5.3 Analyse des besoins

L'analyse des besoins joue un rôle clé dans tout projet en permettant de réduire le risque d'échec du projet. L'analyse des besoins pour les applications d'ED est définie comme le processus de développement des besoins selon un processus itératif et coopératif d'analyse du problème, de documentation des observations résultantes dans divers formats de représentation et de vérification des résultats obtenus [6]. Deux types de besoins sont distingués : les besoins fonctionnels et les besoins non fonctionnels [15].

- **les besoins fonctionnels** : sont ceux qui caractérisent le système (les besoins en matière de performance, de type de matériel ou de type de conception). Ils peuvent concerner les contraintes d'implémentation (langage de programmation, type SGBD, système d'exploitation, . . . etc.) [1].
- **les besoins non fonctionnelles** : Un besoin non fonctionnel est défini comme un attribut ou une contrainte du système comme la flexibilité, la performance, la sécurité, etc.

Deux méthodes sont utilisées pour collecter les besoins : une collecte orientée source et une collecte orientée utilisateur [15]. Les approches considérant une phase de définition des besoins des utilisateurs lors de la conception de l'entrepôt de données sont appelées approches orientées besoins ou approches descendantes.

Les approches orientées sources ou ascendantes se limitent à identifier les besoins de l'entrepôt de données à partir de l'ensemble des données disponibles au niveau des sources. L'analyse des besoins dans un projet d'entrepôt de données passe par les activités suivantes [15] :

- Planning de gestion des besoins : consiste à :
  - définir les objectifs du projet par les utilisateurs et les concepteurs.
  - définir les règles d'intégration des sources de données.
  - établir un planning de gestion des besoins du projet.

- Spécification des besoins: s'effectue par un processus itératif d'acquisition ou collecte des besoins, ensuite de représentation et spécification des besoins.
- Validation des besoins: consiste à valider les modèles initiaux construits lors de l'étape de spécification des besoins, avec les utilisateurs ainsi qu'avec les sources de données existantes. La validation des besoins est menée par l'équipe de développement, les utilisateurs et les experts du domaine.
- Suivi et gestion de l'évolution des besoins: la gestion doit être effectuée à deux niveaux :

1. une gestion de l'évolution des besoins des utilisateurs.
2. une gestion de l'évolution de l'architecture des sources.

1.5.4 La modélisation conceptuelle

La modélisation conceptuelle consiste à créer un schéma conceptuel pour la base de données, à partir des besoins déjà collectés et analysés, en utilisant un modèle de données conceptuelles de haut niveau, tel que le modèle entité-association ou digramme de classe UML. Ce schéma représente les objets du monde réel de la manière la plus réaliste possible [15].

1.5.5 La modélisation logique

Il y a trois approches principales pour représenter le modèle logique d'un ED, selon la façon dont le cube de données est stocké :

1.5.5.1 ROLAP ("relational On-Line Analytical Processing")

Les systèmes qui stockent les Données dans les bases de données relationnelles et utilise une extension du langage SQL pour traiter ces données.

Chaque fait correspond à une table appelée table de faits et chaque dimension correspond à une table appelée table de dimension. La table de faits possède comme attributs les mesures d'activités et les clés étrangères vers les tables de dimension. Ce modèle a pour

avantage l'utilisation des systèmes de gestion de bases de données existants, ce qui réduit le coût de mise en œuvre. Plusieurs modèles ont été proposés pour la modélisation des systèmes ROLAP, les plus utilises sont : le schéma en étoile, le schéma en flocon de neige [2].

a) Le schéma en étoile : Dans ce schéma, il existe une relation pour les faits et plusieurs pour les différentes dimensions autour de la relation centrale. La relation de faits contient les différentes mesures et une clé étrangère pour faire référence à chacune de leurs dimensions.

La figure 4 montre le schéma en étoile en décrivant les ventes réalisées dans les différents magasins de l'entreprise au cours d'un jour. Dans ce cas, nous avons une étoile centrale avec une table de faits appelée Ventes et autour leurs diverses dimensions : Temps, Produit, Magasin et Clients [7].

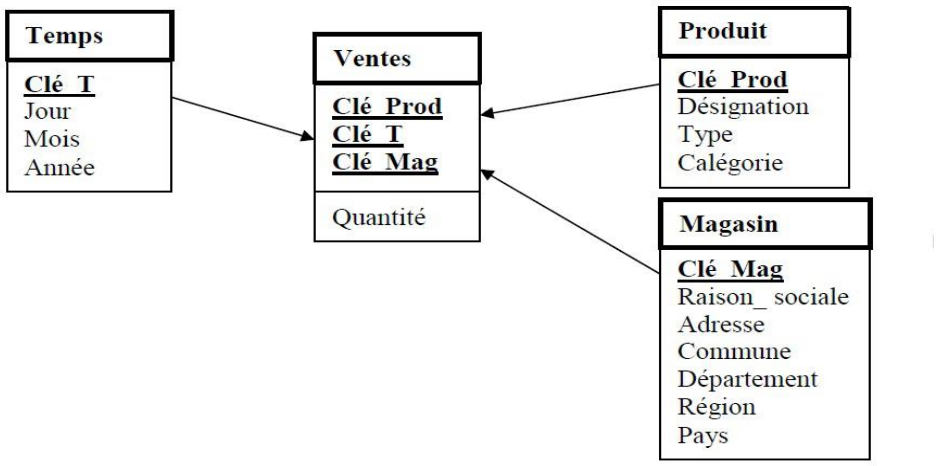


Figure 5 – Exemple de modélisation en étoile.

b) Le schéma en flocon de neige : Il dérive du schéma précédent avec une relation centrale et autour d'elle les différentes dimensions, qui sont éclatées ou décomposées en sous hiérarchies. L'avantage du schéma en flocon de neige est de formaliser une hiérarchie au sein d'une dimension, ce qui peut faciliter l'analyse. Un autre avantage est représenté par la normalisation des dimensions, car nous réduisons leur taille.

La figure 5représente un schéma en flocon de neige. Sur ce schéma, deux hiérarchies sont représentées : (Jour, Mois, Trimestre, Année) et (Ville, Département, Pays) [7].

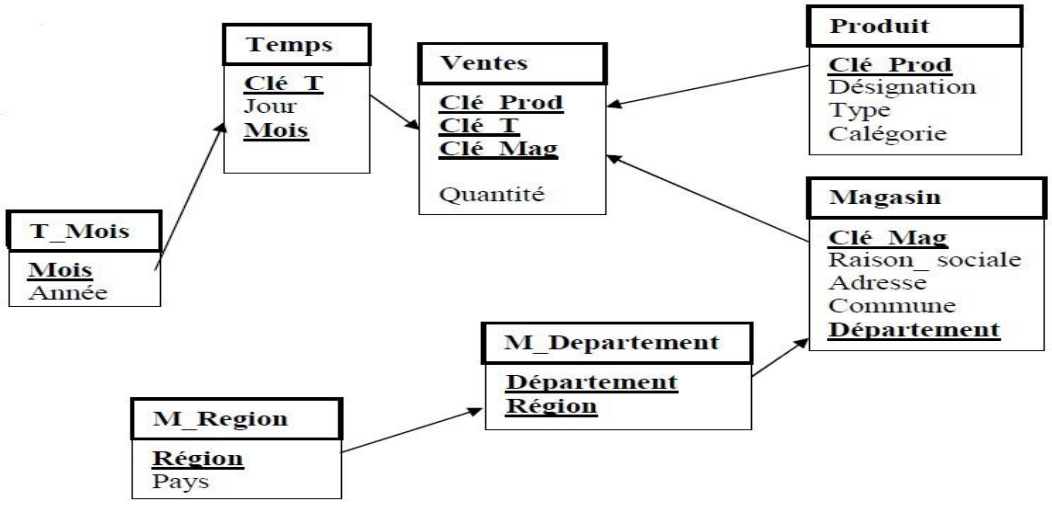


Figure 6– Exemple de modélisation en flocon de neige.

1.5.5.2 MOLAP (”Multidimensional On-Line AnalyticalProcessing”)

Les systèmes stockent les données dans un SGBD multidimensionnel sous la forme d'un tableau multidimensionnel où chaque dimension est associée à une dimension du cube. L'intérêt de cette approche est l'optimisation du temps d'accès, mais elle présente certaines limites telles que :

- le besoin de redéfinir des opérations pour manipuler les structures multidimensionnelles.
- la difficulté de la mise à jour et de la gestion du modèle.
- la consommation de l'espace lorsque les données sont éparses, ce qui nécessite l'utilisation des techniques de compression. [2]

1.5.5.3 HOLAP (”Hybrid On-Line Analytical Processing”)

Le système HOLAP représente les données fréquemment utilisées (généralement les données agrégées) dans un tableau multidimensionnel, et représente les données non fréquemment utilisées dans un schéma relationnel [2].

- **Phase ETL (”Extract-Transform-Load”)**

Cette phase consiste à extraire les données à partir des sources sélectionnées et à effectuer les transformations nécessaires pour assurer le chargement des données des sources

au niveau du schéma cible de l'entrepôt. Les données sont extraites à partir des sources de données hétérogènes. Elles sont ensuite propagées dans une zone de stockage temporaire où auront lieu leur transformation, homogénéisation et nettoyage. Enfin, les données sont chargées dans l'entrepôt de données cible [8].

- **Phase de déploiement**

Cette phase consiste à choisir la plate forme adéquate sur laquelle l'entrepôt cible sera déployé. Plusieurs plates-formes peuvent candidats pour stocker l'entrepôt (Clusters centralisés, machines parallèles, Cloud, etc.) [6].

- **La Modélisation physique**

Cette étape consiste à implémenter physiquement le modèle logique de l'entrepôt de données et à spécifier les techniques et les schémas d'optimisation de l'entrepôt, accompagné d'une spécification détaillée des caractéristiques physiques de l'entrepôt de données (les types de données, la segmentation des tables, les paramètres de stockage, la spécification des clés des tables, la gestion des instances). Le choix et l'application des techniques d'optimisation se fait également lors de la phase physique nous détaillons ce point dans le chapitre suivant [6].

- **Les techniques d'optimisation**

Afin d'adopter une politique d'optimisation d'un entrepôt de données, il faut choisir la structure d'optimisation, la nature de sélection et l'algorithme de sélection. Ces techniques appartiennent à deux catégories [2] :

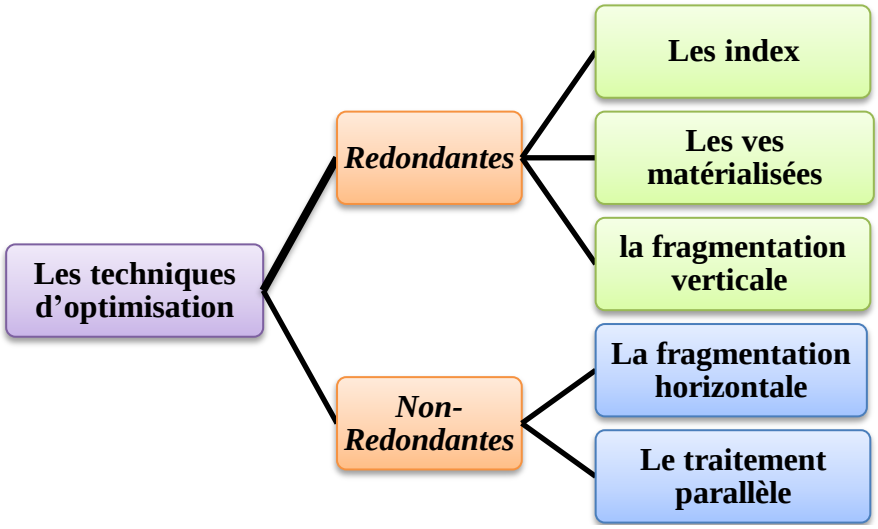


Figure 7 – Les techniques d'optimisation.

– ***Redondantes:*** Les techniques redondantes sont des techniques d'optimisation des requêtes qui nécessitent un stockage physique des données et une maintenance dont l'utilisation produit une duplication des données. On distingue : les index, les vues matérialisées et la fragmentation verticale.

  - **Les index :** Les techniques d'indexation ont été largement étudiées et constituent une option très importante pour la phase de conception physique des bases de données traditionnelles et avancées [6].

    Les index sont utilisés pour améliorer le temps d'accès aux données. La définition des index se fait souvent pendant l'exploitation de la base de données. Certaines techniques d'indexation sont apparues dans le contexte d'entrepôts de données, vu la nature de requêtes OLAP. Ce sont par exemple les index binaires, les index de jointures binaires, les index de jointure en étoile, etc. [6].

  - **Les vues matérialisées :** Une vue est une requête nommée. Elle est dite matérialisée si son résultat est stocké physiquement. Les vues améliorent l'exécution des requêtes en pré-calculant les opérations les plus coûteuses comme la jointure et l'agrégation, et en stockant leurs résultats dans la base. Dans cette situation, certaines requêtes nécessitent seulement l'accès aux vues matérialisées

et par conséquent sont exécutées plus rapidement, nous détaillons dans le chapitre suivant [5].

- **La fragmentation verticale :** La fragmentation verticale consiste à diviser une relation en sous relations appelées fragments verticaux résultant de l'application de l'opération de projection. La fragmentation verticale favorise le traitement des requêtes de projection portant sur les attributs utilisés dans le processus de la fragmentation, en limitant le nombre de fragments auxquels accéder. Son inconvénient est qu'elle requiert des jointures supplémentaires lorsqu'une requête accède à plusieurs fragments [5].

– ***Non redondantes :*** Les techniques non redondantes sont des techniques d'optimisation des requêtes qui ne nécessitent pas un stockage physique des données, ces techniques ne dupliquent pas les données mais réorganisent leur représentation physique. On distingue : la fragmentation horizontale et le traitement parallèle.

- **La fragmentation horizontale :** La fragmentation horizontale consiste à diviser un objet de la base de données (table, index, vues) en sous-ensembles de lignes appelés fragments horizontaux résultant de l'application de l'opération de restriction. La reconstruction de l'objet à partir de ces fragments horizontaux est obtenue par l'opération d'union de ces fragments et dans le chapitre suivant nous allons détailler [7].
- **Le traitement parallèle :** le traitement parallèle permet d'exécuter les opérations en parallèle pour un gain de temps de traitement [7].

1.5.6 La maintenance et la gestion de l'évolution

Cette phase implique l'optimisation de ses performances périodiquement. L'évolution de l'entrepôt de données concerne la mise à jour de son schéma en fonction des différents changements survenant au niveau des sources ou des besoins des utilisateurs [6].

1.6 Les facteurs qui influant les traitements des requêtes dans ED

Dans cette section, nous allons voir les facteurs qui influant les traitements des requêtes.

