N° D'ORDRE : 05/2011-M/INF

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
« HOUARI BOUMEDIENE »
DEPARTEMENT D'INFORMATIQUE

Mémoire

Pour l'obtention du diplôme de Magistère

EN : INFORMATIQUE

Option : *Programmation et Systèmes*

Présenté par :

Ibtissam FRIHI

Distribution d'espace disque entre les vues matérialisées et les index dans un entrepôt de données

Soutenu publiquement le 31/05/2011, devant le jury composé de :

Mme- Dalila BOUGHACI	Maître de Conférences /A, à l' USTHB	Président
Mme- Habiba DRIAS	Professeur, à l' USTHB	Directeur de mémoire
Mme- Nadjet KAMEL	Maître de Conférences /A, à l' USTHB	Examinateur
Mr- Samir KECHID	Maître de Conférences /A, à l' USTHB	Examinateur
Mr- Kamel BOUKHALFA	Maître de Conférences /A, à l' USTHB	Examinateur

Remerciements

« Louange à Dieu le tout puissant qui nous a aidé à achever ce travail de recherche, et que le salut soit sur son prophète Mohamed ».

Je remercie Mme Drias Habiba , Professeur à l'USTHB, Alger, pour m'avoir ouvert l'opportunité de travailler sous sa direction, pour m'avoir permis, par la même occasion, de mettre un pas dans le grand monde de la recherche, pour sa disponibilité, et ses observations qui ont contribué à améliorer la qualité de ce travail.

Je tiens à remercier très sincèrement l'ensemble des membres du jury qui me font le grand honneur d'accepter de juger mon travail.

Ma profonde reconnaissance va à mes parents qui m'ont toujours soutenue, toujours encouragée. Leur appui constant m'a permis d'accomplir de grandes choses et de partir bien préparée pour la vie. Merci de m'avoir supporté durant toutes mes études universitaires.

Merci à ma famille pour tous les moments de bonheur partagés.

A mes amis de tout temps, à celles et ceux qui ont su trouver les mots pour m'aider à franchir les obstacles et à avancer dans la vie.

Au terme de ce modeste travail, je voudrais adresser mes vifs remerciements à toutes les personnes qui ont contribué, de près ou de loin, à accomplir ce présent travail.

Résumé

Le volume d'information contenu dans un entrepôt de données est très important, d'où la prise de décision et son efficacité est devenue cruciale pour l'administrateur de l'entrepôt de données. L'interrogation des données de l'entrepôt se fait par des requêtes très complexes vu le nombre d'agrégation et de jointures quelle utilise. Ce qui implique un temps de réponse très important.

Afin de réduire le temps d'exécution de ces requêtes plusieurs techniques ont été proposées dans la littérature, à savoir les vues matérialisées, les index, la fragmentation. La matérialisation des vues et l'indexation sont deux techniques les plus efficaces pour améliorer le temps de réponse des requêtes.

Dans ce travail, nous proposons une approche de sélection des index binaires de jointures et une autre pour la sélection des vues dans un entrepôt de données ; tout en vérifiant la contrainte d'espace de stockage. Nos approches sont basées sur l'utilisation des méta-heuristiques à savoir les colonies de fourmis. Une étude expérimentale et des tests comparatifs proposés, montrant l'efficacité de notre approche.

Mots clés : Entrepôt de données, vues matérialisées, index binaire de jointures, espace de stockage, PSI, PSV, Colonie de fourmis.

Abstract

The volume of information contained in a warehouse of data is very significant, from where the decision-making and its effectiveness became crucial for the administrator of the warehouse of data. The interrogation of the data of the warehouse is made by very complex requests because of the huge number of aggregations and joints used, which provokes a very significant response time.

In order to reduce the execution time of these requests several techniques were proposed in the literature, namely the materialized sights, the indices and fragmentation. Materialization of the sights and the indexing are two techniques that are most effective to improve the response time of the requests.

In this work, we propose an approach of selection of the binary joint index and another for the selection of the sights in a data warehouse; under the constraint of checking the space of storage. Our approach is based on the use of meta-heuristic of the colonies of ants. An experimental study and comparative tests are performed; they show the effectiveness of our approach.

Key words: Data warehouse, materialized views selection, binary joint index selection, spaces storage, Colony of ants.

Sommaire

Liste des figures

Introduction générale

I. Introduction et problèmatique

L'évolution fulgurante des technologies a permis de générer de très grandes quantités de données produites et manipulées par les entreprises. Le besoin de faire plus que de simples traitements sur ces données s'est fait alors ressentir. De ce fait, les systèmes décisionnels ont été élaborés d'un besoin des entreprises à fournir aux décideurs des moyens d'accéder aux données de leurs propres systèmes dans le but de piloter leurs activités. L'interrogation de ces données nécessite l'utilisation des requêtes complexes, coûteuses en temps de réponse et en ressources informatiques. Un élément clé dans l'architecture d'un système décisionnel est l'entrepôt de données.

L'entrepôt de données possède une structure scalable permettant l'ajout continuel de nouvelles sources de données, ce qui provoque l'augmentation continuelle du volume de données. Dans un tel système, l'entrepôt doit répondre efficacement aux requêtes, et il doit être capable d'offrir les meilleures décisions dans un laps de temps très cour. Cependant, le processus d'analyse se base sur des requêtes complexes intégrant de multiples jointures et d'agrégation. L'objectif et de pouvoir exécuter ces requêtes complexes afin de prendre des décisions et restituer les résultats aux décideurs dans des délais raisonnables afin de faire face au changement du marché.

Dans le but de supporter efficacement ces requêtes, l'utilisation des structures d'optimisation des requêtes s'avère indispensable afin d'améliorer les performances de l'entrepôt de données. Plusieurs techniques ont été proposées dans la littérature à savoir les vues matérialisées, les techniques d'indexation et la fragmentation. Notre travail concernera le problème de sélection des vues matérialisée et des index en se basant sur utilisation des colonies de fourmis. L'utilisation des fourmis artificielles dans la résolution des problèmes d'optimisation combinatoire NP-Complets est une activité très prometteuse. Cette approche tire sa capacité par le transfert d'apprentissage au sein de la colonie d'une manière stigmergique qui utilise l'environnement pour communiquer le choix de bonnes solutions en se basant sur la visibilité et le dépôt de phéromone. L'émergence vers les solutions optimales, en un temps fini, a fait de cette méthode approchée une possibilité de recours convenable.

II. Objectif

Nous voulons réaliser à travers ce travail un système de colonies de fourmis pour la sélection des vues matérialisées et des index binaires de jointures dans un entrepôt de données sous la contrainte d'espace de stockage.

Alors nos objectifs sont :

- Etudier les différents techniques d'optimisation des performances de l'entrepôt de données.
- Mettre l'accent sur les méta-heuristique en particulier les colonies de fourmis.
- Proposer deux nouvelles démarches pour la sélection des vues matérialisées et la sélection des index binaire de jointure.
- Distribuer l'espace disque de l'entrepôt de données entre ces deux structures d'optimisation de façon à ameliorer le coût d'exécution des requêtes décisionnelles.

III. Organisation du mémoire

Notre mémoire est organisé autour de deux parties comportant chacune trois chapitres. La première partie se propose de présenter : (1) un état de l'art des systèmes décisionnels et plus particulièrement les entrepôts de données, (2) un état de l'art des méta-heuristiques et plus particulièrement les colonies de fourmis, (3) Une études détaillée des différentes techniques d'optimisation des performances de l'entrepôt de données. La deuxième partie concerne la présentation de nos démarches proposées pour la sélection des vues matérialisées et des index en utilisant les colonies de fourmis.

Chapitre 1: Dans ce chapitre nous présenterons l'état de l'art des systèmes décisionnels. Nous parlerons, des processus OLTP et OLAP, de l'entrepôt de données, de la modélisation multidimensionnelle et enfin du cube de données.

Chapitre 2: Dans ce chapitre, nous présentons le problème de sélection des techniques d'optimisation dans les bases de données en général et les entrepôts de données en particulier. Nous présentons un état de l'art sur les travaux consacrés à la sélection des vues matérialisées et les index.

Chapitre 3: Dans ce chapitre, nous présenterons un état de l'art sur les méta-heuristiques, plus particulièrement les colonies de fourmis. C'est cette méta-heuristique que nous avons utilisé dans le cadre de notre travail pour prendre en charge le problème de sélection des vues matérialisées et des index.

Chapitre 4 : Ce chapitre présente notre approche proposée pour la sélection des vues matérialisées en utilisant les colonies de fourmis.

Chapitre 5 : Ce chapitre présente notre approche proposée pour la sélection des index bitmap de jointures en utilisant les colonies de fourmis. En faite, l'idée et d'adapter le problème de la recherche d'information en utilisant les colonies de fourmis au problème de sélection des index binaire de jointures dans un entrepôt de données.

Chapitre 6 : Ce chapitre est consacré à l'étude expérimentale des deux démarches adoptées.

Enfin le **chapitre 7** conclut ce travail en récapitulant les résultats principaux. Cette conclusion nous a permis aussi de lister quelques perspectives de prolongement de nos travaux.

Partie 1 : Etat de l’art

Chapitre 1 : Informatique de décision et l'entrepôt de données

1. Introduction

Dans l'informatique décisionnelle, les entreprises exploitent de grands volumes de données issues de différentes sources hétérogènes. Les structures qui accueillent ce flot important de données sont des entrepôts de données ou *data warehouse*. Ils sont construits sur une nouvelle architecture permettant d'extraire l'information, une architecture bien différente de celle prévue pour l'informatique de production. L'entrepôt de données permet de prendre des décisions plus éclairées plus rapidement et de donner aux décideurs un accès simple et rapide à leurs données.

Ce chapitre présente les principales notions relatives aux systèmes décisionnels, en particulier les entrepôts de données. Dans la deuxième section nous présentons la définition, l'architecture, les données d'un système décisionnel. Dans la troisième section nous introduisons les axes de recherche principaux dans le domaine des entrepôts de données, nous commençons par donner sa définition, l'architecture, les spécificités et les objectifs d'un entrepôt de données. La quatrième section mis l'axe sur la différence entre les systèmes OLTP et les systèmes OLAP. Nous décrivons par la suite la modélisation multidimensionnelle utilisée pour la conception des entrepôts de données ainsi que les différents modèles de données associés aux entrepôts.

2. Les systèmes décisionnels

2.1 Définition

Un système décisionnel est un ensemble de solutions informatiques permettant l'analyse des données de l'entreprise, afin d'en dégager des informations qualitatives nouvelles, de déceler des informations macroscopiques cachées dans de gros volumes de données. Il regroupe un ensemble d'informations et d'outils mis à la disposition des décideurs pour supporter de manière efficace la prise de décision.

Les systèmes décisionnels sont dédiés au management de l'entreprise pour aider au pilotage de l'activité, et sont indirectement opérationnels car ils n'offrent que rarement le moyen d'appliquer les décisions. Ils constituent une synthèse d'informations opérationnelles, et sont internes ou externes, choisies pour leur pertinence et leur transversalité fonctionnelles, et sont basés sur des structures particulières de stockage volumineux à savoir l'entrepôt de données.

Le principal intérêt d'un système décisionnel est d'offrir au décideur une version transversale et l'entreprise intégrant ses dimensions. Ces systèmes mettent en jeux quatre éléments essentiels :

- ***Les sources de données :*** elles sont nombreuses, variées, distribuées et autonomes. Elles peuvent être internes (base de production) ou externe (base de partenaires) à l'entreprise.

- ***L'entrepôt de données :*** c'est le lieu de stockage centralisé des informations utiles pour les décideurs.

- ***Les magasins de données :*** ce sont des extraits de l'entrepôt orientés sujet. Les données sont organisées de maniére adquate pour permettre des analyses rapides à des fins de prise de decisions.

- ***Les outils d'analyse :*** ils permettent de manipuler les données suivant des axes d'analyses. L'information est visualisée à travers des interfaces interactives et fonctionnelles dédiés à des décideurs souvent non informaticiens(directeur,chef de services,…)

Les outils d'aide à la décision ont vu leurs utilisation de plus grande étant donnée qu'ils permettent à l'entreprise d'être plus réactive face aux diverses fluctuations des marchés et aux défis des concurrents.

2.2 L'architecture des systèmes décisionnels

Les architectures des systèmes décisionnels sont considérées comme des architectures à trois niveaux : l'entrepôt de données constitue le premier niveau, le service du deuxième niveau est instauré par le serveur OLAP et les clients sont mis en œuvre au dernier niveau, comme illustré par La figure 1-1.

La conception d'un entrepôt repose sur une étude fine des besoins des décideurs afin de ne stocker que l'information utile mais aussi pour tenir compte des volumes importants de données engendrés par les mécanismes d'historisation. En effet, le stockage de l'évolution des données constitue une contribution majeure des entrepôts à la prise de décision, face à des bases de production qui, de part leur finalité, stockent rarement les historiques. Une fois extraites les données de l'entrepôt doivent être mises à jour périodiquement afin de préserver leur intérêt pour la prise de décision.

Un magasin est un entrepôt thématique (orientée sujet), principalement dédie à une classe de décideurs. L'objectif est d'adapter au mieux les structures de données à l'utilisation qui en sera faite. Ainsi les analyses de données selon différents critères (temps, lieu, responsable, ...) sont facilitées lorsque ces données sont organisées selon un modèle multidimensionnel.

2.2.1 Processus de construction

La mise en place d'un système décisionnel est une tâche complexe qui recouvre de nombreuses difficultés. L'intégration se propose de résoudre les problèmes d'hétérogénéité [KED99] des différentes sources de données en intégrant celles-ci dans une source globale. Cette source globale est virtuelle. Cela signifie que les données utilisées pour la décision restent stockées dans les sources de données et sont extraites uniquement au moment des mises à jour de l'entrepôt.

La construction consiste à extraire les données pertinentes pour la prise de décision, puis à les recopier dans l'entrepôt de données, tout en conservant, le cas échéant, les changements d'états des données [INM94, PED99, YAN98, YAN00, MEN00]. La réorganisation permet de restructurer les données dans des magasins de données; la réorganisation des données vise à supporter efficacement les processus d'interrogation et d'analyse tels que les applications OLAP ("On-Line Analytical Processing") [E.C93] et la fouille de données (Data Mining) [FAY96, CRI99]. Pour ce faire, les données importées dans les magasins doivent être organisées dans un modèle facilitant la décision et adaptée aux outils d'analyse (base multidimensionnelle, tableaux de données,…etc.). L'interrogation permet de connaître, mesurer et prévoir (prise de décisions) au travers de la manipulation des données du magasin. Les manipulations peuvent recouvrir plusieurs aspects:

- Une simple consultation des données d'un tableau avec génération de graphiques associées,
- Requêtage graphique sur une base de données,
- Combinaison d'opérations multidimensionnelles sur une base de données adaptée.

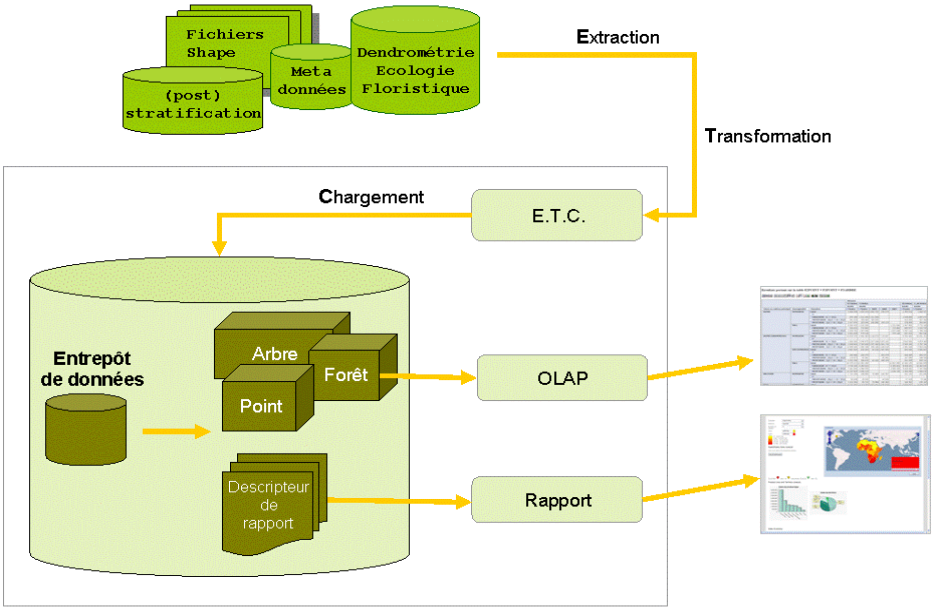

Figure 1-1 Architecture détaillée des systèmes d'aide à la décision

2.2.2 De l'extraction à l'intégration

L'intégration des données provenant de sources hétérogènes est à la base de la notion d'entrepôt de données. Diverses architectures réalisent la production de données intégrées et ont en commun les composantes suivantes :

- des outils d'extraction de données,
- des outils de nettoyage de données et
- des outils d'intégration de données.

2.2.3 L'intégration

Deux types d'intégration sont possibles: (1) l' intégration des schémas des sources et (2) l'intégration des données.

L'intégration des schémas des sources fait apparaître des conflits, depuis longtemps bien répertoriées dans la littérature. Les principaux conflits pouvant survenir entre deux schémas sont les suivants: (1) problèmes de terminologie, (2) incompatibilités de contraintes, (3) conflits de structures et (4) conflits de représentation.

(1) Un conflits de terminologie survient lorsqu'un même objet du réel est désigné par des noms différents ou au contraire lorsqu'un même nom est utilisé pour deux objets différents. Ces cas peuvent correspondre à des problèmes de synonymie ou d'homonymie, mais sont le plus souvent dus à une différence de niveau de généricité (ex: "personne" et "étudiant"), ou à des converses (ex: "vente" et "achat").

(2) Un conflits de contraintes apparaît lorsque sur deux concepts établis comme équivalents ont des contraintes incompatibles.

(3) Les conflits de structures sont caractérisés par un choix différent de propriétés à stocker pour un même concept du réel. Par exemple, on peut définir une personne dans une vue par son numéro, son nom et son âge, et dans une autre vue par son nom, son prénom et son adresse.

(4) On détecte un conflit de représentation lorsque deux représentations différentes sont choisies pour les mêmes propriétés d'un même concept. Par exemple la date de commande peut être inclue dans la commande ou former un objet relié à la commande.

Deux principales approches permettent un accès unique à des sources de données hétérogènes: une approche virtuelle (souvent appelée approche par médiateur) et une approche matérialisée (approche par entrepôt). Les approches virtuelles sont basées sur une hiérarchie de médiateurs, correspondant à des vues virtuelles, au-dessus des extracteurs. Les données ne sont stockées que dans leur source d'origine. Dans l'approche matérialisée, les données sont effectivement extraites, nettoyées, intégrées et stockées dans un entrepôt. Les requêtes sont posées directement sur les données de l'entrepôt. Les métadonnées décrivant le schéma globale de l'entrepôt doivent inclure le plus d'explications possible sur l'origine, la signification des données, elles jouent le rôle d'une carte routière pour s'orienter et poser des requêtes dans l'entrepôt. Un des problèmes majeurs à résoudre dans cette approche est celui de la répercussion dans l'entrepôt des mises à jour effectuées sur les sources. Dans une architecture d'entrepôt, on distingue la matérialisation initiale d'une vue, et la maintenance de la vue. Pour le peuplement initial de la vue, les requêtes sont généralement écrites par l'administrateur.

2.3 Les données opérationnelles et les données décisionnelles

Le tableau suivant montre les principales différences entre les données du système opérationnels et les données du système décisionnel.

Données opérationnelles	Données décisionnelles
Orientées application, détaillées, précise au moment de l'accès	Orienté activité (thème, sujet), condensées, représentent des données théoriques.
Mise à jour interactive possible de la part des utilisateurs	Pas de mise à jour interactive de la part des utilisateurs
Accéder de façon unitaire par une personne a la fois	Utilisées par l'ensemble des analystes, gérées par sous ensemble.
Cohérence atomique	Cohérence globale
Haute disponibilité en continue	Exigence différente, haute disponibilité ponctuelle
Unique (pas de redondance en théorie)	Peuvent être redondante
Structure statique, contenu variable	Structure flexible
Petite quantité de données utilisées par un traitement	Grande quantité de données utilisées par le traitement.
Réalisation des opérations au jour le jour	Cycle de vie différent.
Forte probabilité d'accès	Faible probabilité d'accès
Utilisées de façon répétitive.	Utilisées de façon aléatoire.

Tab1.1 : Différences entre les données du système de production et les données décisionnelles.

3. L'entrepôt de données

3.1 Définition

La définition classique de l'entrepôt de données est donnée par **Bill Inmon** dans son livre « **Building data warehouse** » **:** L'entrepôt de données est une collection de données, orientée sujet, intégrée, non volatile et historiée et orientée pour le processus de décision. Ces principales caractéristiques sont donc les suivants :

- ***Orientées sujet :*** les données de l'entrepôt s'organisent par sujet ou thème, ce qui permet de regrouper toutes les données pertinentes et nécessaires pour toutes analyses relatives à ce théme.

- ***Intégrée :*** les données d'un entrepôt de données proviennent de plusieurs sources hétérogènes. L'intégration consiste à résoudre le problème d'hétérogénéité des modèles, des schémas et de la sémantique.

- ***Non volatile :*** la non volatilité des donnée est une conséquence de l'historisation. En faite, une donnée entrée dans l'entrepôt de données n'a pas une vocation à être supprimée. Donc, une requête lancée à différentes dates sur les mêmes données doit toujours retourner les mêmes résultats.

- ***Historiée :*** les données d'un entrepôt de données doivent être datées et non remplacées par des mises à jour, afin de suivre le temps d'évolution de différentes valeurs des indicateurs à analyser.

3.2 Les objectifs du data warehouse

L'objectif d'un entrepôt de données, est de servir d'intermédiaire en stockant différentes données issues des applications de production en vue d'être sondé afin de recueillir les informations nécessaires à la prise de décision.

- **Accès aux informations de l'entreprise :** L'entrepôt de données assure l'accès aux informations de l'entreprise et de l'organisation.

- **Les informations d'un entrepôt de données sont cohérentes :** cela signifie que les requêtes faites à des moments différents doivent fournir les mêmes résultats. La cohérence veut aussi dire que lorsque les personnes demandent la définition de l'élément « contrat », elles obtiennent une réponse leur permettant de savoir ce qu'elles obtiendront de la base de données. La cohérence implique que les données sont chargées dans leur totalité. Les données d'un entrepôt doivent pouvoir être séparées et combinées au moyen de toutes les mesures possibles de l'activité.

- **Les outils de présentation d'informations font partie de l'entrepôt :** L'entrepôt de données ne comporte pas seulement des données, mais aussi un ensemble de requêtes, d'analyses et de présentations des informations.

- **Les données publiées sont stockées dans l'entrepôt :** L'entrepôt de données est le lieu où sont publiées des données qui ont déjà servi. Les données sont soigneusement rassemblées à partir de sources d'informations situées à différents endroits de l'organisation. Elles sont nettoyées, leur qualité est vérifiée et elles ne sont diffusées que

si elles sont prêtes à être utilisées. Si l'information est peu fiable ou incomplète, les données ne peuvent être publiées à destination de la communauté des utilisateurs.

- **Qualité de l'information d'un entrepôt de données :** La qualité de l'information d'un entrepôt de données est très importante. En effet, comment obtenir des analyses fiables si les données brutes sont de mauvaise qualité ? L'entrepôt de données ne peut remédier à la mauvaise qualité des données ou à l'absence d'une donnée. La seule façon de remédier à la médiocre qualité des données consiste, pour les personnes concernées par la saisie des données et pour le management, à retrouver la source des informations et à mettre en place de meilleurs systèmes ou à mieux faire comprendre l'importance de la qualité des données.

3.3 Les composants d'un entrepôt de données

La figure 1-2 illustre l'architecture d'un système datawarehouse. Les composants principaux d'un entrepôt de données sont [KIM01] [BOL02] :

- **Les systèmes sources :** Système opérationnel d'enregistrement. Il permet de capturer les transactions liées à l'activité de l'entreprise. Il s'agit souvent de ce que l'on appelle les applications de gestion de l'entreprise. La principale priorité du système source est le temps de disponibilité.

- **La zone de préparation des données :** Cette zone comprend tout ce qui se trouve entre les systèmes sources et le serveur de présentation. Elle regroupe l'ensemble des processus qui nettoient, transforment, combinent, archivent, suppriment les doublons. Elle prépare les données sources en vue de leur intégration et de leur exploitation dans le data warehouse. La frontière qui détermine la zone de préparation des données est que la zone de préparation des données ne doit en aucun cas être accessible à l'utilisateur final par requêtes ou par un quelconque autre service de présentation.

- **Le serveur de présentation :** Ce composant correspond à la machine cible sur laquelle l'entrepôt de données est stocké et organisé pour répondre en accès direct aux requêtes provenant des utilisateurs, des générateurs d'états ou d'autres applications. Sur le serveur de présentation, les données seront stockées et présentées sous une forme dimensionnelle, de façon à faciliter l'accès pour l'utilisateur final. Dans la majorité des cas, le serveur est basé sur une base de données relationnelle, de sorte que les tables y sont organisées sous forme de schémas en étoile.

- **Le data mart :** Le data mart est défini comme un sous-ensemble logique d'un entrepôt de données. Il représente un projet réalisable. Le data mart comme la réduction de l'entrepôt de données à un seul processus ou à un groupe de processus ciblant un groupe métier spécifique. Les data marts sont basés sur des données détaillées et peuvent contenir des agrégats de données destinés à optimiser les performances.

- **L'entrepôt de données ou data warehouse :** L'entrepôt de données correspond à la source de données interrogeable de l'entreprise. Selon la définition du data mart, on peut voir l'entrepôt comme l'union de tous les data marts qui le composent. L'entrepôt de données est alimenté par la zone de préparation des données.

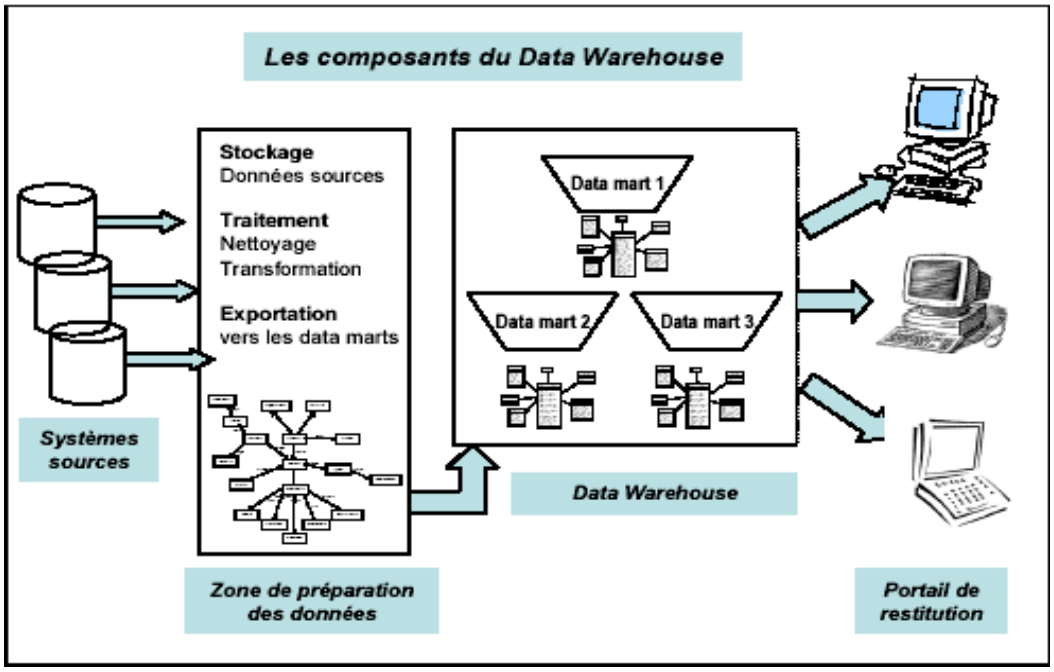

Figure 1-2 Les composants du data warehouse.

- **Portail de restitution :** permet de représenter ce que voient les utilisateurs, les outils avec lesquels ils travaillent. Sous ce terme sont regroupées toutes les applications qui s'appuient sur les données du data warehouse pour les restituer soit à l'utilisateur, soit à une autre application. Les services offerts par le portail de restitution sont les services d'accès aux données, les applications de modélisation et de data mining. Les services d'accès aux données comprennent : la navigation dans l'entrepôt, dans les métadonnées, la surveillance de l'activité, la gestion des requêtes et la génération d'états standards. Les applications de modélisation offrent différents types d'analyses basées sur des modèles tels que modèles financiers, systèmes d'évaluation des clients, optimisation des processus et prévisions, ainsi que les activités centrales du datamining telles que la catégorisation, la classification, l'estimation et prédiction et finalement le regroupement par affinité.

- **Métadonnées :** Ce sont toutes les informations de l'environnement du data warehouse qui ne constituent pas les données proprement dites. Ce sont les « données sur les données ».

3.4 La structure d'un entrepôt de données

L'entrepôt de données comporte quatre classes de données, organisées selon un axe historique et un axe de synthèse : des données détaillées, agrégées, historiées et les métadonnées [FRA01]. La figure 1-3 illustre les différentes classes de données d'un entrepôt de données.

1. **Les données détaillées:** Elles reflètent les événements les plus récents. Les intégrations régulières des données issues des systèmes de production vont habituellement être réalisées à ce niveau.

2. **Les données agrégées:** Elles correspondent à des éléments d'analyse représentatifs des besoins utilisateurs. Elles constituent déjà un résultat d'analyse et une synthèse de l'information contenue dans le système décisionnel, et doivent être facilement accessibles et compréhensibles. Cette couche correspond donc à des analyses «à priori ».

3. **Les métadonnées:** les métadonnées constituent l'ensemble des données qui décrivent des règles ou processus attachés à d'autres données. Ces dernières constituent la finalité du système d'information. Les métadonnées sont intégrés dans un référentiel et contiennent des informations sur la sémantique des données, leur localisation, ainsi que des règles de descriptions et finalement la structure de la base qui implémente l'entrepôt de données.

4. **Les données historiées:** Chaque nouvelle insertion de données provenant du système de production ne détruit pas les anciennes valeurs, mais créée une nouvelle occurrence de la donnée. Cependant, il n'est pas toujours nécessaire de conserver à un niveau détaillé les données les plus anciennes.

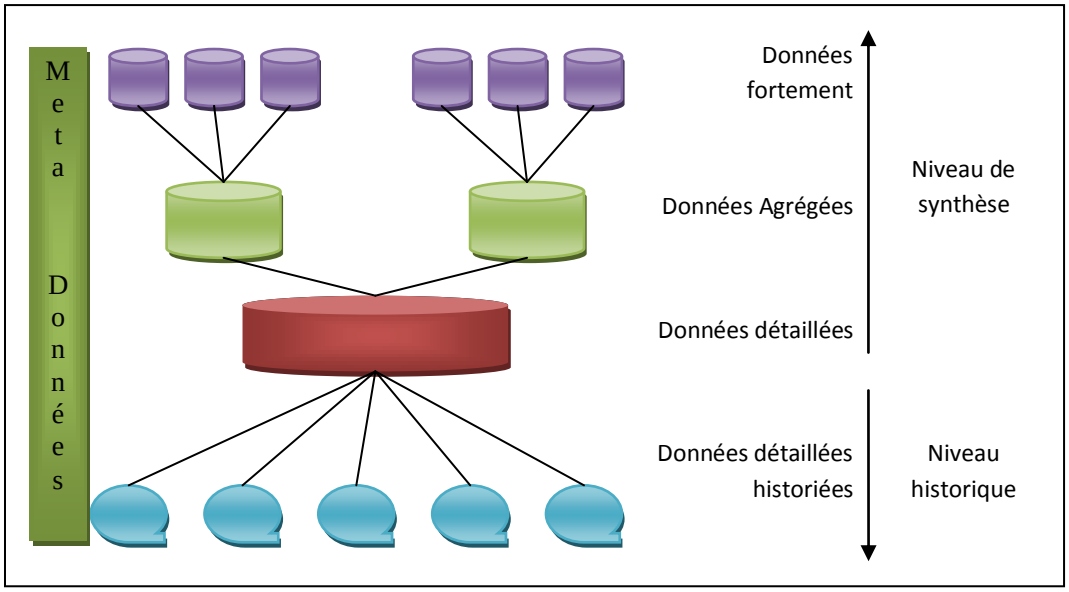

Figure 1-3 Les différentes classes de données d'un entrepôt de données.

3.5 L'architecture d'un entrepôt de données

Un entrepôt de données stocke, sous une forme intégrée, des données extraites d'un ensemble de sources de données. Les données sont extraites des sources de données, transformées dans un format commun et combiné avec les données existant dans l'entrepôt de données. L'architecture du data Warehouse comporte trois niveaux essentiels [BON99]:

- Le niveau **extraction de données** des bases (de données) existantes est réalisé par un outil d'ETL qui détecte les mises à jour sur les bases opérationnelles ou les fichiers d'entreprise afin de les envoyer dans l'entrepôt.

- Le niveau **collecte de données** intègres les mises à jour en provenance des différentes bases concernées de l'entreprise et les stocke dans la base de données de l'entrepôt en respectant son organisation par sujet.

- Le niveau **analyse de données** permet la formulation de requêtes afin de cibler sur les faits étudiés, l'analyse des tendances des données, l'aide à la prise de décision, la découverte de connaissances ; ce niveau est mis en oeuvre à partir d'outils réalisant des extractions par requêtes et des présentations graphiques variées.

