Table des matières

Liste des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction

Sommaire

1.1 Introduction

De nos jours, les entreprises évoluent dans un environnement concurrentiel, à croissance rapide, caractérisé particulièrement par un grand développement des technologies de l'information et de la communication. Dans un tel environnement, les entreprises font face à un gros flux d'informations. L'exploitation de ce flux d'informations est devenue une action très importante, que chaque entreprise doit entreprendre pour mieux améliorer ses performances, et assurer son existence ainsi que sa prospérité économique. Pour atteindre ces objectifs, les entreprises modélisent ce flux d'informations par l'usage des systèmes d'information. Ces derniers, collectent, traitent, gèrent, stockent, et diffusent les informations au sein de l'organisation pour accomplir des tâches opérationnelles bien définies. Les systèmes d'informations de gestion sont conçus pour exécuter des processus OLTP (On-Line Transaction Processing) sur des données opérationnelles. Les processus OLTP effectuent des opérations transactionnelles simples sur un petit nombre d'enregistrements de données tel que : la consultation des données, l'ajout, la modification, la suppression,...etc., dans le but d'assurer le bon fonctionnement journalier de l'entreprise. Cependant, les systèmes OLTP ont un support limité aux fonctions d'aide à la prise de décisions. Ainsi, pour réussir économiquement, les entreprises ont besoin, en plus des systèmes d'information de gestion, des systèmes d'information permettant aux managers et aux analystes de prendre des bonnes décisions au temps opportun pour répondre à des problèmes et profiter des différentes opportunités. Ces systèmes sont dotés de techniques d'analyse appliquées sur des informations consolidées et historisées, ils sont connus sous le nom de : Systèmes d'Aide à la Décision (Decision Support Systems). L'une des composantes importante des systèmes d'aide à la décision, est l'entrepôt de données. Ce dernier, constitue le noyau de l'architecture décisionnelle. Il fournit des données réconciliées, intégrées, et unifiées, extraites à partir des bases de données opérationnelles des différents systèmes d'information de l'entreprise. Modélisées de façon multidimensionnelle, les données de l'entrepôt sont accédées par différents outils d'accès appelés : les systèmes OLAP (Online analytical processing). Les processus OLAP interrogent l'entrepôt de données par des requêtes décisionnelles utilisées pour la restitution des données au profit des décideurs. Cependant, le traitement de ces requêtes nécessite un temps assez élève en raison : (i) de leur complexité (utilisation de plusieurs opérations de jointures, des opérations d'agré-

gations, et des opérations de groupements), et (ii) du gros volume de données de l'entrepôt sur lequel elles sont exécutées. Malheureusement, ce temps de traitement participe de façon directe à l'augmentation des délais de prise de décisions. Face à cette situation, la minimisation du temps de traitement des requêtes est devenue l'une des tâches d'administration d'entrepôt de données les plus importantes. Dans ce contexte, plusieurs techniques ont été proposées tel que : les indexes, les vues matérialisées, et la fragmentation de données. Parmi les techniques de fragmentation de données, nous citons la fragmentation horizontale primaire et la fragmentation horizontale dérivée qui ont été adapté à partir des bases de données distribuées, et qui ont connu une large utilisation par la communauté des chercheurs dans le domaine des entrepôts de données. Dans cet axe, les travaux existants de fragmentation horizontale ont proposés : (i) de fragmenter horizontalement (la fragmentation horizontale primaire) les tables de dimension par les prédicats de sélection utilisées par la charge de requête interrogeant l'entrepôt de données, ensuite, (ii) de dériver les fragments de faits par des opérations de semi-jointure entre la table de faits et chacun des fragments des tables de dimension (la fragmentation horizontale dérivée). L'utilisation de ce type de fragmentation a contribué à la minimisation du temps de traitement des requêtes. Malheureusement, dans ce type de fragmentation, l'explosion du nombre de fragments de faits, calculé en fonction du nombre de dimensions fragmentées et de leur nombre de fragments, rend la tâche de maintenance de ces derniers assez coûteuse et difficile pour l'administrateur de l'entrepôt de données. Pour remédier à ce problème d'explosion du nombre de fragments, des techniques de contrôle du nombre de fragments ont été proposées tel que : l'utilisation d'un seuil, la réduction du nombre de prédicats de sélection, la réduction du nombre de dimension à fragmenter... etc. Toutefois, des études expérimentales ont prouvé qu'un nombre élevé de fragments permet de minimiser significatives le temps de traitements des requêtes.

Face à cette situation, et afin de trouver un compromis entre une amélioration de performance des requêtes (aidant les managers à une bonne prise de décisions), et une réduction du nombre de fragment (facilitant leur maintenance à l'administrateur de l'entrepôt de données), nous avons proposé une nouvelle approche de fragmentation horizontale de la table de faits sans passer par la fragmentation horizontale des tables dimension. Notre approche de fragmentation, nous a permis à la fois de :

1. Atteindre des taux de performances meilleurs que ceux réalisés par la fragmen-

tation horizontale primaire et fragmentation horizontale dérivée,

2. Faciliter la tâche de maintenance à l'administrateur de l'entrepôt de données, vu que nous générons un nombre très réduit de fragments de faits qui ne dépend que du nombre de prédicats de sélection utilisés par les requêtes.

1.2 Contributions

Comme déjà mentionné ci-dessus, nous nous focalisons sur l'optimisation des performances des requêtes décisionnelles, à savoir minimiser leur temps de traitement pour améliorer le processus de prise de décisions. Pour atteindre cet objectif, nous proposons de fragmenter horizontalement l'entrepôt de données (Kechar and Nait-Bahloul, 2017). Contrairement à la fragmentation horizontale primaire et à la fragmentation horizontale dérivée, nous visons la génération d'un schéma de fragmentation horizontale :

1. Contenant un nombre réduit de fragments ne dépendant plus du nombre de dimensions et de leur nombre de fragments, et
2. Offrant un taux élevé de performance des requêtes.

Nos principales contributions pour trouver le schéma de fragmentation horizontale vérifiant les deux critères précédents sont :

1. Par un raisonnement mathématique basé sur des formules de calcul de coût utilisant l'algorithme de jointure par hachage, nous définissons cinq conditions que nous appelons : *conditions de quasi-optimalité*, que le schéma de fragmentation horizontale doit vérifier. Ces conditions tiennent en compte les points suivants :
   - La maximisation du nombre de requêtes bénéficiant de la fragmentation,
   - La minimisation du temps de traitement de chaque requête bénéficiant de la fragmentation,
   - La réduction de la détérioration du temps de traitement des requêtes non bénéficiant de la fragmentation,
   - La génération d'un minimum de fragments horizontaux.
2. Pour satisfaire nos conditions de quasi-optimalité, nous proposons une nouvelle approche de fragmentation horizontale de l'entrepôt de données. Notre approche consiste à fragmenter uniquement la table de faits horizontalement

sans passer par la fragmentation primaire des tables dimension. Nous générons les fragments de faits par des prédicats de sélection que nous choisissons parmi l'ensemble des prédicats de sélections utilisés par la charge de requêtes décisionnelles. Nous appelons les prédicats sélectionnés pour fragmenter la table de faits : *les prédicats de fragmentation*.
Pour une sélection efficace des prédicats de fragmentation, nous développons un algorithme qui exploite conjointement :
- Les sélectivités des prédicats de sélection sur la table de faits,
- Leur nombres d'occurrences dans la charge de requêtes, ainsi que,
- Leur fréquences d'accès à l'entrepôt de données.

3. Afin d'assurer le principe de transparence d'accès aux données de l'entrepôt après la fragmentation horizontale, nous proposons une stratégie de réécriture et de traitement des requêtes sur l'entrepôt de données fragmenté.

Nous présentons à la fin une étude expérimentale sur le Benchmark APB-1 pour prouver l'efficacité de :
- Nos conditions de quasi-optimalité,
- Nos solutions proposées pour satisfaire ces conditions, et
- Notre algorithme de fragmentation horizontale de la table de faits.

1.3 Organisation de la thèse

- *Chapitre 2 : Entrepôt de données : Concepts de base*. Dans ce chapitre, nous introduisons les concepts de base d'un entrepôt de données, à savoir, ses caractéristiques, sa modélisation multidimensionnelle, la structure des requêtes OLAP, ainsi que les principales opérations OLAP.
- *Chapitre 3 : Techniques d'optimisation de performances dans les entrepôts de données*. Nous consacrons ce chapitre aux techniques d'optimisation de performances des requêtes dans les entrepôts de données. Nous introduisons les définitions et les concepts de base : des index, des vues matérialisées, de la fragmentation verticale, et de la fragmentation horizontale de données. Nous passons en revue les travaux concernant les index, les vues matérialisées, et la fragmentation verticale. À la fin de ce chapitre, nous présentons les principaux avantages de la fragmentation horizontale dans les entrepôts de données par rapport aux autres techniques.

- *Chapitre 4 : Étude des techniques de fragmentation horizontale dans les entrepôts de données.* Vu les avantages de la fragmentation horizontale soulignés dans le chapitre précédent, nous présentons dans le chapitre 4 une étude détaillée des différents travaux de fragmentation horizontale dans les entrepôts de données.
- *Chapitre 5 : Problème d'optimisation de performances par la fragmentation horizontale : Nouvelle formalisation.* Nous présentons, dans ce chapitre, les inconvénients des techniques de fragmentation horizontale existantes, que l'étude présentée dans le chapitre 4 a permis leur identification. Pour remédier à ces inconvénients, nous proposons une nouvelle formulation du problème d'optimisation de performances par la fragmentation horizontale de l'entrepôt de données. Selon cette formulation, nous définissons cinq conditions de quasi-optimalité qu'un schéma de fragmentation horizontale doit satisfaire. Pour satisfaire ces conditions, nous proposons des solutions que nous implémentons à travers un algorithme de fragmentation horizontale détaillé dans le chapitre 6.
- *Chapitre 6 : Nouvelle approche de fragmentation horizontale dans les entrepôts de données.* Dans le chapitre 6, nous détaillons notre algorithme de fragmentation horizontale de la table de faits et nous proposons une stratégie de réécriture et de traitement des requêtes sur l'entrepôt de données fragmenté. La réécriture des requêtes, nous permet d'assurer le principe de transparence d'accès aux données de l'entrepôt fragmenté.
- *Chapitre 7 : Études expérimentales et validation.* Afin de prouver l'efficacité de notre approche de fragmentation horizontale de l'entrepôt de données, nous détaillons dans ce chapitre les différentes expérimentations.
- *Chapitre 8 : Conclusion et perspectives.* Dans ce chapitre, nous présentons une conclusion générale de notre travail ainsi que les perspectives de travaux futurs.

Chapitre 2

Entrepôt de données : Concepts de base

Sommaire

2.1 Introduction

De nos jours et dans un monde économique concurrentiel, chaque entreprise doit être attentive à ses performances et doit toujours maintenir une longueur d'avance sur ses concurrents potentiels. Pour atteindre ces objectifs, les managers assistés par les analystes, doivent toujours prendre les décisions les plus pertinentes dans des délais plus courts. Malheureusement, les bases de données transactionnelles conçues pour les processus OLTP (On-Line Transaction Processing) ont un support limité aux fonctions d'aide à la prise de décisions, car elles ne permettent pas l'extraction de toutes les informations nécessaires à la prise des décisions intelligentes, rapides et essentielles à l'évolution de l'entreprise. Cependant, effectuer des analyses sur les données transactionnelles pour des fins décisionnelles affecte négativement et de façon considérable les performances du système transactionnel, ce qui par conséquent alourdit les opérations quotidiennes de l'entreprise. De ce fait, il est devenu important de prendre en considération le traitement analytique des données pour tirer le maximum d'informations permettant aux managers un pilotage optimal des performances de l'entreprise. Ainsi la séparation entre le transactionnel et le décisionnel est devenu nécessaire à cause des nombreuses différences résumées dans le Tableau 2.1. Ces raisons ont donné naissance aux systèmes d'aide à la décision.

Golfarelli and Rizzi (2009); Kimball and Ross (2013) définissent le système d'aide à la décision comme étant l'ensemble des outils informatiques (matériels et logiciels) permettant l'analyse des données opérationnelles issues des systèmes d'information des entreprises. Ces données sont extraites et transformées en une vision orientée décideurs afin de faciliter leurs restitutions et leurs analyses. La Figure 2.1, décrit l'architecture d'un système d'aide à la décision constitué de :

- Les sources des données,
- La zone intermédiaire de données,
- La zone de présentation de données, et
- Les outils d'accès aux données.

1. *Les sources des données* : réunit généralement les données issues de plusieurs systèmes sources qui peuvent être hétérogènes et que l'entreprise utilise pour ces tâches quotidiennes, tels que : des fichiers plat (XML, CSV,... etc), des systèmes de bases de données ... etc. Les systèmes sources sont considérés comme externes à l'entrepôt de données et ne doivent en aucun cas être altérés.

TABLEAU 2.1 – Comparaison entre Bases de données et Entrepôt de données

	Base de données	**Entrepôt de données**
Objectif	Traitement transactionnel	Aide à la prise de décisions
Interaction avec l'utilisateur	Interrogation : Lecture/écriture	Interrogation-Lecture
Données	Courantes, exhaustives	Résumées, Historiques, orientées sujet
Usage	Support de l'opération de l'entreprise	Support de l'analyse de l'entreprise
Requêtes	Simples	Complexes
Utilisateur	Nombreux, Employés	Peu nombreux, Décideurs
Principe de conception	Troisième forme normale	Conception multidimensionnelle

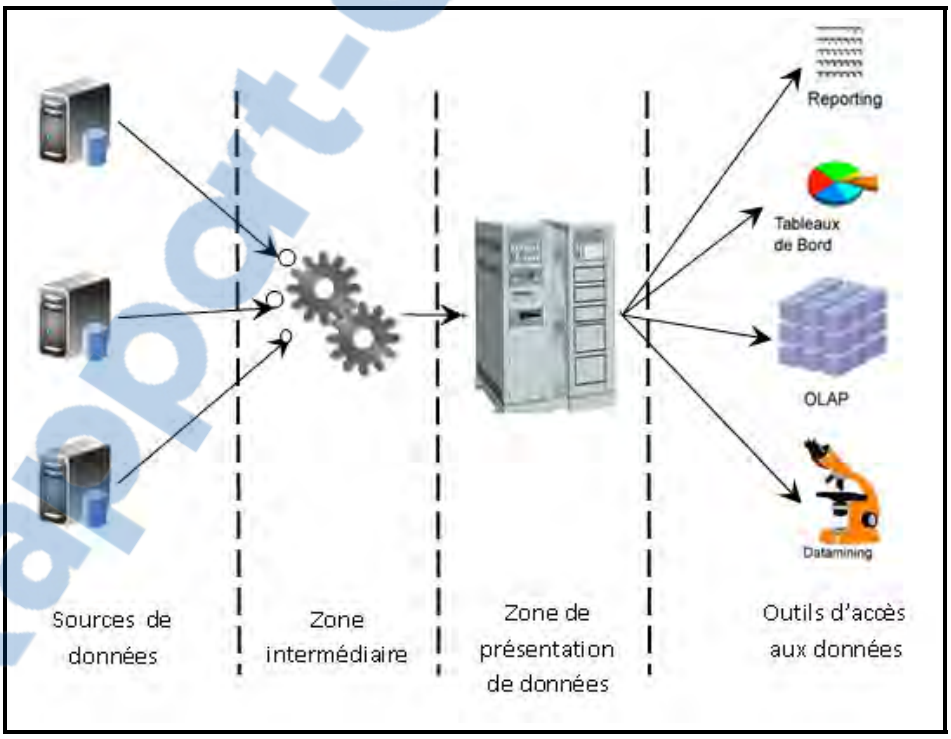


FIGURE 2.1 – Architecture du système décisionnel

L'enjeu principal est l'extraction des données nécessaires à la prise de décision pour leur chargement dans l'entrepôt de données sans affecter les performances et la disponibilité des systèmes sources. Dans les systèmes sources, peu de données historiques sont mises à jour, si l'entreprise dispose d'un bon entrepôt de données, les systèmes sources peuvent être allégés d'une grande partie de la responsabilité de représenter le passé.

2. *Les zones intermédiaires de données* : sont des zones de stockage des données issues des différents systèmes de production opérationnels. Les données dans cette zone subissent une suite de fonctions, commençant par l'extraction des données des différentes sources, leur transformation (traitement des données manquantes, des données dupliquées, les erronées, la conversion du format des données sources en format spécifique... etc), et finalement leur chargement dans l'entrepôt de données selon deux méthodes (le rafraîchissement ou la mise à jour) (Theodorou et al., 2016). Il est à noter que l'accès à la zone intermédiaire est interdit aux utilisateurs finaux.

3. *La zone de présentation de données* : est l'emplacement où les données sont organisées, uniformisées, enregistrées, et rendues disponibles pour des interrogations directes par des utilisateurs, des générateurs de rapports, et d'autres applications analytiques. La zone de présentation fait références à l'entrepôt de données (Data Warehouse) ou aux magasins de données (Datamarts) dont les données sont représentées en fonction des besoins d'analyses souhaitées.

4. *Les outils d'accès aux données* : les outils d'accès à l'entrepôt de données constituent la composante finale. Ils sont responsables de la réalisation des opérations d'analyse des données. Ces outils sont choisis en fonction des besoins des l'utilisateur en termes d'informations. Parmi ces outils d'accès, nous citons (Golfarelli and Rizzi, 2009; Ponniah, 2011) :
   - Les outils de reporting : Un rapport est définit par une requête utilisant des restrictions et des agrégations sur des données multidimensionnelles et par une disposition pour la visualisation des résultats (par exemple : tableau, diagrammes, histogrammes,... etc). Ces outils permettent la génération de rapports périodiques pour des utilisateurs nécessitant l'accès régulier aux informations, ainsi que la génération des tableaux de bord contenant différents indicateurs de performances au profit des managers.

- Les outils d'analyse OLAP (On-Line Analytical Processing) : permettent aux utilisateurs d'exécuter des requêtes multidimensionnelles complexes, appelées requêtes OLAP ou requêtes décisionnelles. Contrairement aux requêtes transactionnelles, les requêtes OLAP examinent un large volume de données même si leurs résultats sont de tailles petites (un petit nombre d'enregistrements).

Dans la section qui suit, nous mettons l'accent sur l'entrepôt de données. Nous présentons ses caractéristiques, sa modélisation multidimensionnelle, la structure des requêtes OLAP, ainsi que les principales opérations OLAP.

2.2 Définition d'un entrepôt de données

Bill Inmon, est l'un des premiers à avoir employé le terme *entrepôt de données* dans les années 90. Ce dernier le définit comme suit :
"*A data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions*" (Inmon, 1992).

Qui se traduit en français en : Un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles et évolutives dans le temps, utilisées comme support d'aide à la décision.
Cette définition englobe différentes caractéristiques (Figure 2.2), que nous explicitons ci-dessous.

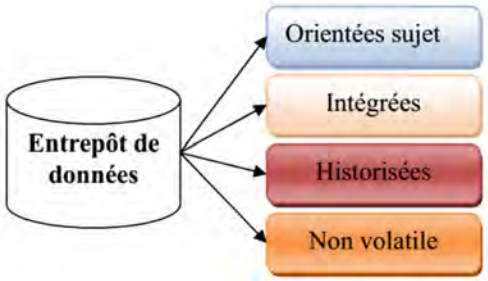


FIGURE 2.2 – Caractéristiques d'un entrepôt de données

2.3 Caractéristiques des entrepôts de données

- *Orientées sujet* : Les données sont organisées par domaine de sujet tel que les ventes dans une entreprise. Ce qui fournit une vue simple autour du sujet à analyser. Par exemple à partir des données ventes d'une entreprise, un entrepôt de données orienté ventes peut être construit. Ce dernier permet de répondre aux questions de genre : qui était notre meilleur client pour cet article l'année dernière.
- *Intégrées* : Parmi tous les aspects d'un entrepôt de données, l'intégration est la plus importante. Les données de l'entrepôt proviennent de différentes sources éventuellement hétérogènes. L'intégration consiste à résoudre les problèmes d'hétérogénéité des systèmes de stockage, des modèles de données, de sémantique de données...etc.
- *Non volatiles* : Les données de l'entrepôt sont essentiellement utilisées en mode de consultation. Une fois chargées dans l'entrepôt de données, elles ne sont plus modifiées.
- *Historisées* : La prise en compte de l'évolution des données est essentielle pour la prise de décision, qui par exemple utilise des techniques de prédiction en s'appuyant sur les évolutions passées pour prévoir les évolutions futures.

2.4 Modélisation multidimensionnelle d'un entrepôt de données

Les modèles utilisés pour concevoir des bases de données transactionnelles ne sont pas adaptés pour la modélisation d'entrepôt de données. Les transactions dans de telles bases de données sont constituées des requêtes simples et prédéfinies. Dans un environnement d'entrepôt de données, les requêtes sont plus complexes. Elles sont caractérisées par plusieurs opérations de jointures et d'agrégations, ce qui rend leur temps de traitement assez élevé. Ces requêtes sont désignées par le terme requêtes OLAP, leur traitement est basé sur la modélisation multidimensionnelle de données. Le modèle multidimensionnel de données, parfois désigné sous le nom de cube de données (Data Cube) s'est révélé être le modèle le plus avantageux pour la conception d'entrepôts de données. Les cellules du cube de données correspondent à un

ensemble de mesures (faits). Une mesure est vue comme un point dans un espace multidimensionnel dont les axes sont représentés par les dimensions décrivant le fait (Figure 2.3).

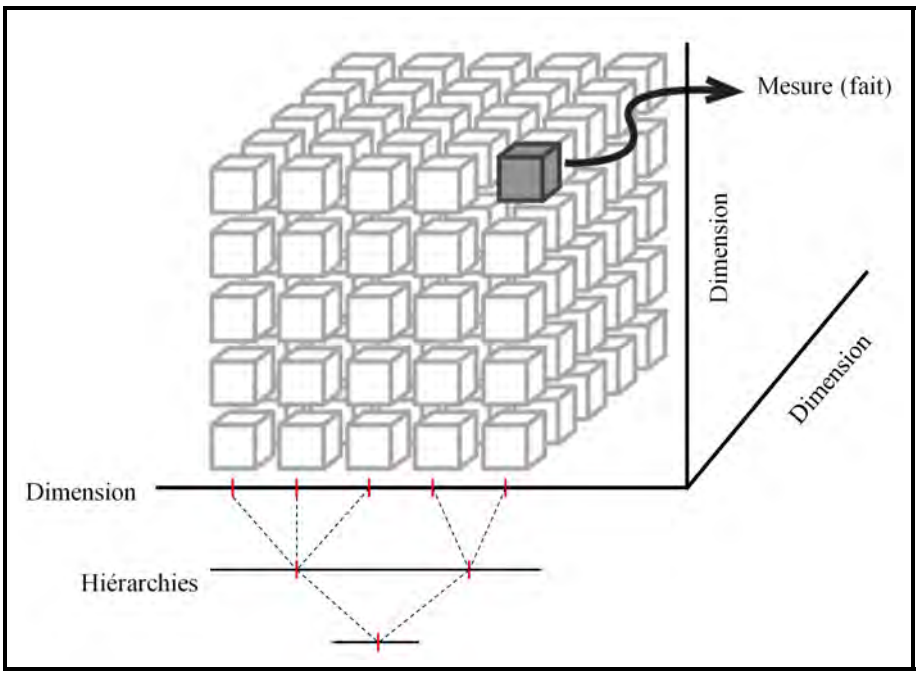


FIGURE 2.3 – Cube de données

Selon la Figure 2.3, le modèle multidimensionnel utilise trois concepts importants (Hobbs et al., 2005a; Ponniah, 2011; Kimball and Ross, 2013), à savoir :

- Concept de fait : un fait modélise le sujet d'analyse. Il est formé de mesures correspondantes aux informations de l'activité analysée. Les mesures d'un fait sont numériques et généralement valorisées de manière continue. Elles peuvent être additionnées, dénombrées, comme on peut calculer leur minimum, leur maximum, la moyenne... etc. Elles sont de type additives (peuvent être agrégées selon n'importe qu'elle dimension) ou semi-additives (peuvent être agrégées selon certaines dimensions).
- Concept de dimension : une dimension modélise une perspective d'analyse. Elle se compose d'attributs (paramètres) correspondant aux informations faisant varier les mesures de l'activité analysée. Les attributs d'une dimension sont des descripteurs textuels et de qualité (c.-à-d. aucune valeur manquante, obsolète, erronée, etc.). Ils sont utilisés pour restreindre la portée des requêtes afin de limiter

la taille des réponses. La puissance analytique d'un entrepôt de données est en relation étroite avec la richesse et la qualité des attributs des dimensions.

- Concept d'hiérarchie de dimension : une hiérarchie de dimension sert à restreindre ou à accroître les niveaux de détail de l'analyse. Elle regroupe des d'attributs organisés du niveau de granularité le plus fin vers le niveau de granularité le plus général. L'attribut qui permet d'identifier un niveau d'agrégation est appelé paramètre, tandis que les autres attributs de ce même niveau qui complètent la sémantique de ce paramètre sont appelés attributs faibles.

A partir de ces concepts, il est possible d'établir un modèle de données simple qui correspond au besoin de la modélisation multidimensionnelle. Ce modèle est appelé le schéma en étoile (star schema) (Kimball and Ross, 2013; Sohail and Dominic, 2015).

2.4.1 Le schéma étoile

Le schéma étoile est une structure simple (facile à comprendre) pour modéliser de façon multidimensionnelle les faits à analyser suivants différentes dimensions, en plus il offre des meilleures performances pour les requêtes d'analyse (Kimball and Ross, 2013; Sohail and Dominic, 2015). La Figure 2.4, montre le modèle Entité-association d'un schéma en étoile. L'entité centrale représente les faits à analyser. Les entités entourant les faits représentent les dimensions (les axes) d'analyse. La table de fait prend son nom du sujet analysé. Par exemple la Figure 2.5 représente la gestion d'une chaîne de magasins. L'entité centrale représente les ventes des produits qui sont les faits concernés par l'analyse. Les dimensions utilisées pour l'analyse des ventes sont : produit, magasin, client, et temps. Il est à noter que les dimensions du schéma en étoile sont non-normalisées, ce qui offre plus de performances pour le traitement des requêtes OLAP en évitant des opérations de jointure entre les hiérarchies de dimensions (Atzeni et al., 1999).

2.4.2 Le schéma en flocon

Le schéma en flocon est une version complexe du schéma en étoile, qui consiste à décomposer (normaliser) les dimensions du modèle en étoile en sous hiérarchies. La Figure 2.6 illustre la modélisation en flocon correspondant au schéma étoile de la Figure

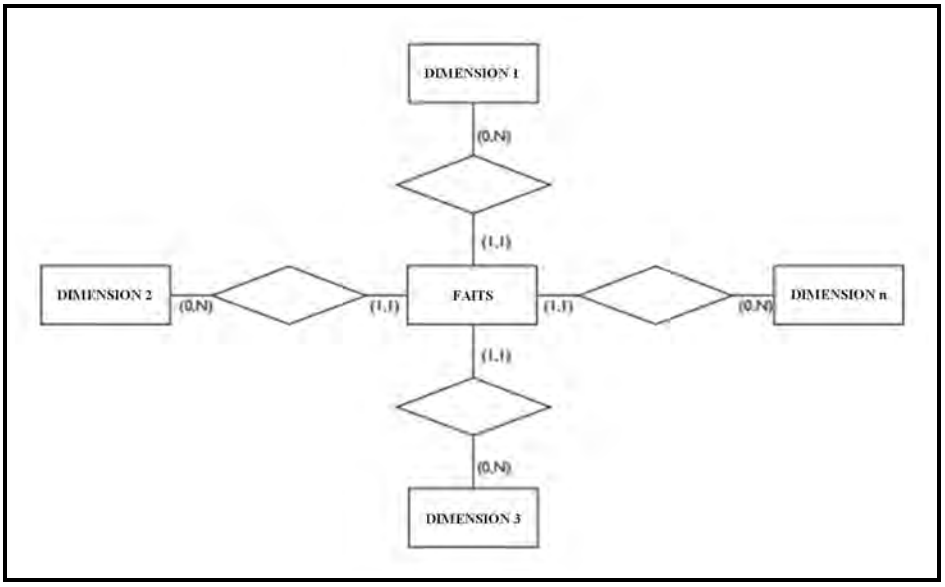


FIGURE 2.4 – Schéma étoile

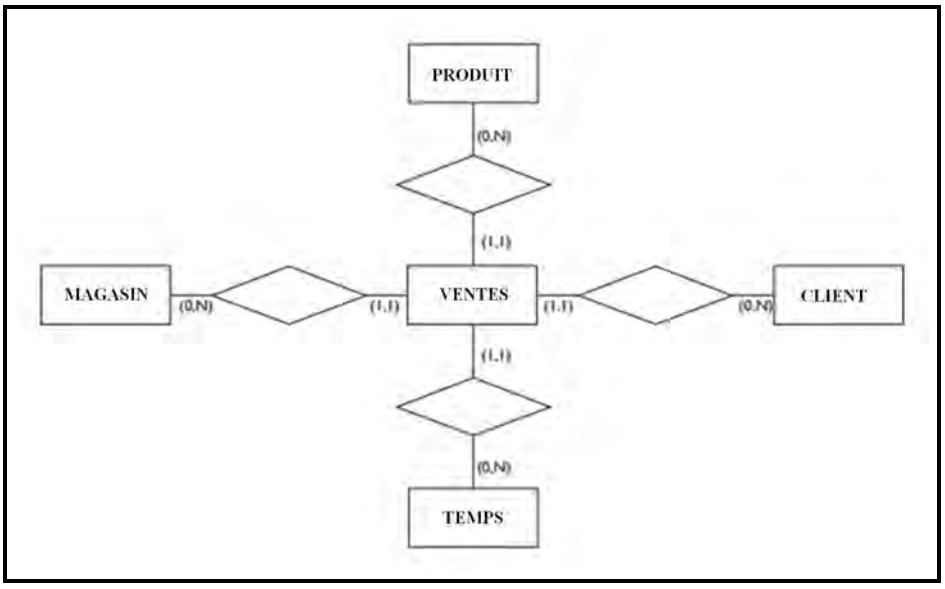


FIGURE 2.5 – Schéma étoile : Exemple

2.5 après normalisation des dimensions : Produit, Magasin, et Temps en hiérarchies de dimension suivantes : (Magasin↦Ville↦Région↦Pays), (Jour↦Mois↦Année), (Produit↦Catégorie)

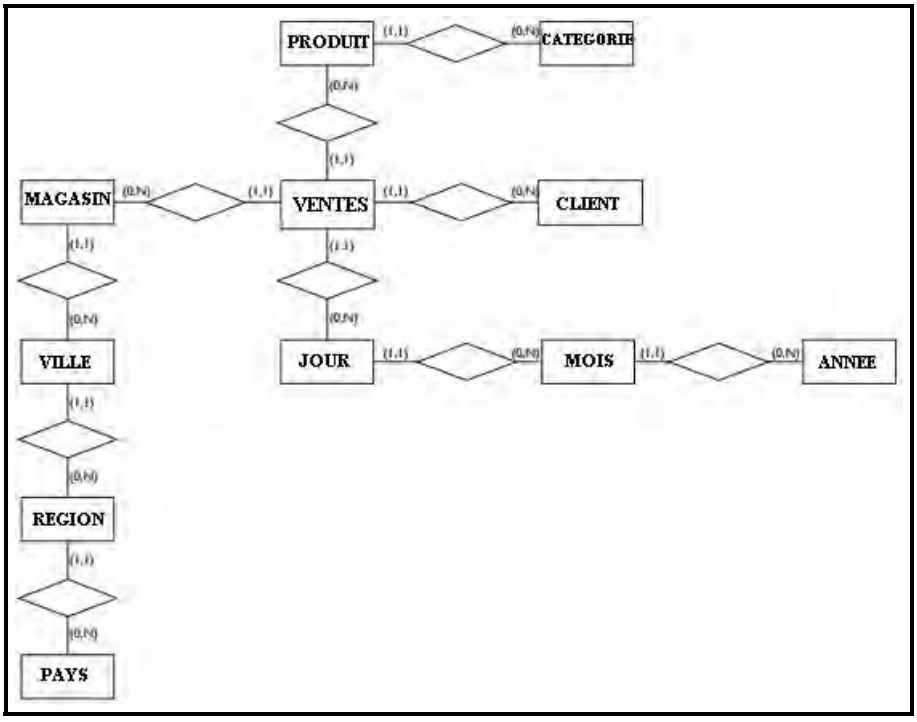


FIGURE 2.6 – Schéma en flocon : Exemple

2.5 Les implémentation OLAP

Garcia-Molina et al. (2008) définissent deux type de cubes de données : le cube brute de données et le cube formel de données . Cette distinction entre ces deux types de cubes est reflétée par deux systèmes d'implémentations multidimensionnelles :

- Les systèmes ROLAP, et
- Les systèmes MOLAP.

2.5.1 ROLAP (Relational On-Line Analytical Processing)

L'implémentation ROLAP consiste à utiliser un système de gestion de bases de données relationnelles (SGBDR), dans lequel le modèle multidimensionnel se traduit de la manière suivante :

- Les faits correspond à une table appelée table de faits, qui contient les mesures à analyser. Dans notre exemple de ventes, les mesures correspondent aux deux attributs : Quantité et Montant. La table de faits est caractérisée par :
  - ◇ Une clé composée de la concaténation des clés primaires des tables dimensions,

  - ◇ Un grand nombre d'enregistrements,
  - ◇ Un petit nombre d'attributs.
- Chaque dimension correspond à une table appelée table de dimension. Une dimension représente un axe sur lequel les faits peuvent être analysés. Les points suivants caractérisent chaque dimension dans un schéma étoile implémenté sous ROLAP :
  - ◇ Chaque enregistrement dans une table dimension est identifié de façon unique par une clé primaire,
  - ◇ Une table dimension est caractérisée par un nombre important d'attributs (colonnes),
  - ◇ Les attributs d'une table dimension sont textuels,
  - ◇ Les dimensions ne sont pas normalisées,
  - ◇ Un petit nombre d'enregistrements comparé à la table de faits.

Comme exemple, le schéma étoile (Entité/Association) de la Figure 2.5 se traduit dans un système ROLAP par l'ensemble des tables suivantes :

$Ventes(\underline{CodeProd}, \underline{CodeMagasin}, \underline{CodeClient}, \underline{Jour}, Quantite, Montant)$
$Produit(\underline{CodeProd}, Designation, Categorie, SousCategorie, Marque, Poid)$
$Magasin(\underline{CodeMagasin}, Nom, Superficie, Ville, Region, Pays)$
$Client(\underline{CodeClient}, Nom, Prenom, Adresse, Telephone, Email)$
$Temps(\underline{Jour}, Semaine, Mois, Trimestre, Annee)$

$CodeProd$, $CodeMagasin$, $CodeClient$, et $Jour$ représentent respectivement les clés primaires des dimensions : Produit, Magasin, Client, et Temps. Chaque enregistrement de la table de faits est identifié par la clé concaténée :
$CodeProd, CodeMagasin, CodeClient, Jour$.

