REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE AMAR TELIDJI – LAGHOUAT –

Faculté des **S**ciences

Département de **M**athématique et **I**nformatique

MÉMOIRE

Présenté par :

BENMELOUKA Ahmed

Pour l'obtention du diplôme de MAGISTER en Informatique

Option : **I**nformatique **R**épartie et **M**obile (**IRM**)

THÈME

Recherche des règles d'association pour la sélection des index multi-attributs

Soutenu le 20/06/2013, devant le jury formé de :

Mme Hadda CHERROUN	Maître de conférences	Université de Laghouat	Présidente
Mr Mohamed-Bachir YAGOUBI	Professeur	Université de Laghouat	Examinateur
Mr Abdelouahab MOUSSAOUI	Professeur	Université de Sétif	Examinateur
Mr Youcef OUINTEN	Maître de conférences	Université de Laghouat	Encadreur
Mr Benameur ZIANI	Maître assistant	Université de Laghouat	Co-Encadreur

N° d'ordre : ...../2013-M/INF

Dédicaces

A mes chers parents.

A mes frères et sœurs.

A ma femme et mes enfants.

Remerciements

Je tiens à exprimer toute ma reconnaissance à Messieurs Youcef Ouinten et Benamar Ziani, respectivement directeur et co-directeur de ce mémoire, pour m'avoir accueilli, encadré et mis à ma disposition tous les moyens nécessaires et logistiques, leur patience, également par leurs idées, et avec lesquels j'ai eu de nombreux échanges durant l'accomplissement de ce travail.

J'adresse tous mes remerciements à Madame Cherroun Hadda, pour avoir accepté de présider le Jury de soutenance, je remercie également Monsieurs Mohamed Bachir Yagoubi professeur à l'université de Laghouat, pour son accord d'être membre du jury malgré ses occupations.

Je tiens à remercier Monsieur Moussaoui Abdelouhab pour sa participation au jury ainsi que pour ses conseils pendant l'année théorique.

Je dois également exprimer ma gratitude à Youcef Ariouet qui m'a accompagné tout au long de ce mémoire et qui a su supporter avec bienveillance mon humeur des derniers moments.

Je n'oublierai pas Monsieur Ouled Mohamed Bachir pour son aide précieuse, à qui je témoigne toute mon amitié.

Je remercie mes amis pour leur soutien moral, et mes collègues de travail.

Résumé

Un entrepôt de données est une base de données spécifique contenant des informations historisées destinées aux processus d'aide à la décision. L'efficacité de l'interrogation d'un entrepôt de données est liée à sa conception physique. Cette conception repose sur la sélection d'index pertinent permettant de réduire le coût des requêtes complexes définies sur l'entrepôt. La sélection d'index est un problème NP-complet car le nombre d'index possibles est exponentiel vis à vis du nombre total d'attributs candidats à la procédure d'indexation. Les travaux réalisés pour résoudre ce problème proposent des méthodes d'optimisation pour réduire cette complexité et recommandent des configurations d'index permettant de minimiser le coût d'une charge de requêtes en respectant la contrainte d'espace de stockage disponible pour les index créés. Dans ce mémoire nous proposons une approche de résolution du problème de sélection d'index basée sur la recherche des règles d'association ; au delà de leur utilisation pour la prédiction, nous les avons adaptés pour la sélection des index multi-attributs afin d'améliorer le coût d'exécution en respectant l'espace de stockage.

Nous avons choisi pour valider notre approche de manière expérimentale d'élaborer un outil *ISAR* qui prend en entrée une charge de requête issu du benchmark APB1 (Council, 1998) et en sortie il génère une configuration d'index finale. Les résultats expérimentaux obtenus montrent que notre approche permet d'améliorer le temps de traitement des requêtes et le coût de stockage comparé à des travaux antérieurs.

MOTS-CLÉS : Entrepôt de données, index de jointure binaire, motifs fréquents, règles d'association

Abstract

A data warehouse is a database containing historical information for the process of decision support. The efficiency of querying a data warehouse is linked to its physical design. This design is based on the selection of relevant indexes to reduce the cost of complex queries defined on the warehouse. Index selection problem is an NP-complete, because the number of possible indexes is exponential with respect to the total number of attributes candidates for indexing procedure. The work done to solve this problem proposed optimization methods to reduce this complexity and recommend configurations index to minimize the cost of query load respecting the constraint of available space for indexes created. In this paper we propose an approach to solving the problem of index selection based on the research of association rules, beyond their use for prediction.

We have adapted to the selection of multi-attribute index in order to improve cost performance respecting storage space.We chose to validate our approach experimentally to develop a tool *ISAR* which takes as input a query load from the benchmark APB1 (Council, 1998) and output it generates an index configuration finals. The experimental results show that our approach improves the processing time of applications and the cost of storage compared to previous work.

KEYWORDS : Data Warehouse binary join index, frequent patterns, association rules

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Face à la mondialisation et à la concurrence grandissante, l'informatique décisionnelle est devenue cruciale ; le problème des entreprises est d'exploité efficacement d'importants volumes d'informations enrichi en permanence par des applications informatiques transactionnelles dont la taille se mesure en téraoctets (2^{40} octets), voire en pétaoctets (2^{50} octets) stockés dans des entrepôts de données qui sont une véritable mine d'informations destinées aux systèmes décisionnels aux profits des dirigeants et décideurs des entreprises pour exécuter des requêtes complexes exigeant un temps de réponse raisonnable pour les besoins décisionnels. Le modèle schéma en étoile est le plus utilisé pour les entrepôts de données [1] ; ce schéma se constitue principalement d'une table de faits de très grande taille liée par des clés étrangères à un ensemble de tables appelées tables de dimension. La table des faits contient toutes les données observables de l'entreprise collectées durant des périodes de fonctionnement. Les tables de dimension contiennent des données qui représentent des axes d'analyse. Les requêtes définies sur un schéma en étoile (connues par requêtes de jointure en étoile) sont en générale caractérisées par des opérations de sélection sur les tables de dimension, suivies de jointures avec la table des faits, leurs exécutions peuvent prendre un temps énorme. Pour optimiser ces requêtes, l'administrateur est amené à effectuer une tâche importante : la conception physique, durant cette phase l'administrateur choisit parmi un ensemble de techniques d'optimisation. Ces techniques se divisent en deux catégories : redondantes tels que les index et les vues matérialisées et non redondantes tel que la fragmentation horizontale. Dans notre travail, nous nous intéressons à la première catégorie et en particulier les index de jointure binaire qui sont très adaptés pour réduire le coût de telles requêtes.

Dans ce mémoire nous nous intéressons à une technique d'optimisation pour sélectionner une configuration d'index de jointure binaire qui permet d'optimiser le coût et l'espace de stockage pour une charge des requêtes. nous proposons une nouvelle approche basée sur la recherche des règles d'association pour la sélection des index multi-attributs. Il s'agit de s'appuyer sur la puissance des règles d'association notamment les corrélations entre attributs dans la charge de requête afin d'extraire une configuration d'index.

Pour présenter notre approche, nous avons retenu pour ce mémoire une organisation composée de quatre chapitres. Dans le premier chapitre, nous présentons le contexte de notre approche à travers un rappel sur les concepts de base des entrepôt de données et les techniques d'optimisation. Dans le deuxième chapitre est consacré à l'étude du problème de sélection d'index de jointure binaire à savoir les étapes de sélection d'index de jointure binaire ; une formalisation du problème et en fin un état de l'art des travaux antérieurs liés à ce type de problème. Le troisième chapitre est consacré à la description détaillée de notre approche ; nous présentons d'abord le concept des règles d'association ; nous définissons la démarche pour la construction des règles d'association, des définitions des concepts de base, une présentation de l'algorithme Apriori puis nous décrivons en détail les démarches de notre approche pour la sélection d'index multi-attribut par la recherche des règles d'association. Dans le quatrième chapitre, une étude expérimentale qui montre les résultats obtenus ainsi qu'une comparaison avec l'approche des motifs maximaux ; en utilisant un modèle de coût.

Nous concluons ce mémoire en rappelant la problématique étudiée ainsi que les résultats obtenus et en proposant quelques perspectives.

Chapitre I

Techniques d'optimisation des entrepôts de données

I.1 Introduction

Le système décisionnel au sein des entreprises permet de déterminer la stratégie à court, moyen et à long terme grâce à l'exploitation d'un nombre important de données stocké dans des entrepôts de données qui servent de fondation aux applications décisionnelles. Quatre éléments essentiels constituent l'architecture des systèmes décisionnels [2] : les sources de données, l'entrepôt de données, les magasins de données et les outils d'analyse et d'interrogation.
Les sources de données sont nombreuses, variées, distribuées et autonomes. Elles peuvent être internes (bases de production) ou externes (Internet, bases des partenaires) à l'entreprise.
L'entrepôt de données est le lieu de stockage centralisant des informations d'un organisme ou entreprise utiles pour les décideurs. Il met en commun les données provenant des différentes sources et conserve leurs évolutions.
Les magasins de données sont des extraits de l'entrepôt orientés sujet. Les données sont organisées de manière adéquate pour permettre des analyses rapides à des fins de prise de décision.
Les outils d'analyse permettent de manipuler les données suivant des axes d'analyse. L'information est visualisée à travers des interfaces interactives et fonctionnelles dédiées à des décideurs souvent non informaticiens (directeurs, chefs de services,...) les entrepôts de données conservent l'historique des données avec l'ajout en permanence des nouveaux données provenant de plusieurs sources, cela provoque l'augmentation sans cesse de la taille de stockage Par

conséquent, un grand travail s'avère donc nécessaire pour optimiser l'exécution des requêtes qui sont devenues de plus en plus complexe ce qui exige des techniques d'optimisation qui auront pour rôle d'améliorer les temps d'exécution. Les techniques d'optimisation des entrepôt de données se divisent en deux classes [3] : Les techniques redondantes et les techniques non redondantes, la première classe contient : les index et les vues matérialisées dont l'utilisation produit une duplication des données ; la deuxième classe contient la fragmentation horizontale et le traitement parallèle qui ne dupliquent pas les données mais qui permettent de réorganiser leurs représentation physique pour un gain de temps d'exécution. Avant de présenter ces techniques nous allons voir dans ce qui suit la définition et les concepts de base des entrepôt de données.