Afin d'implémenter un entrepôt de données, trois type d'architecture sont possibles [CNA98]:

- **Architecture réelle :** elle est généralement retenue pour les systèmes décisionnels. Le stockage des données est réalisé dans un SGBD séparé du système de production. Le SGBD est alimenté par des extractions périodiques. Avant le chargement, les données subissent d'importants processus d'intégration, de nettoyage, de transformation. L'avantage est de disposer de données préparées, pour les besoins de décision et répondant aux objectifs du Data Warehouse. Les inconvénients sont le coût de stockage supplémentaire et le manque d'accès en temps réel.

- **Architecture Virtuelle :** cette architecture n'est pratiquement pas utilisée pour l'entrepôt de données. Les données résident dans le système de production. Elles sont rendues visibles par des produits Middleware ou par des passerelles. Il en résulte deux avantages : pas de coût de stockage supplémentaire et l'accès se fait en temps réel. L'inconvénient est que les données ne sont pas préparées.

- **Architecture Remote :** c'est une combinaison de l'architecture réelle et de l'architecture virtuelle. Elle est rarement utilisée. L'objectif est d'implémenter physiquement les niveaux agrégés afin d'en faciliter l'accès, et de garder le niveau de détail dans le système de production en y donnant l'accès par le biais de Middleware ou la passerelle.

4. Systèmes OLTP vers les systèmes OLAP

Les bases de données sont utilisées dans les entreprises pour gérer les importants volumes d'informations contenus dans leurs systèmes opérationnels. Ces données sont gérées selon des processus transactionnels en ligne OLTP : On-Line Transactional Processing qui se caractérise de la manière suivante [COD93] [INM94] [KIM96] [CHA97] :

- ils sont nombreux au sein d'une entreprise,
- ils concernent essentiellement la mise à jour des données,
- ils traitent un nombre d'enregistrements réduit,
- ils sont définis et exécutés par de nombreux utilisateurs.

L'exploitation de l'information contenue dans ces systèmes opérationnels est devenue une préoccupation essentielle pour les dirigeants des entreprises qui désirent améliorer leur prise de décision. Les entreprises sont donc à la recherche de systèmes supportant efficacement les applications d'aide à la décision. Ces applications décisionnelles utilisent des processus d'analyse en ligne de données OLAP : **O**n-**L**ine **A**nalytical **P**rocessing [COD93] qui répondent aux besoins spécifiques des analyses d'information. Les processus OLAP se caractérisent par : [COD93] [INM94] [KIM96] [CHA97] :

- ils sont peu nombreux, mais leurs données et traitements sont complexes,
- il s'agit uniquement de traitements semi-automatiques visant à interroger, visualiser et synthétiser les données,
- ils concernent un nombre d'enregistrements importants aux structures hétérogènes,
- ils sont définis et mis en oeuvre par un nombre réduit d'utilisateurs qui sont les décideurs.

Le Tableau suivant compare les caractéristiques des processus OLTP et OLAP.

	Processus OLTP	**Processus OLAP**
Données	Exhaustives Courantes Dynamiques Orientées application	Résumées Historiques Statiques Orientées sujets (d'analyse)
Utilisateurs	Nombreux Variées (employés, directeur …) Concurrents Mise à jour et interrogations Requêtes prédéfinies Réponses immédiates Accès a peu d'information	Peu nombreux Uniquement les décideurs Non concurrents Interrogation Requêtes imprévisible et complexes Réponses moins rapides Accès à nombreuses informations

Tableau 1.2 : les caractéristiques des processus OLTP et OLAP.

5. Modélisation multidimensionelles

La modélisation dimensionnelle est une méthode de conception logique souvent associée aux entrepôts de données. Elle vise à présenter les données sous une forme standardisée intuitive et qui permet des accès performants [BOL02]. Elle consiste à considérer un sujet analysé comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse.

Dans cette approche, les données sont modélisées selon plusieurs axes d'analyses, pouvant représenter des notions variées, tels que le temps, la localisation géographique, le code identifiant les produits....etc. Ce format multidimensionnel est connu sous le nom d'hypercube**.**

Ce modèle présente l'avantage d'être facilement compris par un utilisateur final. Il lui suffit de choisir le sujet étudié puis de sélectionner en appliquant des critères sur les tables « dimension». Chaque modèle dimensionnel se compose d'une table contenant une clé multiple, ***la table de Faits***, et d'un ensemble de tables plus petites nommées ***tables de dimension***. Chaque table « dimension » possède une clé primaire unique qui correspond exactement à l'un des composants de la clé multiple de la table de faits [BOL02].

5.1 Le modèle ROLAP

Conceptuellement, cette modélisation multidimensionnelle a donné naissance aux concepts de fait et de dimension [KIM96].

5.1.1 Définition du fait

Le fait représente le sujet à analyser, il est formé de mesures correspondantes aux informations de l'activité analysée [TES00]. Par exemple pour la gestion des commandes, les mesures peuvent être la *quantité* du produit commandé et le *montant* de la commande.

5.1.2 Définition de la dimension

Une dimension modélise une perspective de l'analyse. Elle se compose de paramètres correspondants aux informations faisant varier les mesures de l'activité. Elle sert à enregistrer les valeurs pour lesquelles sont analysées les mesures de l'activité (fait). Les paramètres sont organisés de manière hiérarchique en partant de faible niveau vers un niveau plus détaillé, ces paramètres permettent de restreindre la portée des requêtes afin de limiter la taille des réponses [TES00]. Par exemple la gestion des commandes peut être analysée selon les dimensions *Client*, *Magasin*, et *Temps*, et pour la dimension *temps*, nous pouvons avoir la hiérarchie suivante : *année*, *semestre*, *trimestre*, *mois*, *semaine* et *jour*. Les dimensions sont caractérisées par des attributs de dimensions.

5.1.3 Modèles en étoile, en flocon et en constellation

A partir du fait et des dimensions, il est possible d'établir une structure de données simple qui correspond au besoin de la modélisation multidimensionnelle.

5.1.3.1 Le modèle en étoile

Cette structure est constituée du fait central et des dimensions. Ce schéma est le plus répondu dans les systèmes d'implémentation des entrepôts de données. C'est un schéma relationnel où la table des faits est normalisée, mais les tables de dimension ne sont pas normalisées [EM 05]. La figure (1-4) montre un schéma ROLAP en étoile avec Vente comme table de fait et Temps, Catégorie et Géographie comme dimensions.

Cette représentation facilite l'analyse selon plusieurs perspectives. Les requêtes typiques de ce schéma sont appelées requêtes de jointure en étoile [BEL00] et ont les propriétés suivantes:

- Il y a des jointures multiples entre la table des faits et les tables de dimension.
- Il n'y a pas de jointure entre les tables de dimensions.
- L'opération de sélection est souvent effectuée sur les tables de dimensions.

Cependant l'inconvénient de ce type de représentation est : d'un part la non-normalisation des données. En conséquence, il existe une redondance des données dans les tables de dimensions. Et d'autre part, ce schéma ne reflète pas de manière explicite les hiérarchies des dimensions [GE02].

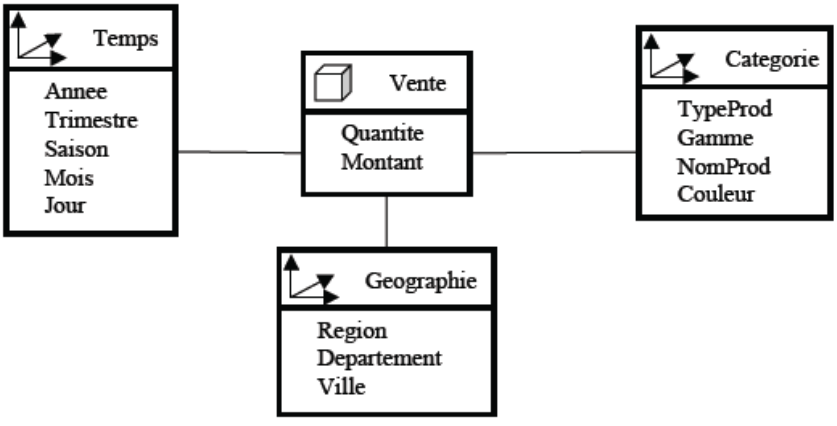

Figure 1-4 Exemple d'une modalisation en étoile.

5.1.3.2 Modèlisation en flocon

Une modélisation en flocon consiste à décomposer les dimensions du modèle en étoile en sous hiérarchies. Cette modélisation est donc une émanation de la modélisation en étoile; le fait est conservé et les dimensions sont éclatées conformément à sa hiérarchie des paramètres.

La Figure (1-5) illustre la modélisation en flocon ; nous décrivons le modèle en étoile de la Figure (1-4) en dénormalisant chacune de ces dimensions, formant ainsi une sorte de flocon.

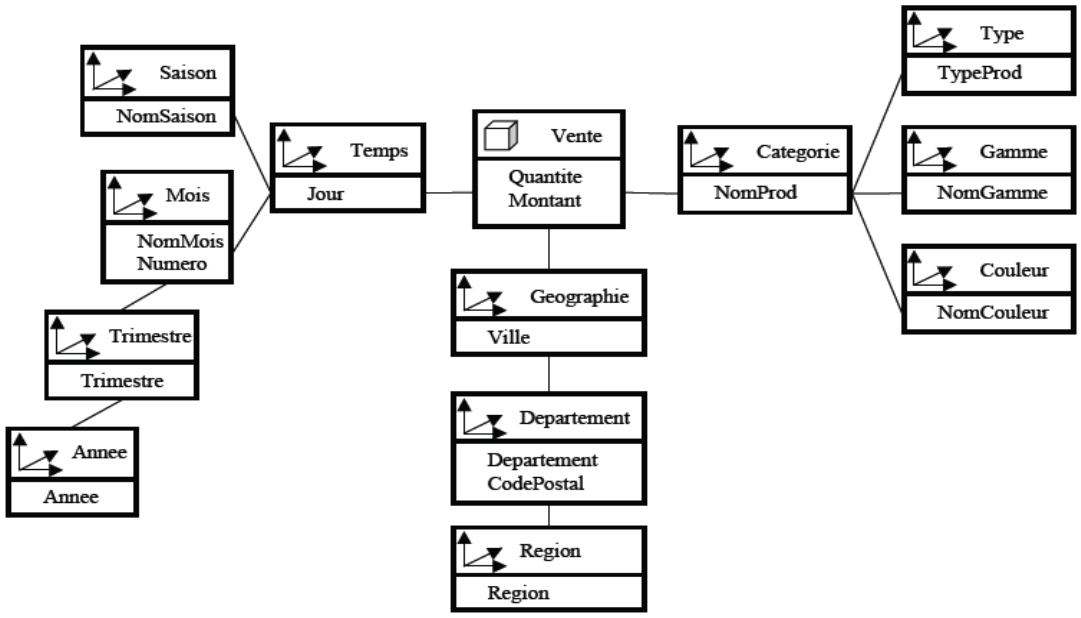

Figure1- 5 Exemple d'une modélisation en flocon.

Cette représentation réduit la redondance des données ce qui offre une meilleure visualisation et compréhension des données. Par contre, elle augmente le temps d'exécution des requêtes qui nécessite plus de jointures.

5.1.3.3 Modèlisation en constellation

Il s'agit de fusionner plusieurs modèles en étoile qui utilisent des dimensions communes. Cette représentation comprend donc plusieurs faits et des dimensions communes ou non. Elle est employée lorsque les faits analysés ne sont pas tous déterminé par les mêmes dimensions. Notons que les tables de dimensions partagées par plusieurs tables de fait doivent être exactement les mêmes.

La Figure (1-6) illustre la modélisation en constellation ; nous décrivons une constellation constituée de deux schémas en étoile : l'un correspond aux ventes effectuées dans les pharmacies et l'autre analyse les prescriptions des médecins.

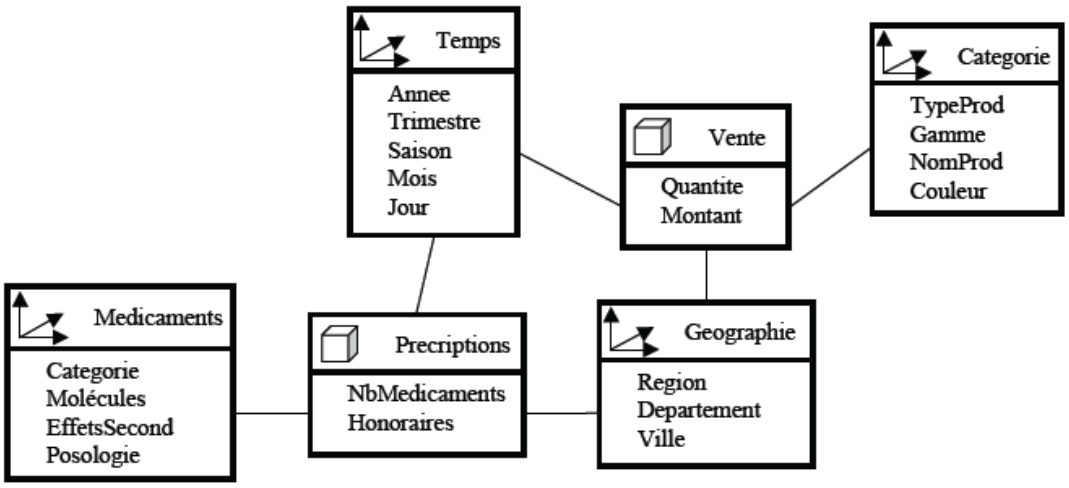

Figure 1-6 Exemple d'une modélisation en constellation.

5.2 Le modèle MOLAP

Les systèmes de type MOLAP stockent des données dans un SGBD multidimensionnel sous la forme d'un tableau multidimensionnel. Chaque dimension de ce tableau est associée à une dimension du cube (cube de données). Seules les valeurs de données correspondant aux données de chaque cellule sont stockées. Ces systèmes demandent un pré-calcul de toutes les agrégations possibles. En conséquence, ils sont plus performants que les systèmes traditionnels, mais plus difficiles à mettre à jour et à gérer. Les systèmes MOLAP apparaissent comme une solution acceptable pour le stockage et l'analyse d'un entrepôt de données lorsque la quantité estimée des données d'un entrepôt ne dépasse pas quelque Giga octets et lorsque le modèle multidimensionnel évolue peu. Mais lorsque les données sont éparses, ces systèmes sont consommateurs d'espace et des techniques de compression doivent être utilisées.

L'avantage principal de cette technique est le gain considérable en temps d'exécution des requêtes, vu que l'accès aux données est direct. Par contre, elle présente les inconvenantes suivantes [BKS00]:

- Redéfinir les opérations SQL pour manipuler des structures multidimensionnelles.
- La difficulté de la mise à jour et de la gestion du modèle,
- La consommation de l'espace lorsque les données sont creuses, ce qui nécessite l'utilisation des techniques de compression.

5.3 Comparaison entre MOLAP et ROLAP

A partir de définition de ses deux modèles on peut tirer déjà quelques différents de forme de stockage des données. Dans le cas de ROLAP, il s'agit plus d'une simulation du comportement d'un SGBD multidimensionnel faite à partir d'un SGBD relationnel alors que dans MOLAP les données sont vraiment stockées dans une structure multidimensionnelle (structure en cube) , et l'accès se fait directement dans ses cubes.

La conception du schéma de données pour ROLAP est particulière : schéma en étoile, en flocon de neige ou en constellation. Des vues matérialisées sont utilisées pour la représentation multidimensionnelle. Dans ce cas les requêtes OLAP sont traduites en SQL et

les index sont utilisés pour faciliter l'accès aux différentes vues. Parmi les points forts de ROLAP, on note sa souplesse, et son évolution facile en plus du faite qu'il permet de stocker de gros volumes de données. A l'inverse, un de ses points faibles est qu'il soit peu efficace pour les calcules complexes, notamment pour les requêtes qui demandent un nombre important de jointure d'agrégation, qui nécessite l'accès simultané à plusieurs tables de dimensions et de faits.

Dans MOLAP, les cubes de données sont directement modélisés, et ils sont implémentés comme des matrices à plusieurs dimensions, et ils sont indexés sur leurs dimensions. Cette conception architecturale complexe permet à MOLAP d'être très rapide, vu qu'il n'a pas besoin de consulter des tables comme dans le cas de ROLAP pour fournir des résultats. Mais cette rapidité a un prix ; le modèle MOLAP ne supporte pas de très gros volumes de données er surtout il n'est pas facile à mettre en œuvre [TES 00].

6. Synthèse

Nous avons présenté dans ce chapitre les aspects de datawarehousing qui est un volet important de l'informatique de décision, si bien qu'un système d'aide à la décision dépourvu d'un entrepôt de données.

L'entrepôt de données doit collecter l'ensemble de l'information utile aux décideurs à partir des sources de données. L'objectif de l'entrepôt est de centraliser l'information décisionnelle en assurant l'intégration des données extraites, la pérennité de ces données dans le temps et la conservation de leurs évolutions.

Les entrepôts de données à proprement parlé sont inexploitable (du moins efficacement) juste après la fin de l'étape de l'alimentation. Vu le nombre faramineux d'enregistrements contenue dans cette structure, il est important de faire appel à des techniques spécifiques pour rendre l'exploitation de l'entrepôt la plus profitable possible. Ces structures d'optimisation appliquées aux données de l'entrepôt feront l'objet d'étude de notre prochain chapitre.

Nous avons défini le système décisionnel comme un système d'information dédié aux applications décisionnelles qui comprend essentiellement deux types de composants : un entrepôt de données et des magasins de données.

Les données stockées dans un entrepôt doivent correspondre à une structuration adaptée des données (selon plusieurs axes d'analyses) reflétant la vision des analystes. Cette représentation des données est basée sur une modélisation multidimensionnelle. L'approche multidimensionnelle vise à modéliser les données conformément à la perception des analystes: les données sont représentées suivant les différents axes d'analyses possibles. Le modèle multidimensionnel comprend un fait contenant les mesures à analyser et des dimensions contenant les paramètres de l'analyse. Dans chaque dimension, les paramètres sont hiérarchisés selon des niveaux de détail.

Dans le chapitre suivant, nous décrivons les différentes structures d'optimisation des performances de l'entrepôt de données.

Chapitre 2 : Optimisation des performances d'un entrepôt de données.

1. Introduction

Afin d'adopter une politique d'optimisation d'un entrepôt de données, il faut être en mesure de choisir la technique d'optimisation adéquate, la nature de sélection nécessaire et l'algorithme de sélection [BBC08], afin d'assurer une réduction de la complexité d'exécution des requêtes, du temps et du coût d'exécution.

Dans la littérature, il existe deux classes de structures d'optimisation [BKS00]: Les techniques redondantes dont l'utilisation produit une duplication des données : les index, les vues matérialisée et la fragmentation verticale. Les techniques non redondantes ne dupliquent pas les données. Parmi ces techniques on trouve la fragmentation horizontale et le data clustering qui permettent de réorganiser la représentation physique des données, le traitement parallèle qui permet d'exécuter les opérations en parallèle pour un gain de temps de traitement.

L'objectif de ce chapitre est tout d'abord de décrire les différentes structures d'optimisations existant dans la littérature. Ensuite de discuter l'existant en matière de techniques de sélection d'index et de vues matérialisées dans le contexte des entrepôts de données.

2. Classification des techniques d'optimisation

Plusieurs techniques d'optimisation ont été proposées dans la littérature. Nous pouvons les classer en deux catégories principales [BEL07, BEL08] : techniques redondantes et techniques non redondantes. Les techniques redondantes optimisent les requêtes, mais exigent un coût de stockage et de maintenance. Cette catégorie regroupe les vues matérialisées [GUP97], les index [HAR97] etc. Les techniques non redondantes ne nécessitent ni coût de stockage ni coût de maintenance. Cette catégorie regroupe la fragmentation horizontale [BEL06], le traitement parallèle [STO00], etc. La figure (2-1) montre une classification des principales techniques d'optimisation.

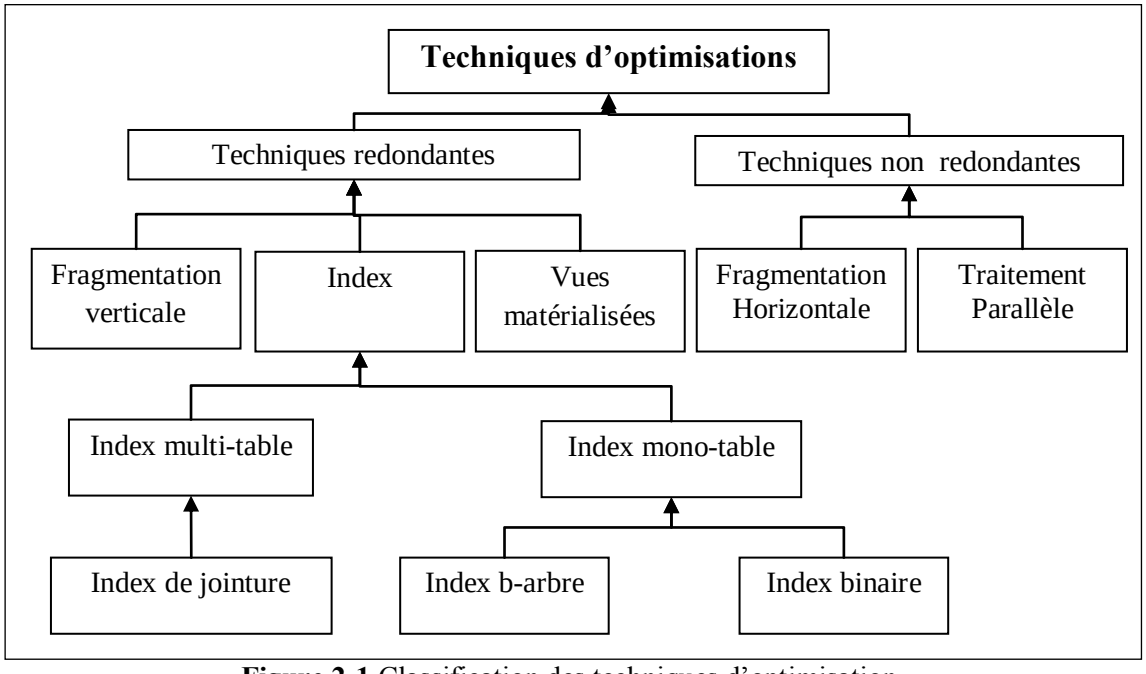

Figure 2-1 Classification des techniques d'optimisation.

3. Matérialisation des vues

3.1 Définition d'une vue

Une vue est une requête nommée [BKS00]. C'est une table contenant les résultats d'une requête dans le but d'améliorer l'exécution des requêtes en pré-calculant les opérations les plus coûteuses comme la jointure et l'agrégation, et en stockant leurs résultats dans la base de données. En conséquence, certaines requêtes nécessitent seulement l'accès aux vues matérialisées et s'exécutent ainsi plus rapidement.

3.2 L'utilisation des vues matérialisées

La matérialisation des vues permet de rendre disponible les résultats intermédiaires pouvant être utilisés par des requêtes complexes. Ceci réduit ainsi la complexité et le coût d'exécution de ces requêtes. Dans la pratique, la matérialisation des vues présente une large utilisation, en allant de l'informatique distribuée à l'informatique mobile. Les vues matérialisées dans l'informatique distribuées sont utilisées pour la duplication des données au niveau des différents sites distribués. Les réplicas permettent de résoudre les requêtes uniquement par l'accès locaux. En ce qui concerne l'informatique mobile, et vu l'essor considérable que cette dernière a pris au cours de la dernière décennie grâces aux performances croissant du matériels à des coûts de plus en plus faibles, des problèmes de déconnexion incohérents aux environnement sans fil sont apparus, c'est pour cette raison que ces machines sont souvent contraints à copier localement des données, et ceci en se servant de vues matérialisées.

3.3 Formalisation du problème PSV

Le problème de sélection de vues matérialisées consiste à construire une configuration de vues optimisant le coût d'exécution d'une charge donnée. Cette optimisation peut être réalisée sous certaines contraintes telles que l'espace de stockage alloué aux vues sélectionnées ou une borne supérieure du coût de maintenance des vues sélectionnées.

Formellement, étant donnée un ensemble de requêtes $\{Q_1, Q_2, \ldots, Q_n\}$, le PSV consiste à sélectionner un ensemble de vues $\{V_1, V_2, \ldots, V_k\}$ minimisant une fonction objective qui peut être le coût total de calcul, et/ou coût de maintenance des vues et satisfaisant une contrainte de ressources, la capacité de stockage par exemple. Dans la littérature nous distinguons deux catégories principales de PSV (Problème de Sélection des Vues), le PSV statique et le PSV dynamique.

- **Le PSV statique :** Consiste à sélectionner un ensemble de vues afin de minimiser le coût total d'évaluation des requêtes et/ou le coût de maintenance sous la contrainte de la ressource. Dans ce cas, l'ensemble des requêtes est connu a priori. Donc, la sélection se fait en supposant que l'ensemble de requêtes n'évolue pas d'où la nécessité d'une reconstitution totale en cas d'évolution.

- **Le PSV dynamique :** Dans ce cas, la matérialisation se fait d'une manière dynamique en procédant par rafraichissement, c'est-à-dire qu'on élimine les vues les moins utilisées par exemple en cas de dépassement d'espace autorisé.

Le problème de sélection de vues matérialisées est NP-complet [GUP99]. Pour le résoudre il faut utiliser des méthodes non exactes comme les heuristiques qui tentent d'élaborer une solution optimale ou quasi-optimale en réduisant la complexité de sélection. En effet, si *d* est

le nombre de dimensions dans le schéma d'un entrepôt de données, et qui ne contient aucune hiérarchie, alors le nombre de vues candidates à sélectionner est égal à 2^d [BEL00].

Le PSV a été largement étudié tant pour l'approche MOLAP (*Multidimentional On-Line Analytical Processing*) que pour l'approche ROLAP (*Relational On-Line Analytical Processing*). Dans le PSV de type MOLAP, le cube de données est considéré comme une structure primordiale pour la sélection des vues. Chaque cellule du cube est considérée comme une vue potentielle. Dans le PSV de type ROLAP, chaque requête est représentée par un arbre algébrique. Chaque cellule du nœud est considérée comme une vue potentielle.

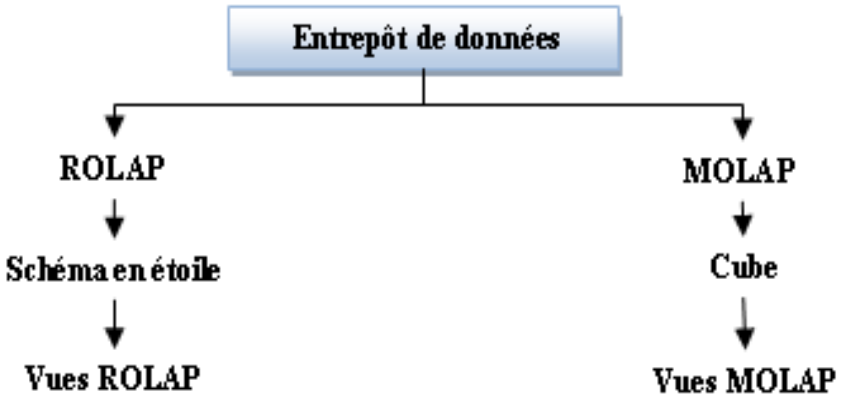

Figure 2-2 Les types de PSV.

3.4 Les travaux consacrés à la sélection des vues matérialisées

3.4.1 Les travaux d' Harinarayan et aL

Harinarayan et aL [HRU96] ont proposé un algorithme d'optimisation à base du treillis de vues. Il admet principalement en entrée un treillis de vues, et exploite un modèle de coût pour construire une configuration de vues à matérialiser.

Un treillis de vues est un graphe acyclique orienté qui représente la relation hiérarchique qui existe entre toutes les vues. Les nœuds représentent les vues (ou requêtes) et les arêtes représentent la relation de dépendance entre les vues. Chaque élément du treillis a une borne inférieure et une borne supérieure, pour deux éléments *x*, *y* de ce graphe il existe un chemin de y vers x si et seulement si la vue *x* dépend de la vue *y*. Les cellules du cube de données sont organisées dans des ensembles différents basés sur la position du **ALL** dans leur adresse. La valeur ALL représente un ensemble sous lequel l'assemblement est calculé. Les requêtes qui correspondent aux différents ensembles de cellules diffèrent seulement avec la clause groupe by.

Exemple : Dans la figure (2-3), nous avons trois dimensions : produit, fournisseur et client représenté respectivement par P, F et C. On peut les grouper de huit façons différentes :

1) Produit, Fournisseur, Client.
2) Produit, Client.
3) Produit, Fournisseur.
4) Fournisseur, Client.
5) Produit.

6) Fournisseur.
7) Client.
8) Aucun.

Aucun, veut dire qu'il n'y a aucun attribut dans la clause group by, c'est-à-dire qu'elle représente la vue qui contient les ventes totales pour toutes les dimensions.

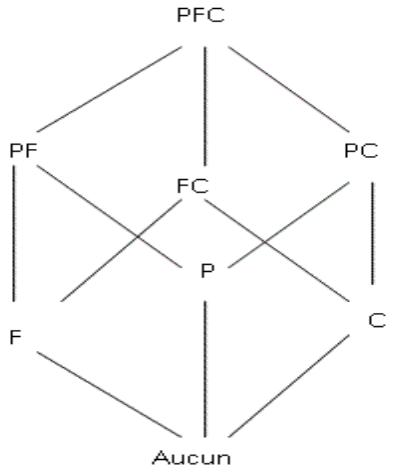

Figure 2-3 Exemple de treillis de vue.

Une relation de dépendance sur les requêtes dans un treillis de données est définit comme suit : étant donnée deux requêtes Q_1 et Q_2. Q_1 est dépendante de Q_2 ($Q_1 \leq Q_2$) si et seulement si Q_1 peut être exécutée en utilisant seulement les résultats de Q_2. L'opérateur (≤) impose un ordre partiel sur les requêtes, c'est à dire que l'opérateur ≤ peut ne pas être utilisé pour comparer certaines requêtes.

Afin de concevoir un treillis de vues on suppose que l'opérateur ≤ est un ordre partiel et qu'il y a un élément sommet, qui représente une vue sur laquelle toutes les autres vues dépendent. Le treillis est définit par l'ensemble des éléments L (requêtes ou vues) et la relation de dépendance ≤ par (L, ≤). Pour deux élément a et b du treillis (L,≤), $a < b$ veut dire que $a \leq b$ et $a \neq b$. L'ancêtre et le descendant d'un élément d'un treillis sont définit comme suit :

Ancêtre (a) = $\{b / a \leq b\}$.
Descendant (a) = $\{b / b \leq a\}$.

Le descendant immédiat d'un élément a est donné comme suit : Next (a) = $\{b / a < b$, il n'existe pas de c tel que $a < c$, $c < b\}$. Chaque élément du treillis est l'ancêtre et le descendant de lui-même.

L'algorithme 1, initialise l'ensemble des vues à matérialiser à la vue représentant la racine du treillis de vues. À chaque itération, dont le nombre est fixé à k, l'algorithme parcourt le treillis pour chercher la vue apportant le meilleur bénéfice. Cette vue est ajoutée à l'ensemble $Config_V$. Dans le cas où le modèle de coût prend en compte la taille des vues, au lieu de fixer le nombre k, l'espace de stockage alloué aux vues est prédéfini.

Algorithme 1 : Algorithme se sélection de vues de Harinarayan et al.
Config ← {vue de la borne supérieure du treillis} pour i=1 à k faire Sélectionner une vue v ∉ $Config_V$ tel que B (v, $Config_V$) soit maximal $Config_V$ ← $Config_V$ ∪ {v} fin pour retourner ($Config_V$)

L'algorithme présenté utilise un modèle de coût pour calculer le coût d'exécution et le bénéfice apporté par la matérialisation des vue. Etant donné $C(v)$ le coût d'une vue v, $B(v,ConfigV)$ le bénéfice apporté par la vue v, sachant qu'un ensemble $Config_V$ est préalablement sélectionné, est défini comme suit [HRU96].

- Pour chaque vue $w \leq v$, définir la quantité B_w :

a) soit u la vue de plus petit coût telle que $w \leq u$,

b) si $C(v) < C(u)$, alors $B(w) = C(v) - C(u)$. Sinon, $B(w) = 0$.

- $B(v, config_v) = \sum_{w \leq v} B(w)$

Ce modèle a été étendu en considérant le bénéfice de chaque vue par unité d'espace de stockage de cette vue. Dans ce cas, les attributs de regroupement sont supposés avoir la même cardinalité r, et les données du cube sont distribuées aléatoirement. Sous ces conditions, la taille d'une vue v est donnée par la formule suivante :

$$|v| = \begin{cases} r^i, r^i < m \\ m, Sinon \end{cases}$$

Où m désigne le nombre de cellules dans la borne supérieure du treillis et i le nombre d'attributs de regroupement de la vue v. Ce modèle a été adapté dans le cas où les attributs de regroupement ont des cardinalités différentes. Dans ce cas, la fréquence des requêtes et le coût de maintenance des vues ne sont pas pris en compte.

3.4.2 Les travaux de Baralis et al.

Baralis et al [BPT97] proposent deux heuristiques pour la construction de la configuration de vues à matérialiser. La première consiste à réduire le treillis des vues candidates en ne considérant que les vues associées aux requêtes de la charge. La deuxième heuristique élague les vues qui ne contribuent pas à l'amélioration des performances. L'idée est de déterminer à partir de quel niveau d'agrégation il n'est plus pertinent de matérialiser, en se basant sur l'estimation de la taille des vues.