2.5.2 MOLAP (Multidimensional On-Line Analytical Processing)

L'implémentation MOLAP utilise un système de bases de données multidimensionnel pur qui gère des structures multidimensionnelles natives. Ces structures multidimensionnelles sont représentées par des tableaux à n dimensions. Les implémentations MOLAP offrent des meilleures performances pour le traitement des requêtes OLAP (Hobbs et al., 2005b). La Figure 2.7 montre un exemple de cube de données re-

présentant l'analyse des activités d'une chaîne de ventes de produits de différentes catégories. Les quantités des ventes sont les valeurs des cellules du cube de données. Elles représentent l'indicateur d'analyse disponible du sujet Ventes (les faits). Cet indicateur est analysé en fonction de trois dimensions : Magasin, Dates, et Produits. Chaque dimension dispose de plusieurs niveaux de hiérarchies qui permettent d'obtenir des analyses plus ou moins détaillées. Par exemple la dimension Magasin est constituée des hiérarchies Ville, Région, Pays.

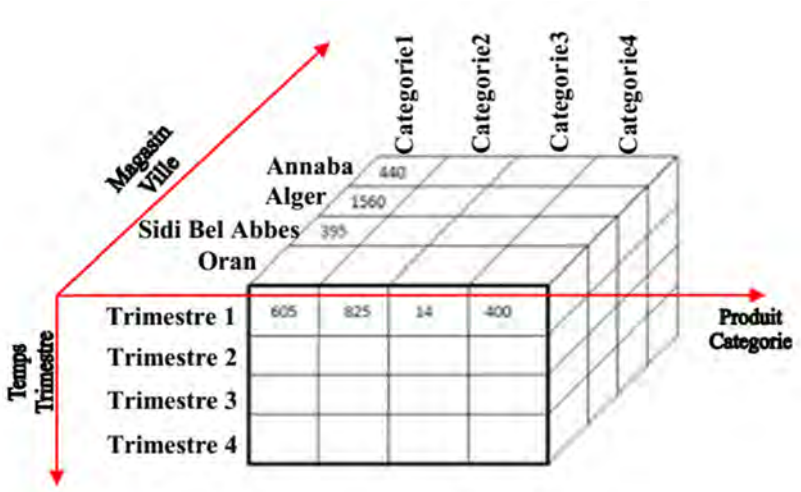


FIGURE 2.7 – Cube de données des Ventes de produits

2.6 Requêtes et opérations OLAP

2.6.1 Structure des requêtes OLAP

L'analyse de données de l'entrepôt modélisé par un schéma étoile nécessite une extraction d'un sous ensemble de faits et de dimensions. Cette extraction est réalisée par des requêtes appelées requêtes décisionnelles ou requêtes OLAP, qui respectent la règle suivant : les dimensions sont utilisées pour sélectionner et grouper les données, alors que les fonctions d'agrégation sont appliquées aux faits (mesures) (Atzeni et al., 1999). Les requêtes OLAP utilisent des opérations de jointure entre la table de faits et une ou plusieurs tables de dimension, des opérations de groupement, de tri, et des fonctions d'agrégation comme montré par la requête OLAP suivante :

$select\ D_1.C_1, ..., D_n.C_n, Aggr_1(F.C_1), ..., Aggr_M(F.C_n)$

$from\ Fact\ as\ F, Dimension_1\ as\ D_1, ..., Dimension_N\ as\ D_N$
$where\ ConditionDeJointure(F, D_1)$
$and\ ...$
$and\ ConditionDeJointure(F, D_N)$
$and\ ConditionDeSelection$
$group\ by\ D_1.C_1, ..., D_n.C_n$
$order\ by\ D_1.C_1, ..., D_n.C_n$

Où :

- Di (i=1..N) sont les tables de dimension,
- F est la table de faits,
- $D_i.C_j$ est la projection de la colonne j de la dimension i,
- $Aggr_k(F.C_i)$ (k=1..M) est la fonction d'agrégation (sum, $count$, max, min,... etc) appliquée à la mesure C_i de la table de faits F,
- $ConditionDeJointure(F, D_i)$: correspondent à $F.cle_i$=$D_i.cle$ où cle_i est la clé étrangère référençant la dimension D_i dans la tables des faits F,
- $ConditionDeSelection$: correspondent aux restrictions appliquées aux enregistrements des tables dimension.

Par exemple, un utilisateur peut s'intéresser à la somme des quantités du produit X vendu entre le mois de $Février$ et le mois de $Avril$. La requête de cet utilisateur se traduit par la requête OLAP suivante :

$select\ Temps.Mois, Produit.Designation, sum(Quantite)$
$from\ Ventes, Temps, Produit$
$where$
$Ventes.Jour = Temps.Jour$
$and\ Ventes.CodeProd = Produit.CodeProd$
$and\ Produit.Designation =' X'$
$and\ Temps.Mois\ between\ 'Février'\ and\ Avril'$
$group\ by\ Temps.Mois, Produit.Designation$
$order\ by\ Temps.Mois, Produit.Designation$

2.6.2 Les Opérations OLAP

2.6.2.1 Forage vers le bas (Drill-down)

Le Drill-down permet la désagrégation des données (naviguer de données moins détaillées vers des données plus détaillées) en utilisant les niveaux d'hiérarchies de dimensions d'analyse. Par exemple dans la Figure 2.8, au lieu de s'intéresser aux quantités vendues par Trimestres, un utilisateur peut forer vers le bas sur le Trimestre. Ainsi, il peut analyser les quantités vendues par les Mois des Trimestres en passant du niveau d'hiérarchie Trimestre vers le niveau d'hiérarchie Mois.

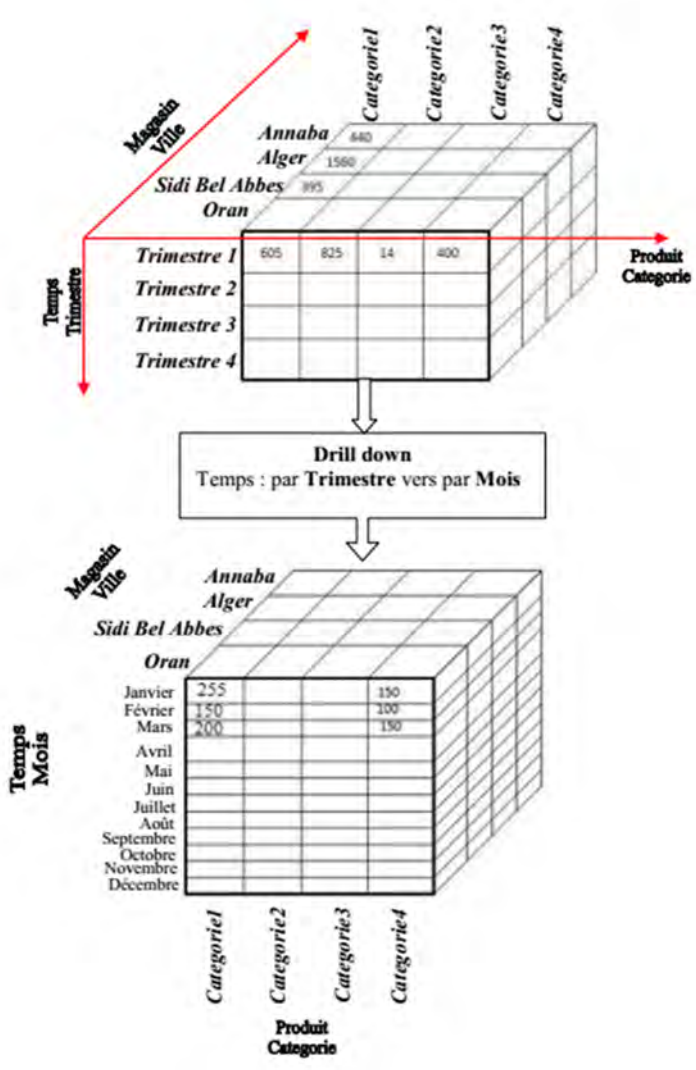


FIGURE 2.8 – Forage vers le bas (Drill-down)

2.6.2.2 Forage vers le haut (Roll-up)

Le Roll-up est l'opération inverse de l'opération Drill-down. L'opération forage vers le haut permet l'élimination d'une hiérarchies de dimension d'analyse par l'agrégation des données. Figure 2.9, montre l'opération Roll-up appliquée au cube de données qui permet d'extraire les données ventes de produits par trimestres et par régions au lieu de les extraire par ville en opérant sur les hiérarchies de la dimension Magasin.

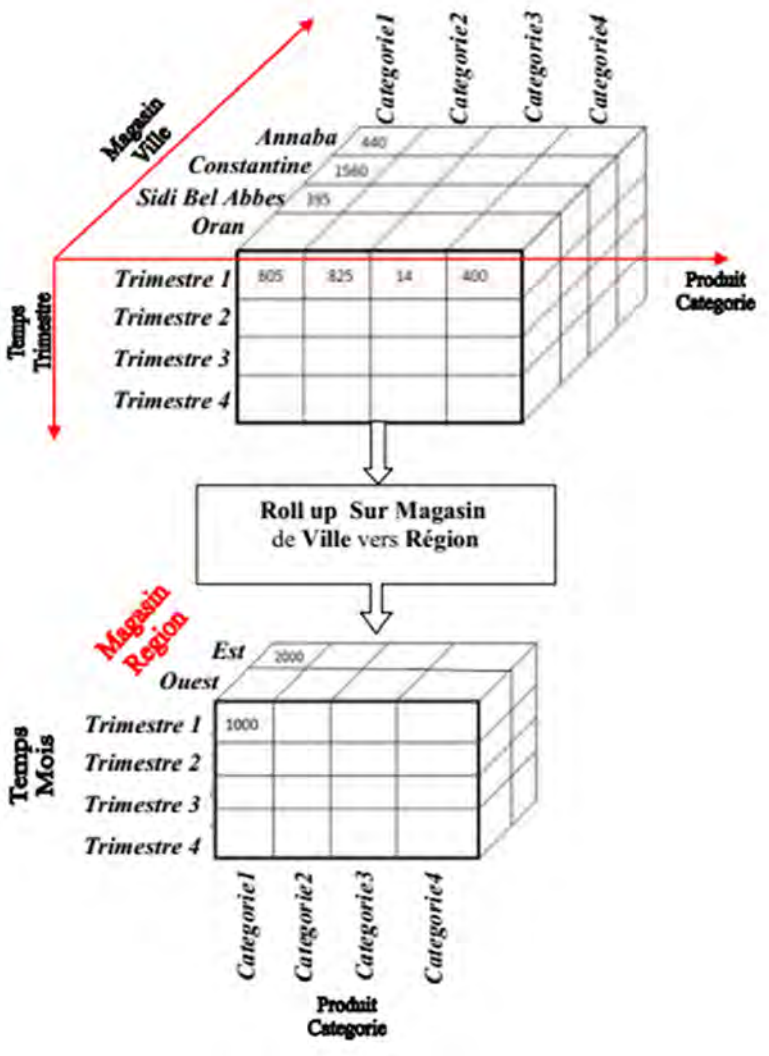


FIGURE 2.9 – Forage vers le haut (Roll-up)

2.6.2.3 Tranchage (Slice)

Consiste à sélection une des dimensions et la réduite à une seule valeur. Figure 2.10, permet de restituer les ventes des produits par ville et par le Trimestre 1. La dimension Temps est réduite à une seule valeur : Trimestre 1.

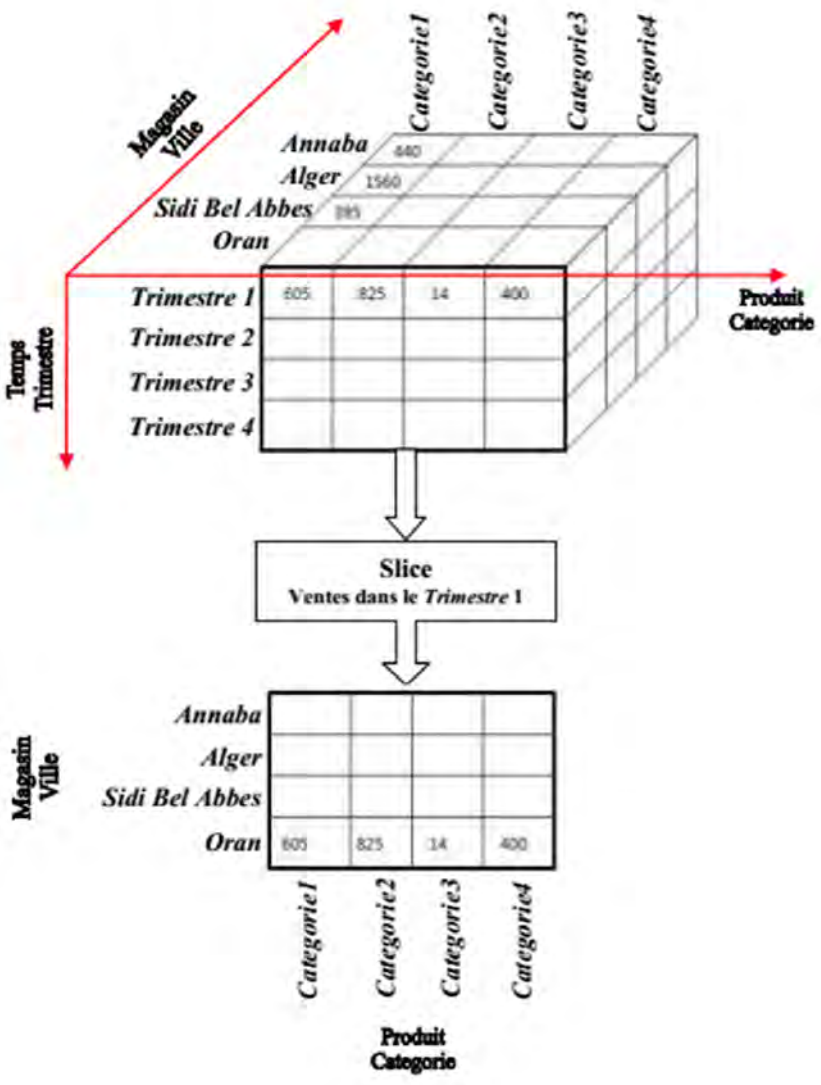


FIGURE 2.10 – Tranchage (Slice)

2.6.2.4 Extraction d'un bloc de données (Dice)

Permet d'extraire un sous-cube de données à partir du cube initial en sélectionnant deux ou plusieurs dimensions. Dans la Figure 2.11, le résultat de l'opération Dice sur le cube initial est représenté par le sous cube des ventes des produit de Catégorie1 et Categorie2 durant la période Trimestre1 et Trimestre2 dans les magasins des villes Oran et Sidi Bel Abbes.

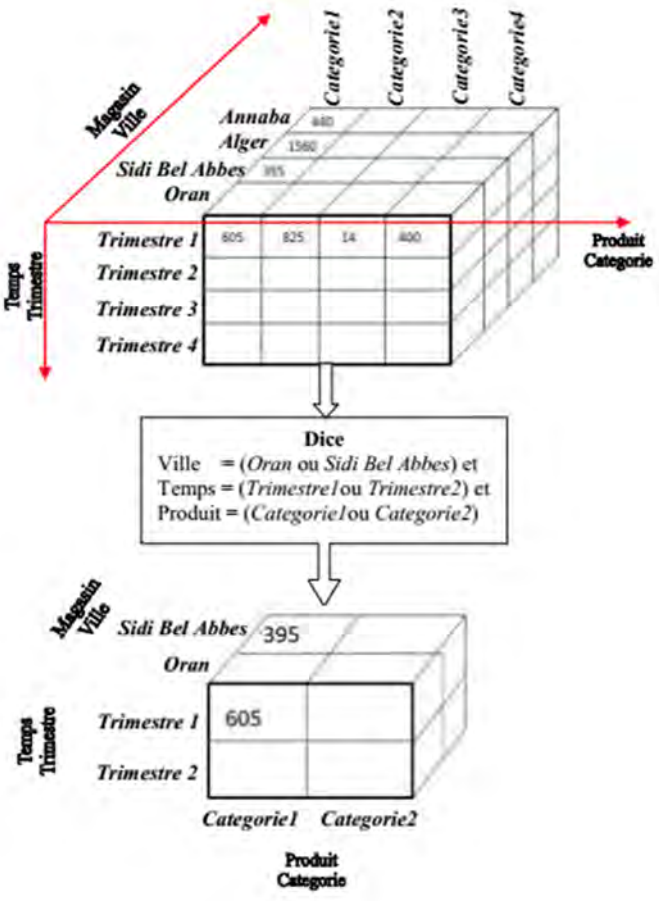


FIGURE 2.11 – Extraction d'un bloc de données (Dice)

2.7 Conclusion

Dans ce chapitre, nous avons introduit les principaux concepts d'entrepôt de données, à savoir ses caractéristiques, sa modélisation multidimensionnelle, ainsi que les différentes opérations OLAP réalisées par les requêtes décisionnelles. Dans le chapitre suivant, nous présentons les techniques utilisées ainsi que les travaux proposés pour optimiser les performances des requêtes OLAP dans les entrepôts de données.

Chapitre 3

Techniques d'optimisation de performances dans les entrepôts de données

Sommaire

3.1 Introduction

Avec l'évolution des technologies de l'information, les entreprises manipulent un gros volume d'informations circulant au sein de ses structures internes et échanger avec son environnement extérieur. Afin de garder une place dans un environnement assez compétitif, les entreprises collectent ces informations et les stockent dans un entrepôt de données, dont le volume ne cesse d'augmenter. L'entrepôt de données est interrogé par des requêtes complexes nécessitant des temps de traitement très élevés, ralentissant ainsi le processus de prise de décisions qui nécessite une grande interactivité. De ce fait, des techniques visant la réduction du coût d'exécution des requêtes s'avèrent nécessaires. Dans cet axe, plusieurs techniques d'optimisation de performances dans les entrepôts de données ont été proposées (Harinarayan et al., 1996; Gupta and Mumick, 1999; Chmiel et al., 2009; Bouakkaz et al., 2012; Ziani and Ouinten, 2013; Talebi et al., 2013; Wagner and Agrawal, 2013; Thenmozhi and Vivekanandan, 2014; Wojciechowski and Wrembel, 2014; Zamanian et al., 2015; Gosain and Heena, 2016; Jong Wook and Sae-Hong, 2016; Ghorbel et al., 2016). Elles sont classées en deux catégories principales (Bellatreche et al., 2007; Wrembel, 2012) : (i) les techniques redondantes (les vues matérialisées, les index, la fragmentation verticale) qui nécessitent un espace de stockage additionnel plus un coût de maintenance, et (ii) les techniques non redondantes (la fragmentation horizontale et le traitement parallèle) ne nécessitant aucun espace de stockage additionnel (Figure 3.1).

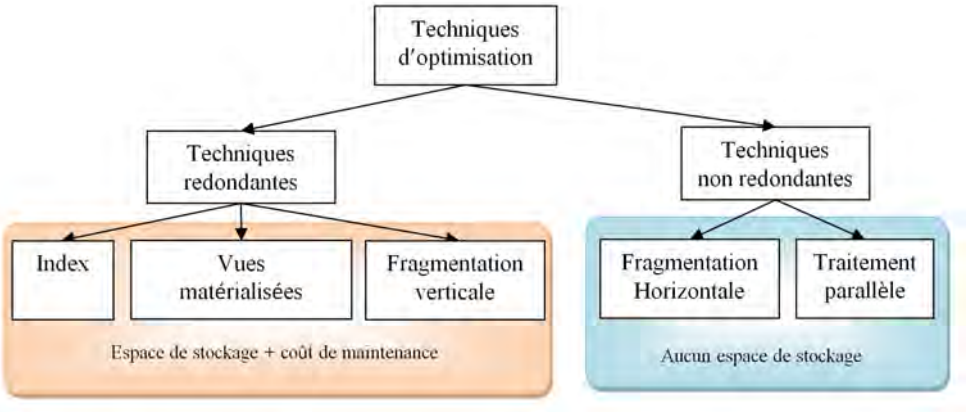


FIGURE 3.1 – Les techniques d'optimisation : Redondantes et Non Redondantes

3.2 Techniques et travaux d'optimisation

3.2.1 L'indexation des données

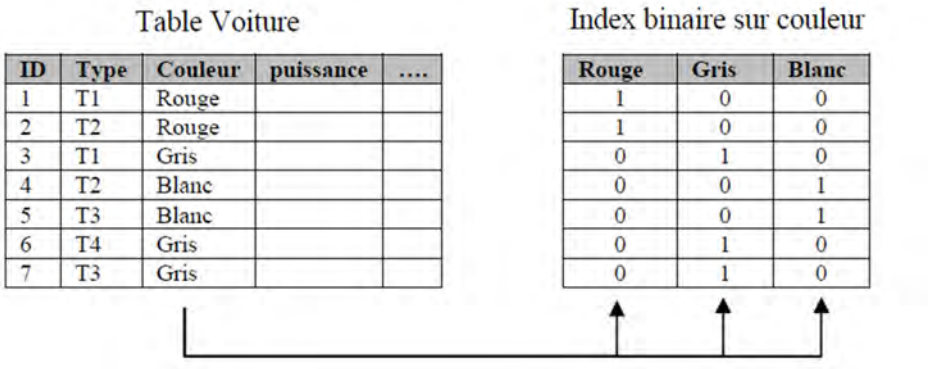
3.2.1.1 Définition

Un index sur une table R est une structure de données auxiliaire nécessitant un espace de stockage additionnel sur le disque dur. Il est conçu pour améliorer le temps de traitement des requêtes interrogeant la table R. En effet, un index sur R, nous permet de trouver efficacement (rapidement) tous les tuples de R satisfaisant des conditions de recherche sur les clés de recherche de l'index (Ramakrishnan and Gehrke, 2003; Claudia Imhoff, 2003).

3.2.1.2 Types d'index

3.2.1.2.1 Index binaires (Index Bitmap) : La structure d'un index binaire sur une colonne C d'une table T, contient une série de vecteurs de bits. Chaque vecteur correspond à une valeur v unique dans la colonne C. Chaque vecteur contient un bit pour chaque enregistrement t de la table T. Ainsi, la longueur d'un vecteur (nombre de bits) est égale au nombre d'enregistrement dans la table T. Le bit prend la valeur 1 si la valeur v de C est présente dans l'enregistrement t, sinon il prend la valeur 0. L'idée générale est de coder les valeurs d'une colonne par une séquence de bits, chacun pour une valeur possible. Par exemple une colonne sexe peut être codée par 10 ou 01, le 1 de la première position dénote Homme, et 1 dans la deuxième position dénote Femme. La Figure 3.2 montre un index binaire sur la colonne couleur ayant trois valeurs possibles : *rouge*, *gris*, *blanc*.

En ce qui concerne la taille de l'index bitmap, elle dépend totalement de la cardinalité de la colonne (attribut) indexée et du nombre d'enregistrement de la table indexée. Ce type d'index est très répondu dans les entrepôts de données, dont les tables dimension contiennent des attributs de faible cardinalité. Les avantages offerts par l'index binaires sont : (i) la possibilité d'utiliser des opérations binaires (and, or, ...etc.) entre les vecteurs de bits des attributs indexés pour répondre à une requête donnée, et (ii) les index bitmap peuvent être plus dense que les index B-arbre et offre la possibilité d'utiliser les techniques de compression (O'Neil and Quass, 1997).



ID	Type	Couleur	puissance	....
1	T1	Rouge		
2	T2	Rouge		
3	T1	Gris		
4	T2	Blanc		
5	T3	Blanc		
6	T4	Gris		
7	T3	Gris		

Rouge	Gris	Blanc
1	0	0
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

FIGURE 3.2 – Index binaire

3.2.1.2.2 Index B-arbre : De nos jours la majorité des systèmes de bases de données utilisent les index B-arbre comme méthode d'indexation par défaut. Ainsi, ces systèmes créent automatiquement des index B-arbre sur la clé primaire. L'index B-arbre est organisé sous forme d'arbre avec une racine, des nœuds internes appelés branches, et des nœuds feuilles (Ramakrishnan and Gehrke, 2003). La racine, ainsi que les nœuds internes contiennent des entrées qui pointent vers d'autres niveaux de l'index. Les nœuds feuilles contiennent les clés d'index et des pointeurs vers les adresses physiques où les enregistrements correspondant sont stockés. La Figure 3.3, montre un index B-arbre sur l'attribut âge de la table *Employe* contenant les informations des employés d'une entreprise.

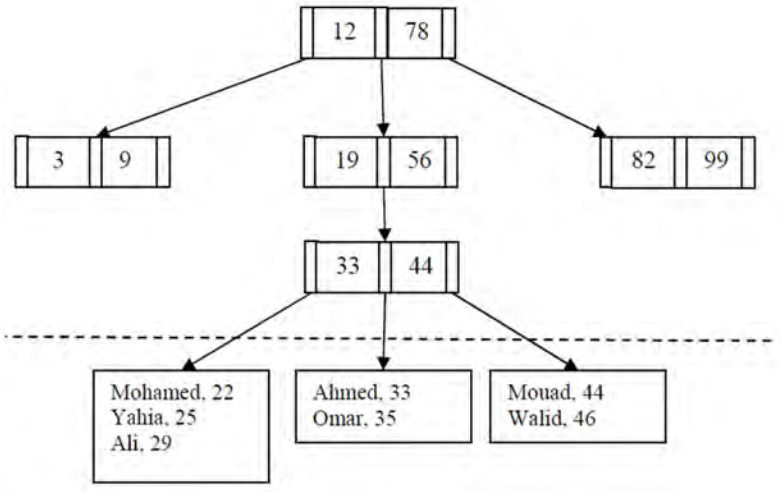


FIGURE 3.3 – Index B-Arbre sur l'attribut âge de la table Employe

3.2.1.2.3 Index de projection : L'index de projection est défini sur une colonne C d'une table T. Il est constitué d'une séquence de valeurs de la colonne C stocké dans le même ordre que celui des enregistrements à partir desquelles elles sont extraites. Ainsi, si une requête accède à la colonne C, alors seul l'index de projection sur C est chargé en mémoire pour répondre à cette requête. La Figure 3.4, montre un index de projection sur l'attribut $A2$ de la table R.

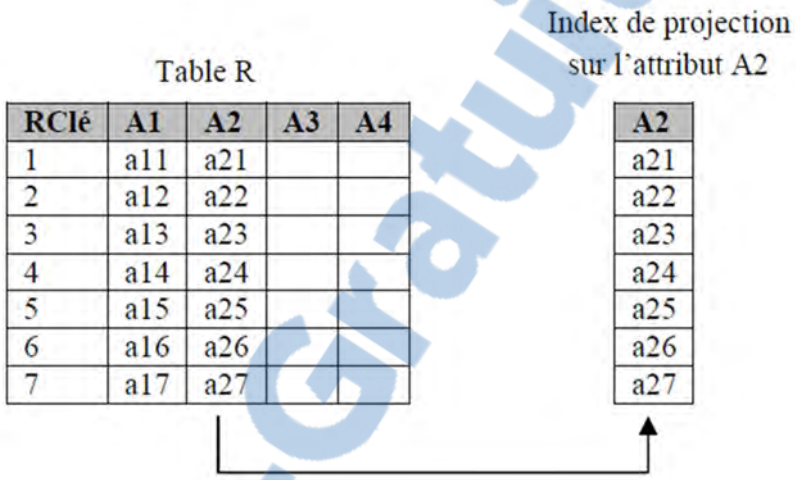


FIGURE 3.4 – Index de projection

3.2.1.2.4 Index de jointure : Valduriez (1987) définit un index de jointure entre une table R et une table S, comme une abstraction de la jointure entre R et S sur l'attribut A de R et l'attribut B de S. L'index de jointure est une relation binaire de taille petite ne contenant que les identificateurs des deux relations à joindre. Dans le cas des entrepôts de données, l'utilisation de l'index de jointure s'avère très utile pour accélérer l'opération de jointure entre une table de dimension et la table de faits. L'index de jointure est le résultat d'une jointure entre les tables R et S sur les attributs de jointure A, et B suivi d'une projection des identificateurs des deux tables à partir du résultat de la jointure. La Figure 3.5, montre l'index de jointure de la table R avec la table S. La taille de l'index de jointure dépend de la sélectivité de la jointure (Valduriez, 1987). Si la jointure a une bonne sélectivité (faible sélectivité)- qui est le cas dans les entrepôts de données : jointure sur les clés étrangères-, alors la taille de l'index de jointure sera très petite et contribuera significativement à l'amélioration des performances des requêtes de jointure. Dans

le cas contraire (c.-à-d. une forte sélectivité de la jointure), la taille de l'index de jointure sera très grande et se rapproche de la taille du produit cartésien des deux tables R et S.

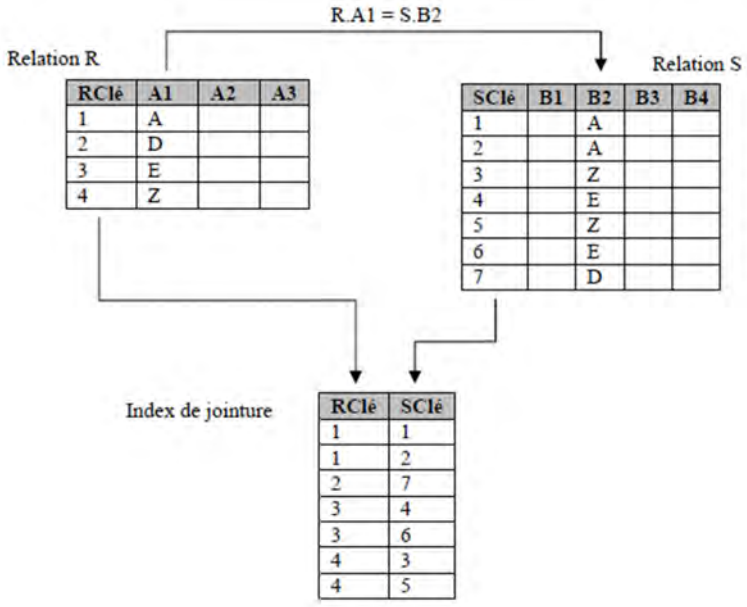


FIGURE 3.5 – Index de jointure

3.2.1.2.5 Index de jointure en étoile : L'utilisation de l'index de jointure (Valduriez, 1987) dans un entrepôt de données peut être très difficile, car plusieurs jointures sont effectuées entre les tables de dimension et la table de faits. Pour remédier à ce problème (Systems, 1997) ont proposé un index de jointure en étoile. Ce dernier peut contenir toute combinaison des identifiants des enregistrements de la table de faits et des identifiants des enregistrements des tables dimension. L'index de jointure en étoile est dit complet, si toutes les tables de dimension sont jointes avec la table de faits. Il est très bénéfique pour l'amélioration de performances de n'import quelles requête, sauf qu'il nécessite un grand espace de stockage. Si quelques tables de dimension sont jointes à la table de faits, alors l'index de jointure en étoile est dit partiel. Figure 3.6, montre un exemple d'index de jointure entre une table de faits F et deux table de dimension R et S. T_idj représente l'identificateur de l'enregistrement j dans la table T.

Identificateur de la table de faits	Identificateur de la dimension R	Identificateur de la dimension S
F_id1	R_id1	S_id1
F_id1	R_id2	S_id1
....	....	....
....	....	....
F_idm	R_idi	S_idj
....	....	....

FIGURE 3.6 – Index de jointure en étoile

3.2.1.2.6 Index de jointure binaire : Contrairement aux index binaires dont lesquels les attributs indexés appartiennent à la même table, l'index de jointure binaire est défini sur des attributs pouvant appartenir à plusieurs table (O'Neil and Graefe, 1995). Il permet de réduire le volume de données lors de la jointure en effectuant en avance les restrictions sur ces données. Dans le cas des entrepôts de données modélisé par un schéma en étoile, l'index de jointure binaire est défini sur la table de faits en utilisant des attributs appartenant aux tables de dimension. Par exemple, la Figure 3.7, montre un index de jointure binaire construit sur la table F en utilisant l'attribut couleur de la table R. L'utilisation de l'index de jointure binaire de la Figure 3.7, est très favorable lors de traitement des requêtes de types :

$select *$
$from\ F, R$
$where$
$F.RCle\ =\ R.RCle\ and$
$R.Couleur\ =\ 'Gris'$

L'index de jointure binaire est appelé index binaire de jointure multiple dans le cas où l'index est construit à partir des restrictions sur les valeurs d'attributs de différentes tables dimensions (Vanichayobon and Gruenwald, 1999)

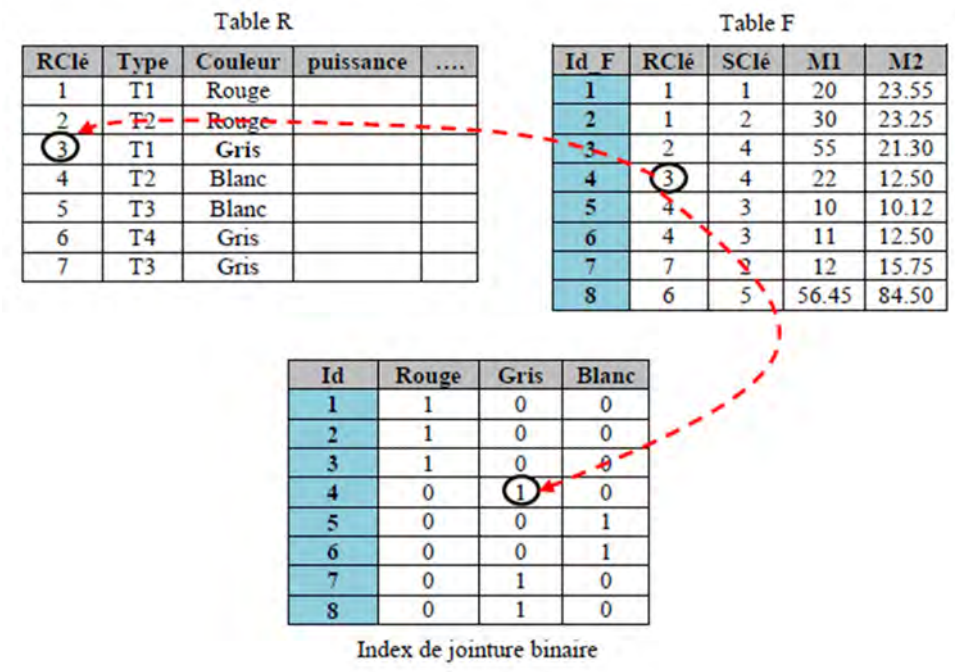


FIGURE 3.7 – Index de jointure binaire

3.2.1.3 Travaux sur l'indexation de données

Le nombre d'index pouvant être créés dans un entrepôt de données est généralement très large. Créer la totalité de ces index contribue à l'amélioration de performances de l'entrepôt de données. Malheureusement, le stockage physique de ces index nécessite un espace mémoire important sur le disque dur. En plus de leur maintenance, qui suivant leur nombre, devient une tâche difficile à l'administrateur de l'entrepôt de données. Ce problème est défini par la communauté des chercheurs dans les bases de données comme le problème de sélection d'index. Résoudre ce problème, revient à chercher une configuration d'index offrant un maximum d'amélioration de performances et impliquant moins d'espaces de stockage et moins de temps de maintenance. Dans cette optique :

Ziani and Ouinten (2013) ont proposé une approche de sélection d'index de jointure binaire pour améliorer les performances d'un entrepôt de données. Selon les auteurs, les approches de sélection d'index proposées, procèdent automatiquement à la sélection d'une configuration finale d'index à partir d'une exploration d'un large ensemble d'index candidats. Cependant, et contrairement à ces techniques, pour ré-

duire l'ensemble d'index candidats, les auteurs ont proposé une nouvelle approche basée sur les motifs fréquents (frequent itemsets) connue comme une technique du domaine de Data Mining. Ils commencent par construire un contexte d'extraction en identifiant les attributs indexables à partir d'une charge de requêtes. Ensuite, ils effectuent une extraction basée sur des contraintes (ex. la largeur des index générés, la cardinalité des attributs dans les index générés ... etc.) pour générer la configuration désirée d'index. Les auteurs ont souligné que, l'association entre l'utilisation des items fréquents basée sur des contraintes et l'intervention de l'administrateur de l'entrepôt de données ont contribué significativement à une sélection d'index efficace offrant des meilleurs performances.