I.2 Les entrepôts de données

Les données propres d'une entreprise sont issue de différentes sources hétérogènes et leurs volumes sont destinés à augmenter sans cesse au fil de temps. Il devient fondamental de rassembler et d'homogénéiser les données afin de permettre l'analyse des indicateurs pertinents pour faciliter la prise de décisions. Cette gigantesque base de données s'appelle *entrepôt de données* est mis à la disposition des dirigeants et décideurs de l'entreprise pour anticiper et rester concurrents.

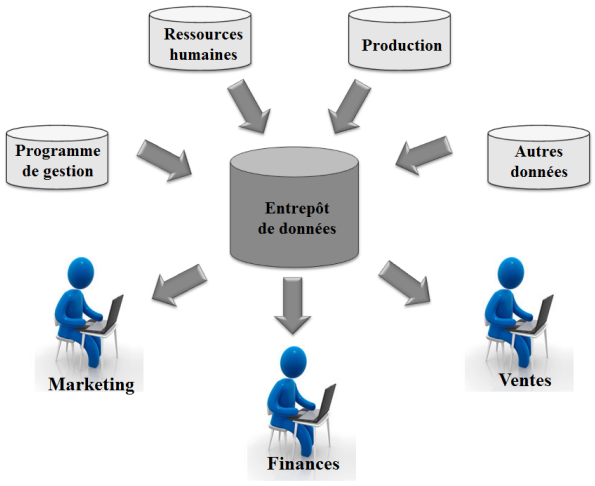


FIGURE I.1 – Entrepôt de données

I.2.1 Définition d'un entrepôt de données

Un entrepôt de données (data warehouse) est un lieu de regroupement de différentes données de l'entreprise en vue de la constitution d'un système d'information décisionnel(figure I.1). Un des précurseurs du concept d'entrepôt de données, *Bill Inmon* le définit comme suit : « *Un entrepôt est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision.* » Cette définition met l'accent sur quatre caractéristiques d'un entrepôt de données qui sont donc :

1. **Données orientées sujet :**

   Contrairement aux données des systèmes de production, les données sont organisées selon des sujets majeurs et des métiers de l'entreprise, l'avantage de cette représentation est de faciliter les analyses sur des sujets liés aux différentes activités de l'entreprise.

2. **Données intégrées :**

   Les données de l'entrepôt de données concernent les différents services et métiers de l'entreprise, pour palier au problème de l'hétérogénéité et la divergence des forme des données l'intégration permet la mise en forme et l'unification des données afin d'avoir un état cohérent qui nécessite une normalisation de données ,mais aussi une bonne maîtrise de la sémantique et la prise en compte des règles de gestion.

3. **Données historisées :**

   Pour suivre l'évolution dans le temps des différentes valeurs des indicateurs à analyser plusieurs versions de données sont stockées dans l'entrepôt de données . Ainsi un référentiel temps est associé aux données pour permettre des analyses comparatives dans le temps.

4. **Données non volatiles :**

   les données stockées au sein de l'entrepôt de données ne peuvent pas être supprimées, afin de conserver la traçabilité de ces dernières pour des analyses dans le temps sur un intervalle de longue période.

I.2.2 Notions de base

Fait :

Dans un entrepôt de données le sujet analysé est représenté par le concept de fait qui est une valeur ou une mesure correspondante aux informations de l'activité de l'entreprise. ces mesures permettent de résumer un grand nombre d'enregistrements en quelques enregistrements (exemple : le maximum, la moyenne...).

Table de fait :

Une table de fait stocke des informations quantitatives à des fins d'analyses, elle est constituée de deux types de colonnes : La colonne de clés étrangères ; qui permet la jointure avec les tables de dimension ; les colonnes contiennent les données des mesures en cours d'analyse.

Table de dimension :

Chaque table de dimension représente un axe d'analyse, il s'agit d'une collection d'informations de référence sur un événement mesurable comme par exemple des produits, des clients d'une entreprise ou d'une période de temps, etc.

I.2.3 Architecture d'un entrepôt de données

Les différents composants de l'architecture fonctionnelle d'un système à base d'entrepôt de données s'organisent en 3 niveaux :

1. Extraction des données à partir des différentes sources.
2. Organisation et intégration des données dans l'entrepôt.
3. Accès aux données intégrées et analyse de ces dernières dans une forme efficace et flexible.

Dans la première et la deuxième phase, les données issues de différentes sources de données sont extraites, nettoyées et intégrées par sujet dans l'entrepôt de données avec un rafraîchissement au fur et à mesure des mises à jour ; durant la troisième phase, un système OLAP se charge de présenter les informations demandées par les utilisateurs sous plusieurs formes : tableaux, rapports, statistiques, etc[4].

I.2.4 Modélisation des entrepôts de données

L'organisation d'un entrepôt de données est primordiale afin de faciliter l'exploitation des données à analyser par les décideurs et en conséquence l'exécution des requêtes complexes pour obtenir des résultats sur différents sujets (achat, vente ...) ; pour bien s'adapter à ce type de présentation, les données d'un entrepôt de données sont représentées sous forme multidimensionnelle qui est le mieux approprié pour le support des processus d'analyse[4].

Le cube de données :

Le modèle type d'un entrepôt de donnée est le cube de données ayant une forme multidimensionnelle lui permettant d'être bien exploitable par les applications OLAP (Figure : I.2), ce modèle permet d'observer les données selon plusieurs perspectives ou axes d'analyse et facilite l'accès aux données. Le cube de données organise les données en une ou plusieurs dimensions qui déterminent une mesure d'intérêt. il s'agit d'une abstraction très proche de la manière dont l'analyste voit et interroge les données. cette restructuration d'un cube est l'ensemble des opérations à entreprendre dans le cube.

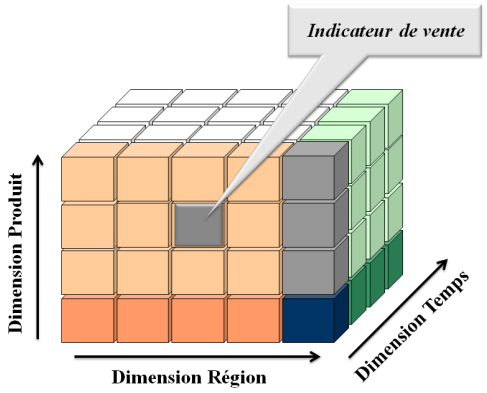


FIGURE I.2 – Cube de données

Opérations liées à la structure :

l'ensemble des opérations sur le cube de données appelé restructuration ou chaque opération de restructuration permet d'avoir un nouveau cube de données à des fins d'analyses, le cube initial est obtenu par restructuration réciproque. Ces opérations sont : pivot, swich, split, nest et

push.

- *Pivot :* Cette opération consiste à effectuer à un cube une rotation autour d'un des trois axes passant par le centre de deux faces opposées, de manière à présenter un ensemble de faces différent.
- *Switch :* Cette opération consiste à interchanger la position des membres d'une dimension.
- *Split :* Elle consiste à présenter chaque tranche du cube, et à passer d'une représentation tridimensionnelle d'un cube à sa représentation sous la forme d'un ensemble de tables. D'une manière générale, cette opération permet de réduire le nombre de dimensions d'une représentation. On notera que le nombre de tables résultant d'une opération split dépend des informations contenues dans le cube de départ et n'est pas connu à l'avance.
- *Nest :* Cette opération permet d'imbriquer les membres. L'un de ses intérêt est qu'elle permet de grouper sur une même représentation bi-dimensionnelle toutes les informations (mesures et membres) d'un cube, quel que soit le nombre de ses dimensions. L'opération réciproque, "unnest", reconstitue une dimension séparée à partir des membres imbriqués.
- *Push :* Cette opération consiste à combiner les membres d'une dimension aux mesures du cube, et donc de faire passer des membres comme contenus de cellules. L'opération réciproque appelée pull, permet de changer le statut de certaines mesures d'un cube en membres, et de constituer une nouvelle dimension pour la représentation du cube, à partir de ces nouveaux membres ; l'opération inverse qui change l'état des mesures en membres est appelée « *pull* ». [5]

Schéma multidimensionnel

L'objectif dans ce type de schéma est de donner une vision multidimensionnelle des données. Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse. Dans le modèle multidimensionnel, le concept central est le *cube de donnée*. Un cube de données est composé d'un ensemble de *cellules*, chaque cellule représente une mesure prise à l'intersection des membres des dimensions. La localisation de la cellule est faite à travers les axes, qui correspondent chacun à une dimension.

Schémas relationnels

Les principaux schémas sont :

1. le schéma en étoile.
2. le schéma en flocon de neige.
3. le schéma en constellation.