Formellement, étant donnée FD et A des dépendances fonctionnelles et d'attributs, respectivement. L'algorithme 2 considère les dépendances fonctionnelles $\boldsymbol{fd_i \in FD}$ pour identifier quand des attributs dans A, qui apparaissent dans la partie droite $\boldsymbol{A^r}$ des dépendances fonctionnelles, produisent une vue dont la taille estimée est non significativement (suivant la valeur d'un seuil p fournie par l'utilisateur) différente de la taille de la vue obtenue en prenant les attributs de la partie gauche Al. Lorsque ce cas se produit, les attributs de la partie droite sont remplacés par ceux de la partie gauche. En effet, la vue v_1 obtenue en utilisant les attributs de la partie gauche des dépendances fonctionnelles est remplacée par la vue v_2 obtenue en utilisant la partie droite car $v_1 \leq v_2$.

Algorithme 2 Algorithme de l'heuristique de réduction de Baralis *et al.*

Entrée A un ensemble d'attribut et $\overline{p}$ un seuil
  répéter
    $stop \leftarrow$ true
    pour tout $fd_i \in FD$ **faire**
      si $\left((A \cap A_i^r \neq \emptyset) \text{ et } \overline{p} \text{ x } (|v^{A_i^l}| < |v^{A \cap A_i^r}|)\right)$ **alors**
        $A \leftarrow A - A_i^r$
        $A \leftarrow A \cup A_i^l$
        $stop \leftarrow$ false
      fin si
    fin pour
  jusqu'à *stop*
  retourner A

Le modèle du coût utilisé est définit comme suite : Soit $(L, \leq)$ un treillis de vues, pour répondre à une requête Q, on choisit un ancêtre de Q, appelé Q_a, déjà matérialisé. Cela implique qu'on doit aller à la table correspondante à Q_a pour répondre à Q. Donc, le coût de réponse à Q est le nombre de lignes de la table de Q_a utilisées pour répondre à Q. Ce problème est approché selon trois points de vue : (1) optimiser les temps, (2) optimiser l'espace de stockage ou bien (3) optimiser le temps tant que la contrainte d'espace de stockage n'est pas atteinte. Baralis et al. [BPT97] intègrent ces paramètres dans leur modèle de coût. La fonction de coût s'exprime alors :

$$C(Q,V) = \sum_{q_i \in Q} f_{q_i} C(q_i,V) + \underbrace{\sum_{v_i \in V} f_u C_u(v_i)}_{M(V)}$$

Où f_{qi}, f_{iu}, $C(q_i, V)$ et $C_u(v_i)$ sont respectivement la fréquence d'une requête d'interrogation q_i, la fréquence des mises à jour des vues, le coût de la requête q_i en présence des vues de l'ensemble V et le coût de maintenance de la vue v_i.

3.4.3 Les travaux de Gupta et al.

Gupta et al [GUP97] Ont basé dans leur recherche sur l'utilisation d'un plan d'exécution d'une requête afin de traduire les opérations réalisées à partir des tables de base pour fournir les résultats de cette requête. Chaque plan d'exécution d'une requête peut être vu comme une expression AND-DAG et tous les plans d'exécution considérés ensemble, forment une expression AND-OR-DAG. L'union des expressions AND-OR-DAG de chaque requête d'une charge donnée forme un graphe de vues AND-OR [VVK02]. Un graphe de vues AND-OR est un graphe bipartite, acyclique et orienté composé de deux types de nœuds : des nœuds d'opération et des nœuds d'équivalence [BB03].

3.4.4 Les travaux de Baril et al.

Baril et al.[BB03] Ont proposé un algorithme basé sur le calcul de niveaux de matérialisation pour chaque requête du graphe AND-OR. Un niveau de matérialisation est associé à chaque vue représentée par un nœud d'équivalence. Un niveau défini comme suit :

- ***level*(*root*) = 1**, la racine du graphe **root** représente le résultat de la requête ;
- ***level*(*view*) = *level*(*parent*(*view*)) + 1**, où la fonction ***parent*** donne le parent d'une vue du graphe.

La première étape de l'algorithme consiste à sélectionner un ensemble de vues candidates. L'idée est de chercher, pour chaque requête, un niveau de matérialisation intermédiaire entre

le premier niveau correspondant à la racine et le dernier niveau correspondant aux tables de base. En effet, la matérialisation du premier niveau donne un coût de traitement peu élevé et un coût de maintenance très élevé. En revanche, la matérialisation du dernier niveau donne un coût de traitement élevé et un coût de maintenance nul.

Ensuite, pour chaque requête, il faut calculer le niveau de matérialisation intermédiaire suivant la fréquence d'interrogation et de mise à jour des tables impliquées dans cette requête. Ce calcul est réalisé en deux temps. Dans un premier temps, les vues apportant un bénéfice local positif sont présélectionnées. Dans un deuxième temps, le coût total de chaque niveau de matérialisation, puis le niveau dont le coût d'exécution et de maintenance est minimal, sont calculés.

Après la recherche des vues candidates de chaque requête, la deuxième partie du processus d'optimisation consiste à chercher parmi ces vues celles qui optimisent le bénéfice global sous la contrainte d'espace de stockage. Dans la première étape de cette partie, les vues ayant un bénéfice global négatif sont élaguées. La configuration finale de vues *ConfigV* est ensuite sélectionnée en assimilant le problème de sélection de vues au problème du sac à dos, dont les objets sont les vues candidates, la contrainte est l'espace de stockage qui définie le poids des objets et la fonction à optimiser est bénéfice global.

3.4.5 Les travaux de Valluri et al.

Valluri et al. introduisent la notion de pertinence d'une vue matérialisée dans la sélection de vues basée sur le plan d'exécution des requêtes [VVK02]. La pertinence d'une vue indique comment la présence d'une vue dans une configuration peut affecter le calcul du bénéfice des autres vues de cette configuration. Par conséquent, le coût de traitement des requêtes et de maintenance des vues.

La pertinence entre deux vues dépend de la relation mère-fille qui peut exister entre ces vues, de l'appartenance ou de la non-appartenance de ces vues à la configuration et du bénéfice qu'apportent ces vues séparément. Les auteurs ont défini douze cas possibles de calcul de la pertinence entre deux vues matérialisées. La fonction *Pertinence* se base sur le calcul du bénéfice et la fonction *Materialization* indique dans quel cas il est pertinent de matérialiser ou pas et quelles sont les vues à matérialiser.

L'algorithme 3 initialise *ConfigV* à l'ensemble vide et parcourt le graphe de vues AND *G* tant que ses nœuds ne sont pas étiquetés comme « non visités ». À chaque itération, une matrice de pertinence des vues est calculée selon les cas présentés. Le couple de vues maximisant la pertinence est recherchée. Sa valeur de pertinence est ensuite mise à *1* car ce couple de vues n'est pas considéré dans les itérations futures.

La fonction *Materialization* indique si, pour ce couple de vue, la matérialisation est pertinente ou pas et quelles sont les vues à matérialiser. La matérialisation est effective si l'espace de stockage *S* n'est pas atteint. Le nœud du graphe correspondant aux vues du couple est finalement étiqueté comme visité.

Algorithme 3 Sélection des vues basée sur la pertinence
Entrée Un graphe And G et un espace de stockage S $Config_V \leftarrow \emptyset$ Etiqueter chaque nœud de G comme « non visité » **Tant que** tous les nœuds de G ne sont pas visités **faire** Calculer la matrice de pertinence des vues

```
Soit (v1, v2) un couple de vue tel que leur pertinence est maximale
Si Pertinence (v1, v2) < 0 alors
  Quitter
Fin si
Pertinence (v1, v2)=∞
Si Matérialisation (v1, v2) ≠ ∅ alors
  L← Matérialisation (v1, v2)
  Si taille (Config_V ∪ L) <S alors
    Config_V← Config_V ∪ L
  Fin si
  Matérialisation (v1, v2)← ∅
Fin si
Etiqueter comme « visité » le nœud correspondant aux vues matérialisées.
Fin tant que
Retourner (ConfigV)
```

L'algorithme proposé procède en deux niveaux. Le niveau le plus haut recherche, à partir des plans d'exécution locaux de chaque requête, le meilleur plan d'exécution global de ces requêtes. Le niveau le plus bas choisi le meilleur ensemble de vues minimisant le coût de plan d'exécution global. Dans chaque niveau, le problème d'optimisation est résolu à l'aide d'un algorithme évolutionniste. Deux représentations des solutions correspondant à chaque niveau sont à distinguer : représentations du plan d'exécution global et des vues matérialisées.

Plan d'exécution global : Pour n requêtes q_i; $i.n$, un plan d'exécution global peut être représenté sous forme d'un vecteur de n entiers $P_{1i}, ..., P_{kn}$, où P_{ki} indique le $k$ème plan d'exécution local de la requête q_i.

Vues matérialisées : La représentation des vues matérialisées dans le niveau bas est basée sur les graphes de vues AND. Chaque graphe est encodé sous forme d'une chaîne de caractères binaires. Chaque caractère correspond à un nœud du graphe. Si un caractère est à un, alors la vue dérivée de ce nœud correspondant à ce caractère est matérialisée. Dans le cas contraire, c'est-à-dire un caractère mis à zéro, la vue n'est pas matérialisée. La représentation binaire des graphes de vues facilite l'implémentation des opérations de croisement, de mutation et de sélection des algorithmes évolutionnistes.

3.4.6 Les travaux de Smith et al.

Cette méthode de sélection est basée sur le principe des ondelettes [SLJ04]. L'idée est de décomposer le cube de données en une hiérarchie d'éléments-vues structurés sous forme d'un graphe. Ces éléments fournissent une structure granulaire pour synthétiser les vues agrégées.

Un algorithme de sélection d'un ensemble d'éléments-vues est proposé pour matérialiser un cube de données.

Une vue agrégée d'un cube de données A est générée en agrégeant totalement A le long d'une ou plusieurs dimensions. Trois types d'agrégations sont possibles sur un cube A, une agrégation totale S^m, une agrégation partielle P^m ou une agrégation résiduelle R^m sur un nombre quelconque de dimensions i_m. Le cube généré par cette agrégation est appelé un *éléments-vue*. Par conséquence nous distinguons deux types d'éléments-vues; les éléments-vues résiduels et les éléments-vues intermédiaires. Les premiers sont générés par une agrégation résiduelle R^m. Tout élément-vue non résiduel est dit intermédiaire. Un ensemble d'éléments-vues est un ensemble d'agrégation partielle ou résiduelle d'éléments-vues. Cet ensemble est dit complet si le cube initial peut être reconstruit à partir des éléments-vues de

cet ensemble. Ce dernier est dit redondant s'il n'existe pas d'éléments-vues pouvant être générés à partir d'autres éléments du même ensemble.

Étant donnée la fréquence d'accès aux vues, les auteurs propose deux algorithmes de sélection d'éléments-vues. Dans le premier algorithme, les éléments-vues sélectionnés constituent une base non redondante et complète minimisant le coût des requêtes définies sur ce cube et le coût de stockage des éléments-vues de la base. Le deuxième algorithme relaxe la contrainte de non redondance pour sélectionner un ensemble d'éléments-vues minimisant le temps d'exécution des requêtes sous la contrainte d'espace de stockage. Notons qu'un ensemble d'éléments-vues forme une base pour un cube de données A si cet ensemble est complet. De plus, si cet ensemble n'est pas redondant, l'ensemble forme une base non redondante. La notion de bases d'éléments est importante pour la sélection des éléments-vues à matérialiser car une base d'éléments-vues non redondante peut représenter le cube de données sans surcharge au niveau du volume des données.

Un graphe d'éléments-vues organise ces éléments et fournit une structure de données pour évaluer la complétude, la non redondance et le bénéfice d'un ensemble d'éléments-vues. Il organise l'ensemble des éléments-vues suivant leurs dépendances directes et indirectes.

La complétude et la redondance d'un ensemble d'éléments-vues peuvent être déterminées en évaluant sa couverture dans un plan de fréquence à d dimensions correspondant à chaque bloc du graphe d'éléments-vues. À chaque élément-vue est associée une position X_m et une taille W_m dans le plan à d dimensions.

3.4.7 Travaux d'Agrawal et al

Agrawal et al. proposent une approche automatique de sélection d'index et de vues matérialisées basée sur l'analyse syntaxique de la charge des requêtes [ACN00, ACN01]. L'architecture de leur système est présentée par la Figure (2-4). Le système prend en entrée une charge de requêtes estimée représentative et procède en plusieurs étapes pour proposer une configuration de vues à matérialiser.

La première étape consiste à identifier des index, des vues matérialisées et des index sur ces vues qui soient syntaxiquement pertinents et potentiellement exploitables par les requêtes de la charge. Nous ne nous intéressons dans cette section qu'à l'étude de la sélection des vues matérialisées. Les vues matérialisées syntaxiquement pertinentes sont déterminées à partir de sous ensembles de tables (table-subsets). Un sous-ensemble de tables est composé d'une ou plusieurs tables présentes dans la base de données. Une vue est dite pertinente si l'ensemble des sous-tables dont cette vue est dérivée le sont aussi. Un sous-ensemble de tables T est dit syntaxiquement pertinent si la matérialisation d'une vue sur T est bénéfique pour au moins une requête de la charge et que cette vue réduit le coût de cette charge. La pertinence du sous-ensemble de tables T est mesurée par les deux métriques suivantes :

- TS – Cost(T) le coût total des requêtes exploitant les tables de T
- $TS - Weight(T) = \sum_i Cost(q_i) \frac{\text{somme des tailles des tables dans T}}{\text{Somme des tailles des tables exploit ées par qi}}$

Dans un premier temps, les sous-ensembles de tables dont le coût, calculé par *TS-Cost*, ne dépasse pas un seuil donné par l'utilisateur, sont élagués. La métrique *TS-Weight* est appliquée sur les sous-ensembles de tables restantes.

La deuxième étape consiste à solliciter l'optimiseur de requêtes afin d'élaguer certaines vues matérialisées dérivées des sous-ensembles de tables générés dans l'étape précédente. Une vue est élaguée si elle n'est bénéfique à aucune requête de la charge. L'idée est que si une vue n'est bénéfique à aucune requête de la charge, elle ne fait pas, par conséquent, partie de la configuration finale de vues. Étant donnés une requête q_i de la charge, un ensemble de vues V_i, et éventuellement un ensemble d'index sur ces vues, la meilleure configuration de vues et d'index est choisie pour q_i. Ce choix est réalisé à l'aide d'un algorithme glouton. À la fin de la deuxième étape, la meilleure configuration de vues et d'index est sélectionnée pour chaque requête de la charge.

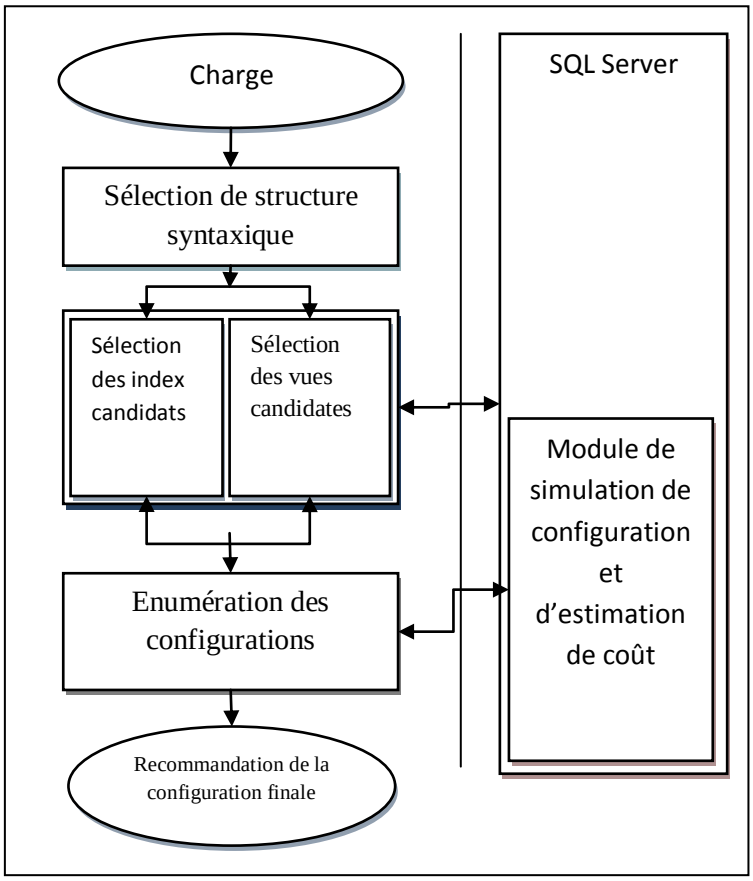

Figure 2-4 Architecture de système d'Agrawal et al.

Dans la troisième étape, il s'agit de fusionner les vues proposées afin qu'elles soient bénéfiques à plusieurs requêtes. La fusion est réalisée en gardant la structure en commun des vues parentes et en généralisant les différences de ces vues parentes. Cette fusion est guidée par un modèle de coût. En effet, si le coût d'exploitation de la vue v_{12}, obtenue par la fusion des vues v_1 et v_2, est seulement légèrement supérieure (en fonction d'un seuil fixé par l'utilisateur) au coût d'exploitation des vues parentes, alors la fusion n'est pas pertinente.

3.4.8 Les travaux de Bellatreche

L'idée principale de l'algorithme proposé par Bellatreche [BEL00], est la fusion des sous expressions communes trouvées dans le graphe de requêtes. L'algorithme *select_vue* proposé permet de sélectionner une configuration de vues minimisant la somme des coûts d'évaluation de requêtes et de maintenance des vues sélectionnées.

L'algorithme se caractérise par les quartes étapes suivantes :

- *La contruction des arbre algébriques :* cette étapes consiste à contruire un arbre algébrique pour chaque requête. Un plan d'éxécution optimal est obtenu pour chaque requête, en appliquant des règles de transformation des arbres.

- *La génération du PMEV :* une fois les graphes optimaux obtenues, les sous expressiosns communes sont fusionnées en PMEV (Plan Multiple d'Exécution des Vues).

- *Attributions des coûts :* à chaque nœud est ossocié un poid statique représantant le bénéfice, et un poid dynamique représentant de coût de maintennace.
- Le sélection des vues : cette étape consiste à ordonner les nœuds du PMEV en fonction de leurs poinds statiques, ensuite on calcule leurs poids dynamique, si ce dernier est positif la vue sera matérialsée.

3.4.9 Les travaux d'Aouiche

Le principe de cette stratégie de sélection de vues matérialisées est présenté à la Figure (2-5). La première étape construit à partir de la charge un contexte de classification. Ce contexte est modélisé comme une matrice ayant autant de lignes que de requêtes et autant de colonnes que d'attribut extraits des requêtes de la charge. La classification exploite des mesures de similarité et dissimilarité permettant de regrouper l'ensemble des requêtes qui sont syntaxiquement similaires. La similarité (respectivement la dissimilarité) sont calculés en utilisant les concepts du data mining à savoir le motif fréquent. Dans chaque groupe de requêtes, un processus de fusion est utilisé afin de construire une configuration de vues candidates. La configuration finale de vues est ensuite créée à l'aide d'un algorithme glouton.

Un modèle de coût est exploité par la suite afin d'évaluer le coût d'accès aux données en utilisant les vues, ainsi que le coût de stockage de ces vues. Aouiche [AOU02], procède comme suit pour proposer une configuration de vues à matérialiser :

- Extraction de la charge de requêtes,
- Analyse de la charge pour extraire les attributs représentatifs des requêtes,
- Construction du contexte de classification non supervisée des requêtes,
- Application de l'algorithme de classification Kerouac [JN03],
- Fusion des requêtes de chaque classe obtenue par classification pour construire l'ensemble des vues candidates,
- Construction de la configuration finale de vues à matérialiser.

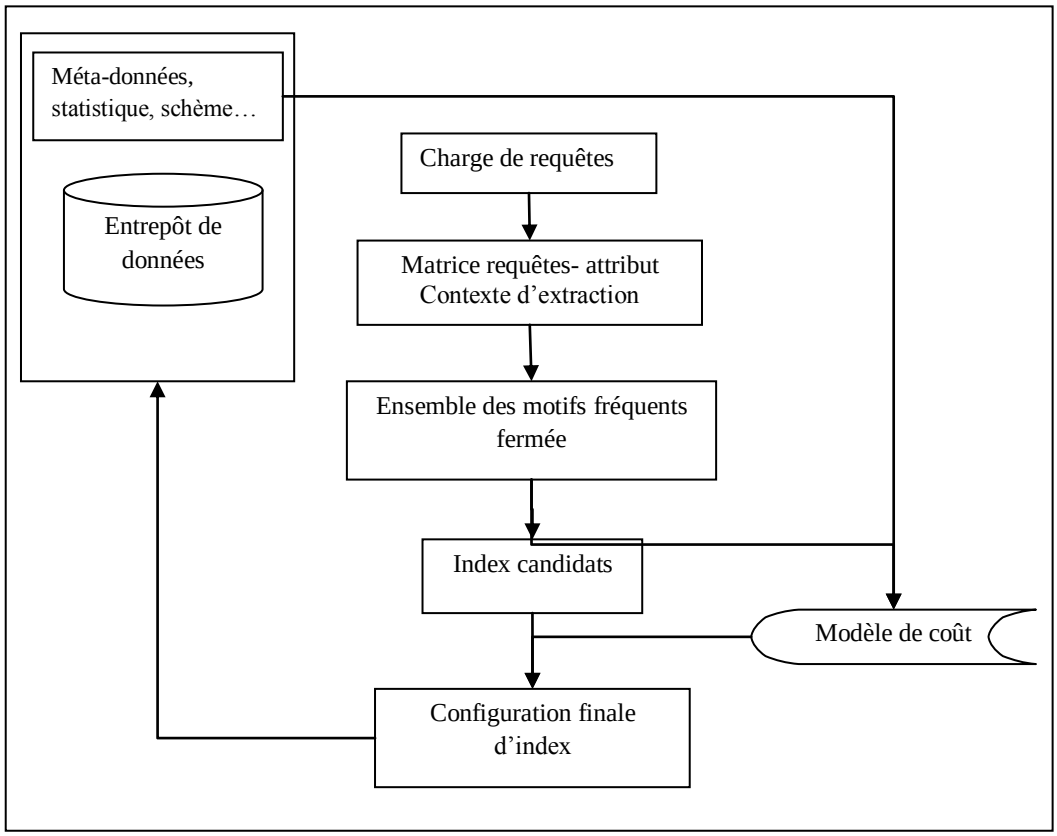

Figure 2-5 Architecture de système d'Aouiche et al.

3.4.10 Synthèse

Nous résumons dans tableau suivant les travaux traitant le problème de sélection des vues matérialisées selon: la méthode de construction des vues candidates, la méthode de construction de la configuration finale de vues, et le modèle de coût utilisé.

Travaux	Construction de l'ensemble de vues condidates	Algorithme de sélection final des vues	Modèle de coût
Harinarayan et AL	Treillis	Glouton	Mathématique
Baralis et al	Treillis	Glouton	Mathématique
Baril et al	Plan d'exécution	Sac à dos	Mathématique
Gupta et al.	Graphe de vues AND/OR	Glouton	Mathématique
Valluri et al.	Plan d'exécution	Glouton	Mathématique
Smith et al.	Ondelette		Mathématique
Agrawal et al.	Analyse de la charge	Glouton	Mathématique
Bellatreche	Analyse de la charge	Glouton	Mathématique
Aouiche	Analyse de la charge	Glouton	Mathématique

4. Les index

4.1 Les techniques d'indexation

Dans les systèmes de gestion de bases de données, l'accès aux données est d'autant plus lent que la base de données est volumineuse, ce qui nécessite un parcours séquentiel des données. Cette opération est très lente et pénalise l'exécution des requêtes, notamment dans le cas des

opérations de jointure où ce parcours doit souvent être effectué de façon répétitive. D'où l'utilisation des index s'impose afin d'améliorer le temps d'accès aux données tout en créant des chemins d'accès directs.

Les index peuvent être définis sur une seule colonne d'une relation, ou sur plusieurs colonnes de la même relation. Leurs utilisation peut optimiser les requêtes qui requière d'ordonner les tuples selon un ou plusieurs attributs comme les clauses GROUP BY et ORDER BY. Ils permettent aussi de réduire le nombre d'opérations d'entrée sortie pour les opérations de jointures.

Nous présentons les différents types d'index dans le contexte de bases de données ou entrepôts de données. Nous abordons par la suite l'intérêt des index ainsi que les index les plus appropriés dans les entrepôts de données.

4.1.1 B-arbre

Un index B-arbre [BM72] est une liste chaînée de nœuds dont la valeur est celle de l'index. Les feuilles de l'arbre font référence à une seule valeur, si cet index est construit sur un attribut clé, ou plusieurs valeurs, si cet index est construit sur un attribut non-clé des n-uplets de la table indexée. Cette référence spécifie l'emplacement physique du n-uplet sur le disque [BM72]. Ce type d'index permet d'organiser les fichiers pour des applications demandant des insertions, des suppressions et des recherches par intervalle de clé, ainsi que pour les mises à jour. Ce qui explique son utilisation dans la plus part des SGBD.

La figure (2-6) montre un exemple de B-arbre construit sur la table Films définie par le schéma suivant : **Films** (Film_ID, Film_Titre, Film_Année,...).

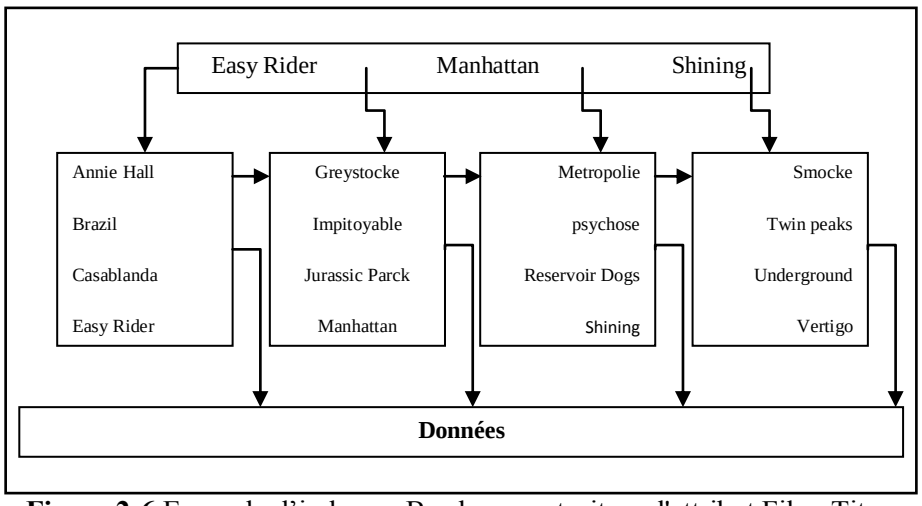

Figure 2-6 Exemple d'index en B-arbre construit sur l'attribut Film_Titre.

4.1.2 L'index de Hachage

Le principe du hachage est de calculer le numéro de bloc contenant un ensemble de pages, à partir d'une clé ***c*** et une fonction d'hachage dite ***h***. La recherche d'un enregistrement s'effectue ensuite séquentiellement dans le bloc. La table des blocs peut être résidente en mémoire centrale si elle est petite, et dans le cas contraire il faudra la ramener de la mémoire secondaire par partie en appliquant la fonction de hachage. Les tables de hachages sont des structures de données très couramment utilisées en mémoire centrale afin d'organiser des

ensembles et de fournir un accès performant à leurs éléments. Notons qu'une fonction de hachage mal conçue affecte tous les n-uplets à la même adresse et la structure dégénère vers un simple fichier séquentiel.

La figure (2.7) montre un exemple d'index de hachage construit sur la table Films déjà définie. La fonction de hachage est : ***H* (*titre*) = *rang* (*titre* [0]) *mod5***, où ***titre* [0]** désigne la première lettre du titre d'un film.

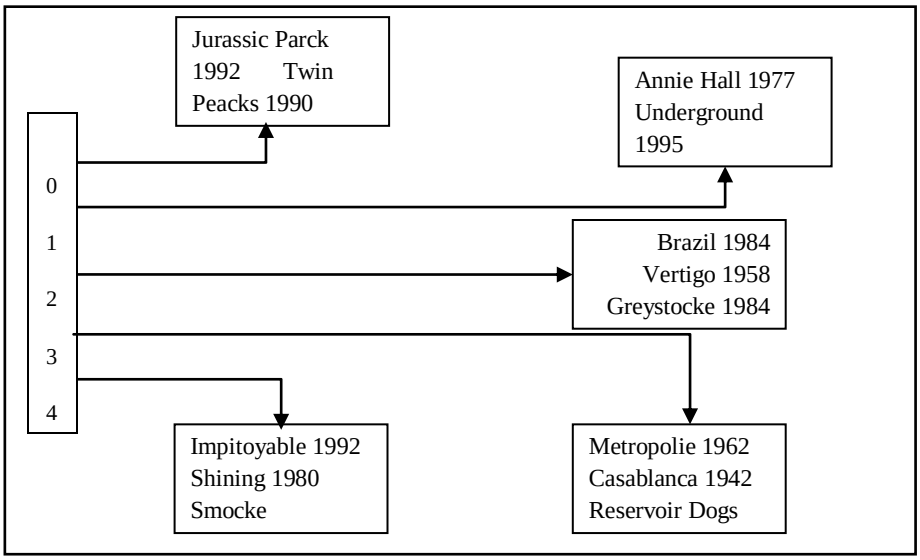

Figure 2-7 Exemple d'index de hachage construit sur l'attribut Film_Titre.

4.1.3 L'index bitmap

Un index bitmap considère toutes les valeurs possibles pour un attribut donné, que la valeur soit présentée ou non dans la table. Pour chacune de ces valeurs possibles, on stocke un tableau de bits avec autant de colonnes que de valeurs distinctes de la colonne indexée et autant de ligne que la table. Un bit est à 1 si la ligne de la table a la valeur correspondante à la colonne, 0 dans le cas contraire. La figure (2-8) montre un exemple d'index bitmap de jointure.

Index sur le sexe de l'employé

Nombre de colonne = nombre de valeurs distinctes

Matricule	Sexe	
	F	M
952636	1	0
596836	0	1
598412	1	0
596322	1	0
599823	0	1
599823	0	1

Figure 2-8 Exemple d'index Bitmap.

Ce type d'index est souvent utilisé lorsque les données de la table ne sont presque jamais modifiées, à savoir les applications décisionnelles OLAP. Ce type d'indexation est alors moins performant car sa mise à jour étant assez coûteuse.

4.1.4 L'index de projection

Les index de projection sur une colonne sont constitués d'une séquence stockée de valeurs de la colonne, qui apparaissent dans le même ordre que le numéro de n-uplet dans la table dans laquelle les valeurs ont été extraites [BKS00]. Ils représentent une copie parfaite de toute une colonne d'un attribut donné. Ce type d'index est utilisé non seulement pour améliorer d'autres techniques d'indexation traitant des requêtes qui implique deux ou plusieurs colonnes mais aussi il est très bénéfique pour l'optimisation des requêtes contenant la clause GROUP BY. La figure (2-9) montre un exemple d'index de projection.

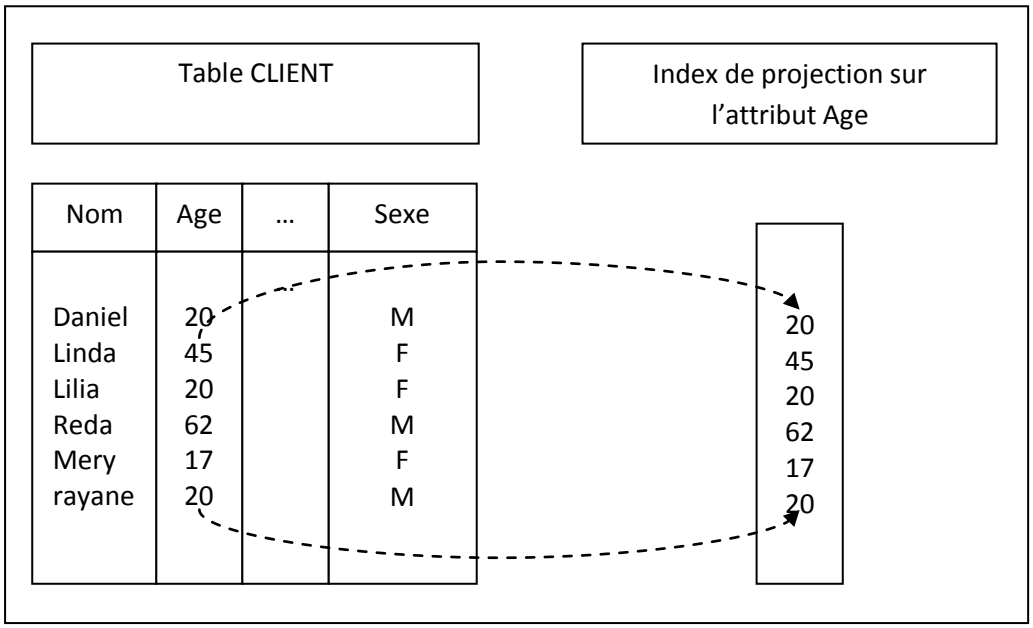

Figure 2-9 Exemple d'un index de projection.

4.1.5 L'index de jointure

L'index de jointure matérialise des liens entre deux tables par un biais à deux colonnes contenant les identifiants des n-uplets joints. Ce type d'index est souvent utilisé par les bases de données relationnelles afin d'optimiser les opérations de jointures. En faite, les requêtes complexes définies sur ce type de base de données demandent fréquemment des opérations de jointure entre plusieurs tables. La figure (2-10) montre un exemple d'index de jointure.