Talebi et al. (2013) ont proposé de résoudre le problème de sélection de vues matérialisées et d'index dans les entrepôts de données en utilisant la programmation linaire en nombres entiers (Integer Programming). Si k est le nombre total d'attributs dans les tables dimension de l'entrepôt de données, alors (i) le nombre de vues pouvant être crées est égal à 2^k, (ii) sur chaque vue v_j dans l'ensemble des vues V, le nombre d'index possible est égal à $x!$, où x est le nombre d'attribut dans v_j. Ainsi, pour un k très élevé le nombre de vues ainsi que le nombre d'index sur ces vues sera très large, ce qui par conséquent rendra la résolution du problème de sélection de vues matérialisées et d'index assez complexe. En utilisant le modèle de coût défini dans (Gupta et al., 1997) pour calculer le coût de réponse d'une requête q sur une vue v (sans ou avec l'utilisation de l'index I défini sur v), les auteurs ont proposé une formulation du problème de sélection de vues et de sélection d'index sous la contrainte de l'espaces de stockage. Ainsi, pour une instance du problème (D,Q,b) où D est l'entrepôt de données, Q est la charge de requêtes, b est la limite de l'espace de stockage, et pour un ensemble de vues et d'index noté VI, les auteurs ont défini respectivement la notion de faisabilité et d'optimalité comme suite :

1. Si chaque requête dans Q peut être exécutée en utilisant les vues dans VI et l'ensemble VI satisfait la limite de stockage b, alors VI est faisable pour (D,Q,b),

2. Si un VI^* est faisable et s'il minimise le coût de traitement de Q sur D parmi tous les VI faisables, alors VI^* est un ensemble optimal de vues et d'index pour (D,Q,b).

A partir de la formulation du problème de sélection de vues et d'index et à base de variables et de contraintes définies sur ces variables, les auteurs ont modélisé le problème par un modèle de programmation linaire en nombres entiers noté $IP1$, dans lequel ils ont pris en considération toutes les vues et tous les index possibles appartenant à l'espace de recherche. Cependant, comme mentionné ci-dessus, dans les cas réels, le nombre de vues et d'index peut être assez grand, et par conséquent la taille du modèle $IP1$ sera assez large. En exploitant les propriétés du problème de sélection de vues, les auteurs ont proposé un passage du modèle $IP1$ à un autre modèle $IP2$, en réduisant la taille de l'ensemble de vues et d'index par la suppression de vue et d'index non pertinents. Cette réduction dans la taille de l'ensemble des vue et d'index a conduit à la réduction du modèle de la programmation linaire en nombre entier facilitant ainsi sa résolution. Ce qui par conséquent a permis de générer une configuration de vues et d'index offrant une optimisation optimale de performances des requêtes.

Chmiel et al. (2009) ont proposé un index binaire organisé en hiérarchies noté HOBI (Hierarchically Organized Bitmap Index) pour indexer les données des tables dimension ainsi que leurs hiérarchies. Contrairement à la majorité des technique d'indexation proposées dans les entrepôt de données modélisés en étoile, les auteurs ont proposé d'indexer un entrepôt de données représenté par un schéma en flocon composé d'une table de faits reliée aux tables dimension dont chacune est composée d'une hiérarchie (Section 2.4.2). Ainsi, l'index HOBI est composé de plusieurs index binaire dont chacun est créé pour chaque niveau de hiérarchie de dimension, de sorte que l'index HOBI du niveau supérieur représente les agrégations des index binaire du niveau inférieur. Comme déjà connue, si une dimension D est composée de x niveaux d'hiérarchies N_1, N_2,..., et N_x, alors plusieurs tuples de niveau N_i (niveau inférieur) sont référencés par un seul tuple de niveau N_{i+1} (niveau supérieur). En plus, les tuples de chaque niveau N_i sont identifiés par les valeurs de l'attribut clé de N_i. En exploitant ces notions, les auteurs ont créé un index de jointure binaire pour l'attribut clé du niveau N_1, qui est composé de k vecteurs binaires, où k est le nombre de valeur distincte de l'attribut clé indexée dans le niveau N_1. Pour indexer l'attribut clé du niveau N_2, les auteurs ont créé pour chaque valeur v_i de la clé de N_2 un vecteur binaire dont ses valeurs sont calculées en effectuant un OU logique entre les vecteurs des valeurs des clés du niveau N_1 référencées par vi. Ainsi, les index

des niveaux N_{i+1} sont calculés de la même façon. Par une évaluation expérimentale sous oracle 10g, les auteurs ont prouvé l'efficacité de leur index HOBI comparée par rapport à l'index de jointure binaire créé par Oracle10g.

Dans le but d'améliorer l'index HOBI, les auteurs dans (Morzy et al., 2012) ont proposé d'étendre ce dernier en une version notée Time-HOBI. L'index Time-HOBI combine l'index HOBI avec l'index temps associé à la dimension temps. Car, dans les cas réel, la dimension temps est incluse dans la majorité des entrepôts de données. Ainsi, l'utilisation de l'index Time-HOBI permet d'éviter la jointure de la table de faits avec la dimension temps, ce qui par conséquent améliore d'avantage les performances des requêtes de jointure.

Dans (Wojciechowski and Wrembel, 2014), les auteurs ont étendu l'index Time-HOBI par l'ajout des agrégations partielles. Les agrégations partielles consistent à calculer les valeurs des fonctions d'agrégations sur l'ensemble des mesures pour une dimension Di, un niveau de hiérarchie de dimension, des instances d'un niveau de hiérarchie de dimension, et pour un intervalle de temps. Ces valeurs sont utilisées : (i) pour filtrer les segments des index binaires dans l'index HOBI qui ne contribuent pas dans les critères de sélection sur des agrégations, ou (ii) pour calculer des agrégations dans un niveau supérieur basé sur les agrégations d'un niveau inférieur dans une hiérarchie de dimension.

3.2.2 Les vues matérialisées

3.2.2.1 Introduction

Une vue matérialisée constitue un objet physique de la base de données. Elle contient les données résultat d'une requête qui peut être une requête nommée (une vue). La matérialisation des vues est considérée comme une technique très utile :

- Pour minimiser le temps de réponse des requêtes décisionnelles dans les entrepôts de données (pré-calcul des jointures).
- Pour éviter les coûts de transfert des données sur le réseau dans une architecture distribuée d'entrepôt de données (dupliquées les données sur chaque site de l'architecture).
- Pour contrôler l'accès aux données (c.-à-d. seules les données autorisées sont mises à la disposition des utilisateurs à travers des vues matérialisées).

Cependant, la sélection des vues à matérialiser sous contrainte de maintenance (l'espace de stockage, le temps de maintenance total des vues matérialisées) constitue un problème majeure.

Gupta and Mumick (1999) ont défini le problème de sélection de vues matérialisées comme suit : étant donnée (i) un ensemble de n requêtes fréquentes interrogeant un ensemble de tables, (ii) un ensemble de m vues, et (iii) une contrainte de maintenance (soit un espace de stockage ES, soit un temps de maintenance TM), le problème de sélection de vues matérialisées consiste à sélectionner parmi les m vues un sous ensemble de k vues à matérialiser de sorte que le temps de réponses des n requêtes soit minimisé sous la contrainte que l'espace de stockage totale des k vues matérialisées ne dépasse pas ES ou le temps de maintenance des k vues matérialisées ne dépasse pas TM. Dans ce contexte, plusieurs travaux dans les entrepôts de données ont proposé divers algorithmes pour résoudre le problème de sélection des vues matérialisées. Parmi ces travaux, nous citons ceux proposés dans (Gupta and Mumick, 1999; Zhixiang and Dongjin, CCA 2014; Gosain and Heena, 2016; Goswami et al., 2016; Sohrabi and Ghods, 2016).

3.2.2.2 Travaux sur la matérialisation des vues

La matérialisation (le pré-calcul et le stockage) de toutes les vues est impossible, due : (i) à leur grand nombre qui croit exponentiellement en fonction des attributs des dimensions utilisés par les requêtes interrogeant l'entrepôt de données, et (ii) à la contrainte de disponibilité de l'espace de stockage. De ce fait, la sélection d'une configuration optimale de vues matérialisées constitue une solution importante d'un côté et un problème assez complexe de l'autre côté (Baralis et al., 1997). Dans cette optique, plusieurs techniques ont été proposées pour résoudre le problème de sélection de vues matérialisées. Leur objectif principal est l'optimisation de performances des requêtes tout en respectant une contrainte de limite d'espace de stockage ou/et la contrainte de temps de maintenance.

Harinarayan et al. (1996) ont introduit la structure de treillis composée de nœuds représentant toutes les vues pouvant être créés à partir des dimensions de l'entrepôt de données par des agrégations à des niveaux différents. Ainsi pour n dimensions, le treillis sera composé de 2^n vues (nœuds). Un chemin existe entre deux nœuds n_i et n_j s'il existe une relation de dépendance notée $n_i \leq n_j$, qui signifie que si

le résultat d'une requête q peut être obtenu à partir de n_i alors le même résultat de q peut être aussi obtenu à partir de n_j. Matérialiser toutes les vues du treillis permet aux requêtes d'avoir les meilleurs temps des réponses. Malheureusement, pour une importante valeur de n, cette matérialisation nécessitera un grand espace de stockage.

Wagner and Agrawal (2013) ont proposé une sélection de vues matérialisées pondérées (PSVM-pondéré). La pondération consiste à attribuer un poids à chaque vue, qui est en fonction de la fréquence d'accès et/ou de l'importance de la requête accédant à la vue. Les auteurs ont proposé une approche basée sur un algorithme évolutif pour déterminer l'ensemble de vues à matérialiser de tel sorte que le nombre maximum de page disque soit minimisé. À partir : (i) d'un entrepôt de données, (ii) une liste de vues organisée sous forme de treillis, (iii) un poids associé à chaque vue, (iv) le nombre de pages occupées par chaque vue, (v) le nombre maximum de page disque pouvant être occupées par la matérialisation de vues, ils ont procédé de la façon suivante :

1. Une population de solutions candidates est crée aléatoirement. Selon la formulation du PSVM-pondéré, la population des solutions candidates est composée de plusieurs ensembles de vues construit à partir du treillis des vues possibles. Chaque ensemble doit respecter les contraintes suivantes : (i) la vue racine est incluse dans chaque ensemble, (ii) aucune vue n'est dupliquée dans l'ensemble, et (iii) le nombre de vues dans un ensemble doit être entre 2 et n, où n est le nombre de vues pouvant être matérialisées,
2. Chaque solution est classée en fonction de la valeur Z représentant le nombre maximal pondéré de pages qui doivent être récupérées pour obtenir les vues dans la solution,
3. Les solutions pertinentes ayant des valeurs Z petites sont sélectionnées pour effectuer les opérations génétiques (sélection, croisement, mutation). Si deux solutions ont la même valeur Z, alors celle contenant un minimum de vues est favorisée,
4. Des opérations de croisement et/ou de mutation sont appliquées à la nouvelle population obtenue précédemment. Ces opérations sont répétées pour un certain nombre d'itérations ou jusqu'à ce que aucune amélioration dans la qualité de la solution n'est remarquée,

5. La solution finale, est celle ayant le plus haut classement dans la dernière population générée.

Gosain and Heena (2016) ont proposé une approche utilisant l'algorithme d'optimisation Particle Swarm pour parcourir les nœuds du treillis à la recherche d'un ensemble optimal de vues à matérialiser avec une limite d'espace de stockage à ne pas dépasser. Les solutions candidates sont codées sous forme de chaînes binaires de longueur n, ou chaque bit correspond à une vue dans le treillis. Chaque bit i d'une solution peut prendre soit la valeur 1 ou la valeur 0, indiquant respectivement la matérialisation ou non de la vue v_i correspondante. En plus, chaque solution est caractérisée par une position dans un espace N-dimensions et un paramètre de vélocité varié entre 0 et 1, qui défini la probabilité avec laquelle un bit dans la solution (chaîne binaire) est changé. Commençant par une population initiale de n particules, où chaque particule constitue une solution candidate, l'algorithme parcourt l'espace de recherche (l'espace des solutions possibles) par des déplacements d'une solution à une autre en changeant la valeur du paramètre vélocité, et en évaluant la qualité de chaque solution par rapport à son voisinage. L'algorithme s'arrête lorsque pour une solution donnée, aucune amélioration (minimisation) du coût d'exécution des requêtes n'est observée. Ainsi, les vues correspondantes aux bits ayant la valeur 1 dans la solution retournée sont matérialisées.
Une étude exhaustive des différentes techniques et algorithmes adressant le problème de sélection de vues matérialisées, a été menée dans (Goswami et al., 2016).

3.2.3 La fragmentation de données

En général, l'entrepôt de données contient des tables assez larges, en particulier la table de faits qui peut contenir des millions d'enregistrements. Devant un tel gros volume de données, des problèmes de performances (augmentation du temps) peuvent exister lors :

- Du chargement des tables,
- De la création et de la maintenance des index sur les tables,
- De l'archivage et de la récupération des données,
- Du traitement des requêtes.

Remédier à ces divers problèmes de performances, peut être réalisé par la fragmentation (le partitionnement) du gros volume de données.

La fragmentation consiste à diviser une table en plusieurs partitions appelées fragments, de manière à ce que la combinaison des fragments recouvre l'intégralité de la table source sans ajout ni perte de données. Dans la littérature, la fragmentation de données existe sous forme de trois types (Ponniah, 2011; Özsu and Valduriez, 2011) : la fragmentation verticale, la fragmentation horizontale et la fragmentation mixte.

3.2.3.1 La fragmentation verticale

La fragmentation verticale d'une table R consiste à diviser R en fragments FV_1, FV_2,..., FV_n, appelés fragments verticaux. Chacun des fragments FV_i contient :

- Un sous ensemble d'attributs qui sont des projections appliquées à la table R,
- La clé primaire de R pour permettre la reconstruction de la table originale R par jointure avec les autres fragments verticaux, et
- Le même nombre d'enregistrement que dans R.

L'objectif de la fragmentation verticale de R, consiste à minimiser le temps de traitement des requêtes interrogent R en les exécutant sur les fragments FV_i.
Par exemple la table R contenant n attributs peut être fragmenté verticalement comme suite :

La table R : $R(C, a_1, a_2, a_3, ..., a_n)$
Les fragments verticaux :
$FV_1(C, a_{11}, a_{12}, a_{1x})$,
$FV_2(C, a_{21}, a_{22}, a_{2y}), ...,$
$FV_k(C, a_{21}, a_{22}, a_{2z})$

Les attributs a_{ij} de FV_i appartiennent à l'ensemble des attributs $\{a_1, a_2, ..., a_n\}$ de la table R.

Cependant, la complexité de la fragmentation verticale est en fonction étroite avec le nombre d'attributs non clé de la table R. Ainsi, pour n attribut non clé, il existe $B(n)$ fragments, où $B(n)$ représente le $n$ième nombre de Bell (Barsky and Benzaghou, 2004). Par exemple si $n = 10$, $B(n) = 115975$, n=15, $B(n) = 1382958545$.
Ces valeurs énormes ont conduit à l'utilisation des heuristiques suivantes :

- Le regroupement : qui consiste à affecter chaque attribut à un fragment, ensuite procéder par le regroupement des fragments jusqu'à satisfaire un certain critère

d'optimisation.

- L'éclatement : qui consiste à diviser la table R en fragments verticaux basé sur le comportement d'accès des requêtes aux attributs.

Il est à noter que la fragmentation verticale favorise l'optimisation de performances des requêtes de projection. La Figure 3.8, montre les trois fragments verticaux FV_1, FV_2, et FV_3 résultat d'une fragmentation verticale de la table R.

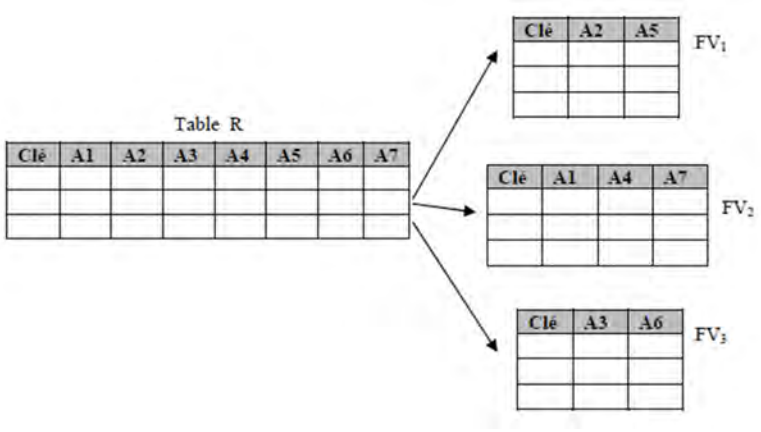


FIGURE 3.8 – La fragmentation verticale de R en trois fragments verticaux FV_1, FV_2, et FV_3

3.2.3.2 La fragmentation horizontale

La fragmentation horizontale d'une table R, consiste à partitionner R en fragments horizontaux contenant chacun un sous ensemble de tuples de R. Les sous ensemble de tuples appartenant à un fragment horizontal vérifient généralement une clause de prédicats de sélection satisfaite par des opérations de restriction appliquées aux tuples de R.

La fragmentation horizontale se divise en deux variantes : la fragmentation horizontale primaire et la fragmentation horizontale dérivée (Özsu and Valduriez, 2011).

3.2.3.2.1 La fragmentation horizontale primaire : La fragmentation horizontale primaire d'une table R, s'effectue par des opérations de restriction sur les tuples de R. Elle favorise le traitement des requêtes de restriction en minimisant l'accès aux données (tuples) non nécessaire.

Formellement, la fragmentation horizontale primaire de la table R, génère des frag-

ments horizontaux FH_i, tel que $FH_i = \delta_{cli}(R)$, où cli est une clause de prédicats de sélection, δ est l'opérateur de restriction appliqué aux tuples de R (Figure 3.9). Dans

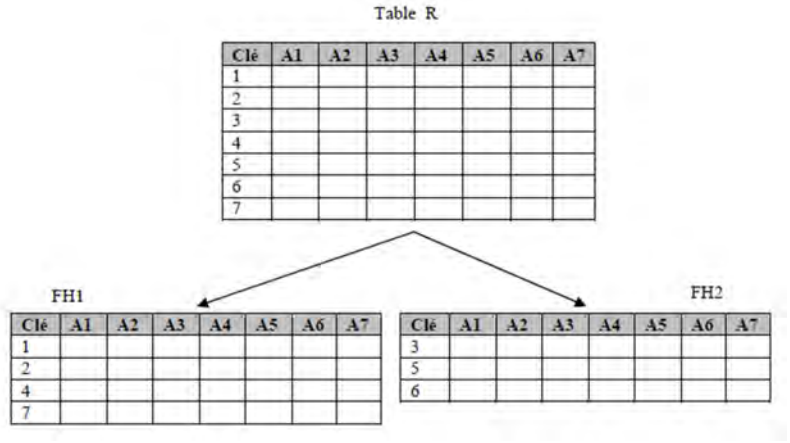


FIGURE 3.9 – La fragmentation horizontale primaire

la plupart des systèmes de gestion de bases de données actuels tels que : Oracle, SQL Server, MySQL, une table relationnelle peut être fragmentée horizontalement selon l'une des techniques suivantes (Hobbs et al., 2005a; Wrembel, 2012) :

- Partitionnement par intervalle (Range partitioning),
- Partitionnement par valeurs discrètes (List partitioning),
- Partitionnement par hachage (Hash patitioning), et
- Partitionnement composé (composite patitioning).

Partitionnement par intervalle : Dans le partitionnement par intervalle, la table est fragmentée en partitions basées sur les intervalles de valeurs de l'attribut de partitionnement, de telle sorte que chaque partition contient les lignes correspondant à un intervalle donné. Par exemple une table R ayant une colonne Date comme clé de partitionnement, peut être fragmentée comme montré dans la Figure 3.10.

Partitionnement par valeurs discrètes : Dans ce type de partitionnement, chaque partition est explicitement définie par une liste de valeurs discrètes de l'attribut de fragmentation. Par exemple une table R ayant une colonne Région comme clé de partitionnement, peut être fragmentée comme montré dans la Figure 3.11.

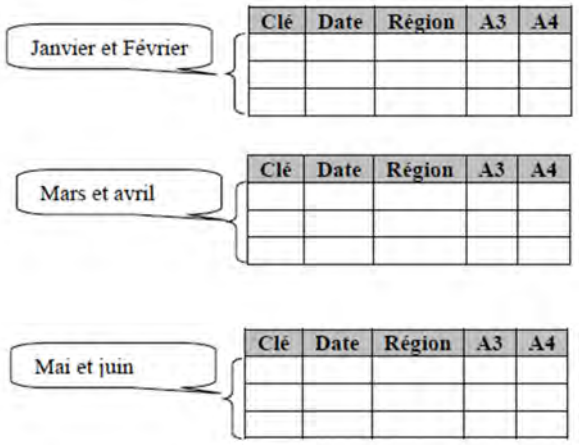


FIGURE 3.10 – Partitionnement par intervalle

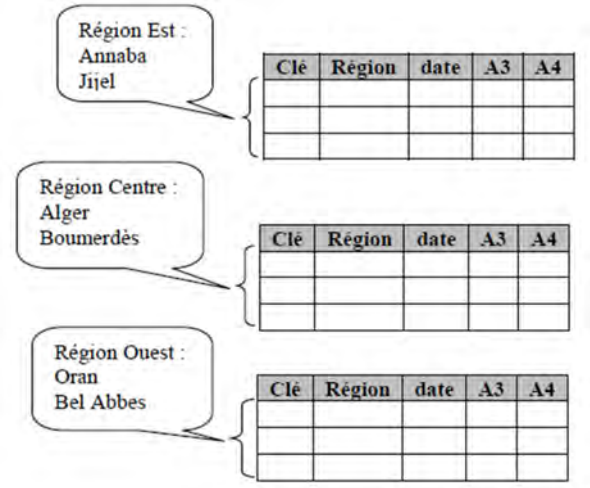


FIGURE 3.11 – Partitionnement par valeurs discrètes

Partitionnement par Hachage : Dans le partitionnement par hachage, les données sont placées dans des partitions selon une fonction de hachage appliquée sur l'attribut de partitionnement sélectionné (Figure 3.12).

Partitionnement composite Le partitionnement composite est une combinaison des méthodes de base de partitionnement de données. Une table est partitionnée par l'une des méthodes citées précédemment (par intervalle, par valeurs discrètes, ou par Hachage). Ensuite, chaque partition résultat est subdivisée en sous partitions en utilisant une autre méthode de partitionnement.

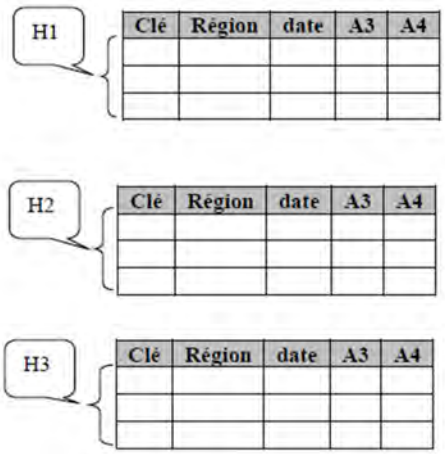


FIGURE 3.12 – Partitionnement par Hachage

3.2.3.2.2 La fragmentation horizontale dérivée La fragmentation horizontale dérivée est appliqué à une table S membre d'une table R. En d'autres mots, pour appliquer une fragmentation horizontale dérivée à S, il faut que la table S soit liée à la table R par une clé étrangère. Ainsi, S est appelée table membre, et ses fragments horizontaux sont dérivés par des opérations de semi-jointure entre S et les fragments horizontaux de R.
Formellement :
Si la table R est fragmentée en R_1,R_2,...,R_n, alors
$S_1 = S \ltimes R_1$; $S_2 = S \ltimes R_2$;... ; $S_n = S \ltimes R_n$
La Figure 3.13, montre un exemple de fragmentation horizontale primaire d'une table R, et une fragmentation horizontale dérivée d'une table S.

3.2.3.3 La fragmentation mixte

La fragmentation mixte (hybride) d'une table R, consiste à appliquer une fragmentation horizontale à R, ensuite appliquer à chaque fragment horizontal une fragmentation verticale, ou vice-versa (Figure 3.14). La fragmentation mixte est généralement appliquée pour atteindre des performances mieux que celles offerte par l'une des deux fragmentations : horizontale ou verticale.

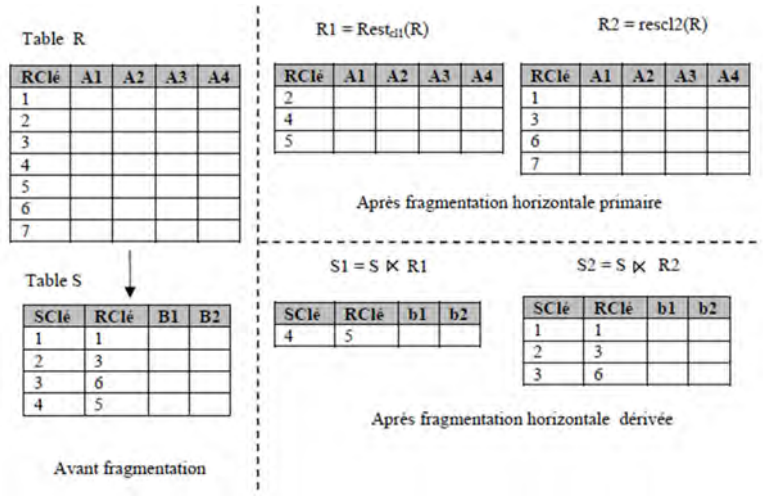


FIGURE 3.13 – La fragmentation horizontale dérivée

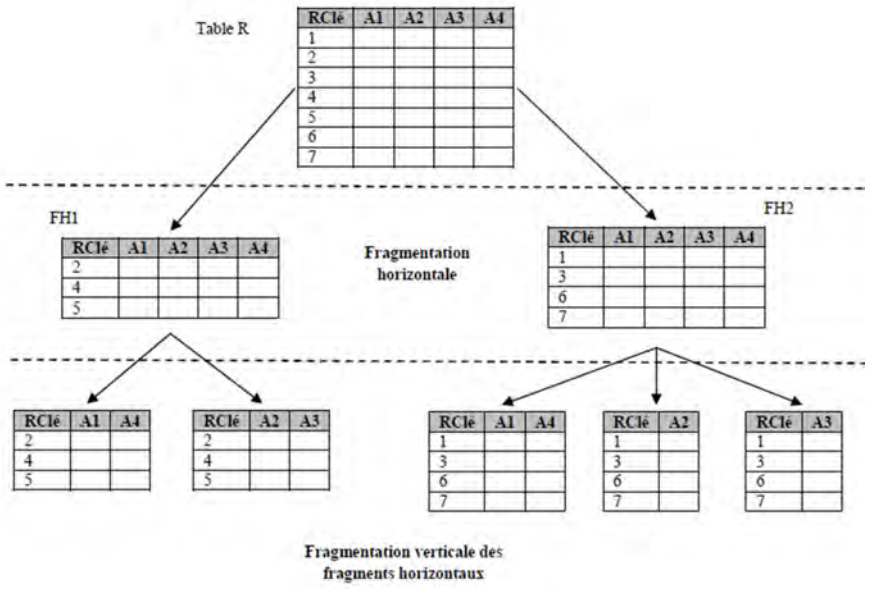


FIGURE 3.14 – La fragmentation mixte

3.2.3.4 Travaux sur la fragmentation

Trouver un schéma de fragmentation verticale permettant de minimiser considérablement le temps de réponse des requêtes, constitue un problème assez complexe (Section 3.2.3.1). Afin de résoudre ce problème, plusieurs approches de fragmentation verticale ont été proposées. Parmi ces travaux, nous citons celui de Bouakkaz et al. (2012) et Jong Wook and Sae-Hong (2016).

Dans (Bouakkaz et al., 2012), les auteurs ont développé une approche de fragmentation verticale basée sur les motifs fréquents en utilisant l'algorithme FP-Max (Maximal Frequent Itemsets). Afin de générer un schéma de fragmentation verticale optimal, les auteurs procèdent suivant les trois étapes suivantes : (i) l'extraction des motifs fréquents, (ii) la génération des fragments verticaux, et (iii) la sélection du schéma de fragmentation optimal. À partir d'une charge de requêtes avec leur fréquences d'accès et un support minimal prédéfini, la première étape consiste à découvrir l'ensemble des motifs fréquents en utilisant l'algorithme FP-Max. Ces motifs sont des combinaisons d'attributs qui ont un support supérieur au support minimal. Dans la deuxième étape, les motifs fréquents générés précédemment sont utilisés pour construire des schémas de fragmentation verticaux possibles contenant des partitions disjointes d'attributs. Dans la dernière étape, le coût de chaque schéma de fragmentation est calculé sur la base d'un modèle de coût. Le schéma ayant le coût inférieur est sélectionné pour partitionner l'entrepôt de données.

Dans un système de stockage orienté colonnes d'un entrepôt de données (c.-à-d. chaque attribut est enregistré comme étant une partition appelée partition single-colonne), Jong Wook and Sae-Hong (2016) ont proposé de grouper les attributs fréquemment accédés par les requêtes dans une même partition appelée partition multi-colonne. Ainsi, si les colonnes c_1 et c_2 sont accédées fréquemment, alors stocker c_1 et c_2 dans la même partition permet de réduire le nombre d'entrées/sorties lors du traitement des requêtes utilisant c_1 et c_2, ce qui par conséquent améliore leurs performances. Afin de générer les meilleures partitions multi-colonne, les auteurs procèdent en deux phases. Dans la première phase, ils construisent la matrice d'usage (mxn), où m représentes le nombre de requêtes de la charge et n correspond au nombre d'attributs (colonnes) dans la table T à fragmenter verticalement. Une fois la matrice d'usage construite, ses colonnes sont groupées en partitions en utilisant le clustering agglomératif basé sur la similarité cosinus calculée entre chaque deux colonnes. Les clusters obtenus dans cette étape, forment l'ensemble R des partitions multi-colonnes candidates utilisées par la suite pour sélectionner la meilleure partition. À partir de l'ensemble R, les auteurs ont généré l'ensemble de touts les configurations de partition multi-colonnes possible noté P. Chaque configuration est évaluée, et celle minimisant le coût d'accès de l'ensemble des requêtes sur la table T sous la contrainte de limite de stockage est retenue comme schéma de fragmentation

verticale.
Cependant, due à la complexité de la fragmentation verticale, la fragmentation mixte, qui utilise une fragmentation verticale, a reçu peu d'intérêt dans les entrepôts. Parmi les travaux existant nous citions (ZIYATI et al., 2008).

3.3 Avantages et inconvénients des techniques d'optimisation

La minimisation du temps de traitements des requêtes, constitue le principal avantage des travaux que nous venons d'exposer jusqu'ici sur les indexes, les vues matérialisées, et la fragmentation verticale des données. Toutefois, malgré cet avantage, ces techniques ont les inconvénients suivants :

- L'espace de stockage additionnel pour stocker les index, les vues matérialisées, et les fragments verticaux.
- Difficulté de la tâche de l'administration, particulièrement lorsque le nombre d'index, et/ou des vues matérialisées est très élevé.
- La reconstruction d'une table fragmentée verticalement, nécessite l'utilisation des opérations de jointures entre les fragments verticaux.
- La fragmentation verticale de l'entrepôt de données ne concerne que les tables dimension. Cette fragmentation n'a pas d'effet considérable sur l'optimisation des performances des requêtes, puisque la table de faits n'est pas fragmentée.

Les inconvénients cités ci-dessus ont des impacts négatifs sur les performances de l'entrepôt de données en général, et sur le les performances des requêtes décisionnelles en particulier. Comme alternative aux indexes, aux vues matérialisées, et à la fragmentation verticale, la fragmentation horizontale de l'entrepôt de données permet d'optimiser les performances des requêtes sous aucune contrainte d'espace de stockage. En plus, elle offre les possibilités et les avantages suivants (Ponniah, 2011) :

- Définir des indexes au niveau de chaque fragment horizontal,
- Construire des vues matérialisées de tailles petites à partir des partitions horizontales,
- Fragmenter les vues matérialisées existantes,

- La sauvegarde et la récupération des partitions de données (les fragments) réduit le temps d'arrêt de l'utilisation de l'entrepôt de données. En plus, une partition tout en entière peut être mise hors service pour effectuer des tâches de maintenances
- Préserver les données contre la corruption,
- Limiter le nombre d'accès inutiles aux données,
- La reconstruction d'une table fragmentée horizontalement utilise des opérations d'union entre les fragments. Ces opérations sont moins coûteuse en temps et en espace mémoire comparées aux opérations de jointure utilisées dans la fragmentation verticale.

3.4 Conclusion

Dans ce chapitre, nous avons présenté les différentes techniques d'optimisation de performances dans les entrepôts de données. Nous avons détaillé quelques travaux concernant les index, les vues matérialisées, et la fragmentation verticale. À la fin de ce chapitre, nous avons présenté les avantages de la fragmentation horizontale par rapport aux autres techniques d'optimisation. En prenant en considération ces avantages, nous consacrons le chapitre qui suit à l'étude et à l'analyse des travaux de fragmentation horizontale proposés dans les entrepôts de données.