Dimension 1 : Le modèle logique d'entrepôt de donnée

La conception logique d'un entrepôt de données vise à organiser et stocker les informations des sources de données par sujet fonctionnel. Les entrepôts de données sont modélisés par plusieurs schémas : schémas en étoile, en flocon de neige et schéma en constellation. Nous nous intéressons à la modélisation en schéma en étoile [8].

Le tableau ce dessus montre la différence d'utilisation de deux modélisations :

Tableau 1– La différence d'utilisation de modélisation.

	Schéma en étoile	**Schéma en flocon de neige**
Structure de schémas	Contient du table fait et table de dimension.	Contient des tables de sous dimension comprenant des tables de fait et de dimension
Modèle de données	Top-down	Bottom-up
Complexité du requête	Faible	Haute
Jointure utilisée	Moins	Grand nombre de jointure
Utilisation du l'espace	Plus	Moins
Temps de consommation lors de l'exécution de la requête	Prend moins de temps	Prend plus en raison de l'utilisation excessive de la jointure

Dimension 2 : Requêtes OLAP

Les requêtes typiques sont définies sur le schéma logique le plus répandu, le schéma en étoile sont appelées requêtes de jointure en étoile. Ils offrent un temps de réponse rapide pour les requêtes. Ils ont les caractéristiques suivantes [8]:

- Il y a des jointures multiples entre la table des faits et les tables de dimension.
- Il n'y a pas de jointure entre les tables de dimensions.
- Chaque table de dimension impliquée dans une opération de jointure a plusieurs prédicats de sélection sur ses attributs descriptifs.

ROLAP utilise une extension des langages SQL qui inclut des opérations OLAP telles que cube, roll-up, drill-down, etc. Pour MOLAP, est principalement utilise le langage MDX (expressions multidimensionnelles) [8].

La syntaxe générale de requêtes SQL

SELECT <Liste de projection><Liste d'agrégation>

FROM <Nom de la table des faits><Liste de noms de tables de dimension>

WHERE <Liste de prédicats de sélection& jointure>

GROUP BY <Liste des attributs de tables de dimension>

La syntaxe générale de requêtes MDX

SELECT <spécification d'axe>

From<cube>

[WHERE <spécification de slice>]

Dimension 3:L'optimisation des requêtes

L'optimisation des requêtes est un processus essentiel pour tout SGBD relationnel qui sert à trouver meilleur plan d'exécution d'une requête donnée en trouvant un ensemble de solution. Il existe deux tâches principales à réaliser

1. Trouver l'ensemble des plans d'exécution pour une même requête donnée
2. Choisir parmi ces alternatives des plans les plus performantes pour exécuter la requête en se basant sur certains critères.

Il y a deux genres d'optimiseurs : Optimiseurs de requêtes individuelles et optimiseurs de requêtes multiples (dans le chapitre suivant nous allons détailler) [8].

Dimension 4: la conception physique

La conception physique est le processus de détermination de l'organisation du stockage de données et des caractéristiques d'accès aux données afin d'assurer l'intégrité, la sécurité et les performances du système, il faut de choisir les bonnes structures d'optimisations physiques [6].

Plusieurs structures ou des techniques d'optimisation ont été proposées dans le contexte d'entrepôt de données comme nous avons vu précédemment afin d'optimiser les requêtes.

1.7 Conclusion

Nous avons vu que les entrepôts de données représentent le cœur des systèmes décisionnels de l'entreprise. C'est une structure qui a pour but de regrouper les données de l'entreprise à des fins analytiques et pour aider à des prises de décisions stratégiques.

Dans ce chapitre nous avons présenté les entrepôts de données, aussi leurs caractéristiques et leurs architectures générales. Nous avons aussi vu le cycle de vie des entrepôts de données avec le modèle multidimensionnel qui peut être implémenté sur trois modèles : MOLAP, HOLAP et ROLAP**.** Deux schémas principaux les plus utilisés pour la modélisation des systèmes ROLAP : le schéma en étoile et le schéma en flocon de neige, nous avons mis l'accent sur la classification des différentes techniques d'optimisation. Ces techniques appartiennent à deux catégories : redondantes comme les index et les vues matérialisées et non redondantes comme la fragmentation horizontale et le traitement parallèle. Dans le chapitre suivant, nous détaillerons les différentes étapes de la phase de conception physique de l'entrepôt de données et surtout notre problématique et étudierons les travaux existant pour résoudre notre problème.

Chapitre 2

Plan d'exécution global MVPP

2.1 Introduction

Dans ce chapitre, en section 2 nous allons commencer par introduire notre problématique.

Les sections3 et 4 présentent le processus d'optimisation et exploitation plan global pour la sélection des structures d'optimisation. La section 5 décrit quelques travaux de construction du plan global MVPP. La dernière section conclut le chapitre.

2.2 Problématique

Vu la masse importante des données dans ED manipulées par différentes entreprises actuellement, il nécessite de répondre rapidement et efficacement à large et complexe nombre des requêtes décisionnelles, Ces requêtes connues sous le nom « big query », qui sont en grande interaction, partagent plusieurs opérations algébriques en commun, tel que la jointure qui représente l'opération la plus coûteuse et qui diminue les performances de l'entrepôt de données. En plus de ce qui précède, le problème connu sous le nom optimisation multi requête souffre de scalabilité car ne prend pas en considération les nouveaux aspects apportés par le contexte big query qui sont : volume de données, volume de requête et big interaction. Ainsi, il faut optimiser la performance totale des requêtes par la génération un plan d'exécution global dans le contexte big query au but d'augmenter la performance de l'entrepôt.

On peut formaliser notre problème de construction du plan global utilisé pour la sélection des structures d'optimisation dans un entrepôt de donnée comme suit :

- ***Input***
  - Un entrepôt de données (Data Warehouse DW).
  - Une charge de requête (workload Q).
- ***Out put***
  - Plan d'Exécution Global de Multi requêtes (PEG).
  - Sélection structure d'optimisation *vues matérialisées* ou *la fragmentation horizontale en utilisant PEG*.
- ***Objectif***
  - Réduire le temps d'exécution de Q.

2.3 Processus d'optimisation de requêtes dans Entrepôt de données

L'optimisation des requêtes est un sujet essentiel dans le secteur des bases de données, Elle a comme objectif de ordonnancer de manière optimisée les actes impliquées dans une requête, en utilisant les structures d'optimisation nécessaire pour une exécution en un temps minimum. Cet ordonnancement, dit plan d'exécution [11].

L'une des taches essentielles d'un optimiseur de requête a pour objectif de sélectionner le plan avec une période minimum.

2.4 Types d'optimisation

Il existe deux types d'optimisation, qui sont démontrés dans le schéma ci-dessous :

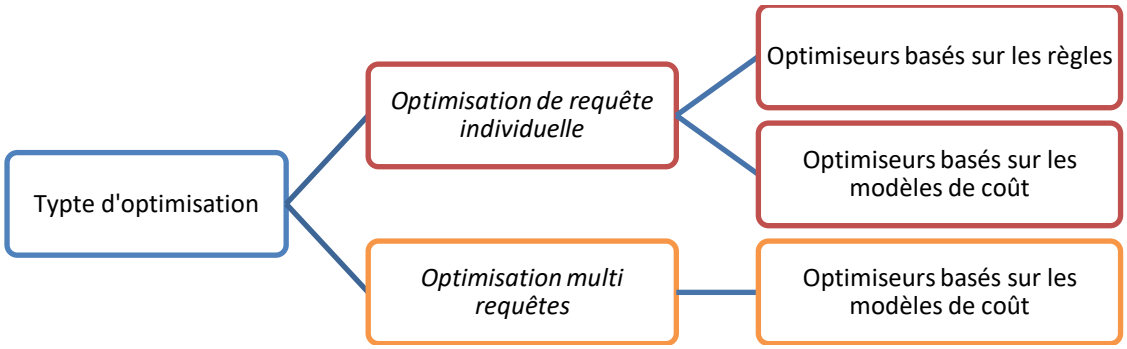


Figure 8 – Type d'optimisation de requêtes.

2.4.1 Optimisation de requête individuelle

Consiste à représenter chaque requête par un arbre algébrique, cette arbre représente une question dont les nœuds terminaux représentent les tables de la base, les nœuds intermédiaires sont des opérations de l'algèbre relationnelle tels que sélection, projection jointure ou union, etc., le nœud racine est le résultat d'une question et les arcs sont les flux de données entres les opérations [13].

2.4.2 Optimisation multi requêtes

Consiste à fusionner les plans individuels de façon à ce que le coût d'exécution des requêtes soit minimisé dans une large échelle de données. Mais trouver la meilleure fusion possible entre les différents plans individuels reste toujours un problème difficile. Ce problème a été largement étudié, sous le nom d'optimisation multi query depuis 80 par Sellis et al qu'ont proposés un algorithme de recherche nommé A * pour produire un plan d'exécution global basé sur tous les plans locaux possibles de chaque requête [8].

Les méthodes proposées par Sellis et al prennent beaucoup d'espace mémoire et du temps de réponse pour la construction du plan d'exécution global optimisé. De fait, plusieurs algorithmes heuristiques sont proposés pour optimiser le temps de construction [8]. Nous allons détailler dans partie travaux.

Il existe plusieurs structure graphique proposée dans le contexte d'optimisation multi requêtes pour la construction du plan global parmi ces structures on trouve :

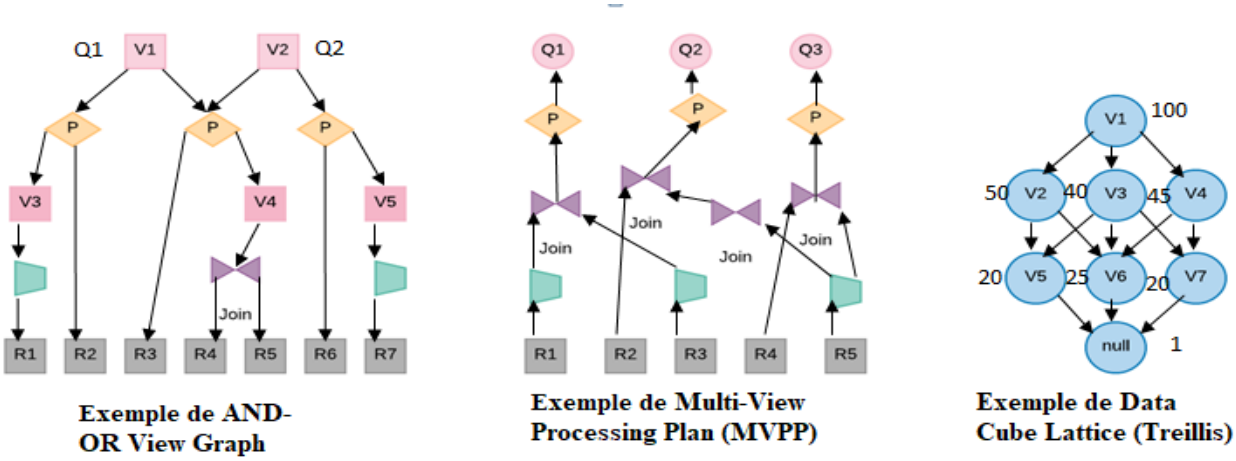


Figure 9– Des exemples de structures de données graphe orienté acyclique [3].

- **Multi-View Processing Plan (MVPP):** est une représentation graphique de la charge proposée dans le contexte d'optimisation multi requête a été introduit par Yang et al [19].

Le MVPP a plusieurs niveaux sont **:** [5]

- Le niveau 0 est les nœuds terminaux qui représentants les tables de base de l'entrepôt.
- Dans le niveau 1, nous trouvons des nœuds qui représentants les opérations algébriques unaires (sélection, projection)
- Dans le niveau 2, les nœuds représentent les opérations algébriques binaires comme la jointure, l'union, etc.
- Le dernier niveau représente les résultats de chaque requête.

- **AND-OR View Graph: est** un graphe acyclique dirigé DAG appelé graphe de vue AND-OR pour les vues. Qui peut être considéré comme l'union de tous les plans d'exécution possibles de chaque requête [3].

Il est composé de deux types de nœuds: les nœuds d'opération et les nœuds équivalents.

- **Nœuds d'opération :** représentants une opération du plan de requêtes tels que sélection, jointure, projection, etc.

- **Nœud d'équivalence** représentants les expressions logiques équivalentes (c'est-à-dire qui produisent le même résultat).

- **Data Cube Lattice (Treillis) :** est une représentation graphique algébrique, proposé dans le contexte d'un stockage de données multidimensionnel, elle contient les nœuds qui sont les requêtes ou les vues, et les arcs représentent la relation entre les vues dans chaque nœud [8].

Dans notre travail nous nous intéressons à la structure *Multi-View Processing Plan (MVPP).*

2.5 Les phases d'optimisation

Le processus d'exécution d'une requête donnée passe par quatre étapes principales qui sont : l'analyse, la réécriture, la planification / l'optimisation et l'exécution.

La figure ci-dessous présente les phases d'optimisation de requêtes.

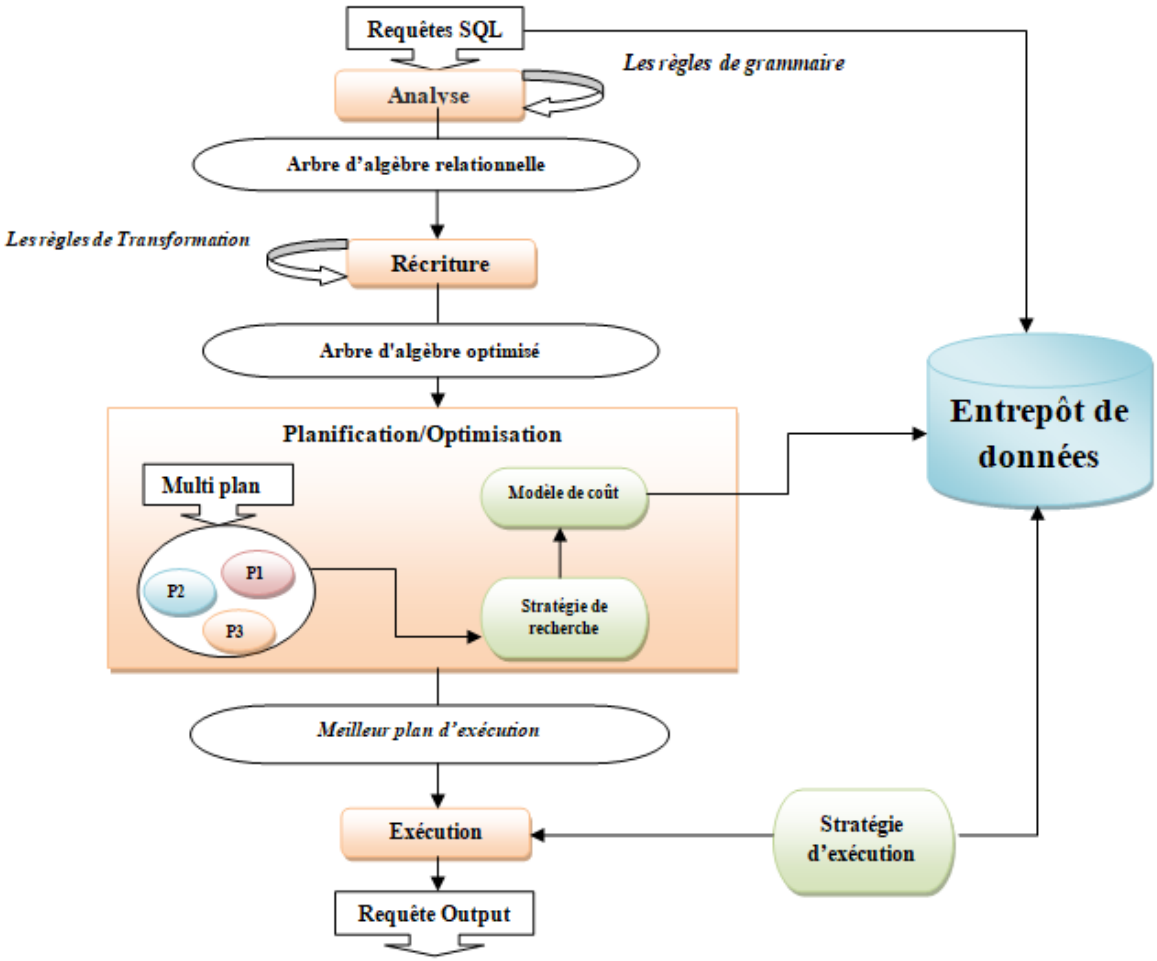


Figure 10– Les phases d'optimisation des requêtes [3].