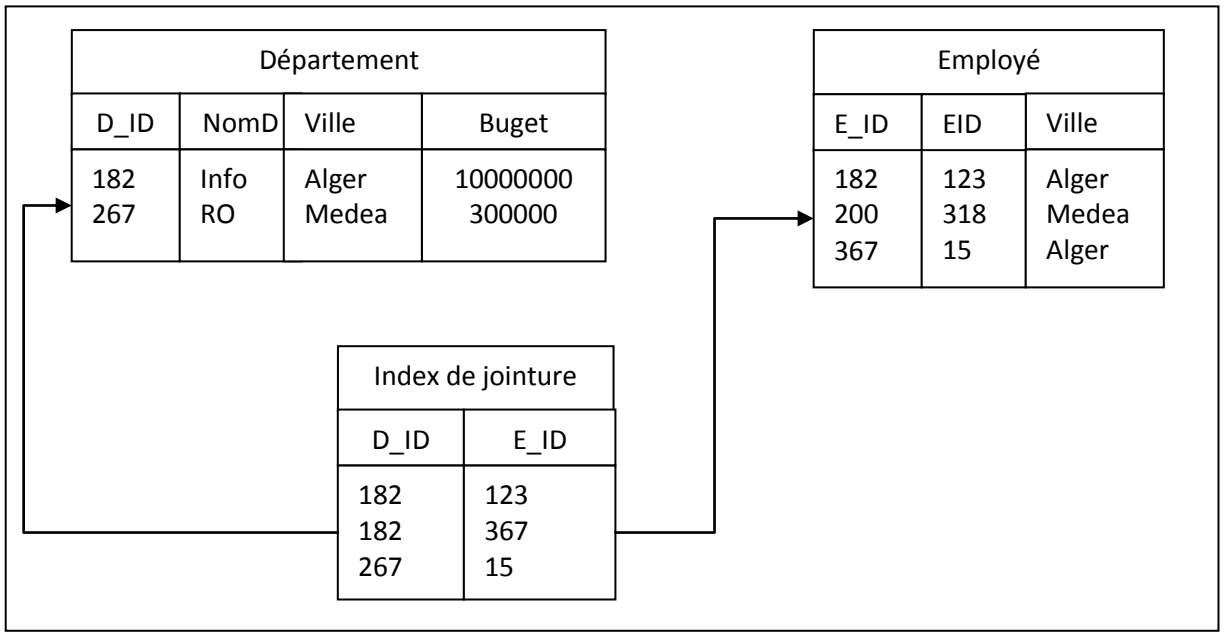

Figure 2-10 Exemple d'un index de jointure.

4.1.6 L'index de jointure en étoile

Ce type d'index est plus adapté aux requêtes définies sur un schéma en étoile. Il peut contenir toute combinaison de clés étrangères de la table des faits. Un index de jointure en étoile peut être défini comme une combinaison contenant la clé de la table des faits et une ou plusieurs clés primaires des tables de dimension.

Nous distinguons deux types d'index de jointure en étoile ; l'index de jointure en étoile complet construit en joignant toutes les tables de dimensions et la table des faits. Et l'index de jointure partiel construit en joignant certains des tables de dimensions avec la table des faits. En conséquence, l'index complet est bénéfique pour n'importe quelles requêtes posées sur le schéma en étoile. Il exige cependant beaucoup d'espace pour son stockage.

4.1.7 Index bitmap de jointure

C'est un index généralement calculé sur la table de faits et l'attribut indexé de faible cardinalité représente un attribut d'une table de dimension. Pour obtenir cet index, il suffit de faire la jointure entre la table de fais et la table de dimension puis de calculer l'index sur la clé de la table de fait avec l'attribut d'indexation. Ainsi, une instance « i » de l'index est à « 1 » si l'instance de la table de dimension correspondante à la valeur de l'attribut indexé peut être jointe avec l'instance « i » de la table de fait, 0 sinon.

4.2 Formalisation du problème PSI

Le problème de sélection des index PSI consiste à partir d'un ensemble de requêtes décisionnelles et la contrainte d'une ressource donnée (l'espace, le temps de maintenance, etc.) à sélectionner un ensemble d'index afin de minimiser le coût d'exécution des requêtes.

Formellement, étant donnée un ensemble I d'index candidats $I = \{i_1, i2, \ldots, i_n\}$, et un ensemble $Q = \{q_1, \ldots, q_m\}$ de requêtes de la charge et S la taille de l'espace disque alloué par

l'administrateur pour stocker les index à sélectionner, le PSI permet de trouver une configuration d'index *ConfigI* tel que :

- Le coût d'exécution des requêtes de la charge soit minimal.
- L'espace de stockage des index de *ConfigI* ne dépasse pas *S*.

Les premières solutions proposées pour résoudre le PSI reposent sur l'intervention de l'être humain, à savoir l'administrateur de l'entrepôt de données, et cela afin de sélectionner l'ensemble des index candidats. Cependant, ce choix dépend du degré d'expertise de l'administrateur. D'où la proposition d'autres stratégies automatiques basées sur l'analyse syntaxiques des requêtes Q. Cette analyse permet d'identifier les attributs intéressants à indexer.

Le problème de sélection d'index est connu NP-Complet. De ce fait, il n'existe pas d'algorithme exhaustif proposant une solution optimale en un temps fini. Les travaux de recherche ont distingué deux types d'algorithme de sélection d'index:

4.2.1 Algorithmes génétiques

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée, en un temps correct, à un problème d'optimisation, lorsqu'il n'existe pas (ou qu'on ne connaît pas) de méthode exacte pour le résoudre en un temps raisonnable. Ils ont été adaptés au problème de sélection d'index [KLT03].

L'algorithme génétique utilise en entrée l'ensemble d'index candidats, ou chaque index est assimilé à un individu afin d'optimiser le coût de la charge en présence d'une configuration d'index. La construction combinatoire des configurations d'index est réalisée à l'aide des opérateurs génétiques de croisement, de mutation et de sélection.

4.2.2 Problème de sac à dos

Le problème de sélection d'index a été assimilé dans certains travaux au problème du sac à dos [ISR83, Gun99, VZZ+00, FR03]. Ce problème a été formulé comme suit:

En entrée:

- L'ensemble d'objets $O = \{o_1, o_2, ..., o_n\}$ ou chaque objet o_i a un poids *poids* (o_i) et un bénéfice *benefice*(o_i).
- Le sac à dos est de taille *S*.

En sortie:

- Un sous-ensemble d'objets $N \subseteq O$ tels que sa taille ait comme borne supérieure *S* et que son bénéfice soit maximum.

Par analogie avec le problème de sélection d'index, un objet est un index, le poids et le bénéfice d'un objet représentent le coût de stockage d'un index et le coût de la charge en présence de cet index, la taille du sac à dos correspond à l'espace disque alloué par l'administrateur pour stocker les index sélectionnés et enfin l'ensemble *N* correspond à la configuration finale d'index.

4.3 Travaux consacrés à la sélection des index dans le contexte des Bases de données

4.3.1 Les travaux de Frank et al.

Frank et al. [FON92] ont proposé un outil d'aide pour le choix d'index dans une base de données. La sélection des index repose sur des échanges entre l'outil et l'optimiseur de requêtes. Ces échanges permettent de calculer le gain de performance qu'apporte l'utilisation d'un ou plusieurs index sur la performance des requêtes. Le gain est défini par la différence entre le coût d'exécution avant et après la création des index. Pour calculer ce coût, l'approche repose sur le modèle de coût utilisé par l'optimiseur de requêtes. La sélection d'index se fait requête par requête en plusieurs itérations. Dans chaque itération, l'outil demande à l'optimiseur de lui donner le meilleur index parmi un ensemble d'index candidats. L'optimiseur lui recommande un index avec le coût de la charge de requêtes en matérialisant cet index. L'échange entre l'outil et l'optimiseur s'arrête lorsqu'il n'y a plus d'index à proposer. L'approche de sélection d'index est constituée des étapes suivantes :

- Une requête de la charge est soumise à l'optimiseur de requêtes avec un ensemble d'index initial.
- L'ensemble des index utilisés pour la requête courante est stocké avec le gain de performance pour la requête.
- De nouveaux ensembles d'index sont générés et l'étape 2 est réitérée jusqu'à effectuer un parcours séquentiel.
- Les gains en performance de chaque index sont cumulés.
- Les index présentant un gain total positif sont enfin proposés à l'utilisateur.

Notons que l'outil proposé ne considère que les index secondaires et le coût de maintenance des index n'est pas prix en compte.

4.3.2 Travaux de Brunel, Rollin et al

Les auteurs [BRD01] ont proposé deux outils de sélection automatiques d'index, à savoir IUS (Index *Usage Statistics*) et ISCA (*Index Selection and Creation Algorithm*).

IUS permet de faciliter à l'administrateur du système le choix des index. Il reçoit en entrée une configuration d'index et une charge de requêtes et renvoie en sortie la même configuration ordonnées suivant leurs coûts pour chaque requête. Le coût des requêtes en présence des index est calculé par un dialogue établit entre l'outil et l'optimisateur des requêtes. Ce dialogue est répété pour tous les index de la configuration afin de leurs donner un rang pour chacune des requêtes de la charge.

L'outil ISCA, se base sur la charge des requêtes, ne contenant que les requêtes de sélection extraite du fichier Log du SGBD. Une table *Requete_Table* est utilisée afin de sauvegarder la somme des rangs, en binaire, des attributs d'une même table figurant dans une requête donnée de la charge. Le rang d'un attribut est un rang établi dans sa table d'appartenance. Par la suite, la fréquence d'apparition de chaque attribut ainsi que la fréquence des attributs combinée entre eux. Les combinaisons d'attributs ayant une fréquence qui dépasse un seuil fixé par l'administrateur du SGBD deviendrons des index crées sur les tables de correspondantes.

4.3.3 Travaux de Gündmn

Gündmn [GUN99] a introduit l'usage des techniques d'indexation dans le processus de sélection d'index. Cet aspect est très intéressant car la qualité d'un index, en termes d'espace de stockage et d'efficacité, dépend de la technique d'indexation adoptée sans prendre en compte les interactions qui peuvent exister entre les index.

Dans cette approche, l'ensemble des index possibles d'une table donnée est partitionnée en plusieurs sous-ensembles appelées classe d'équivalence, et représentant chacun les index définis sur le même attribut. Chaque attribut peut être indexés suivant plusieurs techniques et par conséquent, les index construits sur cet attribut suivant différentes techniques apportent des gains et occupent des espaces de stockage différents. L'administrateur fourni un ensemble d'index candidats qui sera utilisé par le processus de sélection pour y sélectionner un sous-ensemble *I* qui minimise, suivant une erreur tolérée, le coût de traitement des opérations de mise à jour et de sélection sans violer la contrainte d'espace de stockage. L'utilisation de cette méthode nécessite que les choix multiples d'index pour chaque attribut et les fréquences respectives des sélections et des mises à jour soient supposés fournis par l'administrateur du système.

Deux principales étapes caractérisent cette approche, l'optimisation locale et l'optimisation globale. L'optimisation locale permet de sélectionner un ensemble d'index par classe d'équivalence. Un ensemble d'index disjoints est donc sélectionné (disjoints car au plus un index est créé sur chaque attribut). Pour chaque attribut, tous les index candidats sont évalués en utilisant une fonction de coût. Cette fonction calcule le gain apporté par la matérialisation des index. Ce gain représente la différence entre le gain de coût en matérialisant l'index et le coût de création des index. Les index ayant un coût de création supérieur au gain apporté sont automatiquement éliminés. Le résultat de cette étape est un ensemble d'index I où chaque élément de cet ensemble est un index défini sur une seule classe d'équivalence.

L'optimisation globale vise à minimiser le coût d'exécution total en considérant tous les index dans l'ensemble I construit dans l'étape précédente. La fonction objective utilisée représente le gain en coût d'exécution avant et après création d'un ensemble d'index. L'objectif recherché est de trouver le sous-ensemble d'index dans I permettant d'une part; de maximiser le gain, et d'autre part, de vérifier que la taille de ce sous-ensemble d'index ne dépasse pas l'espace de stockage réservé aux index. Notons que la recherche d'une configuration d'index I par cette approche revient à résoudre le problème du sac à dos binaire.

4.3.4 Travaux de Choenni et al.

Choenni at al. [CBC93] ont basé sur un modèle mathématique afin de résoudre le problème de sélection d'index. L'algorithme de sélection d'index utilise en entrée une fonction de coût C, un sous ensemble de requêtes W_{red} contenant les requêtes de la charge W pour lesquelles aucun index parmi la configuration initiale n'est avantageux, et une configuration initiale d'index pertinente pour chaque requête de la charge. L'algorithme renvoie un ensemble d'index avantageux et un autre ensemble d'index désavantageux, pour des groupes de requêtes constituant des sous ensembles de W_{red}. Un index avantageux appartient à la configuration d'index optimale et, par opposition, un index désavantageux n'appartient pas à la configuration optimale.

L'algorithme de sélection permet par la suite de fusionner les ensembles d'index avantageux et les ensembles d'index désavantageux afin de n'est avoir qu'un seule ensemble d'index avantageux et un seul ensemble d'index désavantageux pour pouvoir résoudre le problème se sélection d'index avec une recherche exhaustive. Le temps et la faisabilité de cette recherche dépend du nombre d'index engendré.

4.3.5 Travaux de Kratpka et al.

La solution proposée par Kratpka et al, est basé sur l'utilisation des algorithmes génétiques [KLT03]. L'idée est de diviser l'ensemble initial d'index en plusieurs configurations dites actives selon un certains modèle de coût dans le but de construire un ensemble d'index tel que le temps total nécessaire pour l'exécution de toutes les requêtes soit minimal, et donc, le total des gains est maximal. Un algorithme génétique est appliqué par la suite sur une population d'individus N_{pop} N_{elit} qui représente le nombre d'individus élus pour survivre à la prochaine génération. Les valeurs de la fonction objective à évaluer sont le temps de traitement des requêtes. Le temps calculé est stocké dans une table cache de taille N_{cache} pour accélérer le temps des calculs. Avant de calculer une valeur objectif d'un individu, la table cache est consultée en premier.

4.3.6 Travaux de Valentin et al.

Valentin et al [VZZ+00] ont développé un système dit le système DB2 Advisor (Figure 2-11). Ce système admet en entrée une charge de requêtes SQL et renvoie en sortie la configuration d'index qui optimise les requêtes de la charge. Il est composé d'une interface graphique pour la sélection d'index, d'un outil en ligne de commande pour recommander des index, de l'optimiseur de requêtes de DB2 qui évalue et recommande des index et d'une table de conseils, créées temporairement dans un but de conseil utilisées comme un moyen de communication entre l'optimiseur et l'outil db2advis.

L'invocation de DB2 Advisor peut se faire par deux modes : interface graphique ou ligne de commande. L'optimisation en mode graphique appelle l'outil db2advis. Pour chaque requête, cet outil invoque l'optimiseur de requêtes de DB2 afin de recommander ou d'évaluer des index. L'optimiseur stocke les index dans une autre table de conseil. L'outil db2advis peut être invoqué directement en ligne de commande.

L'algorithme de sélection d'index est vu comme une extension de l'optimiseur de requêtes de DB2. En effet, l'optimiseur de DB2 est amélioré par l'injection de plusieurs index, dits index virtuels, et de leurs métadonnées stockées temporairement le temps du processus d'optimisation dans une table de conseil. Les index virtuels sont extraits des requêtes de la charge en effectuant une analyse syntaxique. En présence de ces index et des statistiques de la base de données, l'optimiseur génère le plan d'exécution optimal d'une requête donnée. Si ce plan contient un ou plusieurs index, ces derniers sont recommandés. Une fois la recommandation d'index est faite, l'outil db2advis modélise le problème de sélection d'index comme le problème du sac à dos. L'avantage de la solution est le fait que l'algorithme de sélection des index est intégré dans l'optimiseur ce qui supprime pratiquement le coût des appels fréquent à l'optimiseur.

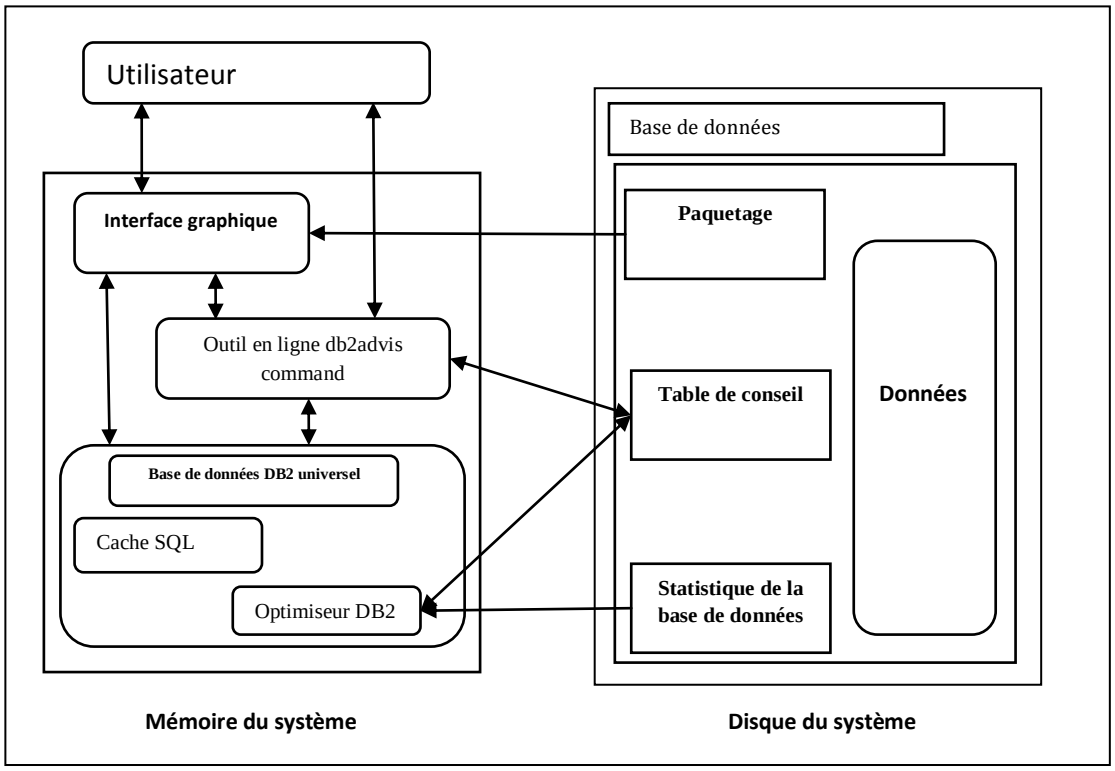

Figure 2-11 Architecture du système DB2advisor de Valentin et al.

4.3.7 Travaux de Chaudhuri et al.

Dans le but de mettre en œuvre un outil qui contribue à résoudre le problème de la conception physique automatique, en particulier la sélection d'index, Microsoft a lancé l'outil AutoAdmin pour l'auto administration des bases de données, en assurant les performances comparables à celle d'une base de données gérée uniquement par l'administrateur humain [MIC01].

L'architecture générale de l'outil de sélection d'index (*Index Selection Tool*) **[**CHN97**]** est représentée par la figure Figure (2-12). L'IST permet d'extraire un ensemble d'index candidats souhaitable pour une base et une charge de données. Notons que les index peuvent être mono ou multi indexés. Un modèle de coût est ensuite appliquer pour estimer le coût de toutes les requêtes de la charge de données.

Dans la première itération, l'IST ne prend en charge que les index mono-attribut. Dans la deuxième itération, il ne considère que les index mono-attribut déjà trouvé dans la première itération et les index à deux attribut; et ainsi de suite dans les itérations suivantes. Cet algorithme de sélection passe par les trois phases:

- **La sélection des index candidats :** Pour chaque requête de la charge, IST considère l'ensemble des attributs indexables comme les index candidats de départ. L'idée est de construire une configuration d'index à partir de ces index candidats, en se base sur le coût estimé par l'optimiseur de requêtes du SGBD **[**CHN97**]**. L'ensemble d'index résultat est l'union des configurations obtenues pour chaque requête.

- **Élagage des configurations d'index :** L'élagage permet de supprimer un certain nombre d'index qui ne sont d'aucune utilité ou qui amènent un gain de performance peu

significatif. Il se fait par un algorithme glouton, via un dialogue permanent avec l'optimiseur de requêtes . Le résultat est un ensemble d'index constitué de l'union des configurations obtenues après élagage. Cet ensemble d'index est dite mono attribut.

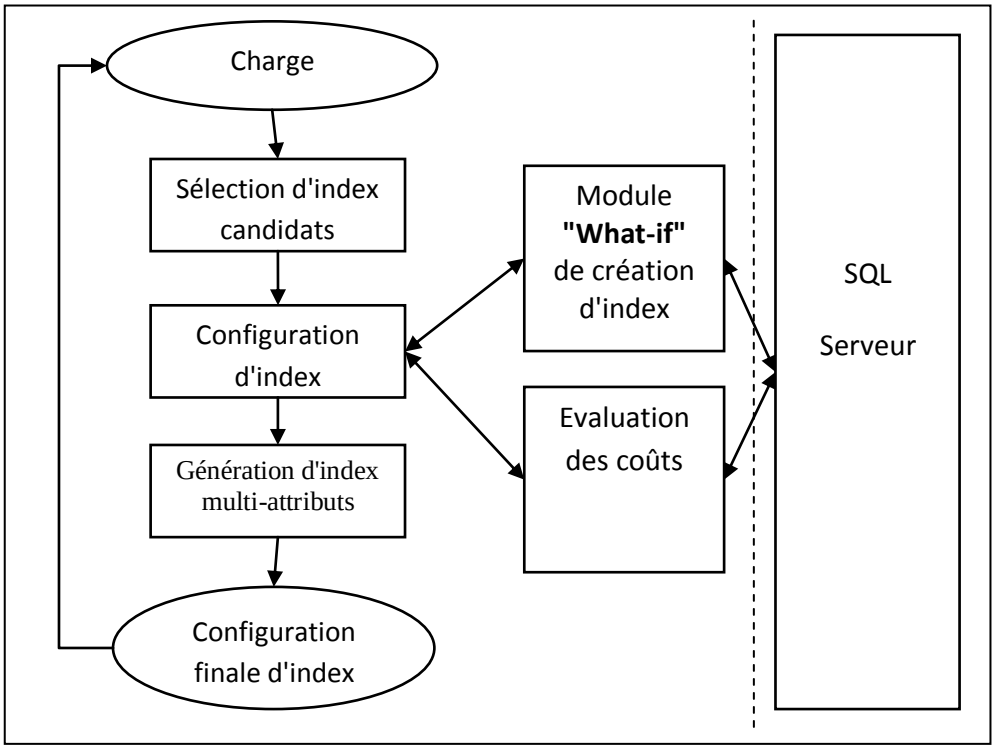

Figure 2-12 Architecture générale d'IST.

- **Génération des configurations d'index multi-attributs :** permet de construire des index multi attribut en utilisant les index mono-attribut obtenus après élagage, dans le but de réduire le nombre de structures, améliore le taux d'optimisation des requêtes et réduit l'espace de stockage.

- **Sélection de la configuration finale :** à partir de l'ensemble de *n* index obtenu précédemment, les *k* meilleurs sont sélectionnés par un algorithme glouton. A chaque itération, l'algorithme ajoute un index et fait appel à l'optimiseur pour évaluer le coût d'exécution des requêtes. L'algorithme s'arrête lorsque l'ajout d'un index n'apporte aucune amélioration.

L'inconvénient majeur de cette démarche est l'explosion du nombre d'index générés à la troisième étape, ce qui rend le nombre de combinaisons possibles de fusion d'index encore plus complexe. De plus, les appels fréquents de l'optimiseur dégradent largement les performances.

4.3.8 Travaux de Feldman et al.

Feldem et al. ont proposé un outil d'assistance pour la sélection des index à base de connaissances appelé DINNER [FR03]. Le principe du système proposé est d'utiliser une base de connaissances extraite de différentes sources : l'expertise de l'administrateur de la base de données, le manuel du SGBD utilisé, des cours sur l'administration des bases de données, etc. Ces connaissances sont modélisées par la suite par un schéma de représentation

de connaissances afin de trouver les index possibles, les chemins d'accès qui utilisent ces index, et le coût des chemins d'accès.

La première phase de sélection permet la construction d'un graphe d'exécution de chaque requête. Ce graphe représente l'ensemble des solutions possibles que peut utiliser cette requête. L'idée est de décomposer les requêtes de jointures en plusieurs requêtes définies sur une seule table. Un nœud du graphe représente les solutions intermédiaires, les feuilles du graphe d'une jointure représentent les requêtes définies sur une seule table, et la racine du graphe représente l'ensemble des solutions possibles. Un nœud intermédiaire peut être un OR, dans le cas ou si ses fils immédiats offrent plusieurs solutions alternatives ; ou il peut être un nœud AND si les solutions qu'il représente sont composées des solutions représentées par ses fils immédiats. Après la génération des graphes de chaque solution, l'outil trouve le chemin d'accès possédant une forte probabilité d'être choisi par l'optimiseur lors de l'exécution d'une requête et estime le temps de parcours et l'espace de stockage de ce chemin d'accès. Le résultat de cette étape est un graphe pour chaque requête avec les coûts et les espaces de stockage estimés. Par la suite, l'outil élimine les mauvaises solutions suivant certains critères

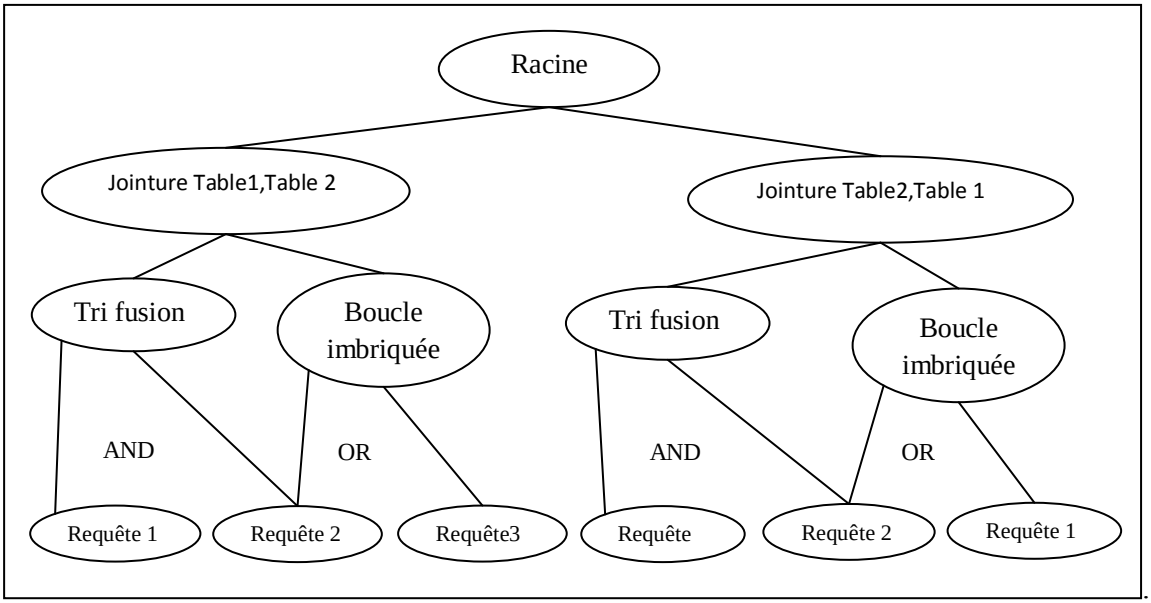

Figure 2-13 Graphe d'exécution pour une requête

L'étape suivante est l'unification, sans perte de solutions, des graphes des requêtes définies sur une seule table. Le but est de regrouper, sous un même arbre, les arbres résultant de la transformation et référençant la même table. Le coût des jointures est ensuite calculé sur la base des coûts des graphes des requêtes définies sur une seule table. Les meilleures solutions de la jointure sont ainsi trouvées. À la fin de cette étape, une configuration d'index, avec leur coût et leur espace de stockage, appartenant aux meilleurs graphes de jointure est obtenue. Le problème consiste à trouver dans cette configuration les meilleurs index. Ce problème est assimilé au problème du sac à dos.

4.4 Travaux consacrés à la sélection des index dans le contexte des entrepôt de données

4.4.1 Travaux de Golfarelli et al.

Golfarelli et al. [GRS02] ont proposé une approche heuristique qui sélectionne un ensemble d'index optimal à partir du schéma logique d'un entrepôt de données comprenant des vues matérialisées, une charge, des statistiques et une contrainte sur l'espace disque dédié aux

index. Le but est de déterminer le schéma physique optimal et un jeu d'index minimisant le temps d'exécution des requêtes tout en respectant la contrainte d'espace de stockage.

L'algorithme proposé procède en trois étapes distinctes. La première consiste à initialiser l'ensemble des index candidats *I* et optimaux *O*, ainsi que l'espace de stockage *S* nécessaire pour stocker les index à sélectionner. La deuxième étape sélectionne d'une manière gloutonne, à partir de l'ensemble des index candidats, ceux apportant un meilleur bénéfice par unité de stockage. Si, un tel index existe, il est ajouté à l'ensemble *O*. Si après un ajout, il arrive que tous les attributs composant la clé primaire d'une table de faits soient indexés, alors ces index sont transformés en un seul index multi-attributs construit sur la clé primaire de la table de faits. Après avoir sélectionné tous les index, la troisième étape permet de choisir les index primaires pour les tables de faits restantes.

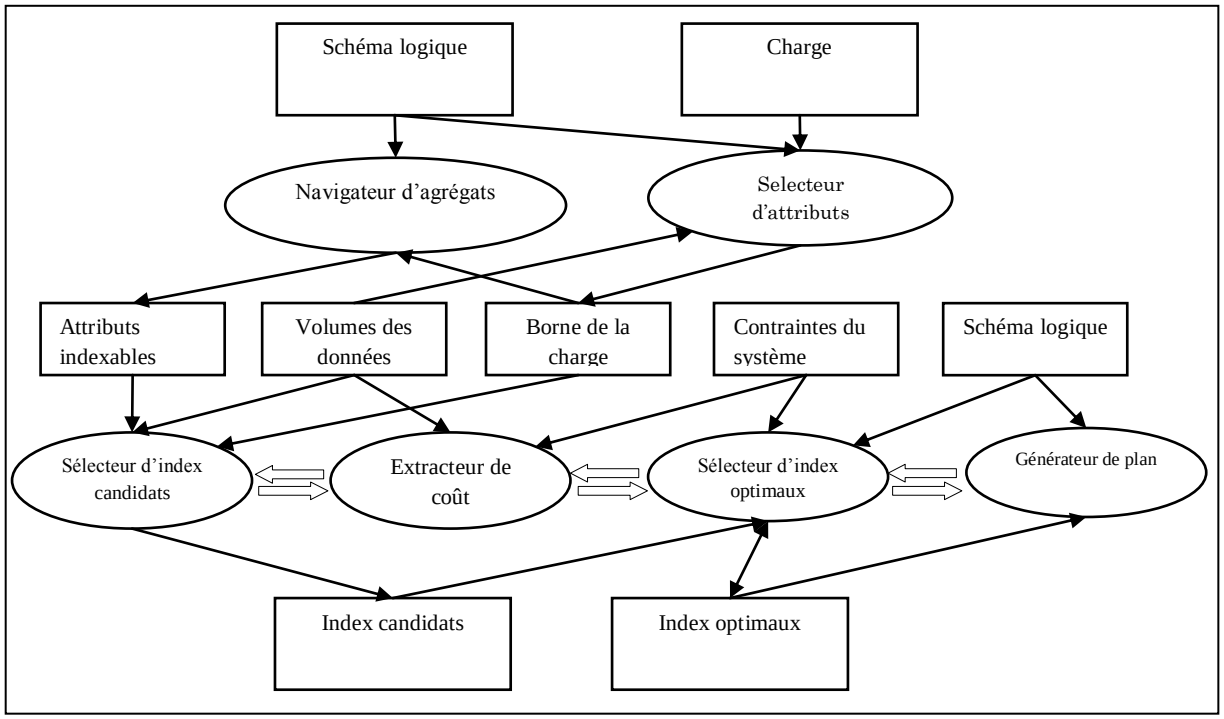

Figure 2-14 Architecture du système de sélection d'index proposé par Golfarelli et al.

4.4.2 Travaux de Labio et al

Les auteurs [LQA97] ont proposé une approche permettant de sélectionner une configuration optimale de vues et d'index, afin de réduire le temps de réponse des requêtes dans un contexte d'entrepôt de données, tout en minimisant le coût de maintenance des ces structures physiques. Cette approche utilise un algorithme A* pour résoudre le problème de **VIP**(***View Index Selection***). L'algorithme proposé utilise en entrée l'ensemble de toutes les vues à matérialiser et des index possibles, ainsi que l'ordre de propagation des données entre les vues et les index. Le résultat de l'algorithme est un sous ensemble de vues et d'index minimisant le coût total de maintenance. Au départ, l'ensemble des vues et des index est vide. L'algorithme A* ajoute ces structures d'une manière incrémentale, il s'arrête lorsque tous les vues et les index sont considérées et la solution trouvée présente un coût total minimum.

4.4.3 Travaux de Aouiche

Aouiche et al [AOU02] Ont basé dans leurs travaux sur les concepts du ***Data Mining*** à savoir la recherche des items fréquents, afin de sélectionner les index dans un entrepôt de

données modélisé par un schéma en étoile. L'idée de cette approche est d'utiliser un algorithme glouton, qui permet de rechercher les motifs fréquents à partir d'un contexte d'extraction donné. Nous commençons par donner les définitions des motifs fréquents et fermés puis présenter la démarche de sélection d'IJB, par la suite nous présentons la fonction objective utilisée par un algorithme glouton pour la sélection des index finaux et nous terminons par une analyse de l'approche.

Définition d'un motif fréquent : étant donnée $I=i_1, i_2,\ldots, i_m$ un ensemble de *m* items et $B=t_1, t_2,\ldots, t_n$ une base de données de n transactions, ou chaque transaction est composée d'un ensemble d'item $I' \sqsubseteq I$. Un sous ensemble *I'* de taille K est appelée un K-item set. Une transaction contient un motif *I'* si est seulement si $I' \sqsubseteq t_i$. Le support d'un motif I' est la proportion de transaction de B qui contiennent *I'*. Le support est donné par la formule suivante : $\text{support}(I') = \frac{|\{t \in B, I' \subseteq t\}|}{|t \in B|}$

Un motif est dite fréquent si et seulement si son support est inferieur a un seuil minimum *minsup*.