Schéma en étoile

C'est le schéma le plus connu dans les systèmes d'implémentation des entrepôts de données. Dans ce type de schéma, les mesures sont représentées par une table de faits et chaque dimension par une table de dimension. La table des faits référence les tables de dimensions en utilisant une clé étrangère pour chacune d'elles et stocke les valeurs des mesures pour chaque combinaison de clés(Figure I.3).

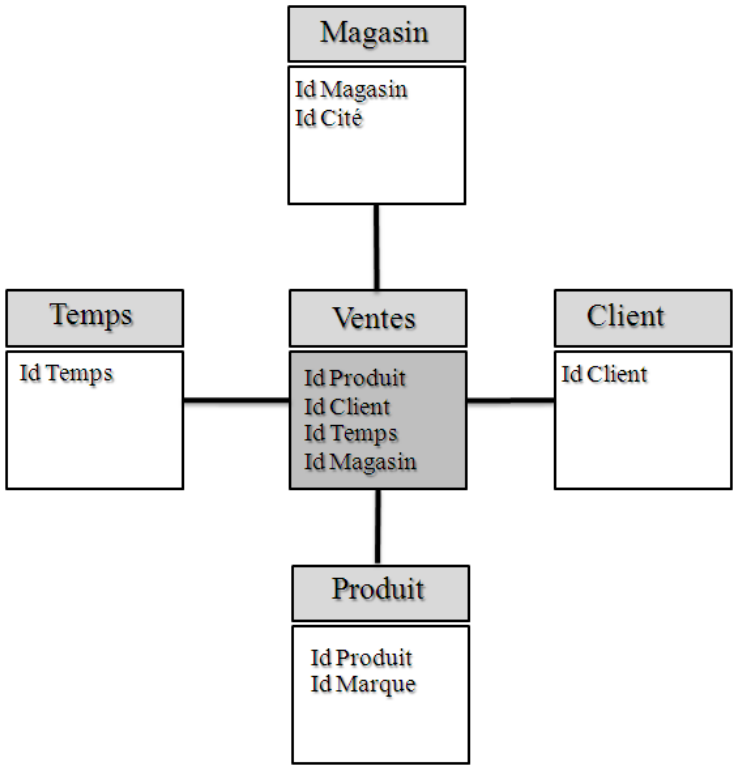


FIGURE I.3 – Schéma en étoile

Schéma en flocon de neige

A la différence du schéma en étoile, le schéma en flocon de neige(Figure I.4) permet de normaliser les tables de dimensions. Cette normalisation réduit la redondance des données mais en contre partie un temps beaucoup plus important pour l'exécution des requêtes qui nécessite plus de jointures [6].

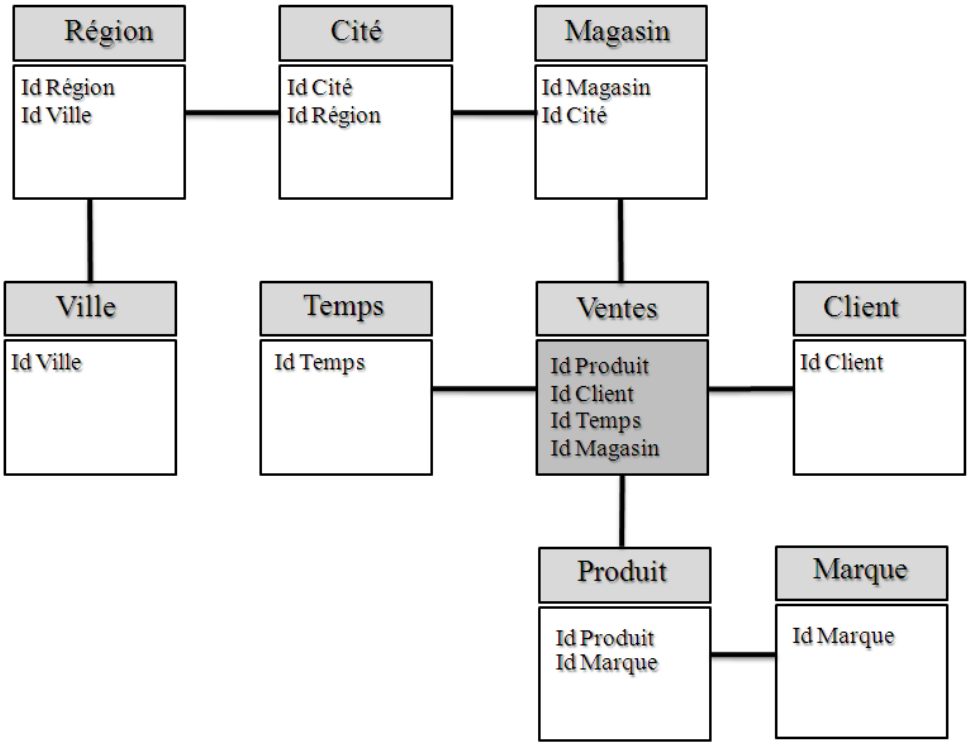


FIGURE I.4 – Schéma en flocon de neige

Schéma en constellation

Dans le schéma en constellation, plusieurs tables de faits partagent les mêmes tables de dimensions, ce modèle est utilisé lorsque les faits ne sont pas déterminés par les mêmes tables de dimensions [7]. La figure I.5 montre le schéma en constellation qui est composé de deux relations de faits. La première s'appelle Ventes ; elle enregistre les quantités de produits qui ont été vendus dans les différents magasins pendant un certain jour. La deuxième relation gère les différentes livraisons par les fournisseurs pendant un certain temps. La relation de faits Ventes partage leurs dimensions Temps et Produits avec la table Livraison ; néanmoins, la dimension Magasin appartient seulement à Ventes. Également, la dimension Fournisseur est liée seulement

à la relation Achats.

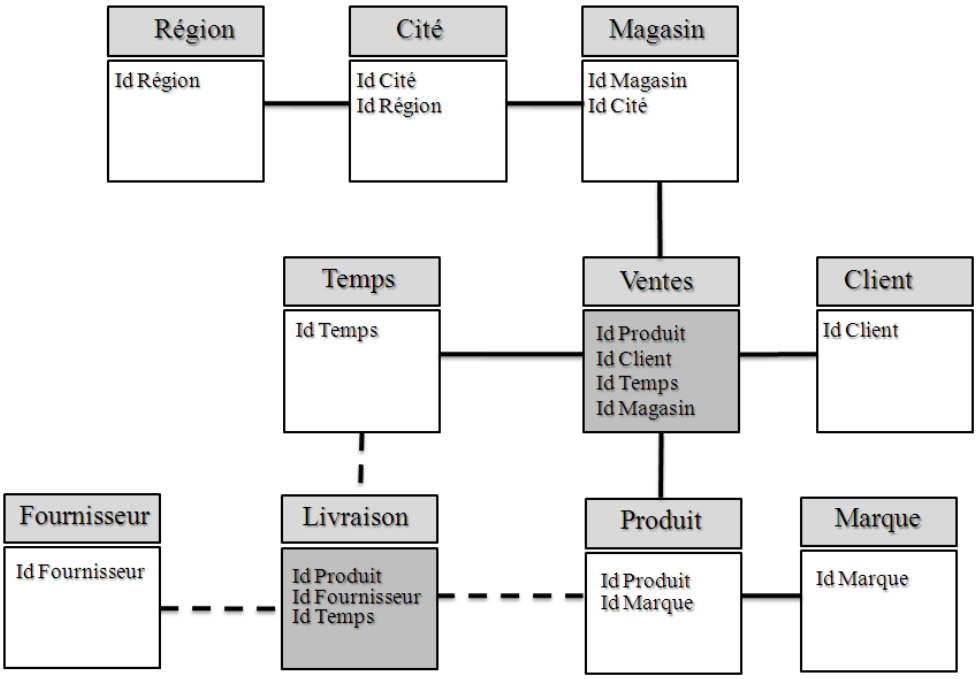


FIGURE I.5 – Schéma de données en constellation

I.2.5 Implémentation du modèle multidimensionnel

L'implémentation du modèle multidimensionnel sur un SGBD réel peut se faire en général selon les modèles : MOLAP (Multidimensional On-Line Analytical Processing) ; ROLAP(Relational On Line Analytical Processing) et HOLAP(Hybrid On Line Analytical Processing). Dans ce qui suit nous décrivons brièvement ces implémentations.

Les systèmes MOLAP

Les systèmes MOLAP (Multidimensional On-Line Analytical Processing) représente le cube de données sous forme d'un tableau multidimensionnel (SGBD multidimensionnel) à n dimensions où chaque dimension du cube est représenté par une dimension du tableau. Par exemple, le cube multidimensionnel de la figure I.2 peut être représenté par le tableau multidimensionnel dans la figure I.6. Le principal avantage de cette implémentation est la performance en temps d'accès car l'accès aux données est direct mais par contre il existe certaines limites à savoir :
- Les opérations de mise à jour sont difficiles à effectuer compte tenu que les valeurs des cellules

du tableau multidimensionnel doivent être recalculées après chaque opération de mis à jour.

- Le besoin de redéfinir des opérations pour manipuler les structures multidimensionnelles.
- Consommation de l'espace lorsque les données sont éparses, ce qui nécessite l'utilisation des techniques de compression.[3].