Chapitre 4

Étude des techniques de fragmentation horizontale dans les entrepôts de données

Sommaire

4.1 Introduction

La fragmentation horizontale des données, a été largement étudiée dans les bases de données distribuées avant d'être utilisée dans les entrepôts de données. En effet, Ozsu and Valduriez (1991) ont défini deux versions de la fragmentation horizontale : la fragmentation horizontale primaire et la fragmentation horizontale dérivée.
Dans la fragmentation horizontale primaire, une table relationnelle T_i est fragmentée horizontalement sur la base des prédicats de sélection utilisés dans les clauses *where* des requêtes interrogeant T_i. Ainsi, dans ce type de fragmentation, les auteurs procèdent selon les étapes suivantes :

1. L'extraction de l'ensemble des prédicats de sélection simple noté Pri à partir de l'ensemble des requêtes interrogeant la table T_i. Un prédicat de sélection simple p_j est défini par $p_j = A_i\ \theta\ valeur$, avec $\theta\ \in \{=, <, >, \neq, <=, >=\}$, et $valeur$ appartient au domaine de l'attribut A_i de la table T_i.
2. La génération d'un ensemble complet et minimal de prédicats de sélection. Dans cette étape l'ensemble Pr_i est utilisé comme entrée de l'algorithme COM_MIN (Ozsu and Valduriez, 1991), qui procède itérativement sur l'ensemble Pr_i et génère un autre ensemble de prédicats de sélection noté Pr'_i
3. La construction d'un ensemble de tous les minterm noté $M_i = \{m_{i1}, m_{i2}, ..., m_{iz}\}$ à partir de l'ensemble $Pr'_i = \{p_{i1}, p_{i2}, ..., p_{im}\}$. L'ensemble M_i est défini comme suit :

$$M_i = \left\{ m_{ij} / \ m_{ij} = \bigwedge_{p_{ik} \in Pri'} p^*_{ik} \right\},\ 1 \leq k \leq m,\ 1 \leq j \leq z$$

   et $p^*_{ik} = p_{ik}$ ou $p^*_{ik} = \neg p_{ik}$
4. L'élimination des minterm de prédicats $m_{ij} \in M_i$ qui sont contradictoire selon l'ensemble d'implication défini à partir de l'ensemble des prédicats Pr'_i
5. Finalement chaque minterm restant dans l'ensemble M_i, correspond à un fragment horizontal f_{ij} défini par $f_{ij} = \sigma_{m_{ij}}(T_i)$ (σ est l'opération de projection.

Il est à noter que si le nombre de prédicats de sélection dans l'ensemble Pr'_i est n, alors le nombre possible de minterm de prédicats défini sur Pr'_i est 2^n.
Dans le cas où un lien existe entre la table T_i et une table S de sort que T_i est la table propriétaire du lien et S est la table membre (c.-à-d. S contient une clé

étrangère de T_i), alors la fragmentation horizontale dérivée de S consiste à effectuer des opérations de semi-jointure entre la table S et chaque fragment de la table T_i. Formellement, chaque fragment horizontal S_{ij} est défini par : $S_{ij} = S \ltimes f_{ij}$.

À partir des définitions de la fragmentation horizontale primaire et de la fragmentation horizontale dérivée présentées ci-dessus, et des avantages offerts par la fragmentation horizontale cités précédemment, la communauté des chercheurs dans les entrepôts de données, ont constaté que l'application de ces deux type de fragmentation peut être très favorable pour l'optimisation des performances des requêtes décisionnelles. Car, dans un entrepôt de données :

1. La table de faits est caractérisée par un grand nombre d'enregistrements,
2. Les prédicats de sélection utilisés par les requêtes interrogeant l'entrepôt de données sont tous définis sur les tables de dimension,
3. La table de faits est membre de chacune des tables dimension (selon le schéma en étoile).

Dans la section suivante, nous étudions en détail les différents travaux sur la fragmentation horizontale dans les entrepôts de données.

4.2 Travaux sur la fragmentation horizontale dans les entrepôts de données

Dans (Noaman and Barker, 1999), les auteurs ont proposé de répartir un entrepôt de données dans un environnement distribué. Ils ont proposé de répliquer les tables dimension dans chaque site de l'architecture distribuée, et de distribuer la table de faits après la fragmentation horizontale de son gros volume de données. Ainsi, pour générer les fragments de la table de faits, les auteurs ont procédé selon les étapes suivantes :

1. Pour chaque dimension D_i :
   (a) Extraire l'ensemble de prédicats simples à partir de la charge de requêtes,
   (b) À partir des attributs des prédicats de sélection, identifier celui ayant le plus haut niveau dans hiérarchie de la dimension D_i,

(c) Supprimer tous les prédicats pour lesquels leurs attributs ont un niveau inférieur à celui de l'attribut identifié précédemment. La suppression de tels prédicats permet d'effectuer les opérations roll-up et drill-down des requêtes OLAP sur les mêmes fragments et de réduire le nombre d'opérations de jointures entre les fragments.

2. Pour chaque dimension, construire l'ensemble des minterm de prédicats à partir de l'ensemble des prédicats, raffiné dans l'étape précédente.
3. Effectuer l'union des ensembles des minterm de prédicats de la dimension.
4. À partir de l'ensemble global des minterm de prédicats des dimensions, générer les minterm de prédicats de la table de faits,
5. Finalement, pour chaque minterm de prédicats de faits, générer un fragment horizontal de la table de faits.

Bellatreche and Boukhalfa (2005) : ont proposé une approche de fragmentation horizontale utilisant l'algorithme génétique, qui est un algorithme itératif de recherche de solution optimale parmi une population de solutions. À partir d'une population initiale de solutions optimisant le mieux la fonction sélective (individus mieux adaptés), l'algorithme génétique répète itérativement les opérations de sélection, de croisement, et de mutation pour générer une nouvelle population de solutions. À chaque génération, les solutions de la nouvelle population sont évaluées par une fonction sélective (une fonction de coût) qui permet de déterminer les meilleures solutions. Pour exploiter l'algorithme génétique dans la fragmentation de l'entrepôt de données (la sélection d'un schéma de fragmentation horizontale optimal considéré comme un individu ou une solution dans le contexte génétique), les auteurs ont commencé par une étape préparatoire englobant les étapes suivante :

1. L'extraction de l'ensemble des prédicats de sélection simples utilisés par les requêtes,
2. Le groupement des prédicats de sélection en sous ensemble par dimension,
3. L'élimination des tables de dimension n'ayant aucun prédicats de sélection,
4. La génération d'un ensemble complet et minimal des prédicats simples pour chaque dimension.

À partir de l'ensemble complet et minimal des prédicats simples, les auteurs ont proposé un codage spécifique pour représenter chaque solution. Ce codage est basé sur le partitionnement des domaines de définition des attributs de fragmentation en sous domaines déterminés par les prédicats simples (Figure 4.1).

Comme montré par la Figure 4.1, le domaine de l'attribut Age est partitionné en

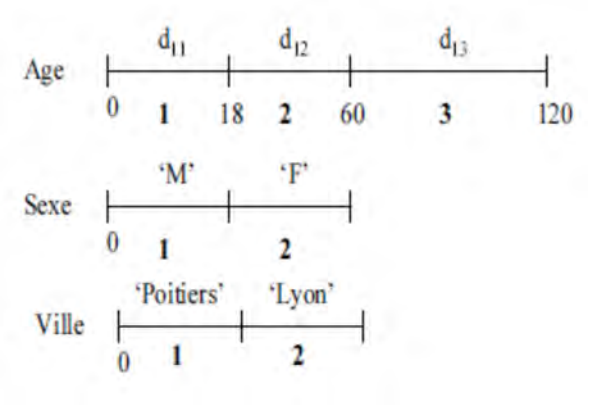


FIGURE 4.1 – Sous domaines des attributs de fragmentation

trois sous domaines, définis respectivement par les prédicats simples (p_1 : $Age \leq 18$), (p_2 : $Age \geq 60$), et (p_3 : $18 < Age < 60$). À chaque partition est attribué un nombre entre 1 et n où n est le nombre de sous domaines. Suivant le même principe, les domaines des attributs $Sexe$ et $Ville$ sont partitionnés en sous-domaines. À partir des partitions de la Figure 4.1, un individu (un schéma de fragmentation) peut être codé comme le montre la Figure 4.2.

À partir de cette figure, on peut déduire que la fragmentation de l'entrepôt de

Age	2	1	2
Sexe	1	1	
Ville	1	2	

FIGURE 4.2 – Schéma de fragmentation

données ne se fera pas sur l'attribut $Sexe$, car tous les sous domaines de l'attribut ont la même valeur, contrairement à l'attribut $Ville$ et à l'attribut Age qui sont utilisés lors de la fragmentation. Par exemple, pour l'attribut Age, les sous domaines d_{11} et d_{13} sont fusionnés, ce qui se traduit par deux prédicats de fragmentation, $Age \leq 18 \vee Age \geq 60$ et $18 < Age < 60$.

Basé sur ce codage, les opérations de mutation et de croisement sont appliquées facilement sur les individus d'une population engendrant ainsi des nouveaux schémas de fragmentation. Concernant la fonction d'évaluation d'un individu, les auteurs ont utilisé un pourcentage calculé sur la base de deux paramètres, à savoir : (i) le nombre de fragments généré par l'individu par rapport au nombre de fragments maximal autorisé (seuil), et (ii) la rapidité d'exécution des requêtes interrogeant l'entrepôt de données, calculée par une formule de coût permettant de calculer le nombre d'entrées/sorties nécessaire aux traitements des requêtes. Selon la solution retournée par l'algorithme génétique, chaque ensemble d'attributs de fragmentation est utilisé pour la fragmentation horizontale primaire de la dimension correspondante. Concernant la table de faits, ses fragments sont dérivés par des opérations de semi-jointure entre la table des faits et chaque combinaison des fragments des dimensions.

Bellatreche et al. (2006) : ont proposé de combiner l'algorithme génétique avec l'algorithme de recuit simulé lors de la recherche d'un schéma de fragmentation optimal. La combinaison de ces deux algorithme permet d'éviter la convergence de l'algorithme génétique vers un optimum local (c.-à-d. un schéma de fragmentation offrant des meilleures performances par rapport à son voisinage), en autorisant le déplacement vers des solutions mauvaises permettant d'explorer une large partie de l'espace des solutions à la recherche d'un schéma de fragmentation horizontale offrant le maximum de performances des requêtes (c.-à-d. la recherche d'un optimum global). Après l'application de l'algorithme de fragmentation basé sur l'algorithme génétique proposé dans (Bellatreche and Boukhalfa, 2005), le schéma de fragmentation horizontale retourné est utilisé comme solution initiale de l'algorithme de recuit simulé. Afin d'améliorer cette solution initiale, l'algorithme de recuit simulé procède selon les étapes suivantes :

1. La solution initiale reçoit la solution retournée par algorithme génétique, l'énergie initiale E reçoit le coût de la solution initiale calculée par la même formule utilisée par l'algorithme génétique, et le paramètre température est initialisé à la valeur 400,
2. Appliquer une transformation à la solution initiale pour obtenir une autre solution, et calculer la valeur de son énergie E' par la formule de coût. Cette transformation consiste à modifier les valeurs des cellules de la solution en cours en les incrémentant ou décrémentant aléatoirement,

3. Si l'énergie de la nouvelle solution E' est inférieure ou égale à celle de solution en cours E, alors affecter la nouvelle solution à la solution en cours, sinon avec une probabilité $e^{(E'-E)/T}$ le déplacement vers la nouvelle solution est accepté,
4. Répéter 100 fois l'étape 2 et l'étape 3,
5. Décrémenter la température $T = T$- 2,
6. Si aucune amélioration (minimisation) dans la valeur de l'énergie n'est enregistrée après 21000 itération alors arrêter l'algorithme, et la solution courante est retenue comme schéma de fragmentation optimal, Sinon allez à l'étape 2.

À partir de la solution finale retournée par l'algorithme de recuit simulé, chaque ensemble d'attributs de fragmentation est utilisé pour la fragmentation horizontale primaire de la dimension correspondante. Ensuite, la table de faits est fragmentée par des opérations de semi-jointure entre la table des faits et chaque combinaison des fragments des dimensions.

Bellatreche et al. (2008) : ont proposé de résoudre le problème d'optimisation des requêtes décisionnelles par l'usage de l'algorithme Hill Climbing. Ce dernier est utilisé pour explorer l'espace des solutions possibles (c.-à-d. les schémas de fragmentation) à la recherche de celle offrant les meilleures performances des requêtes (le schéma de fragmentation quasi-optimal). Le Hill Climbing est un algorithme itératif, qui, commençant par une solution arbitraire, tente de trouver une solution meilleure optimisant une certaine fonction de coût. Ainsi, à chaque itération, des changements sont appliqués à la solution courante pour arriver à une nouvelle solution. Si la solution nouvelle est meilleure, alors les changements sont appliqués à nouveau sur cette dernière jusqu'à ce qu'aucune amélioration ne peut être trouvée. Basé sur le principe présenté ci-dessus et en reprenant le même codage (un tableau multidimensionnel) de la Figure 4.2 pour représenter un schéma de fragmentation, les auteurs ont procédé comme suit :

1. Générer une solution initiale en utilisant le principe d'affinité entre les sous domaines. L'affinité entre deux sous-domaines est égale à la somme des fréquences d'accès des requêtes accédant simultanément à ces deux sous-domaines,
2. Appliquer la fonction *Merge* et la fonction *Split* à la solution courante. La fonction *Merge* fusionne deux sous domaines en un seul, et la fonction *Split* éclate un sous domaine en deux sous domaines. Ces deux fonctions permettent

la génération de nouvelles solutions,

3. Répéter l'épate 2, jusqu'à ce que le temps d'exécution des requêtes n'enregistre aucune amélioration.

Ainsi, la solution retournée par l'algorithme Hill Climbing est utilisée pour fragmenter les tables dimension. Ensuite les fragments de faits sont dérivés à partir des fragments des tables dimension.

Dimovski et al. (2010) : ont proposé une approche de fragmentation horizontale d'un entrepôt de données utilisant l'algorithme génétique. À partir d'un ensemble de schéma de fragmentation appelé population de chromosomes, l'algorithme génétique explore cet ensemble à la recherche d'un schéma de fragmentation optimal. L'exploration de cet espace est réalisée par des opérations de sélection, de croisement, et de mutation appliquées aux chromosomes pour générer de nouvelles populations. Cependant, et contrairement au codage d'un schéma de fragmentation horizontale proposé dans (Bellatreche and Boukhalfa, 2005), les auteurs ont proposé un nouveau codage basé sur la notion d'abstraction de prédicats de sélection. Étant donné une dimension D_i et un ensemble P_i=$\{p_1, ..p_m\}$ contenant m prédicats de sélection définies sur D_i, les auteurs définissent : (i) le domaine abstrait de D_i par rapport à P_i, comme étant un ensemble de vecteurs binaires de longueur m, et (ii) la fonction d'abstraction, qui pour chaque vecteur binaire $(v_1, ..., v_m)$ détermine l'ensemble des tuples t de D_i qui satisfont la conjonction $v_1.p_1$ et...et $v_m.p_m$ avec $(0.p = \neg p)$ et $(1.p = p)$. Selon cette notion d'abstraction de prédicats, chaque vecteur binaire défini un fragment horizontal $Di(v_1, ..., v_m)$, et la dimension D_i est fragmentée en 2^m fragments.

Cependant, pour un schéma d'entrepôt de données $(F, D_1, ..., D_k)$ composé de k dimensions et une table de faits F, les auteurs ont représenté un fragment en étoile par un vecteur binaire composé de k vecteurs binaires chacun correspondant à une dimension D_i (Figure 4.3).

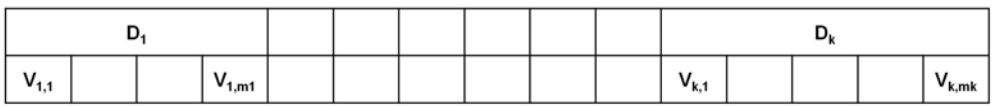


FIGURE 4.3 – Schéma de fragmentation -Individu-

Pour un entrepôt de données fragmenté horizontalement en N fragments, son schéma de fragmentation est représenté par un chromosome composé de N vecteurs binaires dont chacun correspond à un fragment horizontal. Ainsi, à partir d'une population de 200 chromosomes, l'algorithme génétique applique les opérations de croisement de mutation afin de trouver le chromosome (le schéma de fragmentation composé de N fragments) optimisant le mieux le coût d'exécution de l'ensemble des requêtes.

Bellatreche (2012) : a proposé une stratégie de fragmentation dérivée de la table de faits basée sur la sélection des tables dimension pertinentes. Pour un entrepôt de données composé de d tables de dimension D_1,...,D_d, il existe 2^{d-1} possibilités de fragmenter la table de faits. En d'autre termes, les fragments de faits peuvent être dérivés à partir des fragments de D_1, ou de D_2,..., ou de D_d, ou de (D_1 et D_2),..., ou de (D_1 et D_2 et...et D_d). Ainsi, pour sélectionner les tables de dimension à utiliser pour fragmenter la table de faits, les auteurs ont proposé d'utiliser l'une des techniques suivantes :

- Sélectionner les tables de dimensions les plus larges,
- Sélectionner les tables de dimensions ayant plus de fréquence d'utilisation,
- Sélectionner les tables de dimensions ayant un minimum de partage avec la table de faits (le nombre moyen de tuples de la table de faits qui font références aux mêmes tuples de la table dimension).

Une fois les tables de dimension pertinentes sélectionnées, les attributs de fragmentation de chaque dimension sont identifiés. Le partitionnement des domaines de définitions de ces attributs en sous domaine est utilisé pour représenter un schéma de fragmentation horizontale. Ainsi, partant d'une représentation initiale d'un schéma (appelée aussi solution), les auteurs ont utilisé l'algorithme Hill Climbing, qui itérativement améliore la solution jusqu'à ce qu'aucune amélioration du coût d'exécution des requêtes ne soit faite.

Thenmozhi and Vivekanandan (2014) : ont proposé deux nouvelles techniques de fragmentation horizontale d'un entrepôt de données. En effet, en s'inspirant des techniques de fragmentation horizontale utilisant l'algorithme génétique et ceux utilisant l'algorithme Hill Climbing, et en reprenant la même représentation (un tableau multidimensionnel) d'un schéma de fragmentation (Figure 4.2), les auteurs ont :

1. Combiné l'algorithme Hill Climbing avec l'algorithme génétique.

2. Combiné l'algorithme de recherche tabou avec l'algorithme génétique.

L'objectif principal des deux techniques est de trouver un schéma de fragmentation horizontale optimale de l'entrepôt de données.

Gacem and Boukhalfa (2014) : ont traité l'optimisation des performances d'une charge de travail composée d'un très grand nombre de requêtes. Ils ont classifié les requêtes en fonction de leur similarité d'usage par les prédicats de sélection. L'algorithme K-Means utilisé pour générer les K classes de requêtes, affecte à chaque classe un centre représenté par un vecteur de prédicats de sélection. Parmi les prédicats constituant le centre de chaque classe de requêtes, les auteurs ont choisi ceux ayant des poids supérieurs à un seuil donné. Les prédicats de sélection choisis sont utilisés pour écrire une nouvelle requête représentant la classe. Enfin, à partir de la nouvelle charge de travail composé des K requêtes correspondante au K classes, les auteurs ont utilisé l'algorithme développé dans (Bellatreche et al., 2008) pour fragmenter horizontalement l'entrepôt de données.

Bouchakri et al. (2014) : ont proposé un algorithme incrémental de fragmentation horizontale d'un entrepôt de données relationnel. Ils ont exploité le codage d'un schéma de fragmentation basé sur le partitionnement des domaines d'attributs de fragmentation (Figure 4.2) pour définir des opérateurs de transformation. Ces opérateurs sont appliqués à un schéma de fragmentation existant pour le transformer (c.-à-d. fusionner ou éclater des sous domaines d'attributs) en un autre schéma de fragmentation en fonction de la nouvelle requête ajoutée à la charge.

Zamanian et al. (2015) ont développé une approche de partitionnement (fragmentation) par référence basée sur les prédicats de sélection dont l'objectif est d'optimiser les performances des requêtes analytiques interrogeant un large entrepôt de données dans un environnement parallèle. Parmi les tables de l'entrepôt de données, ils sélectionnent une table (appelée table référencée) à fragmenter par l'une des techniques de partitionnement connues, telle que, le partitionnement par hachage, le partitionnement par intervalle... etc. (voir section 3.2.3.2.1). Ensuite, ils dérivent les partitions des autres tables (appelées tables de référence) à partir des partitions de la table référencée en utilisant les prédicats de jointure appelés aussi prédicats de partitionnement. Cependant, ce type de fragmentation horizontale peut dupliquer des tuples dans les fragments des tables de référence. Ainsi, pour minimiser la redon-

dance des données et maximiser leur localité lors du traitement parallèle des requêtes analytiques, les auteurs ont proposé deux algorithmes de fragmentation horizontale, l'un est basé sur le schéma de l'entrepôt de données et l'autre est basé sur la charge de requêtes.

Toumi et al. (2015) : ont constaté que les approches de fragmentation des entrepôts de données existantes ont prouvé leurs efficacité pour des charges de travail composées d'un petit nombre de requêtes utilisant un nombre assez réduit de prédicats de sélection. Ainsi, pour optimiser les performances d'un grand nombre de requêtes utilisant un grand nombre de prédicats de sélection, les auteurs ont proposé une nouvelle approche de fragmentation horizontale d'entrepôt de données relationnel. Les principales phases de l'approche proposée sont :

1. L'extraction des prédicats de sélection à partir de la charge des requêtes, et le calcul des coefficients d'attraction entre les prédicats de sélection en utilisant l'index de Jaccard (Tan et al., 2005).
2. La classification des prédicats de sélection basée sur leurs coefficients d'attraction en utilisant l'algorithme de classification hiérarchique Ward (Ward, 1963),
3. La sélection du meilleur schéma de fragmentation en utilisant l'algorithme Particle Swarm optimization (Jarboui et al., 2007). Dans cette phases, les auteurs utilisent le même codage d'un schéma de fragmentation horizontale (Figure 4.2) ainsi qu'un modèle de coût pour calculer le nombre d'entrées/sorties nécessaire à l'exécution d'une requête.

Ghorbel et al. (2016) : ont proposé de fragmenter horizontalement un entrepôt de données relationnel et de distribuer les fragments dans une architecture décentralisée. La phase de fragmentation de l'entrepôt de données est composée des étapes suivantes :

1. La génération d'un ensemble complet et minimal de prédicats de sélection en utilisant l'algorithme COM_MIN (Ozsu and Valduriez, 1991). Cet ensemble englobe tous les prédicats utilisés par la charge de requêtes,
2. La codification des prédicats par une matrice d'usage par site,

3. Le partitionnement, par l'algorithme K-Means, de l'ensemble des prédicats de sélection en classes selon leur utilisation par les sites de l'architecture distribuée.
4. La réduction du nombre de prédicats de sélection par : (i) l'élimination de ceux utilisés par tous les sites, (ii) la fusion de ceux utilisés dans le même site et qui sont définis sur le même attribut,
5. La fragmentation de l'entrepôt de données en K fragments. Chaque fragment est défini par la conjonction des prédicats de sélection de chaque site après l'étape de réduction.

Matalqa and Mustafa (2016) : ont mené une étude expérimentale sous MS SQL-Server de deux techniques de fragmentation horizontale, à savoir, la fragmentation par intervalle (range partitioning) et la fragmentation par valeurs discrètes (list partitioning). Les résultats obtenus, ont monté que les deux techniques de fragmentation offrent à peu prés le même taux d'amélioration de performances pour la même charge de requêtes. Cependant, la taille de la base de données (18080 et 19072 enregistrements) ainsi que le nombre de requêtes (12 requêtes) utilisés lors de l'évaluation de ces deux techniques de fragmentation, sont loin d'être le cas dans les entrepôts de données.

4.3 Résumé des travaux de fragmentation

L'étude précédente, nous a permis de résumer les différents travaux de fragmentation horizontale dans le Tableau 4.1, en prenant en compte les points suivants :

- Type de fragmentation,
- Nombre de fragments,
- Requête non bénéficiant de la fragmentation,
- Métaheuristiques,
- Règle de validation de la fragmentation (disjonction, complétude),
- Réécriture et traitement des requêtes (transparence d'accès aux données),
- Nombre de fragments généré dans les expérimentations,
- Taille de la table de faits.

Tableau 4.1 – Tableau récapitulatif des travaux de fragmentation horizontale

Travaux	**Type de fragmentation**	**Nombre de fragments**	**Requête non bénéficiant de la fragmentation**	**Méta-heuristique**	**Règle de validation de la fragmentation (disjonction, complétude)**	**Réécriture et traitement des requêtes (transparence d'accès aux données)**	**Nombre de fragments généré dans les expérimentations**	**Taille de la table de faits**
(Noaman and Barker, 1999)	Fragmentation horizontale primaire/dérivée	Réduction du nombre des prédicats de sélection	Non prisent en compte	Non	Oui	Non	Aucune expérimentation	Aucune expérimentation
(Bellatreche and Boukhalfa, 2005)	Fragmentation horizontale primaire /dérivée	Seuil 500-8000	Non prisent en compte	Algorithme génétique (taille population+nombre de génération)	Oui	Non	4000	24786000 (APB-1 benchmark)
(Bellatreche et al., 2006)	Fragmentation horizontale primaire /dérivée	Seuil	Non prisent en compte (11 requêtes parmi15)	Combinaison algorithme génétique et algorithme recuit simulé (température)	Oui	Non	2000 (2785 I/O)	24786000 (APB-1 benchmark)
(Bellatreche et al., 2008)	Fragmentation horizontale primaire /dérivée	Seuil 100-500	Non prisent en compte	Algorithme Hill Climbing	Oui	Non	200	24786000 (APB-1 benchmark)

(Dimovski et al., 2010)	Fragmentation horizontale primaire /dérivée	Seuil	Non prisent en compte	Non	Non	Non	64	1.25×10^9 (Propre jeux de données)
(Bellatreche, 2012)	Fragmentation horizontale primaire /dérivée avec critères de sélection de tables dimension à fragmenter	Seuil 10-100	Non prisent en compte	Algorithme Hill Climbing	Oui	Non	10	33323400 (APB-1 benchmark)
(Thenmozhi and Vivekanandan, 2014)	Fragmentation horizontale primaire /dérivée	Seuil 4000	Non prisent en compte	Algorithme Génétique + Algorithme Hill Climbing / Algorithme Génétique + Algorithme Recherche tabou	Oui	Non	3000/2500	24786000 (APB-1 benchmark)
(Bouchakri et al., 2014)	Fragmentation horizontale primaire /dérivée	Seuil	Non prisent en compte	Algorithme génétique (taille popula-tion+nombre de génération)	Oui	Non	100	24786000 (APB-1 benchmark)

(Gacem and Boukhalfa, 2014)	Fragmentation horizontale primaire /dérivée	Seuil	Non prisent en compte-Prise en compte d'un grand nombre requêtes (500-1000)	Algorithme génétique (taille popula-tion+nombre de génération)	Oui	Non	1000	24786000 (APB-1 benchmark)
(Toumi et al., 2015)	Fragmentation horizontale primaire /dérivée	Seuil	Non prisent en compte	Parcical swarme	Oui	Non	De 400 à 1000	24786000 (APB-1 benchmark)
(Zamanian et al., 2015)	Fragmentation horizontale par référence basé sur les prédicats	/	Non prisent en compte	Non	Oui	Oui	/	TPC-H benchmark
(Ghorbel et al., 2016)	Fragmentation horizontale primaire /dérivée -Avec distribution des fragments sur les sites	/	Non prisent en compte	Non	Non respectées	Non	30	33323400 (APB-1 benchmark)

4.4 Conclusion

Dans ce chapitre, nous avons présenté les différents travaux de fragmentation horizontale primaire et dérivée, proposés dans les entrepôts de données. L'étude détaillée de ces travaux, résumée dans le Tableau 4.1, nous a permis de soulever des inconvénients majeurs.
Dans le chapitre qui suit, nous discutons ces inconvénients, et nous proposons une nouvelle formalisation du problème de sélection de schéma de fragmentation horizontale.

Chapitre 5

Problème d'optimisation de performances par la fragmentation horizontale : nouvelle formalisation

Sommaire

5.1 Introduction

Les techniques de fragmentation horizontale présentées dans le chapitre 4, ont contribué à la réduction du temps de traitement des requêtes décisionnelles. Cependant, nous avons constaté que toutes ces approches suivent la même démarche, à savoir :

1. Elles commencent par une fragmentation horizontale primaire d'une ou de plusieurs tables de dimension, suivie
2. D'une fragmentation horizontale dérivée de la table de faits.

La seule différence majeure, réside dans la fragmentation horizontale primaire appliquée aux tables dimension. Puisque, les fragments de la table de faits sont toujours obtenus par des opérations de semi-jointure entre les fragments horizontaux des tables dimension et la table de faits. Ce constat, nous a permis de soulever quatre inconvénients qui ont un impact important sur l'amélioration des performances des requêtes. Nous discutons ces inconvénients dans la section ci-dessous.

5.2 Inconvénients des techniques de fragmentation horizontale existantes

5.2.1 L'explosion du nombre de fragments horizontaux

Dans les approches de fragmentation horizontale existantes, le nombre de fragments de faits noté K est toujours calculé par la Formule 5.1 ((Bellatreche and Boukhalfa, 2005)).

$$K = \prod_{i=1}^{m} b_i \tag{5.1}$$

où b_i représente le nombre de fragments horizontaux de la dimension D_i et m correspond au nombre de dimensions fragmentées.
Selon la Formule 5.1, le nombre de fragment croit rapidement et peut atteindre des valeurs très grande en fonction du nombre de dimensions et de leurs nombre de fragments. Par exemple, si l'entrepôt de données est composé de 04 dimensions fragmentées respectivement en 7, 5, 6, et 10 fragments horizontaux, alors la table de faits sera fragmentée en 2100 fragments (7 x 5 x 6 x 10 = 2100, selon la Formule 5.1). Ainsi, maintenir un tel nombre de fragments sera une tâche difficile et coûteuse pour

l'administrateur de l'entrepôt de données. Pour résoudre ce problème, les approches de fragmentation horizontale existantes ont proposé divers techniques pour contrôler l'augmentation du nombre de fragments de faits.

Noaman and Barker (1999) : ont proposé de réduire le nombre de prédicats de sélection utilisés pour fragmenter les tables de dimension. Pour chaque dimension Di, les prédicats de sélection définis sur Di dont les attributs ont un niveau inférieur dans la hiérarchie de Di ont été éliminés.

Bellatreche (2012) : a proposé un algorithme de sélection des dimensions pertinentes utilisées pour fragmenter la table de faits. Avant d'entamer le processus de fragmentation, l'auteur a proposé de sélectionner les tables de dimension selon l'un des critères suivants : (i) les tables dimension assez larges en nombre de tuples, (ii) les tables dimension les plus fréquemment utilisées par les requêtes, (iii) les tables de dimension qui partagent un minimum de tuples avec la table de faits. Selon cette stratégie, le nombre de dimensions utilisées dans la fragmentation a été réduit. Par conséquent, et selon la Formule 5.1, cela a permis de réduire le nombre de fragments de faits.

Gacem and Boukhalfa (2014) : ont proposé de réduire le nombre de requêtes de la charge de travail. Ainsi, les prédicats de sélection utilisés par les requêtes supprimées de la charge ne vont pas participer à la fragmentation des tables dimension. Ce qui leurs a permis de réduire le nombre de fragments des tables dimension, ainsi que le nombre de fragments de faits.

Ghorbel et al. (2016) : ont proposé de grouper, en utilisent l'algorithme K-Means, les minterms de prédicats de dimension en K classes en fonction de leur répartition géographique dans une architecture décentralisée. Ainsi, au lieu de générer un fragment de fait pour chaque minterm de prédicats de dimension, ils génèrent K fragments de faits dont chacun correspondant à une classe de minterms de prédicats de dimension.

Cuzzocrea et al. (2009); Kechar and Nait-Bahloul (2014) : ont utilisé l'algorithme K-Means pour générer K fragments de faits.

Bellatreche and Boukhalfa (2005); Bellatreche et al. (2006, 2008); Dimovski et al. (2010); Bouchakri et al. (2014); Thenmozhi and Vivekanandan (2014); Toumi et al. (2015) : ont utilisé un nombre maximal (un seuil W) de fragments de faits fixé par l'administrateur de l'entrepôt de données. Ainsi,

tout schéma de fragmentation composé d'un nombre de fragments supérieur à W sera exclu de l'ensemble des schémas de fragmentation potentiels. Cependant, Bellatreche and Boukhalfa (2005) ont montré qu'un grand nombre de fragments de faits contribue significativement à l'amélioration des performances des requêtes.

5.2.2 La non prise en compte des requêtes non bénéficiant de la fragmentation

Optimiser les performances de la totalité des requêtes (n requêtes) composant la charge de travail interrogeant l'entrepôt de données est le cas le plus souhaitable. Malheureusement, après une fragmentation horizontale de l'entrepôt de données, un nombre y de requêtes parmi les n requêtes, peuvent ne pas bénéficier de la fragmentation horizontale.

Bellatreche et al. (2000) ont observer que la fragmentation horizontale primaire et la fragmentation horizontale dérivée de l'entrepôt de données peut détériorer les performances :

1. Des requêtes contenant des prédicats de sélection ne participant pas dans la fragmentation primaire des tables dimension, et

2. Des requêtes ne contenant aucun prédicat de sélection.

Car, leur traitement nécessite la reconstruction complète de l'entrepôt de données fragmenté. Cette reconstruction est très coûteuse en temps et en espace mémoire, puisque elle est réalisée par plusieurs opération d'union et de jointure entre les fragments horizontaux. En conséquence, un nombre élevé de requêtes non bénéficiant de la fragmentation affecte négativement les performances globales de l'ensemble des requêtes.

Cependant, nous avons constaté qu'aucune solution pour traiter le cas des requêtes non bénéficiant de la fragmentation (par exemple : préserver leur performances après la fragmentation, réduire leur nombre... etc) n'a été proposée par les approches de fragmentation existantes.

5.2.3 Le nombre de fragments nécessaires au traitement des requêtes bénéficiant de la fragmentation

Le nombre de sous-schémas de fragmentation K (Formule 5.1) de l'entrepôt de données dépend du nombre b_i des fragments des dimensions D_i (i=1..m). Ce dernier est en fonction du nombre de prédicats de sélection utilisé pour fragmenter les tables de dimension. Cependant, les prédicats de sélection utilisés pour fragmenter les tables de dimension sont choisis parmi l'ensemble de tous les prédicats extraits à partir des requêtes. De ce fait, le nombre de sous schéma de fragmentation nécessaire (valide) au traitement d'une requête Q est déterminé en fonction du nombre de prédicats de sélection utilisé dans sa clause *where*. Bellatreche et al. (2008) ont montré que :

- Cas 1 : si le nombre de prédicats de sélection utilisé par une requête bénéficiant de la fragmentation est petit, la requête accède à un nombre important de sous schémas de fragmentation. Par conséquent, son temps de réponse ne bénéficie par d'une grande amélioration.
- Cas 2 : si le nombre de prédicats de sélection utilisé par une requête bénéficiant de la fragmentation est grand, le traitement de la requête nécessite l'accès à un nombre réduit de sous schéma de fragmentation (chargement de moins de données), ce qui améliore considérablement son temps de réponse.

Ainsi, malgré l'amélioration des performances globales de la charge de travail réalisée par les différentes approches de fragmentation horizontale primaire et dérivée, nous avons constaté qu'un nombre élevé de requêtes de type Cas 1 diminue significativement le taux de gain en performances globale de la charge de requêtes. Malheureusement, à notre connaissance, aucune approche n'a traité ce type de requêtes.