Définition d'un motif fréquent fermé: un motif fermé est un ensemble maximal de motifs communs à un ensemble d'objets. Un motif $i' \sqsubseteq I$ tel que support(i) ≤ minsup est appelé motif fréquent fermé.

L'algorithme **CLOSE** proposé est basé sur la génération d'un ensemble de motifs fréquents fermé afin de générer moins d'index candidats. Cela permet de réduire la complexité engendré lorsque le nombre de motifs fréquents généré devient très important. Il parcourt l'ensemble des générateurs des motifs fréquents par itération. Durant chaque itération, CLOSE détermine les fermetures des k-générateurs et calcule leurs supports. A la première itération, l'ensemble des 1-générateurs est initialisé aux 1-item sets. A chaque itération, l'algorithme considère un ensemble de k-item sets générateur et détermine l'ensemble des motifs fermés fréquents selon un seuil minimal du support *minsup*. La dernière étape de cet algorithme consiste à créer les (k+1)-générateurs qui seront utilisés lors de l'itération suivante pour construire l'ensemble des motifs fréquents fermés candidats. L'algorithme s'arrête lorsque l'ensemble de k-générateurs fréquents est vide.

L'approche de sélection est composée de six étapes :(1) extraction de la charge de requêtes, (2) analyse de la charge, (3) construction d'un contexte de recherche des motifs fréquents, (4) application de l'algorithme CLOSE sur ce contexte, (5) construction de l'ensemble des index candidats et (6) construction de la configuration d'index finale.

1. ***Extraction de la charge de requêtes :*** la charge de requêtes est extraite à partir du journal des transactions sauvegardé et maintenu automatiquement par le SGBD.

2. ***Analyse de la charge :*** la charge de requêtes obtenue est analysée afin d'extraire l'ensemble des attributs indexables. Ces attributs sont ceux qui font l'objet de prédicats de sélection dans les clauses **WHERE** des requêtes.

3. ***Construction d'un contexte de recherche des motifs fréquents :*** le contexte d'extraction est représenté par une matrice requêtes_attributs construite à partir des requêtes et des attributs indexables. Les lignes de la matrice représentent les requêtes et les colonnes les attributs indexables utilisée par chaque requête.

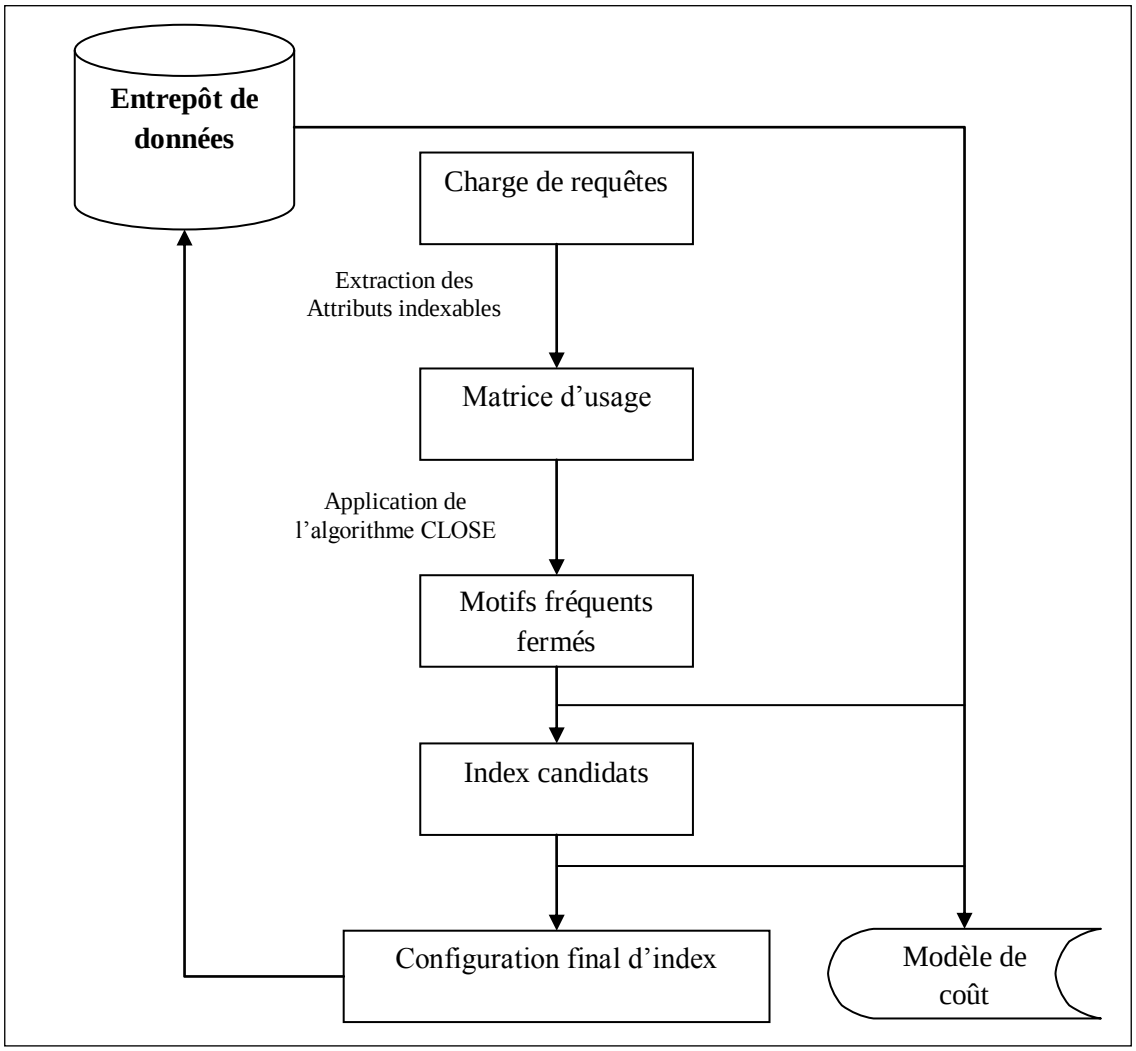

Figure 2-15 Architecture de sélection automatique d'index.

4. ***Application de l'algorithme CLOSE sur ce contexte :*** CLOSE permet d'extarire les motifs fréquents fermé. Chaque motif est composé d'un ensemble d'attributs de l'entrepôt de données. Un motif fréquent permet de générer un index s'il vérifie les conditions suivantes :

- Il contient des clès étrangères de la table des faits afin de pouvoir construire les clauses FROM et WHERE de la requête de création d'index.
- Il contient des clés primaire des tables de dimension, afin de pouvoir construire la clause WHERE pour joindre leurs tables de dimension à la table des faits. Ces table sont ajoutées dans la clause FROM.
- Et un ensemble d'attribut non clès des tables de dimension pour construite la clause ON.

5. ***Construction de l'ensemble des index candidats :*** l'ensembe des index candidats est construit à partir des motifs fréquents fermé résultant de l'application de l'algorithme CLOSE. Le but de cette étape est de verifier si les attributs contenus dans chaque motif fréquent permettent de créer un index binaire de jointure IJB sur l'entrepôt.

6. ***Construction de la configurartion finale :*** durant cette étape, un algorithme glouton est appliqué pour sélectionner une configuration d'index finale aprtir de l'ensembe d'index générés dans l'étape précedante. Cet algorithme procède en plusieurs itérations. La

première itérartion permet de calculer la fonction objectif pour chaque index condidat. La configuration intiale d'index est construite par l'index Imax maximisant la fonction objectif. Durant chaque itérartion, des index verifiant la même condition sont ajouté à la configuration courante. L'algorithme s'arrete si : aucune améliorartion de la fonction objectif n'est possible, tous les index ont été selectionnés ou bien l'espace de stockage disponible est saturé. Nottant que la fonction objectif est basée sur un modèle de coût qui permet de calculer la taille des index selectionnés ainsi le coût d'exécution des requêtes en presences des index.

L'avantage de cette approche est qu'elle prend en considération les interactions entre les JJB et qu'elle peut être appliquée pour n'importe quelle technique d'indexation. En revanche, l'approche utilise un seule critère pour la sélection qui est la fréquence d'utilisation de l'index par un maximum de requêtes, ce choix n'est pas suffisant pour sélectionner les index les plus optimaux.

4.4.4 Travaux de Bellatreche et al

Contrairement aux travaux d'Aouiche qui considère seulement les fréquences d'accès des attributs comme critère de génération des motifs fréquents fermés, les travaux de Bellatreche [BEL00] ajoute d'autre paramètres lors de cette génération à savoir la taille des tables de dimension, la taille de la page système etc. Deux algorithmes sont proposés, ***DynaClose*** et ***DynaCharm*** qui se basent sur une fonction fitness permettant de pénaliser chaque motif fréquent en prenant en compte les paramètres cités ci dessus. Pour un motif fréquent m_i, la fonction fitness est définie comme suit : $fitness(m_i) = \frac{1}{n} \times \left(\sum_{j=1}^{n} \alpha_j \times \sup_j \right)$

Ou n représente le nombre d'attributs non clés A_j dans m_i. sup_j représente le support de A_j et α_j est le paramètre de pénalisation définie par l'équation suivante : $\alpha_j = \frac{|D_j|}{|F|}$ ou $|D_j|$ et $|F|$ représentent respectivement le nombre de pages nécessaires pour stoker la table de dimension Dj et la table des faits F. La valeur minimum de la fonction fitness *minfit* est calculé comme suit: $\min fit = \frac{\min \sup}{|F|} \times \left[\left(\sum_{j=1}^{d} \frac{|D_j|}{d} \right) \right]$ ou d représente le nombre de tables de dimension.

Une stratégie d'élagage d'espace de recherche d'index est utilisée par ces deux algorithmes afin d'éliminer les motifs qui ne peuvent pas générer un index de jointure. L'ensemble d'attributs indexables candidats généré est définie par l'union des attributs non clés appartenant aux motifs fréquents générés. Un algorithme glouton est utilisé pour sélectionner la configuration d'index finale sous une contrainte d'espace de stockage. L'algorithme glouton commence par l'index defini sur l'attribut ayant la cardinalité minimum, ajouter ensuite d'autres index itérativement jusqu'à ce que l'espace de stockage soit consommé ou tous les index sélectionnés.

4.5 Synthèse

Nous avons présenté dans ce chapitre les principaux travaux sur le problème de sélection des index. La plupart des approches proposées commencent par l'identification des attributs indexables, qui peut être manuelle ou automatique, ensuite elles utilisent des algorithmes de sélection (algorithme glouton ou dirigé par des techniques de Data Mining) afin de générer

une configuration d'index finale. La mesure de performance de ces approches est mesurée soit par un modèle de coût mathématique, soit par l'optimiseur du SGBD.

Le tableau suivant présente une comparaison entre les principaux travaux en se basant sur les critères que nous venons de citer.

Travaux	contexte	Sélection des attributs candidats	Algorithme de sélection	Modèle de coût
Frank et al.	BDD	Manuelle	Glouton	Optimiseur + module Whet-if
Brunel,Rollin	BDD	Automatique		Optimiseur
Gündem	BDD	Manuelle	Glouton	Mathématique
Choenni at al.	BDD	Manuelle	Glouton	Mathématique
Kratica et al.	BDD	Manuelle	Génétique	Mathématique
Valentine et al	BDD	Automatique	Glouton	Optimiseur
Chaudhuri et al.	BDD	Automatique	Glouton	Optimiseur
Feldem et al.	BDD	Automatique	Glouton	Mathématique
Golfarelli et al.	ED	Automatique	Glouton	Optimiseur
Labio et al	ED	Automatique	A*	Optimiseur
Aouiche et al.	ED	Automatique	Data Mining + Glouton	Mathématique
Bellatreche et al.	ED	Manuelle	Data Mining + Glouton	Mathématique

5. Discussion

Compte tenu des difficultés que nous avons pour exploiter les données d'un entrepôt directement, le recours aux structures d'optimisation est inéluctable si on veut obtenir des résultats probants. Chacune de ces techniques peut être utilisée à part comme elle peut être combinée avec une autre, toujours dans le but de d'atteindre le plus haut niveau d'optimalité, c'est-à-dire obtenir des temps de réponse de moins élevés, on peut avoir par exemple à combiner une optimisation par index et une vue d'une requête donnée Q_0, par une vue et une fragmentation…etc. Il existe tout de même des combinaisons qui ne sont pas autorisées car n'apportant aucun gain de temps.

Dans ce chapitre nous avons présenté les principales techniques d'optimisation de requêtes utilisées dans les bases de données et les entrepôts en particulier les vues et les index.

Les index et les vues matérialisées sont des structures physiques qui permettent d'accélérer l'accès aux données d'un entrepôt. La plupart des travaux existants dans le domaine de la sélection d'index et de vues matérialisées traitent ces deux structures de manière isolée. Ces structures engendrent cependant une surcharge de maintenance. Par ailleurs, elles partagent le même espace disque.

Le problème de sélection des index dans les bases de données classiques a été étudié depuis les années 70. Depuis, plusieurs travaux ont été menées ayant pour objectif de trouver une solution proche de l'optimale. Ces travaux ont été classés en deux catégories selon le modèle de coût utilisé pour mesurer la pertinence de cette solution. La première catégorie est basé sur

un modèle mathématique pour calculé le coût des index, et la deuxième est basé sur un optimiseur de requêtes du SGBD pour évaluer le coût de chaque index.

Dans le contexte des entrepôts de données, peu de travaux ont été menés pour l'adaptation des différents travaux proposés dans le contexte des bases de données. Ces travaux peuvent être classés en deux catégories: la première tente à optimiser le coût de maintenance des index tandis que la deuxième famille a pour objectif l'optimisation du temps de réponse des requêtes. Dans les deux cas, l'optimisation est réalisée sous la contrainte d'espace de stockage.

Nous avons présenté dans ce chapitre un état de l'art sur les différents travaux consacrés à la sélection des index et des vues matérialisées menés dans les contextes : bases de données classiques et entrepôt de données.

Dans le chapitre suivant, nous présentons un état d'art sur les méta-heuristiques et les colonies de fourmis en particulier.

Chapitre 3 : Les fourmis artificielles

1. Introduction

Le comportement des insectes sociaux est caractérisé par l'auto organisation. Les individus communiquent en changeant les propriétés locales de leur environnement et par le biais de ce moyen de communication limité, une sorte d'intelligence collective émerge [SEG03]. Les fourmis naturelles ont inspiré les « algorithmes à colonies de fourmis » (ou la méta-heuristique ACO).

L'optimisation par colonies de fourmis est une technique d'optimisation biomimétique inspiré par un travail de biologiste en 1983, reprise par des informaticiens en 1988 et largement exploitée et développée par Marco Dorigo dans les années 90. L'idée consiste à imiter le comportement des fourmis réelles qui collaborent, par exemple pour la recherche de sources de nourriture en mélangeant comportement d'exploration aléatoire et suivi des traces chimiques laissées par leur consœurs. Ces traces chimiques, les « phéromones », sont utilisées par les fourmis pour communiquer entre elles de manière indirecte, par le biais de l'environnement, une technique générale connue par les entomologistes sous le nom de stigmergie. C'est cette forme de communication ainsi que l'idée de faire coopérer une foule d'agents simples et localisés qui forme la base de l'heuristique développée par Dorigo.

D'abord appliquée aux problèmes du voyageur de commerce, l'optimisation par colonies de fourmis a rapidement prouvé son efficacité dans le cadre de l'optimisation combinatoire en général est s'est montré particulièrement profitable pour le problème du routage des paquets d'information dans les grands réseaux d'interconnexion. L'optimisation par colonies de fourmis forme aujourd'hui un champs de recherche à part entière [YAN03].

Dans ce chapitre, nous donnons un aperçu des principales caractéristiques des méta-heuristiques et les fourmis artificielles que nous utiliserons par la suite lors de la sélection des vues matérialisées et d'index.

2. Les méta-heuristiques

2.1 Définition d'une heuristique

L'heuristique est une méthode approchée conçue pour un problème particulier dans le but de fournir des solutions de bonne qualité durant un temps de calcul raisonnable. Les méthodes heuristiques utilisent la même information sur le problème durant tout le processus de recherche.

2.2 Définition d'une méta-heuristique

Une méta-heuristique est un schéma de calcul heuristique, général et adaptable à un ensemble de problèmes différents.

Les méta-heuristiques sont des méthodes générales de recherche dédiées aux problèmes d'optimisation difficile [SMH99]. Ces méthodes sont, en général, présentées sous la forme de concept d'inspiration. Elles reprennent des idées que l'on retrouve parfois dans la vie courante. Ces méthodes ont des inspirations de l'éthologie comme les colonies de fourmis, de la physique comme le recuit simulé, et de la biologie comme les algorithmes évolutionnaires et qui nécessitent quelques transformations avant de pouvoir être appliquée à la résolution d'un problème particulier.

Contrairement aux méthodes heuristiques, les méta-heuristiques utilisent une ou plusieurs informations sur le problème qui changent durant le processus de recherche (Algorithme Génétique: adaptation de la population, Optimisation par Colonie de Fourmis: Phéromone, Recuit Simulé : température,…etc.).

2.3 Caractéristique des heuristiques

Les heuristiques possèdent certaines caractéristiques à savoir :

- Elles sont de complexité raisonnable.
- Les solutions obtenues par l'utilisation des heuristiques doivent être proche de la solution optimale.
- La probabilité d'obtenir une solution de mauvaise qualité est faible.
- Et enfin, les heuristiques doivent être simple à mettre en oeuvre.

3. Les fourmis artificielles

3.1 Généralité

Les insectes sociaux, comme les fourmis, les abeilles ou les termites sont généralement imaginés d'une manière simple, des animaux non intelligents. Néanmoins, ils exhibent collectivement des compétences impressives pour résoudre les problèmes. Inspirant de ces insectes, les recherches dans les décennies passées ont guidé en quelques progrès fascinant dans le champ d'algorithmes naturels. Ces algorithmes imitent la nature d'une manière ou d'une autre. Les réseaux de neurones imitent la structure de notre cerveau humain et les algorithmes génétiques simulent l'évolution. Ils sont caractérisés par le parallélisme inhérent, l'adaptation, le feed-back positif et quelques éléments de l'aléatoire [DG04].

Les colonies de fourmis sont des systèmes distribués qui, dans l'aspect de la simplicité de leurs individus, présentent une forte structure d'organisation sociale. Comme un résultat de cette organisation, les colonies de fourmis peuvent accomplir les tâches complexes qui sont en quelques sortes loin dépassé les capacités d'individu pour une seule fourmi. Le champ des algorithmes de fourmis étudie les modèles dérivés à partir de l'observation du comportement réel de fourmis, et utilise ces modèles comme une ressource d'inspiration pour la conception de nouveaux algorithmes pour la solution pour les problèmes d'optimisation.

Les fourmis coordonnent leurs activités à partir de la stigmergie, une forme depuis une communication indirecte s'interposa par des modifications de l'environnement. Le succès de vie des fourmis en communauté, est du à leur faculté mis à profit pour coopérer, communiquer, et apprendre. Les travaux de recherche inspirés des comportements des fourmis ne cessent d'être développés. La méta-heuristique basée sur la colonie de fourmis est devenue une activité fructueuse dans plusieurs domaines d'application. Un parmi les plus exemples qui ont un succès des algorithmes de fourmis est connu sous le nom Ant Colony Optimization », ou ACO. ACO est inspiré par le comportement de colonies de fourmis, et fixe les problèmes d'optimisation discrets [DM04]. Cette méta-heuristique qui a été utilisée avec succès pour résoudre plusieurs problèmes d'optimisation combinatoires.

3.2 Principe de fonctionnement des colonies de fourmis

Pour mieux comprendre la façon dont fonctionnent les colonies de fourmis, prenons l'exemple traditionnellement donné pour illustrer leur capacité à trouver des chemins

optimaux. La figure (3-1) illustre une situation où il y a un nid, où les fourmis vivent, et une source de nourriture, que les fourmis doivent trouver et dont elles doivent ramener les provisions vers le nid [YAN03].

Lorsqu'une ouvrière chargée de rechercher de la nourriture quitte le nid, elle va déposer sur son chemin, à l'aide d'un aiguillon placé sur son abdomen, une fine trace de phéromone d'orientation. Lorsqu'elle retournera au nid après avoir trouvé une source d'approvisionnement, elle déposera à nouveau des phéromones, un peu différentes puisque ici une information sur la qualité du site trouvé sera incluse dans ce message. Dans tous les cas, la phéromone est placée là pour les autres fourmis de la colonie afin de les inciter à choisir cette piste.

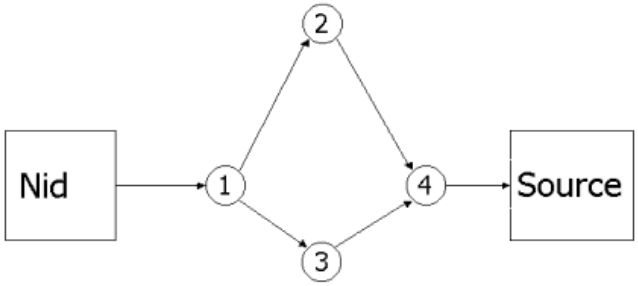

Figure 3-1 Un problème naturel typique : un nid, une source de nourriture et deux chemins, un court, un long [YAN03].

Ce marquage leur permet également de trouver le plus court chemin entre le nid et un site alimentaire. En effet, au départ chaque fourmi choisira l'une des pistes disponibles totalement au hasard. Très rapidement le chemin le plus court sera marqué par plus de phéromone, car le va-et-vient des fourmis y est plus rapide, et pour une même période, plus de fourmis ont le temps de le parcourir, donc de déposer leur message. Les nouvelles fourmis qui quittent le nid ont une tendance naturelle à favoriser la piste qui comporte la trace olfactive la plus importante [ROU01].Cette façon de procéder est déjà efficace en soi mais il manque deux phénomènes essentiels pour qu'elle soit tout à fait complète.

Le premier phénomène est qu'il arrive quelquefois que des fourmis étourdies se trompent et s'écartent du chemin de phéromones. Si, par chance, une fourmi égarée par erreur trouve un chemin plus court, la trace de phéromone qu'elle laissera derrière elle sera plus fraîche, indiquant par là même aux autres fourmis qu'il existe un chemin plus court pour accéder à la nourriture. Ainsi, c'est le mécanisme d'erreur dans le suivi de trace de phéromone qui permet la découverte de raccourcis, aboutissant à l'établissement d'un chemin optimal entre fourmilière et nourriture.

Le deuxième phénomène est que les phéromones s'évaporent dans le temps, rendant ainsi leurs traces éphémères. C'est ce deuxième mécanisme qui permet aux chemins établis de ne pas être statiques, et de s'adapter aux modifications de l'environnement. Supposons maintenant que seul le chemin le plus long soit disponible (une brindille obstrue le chemin court). Les fourmis vont donc l'utiliser et le charger en phéromones. Supposons maintenant que le chemin redevienne tout à coup disponible (la brindille a été poussée par le vent), notre colonie se trouve dans une situation non optimale : une piste de phéromones encourage les fourmis à suivre un chemin qui n'est pas le meilleur puisqu'elles suivent le chemin long alors que le court est à nouveau disponible. La situation est rétablie par le caractère volatil des

phéromones : sans apport de phéromone important, le chemin le plus long finit par s'effacer pour disparaître presque complètement. Grâce à l'action conjuguée de l'évaporation et de la croissance rapide de la concentration en phéromones sur le chemin court, le chemin long va rapidement tomber en désuétude et l'optimalité sera rétablie.

Cet exemple illustre bien la façon dont les colonies de fourmis s'adaptent de manière optimale à leur environnement en pondérant information locale, information stigmergique et comportement aléatoires [YAN03].

4. La Méta-heuristique basée sur les fourmis

4.1Définition

Une méta-heuristique de colonie de fourmis est un processus stochastique construisant une solution, en ajoutant des composants aux solutions partielles. Ce processus prend en compte une heuristique sur l'instance du problème, et des pistes de phéromone changeant dynamiquement pour refléter l'expérience acquise par les agents [DREO, 2005].

En pratique, la méta-heuristique de l'Algorithme de Colonie de Fourmis, a été utilisée dans plusieurs problèmes d'optimisation tels que : le problème de sac à dos, traitement parallèle, problème de voyageur de commerce, etc.

4.2La méta-heuristique ACO

Un parmi les plus exemples qui ont un succès des algorithmes de fourmis est connu sous le nom « Ant Colony Optimization », ou ACO. Les fourmis artificielles utilisées dans la méta-heuristique ACO sont des procédures stochastiques qui construisent des solutions en ajoutant itérativement une nouvelle solution à l'ensemble des solutions existant en tenant en compte de l'information heuristique et de la phéromone artificielle.

L'ACO peut être interprétée comme une extension d'une heuristique traditionnelle de recherche, qui peut être appliquée à plusieurs problèmes d'optimisation combinatoire. Ce qui fait la différence entre l'ACO et les heuristiques traditionnelles c'est l'adaptation de la phéromone durant l'exécution de l'algorithme pour tenir en compte de l'expérience de la recherche. L'interprétation de l'ACO comme une heuristique de construction extensible est évidente pour plusieurs raisons, le composant stochastique dans l'ACO permet aux fourmis de construire une large variété de solutions, au même moment l'utilisation de l'information heuristique peut guider les fourmis vers les meilleures solutions. En plus l'expérience de la recherche des fourmis est utilisée pour influencer dans un chemin évocateur la construction de la solution dans les futures itérations de l'algorithme [DOR01].

4.3Formalisation du problème ACO

Soit le problème de minimisation suivant (S,f,Ω), où S représente l'ensemble des solutions candidates, f la fonction objectif associée à chaque solutions $s \in S$. ($f(s, t)$ est la fonction objectif avec le paramètre temps t dans le cas ou la fonction objective dépend du temps (coût), et Ω est l'ensemble des contraintes. L'objectif est de trouver la solution optimale $S_{opt} \in S$ avec un coût minime et qui respecte la contrainte Ω.

Le problème de représentation d'un problème d'optimisation combinatoire (S,f,Ω) exploité par les fourmis peut être caractérisé comme ceci :

- On donne l'ensemble fini $C = \{c_1, c_2, ..., c_{nc}\}$ de composants, *nc* étant le nombre de composants. (l'ensemble C représente les composants du problème par exemple dans notre cas il représente les vues du treillis).

- L'état du problème est définie en terme de séquences $x = \{c_i, c_j, \ldots, c_k, \ldots\}$ sur les éléments de C. l'ensemble de toutes les séquences possibles sont noté par χ. La taille de la séquence *x* est le nombre de composants dans la séquence, il est noté par $| x |$.

- L'ensemble finit des contraintes Ω définit l'ensemble des états faisables χ', avec $\chi' \subseteq \chi$.

- L'ensemble S* des solutions faisables est données par $S^* \subseteq \chi'$ et $S^* \subseteq S$.

- Le coût $f(s, t)$ est associé à chaque solution candidate $s \in S$.

- Dans certains cas, le coût ou l'estimation du coût $J(x_i, t)$ peut être associé aux états au lieu aux solutions. Si x_j peut être obtenu en ajoutant les composants d'un solution à l'état x_i alors $J(x_i, t) < J(x_j, t)$.

Etant donné ces représentations, les fourmis artificielles construisent les solutions en parcourant le graphe $G = (C, L)$, où les nœuds sont le composants de *C*, et *L* est l'ensemble des arcs qui connectent complètement les composants de *C*. Le problème des contraintes Ω est implémenté dans une politique suivie par les fourmis artificielles [DOR01].

4.4Le comportement des fourmis dans l'ACO

Les fourmis peuvent être caractérisées comme des procédures de constructions stochastiques, qui construisent des solutions en parcourant le graphe $G = (C, L)$. Les fourmis se déplacent dans le graphe en suivant une politique de construction qui est une fonction du problème de la contrainte Ω.

En générale les fourmis essaient de construire les solutions faisables, mais si c'est nécessaire elles peuvent générer des solutions infaisables. Une phéromone peut être associée aux composants $c_i \in C$ et aux connecteurs $l_{ij} \in L$. τi est la phéromone associée aux nœuds et τ_{ij} est la phéromone associée aux arcs). Une valeur heuristique η peut aussi être associée aux nœuds et aux connecteurs (η_i pour les nœuds, et η_{ij} pour les arcs) cette valeur représente en général le coût. Ces valeurs (phéromone et la valeur heuristique) sont utilisées par la fourmi pour définir la probabilité du déplacement dans le graphe. Plus précisément chaque fourmi *k* de la colonie a les propriétés suivantes :

- Elle exploite le graphe $G = (C, L)$ pour chercher les solutions faisables avec le minimum coût.

- Elle a une mémoire M_k qu'elle utilise pour enregistrer les informations sur le chemin parcouru précédemment. La mémoire peut être utilisée pour construire les solutions faisables, évaluer la solution trouvée et retracer le chemin pour déposer la phéromone.

- On peut attribuer un état initial x^k_s et une ou plusieurs conditions de terminaisons e_k. Généralement, l'état initial d'une séquence contient un seul composant ou une séquence vide.

- Lorsqu'elle est dans l'état $x_r = < x_{r-1}, i >$, si aucune condition d'arrêt n'est satisfaite, elle se déplace vers le nœud j (qui appartient à ses voisins non encor visité), sont état devient $< x_r, j > \in \chi$.

- Souvent, les déplacements vers les solutions faisables sont favorisés, soit par la valeur de l'heuristiqueη, ou à travers l'utilisation de la mémoire des fourmis.

- Le déplacement entre les nœuds se fait par une probabilité de déplacement cette dernière inclue le taux de phéromoneτ, la valeur heuristiqueη, sa mémoire M_k et la contrainte du problème.

- La procédure de construction d'une fourmi k s'arrête lorsqu'une des conditions d'arrêt e_k est satisfaite.

- Lorsqu'un composant (nœuds) est ajouté à un l'ensemble des solutions, la fourmi met à jour la phéromone qui lui est associée ou aux connecteurs correspondant. Ceci est appelé mise à jour local de la phéromone ou mise à jour pas à pas en ligne (online step-by-step pheromone update).

- Une fois la solution est construite, elle peut retracer le même chemin et mettre à jour la phéromone associée aux composants ou aux connecteurs sélectionnés. Ceci est appelé mise à jour de la phéromone différée en ligne (online delayed pheromone update).

Il est important à préciser que les fourmis se déplacent de manière concourante et indépendante et chaque fourmi est assez complexe pour trouver une solution. Précisément, une bonne solution émerge en résultat de l'interaction collective des fourmis [DOR01].

5. Exemple d'adaptation des colonies de fourmis à un problème

Le premier exemple des algorithmes à base de colonies de fourmis (ACO) est le AS (Ant System « système de fourmis ») conçu pour le problème de voyageur de commerce [DOR01]. Pour illustrer la manière dont on peut transformer l'observation de colonies de fourmis réelles en algorithme d'optimisation, on reprend l'exemple de ce problème.

Le problème du voyageur de commerce, pour le rappeler brièvement consiste à trouver un chemin Hamiltonien dans un graphe complètement connecté. En termes plus imagés, il s'agit pour un voyageur de commerce de trouver le chemin le plus court pour visiter une et une seule fois chacune des *n* villes dans lesquelles il doit se rendre. Il s'agit sans doute du problème d'optimisation combinatoire NP-complet le plus utilisé comme test pour les nouvelles méthodes d'optimisation [YAN03]. Voici la modélisation du comportement des fourmis qui est proposée pour le problème du voyageur de commerce.

Les fourmis sont placées sur les sommets du graphe. Elles se déplacent d'un sommet à l'autre en empruntant les arêtes du graphe. Chaque fourmi possède les caractéristiques suivantes :

- la fourmi dépose une trace de phéromones sur l'arête (*i, j*) quand elle se déplace de la ville *i* `a la ville *j* .

- elle choisit la ville de destination suivant une probabilité qui dépend de la distance entre cette ville et sa position et de la quantité de phéromones présente sur l'arête, (règle de transition).
- afin de ne passer qu'une seule fois par chaque ville, la fourmi ne peut se rendre sur une ville quelle a déjà traversée, c'est pour cela que la fourmi doit être dotée d'une mémoire.

Pour éviter qu'une fourmi ne revienne sur ses pas, elle conserve la liste des villes qu'elle a déjà traversées. Cette liste, nommée liste-tabou est remise à zéro chaque fois que la fourmi a terminé un tour. La liste-tabou constitue la mémoire de la fourmi [MON00].

Les traces de phéromones sont modélisées par les variables $\tau_{ij}(t)$ qui donnent l'intensité de la trace sur le chemin (i, j) à l'instant t. l'intensité est exprimée par la probabilité de transition du sommet i vers le sommet j. La mise à jour des phéromones est effectuée une fois que toutes les fourmis sont passées par toutes les villes.

6. Synthèse

Dans ce chapitre, nous avons présenté un état de l'art sur les méta-heuristiques, l'optimisation en utilisant les colonies de fourmis. Nous nous sommes intéressées particulièrement aux technique d'optimisation de type ACO.

En effet, cette technique utilise des traces de phéromones pour guider la recherche d'optimum par les fourmis qui mettent à jour ces traces selon les solutions générées. Ces traces prennent la forme d'une distribution de probabilité sur l'espace de recherche.

Dans le chapitre suivant, nous allons présenter notre démarche adoptée pour la sélection des vues matérialisées dans un entrepôt de données.