2.5.1 Analyse syntaxique

Dans cette phase l'analyseur fait vérifier la chaîne de caractères constituant la requête syntaxiquement pour qu'elle soit valide à l'aide d'un ensemble de règles de grammaire, si la syntaxe est correcte traduit la requête en une expression d'algèbre relationnelle équivalente et construit une arborescence de la requête, sinon une erreur est retournée [3].

2.5.2 Réécriture

La phase de réécriture des requêtes traite et récrit l'arbre remis par la phase précédente (Analyse syntaxique) à l'aide d'un ensemble de règle de transformation [8].

2.5.3 Planification/Optimisation

Cette phase consiste à créer un plan d'exécution optimal, examine tout les plans d'exécution possibles de chaque arbre algébrique et choisit le moins coûteux. Cette phase se passe par les étapes suivantes :

- utilise une *stratégie de recherche* qui examine l'espace des plans d'exécution d'une façon particulière.
- compare les différents plans en utilisant les coûts calculés à l'aide *d'un modèle de coûts* tenant compte des coûts de processeur, ainsi que des coûts d'entrée/sortie (E/S).

***La stratégie de recherche (*Search strategy)**

Le rôle d'une stratégie de recherche est d'exploiter l'ensemble des plans d'exécution candidats d'en trouver celui de coût optimum en utilisant les modèles de coût, Pour chaque plan candidat, son coût d'exécution est calculé grâce au modèle de coût et le plan ayant le coût minimal est choisi pour évaluer la requête [17].

Nous distinguons deux classes de stratégies d'optimisation de l'ordre de jointure qui sont les algorithmes déterministes, aléatoires.

Stratégies déterministes : consiste à énumérer tous les plans d'exécutions possibles pour une requête donnée. Tel qu'à partir des sous plans optimisés elles ont établi des plans d'exécution. Pour réduire le cout d'optimisation, cette stratégie élague les sous plans qui ne sont pas capables de diriger au plan optimal aussitôt que probable, et les plans optimaux sont utilisés pour générer le plan global[1].

Les stratégies aléatoires ou **les stratégies randomisées** [1] ces stratégies explorent aléatoirement l'espace des plans. Dans cette classe des stratégies on trouve 3 genres qui sont :

- **Amélioration itérative** : cette stratégie consiste à choisir aléatoirement un ensemble des plans et tente de retrouver pour tous d'eux le plan optimum.
- **Recuit simulé** : vise à optimiser un plan à partir des transformations successives. Ces transformations améliorent le plan. Cette stratégie, bien connue en recherche opérationnelle, permet de converger vers un plan proche de l'optimum.
- **Stratégies génétiques** cette stratégie vise à fusionner deux plans pour en obtenir un troisième.

Modèle de coût

Le but d'un modèle de coût est d'estimer le coût d'exécution des plans. Il permet ainsi de choisir le meilleur plan d'exécution ayant le moindre coût d'exécution. Ce modèle se compose de différents types d'éléments, comme le coût de stockage secondaire, le coût de stockage de mémoire et le coût de calcul [17].

2.5.4 Exécution

Cette phase consiste à traiter les plan fournit par la phase précédente planification/optimisations pour sortir la totalité de tuples exigé au sein d'un mécanisme de pipeline. Chaque fois qu'un nœud du plan est appelé, il doit livrer le prochain tuple, ou signaler qu'il n'y a plus de tuples à livrer [3].

2.6 Exploitation plan global pour la sélection des structures d'optimisation

L'exploitation un plan global a un rôle très important dans l'optimisation mulit-requêtes qui consiste à optimiser la performance totale des requêtes.

Il existe une large panoplie de structures d'optimisation supportées. Chacune de ces structures est liée à un problème d'optimisation. Nous allons étudier deux structures la

première redondante *vues matérialisée* et la deuxième non redondante *fragmentation horizontale.*

2.6.1 Cas d'étude 1 : Les vues matérialisées

Définition1

Une vue est une requête nommée, définit une relation logique construit à partir des tables, mais qui n'existe pas physiquement [8].

Définition2

Une vue matérialisée est une table qui permet de stocker le résultat d'une requête. Les requêtes qui sont fait sur une vue matérialisées, elles sont cherchées directement les données dans celle-ci, sans passer par les table d'origine d'entrepôt de données. Alors elles permettent d'offrir des réponses rapides pour des requêtes [5].

Les vues matérialisées ont plusieurs objectifs, comme l'amélioration de la performance des requêtes.

Les vues sont associées à trois problèmes majeurs :

2.6.1.1 Le problème de sélection des vues matérialisées

L'administrateur d'un entrepôt doit déterminer quelles vues doivent être matérialisées. Il dispose un ensemble de contraintes telles que l'espace de stockage limité et cout de maintenance, cout énergétique...etc. En somme, il ne peut pas tout matérialiser [7].

Le problème de sélection de vue est défini comme la sélection de vues à matérialiser afin de minimiser le temps de réponse de la requête sous une contrainte de ressource[10].

On peut formaliser le problème de sélection des vues matérialisées comme suit :

- ***Input***
  - Un entrepôt de données (Data Warehouse DW).
  - Une charge de requête (workload).

- ***Out put***
  - La sélection des vues matérialisées.
- ***Objectif***
  - Minimiser le coût d'exécution d'une charge de requêtes.

2.6.1.2 La maintenance des vues matérialisées

Un entrepôt de données contient un ensemble de vues matérialisées qui ne résident pas dans l'entrepôt de données en cas de changement ou d'évolution dans les tables de base à cause de mise à jour il faut reporter dans les vues matérialisées, sinon leurs contenus deviendront obsolètes et leurs objets ne représenteront plus la réalité| [9]. Pour résoudre ce problème d'inconsistance des données, une procédure de maintenance des vues doit être mise en place.

Trois stratégies fondamentales ont été proposées [9] :

- Les vues sont mises à jour ***périodiquement*** : les vues sont mises à jour continuellement à des périodes précises.
- Les vues sont mises à jour ***immédiatement*** à la fin de chaque transaction.
- Les modifications sont propagées d'une manière ***différée.*** Dans ce cas, une vue est mise à jour uniquement au moment où elle est utilisée par une requête d'un utilisateur.

2.6.1.3 La réécriture des requêtes en fonction des vues

Consiste à réécrire toutes les requêtes définies sur l'entrepôt en fonction des vues. Mais la sélection de la meilleure réécriture pour une requête reste une tache difficile. Ce processus a été utilisé comme une technique d'optimisation pour réduire le coût d'exécution d'une requête. Il est supporté par la majorité des SGBD multidimensionnels [5].

2.6.2 Cas d'étude 2 : La fragmentation horizontale

Définition1

La fragmentation est une technique d'optimisation logique. Elle consiste à partitionner le schéma d'une base de données en plusieurs sous-schémas dans le but de réduire le temps d'exécution des requêtes [4].

Définition2

La fragmentation horizontale consiste à la décomposition de la table en groupes de lignes appelés fragments horizontaux. Elle permet de réduire la complexité des requêtes décisionnelles exécutées sur un entrepôt de données relationnel [9].

Il existe deux types de fragmentation horizontale:

- Primaire.
- Dérivée.

2.6.2.1 Fragmentation horizontale primaire (FHP)

La Fragmentation horizontale est définie par l'opération de sélection et la reconstruction de la relation se fait par l'union.

Exemple

Client (NCL, Nom, Ville).

Peut être fragmentée:

Client1= SELECT * FROM Client WHERE Ville = "Mostaganem".

Client2= SELECT * FROM Client WHERE Ville <> "Mostaganem".

Reconstruction de la relation initiale: Client = Client1 ∪ Client2.

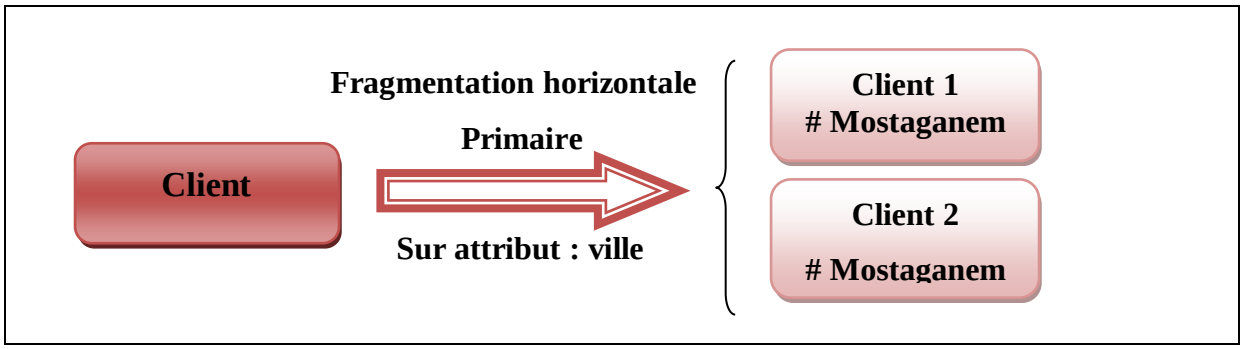


Figure 11 – Principe de la fragmentation primaire sur la table de dimension.

2.6.2.2 Fragmentation horizontale dérivée (FHD)

Ce type de fragmentation consiste à fragmenter une table en fonction des fragments horizontaux d'une autre table a condition il y'a une relation entre ces deux tables cette relation appeler relation père-fils. Fragmentation dérivée à un but pour accélérer les opérations de jointure [5].

Exemple

Commande1= SELECT * FROM Commande WHERE NCL IN (SELECT NCL FROM CLIENT1).

Commande2= SELECT * FROM Commande WHERE NCL IN (SELECT NCL FROM CLIENT2).

Reconstruction de la relation initiale: Commande = Commande1 ∪ Commande2.

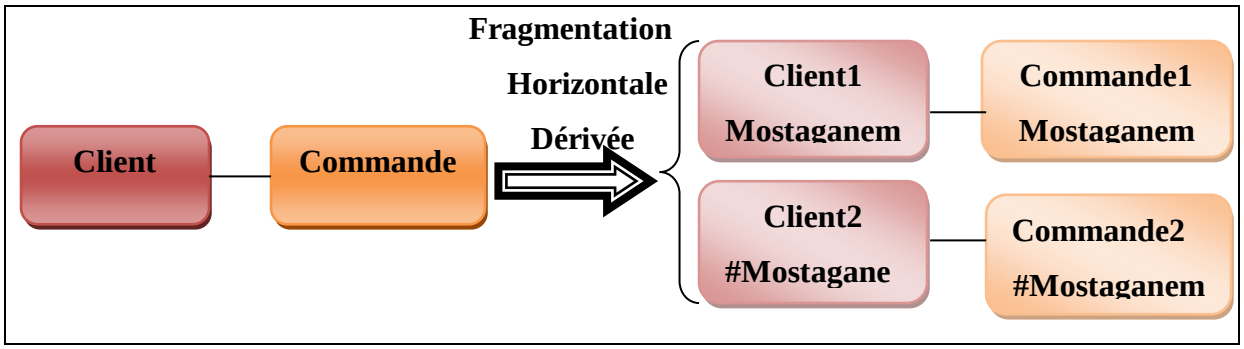


Figure 12 – Principe de la fragmentation dérivée sur la table Commande.

2.6.2.3 Démarche classique de la fragmentation horizontale

Dans les entrepôts de données, la FH est effectuée selon la démarche suivante :

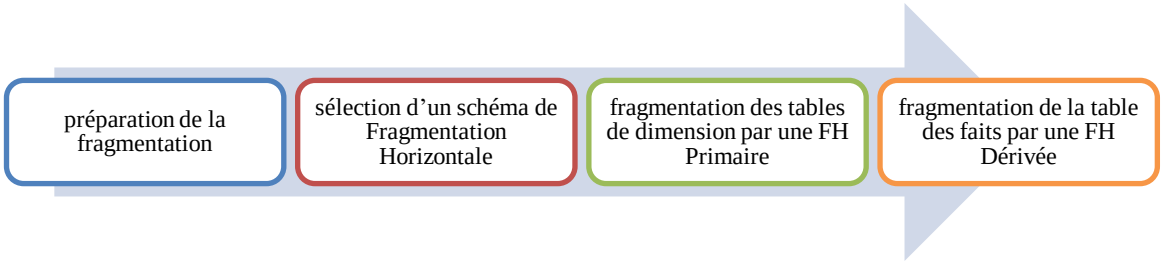


Figure 13 – Démarche de la fragmentation horizontale.

1. Phase de préparation de la fragmentation : [7]

- Sortir la totalité des prédicats de sélection.
- Déterminer les tables candidates pour participer à la FH.
- Générer un ensemble des prédicats complet et minimal pour chaque table candidate.
- Découper le domaine de chaque attribut en sous-domaines.

2. Phase de sélection d'un schéma de Fragmentation Horizontale :

- La génération des schémas de Fragmentation Horizontale.
- L'évaluation de chaque schéma.
- La sélection du meilleur schéma.

3. Phase de fragmentation des tables de dimension par une FH Primaire.

4. Phase de fragmentation de la table des faits par une FH Dérivée.

2.6.2.4 Problème de sélection d'un schéma de fragmentation horizontale

On peut formaliser le problème de sélection d'un schéma de fragmentation horizontale comme suite :

- ***Inputs***
  - Un entrepôt de données ayant un schéma en étoile.