5.2.4 L'effet des métaheuristiques sur la sélection du schéma de fragmentation

Par une étude comparative entre les techniques de fragmentation horizontale combinant l'algorithme génétique avec le Hill Climbing, et l'algorithme génétique avec l'algorithme de recherche tabou, les auteurs dans (Thenmozhi and Vivekanandan, 2014) ont montré l'importance du choix du nombre de génération et la taille des populations. Les résultats de leurs expérimentations ont montré l'effet direct de

ces deux paramètres sur la qualité du schéma de fragmentation généré. Ainsi, les schémas de fragmentation générés par les techniques de fragmentation horizontale proposés dans (Bellatreche and Boukhalfa, 2005; Bellatreche et al., 2006, 2008; Dimovski et al., 2010; Bouchakri et al., 2014; Thenmozhi and Vivekanandan, 2014; Toumi et al., 2015), sont tous concernés par le bon choix des paramètres des méta-heuristiques utilisées.

Afin d'éviter les inconvénients cités ci-dessus, dans la section suivante :

1. Nous proposons une nouvelle formulation du problème d'optimisation de performances par la fragmentation horizontale,
2. Nous définissons cinq conditions que nous appelons conditions de quasi-optimalité, qu'un schéma de fragmentation horizontale doit vérifier, et
3. Nous proposons des solutions pour satisfaire chacune des conditions de quasi-optimalité.

5.3 Nouvelle formulation du problème d'optimisation de performances par la fragmentation horizontale

Étant donné :

1. Un entrepôt de données noté DW modélisé par un schéma en étoile (Figure 5.1), constitué de d tables de dimensions D_1, D_2,..., D_d et d'une table des faits F centrale,
2. Une charge de travail W composée des requêtes décisionnelles les plus fréquentes Q_1,Q_2,..., Q_n, où chaque requête Q_i possède une fréquence d'accès f_i.

Le problème d'optimisation de performance de la charge de travail W en utilisant la fragmentation horizontale primaire est la fragmentation dérivée, consiste à trouver un schéma de fragmentation horizontale composé de K sous schéma, et permettant de réduire le coût d'exécution global de W sur l'entrepôt de données fragmenté. Le schéma de fragmentation horizontale (Figure 5.2) généré par ces deux types

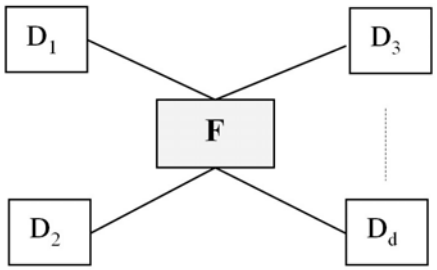


FIGURE 5.1 – Schéma étoile d'un entrepôt de données non fragmenté

de fragmentation est composé d'un grand nombre de sous schéma, calculé par la Formule 5.1. Chaque sous schéma contient un fragment de faits HF_i entouré part les fragments des tables dimension D_{i1},D_{i2},D_{i3},...,D_{in} participant à sa construction par des opérations de semi-jointure avec la table des faits F.

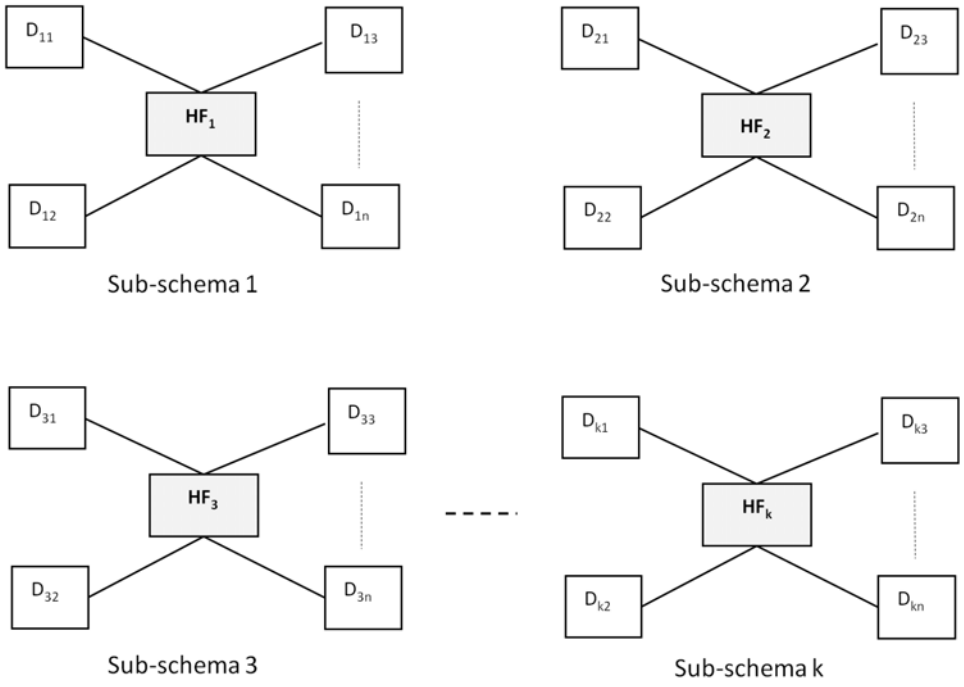


FIGURE 5.2 – Schéma de fragmentation horizontale (primaire/dérivée) de l'entrepôt de données

En plus des inconvénients de la fragmentation horizontale primaire et la fragmentation horizontale dérivée que nous avons déjà soulevés, Bellatreche et al. (2008) ont prouvé que l'utilisation de ces deux types de fragmentation pour trouver un schéma

de fragmentation quasi-optimal est un problème $NP\text{-}Hard$.

Pour éviter ces inconvénients ainsi que cette complexité, nous proposons une nouvelle formulation du problème (Kechar and Nait-Bahloul, 2017), comme suit :
Le problème d'optimisation de performance de la charge de travail W, consiste à trouver un schéma de fragmentation horizontale noté SFH composé de K fragments de faits et permettant de réduire considérablement le coût d'exécution global de W sur l'entrepôt de données fragmenté.

Ainsi, selon notre formulation, seule la table des faits est fragmentée horizontalement sans passer par la fragmentation horizontale des tables dimension (Figure 5.3).

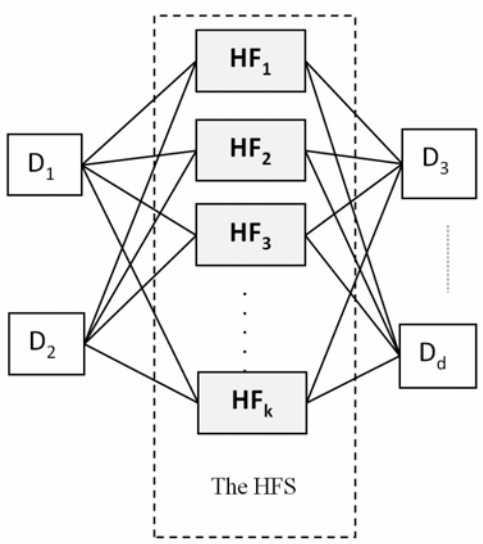


FIGURE 5.3 – Schéma de fragmentation horizontale de la table de faits

Résoudre ce problème, revient à trouver un SFH composé d'un nombre réduit de fragments de faits et minimisant le coût d'exécution des requêtes. Pour atteindre cet objectif, nous définissons sur la base de formules de calcul de coûts, cinq conditions qui doivent être respectées lors de la recherche du schéma de fragmentation horizontale. Nous appelons ces conditions : conditions de quasi-optimalité. Leur satisfaction est fondamentale pour générer un SFH quasi-optimal.

5.4 Conditions de quasi-optimalité du schéma de fragmentation horizontale

Soit :

- $CEG^{SFH}(W)$: le coût d'exécution global de la charge W exécutée sur l'entrepôt de données fragmenté par le SFH de la Figure 5.3, et
- $CE^{SFH}(Q_i)$: le coût d'exécution d'une requête $Qi \in W$ exécutée sur le même entrepôt de données fragmenté.

Formellement, le coût d'exécution global de la charge W est égal à la somme des coûts d'exécution des n requêtes appartenant à W (Formule 5.2).

$$CEG^{SFH}(W) = \sum_{i=1}^{n} EC^{SFH}(Q_i) \tag{5.2}$$

Ainsi, optimiser le $CEG^{SFH}(W)$, est équivalent à trouver un SFH optimisant le coût d'exécution de chaque requête Q_i (i =1.. n). Cependant, trouver un tel schéma de fragmentation sera le cas le plus favorable. Malheureusement, dans les cas réels quelques requêtes peuvent ne pas bénéficier de la fragmentation horizontale de l'entrepôt de données (Section 5.2.2). De ce fait, nous définissons la Formule 5.3 pour calculer le $CEG^{SFH}(W)$ en prenant en compte les requêtes non bénéficiant de la fragmentation horizontale.

$$CEG^{SFH}(W) = \sum_{i=1}^{x} EC^{SFH}(Q_i) + \sum_{j=1}^{y} EC^{SFH}(Q_j) \tag{5.3}$$

où x représente le nombre de requêtes optimisées par SFH notées requêtes RBF (Requêtes Bénéficiant de la Fragmentation), et y représente le nombre de requêtes non optimisées par SFH notées $RNBF$ (Requêtes Non Bénéficiant de la Fragmentation) et x+y =n.

Nous notons ici que dans (Bellatreche et al., 2000), les auteurs ont montré expérimentalement que la fragmentation horizontale primaire et la fragmentation horizontale dérivée de l'entrepôt de données, détériorent considérablement le coût d'exécution globale des requêtes qui ne bénéficient pas de la fragmentation. Car, leur traitement nécessite la reconstruction complète de l'entrepôt de données original, qui est une opération très coûteuses. Elle est réalisée par plusieurs opérations de jointure entre les fragments de dimensions et le fragment de faits appartenant à chaque sous-

schéma de fragmentation, suivies de (K-1) opérations d'union entre les résultats des jointures (où K représente le nombre de sous schéma de fragmentation). Ainsi, un nombre élevé de requêtes de type $RNBF$ ayant des coûts d'exécution important, aura un impact négatif sur le gain en amélioration de performances globales de la charge de travail W.
Cependant, nous avons constaté que les approches de fragmentation horizontale proposées dans les entrepôts de données (Noaman and Barker, 1999; Bellatreche and Boukhalfa, 2005; Bellatreche et al., 2006, 2008; Dimovski et al., 2010; Bellatreche, 2012; Thenmozhi and Vivekanandan, 2014; Bouchakri et al., 2014; Gacem and Boukhalfa, 2014; Toumi et al., 2015; Zamanian et al., 2015; Ghorbel et al., 2016), ne prennent pas en compte les requêtes $RNBF$. De ce fait, et afin de minimiser davantage le $CEG^{SFH}(W)$, le SFH à générer doit vérifier nos deux premières conditions que nous définissons ci-dessus :

Condition 1 : le SFH doit générer un nombre minimal de requête $RNBF$. Autrement dit, selon la Formule 5.3, le nombre y des requêtes $RNBF$ doit être réduit.

Condition 2 : le SFH doit minimiser la détérioration des coûts d'exécution des requêtes $RNBF$. En d'autres termes, le coût d'exécution des requêtes $RNBF$ avant la fragmentation doit être plus ou moins égal à leur coût d'exécution après la fragmentation horizontale de l'entrepôt de données.
Formellement :
Soit $EC^{ED}(Q_j)$ le coût d'exécution de la requête Q_j de type $RNBF$ avant la fragmentation, et $EC^{SFH}(Q_j)$ son coût d'exécution après la fragmentation, alors
$\sum_{j=1}^{y} EC^{ED}(Q_j) = \sum_{j=1}^{y} EC^{SFH}(Q_j) + \epsilon$, (où ϵ doit être une valeur négligeable).

Pour minimiser davantage le $GEC^{SFH}(W)$ (Formule 5.3), il est nécessaire de minimiser le coût d'exécution des requêtes RBF. Ainsi, en plus de nos deux premières conditions concernent les requêtes $RNBF$, il faut trouver un schéma de fragmentation qui minimise considérablement le $CE^{SFH}(Q_i)$ pour tout i= 1..x. À partir de ce

raisonnement logique, nous développons ci-dessous des formules de coûts pour calculer le $CE^{SFH}(Q_i)$. Ces formules de coût sont basées sur l'algorithme de jointure par hachage. Selon Kitsuregawa et al. (1983) et Shapiro (1986), l'algorithme de jointure par hachage est le plus approprié pour le traitement des opérations d'équi-jointures entre une table de grande cardinalité (la table de faits) et une ou plusieurs tables de petites cardinalités (les tables de dimension). Sur la base de ces formules de coût, nous définissons deux autres conditions nécessaires pour minimiser le $CE^{SFH}(Q_i)$, où Q_i est une requête bénéficiant de la fragmentation horizontale.

Pour commencer, le coût d'exécution d'une requête Q nécessitant une équi-jointure entre la table de faits F et la table dimension D, est donné par la Formule 5.4 (en utilisant l'algorithme de jointure par hachage).

$$CE^{ED}(Q) = 3|F| + 3|D| \tag{5.4}$$

où $|F|$, et $|D|$ représente respectivement le nombre de pages disque occupées par la table F, et par la table $D1$. ED représente l'entrepôt de données original sur lequel la requête Q est exécutée.

Maintenant, soit Q_i une requête de type RBF exécutée sur l'entrepôt de données fragmenté selon notre schéma de fragmentation horizontale de la Figure 5.3. À partir de la Formule 5.4, nous déduisons le coût d'exécution de Q_i donné par la Formule 5.5

$$CE^{SFH}(Q_i) = 3(|FH_1| + ... + |FH_l|) + 3|D_1| + \sum_{i=2}^{t} 3|D_i| \tag{5.5}$$

où l et t représentent respectivement le nombre de fragments de la table de faits et le nombre de tables dimensions nécessaires au traitement de Q_i.

En supposant que tous les résultats intermédiaires tiennent en mémoire centrale, la requête Q_i sera exécutée selon les étapes suivantes :

1. Charger en mémoire les l fragments de F,
2. Effectuer une jointure par hachage entre l'union des l fragments ($|HF_1| + ... + |HF_l|$), et la première table de dimension D_1,
3. Charge les $(t-1)$ tables de dimension une après l'autre et effectuer une jointure par hachage avec le dernier résultat intermédiaire.

Sachant que :

- Le nombre de pages occupées par une table R est égale à $|R| = \left(\frac{L(R)}{P}\right) \times \|R\|$ (Elmasri and Navathe, 2010), où $L(R)$, P, et $||R||$ représentent respectivement la longueur en octet d'un tuple dans la table R, la taille d'une page disque en octet, et la cardinalité de la table R (le nombre de tuples dans R),
- La longueur d'un tuple dans la table de faits F est égale à la longueur d'un tuple dans chaque fragments F_i. En d'autres termes : $L(FH_1) = L(FH_2) =...= L(FH_l) = L(F)$, car nous fragmentons la table de faits F horizontalement sans modifier la structure des données.

À partir de ces deux points, nous récrivons la Formule 5.5 de la façon suivante :

$$CE^{SFH}(Q_i) = 3\left(\frac{L(F)}{P}\right)(\|FH_1\| + ... + \|FH_l\|) + 3\left(\frac{L(D_1)}{P}\right)\|D_1\| + \sum_{i=2}^{t} 3\left(\frac{L(D_i)}{P}\right)\|D_i\| \quad (5.6)$$

Dans la Formule 5.6, les valeurs P, $L(F)$, $L(D_i)$, $\|D_i\|$ sont constantes. Seules les tailles des fragments horizontaux HF_1,...,HF_l (c.-à-d. $\|HF_1\|$,...,$\|HF_l\|$) sont variables selon le SFH généré. De ce fait, minimiser le $CE^{SFH}(Q_i)$ sera équivalent à minimiser la valeur du terme $(||HF_1|| + \ldots + ||HF_l||)$.
Mathématiquement, minimiser $(||HF_1|| + \ldots + ||HF_l||)$ est équivalent :

1. À minimiser la taille de chaque fragment horizontal FH_i, avec i=1..l, et
2. À réduire le nombre l de fragments horizontaux.

Ainsi, pour minimiser davantage le $CEG^{SFH}(W)$, nous définissons à partir des points (1) et (2), les deux conditions suivantes :

> **Condition 3** : les fragments horizontaux utilisés par les requêtes RBF doivent avoir des cardinalités inférieures à la cardinalité de la table de faits. Formellement : $||HF_i|| < ||F||$ pour $i = 1..l$.

> **Condition 4** : le traitement des requêtes RBF sur l'entrepôt de données fragmenté par notre SFH doivent nécessiter un nombre réduit de fragments horizontaux. En d'autres termes, il faut minimiser le nombre l.

Finalement, pour faciliter la maintenance des fragments et réduire son coût, nous ajoutons la dernière condition suivante :

> **Condition 5** : le SFH doit être composé d'un minimum de fragments de faits.

Ainsi, nous qualifions tout schéma de fragmentation horizontale SFH vérifiant ces cinq conditions comme schéma de fragmentation quasi-optimal. Nous décrivons dans la section suivante nos propres solutions permettant de satisfaire les conditions de quasi-optimalité définie ci-dessus. Ces solution nous permettrons de trouver le SFH quasi-optimal recherché.

5.5 Satisfaction des conditions de quasi-optimalité du schéma de fragmentation horizontale

- Pour satisfaire la Condition 2 (minimiser la détérioration des coûts d'exécution des requêtes $RNBF$), nous proposons de fragmenter uniquement la table de faits sans passer par la fragmentation horizontale primaire des tables dimension. Comme montré dans la Figure 5.3, les tables dimension ne sont pas fragmentées, et le schéma de fragmentation est composé seulement des fragments horizontaux de la table de faits. Ainsi, pour une requête Q_i de type $RNBF$ utilisant par exemple trois dimensions D_1, D_2, et D_3, si nous fragmentons la table de faits en 105 fragmentes horizontaux, alors le traitement de Q_i nécessitera :

  1. 104 opérations d'union entre les 105 fragments de faits, suivie de
  2. 03 opérations de jointure entre le résultat de l'union et les dimensions D_1, D_2, et D_3.

  Cependant, dans le cas d'une fragmentation horizontale primaire et une fragmentation horizontale dérivée, si par exemple les dimensions D_1, D_2, et D_3 sont fragmentées respectivement en 03 fragments (D_{11}, D_{12},D_{13}), 05 fragments (D_{21}, D_{22}, D_{23}, D_{24},D_{25}), et 07 fragments (D_{31},D_{32},D_{33},D_{34},D_{35},D_{36},D_{37}), alors le nombre K des sous-schémas de fragmentation sera égale à $K = 3{\times}5{\times}7 = 105$. Comme montré par la Figure 5.4, chaque fragment de faits est définie par des opérations de semi-jointure ($FH_1 = F \ltimes D_{11} \ltimes D_{21} \ltimes D_{31}$; $FH_2 = F \ltimes D_{11} \ltimes D_{21} \ltimes D_{32}$; $FH_3 = F \ltimes D_{11} \ltimes D_{21} \ltimes D_{33}$;... ; $FH_{105} = F \ltimes D_{13} \ltimes D_{25} \ltimes D_{37}$).
  Par conséquent, le traitement de Q_i nécessitera :

  1. 315 opérations de jointure, justifiées par 03 opérations de jointure entre le fragment de faits et les fragments des dimensions D_1, D_2, et D_3 appartenant à chacun des 105 sous-schémas de fragmentation (c.-à-d. 105 $\times$ 3= 315),

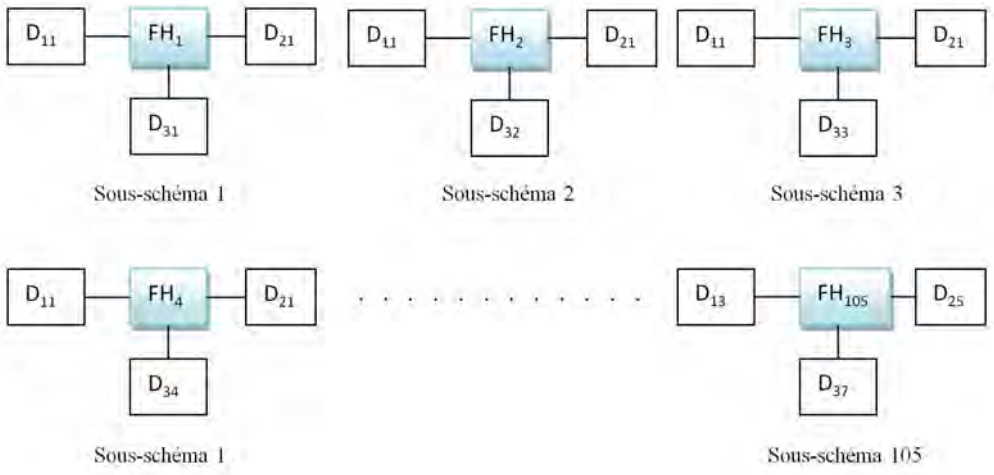


FIGURE 5.4 – Entrepôt de données fragmenté horizontalement (primaire/dérivée) en 105 sous schémas

2. suivie de 104 opérations d'union entre les 105 résultats des jointures.

Nous remarquons clairement à partir de cet exemple, que notre technique de fragmentation nous permet d'économiser 312 opérations de jointure lors du traitement de Q_i de type $RNBF$, ce qui par conséquent contribuera à la minimisation de la détérioration de son coût d'exécution.

En d'autres termes, le traitement de Q_i nécessitera la reconstruction de l'entrepôt de données. Cette reconstruction est réalisée en :

- ◇ 107 opérations (104 opérations d'union + 03 opérations jointure) dans notre cas de fragmentation, ou
- ◇ 419 opérations (104 opérations d'union + 315 opérations jointure) dans le cas d'une fragmentation horizontale primaire suivie d'une fragmentation horizontale dérivée.

- Pour satisfaire la Condition 5 (générer un nombre minimal de fragments de faits), nous proposons de définir chaque fragment horizontal de la table de faits par un prédicat de fragmentation (Définition 4) sélectionné à partir de l'ensemble des prédicats de sélection extrait à partir de la charge de requêtes. Cela, nous permet de générer de façon incrémentale les fragments de faits, dont leur nombre suit une croissance linéaire en fonction du nombre de prédicats de fragmentation. Ainsi, pour z prédicats de sélection le nombre de fragments de faits K sera égal aux pires des cas à z, contrairement à la fragmentation horizontale dérivée, dans laquelle

le nombre de fragments de faits est étroitement lié au nombre m des dimensions fragmentées et à leurs nombre de fragments b_i (Formule 5.1).

Nous notons ici que la solution que nous proposons pour satisfaire la Condition 5, conduit systématiquement à la satisfaction de la Condition 4 (le traitement d'une requête RBF doit nécessiter un nombre réduit de fragments), car toute requête RBF sera exécutée sur un schéma de fragmentation composé d'un nombre réduit de fragments horizontaux.

- Pour satisfaire la Condition 1 et la Condition 3 (optimiser les performances d'un nombre maximal de requêtes et gérer des fragments de faits de tailles petites par rapport à la taille de la table de faits), nous proposons une technique permettant de sélectionner les prédicats de fragmentation définissant les fragments horizontaux de la table de faits. Notre technique est basée sur l'exploitation des concepts suivants :
  - ◇ Les nombres d'occurrences (Définition 1),
  - ◇ Les fréquences d'accès (Définition 2), et
  - ◇ Les sélectivités des prédicats de sélection (Définition 3).

  Que nous définissions ci-dessous :

  Définition 1 : le nombre d'occurrences noté NO_j d'un prédicat de sélection p_j est égal au nombre de requêtes utilisant p_j. Il varie entre 1 et n, où n est le nombre des requêtes dans la charge de travail W.

  Il est important de noter ici que les prédicats de sélection ayant de faible valeur du nombre d'occurrences, ne nous aident pas à satisfaire la Condition 1. Pour cela, nous fixons un nombre d'occurrence minimal noté $NOMin$ comme paramètre pour éliminer de tels prédicats de sélection du processus de fragmentation.

  Définition 2 : la fréquence d'accès notée FA_j d'un prédicat de sélection p_j est égale à la somme des fréquences d'accès des requêtes utilisant p_j.

  Définition 3 : la sélectivité d'un prédicat de sélection p sur la table D sur la quelle il est défini, représente la fraction de tuples de D satisfaisant p (Jonathan, 2006).

  La sélectivité d'un prédicat de sélection p est calculée par la Formule 5.7 (Jonathan, 2006).

$$S_p^D = \frac{\|\delta_p(D)\|}{\|D\|} \tag{5.7}$$

Dans notre approche de fragmentation, nous nous intéressons à la sélectivité de p sur la table de faits F que nous notons S_p^F, où p est un prédicat de sélection défini sur l'une des dimensions D de l'entrepôt de données. Dans le cas où p est sélectionné comme prédicat de fragmentation (Définition 4), la valeur de sa sélectivité aura un impact direct et important sur la taille du fragment généré par p. Pour cette raison, nous calculons la sélectivité du prédicat de sélection p de manière exacte par la Formule 5.8 (Valduriez, 1987).

$$S_p^F = \frac{\|F \ltimes \delta_p(D)\|}{\|F\|} \tag{5.8}$$

où $\|F \ltimes \delta_p(D)\|$ est la cardinalité du résultat de la jointure entre la table des faits F et le résultat de la restriction du prédicat p sur la dimension D, et $\|F\|$ représente la cardinalité de la table de faits F (c.-à-d., le nombre de tuples dans F). Toutefois, la valeur de la sélectivité peut être toujours estimée à partir des statistiques maintenues dans le catalogue du système de gestion de la base de données (Jonathan, 2006).

La sélectivité d'un prédicat de sélection sur la table des faits F, varie entre 0 et 1. Nous soulignons ici que : plus cette valeur se rapproche de la valeur 1 plus la taille du fragment horizontal défini par ce prédicat sera grande et se rapproche de la taille de la table des faits. En conséquence, le coût d'exécution d'une requête sur un tel fragment ne bénéficiera pas d'une grande amélioration de performance. De ce fait, et en plus des sélectivités des prédicats de sélection, nous donnons à l'administrateur de l'entrepôt de données la possibilité de déterminer un seuil S de valeur entre 0 et 1. Ce seuil représente la sélectivité maximale autorisée que tout prédicat de fragmentation doit respecter. De cette façon, nous pouvons contrôler davantage les tailles des fragments par rapport à la taille de la table faits. À partir des trois concepts définis précédemment, à savoir le nombre d'occurrences, la fréquence d'accès, et la sélectivité, nous définissons un prédicat de fragmentation dans la Définition 4 comme suit :

Définition 4 : nous appelons prédicat de fragmentation tout prédicat de sélection ayant :

1. Un nombre d'occurrence maximal supérieur ou égal au nombre d'occurrence minimal $NOMin$ déterminé par l'administrateur de l'entrepôt de données,
2. Une fréquence d'accès maximale,
3. Une sélectivité minimale par rapport aux sélectivités des autres prédicats de sélection, et
4. Une sélectivité inférieure au seuil S fixé par l'administrateur de l'entrepôt de données.

Ainsi, tout fragment de faits défini par un prédicat de fragmentation respectant notre Définition 4 :

◇ Optimisera les performances d'un nombre maximal de requêtes (satisfaction de la Condition 1) dont la somme de leurs fréquences d'accès est maximale (selon les points (1) et (2)).

◇ Aura une taille petite (satisfaction de la Condition 3) comparée à la taille de la table de faits (selon les points (3) et (4)).

5.6 Conclusion

Dans ce chapitre, nous avons présenté les inconvénients des techniques de fragmentation horizontale proposées dans les entrepôts de données. Pour surmonter ces inconvénients, nous avons proposé une nouvelle formulation du problème d'optimisation de performances par l'utilisation de la fragmentation horizontale de la table de faits. Sur la base de formules de coût, nous avons défini des conditions de quasi-optimalité qu'un schéma de fragmentation horizontale doit vérifier. Pour satisfaire ces conditions, nous avons proposé des solutions permettant de générer le schéma de fragmentation quasi-optimal.

Sur la base des solutions proposées, nous détaillons, dans le chapitre suivant, notre approche de fragmentation horizontale de la table de faits.

Chapitre 6

Nouvelle approche de fragmentation horizontale dans les entrepôts de données

■ *We try to solve very complicated problems without letting people know how complicated the problem was* ■

Jonathan Ive

Sommaire

6.1 Introduction

Dans ce chapitre, nous détaillons notre approche de fragmentation horizontale proposée dans le but d'optimiser les performances des requêtes décisionnelles interrogeant un entrepôt de données relationnel (Kechar and Nait-Bahloul, 2017). Afin de générer un schéma de fragmentation quasi-optimal, nous développons un algorithme de fragmentation horizontale basé sur les solutions de satisfactions des conditions de quasi-optimalité (Section 5.4 et Section 5.5). Pour assurer la disjonction entre les fragments horizontaux, la complétude de la fragmentation, et la reconstruction de l'entrepôt de données original, nous proposons un algorithme de validation de notre fragmentation horizontale par rapport aux règles d'exactitudes définies dans (Özsu and Valduriez, 2011). À la fin de ce chapitre, nous proposons une stratégie de réécriture et de traitement des requêtes sur l'entrepôt fragmenté, qui nous permet d'assurer la transparence d'accès aux données.

6.2 Fragmentation horizontale de la table de faits

L'algorithme de fragmentation horizontale que nous proposons est composé de trois étapes schématisées par la Figure 6.1, résumées par le pseudo-code de l'Algorithme 1, et détaillées ci-dessous.

6.2.1 Étape 1 : Extraction des prédicats et construction de la matrice d'usage

À partir de la charge de travail W composée de n requêtes : $W = \{Q_1, Q_2, ..., Q_n\}$, où chaque requête possède une fréquence d'accès f_i :

1. Nous procédons à l'extraction des prédicats de sélection figurant dans la clause *where* de chaque requête. Nous notons l'ensemble de ces prédicats de sélection par SP,
2. Pour chaque prédicat de sélection p_j dans SP (j=1..z, où z est le nombre des prédicats dans SP), nous calculons la sélectivité de p_j sur la table de faits F en utilisant la Formule 5.8,
3. Nous construisons une matrice ayant n lignes et z colonnes que nous appelons

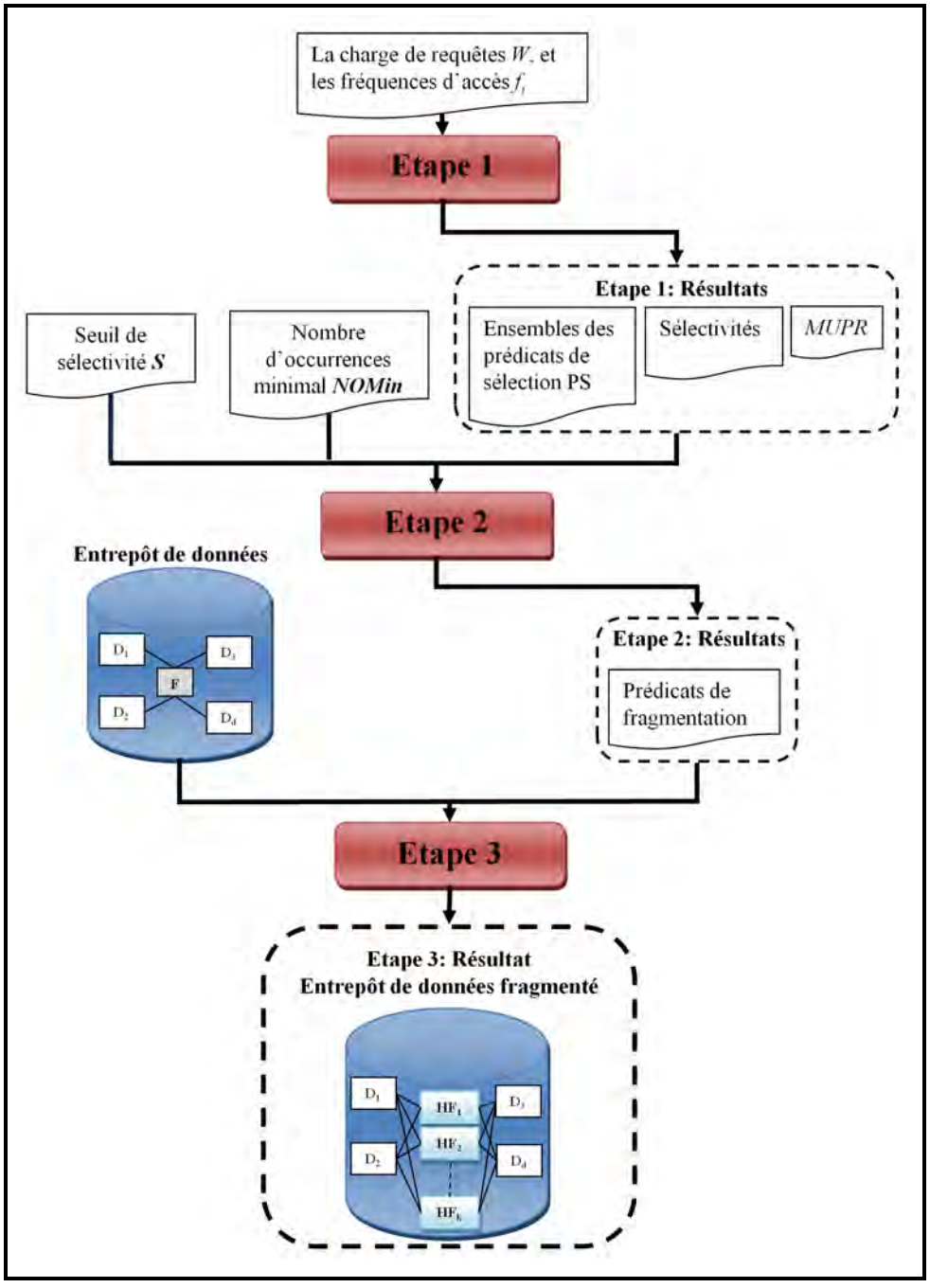


FIGURE 6.1 – Étapes de fragmentation horizontale de la table de faits

Matrice d'Usage des Prédicats de sélection par Requêtes notée $MUPR$ (Tableau 6.1). Pour chaque requête Q_i et pour chaque prédicat de sélection p_j, si p_j figure dans la clause *where* de Q_i alors $MUPQ(i,j)=fi$, sinon $MUPQ(i,j)=0$.