Partie 2 : Contribution

Chapitre 4 : Notre approche de sélection des vues

1. Introduction

L'apparition et le développement des phénomènes économiques comme la mondialisation fait que les entreprises évoluent dans un environnement difficile à appréhender. Il en résulte que la prise de décision stratégique ou politique est de plus en plus complexe et en même temps, elle doit intervenir très rapidement pour ne pas laisser le temps aux concurrents de prendre de l'avance. Les nouvelles technologies de l'information permettent de concevoir des systèmes d'information particulièrement performants et novateurs. Avec l'apparition des entrepôts de données les décideurs peuvent désormais accéder à l'information stratégique, ceci permet à l'entreprise d'être plus réactive [Deci]. Dans un tel système l'entrepôt doit répondre efficacement aux requêtes OLAP, et il doit être capable d'offrir les meilleures décisions dans un laps de temps très cour. Dans le but de supporter efficacement les requêtes OLAP, l'entrepôt a besoins de modéliser ses données sous forme multidimensionnelle et de pré-calculer ou de matérialiser quelques une des requêtes OLAP. Les requêtes qui doivent être matérialisées sont nommées les vues matérialisées.

L'idée est de minimiser le coût total de traitement pour toutes les requêtes OLAP possible en sélectionnant un ensemble de vues à matérialiser et en considérant un ensemble de contraintes. Souvent le schéma de l'entrepôt de données est stocké dans ce qui est connus comme un schéma en étoile, ou sous forme multidimensionnelle. Il y a, en fait, une certaine polémique au sujet de la forme la plus utile. La modélisation multidimensionnelle représente les données de l'entrepôt sous forme cubique, connu sous le nom de cube de données.

Le cube de données est un modèle spécifique de l'entrepôt de données. C'est est une base de données où une valeur critique est organisée par plusieurs dimensions, par exemple les ventes des voitures organisées par modèle, couleur, …etc. [GUP97]. Le cube de données a été introduit dans les premiers systèmes décisionnels et formalisé dans [GRAY96], il s'agit d'une modélisation multidimensionnelle des données facilitant l'analyse d'une quantité selon différentes dimensions [GRA00]. Le problème qui se pose est donc de savoir quel est l'ensemble des vues qu'il faut matérialiser, et qui permet de minimiser le temps total de traitement en considérant certaines contraintes. Ces contraintes peuvent être l'espace mémoire occupé par les vues dans l'entrepôt ou le coût de maintenance.

Nous présentons dans ce chapitre, le principe de notre démarche adoptée pour la sélection des vues matérialisées dans le contexte d'un entrepôt de données. Nous commençons par montrer la motivation de notre choix de la méta-heuristique colonies de fourmis. Par la suite, nous détaillons la modélisation de cette nouvelle démarche avec spécifications du modèle de coût utilisé. Et finalement nous présenterons l'algorithme ACO proposé, afin d'optimiser le temps de traitement dans le cube de données.

2. Motivation

Le problème de sélection de vues matérialisées consiste à construire une configuration de vues optimisant le coût d'exécution d'une charge donnée. Cette optimisation peut être réalisée sous certaines contraintes telles que l'espace de stockage alloué aux vues sélectionnées ou une borne supérieure du coût de maintenance des vues sélectionnées.

De nombreux algorithmes ont été développés pour élaborer une solution optimale ou quasi-optimale pour le problème de sélection de vues en réduisant la complexité de sélection. L'application des méta-heuristiques pour optimiser les performances de l'entrepôt de données

est toujours un objectif ciblé par les chercheurs dans ce domaine. En faite, l'utilisation d'une méta-heuristique facilite la résolution des problèmes NP-Complet à savoir le problème de sélection des vues matérialisées PSV.

3. Modélisation

3.1 Le cube de données

Les serveurs OLAP ont été conçus pour s'intégrer dans un environnement client/serveur afin d'en retirer les possibilités offertes. Les décideurs disposant de postes de travail intelligents accèdent à un serveur de base de données multidimensionnelle. Celui-ci contient un hyper cube prédéfini dans lequel doit être stockée la globalité des données. Ce qui nécessite de s'appuyer sur une information pré-packagée et fortement structurée. En outre, des hiérarchies seront définies pour chaque axe d'analyse. Une fois cette structure multidimensionnelle établie, l'outil OLAP propose des méthodes de navigation dans les données, pour aller vers les informations détaillées dans une hiérarchie, ou pour changer d'axe d'analyse [COS98].

3.2 Représentation des données sous forme d'un cube

Le cube de données est une structure multidimensionnelle permettant l'analyse d'information factuelle en les segmentant sur un ensemble d'axes d'analyse, cet axe représente une dimension. Le cube de données est exploré afin de trouver les informations les plus pertinentes, la valeur de chaque cellule dans le cube est une mesure qui correspond aux différentes dimensions. Comme exemple on considère une base de données qui a trois dimensions : produits, fournisseurs et clients. La mesure qui nous intéresse est la vente totale. Donc pour chaque cellule (p, f, c) dans ce cube de données de 3-D, on enregistre les ventes totales d'un produit p, achetée par un fournisseur f, et vendue à un client c [VRU96].

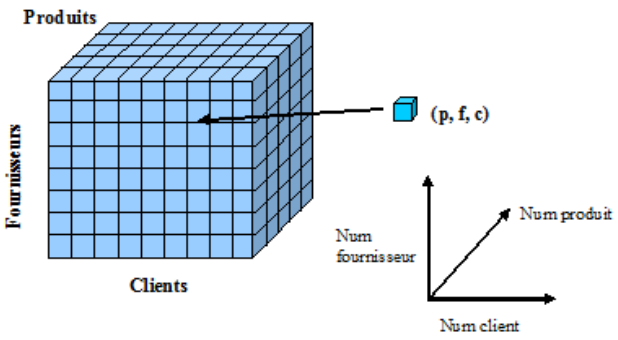

Figure 4-1 Représentation des données sous forme de cube.

3.3 Notion de vues dans un cube

Une vue est définie à partir d'un cube de données par agrégation des quantités selon un sous ensemble des attributs. Pour un cube de dimension k représentant une mesure, il existe 2^k vues avec une fonction d'agrégat. Ces vues peuvent être organisées en un treillis.

Par exemple, le nœud (NumPro, NumFou), présenté dans la figure (4-2) correspond à la vue :

Create view numpron, numfou, count (*), sum (quantité)
From ventecube
Group by numpro, numfou.

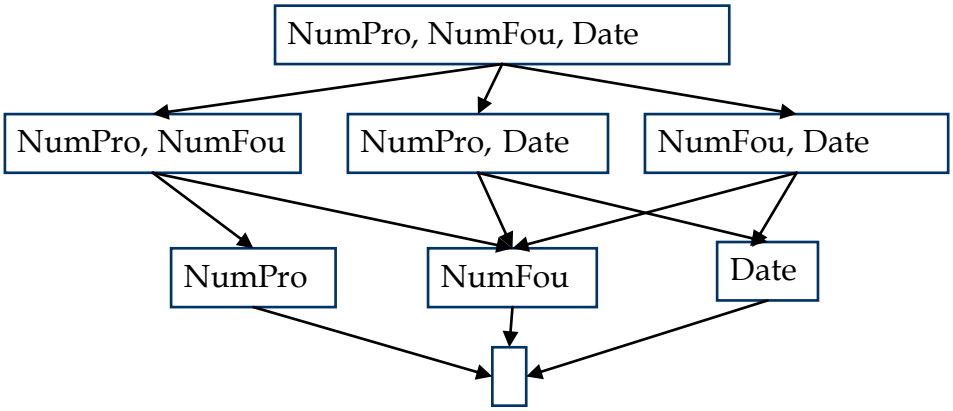

Figure 4-2 Exemple de treillis de vues.

Le nœud singulier vide minimal du treillis représente la somme globale des quantités [GAR00].

3.4 Le treillis de vues

3.4.1 Modèlisation du PSV

Notre algorithme proposé pour résoudre le problème de sélection des vues matérialisées est basé sur l'utilisation d'un treillis de vues. Un treillis de vues matérialisées est un graphe acyclique orienté, $G=(V, E)$ ou V représente l'ensemble des nœuds, et E représente l'ensemble des arêtes dans le graphe G.

Trois poids sont associés à un nœud $v \in$ V :

- r_v : représente le coût initial de traitement.
- f_v : représente la fréquence de la requête.
- g_v : représente la fréquence de la mise à jour.

Et deux poids sont associés à une arête $(v, u) \in$ E :

- $Wq_{u,v}$: représente le coût du traitement de la requête u en utilisant v.
- $Wm_{u,v}$: représente le coût de la mise à jour de u en utilisant v.

Un autre nœud virtuel $v+$ est ajouté au treillis, ce nœud représente l'entrepôt de données multidimensionnel. Pour tous $v_i \in$ V, $vi \leq v+$ tel que $v+ \notin$ V. La taille des données du nœud racine virtuel r+ est la plus grande de toutes les autres tailles des nœuds. Il est important de noter aussi que chaque vue a une taille inférieur à celle de son ancêtre, $r_u \geq r_v$ si $v \leq u$.

3.4.2 Mise en œuvre du treillis

Dans la littérature, il existe plusieurs alternatives d'implémentation possibles d'un treillis de vues à savoir :

1) ***Matérialiser physiquement tout le treillis*** **:** Cette approche donne le meilleur temps de réponse à une requête. Cependant, enregistrer chaque cellule n'est pas une alternative faisable pour les grands cubes de donnes, l'espace consommé devient alors très grand.

2) ***Ne rien matérialiser*** **:** Cette approche est la meilleure en espace consommé mais elle est la plus mauvaise en temps de réponse, car nous sommes obligés d'aller à chaque fois aux données brutes pour répondre à une requête.

3) ***Matérialiser seulement une partie du cube de données*** **:** Dans un cube de données, la valeur de beaucoup de cellules est calculable de ceux d'autres cellules. C'est à dire qu'on peut déduire un ensemble de vues à partir d'autres (une vue contient un ensemble de cellules), c'est pour cela au lieu de matérialiser toutes les vues on matérialise un ensemble de vue qui me permet de répondre à n'importe quelle requête [HRU96].

Le problème qui se pose ici est de savoir quel est le nombre de vues qu'il faut matérialiser et quelles sont les vues qui doivent être matérialisées. En plus l'ensemble de vues matérialisées doit prendre en considération plusieurs critères à savoir :

- La confiuration finale de vues sélectionnées doit minimiser le coût total de traitement des requêtes.
- Elle doit minimiser aussi le coût de mise à jour.
- Il faut sélectionner les vues tel que l'espace consommé et le temps de réponse soient optimisés.

Prendre en considération tous ces critères n'est pas facile pour ne pas dire impossible comme pour le cas de l'espace occupé et du temps de réponse. Si on veut améliorer le temps de réponse, il faut matérialiser le plus de vues possibles, l'idéal est de matérialiser toutes les vues, mais de l'autre coté l'espace devient plus grand avec chaque nouvelle vue matérialisée, et vis versa.

Des algorithmes efficaces pour calculer le cube et ses vues sont nécessaires dans les systèmes d'aide à la décision. Le problème est de trouver des algorithmes permettant d'exploiter le treillis afin d'éviter le calcule répété [GAR00]. Dans ce contexte, notre approche vise à privilégier les vues qui réduit le coût d'exécution de la charge de données, et qui respecte la contrainte d'espace de stockage.

3.5 Fonction objectif

Le problème de la sélection des vues sous la contrainte de l'espace de stockage est formalisé comme suit : sélectionner l'ensemble de vues M qui minimise le coût total du traitement τ(G, M),

$$\tau(G,M) = \sum f_v \times q(v,M) \quad v \in V(G) \qquad (1)$$

Sous la contrainte le l'espace de stockage : Stockage*(M)* $\leq S$ où : Stockage *(M)* est l'espace de stockage de l'ensemble M, et S est l'espace de stockage total alloué aux vues matérialisées.

A chaque nœud du treillis nous définissons :

- Soit *q (u, v)* le coût de traitement de la requête *u* en utilisant la vue matérialisée *v*, *q (u, v*) est la somme des coûts de traitement associés aux arêtes du chemin le plus court de *v* vers *u* plus le coût de traitement initial de *v,* r_v. Si la vue *v* ne peut pas répondre à la requête *u* dans *q (u,v)*, la table de base va être utilisée au lieu de *v*.

- Le coût de traitement peut être différent du coût de maintenance pour une paire de vue.

- Il y a de multiples chemins d'une vue à une requête, nous avons considéré le chemin le plus court, ce chemin représente le chemin le moins coûteux.

Le nombre de vues candidates est généralement proportionnel à la taille de la charge en entrée. Il n'est donc pas pratique de matérialiser toutes les vues candidates car l'espace de stockage est souvent limité. Pour pallier ces limitations, nous utilisons des modèles de coût permettant de ne matérialiser que les vues les plus pertinentes.

Dans la plupart des modèles de coût proposés dans les entrepôts de données, le coût d'une requête *q* est supposé proportionnel au nombre de n-uplets de la vue exploitée par *q*. Nous adoptons le même principe. Notons que ces modèles de coût ont été utilisés par Aouiche et al, et Necir et al.

Soient *ms*(*F*) la taille maximale de la table de fait F, |F| le nombre de n-uplet de F, *Di_ID* la clé primaire de la dimension *Di*, |*Di_ID*| la cardinalité de l'attribut *Di_ID* ou le nombre de n-uplet de la dimension *Di*, et *d* le nombre de dimensions. *ms*(*F*) s'exprime comme suit :

$$ms(F) = \prod_{i=1}^{d} | D_i - ID | \qquad (2)$$

Soient *ms*(*v*) la taille maximum d'une vue *v* ayant les attributs $a_1, a_2, \ldots a_k$ dans sa clause Group by, où *k* est le nombre d'attributs de la clause Group by et |*ai*| la cardinalité de l'attribut *ai*. *ms(v)* est calculée par la formule suivante:

$$ms(v) = \prod_{i=1}^{k} | a_i | \qquad (3)$$

Golfarelli *et al.* [GR98] ont proposé une estimation du nombre de n-uplets d'une vue *v* en utilisant la formule de Yao [YAO77] comme suit :

$$| v | = ms(v) \times \left[1 - \prod_{i=1}^{|F|} \frac{ms(F) \times d - i + 1}{ms(F) - i} \right] \quad (4)$$

$$Où \ d = 1 - \frac{1}{ms(v)}$$

Lorsque le ratio $\frac{\text{ms (F)}}{\text{ms (v)}}$est suffisamment élevé, cette formule est bien approximée par la formule de Cardenas [CAR75]:

$$| v | = ms(v) \times \left(1 - \left(1 - \frac{1}{ms(v)} \right)^{|F|} \right) \quad (5)$$

À partir du nombre de n-uplets de *v*, sa taille en octets, comme est estimé par la formule suivante :

$$taille(v) = |v| \times \sum_{i=1}^{k} taille(d_i) \quad (6)$$

Où *taille* (*di*) dénote la taille, en octets, de la dimension *di* de la vue *v*, et *k* est le nombre de ses dimensions. La taille de chaque dimension peut être obtenue directement à partir des métadonnées de l'entrepôt.

La fonction d'évaluation f d'une solution S est définit comme le bénéfice apporté par la matérialisation des vues sélectionnées:

$$f(S) = \tau(G, \phi) - \tau(G, M) \quad (7)$$

Où M est l'ensemble des vues matérialisées.

4. Approche proposée

4.1Utilisations de l'ACO pour générer la population

Les principes d'ACO sont adaptés au problème d'optimisation binaire de la façon suivante : Nous construisons un graphe où chaque sommet correspond à la position d'un bit et où les arcs correspondent au choix de la valeur du bit. La figure (4-3) (a) représente le graphe contenant les différents sommets qu'une fourmi doit parcourir pour construire une solution.

La fourmi part du premier sommet sur la gauche et choisit un arc, soit « 1 » ou « 0 », pour atteindre le sommet suivant. La décision de choisir l'arc « 1 » ou l'arc « 0 » suit une distribution de probabilité que l'on appelle trace de phéromones dans ACO mais qui peut être ramenée à une seule valeur correspondant à la probabilité de suivre l'arc « 1 ».

Notons par 0*i* et 1*i* les deux arcs correspondant à la position *i* de la chaîne de bits. Les quantités de phéromones de chaque arc sont $\tau 0i$ et $\tau 1i$. Ces deux valeurs réelles peuvent être utilisées pour définir une unique valeur *pi*, la probabilité de générer un « 1 » :

$$P_i = \frac{\tau_{1i}}{\tau_{1i} + \tau_{0i}}$$

La figure (4-3).b illustre la solution *s* = 010 . . . 00 générée par une fourmi.

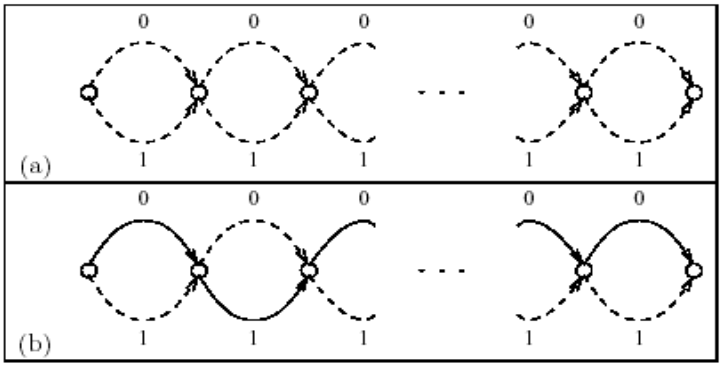

Figure 4-3 Adaptation d'ACO au problème d'optimisation binaire.

Le choix des *l* bits est modélisé par le choix d'un arc « 0 » ou « 1 » entre les *l* + 1 sommets. Initialement, dans ACO, chaque arc 0*i* et 1*i* (i ∈ *{1, . . ., l}*) a une quantité de phéromone τ_{0i} et τ_{1i} fixée à une valeur positive τ_0. Il est assez évident que cette modélisation de la recherche

d'une chaîne binaire sous la forme d'un graphe aussi peu élaboré représente une simplification assez forte de ce que les algorithmes de type ACO étaient habitués à traiter : pour un problème de voyageur de commerce à *l* villes, à chaque sommet, la fourmi doit choisir parmi *l-k* arcs (*k* représentant le nombre de villes déjà explorées) alors que pour cette modélisation, elle ne possède à chaque sommet qu'une alternative entre deux chemins [MON00].

4.2Description de l'approche

Les problèmes de sélection des vues matérialisées dans un entrepôt de données sont très appropriés pour être adapté au développement méta-heuristique. D'où l'idée de notre approche qui consiste à sélectionnées les vues les plus pertinentes à partir de l'ensemble de toutes les vues possible représentée dans un treillis de vues, toute en utilisant les concepts de méta-heuristique et les règles de probabilité indiquées par le comportement de système de fourmi.

L'algorithme ***ACO_Select_Vue*** proposé permet de générer les solutions les plus bénéfiques, qui respectent la contrainte d'espace de stockage alloué aux vues. L'algorithme est basé sur les techniques de colonie de fourmi et utilise un treillis de vues afin de représenter les vues matérialisées. Avant de décrire notre approche de sélection des vues, quelques définitions s'imposent.

- ***La solution du problème :*** Une solution trouvée par la fourmi est l'ensemble des vues qui satisfait la contrainte d'espace de stockage.

- ***L'espace de recherche :*** C'est l'ensemble initial de toutes les vues représentées par un treillis.

- ***La fonction objective:*** La fonction objective est une mesure de performance de la solution. Dans notre cas, cette fonction évalue le bénéfice apporté par la matérialisation des vues.

- ***Les fourmis artificielles :*** Ce sont des agents artificiels qui ont pour rôle de construire des solutions de bonne qualité.

- ***La table de phéromone :*** Des solutions de bonnes qualités sont obtenues par une communication indirecte des fourmis artificielles en lisant et écrivant des informations dans une structure appelée « table de phéromone ». Dans notre cas, cette table est une matrice à deux dimensions, sa première dimension désigne les vues candidates, et sa deuxième dimension désigne les valeurs logiques qu'ils peuvent prendre.

- L***es règles de décisions :*** les règles de décision probabilistes sont des informations heuristiques utilisées pour diriger la construction des solutions des fourmis. Ces information sont calculées au cours de traitement et permettant de donner la probabilité d'exploiter les caractéristiques du problème.

Un algorithme général nommé BPA (à partir des initiales de BSC, PBIL et ACO) définit la façon général dont fonctionnent les algorithmes qui se base sur la fréquence d'apparition des gènes pour résoudre le problème d'optimisation binaire. Avant de donner son corps nous introduisons les définitions suivantes :

Considérant le problème d'optimisation binaire standard qui consiste à trouver dans un espace de recherche $\dot{S} = \{0, 1\}^l$ le minimum d'une fonction d'évaluation *f*. La valeur du bit *i* de la chaîne S sera notée *s* (*i*).

- $V = (p_1, \ldots, p_l)$, avec $pi \in [0, 1]$, représente le vecteur de probabilité qui sera utilisé pour générer des points de *S*. *pi* représente la probabilité de générer un « 1 »;

- $S = (s_1, \ldots, s_n)$, avec $s_i \in \dot{S}$, qui représente les *n* chaînes binaires qui seront générées à chaque cycle. *S* est considérée comme la population dans les algorithmes évolutionnaires [Mon00].

Algorithme BPA

(1) **Initialisation** de $V=(p_1,\ldots,p_l)$ en générale à (0,5,…0,5)
(2) **Tant que** la condition de terminaison n'est pas verifée **faire**
(3) Générer $p=(S_1,\ldots,S_n)$ en utilisant V
(4) Evaluer $f(s_1),\ldots,f(S_n)$
(5) Mettre à jour V selon $(S_1,\ldots,S_n)$ et $f(s_1),\ldots,f(S_n)$
(6) Fin tant que
(7) **Retourner** S+ la meilleur solution trouvée

Figure 4-4 L'algorithme BPA [Mon00].

L'algorithme *ACO_Select_vue* proposé comprend les étapes suivantes :

1. Initialiser les valeurs de phéromone, et du vecteur V.
2. Pour chaque fourmi
   A. Démarrer à partir d'une solution initiale S générée aléatoirement.
   B. Construction des solutions : qui se fait comme suit:
      - Transformation de la solution S en changeant les termes suivants des règles de décision de probabilité.
      - L'amélioration de la solution trouvée S'
   C. Appliquer la mise à jour locale de la phéromone.
3. Appliquer la mise à jour offline de la meilleure solution trouvée pour toutes les itérations.

Ces opérations sont itérées jusqu'à l'obtention d'une solution de très bonne qualité ou jusqu'à ce que un certain temps soit atteint.

Algorithme1 : Description de l'algorithme ACO_Select_Vue proposé.

Début
Initialiser la table de phéromone
Initialisation du vecteur $V = (p_1, \ldots, p_l)$ à 0,5.
Initialisation de la matrice de la phéromone à 0,5 ;
Pout i=1 = max_iter faire
Pour chaque fourmi
Faire
- Générer une solution S aléatoire
- S'= Améliorer (S)
- Si stockage (S') ≤ espace_vue alors
- Si f(S') < f(S) alors

- best= S'
- mettre à jour la phéromone en ligne pour la solution S'
- mettre à jour le vecteur V

Fsi
Fsi
Fait
- Si f (best_sol) < f (best) alors best_sol= best.
- Appliquer la mise à jour offline de la phéromone.
- Mettre à jour le vecteur V

Fait.
Fin

4.3 Algorithme de construction et de transformation de la solution

4.3.1 Algorithme de construction

Dans cette étape une première solution est généré (individus) selon le vecteur de probabilité. Cette étape permet de mettre à jour la matrice de phéromone selon les solutions trouvées. Notons que les solutions trouvées doivent respecter la contrainte de l'espace de stockage.

L'idée est de générer une probabilité *u*, pour chaque bit généré pour la solution *i ;*

- Si u < 1 - *q*0 alors nous diversifions avec la probabilité 1 – *q*0 : le bit *j* prend la valeur « 1 » avec la probabilité *pi*.
- Sinon nous intensifions avec la probabilité *q*0 : le bit *j* prend la valeur suivante :

$$S_i = \begin{cases} 1, & si\ p_i > 0 \\ 0, & sinon \end{cases}$$

*q*0 est une probabilité données au début de l'algorithme

L'algorithme permettant de générer la solution est défini comme suit :

Algorithme2 : Génération d'une première solution.

Paramètre : q_0, α ; k ∈ {0,1} // k représente l'arc « 0 » ou l'arc « 1 ».
Début
Répéter
Début
Pour j = 1 à *n* // n st le nombre d'individus.
Pour i = 1 à *l*
Générer une probabilité *u* ∈ [0, 1]
Si *u* < (1 - q_0) alors // diversifier
Générer une probabilité *w* ∈ [0, 1]
Si *w* < *pi* alors
Le bit *i* de la chaîne Sj prend la valeur « 1 » donc k =1.
Sinon
Le bit *i* de la chaîne Sj prend la valeur « 0 » donc k =0.
Finsi
// Mise à jour local
phero [k][i] ← (1 - α) * phero [k] [i] ;
Sinon // intensifier
Si (pi > 0,5) alors

```
        Le bit i de la chaîne Sj prend la valeur « 1 » donc k =1;
      Sinon  il prend la valeur « 0 » k = 0;
      Finsi
     // Mise à jour local
     phero [k][i] ← (1 - α) * phero  [k] [i] + α τ0 ;
     Finsi
   Finpour ;
   Finpour ;
FIN
Tanque (Stockage () > espace_vue)
 Fin.
```

4.4 Amélioration de la solution

Cette fonction vise à améliorer une solution en essayant maximiser la valeur de sa fonction objective. En lui appliquant une heuristique de recherche locale, qui s'articule autour d'un principe simple : partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante mais tout en respectant la contrainte de l'espace de stockage.

Algorithme3 : Algorithme d'amélioration d'une solution.

Début

Sortie= faux
Tant que (sorti= faux et i<nb_vue)
Faire

- Y=S avec V(i) inversé
- Si Stockage(Y) <= espace_vue alors
- Si f(Y) > f(S) alors
  - S :=Y
  - Sortie= true
- Sinon i :=i+1
- Fsi.
- Fsi

Fait

Fin

4.5 Règle de mise à jour de la phéromone

La mise à jour des phéromones est effectuée une fois que toutes les fourmis sont passées par toutes les vues. Cette mise à jour se fait par une simulation d'évaporation de la phéromone pour toutes les vues, suivis d'un ajout pour les vues à la solution en question. Nous distinguons deux types de mise à jour à savoir : la mise à jour Online et la mise à jour offline.

- **La mise à jour online :** Cette mise à jour locale a pour but de modifier très légèrement la quantité de phéromones sur l'arc choisi par une fourmi afin de pousser les autres à explorer les autres arcs, cela afin d'éviter que toutes les fourmis se suivent. Pour la mise à jour online l'ajout de phéromone suit la règle :

$$phero[v_i, j] \leftarrow (1 - \alpha) phero[v_i, j].$$

// Mise à jour de la matrice de phéromone phero.
Pour k = 0 à 1
Pour i = 1 à l faire
Si S[i] = k alors
$phero[i_i, k] \leftarrow (1-\alpha) phero[i, k]$
Finsi
Finpour ;
Finpour ;

// Mise à jour du vecteur V
Pour i = 1 à l faire
V[i] = phero[1][i] / phero [1][i] + phero [0][i] ;
Finpour ;

- **La mise à jour offline :** Alors que lors de la mise à jour offline est appliqué par la règle (1). Où $\alpha \in [0, 1]$ est un paramètre représentant l'évaporation des phéromones.

$$phero[v_i, j] \leftarrow (1-\alpha) phero[v_i, j] + \alpha \times \frac{1}{1+f(S')} \qquad (1)$$

S représente la meilleur solution générer depuis le début d l'algorithme par toute les fourmis, f(s) désigne son coût de traitement e t $\frac{1}{1+f(S)}$ représente le taux de phéromone à verser sur les arcs.

// Mise à jour de la matrice de phéromone phero.
Pour k = 0 à 1
Pour i = 1 à l faire
Si s++ [i] = k alors
$phero[i_i, k] \leftarrow (1-\alpha) phero[i, k] + \alpha \times \frac{1}{1+f(S)}$
Finsi
Finpour ;
Finpour ;

// Mise à jour du vecteur V

Pour i = 1 à l faire
V[i] = phero[1][i] / phero [1][i] + phero [0][i] ;
Finpour ;

Notons que les traces de phéromones sont donc équivalentes au vecteur de probabilité V utilisé par note algorithme.

5. Conclusion

Nous avons étudié dans ce travail la problématique de la sélection des vues matérialisées dans les entrepôts de données. Nous nous sommes intéressés particulièrement à la sélection des vues en utilisant les colonies de fourmis.

Dans ce chapitre, nous avons proposé une nouvelle démarche de sélection automatique des vues matérialisées dans les entrepôts de données. Cette démarche utilise les systèmes de colonie de fourmi et un treillis de données afin de sélectionner une configuration de vues matérialisées qui réduit le coût d'exécution de la charge de requêtes. À l'aide de modèles de coût, nous ne conservons que les vues les plus avantageuses. Ces modèles estiment le coût d'accès aux données via les vues matérialisées sélectionnées, ainsi que l'espace de stockage de ces vues. L'algorithme de sélection proposé est indépendant du SQBD utilisé, et il n'utilise aucun dialogue avec l'optimisateur de requêtes.

Chapitre 5: Notre approche de sélection d’index

1. Introduction

Les entrepôts de données sont caractérisés par une volumétrie de données très importante. L'interrogation de ces données se fait généralement par des requêtes d'analyse très complexe de type OLAP, comportant beaucoup d'opérations de jointures et d'agrégations. Ceci implique un temps de réponse important. Réduire ce temps de réponse nécessite l'utilisation des techniques d'optimisation à savoir l'indexation.

L'indexation est l'une des techniques les plus efficaces pour améliorer le temps de réponse des requêtes décisionnelles. Cependant, la sélection des index dans les entrepôts de données est l'une des tâches les plus importantes à réaliser par un administrateur afin d'optimiser les performances du système en minimisant le temps d'accès aux données.

Dans ce contexte, nous nous somme intéressées dans notre travail à la sélection des index bitmap de jointure, car ce type d'index est très utilisé dans le contexte des entrepôts de données. En plus, il est performant pour les opérations de jointures et d'agrégations caractérisant les requêtes utilisées dans les entrepôts de données.

La figure (5-1) illustre l'architecture générale de notre approche. Ce dernier est accomplit à travers les différentes étapes suivantes :

- Extraction des attributs indexables.
- Construction du contexte d'extraction.
- Application de l'algorithme de selection sous la contraintes d'espace de stockage.

Nous présentons, dans ce chapitre, le principe de notre démarche adopté pour la sélection des index bitmap de jointures dans un entrepôt de données. Nous commençons par montrer la motivation de notre choix de type d'index, et de la méta-heuristique. Par la suite, nous présenterons l'algorithme ACO proposé afin d'optimiser le coût de traitement de la charge de requêtes sous la contrainte d'espace de stockage.

2. Motivation

Utiliser les mêmes méthodes d'indexation employée dans les systèmes OLTP, dans les entrepôts de données permet de générer des index très volumineux avec un grand nombre de niveau (les B-arbres par exemple). De ce fait, les index les plus adaptés aux entrepôts de données sont les index bitmap de jointure car ils présentent plusieurs avantages à savoir :

- Un faible coût de stockage car l'index est modèlisé par une matrice binaire
- Certaines opérations (opérations de comptage, opérations AND, OR) sont très rapides, elles ne nécessitent par l'accès aux données est sont effectuées en mémoire centrale
- Ces index sont peu performants dans le cas de mise à jours, cela dit cette opération est rare dans le contexte d'entrepôt de données.
- Ces index sont plus efficaces que les B-arbres. En effet, les B-arbres sont largement répandus dans les SGBD, mais ils sont limités lorsqu'il s'agit d'indexer des données volumineuses et des attributs à faible cardinalité [BEL08]

La sélection des index dans un entrepôt de données est un problème NP-Complet. La difficulté de sélection peut être résolue par des méthodes exactes ou des approches

approchées. Les limitations des méthodes exactes ont obligé les chercheurs à utiliser des approches approximatives appelées « les méta-heuristiques ». Parmi les méta-heuristiques les plus récentes, nous trouvons celle basée sur les fourmis artificielles, qui a été appliqué pour résoudre plusieurs problèmes NP-Complets. Notre contribution à travers cette thèse, consistera à appliquer l'algorithme de la colonie des fourmis pour résoudre le problème NP-Complet de la sélection des index dans le contexte des entrepôts de données.