- Une charge de requêtes Q.
- Un seuil W fixé par l'administrateur représentant le nombre maximum de fragments de la table faits dans le schéma final de FH.

- ***Outputs***
  - N fragment de table de fait F1, F2, ...Fn.
- ***Objectifs***
  - Réduction du cout d'exécution de Q.
  - Nombre de fragment N<W.

Le problème de la fragmentation horizontale a comme objectif de obtenir un ensemble de D dimensions fragmenter par une FHP, et un ensemble de N fragments de faits F1, . . .,FN obtenus par une FHD, tels que deux condition doit vérifier : le coût d'exécution des requêtes Q soit minimal, et la contrainte N ≤ W [4].

2.7 Les travaux existants

Il existe plusieurs travaux pour la résolution de problème de génération un plan d'exécution global pour l'optimisation multi-requête dans entrepôt de données. Nous regroupons ces travaux selon les structures d'optimisation illustrée dans la figure ci-dessous.

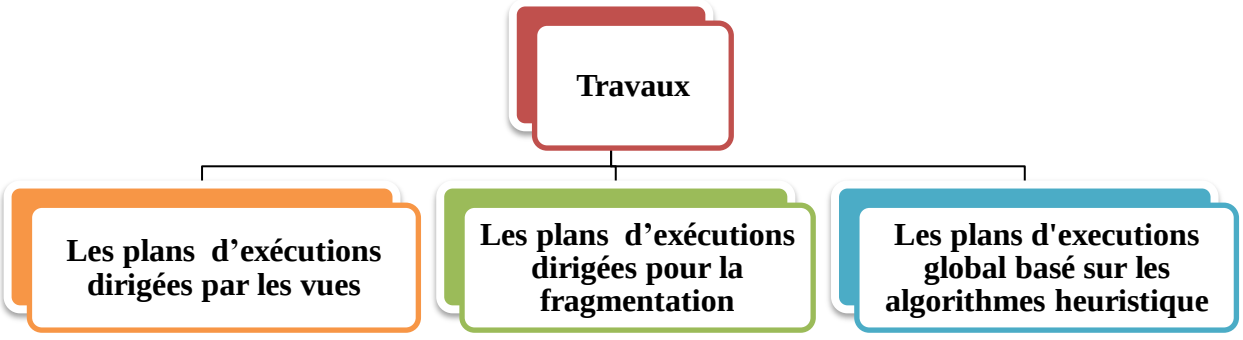


Figure 14 – Les travaux existants.

2.7.1 Les plans d'exécution global basé sur les algorithmes heuristique

Travaux de Sellis et al

On a déjà mentionné que de nombreux algorithmes heuristiques (HA) sont proposés pour améliorer considérablement le temps de génération des plan global, parmi ces algorithmes on trouve l'algorithme de recherche d'espace ***A**** proposé par Sellis et al [18] pour produire un plan d'exécution global basé sur tous les plans locaux possibles de chaque requête. La figure ci-dessous montre les étapes suivies par Sellis et al pour la construction du plan global.

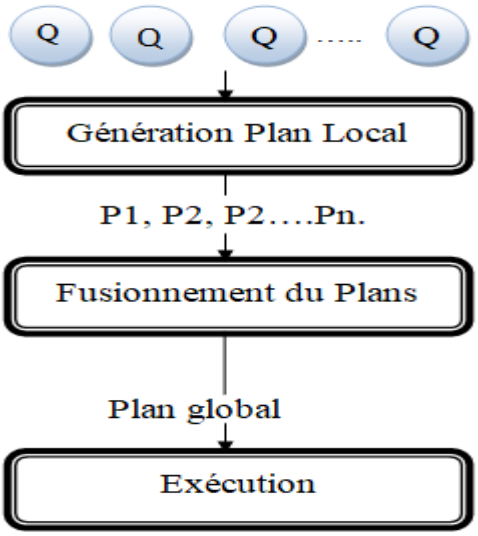


Figure 15 – Les étapes suivies par Sellis et al pour la costruction du plan global [18].

Sellis et al ont débuté par la génération du plan local à l'aide d'un algorithme de programmation dynamique, la deuxième étape *Plan merger* a pour objectif de regrouper l'ensemble des plans locaux en unique plan global, de sorte que les résultats des sous-expressions communes puissent être partagés autant que possible [18].

Ils ont montré que divers algorithmes peuvent être utilisés pour l'optimisation multi requêtes. Des algorithmes plus sophistiqués (tels que HA) peuvent être utilisés pour offrir de meilleurs plans [18].

L'algorithme HA est basé sur la recherche parmi les plans de requêtes locaux et la construction d'un plan d'accès global en choisissant un plan local par requête.

Sellis a modélisé le problème d'optimisation sous la forme d'un problème de recherche d'espace d'états et proposé l'utilisation ***d'un algorithme A ****.

Le principe de l'algorithme A* est de passer de l'état initial Ei vers l'état final Ef tel que le coût du passage d'Ei à Ef soit minimal parmi tous les chemins menant d'Ei à tout état final. Le coût d'un tel chemin est le coût total requis pour le traitement de toutes les n requêtes.

L'algorithme A * est un algorithme de recherche de chemin dans un graphe, C'est l'un des plus efficaces algorithmes de plus court chemin. Il ne donne pas toujours la solution optimale mais il donne très rapidement une bonne solution pour un problème d'optimisation difficile alors il peut donner le bon plan mais pas forcement l'optimal (en fonction de temps d'exécutions).

2.7.2 Plans d'exécution global dirigé par les vues matérialisées

Il existe plusieurs travaux ont été développés pour construire le plan global MVPP et basés sur la structure d'optimisation vue matérialisée VM. Parmi ces travaux ceux de **« yang et al »** et les travaux de **«Boukorca et al ».**

Les travaux de Yang et al. (1997) : Leurs travaux sont présentés dans le contexte des entrepôts de données centralisé, ils ont généré un plan d'exécution global nommé un **MVPP**, ils ont développé un algorithme de sélection des vues matérialisées qui sera détailler par la suite.

Les méthodes proposées pour la génération du MVPP sont basées sur ***la fusion des plans individuels.***

- ***Input :***
  - Une charge de m requêtes Q= {Q1, Q2,…Qm}.
  - Entrepôt de données.

- ***Output:***
  - Choisir un plan pour chaque requête en maximisant la réutilisation résultats intermédiaires.
  - Structure d'optimisation (vues matérialisées).
- ***Objectif :***
  - Réduire le temps d'exécution de Q.

L'algorithme de Yang et al. Procède de la façon suivante :[19][5]

1. La première étape consiste à représenter chaque requête par un arbre algébrique, les auteurs sélectionnent l'arbre optimal (en fonction d'un modèle de coût).
2. Une fois les arbres optimaux identifiés, l'algorithme essaye de trouver des expressions communes entre ces arbres (ou nœud partagé qui sont les jointures).
3. La dernière étape consiste à fusionner les arbres en un seul graphe, appelé plan multiple d'exécution des vues en utilisant les nœuds partagés identifiés.

Ce graphe est utilisé pour rechercher l'ensemble des vues matérialisé pour but de réduire la somme des coûts d'exécutions des requêtes et de maintenance des vues.

Exemple1 : Soit un schéma d'un entrepôt ayant cinq tables et sur lequel cinq requêtes sont définies.

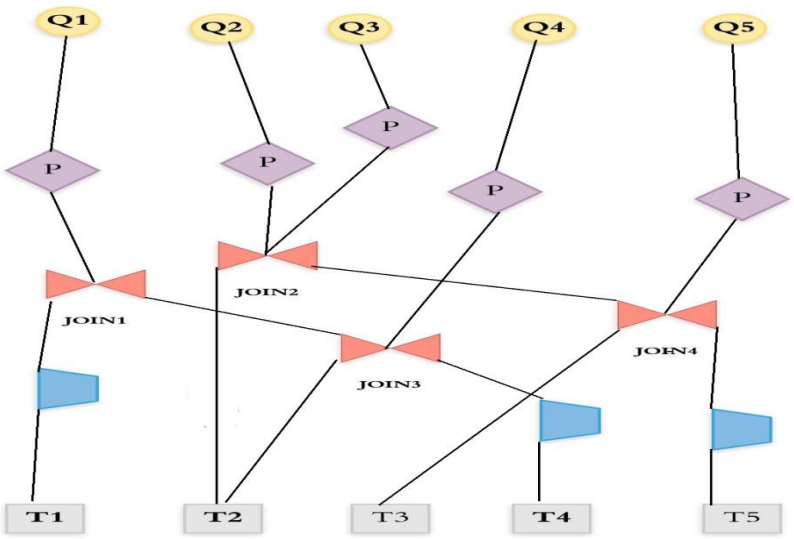


Figure 16 – Principe de base de la sélection de yang et al.

Algorithme « la programmation binaire 0-1 »

L'algorithme passe par les étapes suivantes [8] :

1. Construire pour chaque requête tous les plans possibles *pi*.

2. Identifier tous les sous arbres de jointures possibles *si* pour chaque plan construit.

3. Construction d'une matrice d'usage binaire *A*, où le coefficient *aij* représente la possibilité ou non d'exécution de la requête *qi* par le plan *pj*.

4. Construction d'une matrice binaire *B*, où le coefficient *bij* représente l'appartenance ou non du sous-arbre de jointure *sj* au plan *pi*.

On a finalement$cout_{total} = \sum_{i=1}^{m} \sum_{j=1}^{l} Ecost(s_j) b_{ij}$, où $cost(s_i)$représente le coût de construction de sous-arbres_i.

Les travaux de «Boukorca et al » [8]

Ils ont utilisé des algorithmes et des outils utilisés dans le domaine des circuits électroniques dans le but de garantir le passage à l'échelle, ils ont vu qu'il y'a une similarité entre les circuits électroniques et plan d'exécutions des requêtes.

Leur approche de génération du MVPP d'une charge de requêtes a pour objectif de réordonner les nœuds de jointure entre les différentes requêtes pour maximiser la réutilisation des résultats des nœuds intermédiaires (jointures) pendant de l'exécution des requêtes.

Les étapes pour construire leur MVPP sont :

1. représentation des opérations de jointure par un hypergraphe, appelé hypergraphe de jointure
2. l'algorithme de partitionnement choisi pour fractionner un hypergraphe en plusieurs hypergraphes disjoints.
3. l'algorithme de transformation de l'hypergraphe en un graphe orienté optimal (ayant un coût minimum), qui représente le MVPP des requêtes.

Construction de l'hypergraphe de jointure

Cette étape consiste à extraire tous les nœuds de sélection, de jointure, de projection et d'agrégation selon un ordre déterminé par la forme générale de la requête, et regroupe les nœuds de jointure en plusieurs composantes connexes et ensuite les représenter par un hypergraphe qui contient l'ensemble des sommets (nœuds de jointure), et un ensemble des hyper arêtes qui représente la charge des requêtes.

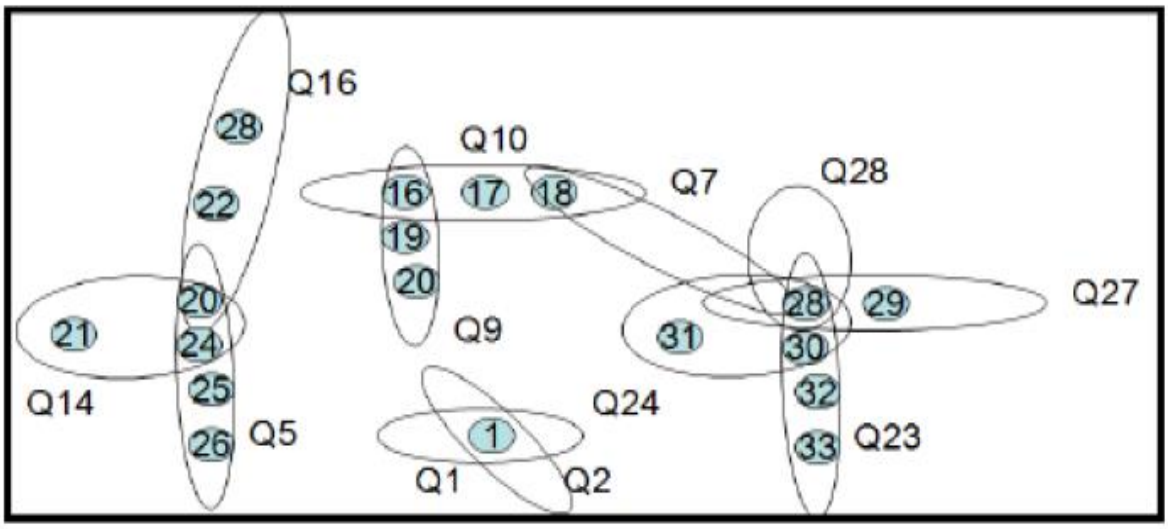


Figure 17 – Exemple d'un Hypergraphe de jointure [8].

Partitionnement de l'hypergraphe

Le principe étant de diviser un hypergraphe en k partitions (nouveaux hypergraphes), de telle sorte que le nombre d'hyper arêtes coupées soit minimal à l'aide de programme de *HMETIS*[1].

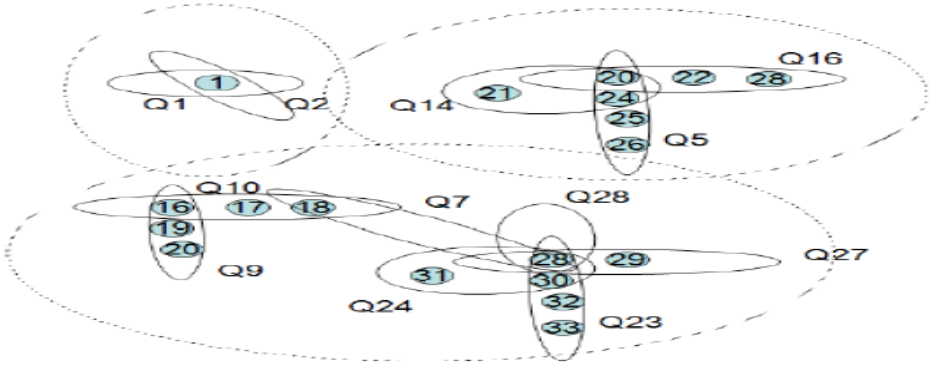


Figure 18– Partitionnement d'un Hypergraphe de jointure [8].

Transformation d'un hypergraphe en graphe

L'étape restante consiste à trouver un ordre entre les nœuds de jointure dans chaque sous hypergraphe pour générer le MVPP global, c'est à dire transformer chaque hypergraphe en un graphe orienté. Dans cette étape ils ont utilisé trois algorithmes le premier *ChercherPivot* a pour objectif de chercher le nœud pivot qui correspondant à la première opération de jointure qui augmenter le bénéfice de réutilisation de son résultat par les autres requêtes. Le deuxième *TransformerhyperGraphe* cet algorithme consiste à transformation de l'hypergraphe en un graphe orienté. Et le dernier algorithme c'est *PartitionnerPivot* cet algorithme permet le fractionnement d'un hypergraphe en deux hypergraphes suivant un nœud pivot et de copier les sommets communs entre les deux hypergraphes.