Produit	ANNEE	2009			2010			2011			2012		
	Magasin	M1	M2	M3	M1	M2	M3	M1	M2	M3	M1	M2	M3
PEUGEOT		150	140	170	180	153	180	194	139	185	189	155	190
OPEL		140	180	194	160	168	203	161	175	207	170	183	199
BMW		99	88	125	104	85	145	105	90	153	150	98	156

FIGURE I.6 – Tableau multidimensionnel

Les systèmes ROLAP

Dans ce type de modèle, la représentation cube de données est assuré par un SGBD relationnel. Chaque fait correspond à une table appelée table de faits et chaque dimension correspond à une table appelée table de dimension. La table de faits possède comme attributs les mesures d'activités et les clés étrangères vers les tables de dimension.Ce modèle a pour avantage l'utilisation des systèmes de gestion de bases de données existant, ce qui réduit le coût de mis en oeuvre [3]. l'avantage principal de la technologie ROLAP est la définition des données complexes et multidimensionnelles en utilisant un modèle relativement simple. La figure I.7 montre une architecture pour le système ROLAP.

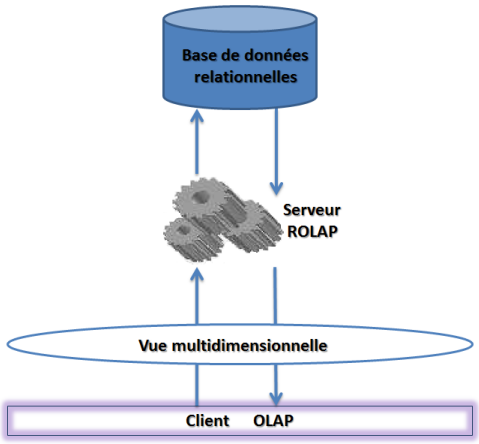


FIGURE I.7 – Architecture ROLAP

Les systèmes HOLAP

Il s'agit d'une combinaison de deux modèles précédents pour bénéficier des avantages des ces modèles ; les données agrégées sont stockées sous formes multi-dimensionnelles pour des meilleurs performances, alors que les données détaillées sont stockées dans des structures relationnelles.

I.2.6 Entrepôt et base de données

Les systèmes de gestion de base de données ont été destinés pour les applications de gestion de système transactionnel à la différence, les entrepôts de données ont été crée pour l'aide à l'analyse et la prise de décision. Ils intègrent toutes les données nécessaires pour fournir une vue globale de l'information aux analystes et aux dirigeants. un résumé des différences entre un système décisionnel et un système transactionnel est montré dans le tableau I.1 suivant :

I.3 Techniques d'optimisation d'un entrepôt de données relationnel

Les tâches d'administration et de tuning sont plus complexes dans le contexte des entrepôts de données par rapport aux bases de données traditionnelles. Plusieurs techniques d'optimisation ont été proposées durant la phase de conception physique d'un entrepôt de données. La figure

Caractéristiques	Système transactionnel	Système décisionnel
Utilisation	SGBD (base de production)	Datawarehouse
Utilisateurs	Gestionnaire de production	Décideur, analyste
Objectifs	Gestion et production	Consultation et analyse
Type d'accès aux données	Lecture écriture	Lecture
Taille de la base	Plusieurs gigaoctets	Plusieurs téraoctets
Ancienneté des données	Récente	Historique
Organisation des données	selon traitement	selon métier
Quantité d'information échangées	Faible	Importante
Requêtes	Simple	Complexe

TABLE I.1 – Différence entre un système décisionnel et un système transactionnel

I.8 dresse une classification de ces structures.

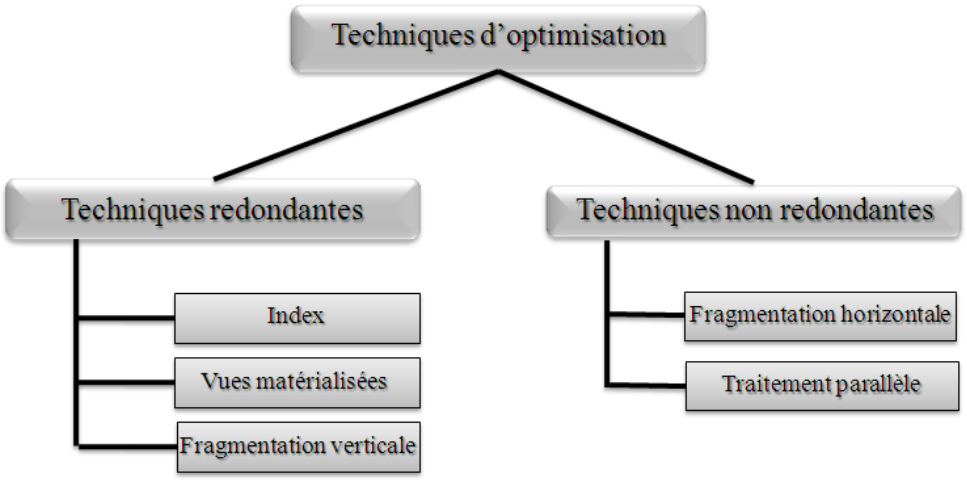


FIGURE I.8 – Classification des techniques d'optimisations

I.3.1 Techniques d'optimisation redondantes

Les techniques redondantes sont des structures d'optimisation qui se caractérisent par la redondance des données des tables dans les structures d'optimisation et qui nécessitent un espace de stockage supplémentaire pour leur implémentation.

Les index

Les index permettent de trouver rapidement une information recherchée dans une base de données. Ils sont stockés physiquement indépendamment des données et peuvent ainsi être supprimées et recrées à tout moment. un index est une structure de données liée à une table de

données permettant d'accélérer la recherche des données. Un index permet un accès direct aux tuples. Chaque entrée d'un index contient la valeur d'un champ d'indexation et une liste de pointeurs sur tous les blocs contenant un tuple dont le champ d'indexation possède cette valeur. Quelle que soit sa nature, tout index est trié, ce qui en permet un parcours dichotomique pour la recherche d'une valeur de champ d'indexation. Les index sont parmi plusieurs techniques d'optimisation proposées durant la phase de conception physique d'un entrepôt de données. Ils permettent de réduire le coût d'exécution des requêtes en minimisant la taille des données manipulées en cours de calcul. Dans la partie suivante Nous allons voir les différents type d'index utilisés pour les bases de données relationnelles et les entrepôts de données.

index B-arbre

Il s'agit d'une structure de données en arbre équilibré à plusieurs niveaux , chaque niveau se compose de plusieurs nœuds ou chaque nœud pointe vers le niveau inférieur, le niveau le plus bas contient les entrées des index ainsi qu'un pointeur vers l'emplacement physique de l'enregistrement correspondant. Les B-arbre et leurs variantes sont intégrés dans la plus part des SGBD Vu les qualités des opérations de recherche et de mis à jour [8].

Index de hachage

L'index de hachage repose sur l'utilisation d'une fonction de hachage. Cette fonction permet, à partir d'une valeur de clé *c*, de donner l'adresse physique *f(c)* d'un espace de stockage où l'élément doit être placé (Figure I.9). le choix de la fonction de hachage est très important pour garantir une bonne performance de l'index. Par exemple, si la fonction donne la même valeur à un nombre important d'éléments, donc l'accès ressemblera à un balayage séquentiel [4].

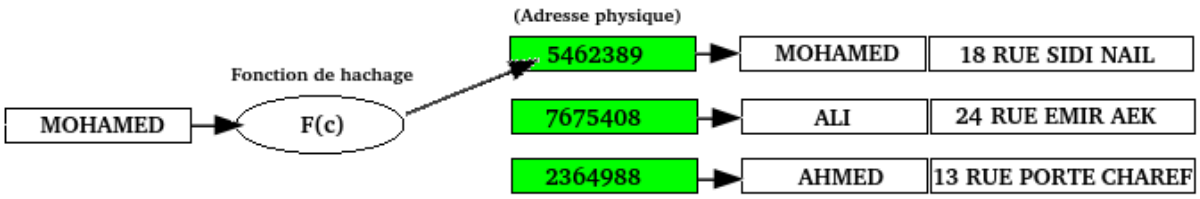


FIGURE I.9 – Exemple de hachage

Index binaire (bitmap index)

Le bitmap est un type spécial d'index des bases de données qui sont traditionnellement considérées pour travailler avec les données représentées par des valeurs booléennes (0,1), un index bitmap considère toutes les valeurs d'un attribut en associant pour chaque valeur un vecteur de 0 et 1. où on assigne un 1 si le tuple de la table de la même ligne possède la valeur de la colonne en cours de l'index, sinon 0 (Figure I.2). Le problème est que pour chaque instance ajoutée dans la table, l'index doit être mise à jour [3]. L'index binaire a été considéré comme le résultat le plus important obtenu dans le cadre de l'optimisation de la couche physique des entrepôts de données[9].

(a) Etudiant

RID	CID	Nom	Age	Ville
1	100	Mohamed	19	LAGHOUAT
2	450	Ali	18	LAGHOUAT
3	300	Omar	20	DJELFA
4	550	Ahmed	18	ALGER
5	400	Khaled	20	DJELFA
6	650	Said	19	ALGER

(b) Index bitmap sur Ville

RID	LAGHOUAT	DJELFA	ALGER
1	1	0	0
2	1	0	0
3	0	1	0
4	0	0	1
5	0	1	0
6	0	0	1

TABLE I.2 – Exemple d'un index Bitmap sur l'attribut Ville.