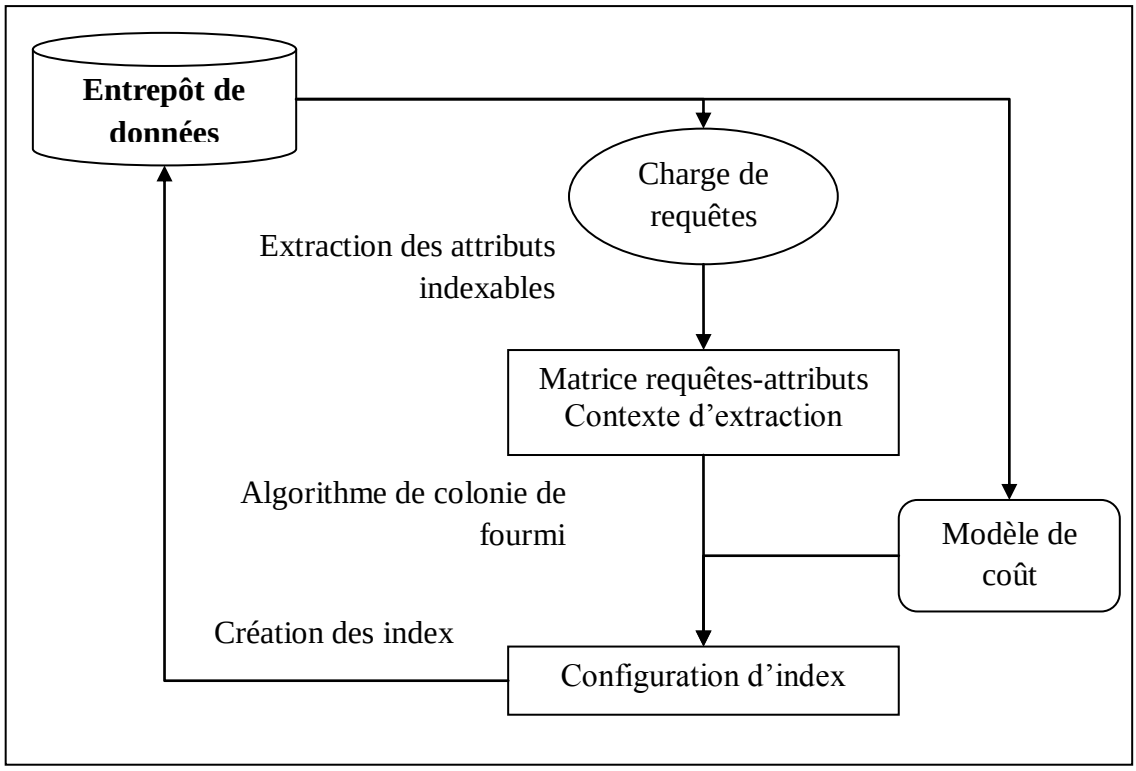

Figure 5-1 Architecture générale.

3. Extraction des attributs indexables

Dans notre approche, les requêtes représentent les transactions et les attributs impliqués dans chaque requête. Il faut donc, identifier tous les attributs pouvant servir à la construction d'un index utile. Pour chaque requêtes de la charge, les attributs indexables sont ceux qui figurent dans l'une des clauses suivantes :

- La clause WHERE d'une requête de sélection SELECT, de modification UPDATE, ou de suppression DELETE.
- La clause GROUP BY et HAVING.
- La clause ORDER BY.

Pour extraire les attributs représentatifs, notre outil procède de la manière suivante : il supprime tous les attributs de sélection d'un SELECT et les noms des tables de la clause FROM. Ensuite il enlève tous les mots clés SQL (WHERE, GROUP BY, ORDER BY, LIKE, IN…), toutes les constantes numériques et chaînes de caractères servant comme opérande et tous les éléments d'énumérations. A la fin, il ne reste que les attributs qui nous intéressent. Notre outil génère des résultats qui vont alimenter la matrice « requêtes-attributs ».

La matrice « requêtes-attributs » représente le contexte de recherche des attributs indexables. La charge de requêtes est un ensemble de requêtes. Une partie de la charge de requêtes qu'on utilisera comme entrée pour extraire la matrice « requêtes-attributs ».

Exemple : concéderons les deux requêtes suivantes extraites de la charge avec laquelle on a fait les expérimentations :

Q1:
```
Select sales.prod_id,avg (amount_sold)
From Sales, promotions
Where sales.promo_id = promotion.promo_id
and promotion.promo_category= 'newspaper'
```

Q2:
```
Select sales.prod_id, avg(amount_sold)
from sales, products, promotions,times
where sales.prod_id= products.prod_id
and sales.promo_id= promotion.promo_id
and sales.time_id= times.time_id
and promotion.promo_category='newspaper'
and times.fiscal_year in ('1998','2000')
group by sales.prod_id, products.prod_category;
```

L'analyse de la requête Q1 donne comme attributs représentatifs les champs suivants: <sales.promo_id>, <promotion.promo_id>, <promotion.promo_category>.

4. Adaptation du problème de sélection des index au problème de recherche d'information

4.1 Description de notre approche

Les problèmes de sélection des index dans un entrepôt de données sont très appropriés pour être adaptés au développement méta-heuristique. En fait, l'idée principale de l'approche consiste à construire des index bitmap de jointure BJIs à partir de l'ensemble de tout le BJIs possible en utilisant les concepts d'heuristique et les règles probabilistes indiquées par le comportement de système de fourmi.

Avant d'introduire notre approche de résolution du problème de sélection des index, nous commençons par donner les définitions suivantes :

- **La solution du problème :** Pour un PSI, une solution trouvée par la fourmi est un index qui satisfait la contrainte d'espace de stockage.

- **L'espace de recherche :** C'est l'ensemble des index bitmap de jointure candidats.

- **La fonction objective ou la similarité :** La fonction objective est une mesure de performance de la solution. Dans le cas d'un PSI, cette fonction évalue la correspondance entre le temps d'exécution des requêtes et l'espace disque utilisé pour stockées les index.

- **Les fourmis artificielles :** Ce sont des agents artificiels qui ont pour rôle de construire des solutions de bonne qualité.

- **La table de phéromone :** Des solutions de bonnes qualités sont obtenues par une communication indirecte des fourmis artificielles en lisant et écrivant des informations dans une structure appelée « table de phéromone ». Dans notre cas, cette table est une matrice à deux dimensions, sa première dimension désigne les vues candidates, et sa deuxième dimension désigne les valeurs logiques qu'ils peuvent prendre.

- **Les règles de décisions :** les règles de décision probabiliste sont des informations heuristiques utilisées pour diriger la construction des solutions des fourmis. Ces information sont calculées au cours de traitement et permettant de donner la probabilité d'exploiter les caractéristiques du problème. Pour un PSI, nous utilisant la règle de décision probabiliste suivante :

$$p(i=0) = \frac{\text{phero[i, 0]}}{\text{phero[i, 0]} + \text{phero[i, 1]}}$$

Qui correspond à la probabilité pour qu'un index i prenne la valeur logique 0. Quant à la probabilité pour que ce terme prenne la valeur 1 elle est déduite à partir de la formule suivante : $p(i=1) = 1 - p(i=0)$.

4.2 La fonction fitness

La fonction fitness que nous proposons est inspirée de la technologie de recherche d'information (RI). En fait, dans ce contexte, des documents sont recherchés des collectes de données quand ils sont semblables à la requête d'utilisateur, c.-à-d., quand ils partagent une partie des mêmes mots-clés avec la requête. Nous constatons l'existence d'une certaine analogie entre les deux paradigmes. Le tableau suivant montre la correspondance entre les concepts des deux problèmes.

Recherche d'information	Data mining
Terme	Attribut
Document = ensemble de termes	BJIs= ensemble d'attributs
Collection de documents	L'ensemble des BJIs candidate
Requêtes= ensemble de termes	Requêtes= ensemble d'attributs
Similarité entre le document et la requête	Similarité entre les requêtes et le BJI

Table- Recherche documentaire contre l'exploitation de donnée.

La similarité entre les index et les requêtes est définis comme suit : Soient q (waq1,waq2,…,waqn) la représentation de chaque requête de la charge, et bji(wabji1,wabji2,….,wabjin) la représentation des index bitmap de jointures. Les requêtes de la charge de données et les BJI sont définis comme un ensemble d'attributs. Notons que waqn (wabjin) correspond au poids de l'attribut ai dans la requête q (dans l'index BJI) et est exprimé par le produit aif *iqf (aif * abjif).

aif représente la fréquence de l'attribut ai dans la requête q (BJI) et iqf (abjif) représente la fréquence inversée de la requête (BJI). Les expressions aif et iqf sont calculé par les formules suivantes :

$$\text{aif} = \log(\, freq(a_i, q) + 1)$$
$$\text{iqf} = (1/k)$$

- **freq(ai,q)** représente le nombre d'occurrence de l'attribut ai dans la requête q (bji),
- **k(m)** represente le nomnre de requête de la charge (BJIs).

La fonction de similarité f(bji,q) peut être calculée en utilisant l'une des formules suivantes:

$$\text{f(bji, q)} = \sum_i (waq_i * wabji_i\,)\ Internal\ product$$

$$\text{f(bji, q)} = \sum_i (waq_i * wabji_i\,) /(\sum_i (waq_i)^2 * \sum_t (wabji_i)^2)^{\frac{1}{2}} \quad (cosine)$$

$$\text{f(bji, q)} = 2\sum_i (waq_i * wabji_i\,) /(\sum_i (waq_i)^2 + \sum_i (wabji_i)^2) \quad (Dice)$$

$$\text{f(bji, q)} = \sum_i (waq_i * wabji_i\,) /(\sum_i (waq_i)^2 + \sum_i (wabji_i)^2 - \sum_i (waq_i * wabji_i)\,) \quad (Jaccard)$$

Les quatre formules définies au dessus sont des formules normalisées à l'exception de la première. La formule de cosinus est la plus adoptée dans la fonction fitness des algorithmes de colonie de fourmi.

5. Algorithme de sélection des index

Dans cette solution, nous présentons un algorithme développé pour adapter le system colonie de fourmis ACS au problème de sélection des index bitmap de jointures. L'algorithme proposé pour cette stratégie comprend les étapes suivantes :

- Initialiser les valeurs de phéromone dans la table 'Phero'.
- Pour chaque fourmi

- Démarrer à partir d'une solution initiale S générée aléatoirement à partir de l'ensemble des index.
- La construction des solutions : se fait comme suit :
- Transformation de la solution S en changeant les valeurs des index suivant des règles de décision probabiliste.
- L'amélioration de la solution S, en appliquant une heuristique de recherche locale
- Une mise à jour en ligne appliquée à la solution déjà amélioré.
- La mise à jour offline de la meilleure solution trouvée pour toutes les itérations.

Ces opérations sont itérées jusqu'à l'obtention d'une solution de très bonne qualité ou jusqu'à ce que, un certain temps soit atteint. Le déroulement de cet algorithme ne permet d'avoir une seule solution, c'est-à-dire un seul index dans la configuration des index. Il sera répété tant que la contrainte d'espace de stockage et satisfaite. Notons que chaque index est représenté par un vecteur d'attribut. L'algorithme proposé pour cette stratégie est le suivant :

Algorithme 5-1: Algorithme de colonie de fourmis pour la sélection des index
Entrée : table de la phéromone **Sortie :** best **1 :** Initialiser la table de phéromone. **2 :** liste_index = ∅, Initialiser la liste des index sélectionnée à vide **3 :** Pour i :=1 à Max-Iter

4 : Pour k=1 à NbAnts
5 : Faire
Générer une solution initiale S aléatoirement.
S' :=S transformer en fonction de la phéromone ;
S'' :=améliorer(S')
Mettre à jour la phéromone online pour la solution s''
Si f(s'')>f(best) alors best :=s'' ;
6: Fait
Si f(best_sol)>f(best) alors best_sol :=best ;
Appliquer la mise à jour offline de phéromone.
7: Retourner (best)

5.1 Algorithme de génération d'une solution par une fourmi

La première étape de notre algorithme, consiste à construire une solution initiale suivant l'algorithme. L'idée est de générer un nombre aléatoirement puis déterminer sa représentation binaire. En faite la représentation binaire sert à définir l'index en fonctions des attributs. Cette dernière qui va subir des transformations à l'étape suivante.

5.2 Algorithme de transformation d'une solution par une fourmi

La deuxième étape de notre approche consiste à la transformation de la solution trouvée à l'étape précédente. En faite, la solution optimale est construite par la communication indirecte des fourmis artificielles en lisant et écrivant des informations dans table de phéromone. La représentation binaire fournie à l'étape précédente subis des transformations en lui effectuant des changements par 0 ou 1 à l'attribut choisi aléatoirement selon une certaine probabilité donnée par la règle de décision déjà définie.

A la fin, la solution trouvé sera comparé avec les valeurs définis dans la table requêtes-attribut afin de vérifier si la représentation définie un index existant parmi les index candidate ou pas. Les solutions prisent en considération sont celles qui existent parmi les index candidats.

Dans notre cas, toutes les valeurs de phéromone sont initialisées à une valeur fixe faible égale à 0.1. Le choix de cette valeurs du au faite que l'algorithme simule le comportement des fourmis et au qu'au début la fourmi dépose une très faible quantité de phéromone.

Algorithme 5-2: Algorithme de transformation d'une solution par une fourmi

Entrée : table de la phéromone,
Sortie : S
1 : **Pour** i :=1 à Max-Change
2 : Faire
3: Générer un nombre aléatoirement N
4: Représentation binaire du nombre généré.
5: i:=N en binaire.
6: Calculer $p(=0) = \frac{\text{phero [i,0]}}{\text{phero [i,0]+phero [i,1]}}$
7 : Générer un nombre réel aléatoirement R tel que 0≤R≤1
8 : Si R≤P (i=0) alors S(i) :=0 ; //Changement de l'attribut choisi par 0
9 : Sinon s(i)=1 ; // Changement par 1.
10: Fait
11: Retourner (S)

Le nombre de changement effectués à chaque solution est donné par le paramètre empirique Max-changes.

5.3 Algorithme d'amélioration de la solution

L'étape suivante de notre approche, consiste à améliorer la solution en essayant de maximiser la valeur de sa fonction objective, en lui appliquant une heuristique de recherche locale, et qui s'articule autour d'un principe simple : partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante. L'amélioration est faite suivant l'algorithme suivant :

Algorithme 5-3 : Amélioration des solutions
Entrée : S **Sortie :**S' **1 :** Sortie :=faux 2 : Tant que (sortie=faux et i< maxi-terme) **3 : Faire** **4 :** Y = S avec bji inversé **5 :** Si f(Y)>f(S) alors **6 :** S':=Y ; **7 :** insert (liste_index, Y) **8 :** Sortie :=vrai **9 :** Sinon i : i+1 ; **10 :** Fsi **11 : Fait** **12 : Retourner** (S') **13 : Fin** ;

5.4 Règle de mise à jours de la phéromone

Les stratégies de mises à jour de phéromone consistent en simulation de l'évaporation de la phéromone pour touts les index, suivis d'un ajout pour les index de la solution en question. L'évaporation est appliquée selon la règle suivante :

$$phero[v_i,J] \quad = (1-\alpha) * phero[v_i,J]$$

- i=1..N,N étant le nombre de index de la solution et J∈ {0,1}.
- α est un paramètre empirique compris entre 0 et 1.
- S est la solution courante trouvée.

Le renforcement de phéromone dans la mise à jour en ligne suit la règle suivante : $phero[b_{ji},J] \quad = phero\left[b_{ji},J\right] + \alpha * f(S'')$. Alors que dans la mise à jour offline, il est appliqué par la règle : $phero[b_{ji},J] \quad = phero\left[b_{ji},J\right] + \alpha * \frac{\text{f(bestsol)}}{f(best)}$, sachant que :

- Best_sol est la meilleure solution trouvée pour toutes les itérations.
- Best est la meilleure solution d'une itération.

Les valeurs de la fonction f sont ensuite recalculées pour chaque index restant dans Configv, car elles dépendent des index sélectionnés et présents dans Configv. Cela aide à prendre en compte les interactions qui peuvent exister entre les index. Nous répétons ces

itérations jusqu'à ce qu'il n'y ait plus d'amélioration ou que toutes les index aient été sélectionnées où l'espace de stockage est atteint($S \leq 0$).

5.5 Stockage des index

Pour accélérer l'accès aux index candidats, nous adoptons une stratégie de stockage des index. L'idée d'enregistrer les meilleures solutions donné par chaque fourmis dans une liste, au fur et à mesure, on supprime les plus mauvaises solutions de la liste pour libérer l'espace de stockage afin d'ajouter à la liste une nouvelle solution. Chaque index est converti en nombre décimale qui représente l'entrée à l'index pour obtenir l'adresse de l'index. Notant que chaque solution trouvée par une fourmi désigne un index, et qu'il faux prendre en considération la taille des index.

L'espace de stockages est réservé aux départs à tous les index de l'entrepôt de données. Par la suite chaque fourmis qui trouve une solution fait appel à la procédure de stockage d'index afin d'allouer l'espace disque nécessaire et mettre à jour l'espace alloué par la suite.

5.5.1 Estimation du coût de la charge

La création de tous les index n'est pas faisable dans le cas où le nombre d'index est relativement grand. Cette infaisabilité est due à certaines contraintes à savoir la limitation des index par table ou la contrainte d'espace de stockage alloués aux index. De ce fait, il faudra sélectionné parmi ces candidates ceux qui respectent les contraintes imposées tout en optimisant le mieux le coût de la charge.

Afin de pouvoir sélectionner les index les plus avantageux à une charge de requêtes, nous avons utilisé un modèle de coût permettant l'évaluation des coûts des index en terme d'espace de stockage et en terme d'accès aux données lors du traitement des requêtes. Notons que ces modèles de coût ont été utilisés par Aouiche et al, et Necir et al.

5.5.2 Coût de stockage des index

Pour un index bitmap, il faut créer un bitmap pour chaque valeur distinct de l'attribut indexé. Ainsi, chaque bitmap contient autant de bit que de N-Uplet de la table à laquelle il appartient. La taille d'un index bitmap construit sur un attribut A appartenant à une table T est donnée par la formule suivante :

$$S = \frac{|\mathrm{A}| \times |\mathrm{T}|}{2}$$

|A| : cardinalité de l'attribut A. c'est le nombre d'attribut distinct de ce dernier.
|T| : cardinalité de la table, le nombre de n-uplet de cette dernière.

Le temps de construction d'un index bitmap dépend de la cardinalité de l'attribut indexé ainsi que la table à laquelle il appartient ce qui donne une complexité de O (|A||T|).

5.5.3 Coût d'accès aux données

L'accès aux données peut se faire en utilisant un index ou pas. Dans le cas ou aucun index ne correspond a une requêtes données, nous supposons que les jointures de cette dernière sont réalisées par hachage. Le coût d'une jointure par hachage entre deux table T1 et T2 est données par la formule suivante :

$$C_{hachage} = 3(p_{t1} + p_{t2}) \quad (1)$$

p_{t1} et p_{t2} représentent le nombre de pages disque nécessaires pour stocker les tables T1 et T2 respectivement, la tailles d'un N-uplet et d'une page disque sont en octet :

$$p_T = \frac{\text{la taille d'un N} - \text{uplet} \times |T|}{S_p}$$

Dans le cas ou un index est utilisé pour le calcul d'une requête, l'accès aux données à travers cet index peut s'effectuer directement en deux étapes à savoir : Coût de parcours de l'index et Coût de lecture des données.

Notons que ce modèle permet l'évaluation de coût de jointure seulement en accédant directement aux données sans prendre en compte la structure physique dont ces index sont implémentés. Ce modèle [AOU02] suppose que les données sont uniformément distribuées. Cette hypothèse est raisonnable et souvent adoptées dans l'élaboration des modèles de coût.

- **Coût de parcours :** au pire des cas tous les bitmaps d'un attribut sont parcourus pour rechercher le bitmap correspondant à une valeur donnée de ce dernièr. Lorsqu'un attribut est lié *d* fois par l'operateur OR dans une requêtes ou la cardinalité d'une clause IN correspondante à cet attribut est égal à d, le coût de parcour doit être multiplié par *d* ce dernier est donné par la formule suivante :

$$C_{parcours} = d\frac{|A||T|}{8S_p}$$

- **Coût de lecture :** le coût de lecture des données à partir du disque, il est donné par la formule suivante:

$$C_{lecture} = P_F\left(1 - e^{\frac{N_r}{P_F}}\right)$$

Avec :

P_F : Le nombre de page disque necessaire pour stocker la table de faits.

N_r: Le nombre de n-uplet lus correspondant à une valeur A. il est donné par la formule suivante : d étant le nombre de bitmap :

$$N_r = d\frac{|F|}{|A|}$$

Le coût d'un index est donc la somme du coût de parcours et du coût de lecture :

$$C_{index} = C_{parcours} + C_{lecture} = P_F\left(1 - e^{\frac{N_r}{P_F}}\right) + d\frac{|F|}{|A|} \quad (2)$$

Lors de l'évaluation des requêtes, trois cas possibles peuvent être distingue selon la couverture de cette dernière par un index ou pas :

- **Aucune couverture(No matching):** Dans le cas ou la requête n'a pas de jointures couvertes pas aucun index, son coût est évalué par la formule (1). Le coût de la requête est égal à la somme des coûts par hachage de chaque jointure.

$$C_{requête} = \sum C_{hachage}$$

- **Couverture partielle(Partial matching):**Le deuxième cas concerne la couverture partielle d'une requête par index. C'est-à-dire qu'il y a quelque jointures qui peuvent être évaluées par index est d'autres non. Dans ce cas, la formule (2) est utilisée pour évaluer les jointures couvertes par l'index. Les autres sont évaluées par la formule (1). Le coût de la requête est donc égal à la somme des coûts par hachage pour les requêtes non couvertes plus le coût de l'index.

$$C_{requête} = C_{index} + \sum C_{hachage}$$

- **Couverture totale (Total matching):**Dans le dernier cas, l'index couvre totalement les jointures de la requête. Ces derniers sont donc évaluer par la formule (2). Le coût de la requête dans ce cas est égal au coût de l'index.

$$C_{requête} = C_{index}$$

Pour pouvoir calculer le coût de la charge en présence des index, chaque requête est évaluée par chaque index individuellement ainsi que sans aucun index. Le minimum des résultats présenté par chaque requête est affecté au coût de cette dernière. Le coût global de la charge avec index est égal à la somme du coût de chaque requête.

6. Conclusion

Nous avons étudié dans ce travail la problématique de la sélection d'index dans les entrepôts de données. Nous nous sommes intéressés particulièrement à la sélection des index Bitmap de jointure en utilisant les colonies de fourmis. Ce type d'index a une est très petite taille comparée à un B-arbre construit sur le même attribut. Il est très performant pour les opérations de jointure et d'agrégation caractérisant les requêtes utilisées dans les entrepôts de données. Certaines requêtes peuvent alors être exécutées très efficacement, parfois sans même recourir à la table contenant les données.

Dans cette partie, nous avons proposé une stratégie de sélection automatique d'index bitmap de jointure dans les entrepôts de données. Cette stratégie permet d'extraire dans un premier temps les attributs indexables à partir d'une charge de données afin de générer des index avec une similarité importante, et qui réduits de plus en plus le coût d'exécution des requêtes en prenant en considération l'espace de stockage alloué aux index. L'algorithme consiste à construire des index en utilisant les colonies de fourmis. Notre démarche se caractérise par les avantages suivants :

- Indépendants du SQBD utilisé.
- Aucun dialogue avec l'optimisateur de requêtes.
- Elimination des attributs clés de l'ensemble de départ permet éviter la génération des attributs fréquents mais inutiles.

Chapitre 6: Etude expérimentale

1. Introduction

Après avoir présenté les démarches adoptées pour la sélection des vues matérialisée et des index binaires de jointure dans les chapitres précédents, nos allons dans ce chapitre, présenté l'implémentation que nous avons utilisée et les tests que nous avons effectués.

2. Schéma de l'entrepôt de données

Le schème en étoile de l'entrepôt de données utilisé est donné par la figure (6-1). Cet entrepôt de données est composé d'une table de faits Sales et de cinq tables de dimensions *Customers, Products, Promotions, Times* et *Channels*. Pour chaque table, nous donnons :

- la taille de la table en Méga octets (Mo)
- la cardinalité de la table (Nombre de n-uplets)
- le nombre de pages de la table
- la taille d'un seul n-uplet de la table en octets

Pour calculer les différents coûts, nous avons fixé les valeurs des paramètres *taille d'une page disque (PS) à 8 Ko* et *taille du pointeur d'une page (π) à 4 Ko*. Ces valeurs sont celles indiquées dans le fichier de configuration d'Oracle.

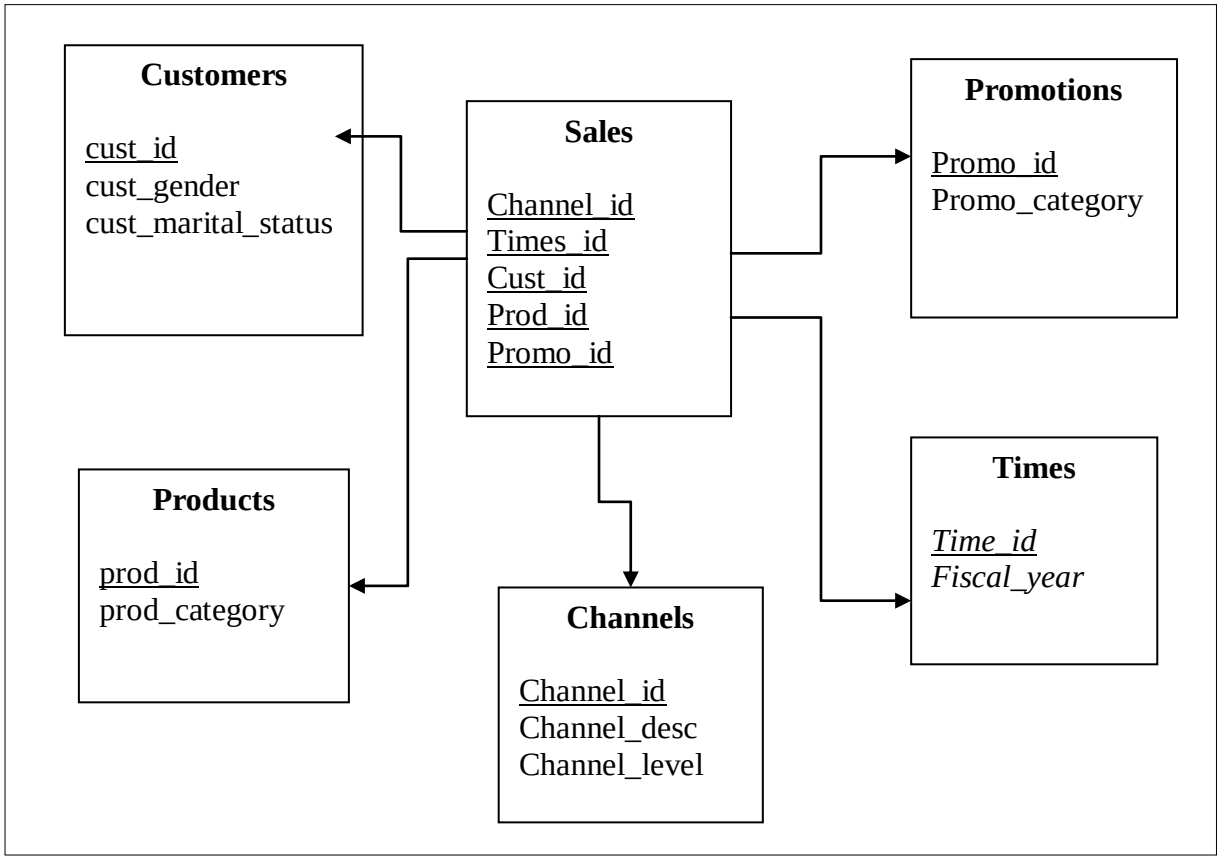

Figure 6-1 Schème en étoile de l'entrepôt de données expérimental.

Nous avons utilisé une charge contenant 40 requêtes, de différentes formes, pour expérimenter notre application aussi bien dans la dimension index que dans la dimension vues. Cette charge est donnée en Annexe de ce document.

Nous avons déroulé nos expérimentations avec et sans considération de la contrainte d'espace de stockage. Dans les deux cas, nous avons mesuré l'espace de stockage occupé par les vues matérialisées et celui occupé par les index, le coût d'exécution et le coût de la charge de données.

3. Réglage des paramètres empiriques de l'algorithme de sélection des vues matérialisées

Pour pouvoir fixer les paramètres empiriques utilisés, plusieurs tests ont été effectués, les résultats obtenus sont schématisés à l'aide des graphes suivants :

3.1 Taille de la population

Ce paramètre représente le nombre d'individus utilisé pour sélectionner la solution initiale. Nous remarquons, qu'à partir de 30 individus, les solutions obtenues sont de bonne qualité. Si nombre d'individu est >50 il n'ya pas une grande amélioration de solution en terme de coût de traitement. Par conséquence, nous avons fixé le paramètre nombre d'individus à 40.

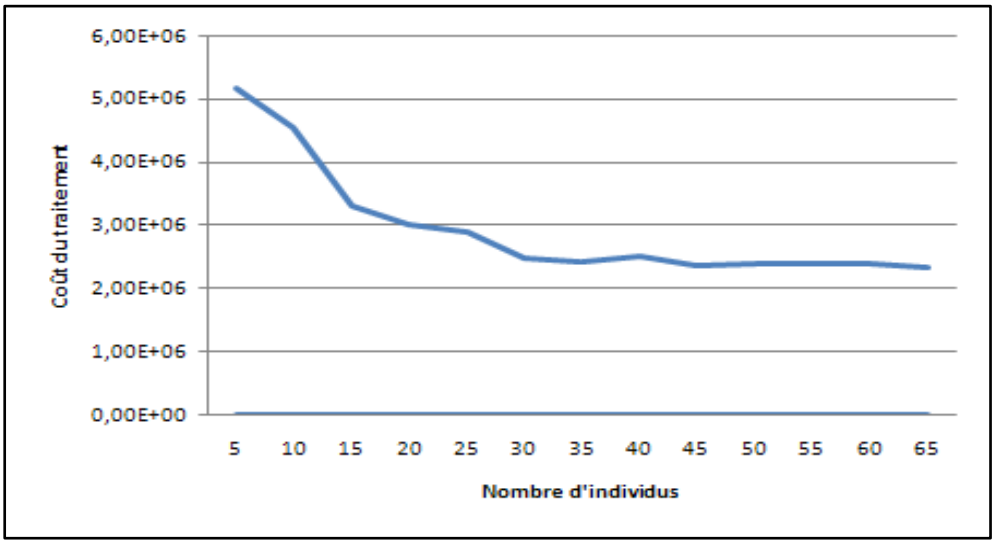

Figure 6-2 Evolution du coût de traitement en fonction de nombre d'individus.

3.2 Paramètre d'évaporation

Ce paramètre représente le taux d'évaporation, nous remarquons que le coût de traitement est minimal lorsque le paramètre d'évaporation varie entre 0.1 et 0.8. Nous fixons ce paramètre à 0.45. Ce qui signifie que le système prend en compte 45 % de quantité d'information du passé et 55% du présent.

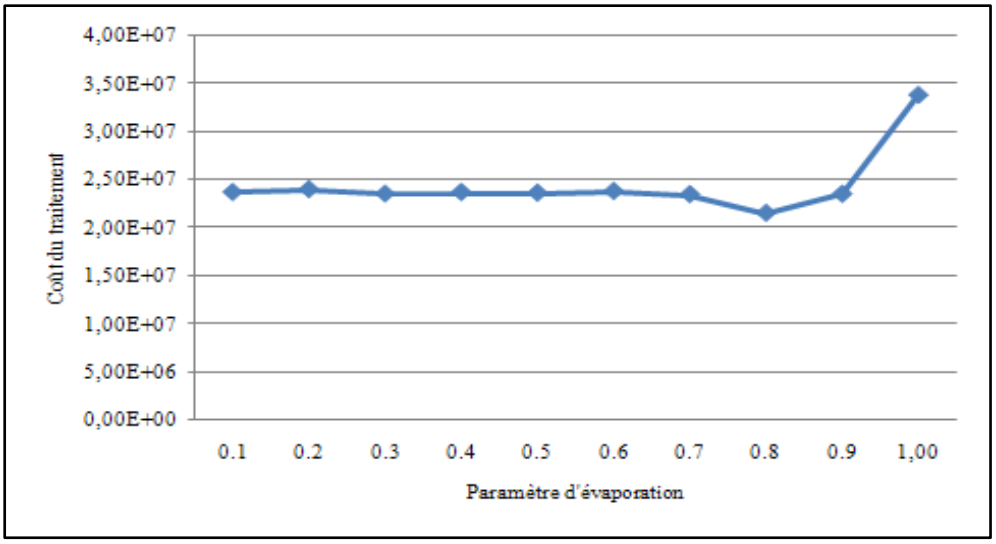

Figure 6-3 Evolution du coût de traitement en fonction du paramètre d'évaporation.

3.3 Le paramètre q0

D'après le graphe, nous remarquerons que le coût du traitement de charge de données incrémente lorsque la valeur de q0 est >0.4, d'où nous fixons q0 à 0.4.

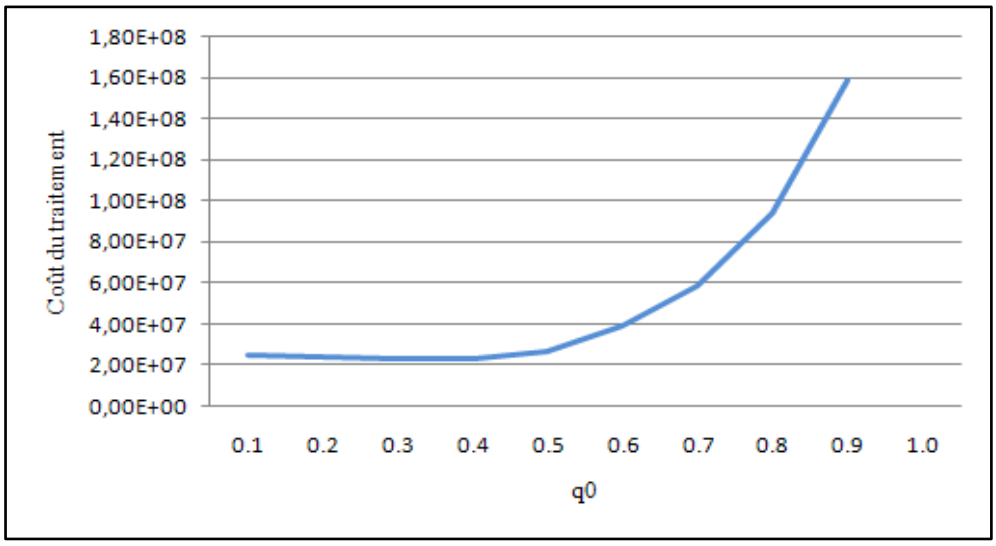

Figure 6-4 Evolution du coût de traitement en fonction de q0.