TABLEAU 6.1 – La matrice d'usage des prédicats par les requêtes $MUPR$

	p_1	p_2	p_3	...	p_z
Q_1	f_1	f_1	0	...	0
Q_2	0	f_2	f_2	...	f_2
Q_3	0	0	f_3	...	f_3
.	.	.	.	...	.
.	.	.	.	...	.
.	.	.	.	...	.
Q_n	0	f_n	0	...	f_n

6.2.2 Étape 2 : Identification des prédicats de fragmentation

Si la matrice d'usage $MUPR$ n'est pas vide, alors

1. Nous créons deux vecteurs NO et FA chacun ayant z cellules, et contenant respectivement les nombres d'occurrences et les fréquences d'accès des prédicats de sélection de l'ensemble SP. Nous initialisons les z cellules de NO et FA par la valeur 0,
2. À partir de la matrice $MUPR$, nous calculons (selon la Définition 1 et la Définition 2) le nombre d'occurrences et la fréquence d'accès de chaque prédicat de sélection comme suit :
   pour chaque colonne j correspondant au prédicat de sélection p_j, et pour chaque ligne i correspondant à la requête Q_i :
   - $FA[j] = FA[j] + MUPQ(i, j)$ (nous additionnons les valeurs des cellules de la colonne j pour calculer la fréquence d'accès du prédicat p_j).
   - si $MUPQ(i, j) \neq 0$ alors $NO[j] = NO[j]+1$ (nous incrémentons de 1 la valeur du nombre d'occurrences du prédicat p_j pour toute valeur de cellule différente de 0).
3. Nous éliminons tout prédicat p_j ayant une sélectivité supérieure au seuil S représentant la sélectivité maximale fixé par l'AED. Pour cela, nous affectons la valeur 0 à son nombre d'occurrences dans le tableau NO (c.-à-d.,si (S^F_{pj}>S) alors $NO[j] \longleftarrow 0$). Ainsi, le prédicat p_j sera exclu du processus de fragmentation de la table de faits,
4. Nous identifions, parmi les prédicats de sélection ayant une sélectivité inférieure à S, le premier prédicat de sélection ayant un nombre d'occurrences

maximal noté $NOMax$. En d'autres termes, nous calculons la valeur maximale du tableau NO ; $NOMax = Maximum(NO)$,

5. Si ($NOMax \geq NOMin$) alors :
   (a) Nous sauvegardons le prédicat de sélection ayant $NOMax$ occurrences dans le tableau des Prédicats Candidats à la Fragmentation noté PCF,
   (b) Nous cherchons dans le tableau NO s'il existe d'autres prédicats de sélection ayant un nombre d'occurrences égal à la valeur $NOMax$. Si c'est le cas, nous les ajoutons dans le tableau PCF,
   (c) Parmi les prédicats candidats à la fragmentation figurant dans le tableau PCF, nous sélectionnons celui qui a une fréquence d'accès maximale notée $FAMax$. S'il existe d'autres prédicats de sélection dans PCF possédant la même valeur $FAMax$, alors nous choisissons parmi eux celui ayant une sélectivité minimale, et nous appelons ce prédicat de sélection : *prédicat de fragmentation.*
   (d) Pour éviter la sélection du même prédicat de fragmentation dans les prochaines itérations, nous supprimons de la matrice d'usage $MUPR$ toute ligne i dans laquelle la valeur du $MUPR(i, e)$ est différente de 0, où e représente l'indice de la colonne du prédicat de fragmentation identifié dans (c). Ainsi, un recalcul du nombre d'occurrences $NO[e]$ sera égal à zéro et par conséquent le prédicat de fragmentation p_e sera exclu du reste du processus de fragmentation.
6. Nous répétons l'Étape 2, jusqu'à obtenir une matrice $MUPR$ vide ou $NOMax$ inférieure à $NOMin$.

6.2.3 Étape 3 : Génération des fragments horizontaux

1. Nous arrêtons le processus de fragmentation,
2. Pour chaque prédicat de fragmentation p_e identifié dans l'étape 2, nous génerons un fragment horizontal noté FH_{p_e}. Les tuples de FH_{p_e} sont définis par une semi-jointure entre la table de faits F et le résultat de la restriction du prédicat de fragmentation p_e sur la dimension D, où D représente la table dimension sur laquelle est défini p_e : $FH_{p_e} = F \ltimes \delta_p(D)$,

3. Nous procédons à la phase de validation de la fragmentation horizontale de la table de faits.

Pour une bonne compréhension de la phase de fragmentation horizontale de la table de faits, nous présentons ci-dessous un exemple illustratif.

6.3 Exemple de fragmentation horizontale de la table de faits

Soit :

- W : la charge de travail composée de 07 requêtes , $W = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7\}$, ayant respectivement les fréquences d'accès suivantes : $f_1 = 20$, $f_2 = 15$, $f_3 = 20$, $f_4 = 15$, $f_5 = 15$, $f_6 = 10$, et $f_7 = 15$.
- $S = 0.7$: le seuil de sélectivité maximal , et
- $NOMin = 2$: le nombre d'occurrences minimal.

6.3.1 Étape 1 : Extraction des prédicats et construction de la matrice d'usage

Nous supposons que :
$SP = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ est l'ensemble des prédicats de sélection extraits à partir de W. $S^F_{p_1} = 0.3$, $S^F_{p_2} = 0.5$, $S^F_{p_3} = 0.8$, $S^F_{p_4} = 0.65$, $S^F_{p_5} = 0.2$, $S^F_{p_6} = 0.35$, et $S^F_{p_7} = 0.4$: sont les sélectivités sur la table de faits des prédicats de sélection calculées par la Formule 5.8.
Nous construisons la matrice d'usage $MUPR$ montrée par le Tableau 6.2.

Tableau 6.2 – La matrice d'usage des prédicats par les requêtes $MUPR$: exemple

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
Q_1	0	20	0	0	20	0	20
Q_2	15	15	0	0	0	15	0
Q_3	0	0	20	0	20	0	20
Q_4	0	15	0	15	0	15	0
Q_5	0	0	15	0	15	0	15
Q_6	10	0	0	10	0	10	10
Q_7	0	15	0	15	0	15	0

6.3.2 Étape 2 : Identification des prédicats de fragmentation

La matrice d'usage $MUPR$ n'est pas vide, alors :

1. Nous initialisons les 07 cellules (une cellule pour chaque prédicat de sélection) de chaque vecteur NO et FA par la valeur 0.

NO :	0	0	0	0	0	0	0

FA :	0	0	0	0	0	0	0

2. À partir de la matrice $MUPR$, nous calculons les nombres d'occurrences et les fréquences d'accès des prédicats de sélection

NO :	2	4	2	3	3	4	4

FA :	25	65	35	40	55	55	65

3. Nous éliminons le prédicat de sélection p_3 du processus de fragmentation. Car sa sélectivité $S^F_{p_3} = 0.8$ est supérieure au seuil $S = 0.7$. Nous changeons son nombre d'occurrences dans le vecteur NO par la valeur 0 : $NO[3] = 0$.

NO :	2	4	2	3	3	4	4

$\Downarrow$

NO :	2	4	0	3	3	4	4

4. Nous identifions le premier prédicat de sélection ayant le nombre d'occurrences maximal dans le vecteur NO. Dans notre exemple, la première valeur maximale de NO est $NOMax = 4$, qui correspond au prédicat de sélection p_2.

NO :	2	4	0	3	3	4	4

5. Nous avons ($NOMax$=4 > $NOMin$=2), alors :

   (a) Nous gardons le prédicat p_2 dans le vecteur des prédicats candidats à la fragmentation PCF, ensuite, nous identifions les autres prédicats candidats. Dans notre exemple, les prédicats de sélection p_6 et p_7 ont la même valeur $NOMax$ que p_2, alors nous les ajoutons au vecteur PFC.

NO :	2	4	0	3	3	4	4

PCF :	p_2	p_6	p_7

   (b) Parmi les prédicats candidats à la fragmentation (c.-à-d. p_2, p_6, et p_7), nous procédons à l'identification du prédicat de fragmentation ayant la fréquences d'accès maximale. Comme p_2 et p_7 ont la même valeur maximale de la fréquence d'accès égale à 65, nous utilisons la sélectivité minimale comme arbitre, ce qui nous permet de choisir p_7 comme prédicat de fragmentation car : $S^F_{p_7} = 0.4 < 0.5 = S^F_{p_2}$.

FA :	25	65	35	40	55	55	65
		⇓					⇓
Sélectivités :	0.3	0.5	0.8	0.65	0.2	0.35	0.4

   (c) Nous supprimons de la matrice $MUPR$ toute ligne pour laquelle $MUPR(i, \mathbf{7})$ est différent de 0. Ainsi, nous supprimons les lignes : 1, 3, 5, et 6 (Tableau 6.3).

6. Avec la nouvelle matrice d'usage (Tableau 6.4) comme entrée, nous répétons l'exécution de l'Étape 2. Pour des raisons de simplicité, nous résumons les résultats des prochaines itérations comme suit :

   (a) Dans la deuxième itération, puisque la matrice d'usage n'est pas vide, les prédicats p_2 et p_6 seront sélectionnés comme prédicats candidats à la fragmentation car, la valeur maximale$NOMax$ de NO est 3 qui correspond aux prédicats p_2 et p_6, et la valeur $NOMax$ est supérieure à $NOMin$. Les

TABLEAU 6.3 – Matrice d'usage $MUPR$: suppression des liges 1,3,5,6

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
Q_1	0	20	0	0	20	0	20
Q_2	15	15	0	0	0	15	0
Q_3	0	0	20	0	20	0	20
Q_4	0	15	0	15	0	15	0
Q_5	0	0	15	0	15	0	15
Q_6	10	0	0	10	0	10	10
Q_7	0	15	0	15	0	15	0

$\Downarrow$

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
Q_2	15	15	0	0	0	15	0
Q_4	0	15	0	15	0	15	0
Q_7	0	15	0	15	0	15	0

TABLEAU 6.4 – Nouvelle matrice d'usage $MUPR$

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
Q_2	15	15	0	0	0	15	0
Q_4	0	15	0	15	0	15	0
Q_7	0	15	0	15	0	15	0

fréquences d'accès de p_2 et p_6 sont égales ($FA[2] = FA[6] = 45$), mais la sélectivité de p_6 est inférieure à celle de p_2. De ce fait, le prédicat p_6 sera sélectionné comme prédicat de fragmentation. Toutes les lignes de la matrice d'usage seront supprimées, car $MUPR(i, \mathbf{6}) \neq 0$, pour chaque ligne i.

(b) Dans la troisième itération la matrice d'usage est vide, alors : (i) nous arrêtons le processus de fragmentation, et (ii) nous générons deux fragments horizontaux de la table de faits en utilisant les deux prédicats de fragmentation p_6 et p_7 identifiés précédemment.

Finalement, nous procédons la phase validation de la fragmentation.

Function 1: SPextraction-MUPRconstruction

input : $Q = \{Q_1, Q_2, ..., Q_n\}$, $QAF = \{f_1, f_2, ..., f_n\}$
output: The set SP of the selection predicates, The *Selectivity* array, The usage matrix $MUPR$

begin

/* extraction of the selection predicates */
$SP \leftarrow \phi$;
foreach $Q_i \in Q$ **do**
  foreach p_j *in the where clause of* Q_i **do**
    $SP \leftarrow p_j$;

/* computation of the selectivities */
foreach $p_j \in SP$ **do**
  $Selectivity[j] \leftarrow \frac{||\delta_{p_j}(D) \bowtie F||}{||F||}$;

/* construction of the usage matrix */
foreach $Q_i \in Q$ **do**
  foreach $p_j \in SP$ **do**
    if p_j *is used by* Q_i **then** $MUPR(i,j) \leftarrow f_i$;
    else $MUPR(i,j) \leftarrow 0$;

Function 2: OccNumber-AccFreq-computation

input : $MUPR$, $Selectivity$, SP;
output: NO, AF;

begin

$z \leftarrow$ `SizeOf(`SP`)`;
$n \leftarrow$ `SizeOf(`Q`)`;

/* Initialization of NO, and AF */
for $j \leftarrow 1$ **to** z **do**
  $NO[j] \leftarrow 0$;
  $AF[j] \leftarrow 0$;

/* Computation of the numbers of occurrences, and the access frequencies */
for $j \leftarrow 1$ **to** z **do**
  for $i \leftarrow 1$ **to** n **do**
    $AF[j] \leftarrow AF[j] + MUPR(i,j)$;
    if *(*$MUPR(i,j) \neq 0$*)* **then**
      $NO[j] \leftarrow NO[j] + 1$;

/* Elimination of the predicates having selectivities greater than S */
for $j \leftarrow 1$ **to** z **do**
  if *(*$Selectivity[j] > S$*)* **then**
    $NO[j] \leftarrow 0$

Algorithm 1: DataWarehouse-Horizontal-Fragmentation

```
input  : Q = {Q_1, Q_2, ..., Q_n}, QAF = {f_1, f_2, ..., f_n} ; S ; NOMin;
output: The horizontal fragmentaion schema
begin
    call of SPextraction-MUPRconstruction(Q,QAF) ; /*1*/
    repeat
        if (MUPR is not empty) then
            call of OccNumber-AccFreq-computation(MUPR,Selectivity,SP) ; /*2*/
            NOMax ← MaxValueOf(NO[]);
            y ← the index of the predicate having NOMax occurrences in NO[];
            if (NOMax > NOMin) then
                Add y to PCF[] ; for j ← 1 to z do
                    if NO[j] = NOMax and j ∄ PCF then
                        Add j to PCF[] ;
                foreach index predicate j in PCF[] do
                    Add the access frequency AF[j] to TempAccFreq[] ;
                AFMax ← MaxValueOf(TempAccFreq[]);
                y ← the index of the value AFMax in the array TempAccFreq[];
                h ← PCF[y] ; /* h : the index of the predicate having AFMax */
                Add h to NewPCF[];
                foreach access frequency af in TempAccFreq[] do
                    y ← the index of af;
                    h ← PCF[y];
                    if af = AFMax and h ∄ in NewPCF[] then
                        Add h to NewPCF[]
                foreach index predicate j in NewPCF[] do
                    add Selectivity[j] to TempSelectivity[];
                MinSelectivity ← MinValueOf(TempSelectivity[]);
                y ← the index of MinSelectivity in TempSelectivity[];
                h ← NewPCF[y] ;
                Add the predicate p_h to FragmentationPredicates[] IndexColumn ← h;
                for i ← 1 to n do
                    if MUPR(i, IndexColumn) ≠ 0 then
                        RemoveRow(i);
    until (MUPR is empty) or (NOMax ≤ NOMin);
    FragmentsGeneration(FragmentationPredicates[]);
    FragmentationValidation();
```

6.4 Validation de la fragmentation horizontale de la table de faits

Toute fragmentation horizontale valide doit vérifier les trois règles d'exactitude (Özsu and Valduriez, 2011), que nous rappelons ci-dessous :

- La disjonction : soit une relation R fragmentée horizontalement en FH_1,..., FH_K, si le tuple t_l est dans FH_j, alors t_l ne doit appartenir à aucun autre fragment horizontal FH_i ($i \neq j$). Formellement : $FH_i \cap FH_j = \emptyset$, pour tout i=1..k, j=1..k, et $j \neq i$).
- La complétude : si une relation R est fragmentée horizontalement en FH_1,..., FH_K, alors chaque tuple dans R doit exister dans au moins un fragment FH_i. Cette règle est très importante, car elle assure qu'aucune donnée ne sera perdue après la fragmentation de la table originale R.
- La reconstruction : si une relation R est fragmentée horizontalement en FH_1,..., FH_K, alors il est nécessaire de définir un opérateur relationnel *op* tel que $R = FH_1$ *op* FH_2 *op*...*op* FH_K.

Dans cette section, nous validons notre fragmentation horizontale de la table de faits par rapport à ces trois règles . La Figure 6.2, schématise les étapes de la validation.

6.4.1 La disjonction entre les fragments horizontaux

Aucune règle assurant la disjonction entre les fragments n'est prise en compte dans la première phase de notre fragmentation horizontale de la table de faits. D'où la possibilité de trouver un tuple t dans deux ou plusieurs fragments horizontaux. Nous appelons ces tuples : *les tuples dupliqués*. Nous les identifions comme suit :

Soit FH_{p_1}, FH_{p_2}, FH_{p_3}, ..., FH_{p_l} des fragments horizontaux de la table des faits générés respectivement par les prédicats de fragmentation p_1, p_2, p_3,...,p_l. Nous définissons l'ensemble CC comme étant l'ensemble de toutes les Combinaisons de Conjonctions entre les prédicats de fragmentation, $CC = \{p_1 \wedge p_2, p_1 \wedge p_3, ..., p_1 \wedge p_2 \wedge p_3 \wedge ... \wedge p_l\}$. Un tuple t est dit tuple dupliqué si et seulement si il satisfait au moins une des conjonctions présente dans CC.
Formellement :

$$t \text{ satisfait } (p_1 \wedge p_2 \ldots \wedge p_h) \equiv \left\{ \begin{array}{c} t \text{ satisfait } p_1 \\ et \\ t \text{ satisfait } p_2 \\ et \\ \ldots \\ et \\ t \text{ satisfait } p_h \end{array} \right\} \equiv \left\{ \begin{array}{c} t \in HF_{p_1} \\ et \\ t \in HF_{p_2} \\ et \\ \ldots \\ et \\ t \in HF_{p_h} \end{array} \right\} \equiv t \text{ est dupliqué.}$$

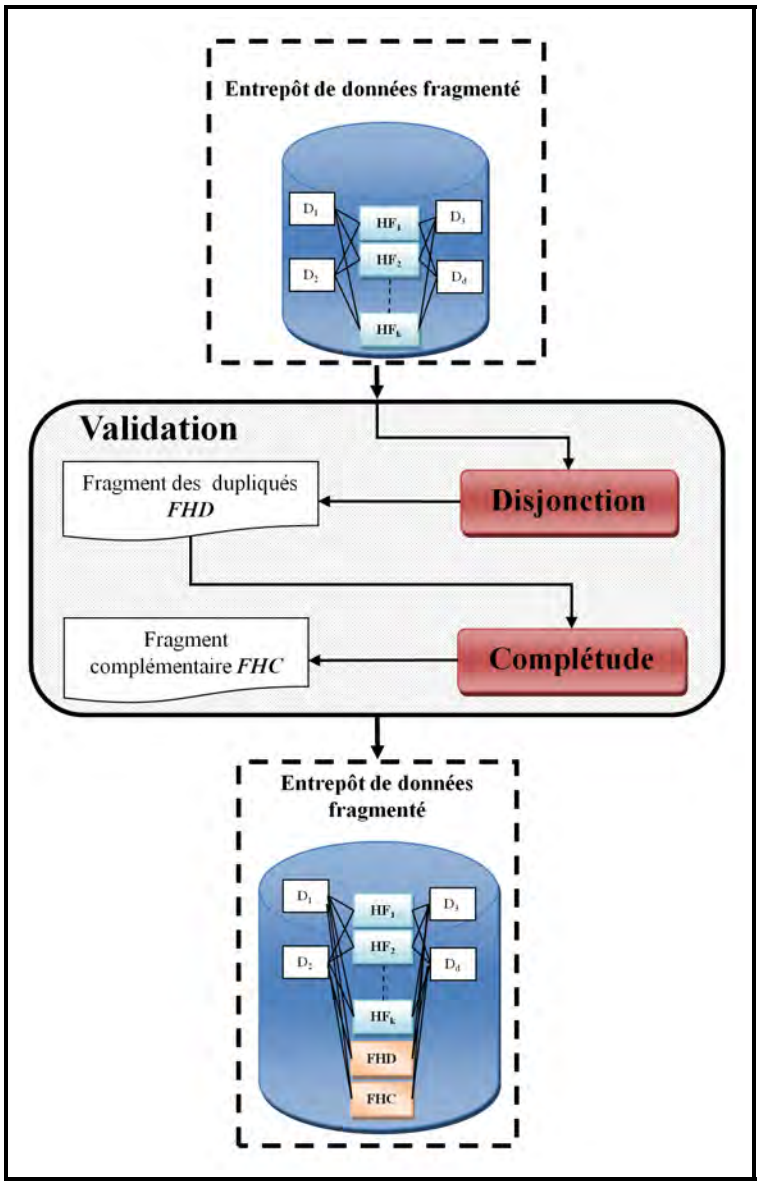


FIGURE 6.2 – Validation de la fragmentation horizontale de la table de faits

Suivant ce formalise, nous assurons la disjonction entre les fragments horizontaux par les deux étapes suivantes :

1. *Identification des tuples dupliqués* : pour identifier les tuples figurant dans plusieurs fragments horizontaux :
   - Nous créons un nouveau fragment appelé fragment horizontal des tuples dupliqués noté FHD.
   - Nous effectuons l'union de tous les fragments horizontaux générés par les prédicats de fragmentation,
   - Nous regroupons les tuples résultat de l'union des fragments par les attributs clés de la table de faits F,
   - Si un tuple figure plus d'une fois dans le résultat du regroupement, alors nous l'identifions comme étant un tuple dupliqué, et nous l'ajoutons dans le fragment FHD,
2. *Raffinage des fragments horizontaux* : deux fragments FH_{p_1} et FH_{p_2} sont disjoint si et seulement si $FH_{p_1} \bigcap FH_{p_2} = \emptyset$. Afin d'assurer cette disjonction, nous supprimons de chaque fragment horizontal FH_i tout tuple t existant dans le fragment horizontale des dupliqués (c-à-d $FH_{p_1} = FH_{p_1}$ - FHD).

Ainsi, les deux étapes précédentes nous permettent de garantir que toute combinaisons d'intersection entre les fragments horizontaux générés par les prédicats de fragmentation ainsi que le fragment des dupliqués est vide.

6.4.2 La complétude

Pour vérifier la règle de complétude, nous générons un fragment horizontal complémentaire noté FHC. Le FHC contient tous les tuples de la table de faits F qui n'appartiennent ni aux fragments horizontaux générés par les prédicats de fragmentation, ni au fragment des dupliqués FHD. Formellement : $FHC = F - (FH_1 \bigcup \ldots \bigcup FH_K \bigcup FHD)$. La complétude nous garantir qu'aucun tuple t de F ne sera perdu après fragmentation (pas de perte de données).

6.4.3 La reconstruction

Nous utilisons l'opérateur relationnel union $\bigcup$ pour assurer la reconstruction de l'entrepôt de données original. Ainsi, l'union de tous les fragments horizontaux, c'est-à-dire les fragments horizontaux générés par les prédicats de fragmentation, le fragment horizontal des dupliqués, et le fragment horizontal complémentaire, nous permet de reconstruire la table de faits originale. Formellement : $F = FH_1 \bigcup \ldots \bigcup FH_K \bigcup FHD \bigcup FHC$.

6.5 Stratégie de réécriture et de traitement des requêtes sur l'entrepôt fragmenté

Les utilisateurs de l'entrepôt de données accèdent aux données sans aucune connaissance de leurs localisations, ce qui est connu sous le nom de la transparence d'accès aux données. Afin de préserver cette transparence d'accès après la fragmentation horizontale de la table des faits, nous procédons à l'identification des fragments horizontaux nécessaires pour la réécriture et le traitement de chaque requête de la façon suivante :
Soit :

- F : la table de faits,
- PSQ_i : l'ensemble des prédicats de sélection figurant dans la clause $where$ de la requête Q_i,
- PF : l'ensemble des prédicats de fragmentation,
- I : l'ensemble des prédicats résultats de l'intersection entre PSQ_i et PF ($I = PSQ_i \bigcap PF$)
- $\|I\|$: le nombre de prédicats dans I (la cardinalité de I),
- FH_{p_i}, le fragment horizontal généré par le prédicat de fragmentation p_i,
- FHD : le fragment horizontal des tuples dupliqués,
- FHC : le fragment horizontal complémentaire.

En se basant sur le nombre de prédicats dans l'ensemble I, nous distinguons les trois cas suivants (Figure 6.3) :

- ($\|I\| = 0$) : signifie que les prédicats de sélection utilisés par Q_i ne participent pas dans la fragmentation horizontale de F. Dans ce cas, nous identifions la requête Q_i comme requête non bénéficiant de la fragmentation (c.-à-d. requête $RNBF$), et nous l'exécutons sur l'union de tous les fragments horizontaux. Sa réécriture, consiste à remplacer F par $FH_1 \bigcup \ldots \bigcup FH_K \bigcup FHD \bigcup FHC$ dans sa clause $from$.
- ($\|I\| = 1$) : signifie que parmi les prédicats de sélection utilisés par Q_i, un seul prédicat est choisi comme prédicat de fragmentation. Dans ce cas, nous identifions la requête Q_i comme requête bénéficiant de la fragmentation (c.-à-d. requête RBF), et nous l'exécutons sur l'union du fragment horizontal généré par le prédicat de fragmentation p_j appartenant à I, et le fragment horizontal des tuples dupliqués FHD. Nous réécrivons la requête Q_i en remplaçant la table de faits F par $FH_{p_j} \bigcup FHD$ dans la clause $from$.
- ($\|I\| > 1$) : dans ce cas, la conjonction de prédicats de sélection dans la clause $where$ de la requête Q_i contient au moins deux prédicats de fragmentation. Toutefois, les tuples satisfaisant une conjonction entre deux ou plusieurs prédicats de fragmentation sont identifiés comme tuples dupliqués. À cet effet, nous exécutons la requête Q_i sur le fragment des tuples dupliqués FHD, nous la réécrivons en remplaçant la table des faits F par FHD, et nous l'identifiant comme requête bénéficiant de la fragmentation.

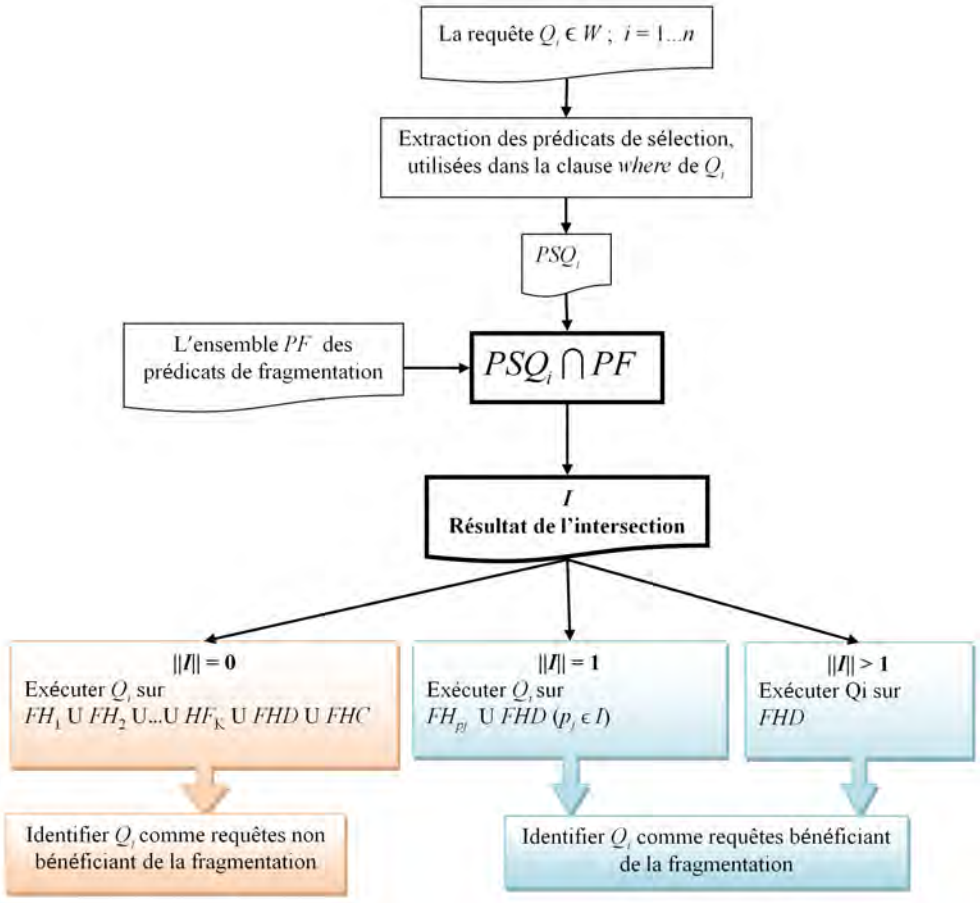


FIGURE 6.3 – Stratégie de réécriture et de traitement des requêtes sur l'entrepôt fragmenté

6.6 Conclusion

Au cours de ce chapitre, nous avons détaillé les différentes phases de notre approche de fragmentation horizontale de la table de faits. En prenant en compte les différentes solutions de satisfaction des conditions de quasi-optimalité que nous avons proposé dans le chapitre 5, nous avons commencé par la génération des fragments de faits en utilisant les prédicats de fragmentation. Pour assurer l'exactitude de notre approche de fragmentation, nous avons créé deux autres fragments horizontaux, à savoir le fragment des tuples dupliqués et le fragment complémentaire. Pour assurer une transparence d'accès aux données de l'entrepôt, nous avons procédé à la réécriture des requêtes de la charge de travail. Dans le but de prouver l'efficacité de l'approche de fragmentation horizontale proposée, nous présentons dans le chapitre 7 une étude expérimentale menée sur le Benchmark APB-1.

Chapitre 7

Études expérimentales et validation

■ *I am results-oriented* ■

Bill gates

Sommaire

7.1 Introduction

Afin de valider nos principales contributions, à savoir :

- Les conditions de quasi-optimalité que nous avons définies dans la section 5.4,
- Les solutions de satisfaction des conditions de quasi-optimalité proposées dans la section 5.5,
- L'approche de fragmentation horizontale de la table de faits développée dans le chapitre 5.6,

nous avons mené une étude expérimentale que nous détaillons ci-dessous.

7.2 Environnement expérimental

7.2.1 Création et alimentation de l'entrepôt de données

Pour mener nos expérimentations, nous avons utilisé le schéma étoile de l'entrepôt de données du benchmark APB-1 (OLAP-Council, 1998), ainsi que son générateur de données. Sous le système de gestion de bases de données relationnel Oracle 10g, nous avons créé trois bases de données notées respectivement ED_1, ED_2, ED_3. Dans chaque base de données, nous avons exécuté les scripts *SQL* de création des tables relationnelles (Annexe 9.1.1) constituant le schéma étoile de l'entrepôt de données. Ce dernier est composé d'une table de faits *actvars* et de quatre tables de dimension *prodlevel*, *custlevel*, *timelevel*, et *chanlevel* (Figure 7.1).

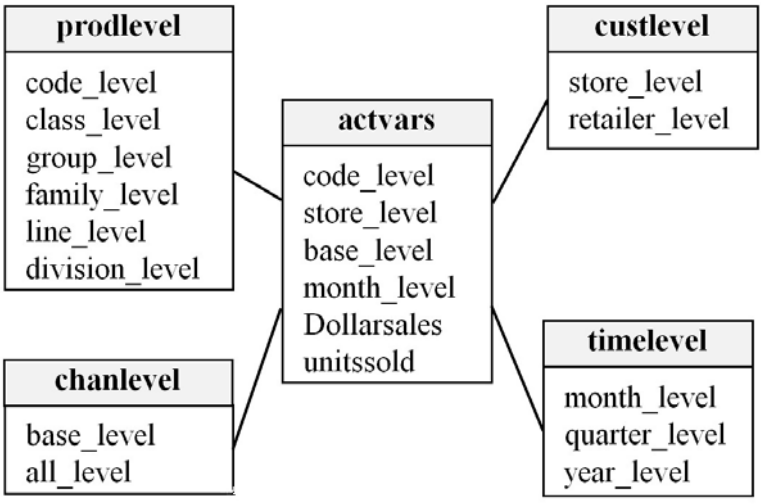


FIGURE 7.1 – Schéma étoile de l'entrepôt de données APB-1

Ensuite, en utilisant le générateur de données, nous avons alimenté respectivement les tables dimension : *prodlevel*, *custlevel*, *timelevel*, et *chanlevel* de chaque entrepôt de données par 18000 tuples, 1800 tuples, 24 tuples, et 19 tuples. En ce qui concerne les tables de faits, nous avons alimenté la table *actvars* de ED_1 par 36 million de tuples (faits), la table *actvars* de ED_2 par 68 million de tuples, et la table *actvars* de ED_3 par 110 million de tuples. La Figure 7.2, schématise le processus de création et d'alimentation de chaque l'entrepôt de données.

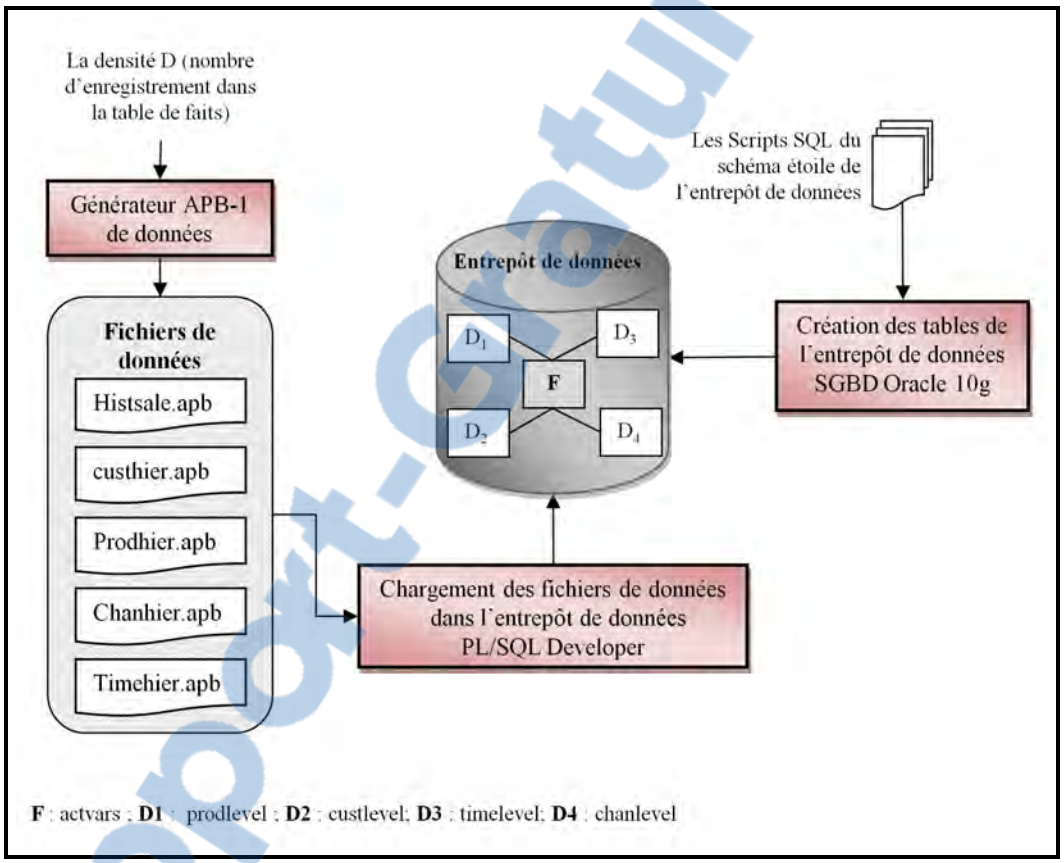


FIGURE 7.2 – Processus de création et d'alimentation de trois entrepôts de données

7.2.2 Architecture de notre implémentation

L'architecture de notre implémentation est composée des modules suivants (Figure 7.3) :

- Module de fragmentation
- Module de validation
- Module d'exécution des scripts SQL
- Module de réécriture des requêtes

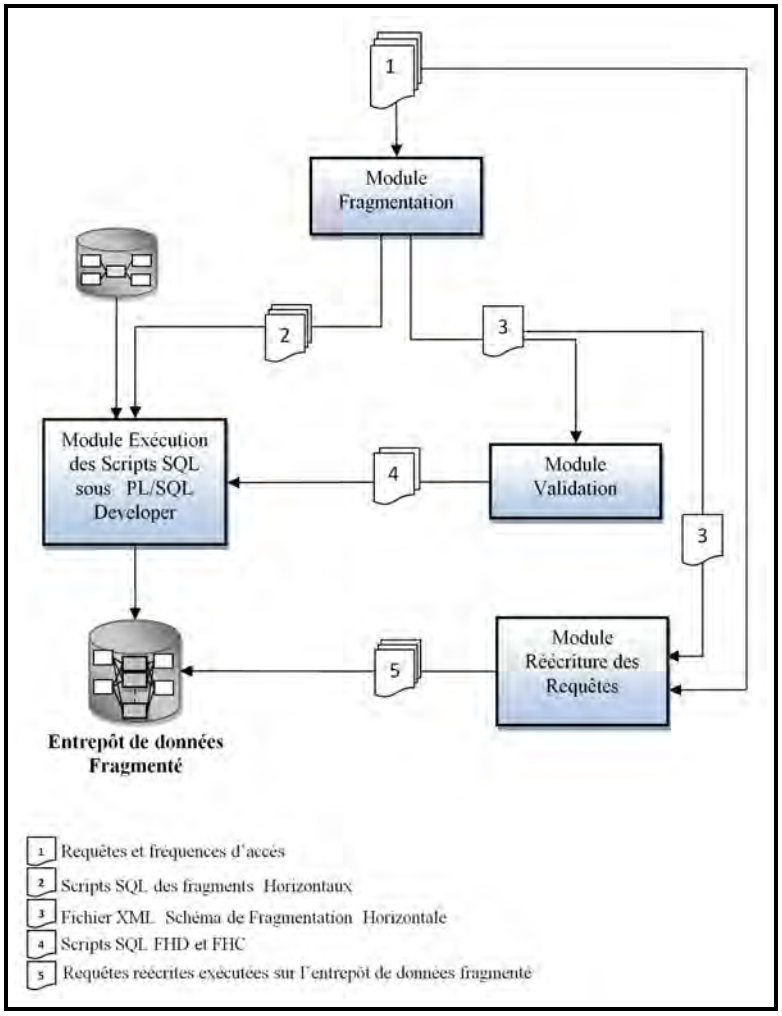


FIGURE 7.3 – Architecture de notre implémentation

À partir d'une charge de travail composée de requêtes SQL avec leur fréquences d'accès (Annexe 9.1.2), le module de fragmentation nous génère : (i) le schéma de fragmentions horizontale de la table de faits (Annexe 9.2.2) vérifiant les conditions de quasi-optimalité, et (ii) les différents scripts SQL des fragments horizontaux (Annexe 9.2.1.1). Le module de validation prend en entrée le schéma de fragmentions horizontale et génère les scriptes SQL, nécessaires pour le calcul du fragment horizontal des tuples dupliqués (Annexe 9.2.1.2) et du fragment horizontal complémentaire (Annexe 9.2.1.3). Le module d'exécution, se charge de la fragmentation physiquement de l'entrepôt de données par l'exécution de l'ensemble des scripts SQL des fragments horizontaux. En se basant sur le schéma de fragmentation horizontale, le module réécriture des requêtes nous génère, pour chaque requête originale, une nouvelle requête (Annexe 9.1.3) réécrite conformément au schéma de fragmentation et s'exécutant directement sur l'entrepôt de données fragmenté. Nous notons ici que nous avons implémenté nos différents algorithmes correspondants aux différents modules avec le langage de programmation JAVA.