Index de jointure

Un index de jointure est une structure contenant des couples de rowids de tuples de deux tables ou plus ; pour améliorer le temps d'accès pour les requêtes multi-tables, et qui sont auto-

matiquement mises à jour lorsque les tables sous-jacentes sont modifiées. Dans les entrepôt de données l'index de jointure est calculé généralement sur la table de fait avec un attribut de faible cardinalité d'une table de dimension. Pour obtenir cet index, il suffit de faire la jointure entre la table de fait et la table de dimension puis calculer l'index sur la clé de la table de fait avec l'attribut d'indexation.

Index de jointure en étoile

Un index de jointure en étoile contient toutes les combinaisons entre des n-uplets de la table de faits et les clé étrangères des tables de dimension, cet index est appliqué aux requêtes OLAP définies sur les entrepôts de donnée modélisé en étoile. si la jointure est construite par toutes les tables de dimension avec la table de fait , l'index de jointure en étoile est dit complet mais dans le cas ou la jointure est appliquée à certaines tables de dimensions avec la table de faits ; l'index est dit partiel . L'index de jointure en étoile complet est très efficace aux requêtes définie sur un entrepôt modélisé en étoile. cependant Il exige cependant beaucoup d'espace de stockage et un coût de maintenance très élevé [8].

	Index de jointure en étoile		
Vente_1	Client_1	Produit_1	Temps_2
Vente_1	Client_2	Produit_1	Temps_2
Vente_3	Client_4	Produit_5	Temps_5

FIGURE I.10 – Index de jointure en étoile

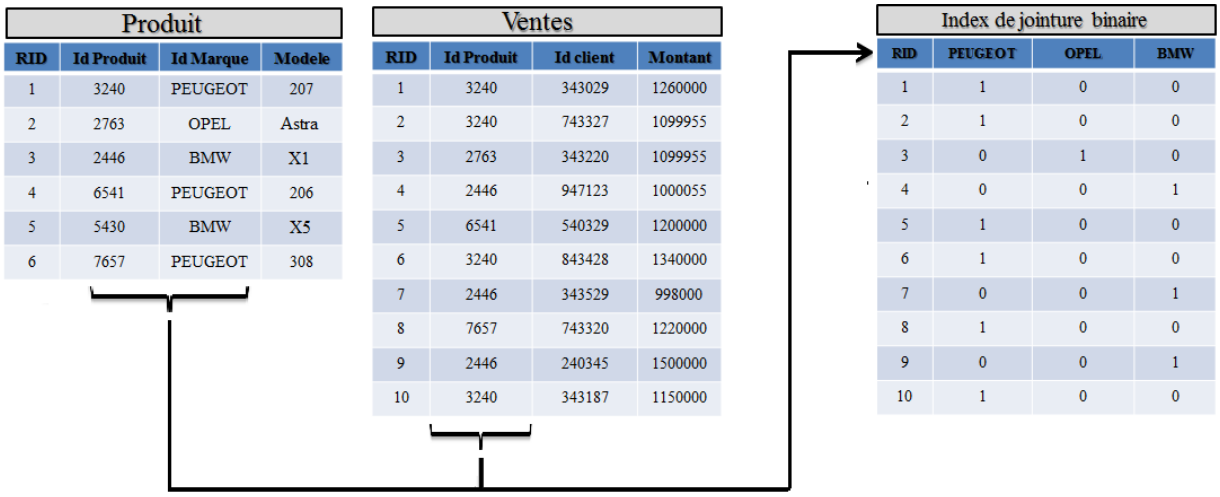
Index de jointure binaire

L'index de jointure binaire (IJB) est une variante des index de jointure, à la différence des index binaires standards où la même table contient les attributs indexés, l'index de jointure (IJB) binaire est défini sur la table des faits en utilisant un ou plusieurs attributs appartenant à plusieurs tables de dimension.

Pour montrer comment un IJB est construit, supposons un attribut A ayant n valeurs distinctes

$v_1; v_2; ...; v_n$ appartenant à une table de dimension D. Supposons que la table des faits F est composée de m instances. La construction de l'index de jointure binaire défini sur l'attribut A se fait de la manière suivante [10] :

1. Créer n vecteurs composés chacun de m entrées ;
2. Le i^{eme} bit du vecteur correspondant à une valeur v_k est mis à 1 si le n-uplet de rang i de la table des faits est joint avec un n-uplet de la table de dimension D tel que la valeur de A de ce n-uplet est égale à v_k. Il est mis à 0 dans le cas contraire.



Produit

RID	Id Produit	Id Marque	Modele
1	3240	PEUGEOT	207
2	2763	OPEL	Astra
3	2446	BMW	X1
4	6541	PEUGEOT	206
5	5430	BMW	X5
6	7657	PEUGEOT	308

Ventes

RID	Id Produit	Id client	Montant
1	3240	343029	1260000
2	3240	743327	1099955
3	2763	343220	1099955
4	2446	947123	1000055
5	6541	540329	1200000
6	3240	843428	1340000
7	2446	343529	998000
8	7657	743320	1220000
9	2446	240345	1500000
10	3240	343187	1150000

Index de jointure binaire

RID	PEUGEOT	OPEL	BMW
1	1	0	0
2	1	0	0
3	0	1	0
4	0	0	1
5	1	0	0
6	1	0	0
7	0	0	1
8	1	0	0
9	0	0	1
10	1	0	0

FIGURE I.11 – Index de jointure binaire

Les vues matérialisées

Les vues matérialisées améliorent le temps de réponse des requêtes les plus fréquente par un stockage physique des résultats,une définition des vues matérialisées est présentée dans [11], comme suit :«les vues matérialisées sont des relations qui contiennent le résultat d'une requête. Elles permettent de pré-calculer des opérations complexes et de les stocker dans la base de données. Ainsi, des requêtes simples nécessitant uniquement un accès aux vues matérialisées, et non plus aux données sources peuvent être exécutées rapidement. Les problèmes majeurs liés à la matérialisation de vues sont : leur sélection et leur maintenance».

Fragmentation verticale

Dans la fragmentation verticale une relation R est divisée en relation $R_1, R_2, ...R_n$ appelées fragments verticaux où chaque relation contient un sous ensemble d'attributs de la relation R avec obligatoirement l'attribut clé ; afin de pouvoir reconstituer la relation R par l'opération de jointure, cette technique est redondante en raison de la duplication de la clé primaire de la table initiale, plus la duplication éventuelle des autres attributs.

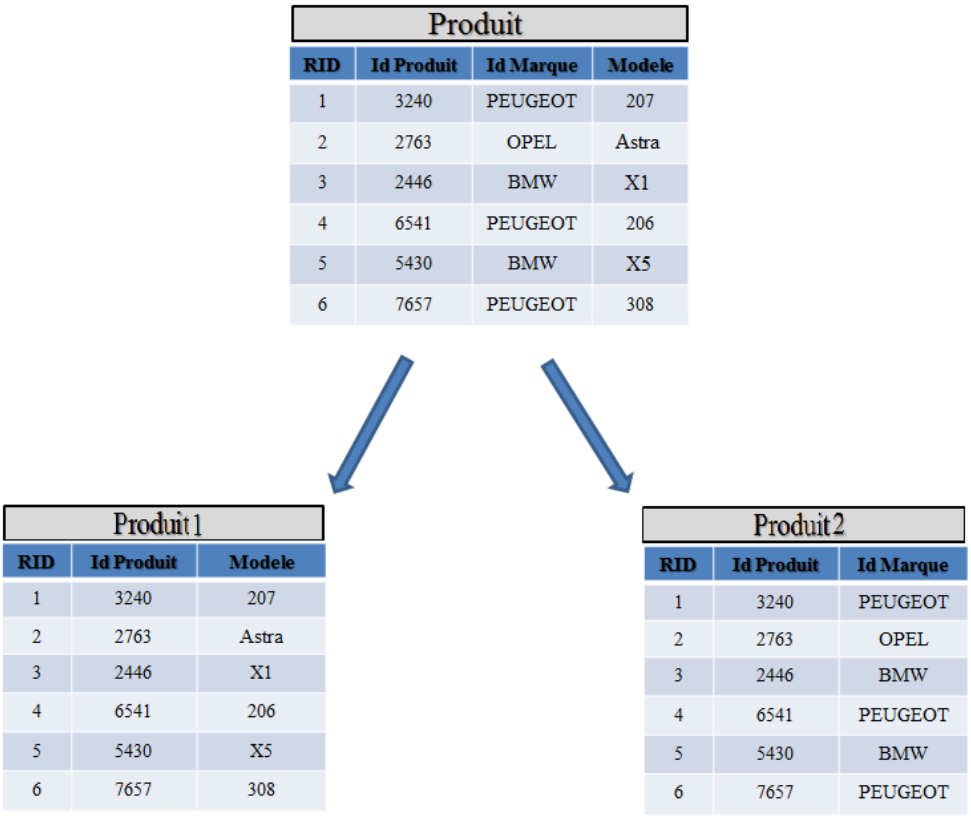


FIGURE I.12 – Fragmentation verticale

I.3.2 Techniques d'optimisation non redondantes

Nous allons présenter dans cette section la fragmentation horizontale et le traitement parallèle, qui ne nécessitent pas un espace de stockage supplémentaire dans le mesure où ces techniques ne dupliquent pas les données à l'inverse des techniques redondantes.