3.4 Le paramètre Max_iter

La valeur minimale de coût de traitement est constatée lorsque Max_iter vaut 30.

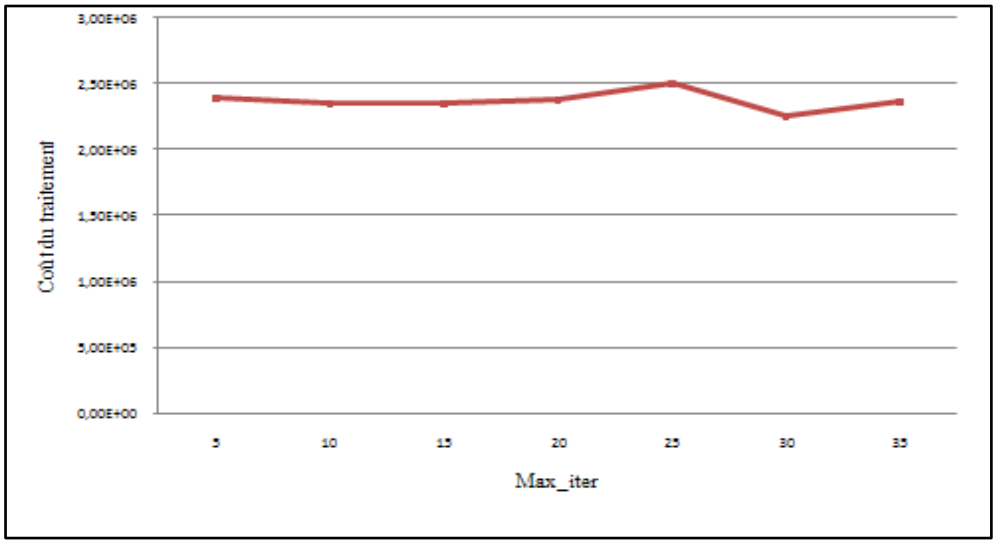

Figure 6-5 Evolution du coût de traitement en fonction de Max_iter.

3.5 Meilleurs résultats obtenus

Afin de valider notre stratégie de sélection des vues matérialisées, Nous avons mesuré le temps d'exécution des requêtes de cette charge dans les cas suivants: sans matérialisées et avec vues matérialisées. Pour la charge de requêtes utilisée, le cout d'évaluation des requêtes est calculé sans considérer les vues matérialisées. les vues sélectionnées en utilisant l'algorithme select_vue réduisent d'une manière significative le cout initiale obtenue par le calcule du cout d'exécution de la charge de requêtes sans matérialisation.

La Figure 6.5 représente la variation de ce temps en fonction de l'espace de stockage. Le cout d'exécution de la charge est calculé par rapport à l'espace total occupé par les vues sélectionnés obtenus en appliquant notre stratégie de sélection des vues. Après avoir fixé les paramètres empiriques, nous présenterons les résultats obtenues suite à l'exécution de notre algorithme (figure 6-6).

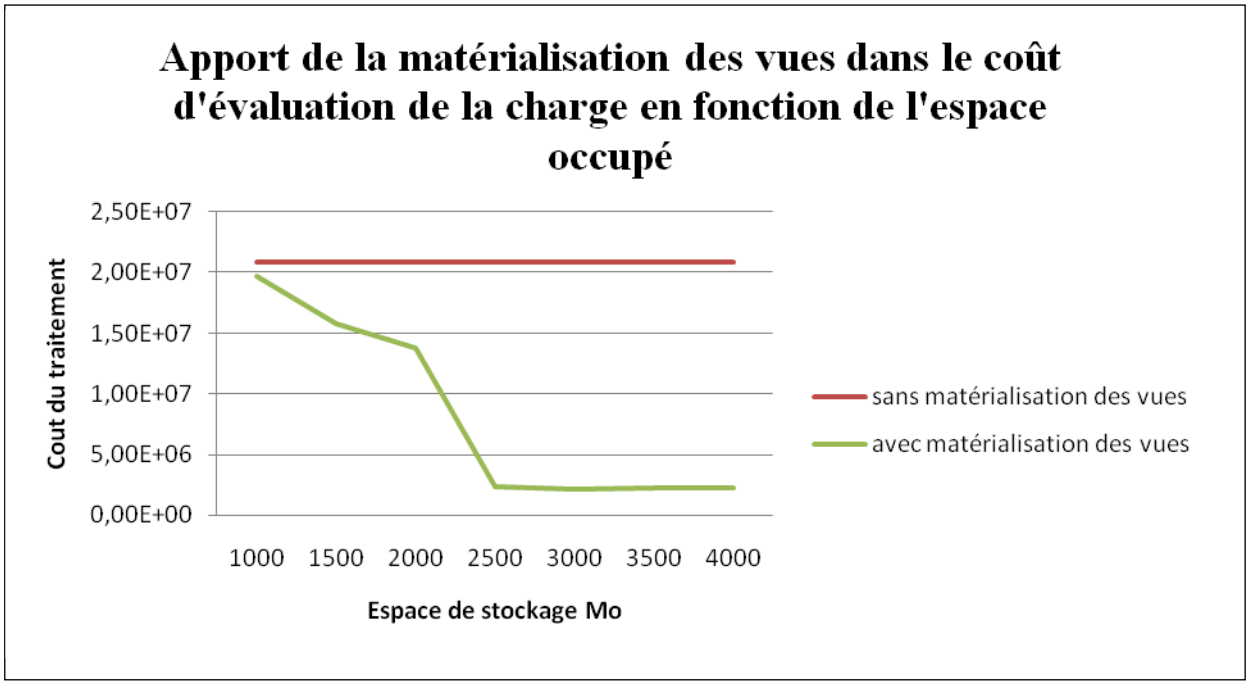

Figure 6-6 Apport de matérialisation des vues en fonction de l'espace de stockage.

4. Réglage des paramètres empiriques de l'algorithme de sélection des index

Pour pouvoir fixer les paramètres empiriques utilisés dans l'algorithme de sélection d'index, des tests sont effectués sur une charge de requêtes de 40 requêtes. Le schéma de l'entrepôt de données utilisées est présenté dans le chapitre précédant. Dans le tableau suivant nous présentons le nombre d'index trouvé et la meilleure solution trouvée par les fourmis en changeant à chaque fois l'espace global alloué aux index.

4.1Le paramètre Max_iter

Nous présentons dans le tableau suivant les résultats obtenues par les expériences effectuées pour fixé la valeur de Max_iter.

	5		10		15		20		25	
Espace	Nb	fitness	nb	Fitness	Nb	Fitness	Nb	fitness	Nb	fitness
100	5	0.5418549	6	0.5989798	10	0.5814156	9	0.55898833	9	0.7380036
500	6	0.6597836	7	0.5556927	6	0.5715523	8	0.6003896	10	0.7048019
1000	4	0.7258623	6	0.52538717	8	0.79719573	6	0.67103827	10	0.79719573

Dans le cas ou la variable max_iter varie entre 10 et 20, les solutions obtenues sont de bonne qualité. Si la valeur de Max_iter dépasse 20 il n'ya pas une grande évolution de solution. Par conséquence, nous avons fixé la valeur de Max_iter à 20.

4.2Le paramètre nb fourmi

Des expériences sont effectuées sur une collection de 40 requêtes fixant les paramètres Max_iter à 20, α à 0.5 et Max_change à 15. A chaque fois que le nombre de fourmis augmente, la similarité augmente considérablement et deviennent ainsi stable. D'après le tableau ci dessous nous avons choisi de fixer le paramètre Nb fourmi à 15.

Espace	5		10		15		20	
	Nb_index	Fitness	Nb_index	fitness	Nb_index	fitness	Nb_index	fitness
100	7	0.66345423	9	0.6618581	10	0.5089338	9	0.60135216
500	9	0.50383157	8	0.47953346	10	0.9181325	7	0.5718713
1000	8	0.5477717	10	0.5312406	9	0.7440091	10	0.55483985

4.3Le paramètre α

Des expériences sont effectuées sur la charge de données fixant le paramètre Max_iter à 20. Le paramètre α représente le taux d'évaporation, d'après les tests effectués, nous l'avons fixé à 0.5, cela signifie que le système prend en compte 50% de quantité d'information du passé et 50% du présent.

	0.01		0.1		0.5		0.7		0.9	
espace	Nb	fitness	nb	Fitness	Nb	fitness	Nb	Fitness	Nb	fitness
100	5	0.6892898	6	0.5587998	9	0.5522276	6	0.44526392	8	0.41538537
500	5	0.5715523	9	0.5848847	8	0.88273335	9	0.54606026	8	0.6220815
1000	7	0.7287811	6	0.9459774	8	0.5595601	9	0.76454103	9	0.91218454

4.4Le paramètre Max_change

La variable Max_change a un effet considérable sur la convergence des solutions. Les tests effectués ont permis de déduire que la valeur de Max_change qui donne les bonnes solutions est 15.

Espace	5		10		15		20	
	Nb_index	Fitness	Nb_index	fitness	Nb_index	fitness	Nb_index	fitness
100	09	0.5827612	09	0.592533	09	0.8508214	06	0.38082626
500	09	0.6696239	09	0.5379874	10	0.6876618	05	0.25728482
1000	09	0.34944993	08	0.44494307	9	0.5680021	07	0.56140366

4.5Meilleurs résultats obtenus

Après avoir fixé les paramètres empiriques, nous présenterons les résultats obtenues suite à l'exécution de notre algorithme de sélection d'index (figure 6-7).

Pour la charge de requêtes utilisée, le cout d'évaluation des requêtes est calculé sans considérer l'indexation. Nous sélection un ensemble d'index bitmap de jointure en utilisant l'algorithme ***select_index***. Ces index réduisent d'une manière significative le cout initiale obtenue par le calcule du cout d'exécution de la charge de requêtes sans indexation.

La courbe verte montre clairement la diminution du coût avec l'augmentation de l'espace occupé par les index, qui permet de sélectionner plus d'index et du coup en faire bénéficier d'avantage de requêtes.

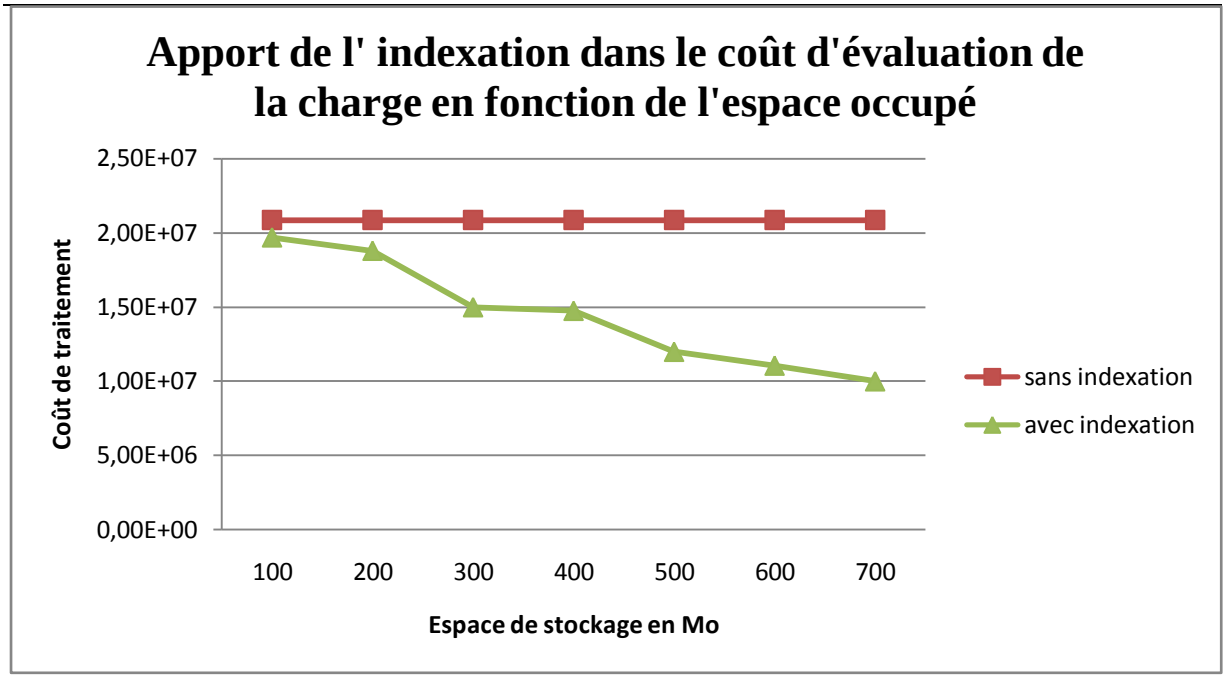

Figure 6-7 Apport de l'indexation en fonction de l'espace de stockage.

Chapitre 7 : Conclusion et perspectives

I. Conclusion

Nous avons étudié dans la cadre de ce travail les différentes techniques d'optimisation des performances de l'entrepôt de données. Nous avons abordé la problématique de sélection des vues matérialisées et des index. Nous nous sommes intéressés particulièrement à la sélection des index bitmap de jointures. Nous nous sommes basés dans ce travail sur les méta-heuristiques à savoir les colonies des fourmis pour définir nos algorithmes proposés.

Nous avons tout d'abord présenté un état d'art sur les entrepôts de données, les principales techniques de modélisation de données de l'entrepôt, les principales techniques d'indexation existantes. Les plus importants travaux consacrés la sélection des vues matérialisées et des index dans les bases et les entrepôts de données ont été présenté. Nous avons présenté aussi un état d'art sur les méta-heuristiques et particulièrement sur les colonies de fourmis.

Deux algorithmes ont été proposés. Un algorithme de sélection des vues matérialisées qui permet de sélectionner un ensemble de vues à matérialiser dans le cube de données en utilisant l'heuristique des colonies de fourmis sous la contraintes d'espace de stockage. Et un algorithme de sélection d'index bitmap de jointures. En fait, nous avons adapté le problème de sélection d'index au problème de recherche d'information. Ces deux algorithmes proposés se caractérisent par les avantages suivants :

- Ils sont Indépendants du SGBD utilisé.
- Les deux algorithmes proposés ne présentent aucun dialogue avec l'optimiseur de requêtes .
- Elimination des attributs clés de l'ensemble de départ permettant de réduire énormement l'espace de recheche.

Enfin, pour montrer l'intérêt de notre approche de sélection, nous avons effectué une série de tests. Ces tests nous permettent de fixer les paramètres empiriques des deux algorithmes de sélection. Les performances de l'indexation et la matérialisation des vues sont évaluées par l'intermédiaire d'un modèle de coût, calculant le coût d'exécution de la charge de requêtes.

II. Perspectives

Le travail mené dans cette thèse ouvre la voie à plusieurs perspectives de recherche. En effet, il serait intéressant d'étudie les approches suivantes :

- Tous les points que nous avons abordés sont en fait des problèmes d'optimisation que nous avons abordé par la méta-heuristique colonies de fourmis. Il serait intéressant de considèrer d'autres types de méta-heuristiques. Il serait intéressant aussi d'utiliser d'autres algorithmes dédiés pour résoudre les problème d'optimisation comme les algorithmes de recuit simulé ou les algorithmes génétiques.

- Nous avons considèré dans notre travail que la seule contraite à respecter était la contrainte d'espace de stockage, il serait intéressant d'intégrer les contraintes de maintenance des index (respectivement des vues matérialisées) dans le processus de sélection.

- Nous avons proposé d'évaluer les performaces de nos algorithmes proposés par des modèles de coûts. Il serait intéressant de confirmer les résultats obtenus par des expérimentions systématiques avec des benchmark, à savoir TPC-H[1] ou bien APB-1[2]. Ce qui nous permettra d'effectuer les tests sur un SGBD réel à savoir Oracle.

- Dans notre travail, nous avons considéré qu'il n y a aucune intéraction entre les vues matérialisées et les index binaires de jountures. il serait intéressant de prendre en considèration les vues indexées lors de la sélection.

- Les index et les vues matérialisées partagent les mêmes ressources d'espace de stockage. Dans notre travail nous avons traité le problème de sélection d'une façon isolée, il serait intéressant de considèrer le problème de sélection simultanée d' index et de vues matérialisées.

- Adapter notre approche de sélection à d'autres schéma de données et plus exactement aux bases de données relationnelles.

III. Publications dans le cadre de ce travail

DRIAS Habiba, FRIHI Ibtissem 'ACO based Approach and Integrating Information Retrieval Technologies in Selecting Bitmap Join Indexes', IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, September 2010, pp 448-451

FRIHI Ibtissem, DRIAS Habiba 'Un système de colonie de fourmis pour la sélection des vues dans un entrepôt de données'. Doctoriales STIC'11, Tébessa

[1] TPC Home page,Tpc benchmark™d (decision support). http://www.tpc.org

[2] **Council, O.** APB-1 OLAP Benchmark, Release II. http://www.olapcouncil.org/research/bmarkly.htm

Bibliographie

[ACN00] **Agrawal S. Chaudhuri S. Narasayya V.R.**
Automated Selection of Materialized Views and Indexes in SQL Databases. in 26th International Conference on Very Large Data Bases (VLDB 2000), Cairo, Egypt, pp. 496_505. 2000.

[AOU02] **Kamel AOUICHE**
Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données. Université Lumière Lyon 2- 2005

[BB03] **Baril X., Bellahsène Z.,**
Designing and Managing an XML Warehouse,
pp.455_473, XML Data Management : Native XML and XML-enabled Database Systems, Addison Wesley. 2003.

[BBC08] **Bellatreche, L., Boukhalfa, K. et Caffiau, S.** « ParAdmin: Un Outil d'Assistance à l'Administration et Tuning d'un Entrepôt de Données ». *Revue des Nouvelles Technologies de l'Information (EDA'2008), Edited by Editionns Cépaduès.* 2008a.

[BEL00] **Ladjel Bellatreche**
L'utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données. PHD. Université de Clermont-Ferrand 2-2000.

[BEL06] **L. Bellatreche, K. Boukhalfa, and H. I. Abdalla**.
Saga : A combination of genetic and simulated annealing algorithms for physical data warehouse design. in 23rd British National Conference on Databases, (212-219), July 2006.

[BEL07] **L. Bellatreche, K. Boukhalfa, and M. K. Mohania.**
Pruning search space of physical database design. In DEXA 2007, pages 479–488, 2007.

[BEL08] **L. Bellatreche, R. Missaoui, H. Necir, and H. Drias.**
A data mining approach for selecting bitmap join indices. Journal of Computing. Science and Engineering, 2(1) :206–223.2008.

[BKS00] **Bellatreche L. Karlapalem K. Schneider M.**
On efficient storage space distribution among materialized views and indices in data warehousing environments, in 9th International Conference on Information and Knowledge Management (CIKM 2000), McLean, USA, pp. 397_404. 2000.

[BM72] **Bayer R. McCreight E.M.**
Organization and Maintenance of Large Ordered Indices, Acta Informatica, 173_189. 1972.

[BOL02] **Nathalie RYSER BOLOGNINI**
Etude pour la création d'un entrepôt de données dans le cadre de l'assurance vie et transformation des données en informations utiles. En vue de l'obtention du Diplôme post grade en informatique et organisation. Université de Lausanne école des hautes études commerciales 2000-2002

[BON99] **Philippe Bonnet**.
Prise en compte des sources de données indisponibles dans les systèmes de médiation. Thèse de Doctorat, Université de Chambéry, Chambéry, France 1999.

[BPT97] **E. Baralis, S. Paraboschi, and E. Teniente.**
Materialized view selection in a multidimensional database. Proceedings of the International Conference on Very Large Databases, pages 156–165, August 1997.

[BRD01] **P.Brunel, V.Rollin, J.Darmont,L.Gruenwald**
DBMS Auto indexing using data mining techniques. Rapport technique. Université d'Oklahoma USA et université lumière Lyon 2 France 2011.

[CAN 01] **Agrawal S. Chaudhuri S. Narasayya V**.
Materialized View and Index Selection Tool for Microsoft SQL Server 2000, in ACM SIGMOD International Conference on Management of Data (SIGMOD 2001), Santa Barbara, USA, p.608. 2001.

[CAR75] **Cardenas A.F.**
Analysis and Performance of Inverted Data Base Structures, Communication of the ACM, 18(5) :253_263. 1975.

[CBC93] **S. Choenni, H. Blanken, and T. Chang.**
Index selection in relational databases. In 5th International Conference on Computing and Information (ICCI 93),Ontario, Canada, pages 491–496, 1993.

[CHA07] **S. Chaudhuri and V. R. Narasayya. Selftuning**
database systems : A decade of progress. In VLDB 2007, pages 3–14, 2007.

[CHA97] **Chaudhuri S., Dayal U.**
"An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD Record, 26(1), 1997.

[CHN97] **S. Chaudhuri and V. Narasayya.**
An efficient cost-driven index selection tool for Microsoft sql server. Proceedings of the International Conference on Very Large Databases, pages 146–155, August 1997.

[COD93] **Codd E.F.**
Providing OLAP (on-line analytical processing) to user-analysts : an IT mandate", Technical Report, E.F. Codd and Associates, 1993.

[CRI99] **Crimmins F., Dkaki T., Mothe J., F.Smeaton A., T_etraFusion**
Information Discovery on the Internet , IEEE Intelligent Systems and their applications, vol. 14, n 4, 1999, p. 55{62, IEEE Computer Society.

[DG04] **D. Gaertner**
Natural Algorithms for Optimisation Problems, Outsourcing
Report; 16 Janvier 2004.

[DM 04] **Dorigo, M., & Stützle,** T. *Ant colony optimization.* MIT Press. 2004

[DOR01] **M.Dorigo,**
The ant colony optimization metaheuristic : algorithms, application, and advances , Technical Report IRIDIA-2000-32, IRIDIA, Université Libre de Bruxelles, Belgium, 2001.

[E.C93] **E. Codd S. C., Salley C.**
Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate , 1993.

[EM 05] **Encinas, M.**
Entrepôts de données pour l'aide à la décision médicale : conception et expérimentation. *Thèse de doctorat, Université Joseph Fourier.* 2005.

[FAY 96] **Fayyad U. Piatetsky-Shapiro G. Smyth P. Uthurusamy R.**
Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996.

[FON92] **Frank M.R., Omiecinski E., Navathe S.B.,**
Adaptive and Automated Index Selection in RDBMS, in 3rd International Conference on Extending Database Technology, EDBT 1992, Vienna, Austria, volume 580 de Lecture Notes in Computer Science, pp. 277_292. 1992.

[FR03] **Feldman Y.A., Reouven J.**
A knowledge_based approach for index selection in relational databases. Expert System with Applications, 25(1) :15_37. 2003.

[FRA01] **J.M Franco., S. De Lignerolles.**
Piloter l'entreprise grâce au data warehouse. Edition Eyrolles 2001.

[GAR00] **G.Gardarin.**
Internet/intranet et base de données, Edition Eyrolles2000.

[GE02] **Guerrero, E.**
Infrastructure adaptée pour l'évolution des entrepôts de données. *Thèse de doctorat,Université de Joseph Fourier.* 2002.

[GR98] **Golfarelli M., Rizzi S.**

A Methodological Framework for DataWarehouse Design,
in 1st ACM international workshop on Data warehousing and OLAP (DOLAP 1998), New York, USA, pp. 3_9. 1998.

[GRAY96] **J.Gray, A. Bosworth, A. Layman and H. Pirahesh,**
« Datacube : a relational Aggregation Operateur Generalizing Group By, Cross-tab, and Sub-totals », IEEE Int. Conf. On Data Enegineering, pp.152-159, 1996.

[GRS02] **M. Golfarelli, , and E. Rizzi, S. Saltarelli.**
Index selection for data warehousing. Proceedings 4th International Workshop on Design and Management of Data Warehouses (DMDW'2002), Toronto, Canada, pages 33–42, 2002.

[GUN99] **Gundem T.I**., _
Near optimal multiple choice index selection for relational databases, Computers & Mathematics with Applications, 37(2) :111_120. 1999.

[GUP97] **H. Gupta.**
Selection of views to materialize in a data warehouse. Proceedings of the 6th International Conference on Database Theory
(ICDT '97), pages 98–112, 1997.

[GUP99] **Gupta H.**
Selection and Maintenance of Views in a Data Warehouse, Thèse de doctorat, Stanford University. 1999.

[HAR97] **H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman.**
Index selection for OLAP.
Proceedings of the International Conference on Data Engineering (ICDE), pages 208–219, April 1997.

[HRU96] **Harinarayan V., Rajaraman A., Ullman J.D.**
Implementing data cubes effciently _, in ACM SIGMOD International Conference on Management of data (SIGMOD 1996), Montreal, Canada, pp. 205_216. 1996.

[INM94] **Inmon W.**
Building the Data Warehouse,
John Wiley and Sons, 1994.

[ISR83] **Ip M.Y.L., Saxton L.V., Raghavan V.V**
On the Selection of an Optimal Set of Indexes , IEEE Transactions on Software Engineering, 9(2) :135_143.1983.

[JN03] **Jouve P. Nicoloyannis N.**
KEROUAC : an Algorithm for Clustering Categorical Data Sets with Practical Advantages _, in International Workshop on Data Mining for Actionable Knowledge (DMAK 2003), Seoul, Korea. 2003.

[KED 99] **Kedad Z., M_etais E.**
Dealing with Semantic Heterogeneity During Data Integration , Proceeding of the18th International Conference on Conceptual Modeling.ER'99, Paris (France), 1999.

[KIM 96] **R-Kimball,**
The data ware house Toolkit. John Wiley & Sons 1996

[KIM01] **Ralph Kimball**
Entrepôts de données, Guide pratique du concepteur de data warehouse
Vuibert 2001

[KLT03] **Kratica J. Ljubic I. Tosic D.**
A Genetic Algorithm for the Index Selection Problem, in Applications of Evolutionary Computing, EvoWorkshops 2003 : EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM, volume 2611 de LNCS, pp. 281_291. 2003.

[LQA97] **W.J.Labio,D.Quass**
Phisical databasedesigne for data warehouse.
Proceedings of the International Conference on Data Engineering (ICDE), 1997.

[MEN00] **Mendelzon A., Vaisman A.**
Temporal Queries in OLAP , Proceedings of 26th International Conference on Very Large Data Bases - VLDB 2000,
Cairo (Egypt), 2000.

[MIC01] Microsoft, AutoAdmin : Self-Tuning and Self-Administering Databases,
http ://www.research.microsoft.com/dmx/AutoAdmin. 2001.

[MON00] **N. Monmarché.**
Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation ». Thèse de doctorat. Ecole Doctorale : Santé, Sciences et Technologies. Décembre, 2000 .

[NRA89] **S.BNavath and M.R.**
Vertical partitioning for database design: a graphical algorithm. ACM SIGMOD, pages 440-450, 1989.

[PED99] **Pedersen T., Jensen C.**
Multidimensional Data Modeling for Complex Data , Proceedings of the International Conference on Data Engineering .
ICDE'99, 1999.

[ROU01] **O.Roux.**
La mémoire dans les algorithmes à colonie de fourmis : applications à l'optimisation et à la programmation automatique ».
Thèse de doctorat. Université du Littoral Côte d'Opale. Décembre 2001.

[SEG03] **M. Segond.**
Application d'un algorithme à colonie de fourmis à la détection de structures retentives en eaux côtières, Laboratoire d'Informatique du Littoral, Maison de la Recherche Blaise Pascal. France, 2003.

[SLJ04] **Smith J.R, Li C.S, Jhingran A,**
A Wavelet Framework for Adapting Data Cube Views for OLAP , IEEE Transactions on Knowledge and Data Engineering, 16(5) :552_565. 2004.

[SMH99] **S. M. Sait et H. Youssef**
iterative computer algorithms with applications in ingineering: solving combinatorial optimization problems, IEEE computers society, 1999.

[STO00] **T. Stöhr, H. Märtens, and E. Rahm. Multidimensional**
database allocation for parallel data warehouses. Proceedings of the International Conference on Very Large Databases, pages 273–284, 2000.

[TES00] **O. Teste.**
Modélisation et manipulation d'entrepôt de données complexes et historiées. Thèse de doctorat. Institue de recherche en informatique de Toulouse.
Laboratoire IRIT-Pole SIG 2000.

[VRU96] **V. Harinarayan, A. Rajaraman, and J. Ullman.**
Implementing data cubes efficiently. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 205–216, June 1996.

[VVK02] **Valluri S.R, Vadapalli S, Karlapalem K,**
View Relevance Driven Materialized View Selection in Data Warehousing Environment , in 13th Australasian Database Technologies (ADC 2002), Melbourne, Australia, pp. 187_196. 2002.

[VZZ+00] **G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelley.**
Db2 advisor : An optimizer smart enough to recommend its own indexes. In 16th International Conference on Data Engineering (ICDE 00), San Diego, USA, pages 101–110, 2000.

[YAN 00] **Yang J., Widom J.**
Temporal View Self-Maintenance in a Warehousing Environment , Proceedings of the 7th International Conference on Extending Database Technology - EDBT 2000, Konstanz (Germany), 2000.

[YAN 98] **Yang J., Widom J.**
Maintaining Temporal Views Over Non-Temporal Information Sources For Data Warehousing , Proceedings of the 6th International Conference on Extending Database Technology. Valencia, Spain, 1998.

[YAN03] **Y. Semet.**
Application de l'optimisation par colonies de fourmis à la structuration automatique de parcours pédagogiques. Mémoire de fin d'études d'ingénieur.
Université de Technologie de Compiègne. France, 2003.

[YAO77] **Yao S.B.**
Approximating Block Accesses in Database Organizations, Communication of the ACM, 20(4) :260_261. 1977.

Sites web

[Deci] : www.decisionnel.net/datawarehouse/dwh.htm

Annexe A : Charge de requêtes

1. select sales.time_id , sum(quantity_sold), sum (amount_sold) from sales, times where sales.time_id = times.time_id and times.fiscal_year ='1998' group by sales.time_id;

2. select sales.time_id, sum(quantity_sold), sum (amount_sold) from sales, times where sales.time_id = times.time_id and times.fiscal_year ='2000' group by sales.time_id;

3. select sales.time_id, sum(quantity_sold), sum (amount_sold) from sales, times where sales.time_id = times.time_id and times.fiscal_year ='1996' group by sales.time_id;

4. select sales.time_id, sum(quantity_sold), sum (amount_sold) from sales, times where sales.time_id = times.time_id and times.fiscal_year ='1998' group by sales.time_id;

5. select sales.time_id, sum(quantity_sold), sum (amount_sold) from sales, times where sales.time_id = times.time_id and times.fiscal_year ='1999' group by sales.time_id;

6. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='newspaper' and products.prod_category ='Women' group by sales.prod_id;

7. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='newspaper' group by sales.prod_id;

8. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='post' group by sales.prod_id;

9. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='year' group by sales.prod_id;

10. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='TV'group by sales.prod_id;

11. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='TV' group by sales.prod_id;

12. select sales.prod_id, count (*) from sales, products, times where sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='1996' group by sales.prod_id order by sales.prod_id;

13. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='internet' and products.prod_category ='Women' group by sales.prod_id;

14. select sales.time_id, sum(quantity_sold), sum (amount_sold) from sales, times where sales.time_id = times.time_id and times.fiscal_year ='1999' group by sales.time_id;

15. select sales.prod_id, avg (amount_sold) from sales, products, promotions where sales.prod_id = products.prod_id and sales.promo_id = promotions.promo_id and promotions.promo_category ='post' group by sales.prod_id;

16. select sales.prod_id, sum(sales.quantity_sold) from sales, channels, products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Internet' and products.prod_category ='Women' group by sales.prod_id;

17. select sales.time_id, sum (quantity_sold) from sales, products, times where sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='1999' group by sales.time_id;

18. select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels,products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Internet' and products.prod_category ='Women' group by sales.channel_id;

19. select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels, products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Catalog' and products.prod_category ='Girls' group by sales.channel_id;

20. select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels, products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Partners' and products.prod_category ='Boys' group by sales.channel_id;

21. select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels,products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Catalog' and products.prod_category ='Women' group by sales.channel_id;

22. select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels, products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Partners' and products.prod_category ='Men' group by sales.channel_id;

23. select sales.prod_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels, products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Internet' group by sales.prod_id;

24. select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id = customers.cust_id and customers.cust_gender='M' group by sales.cust_id;

25. select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id = customers.cust_id and customers.cust_gender='M' group by sales.cust_id;

26. select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id = customers.cust_id and customers.cust_gender='F' group by sales.cust_id;

27. select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels,products where sales.prod_id = products.prod_id and sales.channel_id = channels.channel_id and channels.channel_desc ='Internet' and products.prod_category ='Women' group by sales.channel_id;

28. select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id = customers.cust_id and customers.cust_gender='F' group by sales.cust_id;

29. select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id = customers.cust_id and customers.cust_gender='F' group by sales.cust_id;

30. select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id = customers.cust_id and customers.cust_gender='M' group by sales.cust_id;

31. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id= customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='married' and products.prod_category ='Women' group by sales.cust_id;

32. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='single' and products.prod_category ='Women' group by sales.cust_id;

33. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='married' and products.prod_category ='Women' group by sales.cust_id;

34. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id= customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='married' and products.prod_category ='Women' group by sales.cust_id;

35. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='married' and products.prod_category ='Men' group by sales.cust_id;

36. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='married' and products.prod_category ='Women' group by sales.cust_id;

37. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='1999' and customers.cust_marital_status ='single' and products.prod_category ='Women' group by sales.cust_id;

38. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='married' and products.prod_category ='Men' group by sales.cust_id;

39. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='1998' and customers.cust_marital_status ='married' and products.prod_category ='Men' group by sales.cust_id;

40. select sales.cust_id, avg(amount_sold) from sales, customers, products, times where sales.cust_id = customers.cust_id and sales.prod_id = products.prod_id and sales.time_id = times.time_id and times.fiscal_year ='2000' and customers.cust_marital_status ='single' and products.prod_category ='Women' group by sales.cust_id;