7.3 Études expérimentales et validation

Durant toutes nos expériences, nous avons respecté le scénario suivant :

1. Nous avons calculé le temps de réponse global de la charge de requêtes sur l'entrepôt de données original noté $TRG^{ED}(W)$,
2. Nous avons fragmenté horizontalement l'entrepôt de données (la table de faits) par notre algorithme de fragmentation,
3. Nous avons réécrit les requêtes de la charge W selon le schéma de fragmentation horizontal généré dans (2),
4. Nous avons calculé le temps de réponse global de la charge $W^{'}$ des requêtes réécrites sur l'entrepôt de données fragmenté noté $TRG^{EDF}(W')$.

7.3.1 Expérience 1 : Test initial de performances des requêtes

Sur l'entrepôt de données ED_1, nous avons exécuté les étapes du scénario d'évaluation avec les paramètres suivants : (i) une charge de travail composée de 34 requêtes utilisant 22 prédicats de sélection, (ii) un seuil de sélectivité maximale $S = 0.5$ (c'est-à-dire que les tailles des fragments de faits autorisées ne doivent pas dépasser la moitié de la taille de la tables de faits), et (iii) un nombre d'occurrences minimal $NOMin = 2$ (c'est-à-dire que tout prédicat de sélection ayant un nombre d'occurrence $NO >= 2$, est susceptible d'être choisi comme prédicat de fragmentation). L'exécution de notre algorithme de fragmentation horizontale avec ces paramètres, nous a généré un schéma de fragmentation horizontale composé de 10 fragments de faits, dont 8 fragments sont générés par des prédicats de fragmentation, un fragment correspondant au fragment des tuples dupliqués et un fragment complémentaire. Les résultats de l'évaluation de la charge de requêtes sur ce schéma de fragmentation sont montrés par la Figure 7.4.

7.3.2 Expérience 2 : Impact de la taille de la table de faits sur les performances

Pour étudier l'impact de la taille de la table de faits sur l'optimisation des performances des requêtes, nous avons exécuté (en plus de la première expérience) les étapes du scénario d'évaluation avec les mêmes paramètres (c.-à-d. $S = 0.5$, et $NOMin = 2$) sur les entrepôts de données ED_2 (68 million de faits), et ED_3 (110 million de faits). Les temps de réponses

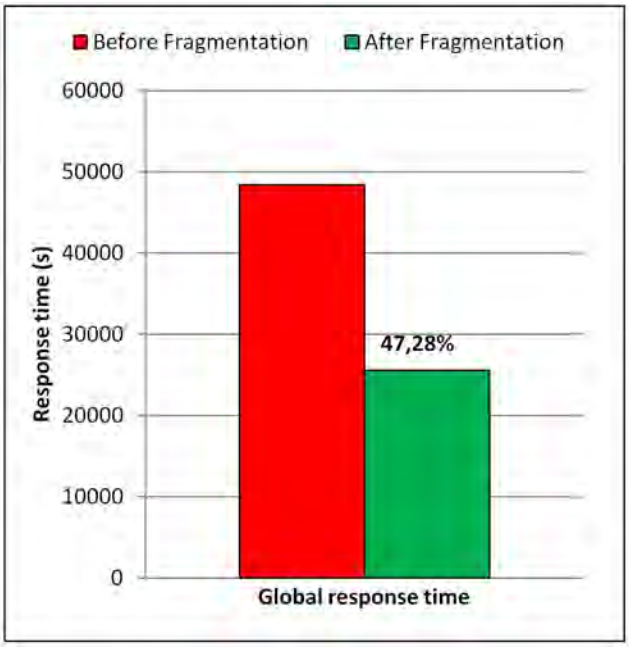


FIGURE 7.4 – Le temps de réponse global de 34 requêtes sur ED_1

globaux des 34 requêtes sur les trois entrepôts de données, avant et après la fragmentation horizontale sont montrés par la Figure 7.5.

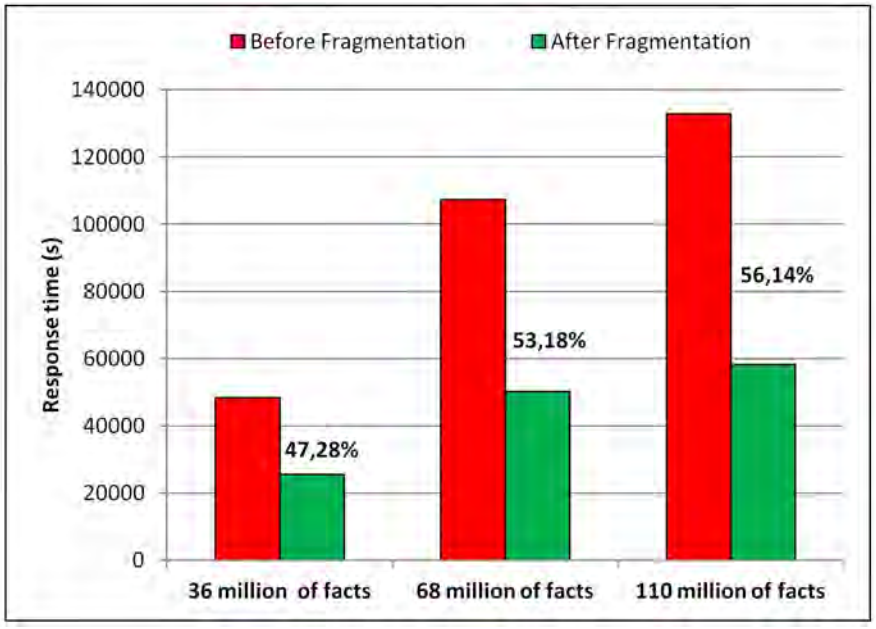


FIGURE 7.5 – Le temps de réponse global de 34 requêtes sur différentes tailles de la table de faits

7.3.3 Expérience 3 : Impact du nombre de requêtes et du nombre de prédicats de sélection sur les performances

Pour étudier l'impact du nombre de requêtes, ainsi que celui du nombre de prédicats de sélection sur l'optimisation de performances, nous avons utilisé dans cette troisième expérience : (i) l'entrepôt de données ED_3 (110 million de tuples), (ii) une charge de travail composée de 51 requêtes utilisant 33 prédicats de sélection, (iii) un seuil de sélectivité maximale $S = 0.5$, et (iv) un nombre d'occurrence minimal $NOMin = 2$. Le schéma de fragmentation horizontale généré par notre algorithme de fragmentation est composé de 15 fragments, dont 13 sont générés par des prédicats de fragmentation. Les résultats de l'évaluation de 51 requêtes avant et après la fragmentation sont détaillés dans la Figure 7.6.

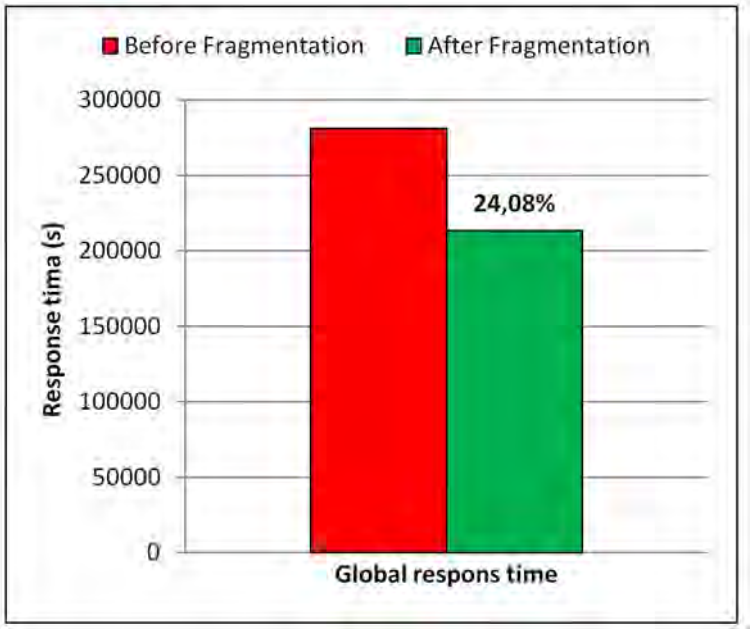


FIGURE 7.6 – Le temps de réponse global de 51 requêtes sur 110 million de faits, avec $NOMin$=2

7.3.4 Expérience 4 : Variation du nombre d'occurrences minimal (paramètre $NOMin$)

Comme nous l'avons déjà mentionné dans la section 5.5, le nombre d'occurrences minimal $NOMin$, nous permet de contrôler le nombre de prédicats de fragmentation. Pour étudier l'influence de ce paramètre sur l'optimisation de performances nous avons, dans cette

expérience, varié la valeur de $NOMin$ de la valeur 2 à la valeur 3, et nous avons gardé les autres paramètres tels utilisés dans l'expérience 3 (c.-à-d. une table de faits de 110 millions de tuples, 51 requêtes, 33 prédicats de sélection, et S=0.5). Avec ces paramètres, notre algorithme nous a généré un schéma de fragmentation composé de 09 fragments dont 07 sont générés par des prédicats de fragmentation. Les résultats de l'évaluation des 51 requêtes sur ce schéma de fragmentation sont résumés par la Figure 7.7.

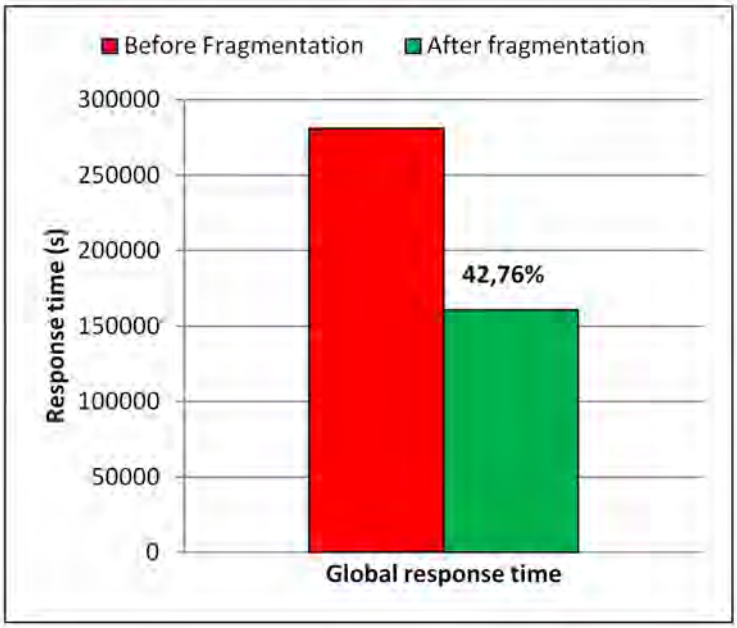


FIGURE 7.7 – Le temps de réponse global de 51 requêtes sur 110 million de faits, avec NOMin = 3

7.4 Discussion

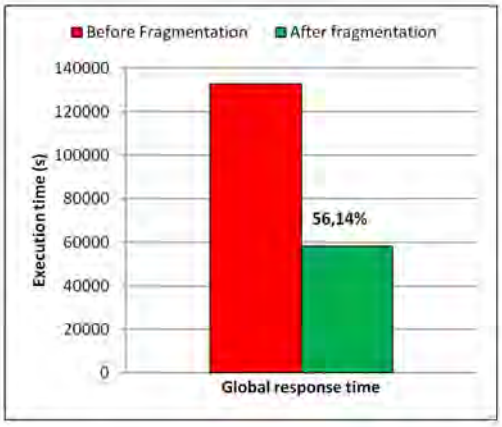
Dans cette section, nous discutons l'efficacité de notre technique de fragmentation horizontale de la table de faits en se basant sur les résultats (c.-à-d. le $TRG^{ED}(W)$, et le $TRG^{EDF}(W')$) des différentes expériences.

À partir des résultats de la Figure 7.4, nous constatons que les performances des requêtes sont considérablement améliorées. En effet, après fragmentation horizontale de la table de faits, le temps de réponse global de 34 requêtes est réduit de 47.28% par rapport au temps de réponse initial calculé sur l'entrepôt de données non fragmenté (original).

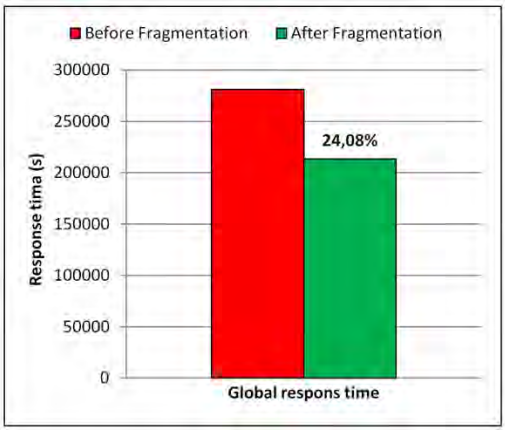
En inspectant les temps de réponses globaux des 34 requêtes montrés par la Figure 7.5, nous

remarquons que malgré l'augmentation de la taille de la table de faits, les performances des requêtes sont toujours préservées. En effet, les taux d'amélioration de performances de 47.28%, 53,18%, et de 56.14% réalisés respectivement sur des tables de faits de 36 million, 68 million, et de 110 million, nous confirment l'efficacité de notre algorithme de fragmentation lors de l'augmentation de la taille de l'entrepôt de données.

Dans la troisième expérience, nous avons augmenté le nombre de requêtes de la charge de travail de 34 requêtes (utilisant 22 prédicats de sélection) à 51 requêtes (utilisant 33 prédicats de sélection). L'évaluation de notre algorithme de fragmentation avec cette nouvelle charge de travail, a permis de réduire le temps de traitement global de la charge de 24.08% par rapport à son temps de traitement sur l'entrepôt de données non fragmenté (Figure 7.6). Cette amélioration de performances, nous prouve l'efficacité de notre algorithme de fragmentation. Cependant, nous avons remarqué que le taux de 24.08% réalisé pour 51 requêtes (Figure 7.8b) est moins important que le taux de 56.14% réalisé pour 34 requêtes (Figure 7.8a).



(a) 34 requêtes, 22 prédicats



(b) 51 requêtes, 33 prédicats

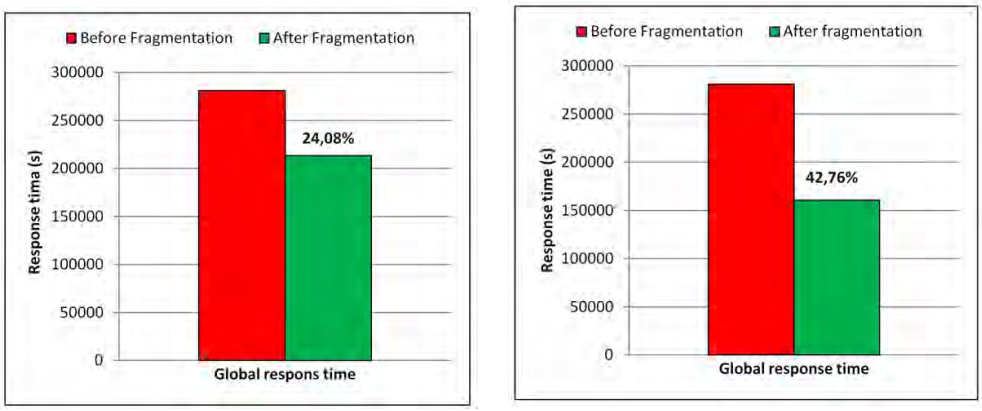
FIGURE 7.8 – Taux d'amélioration de performances sur 110 million de faits

En effet, le nombre de prédicats de fragmentation généré par notre algorithme à augmenter de 8 (pour 34 requêtes) à 13 (pour 51 requêtes), ce qui a engendré :

- 13 fragments de faits pour le cas de 51 requêtes (chaque prédicat de fragmentation définit un fragment de faits (Section 6.2.3)), et
- Une augmentation dans la taille du fragment des tuples dupliqués (FHD), qui dépend du nombre des combinaisons de conjonctions entre les prédicats de fragmentation (Section 6.4.1).

Pour contrôler l'influence du nombre de requêtes et du nombre de prédicats de sélection sur le gain en performances, nous avons exploité le paramètre $NOMin$ (nombre d'occurrences minimal). Ainsi, dans la quatrième expérience, une variation de la valeur de $NOMin$ de 2 à 3, nous a permis d'atteindre une amélioration de 42.76 % (Figure 7.9b) par rapport à 24.08% (Figure 7.9a) réalisée pour $NOMin$=2.

En effet, pour $NOMin$=3, touts les prédicats de sélection ayant un nombre d'occurrences



(a) 51 requêtes - $NOMin$=2 (b) 51 requêtes - $NOMin$=3

FIGURE 7.9 – Taux d'amélioration de performances de 51 requêtes sur 110 million de faits

inférieur à 3 n'ont pas participé au processus de fragmentation de la table de faits. En conséquence, le nombre de prédicats de fragmentation généré par notre algorithme est passé de 13 à 7. Cette réduction, justifie l'augmentation de 18.68% (42.76% - 24.08%) en amélioration de performances, et prouve l'importance du paramètre $NOMin$ dans la génération du meilleure schéma de fragmentation horizontale.

Pour conclure notre discussion, nous résumons ci-dessous les points clés de notre approche de fragmentation horizontale, qui ont contribué fortement à l'amélioration des performances des requêtes décisionnelles et à la réduction du nombre de fragments horizontaux :

- **Le premier point clé - La satisfaction de la Condition 1 (5.4)** : dans toutes nos expériences, nous avons optimisé un nombre maximal de requêtes de la charge de travail W en utilisant conjointement les nombres d'occurrences et les fréquences d'accès des prédicats de sélection lors du choix des prédicats de fragmentation. Par exemple, nous avons optimisé 23 requêtes parmi 34 dans la première expérience, et 30 requêtes parmi 51 dans la quatrième expérience (Figure 7.10 et Figure 7.11). En d'autres termes, nous avons minimisé le nombre y (Formule 5.1) des requêtes non bénéficiant de la

fragmentation horizontale.

- **Le deuxième point clé - La satisfaction de la Condition 3 (5.4)** : l'exploitation des sélectivités des prédicats de sélection combinées avec le seuil de sélectivité maximale S dans le choix des prédicats de fragmentation, nous ont permis de générer des fragments horizontaux de taille très réduite par rapport à la taille de la table de faits. Ainsi, le traitement des requêtes bénéficiant de la fragmentation sur ces fragments a nécessité moins d'opérations d'entrée/sortie, et leur de temps de réponses sont significativement améliorés comme montré dans la Figure 7.11.
- **Le troisième point clé - La satisfaction de la Condition 2 (5.4)** : l'utilisation de la fragmentation directe de la table de fait sans passé par la fragmentation des tables dimension, a permis de minimiser la détérioration de performances des requêtes non bénéficiant de la fragmentation $RNBF$. Par exemple, dans la quatrième expérience, après la fragmentation horizontale de la table de faits, chaque requête $RNBF$ a nécessité un temps d'exécution plus au moins égal à son temps d'exécution sur l'entrepôt de données non fragmenté comme le montre la Figure 7.10. Par conséquent, le temps de réponse global des 21 requêtes (parmi 51) qui n'ont pas bénéficié de la fragmentation est détérioré de 03.84% par rapport à celui calculé avant la fragmentation.

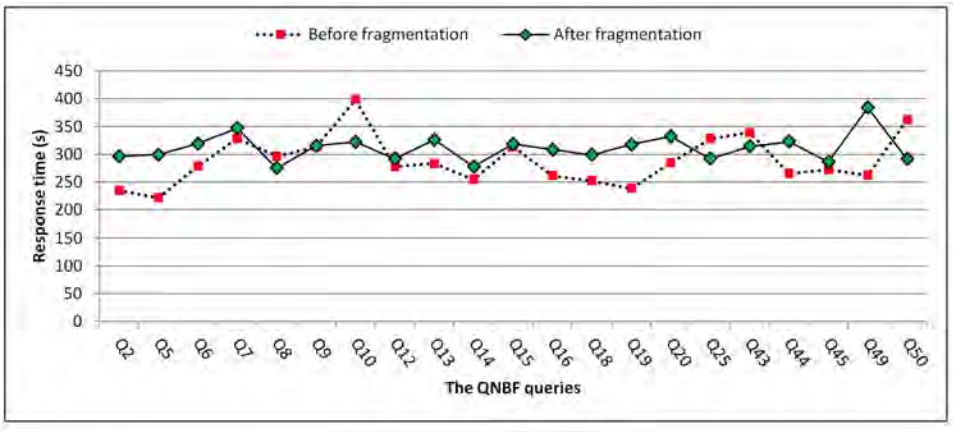


FIGURE 7.10 – Le temps de réponse des 21 requêtes non bénéficiant de la fragmentation

- **Le quatrième point clé - La satisfaction de la Condition 4 (5.4)** : après fragmentation horizontale de la table de faits, le traitement des requêtes bénéficiant de la fragmentation RBF a nécessité au pire des cas deux fragments horizontaux : le fragment horizontal défini par un prédicat de fragmentation et le fragment des tuples dupliqués. Ainsi, le temps de réponse de chaque requête RBF a bénéficié d'une amélioration significative par rapport à celui obtenu avant la fragmentation comme le montre la Figure 7.11.

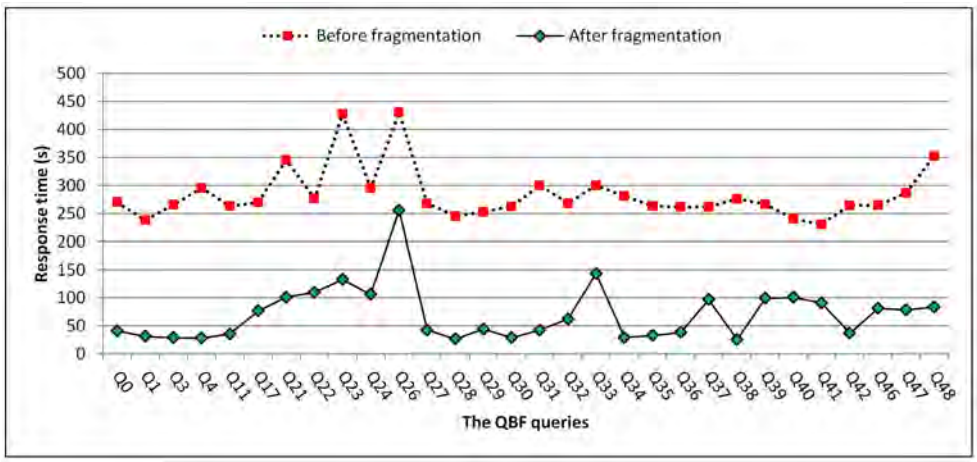


FIGURE 7.11 – Le temps de réponse des 30 requêtes bénéficiant de la fragmentation

- **Le cinquième point clé - La satisfaction de la Condition 5 (5.4)** : l'utilisation des prédicats de fragmentation sélectionnés sur la base des nombres d'occurrences contrôlés par le paramètre $NOMin$ (nombre d'occurrences minimal) lors de la génération du schéma de fragmentation horizontale, nous ont permis de réduire considérablement le nombre de fragments horizontaux. Ainsi, maintenir un petit nombre de fragments (par exemple 09 fragments dans la quatrième expérience) est une tâche facile et moins coûteuse pour l'administrateur de l'entrepôt de données.

Finalement, à partir des résultats satisfaisants obtenus dans toutes nos expérimentations, et à partir des points clés présentés ci-dessus, nous prouvons :

1. La validité de nos cinq conditions de quasi-optimalité définies dans la Section 5.4,
2. L'efficacité de nos solutions exploitant les nombres d'occurrences, les fréquences d'accès, et les sélectivités des prédicats de sélection proposées pour la satisfaction des conditions de quasi-optimalité, et
3. L'efficacité de notre algorithme de fragmentation horizontale de la table de faits.

Chapitre 8

Conclusion et perspectives

Sommaire

8.1 Conclusion

L'utilisation d'un système d'aide à la prise de décision au sein de toute entreprise, contribue significativement à son évolution économique. Les données d'un tel système sont extraites, transformées, intégrées, et stockées dans un entrepôt de données. L'optimisation des performances des requêtes interrogeant l'entrepôt de données pour des fins décisionnelles, a reçu et reçoit toujours une grande importance par les administrateurs des entrepôts de données. Cette optimisation peut être réalisée par différentes techniques, tels que : les index, les vues matérialisées, la fragmentation verticale, et la fragmentation horizontale des données. Dans la littérature, plusieurs techniques d'optimisation ont été proposées.

Dans le Chapitre 2.7, nous avons présenté quelques techniques de sélection d'index, de sélection de vues matérialisées, et de sélection de schéma de fragmentation verticale. Vu les avantages de la fragmentation horizontale par rapport aux autres techniques d'optimisation, nous avons, dans le Chapitre 3.4, étudié les différentes techniques de fragmentation horizontale dans les entrepôts de données. Toutefois, notre étude nous a permis de soulever plusieurs inconvénients que nous avons présenté dans le Chapitre 4.4 (Section 5.2), et parmi lesquels nous citons :

- Le taux d'optimisation de performances réalisé par ces approches de fragmentation est calculé pour l'ensemble des requêtes utilisées pour fragmenter l'entrepôt de données. Ce taux n'est pas partagé sur toutes ces requêtes, car parmi elles il y a celles qui ne bénéficient pas de la fragmentation et qui risque une augmentation de leur temps de traitement après la fragmentation horizontale. Malheureusement, ce type de requêtes (c.-à-d. les requêtes non bénéficiant de la fragmentation) n'est pas pris en compte par les techniques de fragmentation existantes,
- Le nombre de fragments horizontaux généré par ces techniques (Formule 5.1) augmente rapidement en fonction du nombre de dimensions fragmentées et de leur nombre de fragments. Ainsi :
  - ◇ Malgré le gain en performances pour les requêtes bénéficiant de la fragmentation, ce dernier reste toujours limité par le nombre d'opération de jointure entre un grand nombre de fragments horizontaux, en plus
  - ◇ L'augmentation du nombre de fragments rendra leur maintenance assez difficile pour l'administrateur de l'entrepôt de données.

Pour remédier à ces problèmes, dans le Chapitre 4.4 (Section 5.3), nous avons proposé :

- Une nouvelle formulation du problème de fragmentation horizontale des entrepôts de données. Contrairement aux techniques de fragmentation horizontale existantes, le pro-

blème de fragmentation horizontale selon notre formulation consiste à trouver un schéma de fragmentation horizontale de la table de faits sans fragmenter les tables de dimension. Le schéma de fragmentation recherché doit être composé d'un minimum de fragments horizontaux et doit offrir un taux élevé de performances des requêtes interrogeant l'entrepôt de données.

- En se basant sur cette nouvelle formulation et suivant un raisonnement mathématique basé sur des formules de calcul de coût, nous avons défini cinq conditions de quasi-optimalité qu'un schéma de fragmentation horizontale doit satisfaire.

Afin de générer le schéma de fragmentation horizontale vérifiant nos conditions de quasi-optimalité, c'est-à-dire un schéma de fragmentation : (i) composé d'un nombre réduit de fragments , (ii) engendrant un minimum de requêtes non bénéficiant de la fragmentation, (iii) préservant les performances des requêtes non bénéficiant de la fragmentation, (iv) générant des fragments horizontale de taille réduite par rapport à la tables de faits, nous avons développé, dans le Chapitre 5.6 un algorithme de fragmentation horizontale de la table de faits basé sur les prédicats de fragmentation. Nous avons sélectionné chaque prédicat de fragmentation parmi l'ensemble des prédicats de sélection en se basant sur son nombre d'occurrences, sa sélectivité, et sa fréquence d'accès. Pour assurer la transparence d'accès aux données de l'entrepôt après la fragmentation de la table de faits, nous avons, à la fin du Chapitre 5.6, développé une stratégie de réécriture et de traitement des requêtes. Ce dernier permet d'identifier pour chaque requête les fragments de faits appropriés à son traitement.
Afin de valider notre approche de fragmentation horizontale, nous avons mené, dans le Chapitre 7, une étude expérimentale. Les résultats (les taux d'amélioration de performances) de nos évaluations, nous ont permis de prouver l'efficacité de notre algorithme de fragmentation horizontale de la table de faits, dont le schéma de fragmentation généré : (i) répond au nouveau formalisme du problème de fragmentation, (ii) satisfait nos conditions de quasi-optimalité, et (iii) assure la transparence d'accès aux données. Afin de valider notre approche de fragmentation horizontale, nous avons mené, dans le Chapitre 7, une étude expérimentale. Les résultats (les taux d'amélioration de performances) de nos évaluations, nous ont permis de prouver l'efficacité de notre algorithme de fragmentation horizontale de la table de faits, dont le schéma de fragmentation généré : (i) répond au nouveau formalisme du problème de fragmentation, (ii) satisfait nos conditions de quasi-optimalité, et (iii) assure la transparence d'accès aux données.

8.2 Perspectives

Avec l'évolution de l'environnement de l'entreprise, les utilisateurs peuvent être confronté à des situations dans lesquelles des analyses sur des nouvelles vues de données peuvent être nécessaires pour améliorer la prise de décisions en faveur de l'entreprise. Dans ce cas, de nouvelle requêtes peuvent être ajoutées et d'autres peuvent être supprimées, ce qui modifie la charge de travail initiale utilisée par notre algorithme de fragmentation horizontale. Face cette situation, et selon notre approche de fragmentation horizontale, il est nécessaire de ré-exécuter notre algorithme de fragmentation horizontale pour la nouvelle charge de travail. Cependant, pour éviter cette ré-exécution, nous envisageons d'adapter notre algorithme de fragmentation, pour prendre en charge l'aspect dynamique de la charge de requêtes (c.-à-d. l'ajout ou la suppression de requêtes décisionnelles).
Nous envisageons aussi d'étudier la variation de la consommation d'énergie électrique par le serveur dans le quel l'entrepôt de données est stocké. Notre étude portera sur l'apport du schéma de fragmentation généré par notre algorithme de fragmentation horizontale sur la consommation d'énergie électrique lors du traitement des requêtes décisionnelles avant et après la fragmentation horizontale de l'entrepôt de données.
Nous rappelons que notre algorithme de fragmentation horizontale opère dans un environnement centralisé. Dans des travaux futurs, nous envisageons de compléter notre phase de fragmentation horizontale par une phase d'allocation des fragments horizontaux sur les différents sites répartis de l'entreprise.