Fragmentation horizontale

La fragmentation horizontale consiste à partitionner un ensemble de données D en plusieurs partitions D_i appelé fragments, les instances appartenant au même fragment vérifient généralement à un prédicat de sélection p_k ($k = 1..r$ avec r le nombre de prédicats de sélection). le résultat d'une opération de restriction sur l'ensemble D donne un fragment horizontal et le résultat de l'opération union sur l'ensemble des fragments horizontaux permet la reconstitution de la relation source D[12].

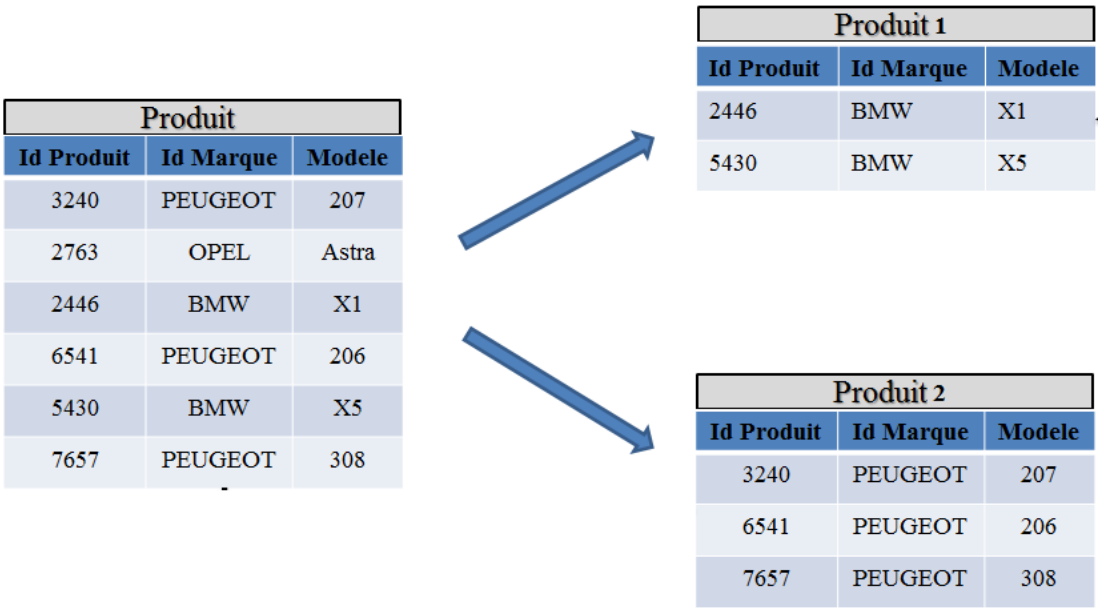


FIGURE I.13 – Fragmentation horizontale de la relation *Produit*

Traitement parallèle

Dans un entrepôt de données, avec la taille gigantesque des données et pour répondre aux requêtes décisionnelles, il est nécessaire d'atteindre la plus grande vitesse de traitement possible ; le traitement parallèle [13] est parmi les solution où l'intérêt réside dans les performances qu'il permet d'atteindre. Le parallélisme présent dans une requête décisionnelle permet d'exécuter simultanément, par des ressources matérielles différentes, plusieurs parties de cette requête en même temps. Les trois architectures les plus utilisées dans le traitement parallèle sont :

1. Mémoire partagée : Les requêtes sont exécutées par plusieurs CPU avec une seule mé-

moire.

2. Disques partagés :Les requêtes sont exécutées par plusieurs CPU avec des mémoires privées à chaque CPU et accèdent aux disques partagés.
3. Sans partage : Dans ce type d'architecture ni la mémoire ni les disques ne sont partagés (tout est privé) ; les requêtes sont exécutées par plusieurs CPU avec des mémoires propres à chaque CPU avec et plusieurs disques privés.

Chapitre II

Problème de sélection d'index dans les entrepôts de données

II.1 Introduction

Un entrepôt de données exige des traitements très complexe où les données sont de plus en plus volumineuses, un administrateur d'entrepôt de données doit choisir une bonne conception physique en sélectionnant un ensemble de techniques d'optimisation afin d'accélérer le temps d'accès et de diminuer l'espace de stockage pour répondre aux requêtes établies par les décideurs de l'entreprise. En effet, et concernant les bases de données, les index simple B-arbre ou les bitmap font leurs preuves. Concernant les entrepôts de données, les index de jointure binaire sont l'une des techniques les plus efficaces.

Nous allons présenter, dans ce chapitre,une étude du problème de sélection d'index de jointure binaire ; une formalisation du problème et en fin un état de l'art des différents travaux antérieurs liés à ce type de problème.

II.2 Problème de sélection des index de jointure binaires

Le problème de sélection d'index est connu NP-Complet [14] ; donc il n'existe aucun algorithme pour une solution optimale en un temps fini. les travaux de recherche de ce thème utilisent des heuristique ou par dataminig pour proposer des solutions proches de la solution optimale et de réduire la complexité du problème.

II.2.1 Formalisation

Le problème de sélection des index de jointure binaire peut être formulé comme suit [26] :

– Etant donné :

1. Un entrepôt de données modélisé par un schéma en étoile formé d'une table de faits $\boldsymbol{F}$ et de tables de dimension $D = \{D_1, ..., D_d\}$
2. Un ensemble de requêtes $Q = \{q_1, ..., q_m\}$ les plus fréquentes avec leurs fréquences d'accès $f = \{f_1, ..., f_m\}$,
3. Une contrainte de stockage $\boldsymbol{S}$.

– L'objectif est de trouver une configuration d'index CI_i en :

– minimisant le coût d'exécution de $\boldsymbol{Q}$:

$$Coût\ (\ CI_i\ ) = MIN\ (\ CI_1\ , CI_2\ \ldots\ CI_n\ )$$

– respectant la contrainte de stockage : $Taille(CI_i) \leq \boldsymbol{S}$

La configuration d'index peut avoir deux types d'index : les index mono-attributs et les index multi-attributs :

– *Les index mono-attributs :* sont définis sur un seul attribut d'une table. Ils sont caractérisés par une seule colonne qui référence les valeurs de l'attribut indexé.

– *les index multi-attributs :* sont définis sur plusieurs attributs issus de plusieurs tables. L'index comporte autant de colonnes que d'attributs indexés et chaque colonne référence les valeurs de chaque attribut.

Complexité :

Soit l'ensemble de N attributs indexable $A = A_1, \ldots, A_n$. Le nombre d'index unique (un seul index) est donné avec la formule[15] :

$$\boxed{I_{mono} = 2^n - 1}$$

Si les index sont multi-attribut, chaque index à un nombre combinatoire de configurations possibles $2^n - 1$ Ainsi le nombre total de configuration d'index multiple (un ou plusieurs index) possible est :

$$\boxed{I_{multi} = 2^{2^n - 1} - 1}$$

II.2.2 Les étapes de sélection d'index

L'approche générique de sélection d'index [4] comporte en générale les étapes suivantes(Figure II.1) :

1. Identification des attributs indexable et élagage de l'espace de recherche :

   après une analyse de la charge de requêtes, la détermination des attributs indexables peut se faire selon deux méthodes :manuelle ou automatique. la première méthode est la sélection manuelle qui fait appelle à l'expérience de l'administrateur pour le choix d'un certain nombre d'attributs candidats pour l'indexation. la deuxième méthode est la sélection automatique repose sur un analyseur syntaxique de la charge de requêtes pour la génération d'un ensemble d'attributs candidats Le but de cette étape est d'élaguer l'espace de recherche des index.

2. Sélection de la configuration finale d'index :

   Le but de cette étape est la détermination de l'ensemble d'index qui va être généré avec le minimum de coût d'exécution des requêtes en respectant l'espace de stockage disponible pour ce faire un modèle de coût est utilisé.

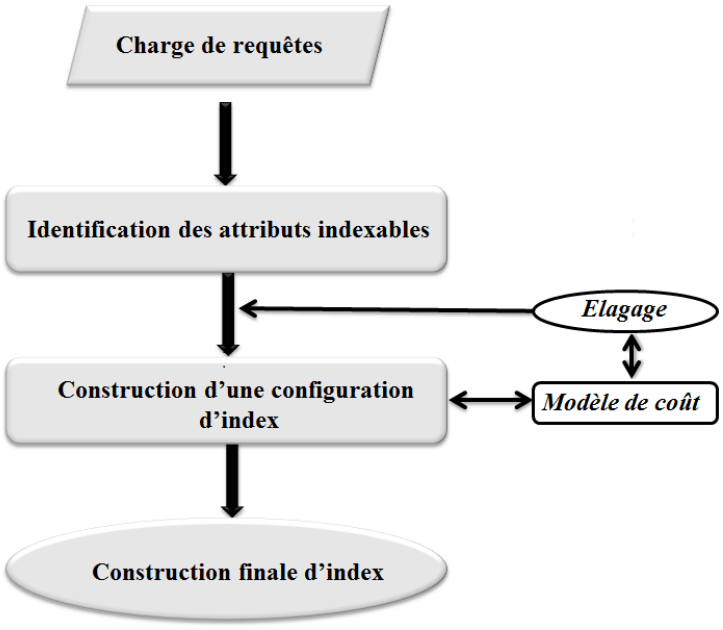


FIGURE II.1 – Approche générique de sélection d'index

II.3 Etat de l'art

Nous présentons dans ce qui suit un état d'art des travaux sur la sélection d'index dans les bases de données et les entrepôts de données.