2.7.3 Plans d'exécution global dirigé pour la fragmentation

Plusieurs travaux se sont intéressés à l'optimisation à grande échelle des entrepôts de données ces travaux ont été basé sur MVPP construit, ils ont exploité le plan globale pour la

[1]http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview

structure d'optimisation fragmentation horizontale dans plate-forme centralisée parmi ces travaux, on trouve **Kerkad et al.**

Les travaux de « kerkad et al »[4]

Dans ces travaux ils proposent une nouvelle approche pour la FH dans les EDR basée sur l'interaction entre les requêtes. Cette interaction est intégrée dans une structure de données incrémentale.

Les étapes suivies dans ces travaux sont :

1) Construction des plans pour chaque requête et les fusionner en un seul graphe MVPP, qui représente graphiquement la charge des requêtes introduit par utilisateur. Notant que les opérations de sélection sont poussées en bas après la construction du MVPP.

2) grouper les requêtes selon les jointures communes : {Q1}; {Q2;Q3}; {Q4;Q7;Q9;Q10}; {Q5;Q6;Q8}, tel que le nombre de groupes obtenu corresponds au nombre de jointures directes avec la table des faits { j1; j2; j3; j4} qui sont les plus couteuse. Ce regroupement fait à la raison de toutes les requêtes dans le même groupe est corrélées au moins dans la première jointure. Si cette première jointure est optimisée par la FH sur ses sélections, toutes les requêtes du même groupe vont y bénéficier.

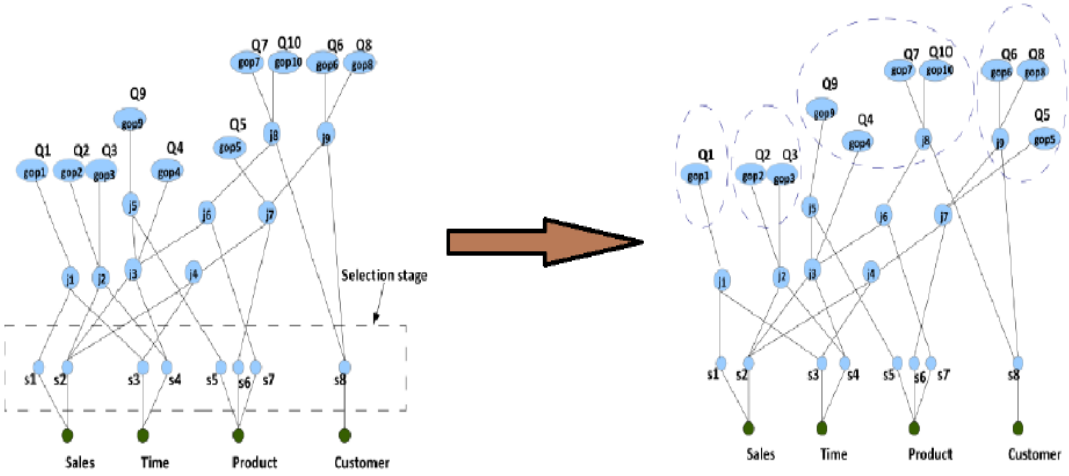


Figure 19 – Exemple de MVPP avec exploitation de l'interaction [4].

3) proposer un algorithme pour ressortir l'interaction entre les requêtes est l'Algorithme de Requête Elue (ARE), l'ARE génère l'ensemble des groupes de requêtes corrélées, tel que chaque couple de requêtes ayant au moins un nœud de jointure commun est dans le même groupe.

L'algorithme élit une requête qui permettra d'aiguiller le processus de FH. Cette requête, que l'on appelle Requête Elue (RE) est considérée comme la plus importante dans son groupe. Tel que la requête ayant le coût minimal est considérée comme l'élue à la raison de la requête ayant un coût minimal contient moins d'opérations de jointure et de sélections.

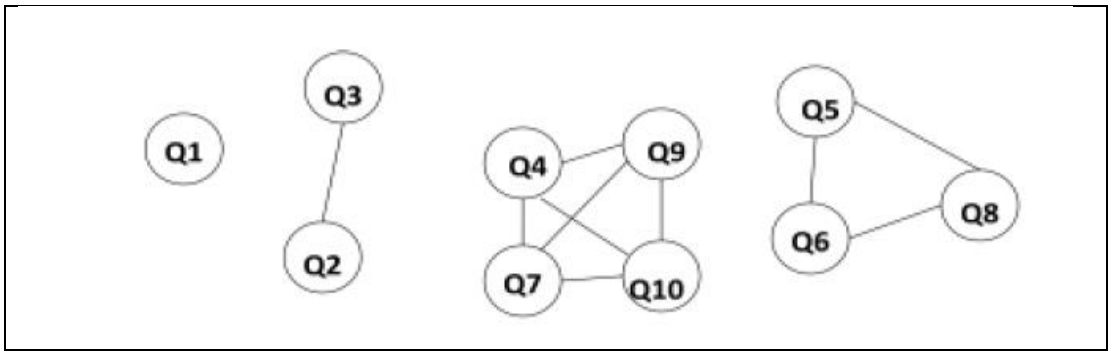


Figure 20 – Groupes de raquêtes obtenus à partir du MVPP [4].

4) propose un codage incrémental qui fait la décomposition d'un domaine d'attribut en sous-domaines à partir du MVPP pour cela elle a utilisé deux fonction : Split verticale et Split horizontal.

Le principe de ce codage est :

1. commencer par un ensemble d'attributs vide
2. pour chaque requête y ajouter ses attributs de sélection en créant de nouveaux vecteurs, chacun contenant un seul champ.
3. A chaque fois qu'une sélection est retrouvée dans la requête encours, l'une des trois opérations est effectuée :
    - Si l'attribut n'existe pas dans le schéma, appliquer un Split Vertical
    - Si l'attribut existe déjà, appliquer le Split Horizontal sur le champ else pour rajouter les nouveaux sous-domaines.

- Finalement, si l'administrateur connaît la valeur restante dans les champs else, il la remplace par cette valeur.

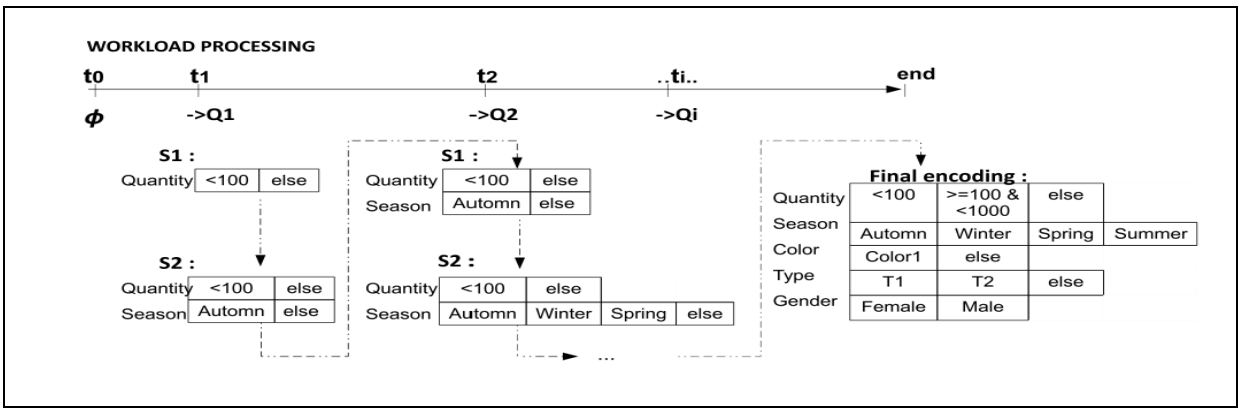


Figure 21– Codage par Split Horizontal et Vertical sur les sous-domaines d'attributs [4].

5) Cette étape consiste à chaque attribut trié subit des opérations de fusion/éclatement (Merge/Split) de la manière suivante :

- On regroupe les sous-domaines qui utilisés par aucune requête élue en une même partition.
- Les sous-domaines les plus utilisés, sont regroupés en une seule partition.

Figure 22– Eclatement et fusion des partitions par corrélation des requêtes [4].

Cette étape permet de créer des partitions en se basant sur les besoins des requêtes les plus importantes.

Si la fragmentation est toujours possible (N < W) tel que N c'est le nombre de fragments dans la table de fait et W nombre de fragments possible, les partitions obtenues sont éclatées selon la corrélation entre les requêtes accédant chaque sous-domaine de la partition.

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

L'ensemble suivant des requêtes est celui des successeurs des RE courantes. Si au moins un groupe contient encore un successeur (parmi les requêtes triées par coût minimal), alors un nouvel ensemble de requêtes élues est généré par les successeurs retrouvés de tous les groupes.

Le même processus est répéter pour élargir le schéma de codage avec les nouveaux attributs et sous-domaines exigé de manière incrémentale. Le processus réitère jusqu'à ce que

N < W ou qu'aucune requête ne reste dans aucun groupe.

2.8 La synthèse

La génération de plan d'exécution global est un problème NP-difficile. Plusieurs travaux ont été développés pour résoudre ce problème avec différent représentation graphique nous somme intéressés dans notre recherche sur les travaux qui construit le plan d'exécution global de type MVPP.

Nous avons vu la principale limitation du travail de Yang et al est la scalabilité de l'algorithme utilisé pour la génération du MVPP car il est impossible de générer tous les plans globaux pour les applications qui contiennent un grand nombre de requêtes. En plus l'algorithme très coûteux en terme de calcul et ne donne pas forcement le MVPP optimal puisque génère tous les MVPP possibles à partir des plans individuels optimaux des requêtes, puis choisit celui qui a le coût minimum.

La méthode de *yang* dans ces travaux de génération de MVPP est une méthode ascendante car a partir des arbres global individuel il a construit son MVPP pour multi

requêtes par contre *Boukorca et al* **[8]** c'est l'inverse ils ont construit en premier le MVPP et les plans individuels des requêtes sont déduits directement**.** *Boukorca et al* dans leur approche ont basé sur les algorithmes de théorie des graphes et *Yang* sur les Algorithme heuristique comme ils ont basé sur une structure d'optimisation(VM) précise pour trouve le MVPP et ce que nous comptons de faire.

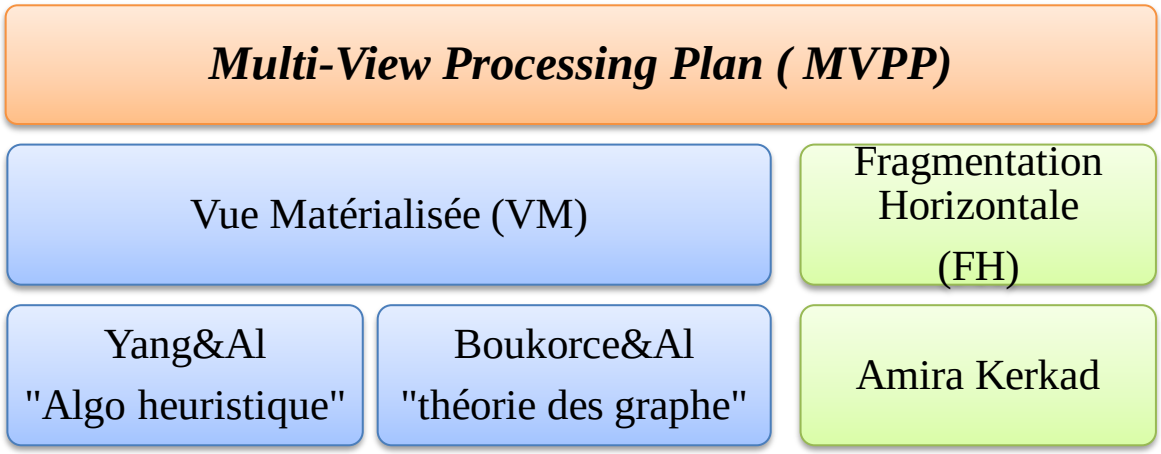


Figure 23– Classification des travaux de MVPP.

2.9 Conclusion

Nous avons présenté à travers ce chapitre le contexte du problème de la construction de plan global qui est utilisé comme une structure de donnée pour la sélection des structures d'optimisation tels que *des vues matérialisées* ou *la fragmentation horizontale*, on a cité quelques travaux qui ont été proposés pour résoudre ce problème et on a terminé par une synthèse qui nous permet de faire la comparaison entre les travaux qui ont été présentés.

Dans le chapitre suivant, nous discutons les détails sur notre approche pour construire notre plan global MVPP.

Chapitre 3

Conception et implémentation

3.1 Introduction

Dans ce chapitre nous proposons une nouvelle approche pour la génération de plan d'exécution global sous forme un MVPP qui sera utilisé pour la sélection des vues matérialisées basant sur les concepts des théories des graphes, plus précisément nous inspirons à partir d'un algorithme qui a été utilisé pour le partitionnement vertical. Le principe de cette nouvelle démarche sera présenté par la suite.

3.2 L'approche proposée

Dans cette approche, nous avons basé sur l'utilisation d'un algorithme des théories des graphes afin d'optimiser la charge des requêtes. Au but de partitionner cette charge des requêtes en des groupes disjoints suivant l'opérateur de jointure.

Notre approche est une variante de l'algorithme Navathe et al [16] quia été utilisé pour définir les partitions verticales des tables.