Chapitre 9

Annexes

Sommaire

9.1 Annexe A

9.1.1 Scripts SQL du schéma de l'entrepôt de données du Benchmark APB-1

Dimension CHANLEVEL	`create table CHANLEVEL ( base_level CHAR(12) not null, all_level CHAR(12) )`
Dimension CUSTLEVEL	`create table CUSTLEVEL ( store_level CHAR(12) not null, retailer_level CHAR(12) not null )`
Dimension PRODLEVEL	`create table PRODLEVEL ( code_level CHAR(12) not null, class_level CHAR(12) not null, group_level CHAR(12) not null, family_level CHAR(12) not null, line_level CHAR(12) not null, division_level CHAR(12) not null )`
Dimension TIMELEVEL	`create table TIMELEVEL ( month_level VARCHAR2(12) not null, quarter_level VARCHAR2(12) not null, year_level VARCHAR2(12) not null )`
Table de faits ACTVARS	`create table ACTVARS ( customer_level CHAR(12) not null, product_level CHAR(12) not null, channel_level CHAR(12) not null, time_level VARCHAR2(12) not null, unitssold FLOAT not null, dollarsales FLOAT not null, dollarcost FLOAT )`

9.1.2 Charge de requêtes

Q_1	Select Time_level,count(*) From ACTVARS , PRODLEVEL , CUSTLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.store_level and PRODLEVEL.CLASS_LEVEL='MUEWH6P2QHES' and CUSTLEVEL.RETAILER_LEVEL='XJ3US59XVRL8' Group by Time_level
Q_2	Select Time_level,count(*) From ACTVARS, PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.CLASS_LEVEL='J107EVF3RMS8' Group by Time_level
Q_3	Select Time_level,count(*) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.CLASS_LEVEL='MORL7JL25XIC' Group by Time_level
Q_4	Select Time_level,count(*) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.GROUP_LEVEL='LF557Z6NW02S' Group by Time_level
Q_5	Select Time_level,count(*) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.GROUP_LEVEL='VBX3RSCXE18L' Group by Time_level
Q_6	Select Time_level,count(*) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.FAMILY_LEVEL='Y6NA46I43K63' Group by Time_level
Q_7	Select Time_level,sum(dollarcost) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.FAMILY_LEVEL='NTF8ZGW0P1ZQ' Group by Time_level

Q_8	Select Customer_level,count(*) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.FAMILY_LEVEL='S2G18UKN7FL2' Group by customer_level
Q_9	Select Customer_level,Avg(unitssold) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.FAMILY_LEVEL='UW619X7WJ66R' Group by Customer_level
Q_{10}	Select Product_level,Sum(dollarcost) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.LINE_LEVEL='VATPH9VKC58A' and PRODLEVEL.CLASS_LEVEL='J107EVF3RMS8' Group by Product_level
Q_{11}	Select Time_level,count(*) From ACTVARS , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.DIVISION_LEVEL='XEAQAD4P39SK' Group by Time_level
Q_{12}	Select Product_level,count(*) From ACTVARS , TIMELEVEL , PRODLEVEL Where ACTVARS.TIME_LEVEL=TIMELEVEL.MONTH_LEVEL and ACTVARS.product_level = PRODLEVEL.code_level and PRODLEVEL.DIVISION_LEVEL='XEAQAD4P39SK' and TIMELEVEL.YEAR_LEVEL=1995 Group by Product_level
Q_{13}	Select Product_level,count(*) From ACTVARS , TIMELEVEL , prodlevel Where ACTVARS.TIME_LEVEL=TIMELEVEL.MONTH_LEVEL and ACTVARS.product_level = PRODLEVEL.code_level and TIMELEVEL.YEAR_LEVEL=1995 and PRODLEVEL.FAMILY_LEVEL='Y6NA46I43K63' Group by Product_level
Q_{14}	Select Product_level,Sum(dollarcost) From ACTVARS , TIMELEVEL Where ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and TIMELEVEL.YEAR_LEVEL=1996 Group by Product_level

Q_{15}	Select Product_level,Avg(unitssold) From ACTVARS , TIMELEVEL Where ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and TIMELEVEL.QUARTER_LEVEL='1996Q2' Group by Product_level
Q_{16}	Select Product_level,ACTVARS.customer_level,Avg(unitssold) From ACTVARS , TIMELEVEL , custlevel Where ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and ACTVARS.customer_level= CUSTLEVEL.store_level and TIMELEVEL.QUARTER_LEVEL='1996Q2' and CUSTLEVEL.RETAILER_LEVEL='KGVR87N28SSM' Group by Product_level, ACTVARS.customer_level
Q_{17}	Select Product_level,Avg(unitssold) From ACTVARS , TIMELEVEL Where ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and TIMELEVEL.QUARTER_LEVEL='1995Q3' Group by Product_level
Q_{18}	Select Product_level,Avg(unitssold) From ACTVARS , TIMELEVEL , prodlevel Where ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and ACTVARS.product_level=PRODLEVEL.code_level and TIMELEVEL.QUARTER_LEVEL='1995Q3' and PRODLEVEL.GROUP_LEVEL='VBX3RSCXE18L' Group by Product_level
Q_{19}	Select Product_level,Customer_level, Max(unitssold) From ACTVARS , CUSTLEVEL Where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and CUSTLEVEL.RETAILER_LEVEL='XJ3US59XVRL8' Group by Product_level,Customer_level
Q_{20}	Select Product_level,Channel_level, count(*) From ACTVARS , CUSTLEVEL Where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and CUSTLEVEL.RETAILER_LEVEL='KGVR87N28SSM' Group by Product_level,Channel_level
Q_{21}	Select Product_level,Time_level, Min(unitssold) From ACTVARS , CUSTLEVEL Where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and CUSTLEVEL.RETAILER_LEVEL='B4V4B103X9RE' Group by Product_level,Time_level

Q_{22}	Select Product_level,Time_level, count(*) From ACTVARS , CHANLEVEL Where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and CHANLEVEL.ALL_LEVEL='AAAAAAAAAAA' Group by Product_level, Time_level
Q_{23}	Select Channel_level,Time_level, Sum(dollarcost) From ACTVARS , CHANLEVEL Where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and CHANLEVEL.ALL_LEVEL='BBBBBBBBBBB' Group by Channel_level, Time_level
Q_{24}	Select Channel_level,Time_level, Sum(dollarcost) From ACTVARS , CHANLEVEL Where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and CHANLEVEL.ALL_LEVEL='CCCCCCCCCCC' Group by Channel_level, Time_level
Q_{25}	Select Product_level,Channel_level, Time_level,count(*) From ACTVARS , CHANLEVEL Where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and CHANLEVEL.ALL_LEVEL='DDDDDDDDDDD' Group by Product_level,Channel_level, Time_level
Q_{26}	Select Customer_level,Channel_level, Time_level, Avg(unitssold) From ACTVARS , CHANLEVEL , CUSTLEVEL Where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and ACTVARS.customer_level = CUSTLEVEL.store_level and CHANLEVEL.ALL_LEVEL='CCCCCCCCCCC'and CUSTLEVEL.RETAILER_LEVEL='B4V4B103X9RE' Group by Customer_level,Channel_level, Time_level
Q_{27}	Select Customer_Level,Time_level, sum(dollarcost) From ACTVARS , CUSTLEVEL , PRODLEVEL Where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and PRODLEVEL.CLASS_LEVEL='J107EVF3RMS8'and CUSTLEVEL.RETAILER_LEVEL='KGVR87N28SSM' Group by Customer_level,Time_level

Q_{28}	Select Customer_Level, Channel_level, Time_level, sum (dollarcost) From ACTVARS , PRODLEVEL , TIMELEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.TIME_LEVEL= TIMELEVEL.month_level and TIMELEVEL.YEAR_LEVEL=1996 and PRODLEVEL.CLASS_LEVEL='MORL7JL25XIC' Group by Customer_level, Channel_level, Time_level
Q_{29}	Select Customer_Level, Time_level, Avg(unitssold) From ACTVARS , CHANLEVEL , PRODLEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and PRODLEVEL.CLASS_LEVEL='MORL7JL25XIC' and CHANLEVEL.ALL_LEVEL='AAAAAAAAAAAA' Group by Customer_level, Time_level
Q_{30}	Select Customer_Level, sum (dollarcost), Avg(unitssold) From ACTVARS , CUSTLEVEL , TIMELEVEL Where TIMELEVEL.YEAR_LEVEL=1996 and CUSTLEVEL.RETAILER_LEVEL='KGVR87N28SSM' and ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and ACTVARS.TIME_LEVEL=TIMELEVEL.month_level Group by Customer_level
Q_{31}	Select Customer_Level, Product_level, Time_level, Sum (dollarcost) From actvars , CUSTLEVEL , PRODLEVEL , TIMELEVEL Where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and TIMELEVEL.YEAR_LEVEL=1995 and CUSTLEVEL.RETAILER_LEVEL='B4V4B103X9RE' and PRODLEVEL.LINE_LEVEL='VATPH9VKC58A' Group by Customer_level,Product_level, Time_level
Q_{32}	Select Customer_Level, Product_level, Channel_level,Avg(unitssold) From ACTVARS , CHANLEVEL , PRODLEVEL , TIMELEVEL Where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and TIMELEVEL.QUARTER_LEVEL='1996Q2' and PRODLEVEL.DIVISION_LEVEL='XEAQAD4P39SK' and CHANLEVEL.ALL_LEVEL='BBBBBBBBBBB' Group by Customer_level, Product_level, Channel_level

Q_{33}	Select Customer_Level, Product_level, Time_level, Channel_level, Min (unitssold) from ACTVARS , CHANLEVEL , CUSTLEVEL , PRODLEVEL , TIMELEVEL Where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and TIMELEVEL.YEAR_LEVEL=1996 and PRODLEVEL.CLASS_LEVEL='J107EVF3RMS8' and PRODLEVEL.GROUP_LEVEL='VBX3RSCXE18L'and PRODLEVEL.FAMILY_LEVEL='UW619X7WJ66R'and CUSTLEVEL.RETAILER_LEVEL='XJ3US59XVRL8' and CHANLEVEL.ALL_LEVEL='AAAAAAAAAAAA' Group by Customer_level, Product_level, Time_level, Channel_level
Q_{34}	select CLASS_LEVEL,RETAILER_LEVEL,YEAR_LEVEL,sum(ACTVARS.unitssold) from ACTVARS , CHANLEVEL , CUSTLEVEL , PRODLEVEL , TIMELEVEL where ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and ACTVARS.TIME_LEVEL=TIMELEVEL.month_level and PRODLEVEL.DIVISION_LEVEL='XEAQAD4P39SK' and custlevel.RETAILER_LEVEL='KGVR87N28SSM' and chanlevel.ALL_LEVEL='BBBBBBBBBBB' and timelevel.YEAR_LEVEL=1995 Group by CLASS_LEVEL,RETAILER_LEVEL,YEAR_LEVEL

9.1.3 Réécriture des requêtes après la fragmentation horizontale : Exemple

Avant la fragmentation horizontale	Après la fragmentation horizontale

Q_1
`select Time_level,count(*)`
`from ACTVARS , PRODLEVEL , CUSTLEVEL`
`where`
`ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and`
`ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.store_level and`
`PRODLEVEL.CLASS_LEVEL='MUEWH6P2QHES' and`
`CUSTLEVEL.RETAILER_LEVEL='XJ3US59XVRL8'`
`group by Time_level`
	Q_1 bénéficiant de la fragmentation
	`select TIME_LEVEL,COUNT(*)`
	`from`
	`(select * from Frag_p15 union all`
	`select * from Frag_Duplique) ACTVARS,`
	`PRODLEVEL,CUSTLEVEL`
	`where`
	`ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and`
	`ACTVARS.CUSTOMER_LEVEL=CUSTLEVEL.STORE_LEVEL and`
	`PRODLEVEL.CLASS_LEVEL='MUEWH6P2QHES' and`
	`CUSTLEVEL.RETAILER_LEVEL='XJ3US59XVRL8'`
	`group by TIME_LEVEL`

Q_4
`select Time_level,count(*)`
`from ACTVARS , PRODLEVEL`
`where`
`ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and`
`PRODLEVEL.GROUP_LEVEL='LF557Z6NW02S'`
`group by Time_level`
	Q_4 non bénéficiant de la fragmentation
	`select TIME_LEVEL,COUNT(*)`
	`from (`
	`select * from Frag_p5 union all`
	`select * from Frag_p15 union all`
	`select * from Frag_p14 union all`
	`select * from Frag_p2 union all`
	`select * from Frag_p10 union all`
	`select * from Frag_p17 union all`
	`select * from Frag_p1 union all`
	`select * from Frag_p16 union all`
	`select * from Frag_Duplique union all`
	`select * from Frag_Complementaire )`
	`ACTVARS,PRODLEVEL`
	`where`
	`ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and`
	`PRODLEVEL.GROUP_LEVEL='LF557Z6NW02S'`
	`group by TIME_LEVEL`

9.2 Annexe B

9.2.1 Scripts SQL des fragments horizontaux

9.2.1.1 Fragments horizontaux générés par les prédicats de fragmentation

```
create table Frag_p2 as
 select actvars.*
from  actvars , PRODLEVEL
where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and
PRODLEVEL.Class_LEVEL='MORL7JL25XIC';

create table Frag_p17 as
 select actvars.* from
 actvars , CUSTLEVEL
 where ACTVARS.customer_level=CUSTLEVEL.store_level and
CUSTLEVEL.RETAILER_LEVEL='B4V4B103X9RE';

 create table Frag_p1 as
 select actvars.* from
 actvars , PRODLEVEL
 where ACTVARS.PRODUCT_LEVEL=PRODLEVEL.CODE_LEVEL and
PRODLEVEL.Class_LEVEL='J107EVF3RMS8';

 create table Frag_p21 as
 select actvars.* from
 actvars , CHANLEVEL
 where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and
CHANLEVEL.ALL_LEVEL='DDDDDDDDDDD';

 create table Frag_p19 as
 select actvars.* from
 actvars , CHANLEVEL
 where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and
CHANLEVEL.ALL_LEVEL='BBBBBBBBBBB';

 create table Frag_p18 as
 select actvars.* from
 actvars , CHANLEVEL
 where ACTVARS.CHANNEL_LEVEL=CHANLEVEL.BASE_LEVEL and
CHANLEVEL.ALL_LEVEL='AAAAAAAAAAA';

 create table Frag_p16 as
 select actvars.* from
 actvars , CUSTLEVEL
 where ACTVARS.customer_level=CUSTLEVEL.store_level and
CUSTLEVEL.RETAILER_LEVEL='KGVR87N28SSM';
```

9.2.1.2 Fragment horizontal des tuples dupliqués FHD

```
/* Vérification de la Disjonction Entre Fragments Horizontaux */
/* Création de la table de l'union de tous les fragments horizontaux */

create table Frag_UNION (
```

```
customer_level CHAR(12) not null,
product_level  CHAR(12) not null,
channel_level  CHAR(12) not null,
time_level     VARCHAR2(12) not null,
unitssold      FLOAT not null,
dollarsales    FLOAT not null,
dollarcost     FLOAT);


 INSERT /*+ APPEND */ INTO Frag_union
 select * from frag_p2;
 COMMIT;
 INSERT /*+ APPEND */ INTO Frag_union
 select * from frag_p10;
 COMMIT;
 INSERT /*+ APPEND */ INTO Frag_union
 select * from frag_p17;
 COMMIT;
 INSERT /*+ APPEND */ INTO Frag_union
 select * from frag_p1;
 COMMIT;
 INSERT /*+ APPEND */ INTO Frag_union
 select * from frag_p16;
 COMMIT;

/* Création du fragment des tuples dupliqués */

create table Frag_Duplique as
select  t.customer_level,t.product_level,t.channel_level,
t.time_level,t.unitssold,t.dollarsales,t.dollarcost,
count(*) aa
from Frag_union t
group by t.customer_level,t.product_level,t.channel_level,
   t.time_level,t.unitssold,t.dollarsales,t.dollarcost
having count(*)>1 ;

/* Raffinage des fragments horizontaux (surpression des tuples dupliqués) */

/* 1 - Création de la nouvelle table du fragment */
 create table frag_p2_new as
 select t1.customer_level,t1.product_level,t1.channel_level,
        t1.time_level,t1.unitssold,t1.dollarsales,t1.dollarcost
 from Frag_p2  t1  WHERE NOT EXISTS
(
 SELECT 1  FROM Frag_Duplique  d
 WHERE d.customer_level = t1.customer_level and
 d.product_level = t1.product_level   and
 d.channel_level = t1.channel_level   and
 d.time_level = t1.time_level
);

/* 2- supprimer la table du fragment actuel */

   drop table frag_p2;

/* 3- renommer la nouvelle table par le nom du fragment */
```

```
 alter table    frag_p2_new rename to    frag_p2;
```

Répéter les opérations 1,2, et 3 pour chaque fragment horizontal

9.2.1.3 Fragment horizontal complémentaire FHC

```
create table frag_complementaire as
select t1.*
from   actvars t1   WHERE not EXISTS  (
         SELECT 1
FROM  Frag_union d
WHERE d.customer_level = t1.customer_level and
      d.product_level = t1.product_level   and
d.channel_level = t1.channel_level   and
d.time_level = t1.time_level );
```

9.2.2 Schéma de fragmentation horizontale

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Schéma de Fragmentation Horizontale de La Table de Faits-->
<SchemaFragmentaion>
  <Fragment id="Frag_p2" Predicat="p2">
    <Requetes id="Q28"/>
    <Requetes id="Q27"/>
    <Requetes id="Q2"/>
  </Fragment>
  <Fragment id="Frag_p10" Predicat="p10">
    <Requetes id="Q31"/>
    <Requetes id="Q11"/>
    <Requetes id="Q10"/>
  </Fragment>
  <Fragment id="Frag_p17" Predicat="p17">
    <Requetes id="Q30"/>
    <Requetes id="Q25"/>
    <Requetes id="Q20"/>
  </Fragment>
  <Fragment id="Frag_p1" Predicat="p1">
    <Requetes id="Q32"/>
    <Requetes id="Q9"/>
    <Requetes id="Q1"/>
  </Fragment>
  <Fragment id="Frag_p16" Predicat="p16">
    <Requetes id="Q33"/>
    <Requetes id="Q29"/>
    <Requetes id="Q26"/>
    <Requetes id="Q19"/>
    <Requetes id="Q15"/>
  </Fragment>
</SchemaFragmentaion>
```

Bibliographie

Atzeni, P., Ceri, S., Paraboschi, S. and Torlone, R. (1999). *Database Systems - Concepts, Languages and Architectures*, McGraw-Hill Book Company. 15, 19

Baralis, E., Paraboschi, S. and Teniente, E. (1997). Materialized views selection in a multidimensional database, *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 156–165. 37

Barsky, D. and Benzaghou, B. (2004). Nombres de bell et somme de factorielles, *Journal de théorie des nombres de Bordeaux* **16**(1) : 1–17. 40

Bellatreche, L. (2012). Dimension table selection strategies to referential partition a fact table of relational data warehouses, *in* T. Özyer, K. Kianmehr and M. Tan (eds), *Recent Trends in Information Reuse and Integration*, Springer Vienna, Vienna, pp. 19–41. 57, 62, 67, 74

Bellatreche, L. and Boukhalfa, K. (2005). An evolutionary approach to schema partitioning selection in a data warehouse, *Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK'05, Springer-Verlag, Berlin, Heidelberg, pp. 115–125. 52, 54, 56, 61, 66, 67, 68, 70, 74

Bellatreche, L., Boukhalfa, K. and Abdalla, H. I. (2006). Saga : A combination of genetic and simulated annealing algorithms for physical data warehouse design, *Proceedings of the 23rd British National Conference on Databases*, Springer-Verlag, Berlin, Heidelberg, pp. 212–219. 54, 61, 67, 70, 74

Bellatreche, L., Boukhalfa, K. and Richard, P. (2008). Data partitioning in data warehouses : Hardness study, heuristics and oracle validation., *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*, Springer-Verlag, Berlin, Heidelberg, pp. 87–96. 55, 58, 61, 67, 69, 70, 71, 74

Bellatreche, L., Karlapalem, K., Mohania, M. K. and Schneider, M. (2000). What can partitioning do for your data warehouses and data marts ?, *Proceedings of the 2000 International Symposium on Database Engineering & Applications*, IDEAS '00, IEEE Computer Society, Washington, DC, USA, pp. 437–446. 68, 73

Bellatreche, L., Missaoui, R., Necir, H. and Drias, H. (2007). A data mining approach for selecting bitmap join indices, *JCSE* **1**(2) : 177–194. 27

Bouakkaz, M., Ouinten, Y. and Ziani, B. (2012). Vertical fragmentation of data warehouses using the fp-max algorithm, *Proceedings of the Innovations in Information Technology (IIT), 2012 International Conference on*, IEEE, AL AIN, UAE, pp. 273–276. 27, 45, 46

Bouchakri, R., Bellatreche, L. and Faget, Z. (2014). Algebra-based approach for incremental data warehouse partitioning, *in* H. Decker, L. Lhotská, S. Link, M. Spies and R. R. Wagner (eds), *Database and Expert Systems Applications : 25th International Conference, DEXA 2014, Munich, Germany, September 1-4, 2014. Proceedings, Part II*, Springer International Publishing, Cham, pp. 441–448.
URL: *http ://dx.doi.org/10.1007/978-3-319-10085-2_40* 58, 62, 67, 70, 74

Chmiel, J., Morzy, T. and Wrembel, R. (2009). Hobi : Hierarchically organized bitmap index for indexing dimensional data, *in* T. B. Pedersen, M. K. Mohania and A. M. Tjoa (eds), *Data Warehousing and Knowledge Discovery : 11th International Conference, DaWaK 2009 Linz, Austria, August 31–September 2, 2009 Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 87–98. 27, 35

Claudia Imhoff, Nicholas Galemmo, J. G. G. (2003). *Mastering Data Warehouse Design : Relational and Dimensional Techniques*, Wiley. 28

Cuzzocrea, A., Darmont, J. and Mahboubi, H. (2009). Fragmenting very large xml data warehouses via k-means clustering algorithm, *International Journal of Business Intelligence and Data Mining* **4**(3/4) : 301–328. 67

Dimovski, A., Velinov, G. and Sahpaski, D. (2010). Horizontal partitioning by predicate abstraction and its application to data warehouse design, *in* B. Catania, M. Ivanović and B. Thalheim (eds), *Advances in Databases and Information Systems : 14th East European Conference, ADBIS 2010, Novi Sad, Serbia, September 20-24, 2010. Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 164–175. 56, 62, 67, 70, 74

Elmasri, R. and Navathe, S. (2010). Chapter 17 - disk storage, basic file structures, and hashing, *in* R. Elmasri and S. Navathe (eds), *Fundamentals of Database Systems*, 6th edn, Addison-Wesley Publishing Company, USA. 76

Gacem, A. and Boukhalfa, K. (2014). Handling very large workloads to effectively partition data warehouses : New approach and experimental study, *Information and Communication Systems (ICICS), 2014 5th International Conference on*, pp. 1–6. 58, 63, 67, 74

Garcia-Molina, H., Ullman, J. D. and Widom, J. (2008). *Database Systems : The Complete Book*, 2 edn, Prentice Hall Press, Upper Saddle River, NJ, USA. 17

Ghorbel, M., Karima, T. and Abdellatif, A. (2016). Réduction du nombre des prédicats pour les approches de répartition des entrepôts de données, *Ingénierie des Systèmes d'Information* **21**(1) : 81–102. 27, 59, 63, 67, 74

Golfarelli, M. and Rizzi, S. (2009). *Data Warehouse Design : Modern Principles and Methodologies*, 1 edn, McGraw-Hill, Inc., New York, NY, USA. 9, 11

Gosain, A. and Heena (2016). Materialized cube selection using particle swarm optimization algorithm, *Procedia Computer Science* **79** : 2–7. 27, 37, 39

Goswami, R., Bhattacharyya, D. K., Dutta, M. and Kalita, J. K. (2016). Approaches and issues in view selection for materialising in data warehouse, *Int. J. of Business Information Systems* **21**(1) : 17–47. 37, 39

Gupta, H., Harinarayan, V., Rajaraman, A. and Ullman, J. D. (1997). Index selection for olap, *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, IEEE Computer Society, Washington, DC, USA, pp. 208–219. 34

Gupta, H. and Mumick, I. S. (1999). Selection of views to materialize under a maintenance cost constraint, *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, Springer-Verlag, London, UK, UK, pp. 453–470.
URL: *http ://dl.acm.org/citation.cfm ?id=645503.656261* 27, 37

Harinarayan, V., Rajaraman, A. and Ullman, J. D. (1996). Implementing data cubes efficiently, *SIGMOD Rec.* **25**(2) : 205–216.
URL: *http ://doi.acm.org/10.1145/235968.233333* 27, 37

Hobbs, L., Hillson, S., Lawande, S. and Smith, P. (2005a). 4 - physical design of the data warehouse, *in* L. Hobbs, S. Hillson, S. Lawande and P. Smith (eds), *Oracle 10g Data Warehousing*, 1st edn, Digital Press, Burlington, pp. 113 – 154. 14, 42

Hobbs, L., Hillson, S., Lawande, S. and Smith, P. (2005b). Chapter 15 - OLAP, *in* L. Hobbs, S. Hillson, S. Lawande and P. Smith (eds), *Oracle 10g Data Warehousing*, Digital Press, Burlington, pp. 671 – 724. 18

Inmon, W. H. (1992). *Building the Data Warehouse*, 1rd edn, John Wiley & Sons, Inc., New York, NY, USA. 12

Jarboui, B., Cheikh, M., Siarry, P. and Rebai, A. (2007). Combinatorial particle swarm optimization (cpso) for partitional clustering problem, *Applied Mathematics and Computation* **192**(2) : 337 – 345.
URL: *http ://www.sciencedirect.com/science/article/pii/S0096300307003347* 59

Jonathan, L. (2006). *Cost-Based Oracle Fundamentals*, Apress, Berkeley, CA, USA. 79, 80

Jong Wook, K. and Sae-Hong, C. (2016). Workload-based column partitioning to efficiently process data warehouse query, *Int. J. of Applied Engineering Research* **11**(2) : 917–921. 27, 45, 46

Kechar, M. and Nait-Bahloul, S. (2014). Hybrid fragmentation of xml data warehouse using k-means algorithm, *in* Y. Manolopoulos, G. Trajcevski and M. Kon-Popovska (eds), *Advances in Databases and Information Systems : 18th East European Conference, ADBIS 2014, Ohrid, Macedonia, September 7-10, 2014. Proceedings*, Springer International Publishing, Cham, pp. 70–82. 67

Kechar, M. and Nait-Bahloul, S. (2017). Performance optimisation of the decision-support queries by the horizontal fragmentation of the data warehouse, *International Journal of Business Information Systems* **26**(4) : 506–537. 5, 72, 83

Kimball, R. and Ross, M. (2013). *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, 3rd edn, Wiley Publishing, New York, NY, USA. 9, 14, 15

Kitsuregawa, M., Tanaka, H. and Moto-Oka, T. (1983). Application of hash to data base machine and its architecture, *New Generation Comput.* **1**(1) : 63–74. 75

Matalqa, S. and Mustafa, S. (2016). The effect of horizontal database table partitioning on query performance, *Int. Arab J. Inf. Technol.* **13**(1A) : 184–189. 60

Morzy, T., Wrembel, R., Chmiel, J. and Wojciechowski, A. (2012). Time–hobi : Index for optimizing star queries, *Information Systems* **37**(5) : 412 – 429. 36

Noaman, A. Y. and Barker, K. (1999). A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse, *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, ACM, New York, NY, USA, pp. 154–161. 51, 61, 67, 74

OLAP-Council (1998). Apb-1 olap benchmark, release ii. [Online] http ://www.olapcouncil.org/research/bmarkly.htm (accessed 14 October 2015). 99

O'Neil, P. and Graefe, G. (1995). Multi-table joins through bitmapped join indices, *SIGMOD Rec.* **24**(3) : 8–11.
URL: *http ://doi.acm.org/10.1145/211990.212001* 32

O'Neil, P. and Quass, D. (1997). Improved query performance with variant indexes, *SIGMOD Rec.* **26**(2) : 38–49. 28

Ozsu, M. T. and Valduriez, P. (1991). *Principles of Distributed Database Systems*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 50, 59

Özsu, M. T. and Valduriez, P. (2011). *Principles of Distributed Database Systems, Third Edition*, Springer, New York. 40, 41, 83, 93

Ponniah, P. (2011). *Data Warehousing Fundamentals for IT Professionals,2 nd Edition*, Wiley.
URL: *https ://books.google.dz/books ?id=aIDjcF9r_hgC* 11, 14, 40, 47

Ramakrishnan, R. and Gehrke, J. (2003). *Database Management Systems*, 3 edn, McGraw-Hill, Inc., New York, NY, USA. 28, 29

Shapiro, L. D. (1986). Join processing in database systems with large main memories, *ACM Trans. Database Syst.* **11**(3) : 239–264. 75

Sohail, A. and Dominic, P. D. D. (2015). From ER Model to Star Model : A Systematic Transformation Approach, *Int. J. of Business Information Systems* **18**(3) : 249–267. 15

Sohrabi, M. K. and Ghods, V. (2016). Materialized view selection for a data warehouse using frequent itemset mining, *Journal of Computers* **11**(2) : 140–148. 37

Systems, R. (1997). Star schema processing for complex queries, White Paper. 31

Talebi, Z. A., Chirkova, R. and Fathi, Y. (2013). An integer programming approach for the view and index selection problem, *Data & Knowledge Engineering* **83** : 111 – 125.
URL: *http ://www.sciencedirect.com/science/article/pii/S0169023X12001000* 27, 34

Tan, P.-N., Steinbach, M. and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 59

Thenmozhi, M. and Vivekanandan, K. (2014). A comparative analysis of fragmentation selection algorithms for data warehouse partitioning, *Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on*, pp. 1–5. 27, 57, 62, 67, 69, 70, 74

Theodorou, V., Abelló, A., Lehner, W. and Thiele, M. (2016). Quality measures for ETL processes : from goals to implementation, *Concurrency and Computation : Practice and Experience* **28**(15) : 3969–3993. cpe.3729.
URL: *http ://dx.doi.org/10.1002/cpe.3729* 11

Toumi, L., Moussaoui, A. and Ugur, A. (2015). Emed-part : An efficient methodology for horizontal partitioning in data warehouses, *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication*, IPAC '15, ACM, New York, NY, USA, pp. 1–7.
URL: *http ://doi.acm.org/10.1145/2816839.2816876* 59, 63, 67, 70, 74

Valduriez, P. (1987). Join indices, *ACM Trans. Database Syst.* **12**(2) : 218–246. **URL:** *http ://doi.acm.org/10.1145/22952.22955* 30, 31, 80

Vanichayobon, S. and Gruenwald, L. (1999). Indexing techniques for data warehouses' queries, *Technical report*, The University of Oklahoma, School of Computer Science. 32

Wagner, N. and Agrawal, V. (2013). Using an evolutionary algorithm to solve the weighted view materialisation problem for data warehouses, *Int. J. Intell. Inf. Database Syst.* **7**(2) : 163–179. 27, 38

Ward, Joe H., J. (1963). Hierarchical grouping to optimize an objective function, *Journal of the American Statistical Association* **58**(301) : 236–244. 59

Wojciechowski, A. and Wrembel, R. (2014). On index structures for star query processing in data warehouses, *in* E. Zimányi (ed.), *Business Intelligence : Third European Summer School, eBISS 2013, Dagstuhl Castle, Germany, July 7-12, 2013, Tutorial Lectures*, Springer International Publishing, Cham, pp. 182–217. **URL:** *http ://dx.doi.org/10.1007/978-3-319-05461-2_6* 27, 36

Wrembel, R. (2012). Data warehouse performance : selected techniques and data structures, *Business Intelligence*, Springer, pp. 27–62. 27, 42

Zamanian, E., Binnig, C. and Salama, A. (2015). Locality-aware partitioning in parallel database systems, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, ACM, New York, NY, USA, pp. 17–30. 27, 58, 63, 74

Zhixiang, Z. and Dongjin, Y. (CCA 2014). Selecting materialized views based on genetic algorithm, *Advanced Science and Technology Letters* **45**(Complete) : 65–69. 37

Ziani, B. and Ouinten, Y. (2013). An improved approach for automatic selection of multi-tables indexes in ralational data warehouses using maximal frequent itemsets, *Intelligent Decision Technologies* **7**(4) : 279–292. 27, 33

ZIYATI, E., DRISS, A. and EL QADI, A. (2008). Algorithms for data warehouse design to enhance decision-making, *WSEAS Transactions on Computer research* **3**(3) : 111–120. 47

ملخص

تعتبر التجزئة الأفقية لمستودعات البيانات أحد التقنيات الرئيسية لتحسين أداء استعلامات دعم اتخاذ القرارات. لا يتحقق هذا التحسين إلا إذا تم تجزئة الحجم الكبير من البيانات الموجودة في جدول الحقائق أفقيا. و لهذا السبب فإن أجزاء جدول الحقائق تستمد دائما من أجزاء جداول الأبعاد. لسوء الحظ، في هذا النوع من التجزئة، يمكن أن يزداد عدد الأجزاء زيادة كبيرة وتصبح صيانتهم صعبة ومكلفة للغاية. وبالتالي، للحد من عدد من الأجزاء ومواصلة تحسين أداء استعلامات دعم اتخاذ القرار، نقترح أن نجزء أفقيا جدول الحقائق من خلال الاستغلال المشترك: الانتقائية من المسندات الاختيار، وأعداد حدوثها، و ترددات النفاذ. وقد أثبتت النتائج التي تحصلنا عليها فعالية منهجيتنا في التجزئة الأفقية لمستودع البيانات.

الكلمات المفتاحية: نظام دعم القرار؛ مستودع البيانات؛ تحسين أداء طلب البحث؛ تجزئة أفقية؛ مسندات الاختيار؛ مسندات. التجزئة

Abstract

The horizontal fragmentation of the data warehouse is considered as one of the important performance optimisation techniques of the decision-support queries. This optimisation is reached only if the large data volume of the fact table is horizontally fragmented. For that, the fragments of the fact table are always derived from the fragments of the dimension tables. Unfortunately, in this type of fragmentation, the fragments number can dramatically increase, and their maintenance becomes quite hard and costly. Thus, to reduce the number of the fragments and to further optimise the decision-support queries performances, we propose to fragment horizontally only the fact table by exploiting jointly: the selectivities of the selection predicates, their occurrence numbers, and their access frequencies. The results we have obtained have proven the effectiveness of our horizontal fragmentation approach to of the data warehouse.

Keywords: Decision support system; Data warehouse; Query performance optimisation; Horizontal fragmentation; Selection predicates; Fragmentation predicates.

Résumé

La fragmentation horizontale des entrepôts de données est considérée comme l'une des principales techniques d'optimisation des performances des requêtes d'aide à la décision. Cette optimisation n'est atteinte que si le gros volume de données de la table de faits est fragmenté horizontalement. De ce fait, les fragments de la table de faits sont toujours dérivés à partir des fragments des tables dimension. Malheureusement, dans ce type de fragmentation, le nombre de fragments peut augmenter considérablement et leur maintenance devient très difficile et coûteuse. Ainsi, pour réduire le nombre de fragments et optimiser davantage les performances des requêtes d'aide à la décision, nous proposons de fragmenter horizontalement que la table de faits en exploitant conjointement: les sélectivités des prédicats de sélection, leurs nombre d'occurrences et leurs fréquences d'accès. Les résultats que nous avons obtenus, ont prouvé l'efficacité de notre approche de fragmentation horizontale de l'entrepôt de données.

Mots clés : Système d'aide à la décision; Entrepôt de données; l'Optimisation des performances de requête; Fragmentation horizontale; Prédicats de sélection; Prédicats de fragmentation.

Résumé

La fragmentation horizontale des entrepôts de données est considérée comme l'une des principales techniques d'optimisation des performances des requêtes d'aide à la décision. Cette optimisation n'est atteinte que si le gros volume de données de la table de faits est fragmenté horizontalement. De ce fait, les fragments de la table de faits sont toujours dérivés à partir des fragments des tables dimension. Malheureusement, dans ce type de fragmentation, le nombre de fragments peut augmenter considérablement et leur maintenance devient très difficile et coûteuse. Ainsi, pour réduire le nombre de fragments et optimiser davantage les performances des requêtes d'aide à la décision, nous proposons de fragmenter horizontalement que la table de faits en exploitant conjointement: les sélectivités des prédicats de sélection, leurs nombre d'occurrences et leurs fréquences d'accès. Les résultats que nous avons obtenus, ont prouvé l'efficacité de notre approche de fragmentation horizontale de l'entrepôt de données.

Mots clés :

Système d'aide à la décision; Entrepôt de données; Optimisation des performances de requête; Fragmentation horizontale; Prédicats de sélection; Prédicats de fragmentation ; Nombre d'occurrences d'un prédicat de sélection; Sélectivité d'un prédicat de sélection; Fréquence d'accès d'un prédicat de sélection; Requêtes non bénéficiant de la fragmentation.