II.3.1 Travaux de sélection d'index dans les base de données

Travaux de *Chaudhuri et al.*

Dans le but d'auto administration des bases de données ; Microsoft a lancé un projet de recherche pour la mise en œuvre d'un outil d'aide à la résolution du problème de la conception physique afin d'améliorer les performances des bases de données notamment la sélection d'index qui est gérée manuellement par l'administrateur dans ce contexte Chaudhiri et al ont développé l'outil de selection d'index IST (Index Selection Tool) sous Microsoft SQL Server 7.0 qui se déroule en trois étapes pour la sélection d'une configuration comme suit : (figure II.2)[16]

1. *Sélection des index candidats :*

   Dans cette étape ; une analyse de la charge de requête pour détecter les d'attributs existant dans les clauses WHERE,GROUP BY et ORDER BY afin de construire l'ensemble des attributs indexables.

2. *Réduction de l'ensemble d'index :*

   L'élimination d'un certain nombre d'index qui ont un gain de performance faible.à l'aide d'un algorithme glouton et a partir de n index candidats construits les k meilleurs sont sélectionnés. Ce module est en échange permanent avec l'optimiseur de requêtes.

3. *Génération d'index multi-attributs :*

   dans cette étape l'IST utilise les index obtenus dans l'étape précédente et qui sont mono-attribut et Pour construire des index multi-attributs, deux fonctions sont utilisées : MC-LEAD et MC-ALL. La fonction MC-LEAD permet de générer un index multi-attribut en combinant un index mono-attribut avec un attribut indexable (n'est encore indexé). La fonction MC-ALL permet de générer un index multi-attributs en combinant deux attributs mono-attributs. La génération des index multi-attributs de taille supérieure se fait selon le même principe.

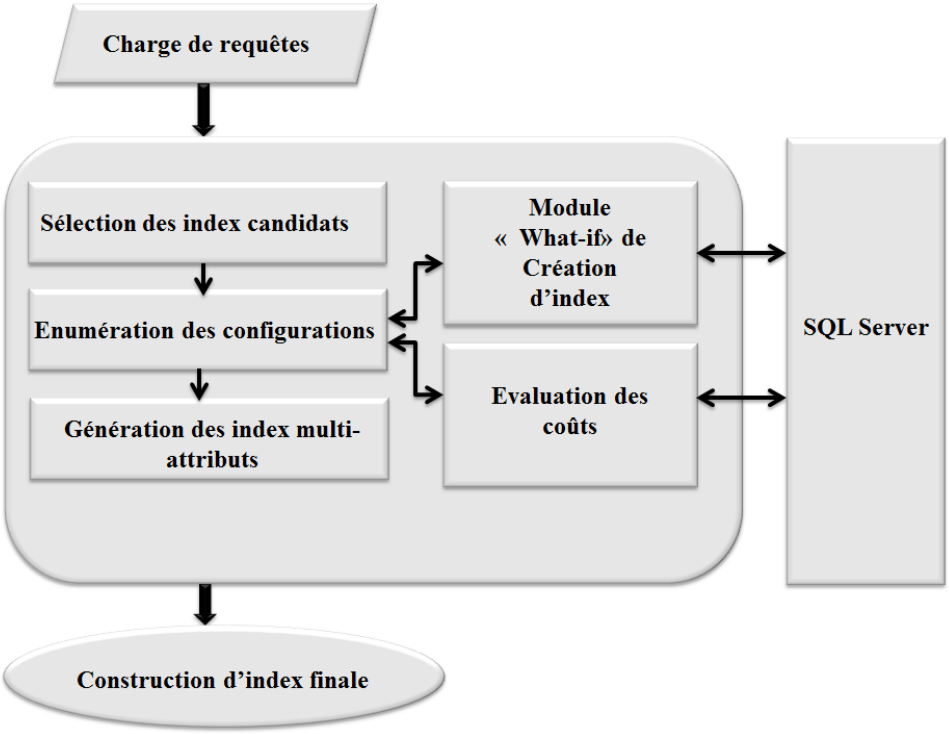


FIGURE II.2 – Architecture de l'outil de sélection d'index ITS

Travaux de *Valentin et al.*

Dans [17] les auteurs proposent une solution pour la sélection d'index sous le système DB2 Advisor d'IBM ; la sélection d'index est établie par un algorithme qui est une extension de l'optimiseur qui génère pour chaque requête le plan d'exécution optimal. Les index utilisés dans ces plans seront recommandés et une fois l'ensemble d'index définit, le problème est modélise comme le problème du sac à dos pour être résolu par heuristiques.

Travaux de *Feldman et al.*

Les auteurs proposent un outil d'assistance à base de connaissance pour la sélection d'index nommé DINNER [18]. A partir des statistiques sur l'ensemble des tables et les différentes requêtes sur ces tables l'outil constitue un modèle de coût, ensuite pour l'ensemble des requêtes, DINNER construit un graphe représentant l'ensemble des solutions possibles qui peut utiliser chaque requête ; Les requêtes de jointures sont décomposées en plusieurs requêtes définies sur une seule table. à partir des meilleurs graphes, on obtient une configuration d'index, avec leurs

coûts et leurs espaces de stockage. l'outil utilise des heuristiques pour trouver, dans cette configuration d'index candidats, les meilleurs index.

Travaux de *kratica et al.*

Dans cette approche *Kratica et al.* proposent un algorithme génétique pour résoudre le problème de sélection d'index [19] ; ils appliquent l'algorithme sur une population d'individus N_{pop}. N_{elit} est le nombre d'individus élus pour survivre à la génération qui suit.La fonction objective à évaluer représente le temps de traitement des requêtes. Elle est stockée dans une table cache de taille *Ncache* pour accélérer le temps des calculs.

II.3.2 Travaux de sélection d'index dans les entrepôt de données

Cette partie est consacrée aux travaux de sélection d'index dans les entrepôt de données selon deux axes : dataminig et métaheuristique .

Travaux de *Golfarelli et al.*

Golfareli et al proposent un algorithme de sélection d'index basé sur un modèle de coût mathématique dont les étapes sont brièvement décrits comme suit :

- Initialisation de l'ensemble des index candidats *I* et l'ensemble des index optimaux *O* ainsi que l'espace de stockage *S* nécessaire pour stocker les index à sélectionner.
- La sélection par un algorithme glouton , à partir de l'ensemble des index candidats *I*, les index apportant un meilleur coût en les ajoutant dans 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 fait[20].

Travaux de *Aouiche et al.*

L'approche *d'Aouiche et Al*[8] comme montre la figure II.3 ; se base sur une technique de fouille de données qui est la recherche des motifs fréquents fermés ou chaque motif est un index de jointure binaire candidat ; pour élaguer l'espace de recherche des index de jointure les auteurs ont choisi d'utilisation de l'algorithme **CLOSE** pour la recherche des motifs fréquents fermés

à partir d'une charge de requêtes d'extraction donnée. Pour arriver à une configuration d'index finale l'approche d'*Aouiche et al.* passe par les étapes suivantes :

1. *Extraction de la charge de requêtes :*

   A partir du journal des transactions du SGBD la charge de requêtes est extraite.

2. *Analyse de la charge :*

   extraction des attributs susceptible d'être des supports d'index par un analyseur syntaxique.

3. *Construction d'un contexte d'extraction :*

   Une matrice "Requêtes-attributs" de taille *nxm* est construite où *m* est le nombre de requêtes et n le nombre des attributs d'index candidats, l'existence d'un attribut indexable dans une requête est symbolisée par *un* et son absence par *zéro* comme montre l'exemple suivant :

   Q1 : A1 A3 A5
   Q2 : A2 A4
   Q3 : A1 A2 A5
   Q4 : A3
   Q5 : A2 A3 A4 A5

   La matrice dans ce cas est :

	A1	A2	A3	A4	A5
Q1	1	0	1	0	1
Q2	0	1	0	1	0
Q3	1	1	0	0	1
Q4	0	0	1	0	0
Q5	0	1	1	1	1

TABLE II.1 – Matrice requête-attribut

4. *Génération des index candidats :*

   L'algorithme CLOSE est appliqué sur le contexte d'extraction pour obtenir des motifs

fréquents. Chaque motif extrait est composé d'un ensemble d'attributs de l'entrepôt de données.

5. *Construction de la configuration d'index finale :*

   A partir de l'ensemble d'index généré dans l'étape précédente, un algorithme glouton est appliqué pour sélectionner une configuration d'index finale.