3.2.1 Principe de L'algorithme Navathe

L'algorithme [16] passe par cinq étapes qui sont illustrés dans la figure suivante :

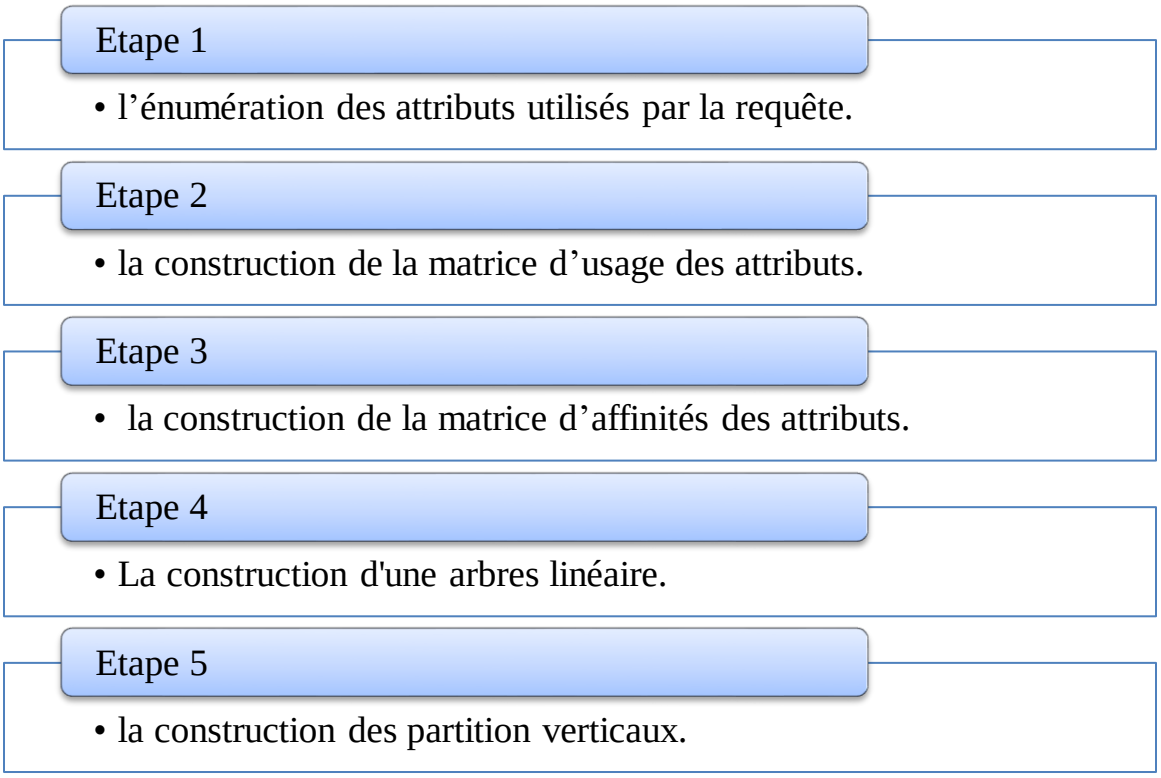


Figure 24– Les étapes de l'algorithme Navathe.

3.2.2 Définition et Concepts fondamentaux

- **Matrice d'Usage des requêtes**

Cette matrice d'usage ***Mu*** modélise l'utilisation des jointures par des requêtes de Q. Possède *L* lignes (nombre de requêtes) et *NJ* (nombre de jointure) colonnes.

- Mu[*l*][*i*] = 1 si la requête Q contient le nœud de jointure nj.
- Sinon Mu[*l*][*i*] = 0.

On trouve une ligne *supplémentaire* est ajoutée à ***Mu*** pour représenter la fréquence d'accès de chaque requête de Q.

- **Matrice d'Affinités des requêtes**

La matrice d'affinité ***Ma*** modélise l'*affinité* entre deux requêtes *Qi* et *Qi'*. Cette matrice est générée de la même manière que la matrice d'affinité des attributs de la fragmentation verticale [16]. C'est une matrice carrée (*Q** Q) symétrique dont les lignes et les colonnes représentent les requêtes.

La matrice d'affinité se définie comme suite dans notre cas:

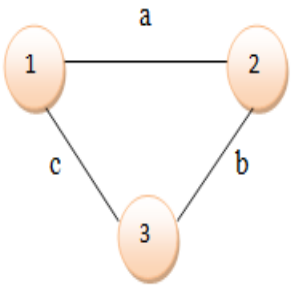
La valeur de chaque élément (i, i') tel que (1<=i, i'<=N) : égale à la somme des Access de toutes les jointures utilisés par les même requêtes.

- **Arbres linéaire**

Un arbre linéaire est un arbre connecté n'a que deux extrémités.

- **Cycle primitif**

Tout cycle du graphe d'affinité un cycle primitif.



Par exemple, supposons que les arêtes a et b aient déjà sélectionnées et que c soit sélectionné ensuite. Donc les arêtes a, b, c forment un cycle primitif.

- **Cycle d'affinité**

Est un cycle primitif qui contient un nœud de cycle ou aucune ancienne arête n'existe. S'il existe une ancienne arête alors il faut que la valeur d'affinité de cette arête inférieure ou égale à la valeur d'affinité minimale du cycle.

- nœud de cycle est ce nœud d'arête de fin de cycle sélectionné précédemment.
- ancienne arête désigne une arête sélectionnée entre la dernière rupture et le nœud de cycle.

3.2.3 Les étapes de partitionnement

- ***Etape 1***

L'algorithme part de la matrice d'affinité(QQ), générée à partir d'Extraction des nœuds de jointure pour chaque requête (usage matrix).

- ***Etape 2***

Commencez depuis n'importe quel Requête(Q), dans notre approche on a commencé par la première requête (Q1).

- ***Étape 3***. Sélectionnez une arête répondant aux conditions suivantes:
  - Il devrait être connecté linéairement à l'arbre déjà construit.
  - Il devrait avoir la plus grande valeur parmi les choix possibles d'arêtes à chaque extrémité de l'arbre.

Cette itération prendra fin lorsque tous les nœuds seront utilisés pour la construction de l'arborescence.

- ***Étape 4***. Lorsque le prochain bord sélectionné forme un cycle primitif:

Si n'existe pas un nœud de cycle, vérifiez la «possibilité d'un cycle» et, le cas échéant, marquez le cycle comme un cycle d'affinité. Considérez ce cycle comme une partition candidate. Passez à l'étape 3.

Si un nœud de cycle existe déjà, ignorez ce bord et passez à l'étape 3.

- ***Étape 5***. Lorsque le prochain bord sélectionné ne forme pas de cycle et qu'une partition candidate existe:

(1) Si aucun ancien bord n'existe, vérifiez la possibilité d'extension du cycle par ce nouveau bord. S'il n'y a pas de possibilité, coupez ce bord et considérez le cycle comme une partition. Passez à l'étape 3.

(2) S'il existe une ancienne arête, changez le nœud du cycle et vérifiez la possibilité d'extension du cycle par l'ancienne arête. S'il n'y a pas de possibilité, coupez l'ancien bord et considérez le cycle comme une partition. Passez à l'étape 3.

3.2.4 Analogie entre l'algorithme et notre approche

Nous avons trouvé une analogie entre l'algorithme de partitionnement vertical ***Navathe*** et notre approche, cette analogie illustré dans le tableau ci-dessous.

Tableau 2– Analogie entre algorithme et l'approche

Analogies	
Nœuds : Attributs.	Nœuds : Requêtes.
Cycle.	Ensemble des requêtes connecté.
Ensemble des partitions.	Ensemble des plan d'exécution.

3.2.5 Exemple de motivation

En prenant un exemple de 8 requêtes issues de benchmark SSB [14]:

0: select sum(lo_extendedprice*lo_discount) as revenue from lineorder, dates where lo_orderdate = d_datekey and d_year = 1993 and lo_discount>= 1 and lo_discount<= 3 and lo_quantity< 25.

1 : select count(*) fromlineorder, dates wherelo_orderdate = d_datekey and d_year = 1993 and lo_discount>= 1 and lo_discount<= 3 and lo_quantity< 25.

2 : select sum(lo_extendedprice*lo_discount) as revenue fromlineorder, dates wherelo_orderdate = d_datekey and d_year = 1993.

3 : select count(*) fromlineorder, dates wherelo_orderdate = d_datekey and d_year = 1993

4 : select sum(lo_revenue), d_yearfromlineorder, dates, part, supplier wherelo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'ASIA' group by d_yearorder by d_year.

5 : select sum(lo_revenue) fromlineorder, part wherelo_partkey = p_partkey and p_brand = 'MFGR#2221'.

6 : select avg(lo_revenue), d_yearfromlineorder, dates, part, supplier wherelo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'ASIA' group by d_yearorder by d_year.

7 : select sum(lo_revenue), d_yearfromlineorder, dates, part, supplier wherelo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'EUROPE' group by d_year, p_brandorder by d_year, p_brand.

- **L'analyse de la charge de requêtes**

Dans cette étape nous avons fait une analyse sur ce groupe de requêtes par l'extraction des tables de la base de données et l'extraction des jointures pour chaque requête.

1. **La 1ér partie : l'extraction des tables de base**

A partir de cette charge de requête, nous avons extrait premièrement toutes les tables de ces requêtes.

- **Les tables :**
  - LINEORDER
  - DATES
  - PART
  - SUPPLIER

2. **La 2éme partie : l'extraction des jointures**

Nous avons extrait aussi toutes les jointures de ces requêtes et chaque jointure est codé par un numéro (10, 11, 12, 13, 14, 15) respectivement.

- **Les jointures :**
  - 10 - LO_ORDERDATE = D_DATEKEY.
  - 11 - LO_ORDERDATE = D_DATEKEY.
  - 12 -LO_ORDERDATE = D_DATEKEY.
  - 13 -LO_PARTKEY = P_PARTKEY.
  - 14 -LO_SUPPKEY = S_SUPPKEY.
  - 15 - LO_SUPPKEY = S_SUPPKEY.

- **Construction de la Matrice d'Usage des requêtes**

A partir de cette analyse on obtient la matrice d'usage illustré dans le tableau ci-dessous.

Tableau 3– Matrice d'Usage des Requêtes

	10	11	12	13	14	15
Q0	1	0	0	0	0	0
Q1	1	0	0	0	0	0
Q2	0	1	0	0	0	0
Q3	0	1	0	0	0	0
Q4	0	0	1	1	1	0
Q5	0	0	0	1	0	0
Q6	0	0	1	1	1	0
Q7	0	0	1	1	0	1
access	25	50	25	31	25	25

- **Construction de la Matrice d'Affinités des requêtes**

A partir de la matrice d'usage nous avons construit la matrice d'affinité. Le tableau ci-dessous représente matrice d'affinité de notre exemple :

Tableau 4– Matrice d'Affinité des Requêtes.

	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Q0	25	25	0	0	0	0	0	0
Q1	25	25	0	0	0	0	0	0
Q2	0	0	50	50	0	0	0	0
Q3	0	0	50	50	0	0	0	0
Q4	0	0	0	0	81	31	81	56
Q5	0	0	0	0	31	31	31	31
Q6	0	0	0	0	81	31	81	56
Q7	0	0	0	0	56	31	56	81

○ Construction du graphe

Dans cette étape, nous représentons la matrice d'affinité par un graphe.

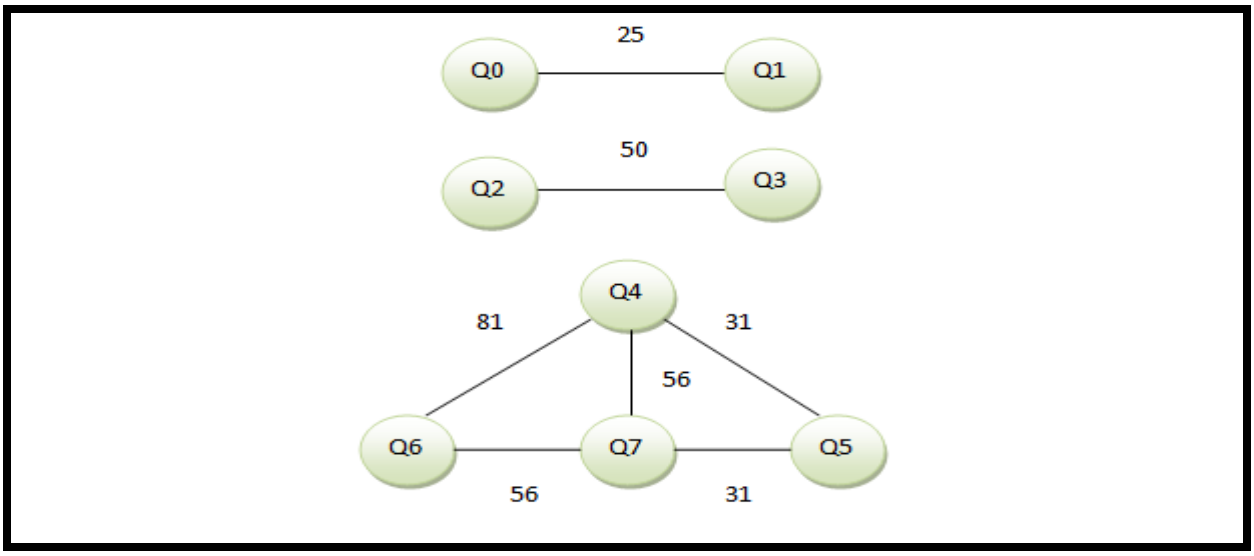


Figure 25– Graphe d'affinité.

○ Construction des partitions

Dans cette étape, nous regroupons les requêtes dans des partitions C, selon les jointures communes entre eux, c'est-à-dire les requêtes Q*i*, *Qi*' sont utilisés en même temps la même jointure. Et le résultat

- C0= {Q0, Q1}, avec jointure Nj = {J10}.
- C1= {Q4, Q6, Q7}, avec ensemble des jointures Nj= {J12, J13, J14, 15}
- C2= {Q5}, avec jointure Nj = {J13}
- C3= {Q2, Q3}, avec jointure Nj = {J11}.

La figure ci-dessous représente les partitions de notre exemple.

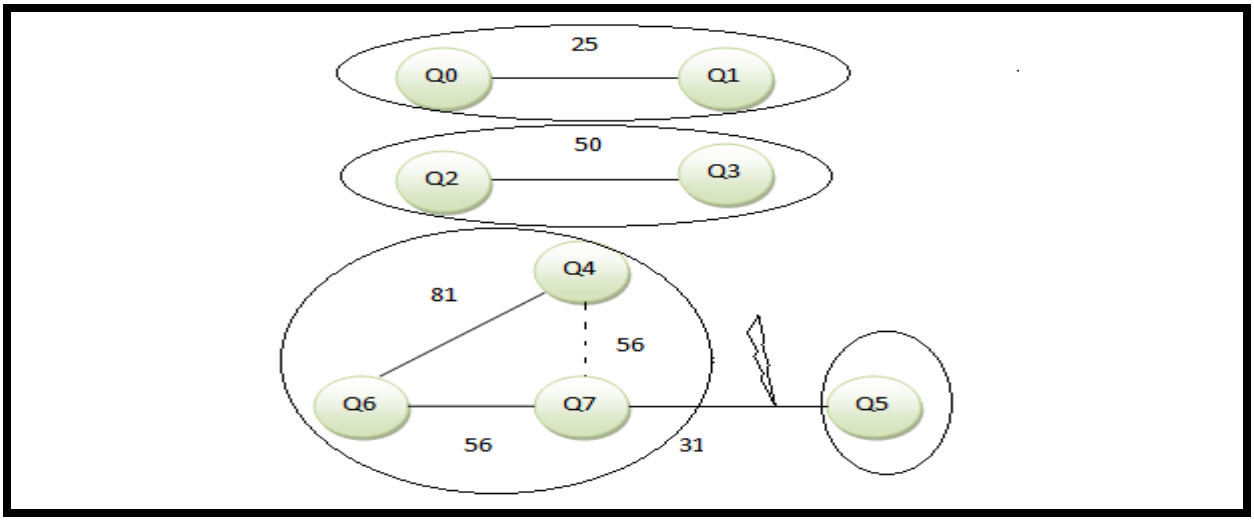


Figure 26– Graphe de partitionnement.

Nous avons remarqué qu'il y a des jointures qui se répètent dans plusieurs partitions (Jointure numéro 13), pour cela nous avons appliqué l'opération d'intersection entre les partitions pour extraire les nœuds communs. Ces nœuds des jointures sont appelés les nœuds de fusion et qui seront utilisé pour la matérialisation.

L'opération d'intersection consiste à fusionner des partitions suivant la jointure commune entre eux, donc si on applique l'intersection sur notre exemple on obtient ces nouvelles partitions :

- C0= {Q0, Q1}, avec jointure Nj = {J10}.
- C3= {Q2, Q3}, avec jointure Nj = {J11}.
- C1 ∩ C2= {Q4, Q5, Q6, Q7}, avec ensemble des jointures Nj= {J12, J13, J14, J15}

Cet opération permet de minimiser le coût d'exécution des requêtes, ce qui est un des nos objectifs.

3.3 Implémentation et conception

3.3.1 Outils d'implémentation

Dans l'implémentation de notre application, nous avons utilisé le langage de programmation et outils tels que java, éclipse et Oracle SQL* Plus pour gérer notre entrepôt de données.

- **Java :** C'est l'un des langages de programmation les plus populaires au monde, fonctionne sur différentes plateformes (Windows, Mac, Linux, etc.), créé en 1995.

  Il est utilisé pour: (Applications mobiles, Applications de bureau, des applications Web, Jeux, Connexion à la base de données).

- **Eclipse : Eclipse IDE** est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

- **Oracle SQL* Plus :** est un outil de requête interactif, permettant de se connecter à une base de données Oracle. Il permet aux utilisateurs de saisir et d'exécuter des commandes SQL, des procédures PL / SQL. Décliné en plusieurs versions (graphique et web), il est principalement distribué avec le produit Oracle Data base.

  Oracle a été le premier SGBD commercialisé sur le marché .Nous avons choisi *ORACLE 10g* comme SGBD. A cause de son efficacité (sécurité, volume de données… etc.).

3.3.2 Notre base de données

Le Star Schéma Benchmark (SSB) est conçu pour mesurer la performance des produits de base de données à l'appui des applications d'entreposage de données classiques, et est basé sur le benchmark TPC-H. SSB comporte une table des faits LINEORDERS qui résulte de la fusion des tables LINEITEM and ORDERS et quatre tables de dimension PART, SUPPLIER, CUSTOMER et DATE [14]. Son schéma logique est illustré dans la figure ci-dessous.

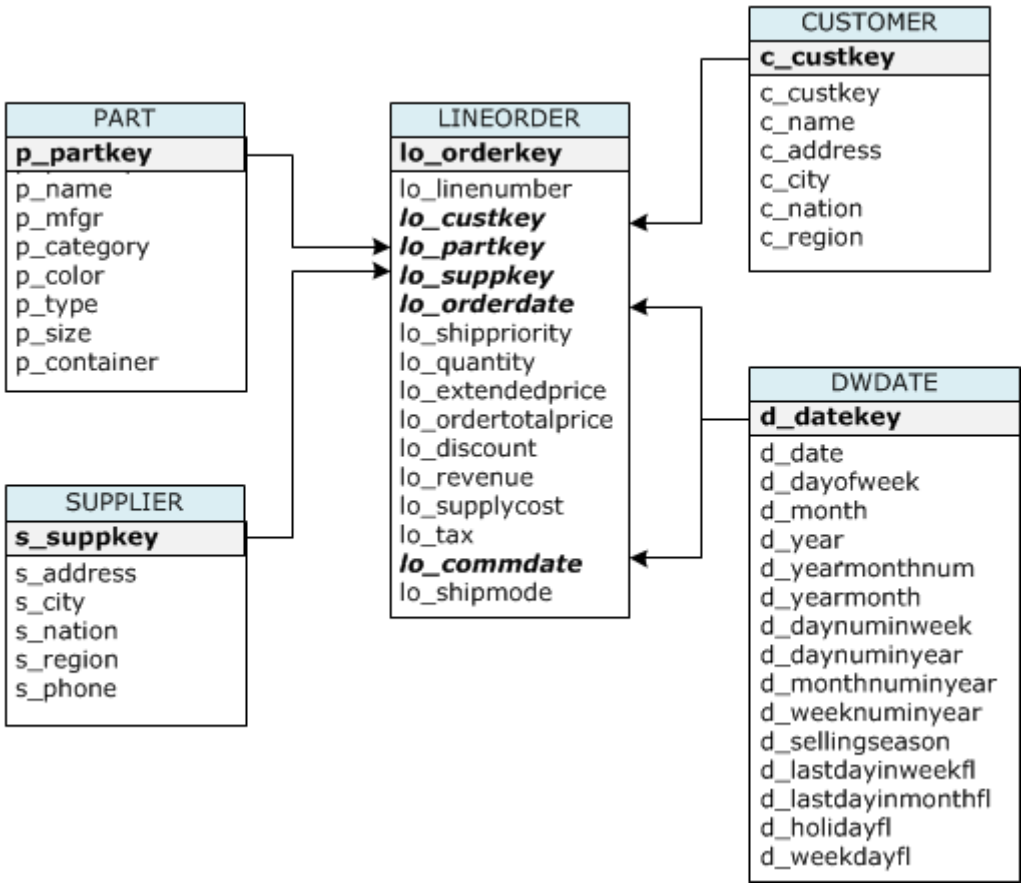


Figure 27– Schéma logique du SSB.

3.3.3 Présentation de l'Application

- **L'interface de connexion de la base de données**

Cette interface permet de faire la connexion sur notre base de données.

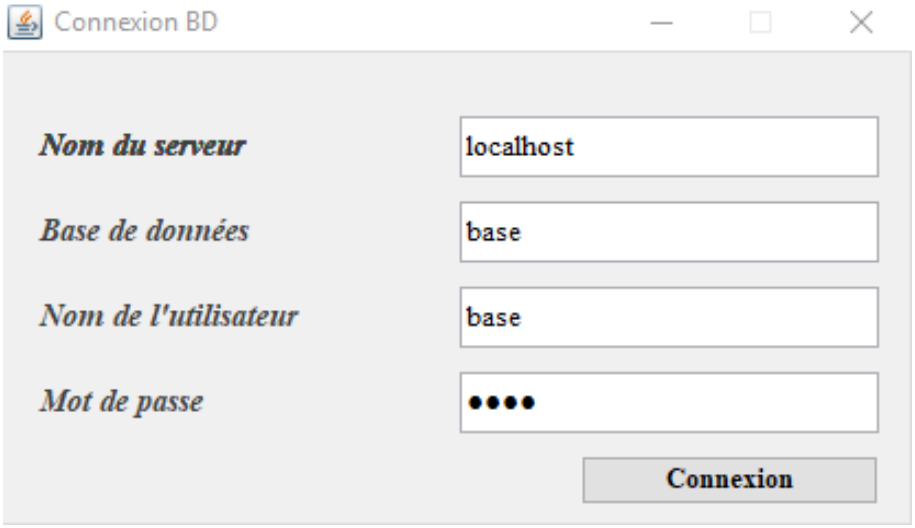


Figure 28– Interface de connexion à la base de données.

- **Chargement des requêtes**

Cette interface permet d'afficher la charge de 8 requêtes de notre base de données, si on clique sur le bouton charger Q, sachant que nous avons travaillé sur 10000 requêtes.

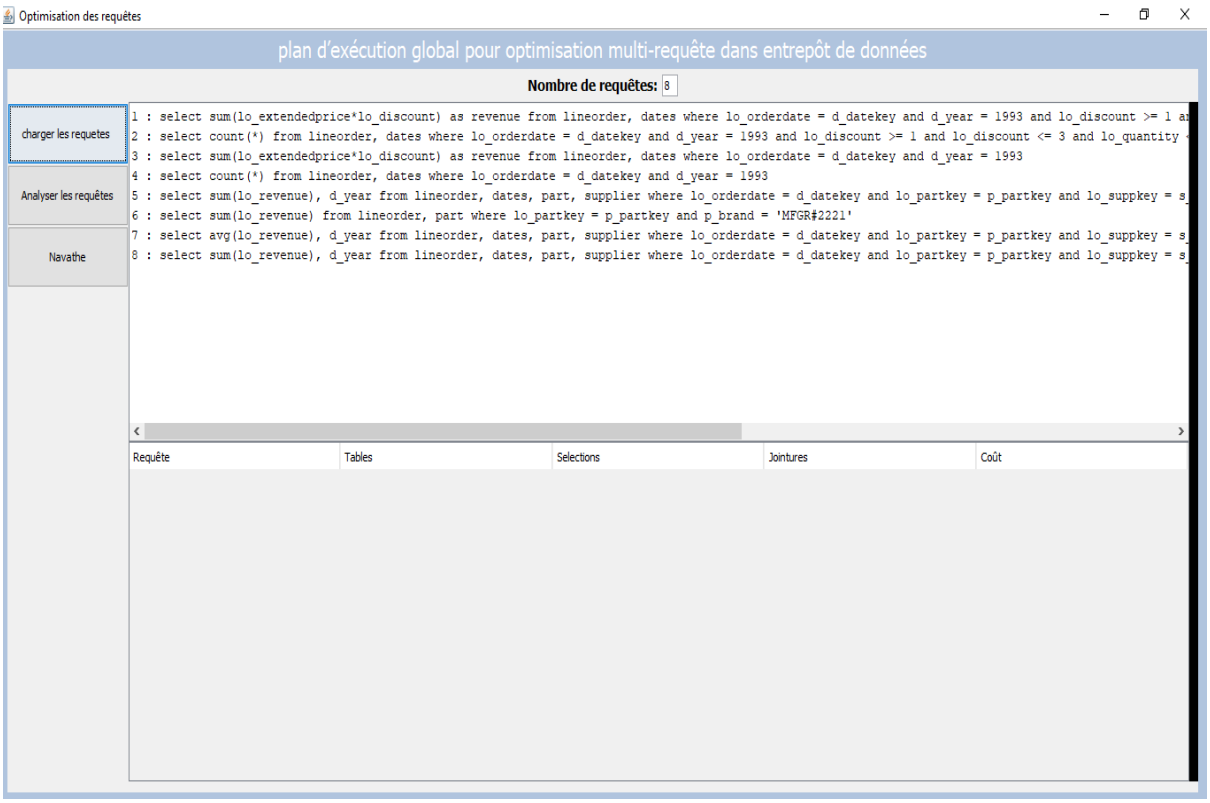


Figure 29– Chargement des requêtes.

- **Analyse des requêtes**

Dans cette interface on analyse la charge des requêtes de l'interface précédente.

- extraire les tables.
- extraire les sélections.
- extraire les jointures.

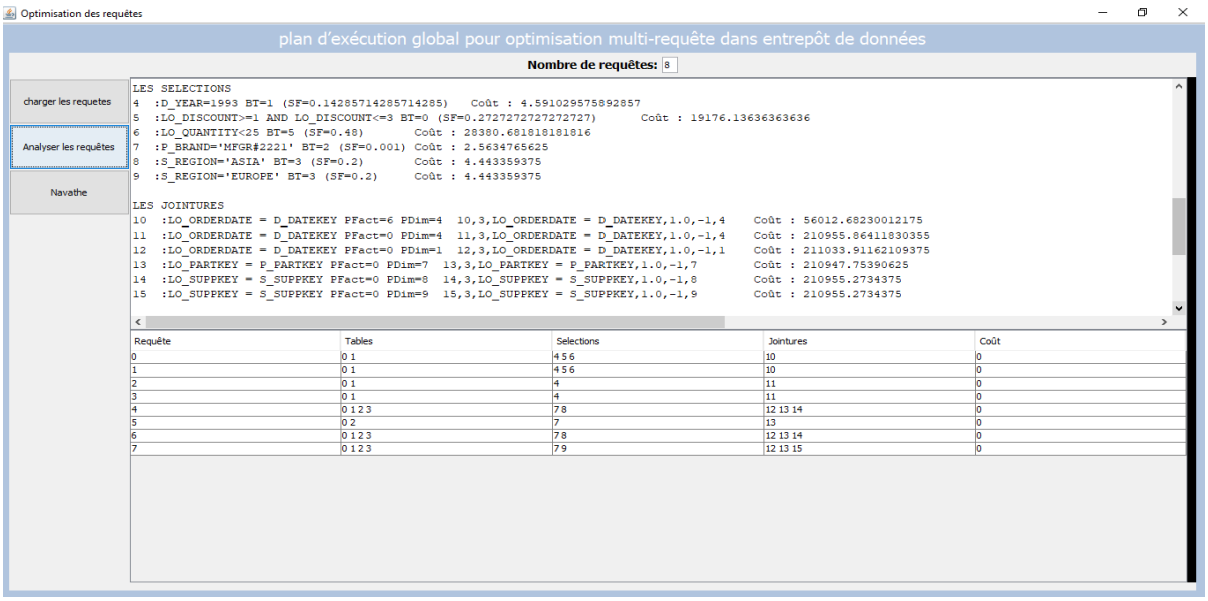


Figure 30– Analyse des requêtes.

- **Application de l'algorithme**

Suit a l'étape d'analyse des requêtes, nous appliquons l'algorithme de partitionnement Navathe. Cette étape permet de partitionner les requêtes en des groupes disjoints suivant l'opérateur de jointure.Cette figure montre l'interface d'application de l'algorithme.

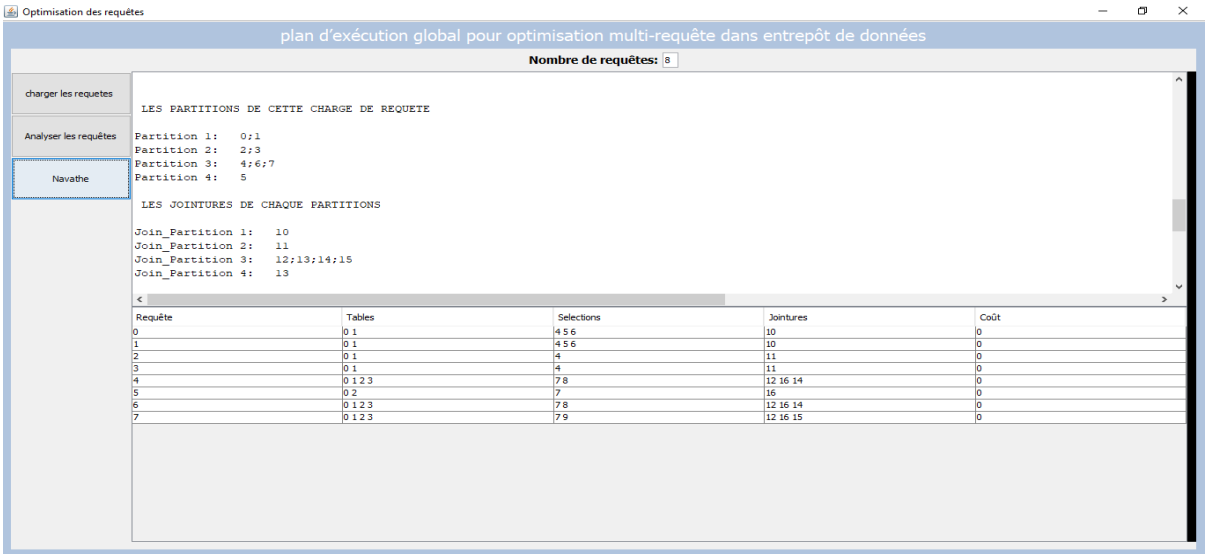


Figure 31 – Application de l'algorithme Navathe.

- **Intersection des partitions**

Cette interface représente l'opération d'intersection. Les résultats obtenus après cette opération sont des nouvelles partitions qui ont un nœud de jointure en commun (nœud fusion).

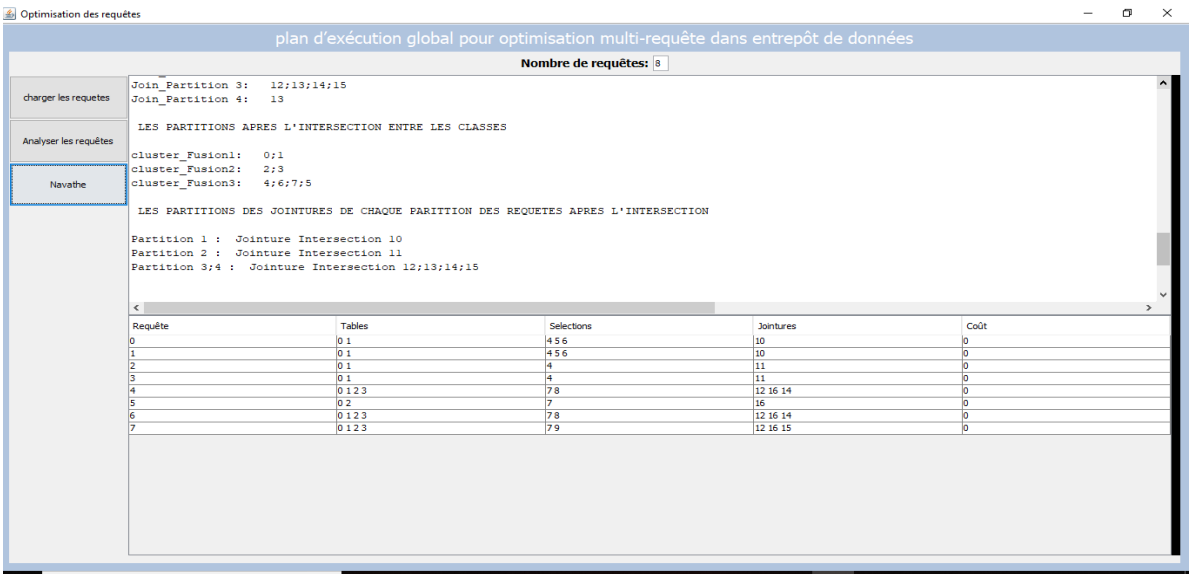


Figure 32 – L'intersection des partitions.

3.4 Le schéma MVPP (Multiple View Processing Plan)

Le MVPP est une représentation graphique de la charge des requêtes introduit par utilisateur, ce plan est composé de plusieurs niveaux tel que le premier niveau représente les tables de base de l'entrepôt, le deuxième niveau représente les sélections, le troisième niveau contient les opérations du jointure et le quatrième niveau représente les projections.

La figure ci-dessous représente le schéma MVPP corresponds a l'exemple de 8 requêtes dessiner par le logiciel Cytoscape[2] .

[2]https://cytoscape.org/

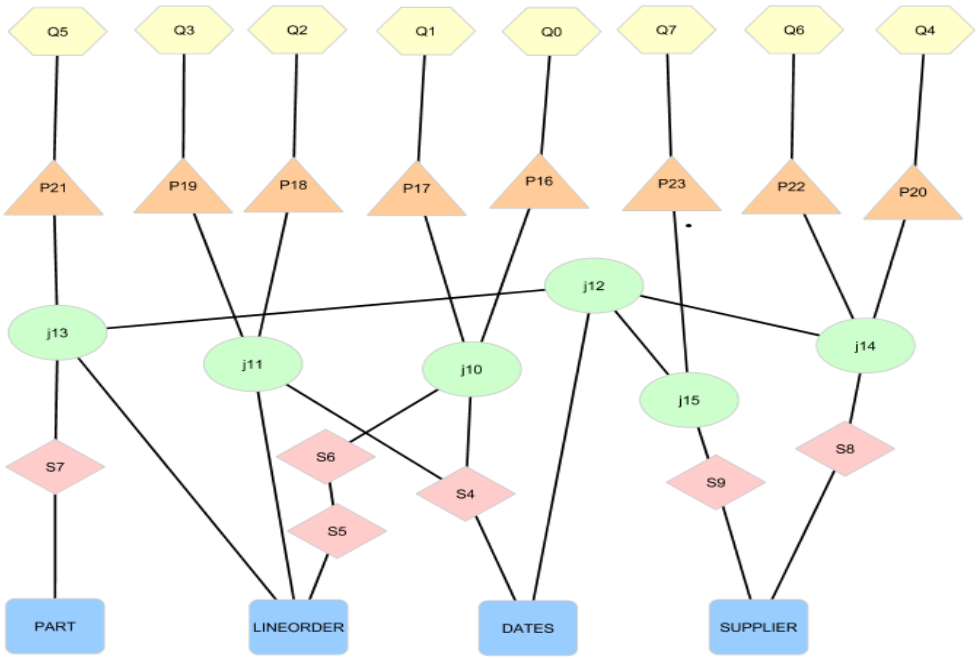


Figure 33 – Le schéma MVPP de 8 requêtes.

3.5 Évaluation et analyse expérimentales

Cette section présente les résultats d'une évaluation expérimentale de notre approche proposée.

Nous avons comparé le temps de génération de MVPP, avec les résultats des travaux présentés dans le contexte génération un plan d'exécution global pour optimisation multi-requêtes. Le tableau ci-dessous représente la comparaison du temps d'exécution de notre approche avec les travaux étudier de yang et de bockourca.

Tableau 5– La comparaison entre les approches.

Nbre de requêtes	20	30	400	600	800	1000	2000	3000	4000
Notre Appr (ms)	86	125	5741	11224	17227	16362	97142	200853	282307
Bokoureca (ms)	88	104	500	900	1980	4593	24749	22479	366839
Yang (ms)	82	514	23600	30654	50000	700000	3600000		

La figure ci-dessous représente l'analyse graphique correspond à la comparaison précédente.

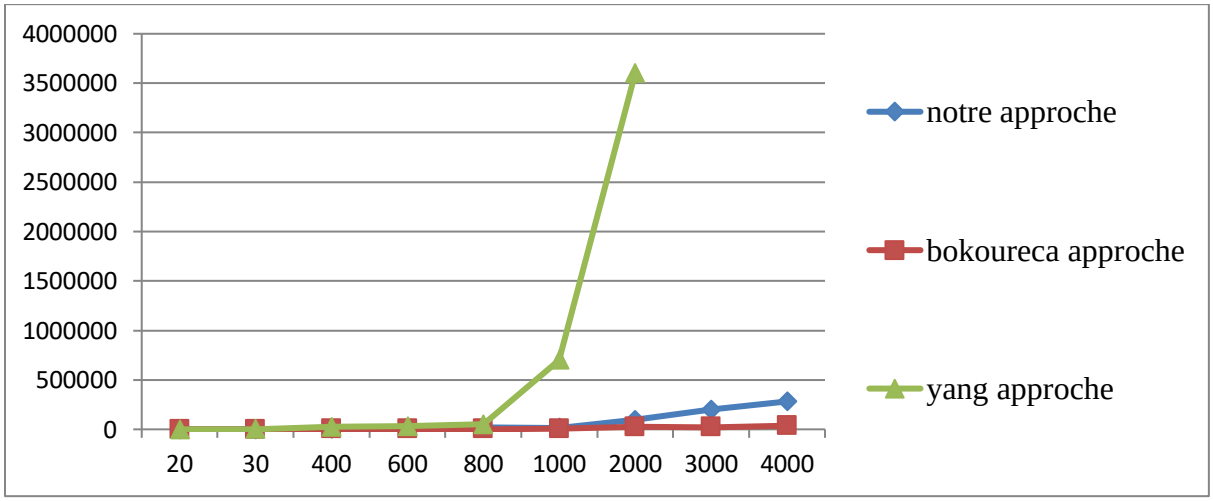


Figure 343 – La comparaison graphique entre les approches.

3.6 Conclusion

Dans ce chapitre, nous avons présenté la partie conception et l'implémentation de notre projet de fin d'étude, nous avons appliqué une nouvelle approche pour résoudre le problème de construction de plan d'exécution global sous forme un ***MVPP*** basant sur l'algorithme de partitionnement N***avathe.***

On a commencé par une présentation de notre approche pour résoudre notre problème ensuite on a choisi un petit exemple pour notre application, et enfin, on a présenté notre application et ses interfaces principales.

Conclusion Générale

L'accès efficace à la donnée a toujours été un enjeu important dans le monde des bases de données. Cette accès permet de manipuler de très importants volumes de données stockées dans des entrepôts de données et causait la complexité des requêtes et la performance. La problématique suggérée dans ce domaine comment générer un plan d'exécution global pour sélectionner des structures d'optimisation afin d'évaluer les performances et répondre rapidement au large et complexe nombre des requêtes .Ces requêtes sont connu par le phénomène « bigquery ».

Nous avons traité dans ce projet le problème de génération d'un plan d'exécution global pour optimisation multi-requêtes, où nous présentons une nouvelle vision afin de résoudre ce problème. La méthode que nous proposons s'appuie essentiellement sur un algorithme basé à la théorie des graphes utilisé dans le contexte de structure d'optimisation redondante pour les partitions verticales.

L'approche proposée est une fonction de combinaison des requêtes visant à optimiser les performances globales d'une charge de travail de requête. La génération Multi-View Processing Plan MVPP avec la méthode que nous avons proposé permet d'avoir une configuration pour l'optimisation multi requêtes et elle va faciliter la sélection des structures d'optimisations dans notre cas les vues matérialisées. Elle permet également le regroupement des requêtes en plusieurs partitions, ce qui va faciliter l'ordonnancement de ces requêtes.

Dans notre application, nous avons comparé le temps de génération de MVPP, avec les résultats des travaux présentés dans le contexte génération un plan d'exécution global pour optimisation multi-requêtes.

Perspectifs

Nous avons développé une méthodologie statique de génération La génération Multi-View Processing Plan MVPP, supposons que la charge de requête n'évolue pas. Dans ce travail, nous n'avons pas pris en considération l'évolution de la charge de requêtes.

Ce travail utilise une technique d'optimisation par les vues matérialisées qui est rendre disponible les résultats intermédiaires pouvant être utilisés par des requêtes complexes, réduisant ainsi la complexité et le coût d'exécution de ces requêtes. Intuitivement, il serait souhaitable d'utiliser technique d'optimisation non redondante comme la fragmentation horizontale permettant une meilleure adéquation au problème à traiter.

Bibliographie

[1] Aljanaby Alaa, Emad Abuelrub, *a Survey of Distributed Query Optimization, The International Arab Journal of Information Technology*, la Jordanie, 2005, pp 48-57.

[2] Amara et Benouaz *Techniques Data Mining pour la sélection d'une configuration d'index de jointure binaire, Architecture d'un entrepôt de donnée* Mémoire Pour l'obtention du diplôme de master en informatique, années universitaire 2010/2011.

[3] Amine ROUKH, *Prise en compte de l'énergie dans la phase d'exploitation des bases de données volumineuses*, Thèse de doctorat, Université de Mostaganem Algérie, 2017

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

[5] Bellatreche Ladjel, *Techniques d'optimisation des requêtes dans les data warehouses, 6th International Symposium on Programming and Systems (ISPS 03),* Algérie, 2003, pp. 81-98.

[6] Benkrid soumia, *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données: application aux plateformes parallèles*, Thèse de doctorat, ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechnique-Poitiers France, 2014.

[7] Boukhalfa Kamel, *De la conception physique aux outils d'administration et de tuning des entrepôts de données*, Thèse de doctorat, Université de Poitiers France, 2009.

[8] Boukorca Ahcène, *Sonic: Scalable multi-query optimization through integrated*

circuits, Database and Expert Systems Applications, Springer Berlin Heidelberg, 2013, pp 278-292.

[9] Daâs Abdelaziz, *Optimisation des requêtes dans le data werehouse*, mémoire magister école doctorale d'informatique (STIC), Stif Algérie, 2018

[10] Gupta Himanshu, *Selection and maintenance of views in a data warehouse*, Thèse de doctorat, université de Stanford San Francisco, 1999.

[11] Lamia SADEG, *Une approche pour l'optimisation des requêtes dirigée par la réutilisation des plans d'exécution*, Laboratoire mathématiques appliquées Ecole Militaire Polytechnique Bordj El Bahri, Algérie, 2009

[12] MAHMOUDI Nesrine, BABA AHMED Farah, *Conception et exploitation d'un entrepôt de donnée au sein de la banque BADR*, Pour l'obtention du diplôme de Master en Informatique Université Abou BakrBelkaid– Tlemcen2016.

[13] Olivier Guibert Cours *Bases de Données*, l'Institut Universitaire de Technologie de l'Université Bordeaux 1, Version 2 du 16 janvier 2018.

[14] O'Neil Patrick, Elizabeth O'Neil, *The star schema benchmark (SSB)*, article, boston, 2007.

[15] Selma khouri, *cycle de vie sémantique de conception de systèmes de stockage et manipulation de données*, Thèse de doctorat, ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechnique-Poitiers France, 2013.

[16] S. Navathe and M. Ra. *Vertical partitioning for database design : a graphical algorithm.1989 ACM SIGMOD* International Conference on Management of *Data*, pp 440–450.

[17] Tianxiao Liu, *Proposition d'un cadre générique d'optimisation de requêtes dans les environnements hétérogènes et répartis*, Base de données [cs.DB]. Université de Cergy Pontoise, 2011. Français.

[18] Timos k.Sellis *Multiple-Query Optimization* University of California, Berkeley, ACM Transactions on Database Systems March 1988, pp 23-52.

[19] Yang Jian, Kamalakar Karlapalem, Qing Li « *Algorithms for materialized view design in data warehousing environment, Proceedings of the 23 rd International Conference on Very Large Data bases (VLDB)*, Athènes Grèce, 1997, pp 136-145.