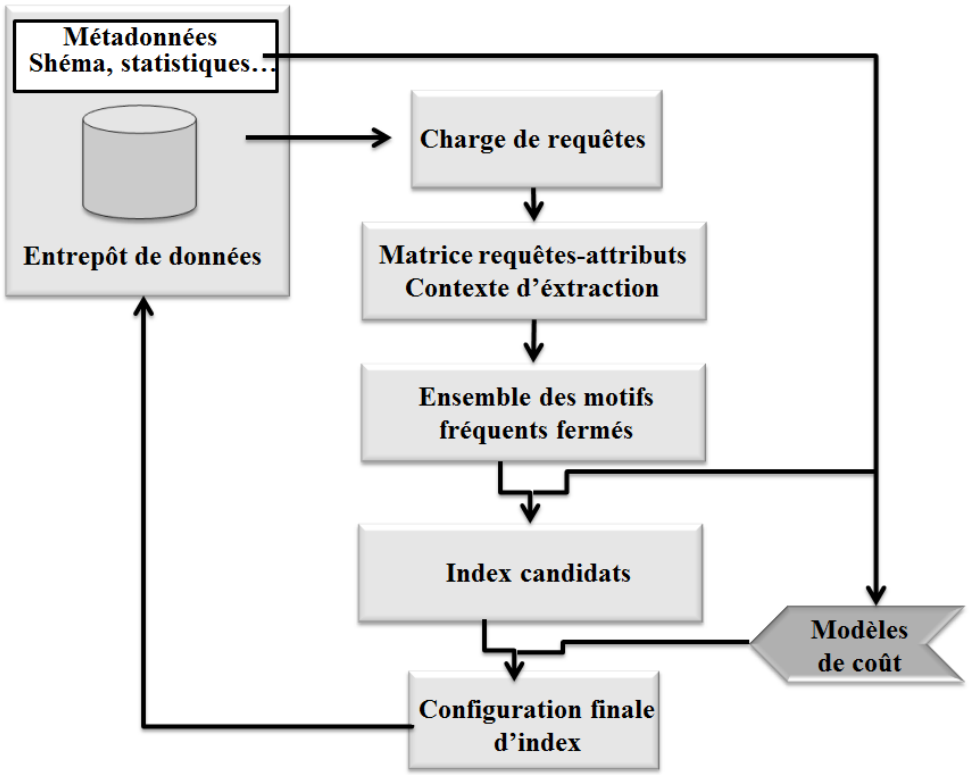


FIGURE II.3 – Architecture de fonctionnement de l'approche de Aouiche

Travaux de *Bellatrache et al.*

L'approche de Bellatreche et al. se base sur la technique de motifs fréquent fermé et proposent d'ajouter d'autres paramètres comme la taille des tables de dimension,la taille de la page système, etc. vu que l'approche de Aouiche et al. peut éliminer des index sur des attributs non fréquemment utilisés mais qui appartiennent à des tables de dimension volumineuses, ce qui ne permet pas d'optimiser une opération de jointure. les auteurs ont montré à travers un exemple que la fréquence d'accès seule ne permet pas de sélectionner un ensemble d'index efficace. Ils proposent donc les algorithmes *DynaClose* et *DynaCharm* qui prennent en compte

ces paramètres[21]. Ces algorithmes utilisent une nouvelle métrique appelée *Fitness* qui utilise, outre la fréquence, les tailles des tables ; pour générer les attributs candidats ; cette approche utilise une méthode de Datamining pour réduire l'espace de recherche après avoir récupérer les attributs indexable et pour sélectionner une configuration d'index finale un algorithme glouton est utilisé , il se base sur un modèle de coût pour définir la fonction objective afin d'évaluer le temps d'exécution de la charge de requête et le coût de stockage de l'index.afin de pénaliser les attributs fréquents définis sur des tables de petites tailles.

Travaux de *Boukhalfa et al.*

L'approche de *Boukhalfa et al.* consiste à prendre en entrée un schéma de l'entrepôt, une charge de requêtes Q de m requêtes les plus fréquentes $Q_1, Q_2, ..., Q_m$ et un quota d'espace disque et renvoie en sortie une configuration d'IJB multi-attributs permettant de réduire le coût d'exécution de la charge sans dépasser le quota d'espace de stockage[4].
Cette approche de sélection se divise en quatre étapes :

- *Identification des attributs indexables :*

  A partir de la charge des requêtes. Tout attribut sur lequel un prédicat de sélection est défini et possédant une cardinalité faible ou moyenne est considéré comme attribut indexable. L'ensemble des attributs indexables est l'union de tous les attributs indexables identifiés sur chaque requête.
- *Construction d'une configuration par requête :*

  Pour chaque requête de la charge des requêtes ; l'index qui couvre tous les attributs indexables utilisés par cette requête est sélectionné.
- *Construction d'une configuration initiale :*

  la configuration initiale est construite à partir de l'union des IJBs obtenus séparément dans l'étape précédente.
- *Construction d'une configuration finale :*

  Pour réduire le coût d'exécution avec le respect de la contrainte de stockage , des éliminations sont fait sur des attributs de la configuration initiale par quatre stratégies dans cette approche à savoir : *élimination des attributs de forte cardinalité*,Attributs *appartenant aux tables moins volumineuses*,*Attributs les moins utilisés*,*Attribut apportant moins de réduc-*

tion de coût , dans le dernier cas un modèle de coût est utilisé pour choisir les meilleures configurations.

Travaux de *Ziani et al.*

L'objectif des travaux de *Ziani et al.* est de trouver une configuration d'index *CI* pour minimiser le coût d'exécution de Q et respectant la contrainte de stockage $(Taille(CI) \leq S)$ l'approche est basée sur la recherche des motifs fréquents maximum. En effet, vu le nombre important des motifs générés par la technique des motifs fréquents fermés, les auteurs proposent l'exploitation les motifs fréquents maximaux. Ces derniers sont moins nombreux que les motifs fréquents fermés ce qui va réduire l'espace de recherche des index candidats. En plus, un motif fréquent maximal à la particularité que tous ses sur-ensembles sont non fréquents. Cette particularité va permettre la génération des index moins nombreux et plus pertinents. pour réaliser leur technique ils considèrent une charge de requêtes, supposée représentative, de l'activité du système ; pour dégager un contexte d'extraction et pour appliquer l'algorithme $fpmax$ (implémenté en java) pour extraire les index candidats. Les tableaux 3 et 4 présentent respectivement des exemples d'une base de transactions et de motifs fréquents maximaux recherchés dans cette base.

TID	**Items**
1	ACTW
2	CDW
3	ACTW
4	ACDW
5	ACDTW
6	CDT

Taille du motif maximal	**Motif Maximal(*min sup* = 2)**
1	-
2	-
3	CDT
4	ACDW,ACTW

TABLE II.2 – Exemple de motifs fréquents maximaux

L'approche proposée est constituée des étapes suivantes :

1. *Extraction de la charge de requête :*
   sélection d'une charge de requêtes, supposée représentative, de l'activité du système ;
2. *Construction du contexte d'extraction :*
   structuration des attributs indéxables contenus dans la charge sous forme d'une base tran-

sactionnelle où les requêtes représentent les transactions et les attributs représentent les motifs. Ceci représente notre contexte d'extraction des motifs fréquents maximaux ;

3. *Construction de l'ensemble d'index candidats :*
   génération des index candidats par l'algorithme *fpmax* implémenté en java ;

II.3.3 Synthèse des travaux de sélection d'index

Nous avons présenté dans la section précédente les différents travaux de sélection d'index dans les bases de données et les entrepôts de données ; dans cette section, le tableau II.3 résume ces principaux travaux selon des critères à savoir : le type de sélection des index candidats automatique ou manuelle ; l'algorithme de sélection d'index finale et en fin le modèle de coût utilisé [4].

	Travaux	Sélection des index candidats	Type d'index sélectionné	Algorithme de sélection finale d'index	Modèle de coût
BDD	Chaudhuri et al. 1997	Automatique	Général sur seule table	Glouton (Ascendant)	Optimiseur et "What-IF"
	Valentin et al. 2000	Automatique	Général sur seule table	Heuristique (Sac à dos)	Optimiseur
	Feldman et al. 2003	Automatique	Général sur seule table	Heuristique (graphe d'exécution)	Mathématique
	Kratica et al. 2003	Manuelle	Général sur seule table	Heuristique (Génétique)	Mathématique
Entrepôt de données	Golfareli et al. 2002	Automatique	liste de valeurs et bitmap	Glouton (Ascendant)	Optimiseur
	Aouiche et al 2005	Automatique par Datamining (motifs fermés)	Index de jointure binaire	Glouton (Ascendant)	Mathématique
	Bellatreche et al. 2007	Automatique par Datamining (motifs fermés + adaptation)	Index de jointure binaire	Glouton (Ascendant)	Mathématique
	Boukhalfa et al. 2010	Automatique	Index de jointure binaire	Heuristique (Élimination d'attribut)	Mathématique
	Ziani et al. 2010	Automatique par Datamining (motifs maximaux)	Index de jointure binaire	-	Mathématique

TABLE II.3 – Synthèse des différents travaux de sélection d'index

Chapitre III

Nouvelle approche de sélection d'index de jointure binaire basée sur la recherche des règles d'association

III.1 Introduction

Le problème de sélection d'index consiste à construire une configuration d'index améliorant le coût d'exécution sous la principale contrainte qui est l'espace de stockage réservé. La démarche générique consiste à sélectionner automatiquement un ensemble d'index candidats à partir d'une charge de requêtes ou en faisant appel à l'expertise de l'administrateur. Étant donnée que théoriquement les règles d'association donnent des résultats plus affinés que les motifs fréquents [22] ; nous avons adapté le problème de sélection des index pour être un problème typique de recherche de règles d'association. Il s'agit d'une recherche automatique pour découvrir des relations avec corrélations intéressantes entre les attributs dans une charge de requête.

La recherche de règles d'association est une approche automatique. l'administrateur n'intervient que pour fixer les seuils minimaux de *support* et de *confiance* qui vont être utilisés durant le processus de la recherche des règles d'association. La variation de ces deux paramètres entraine l'augmentation ou la diminution du nombre de règles d'association produites ; exactement le même effet pour le support minimal dans les motifs fréquents.

Les règles d'associations sont appliquées dans plusieurs domaines. En marketing, par exemple, elles permettent d'identifier les services ou les produits qui sont achetés lors d'une même tran-

saction ou par un même client dans le temps et elles offrent ainsi la possibilité d'identifier des opportunités de ventes croisées. En analysant l'ordre dans lequel les internautes accèdent aux pages d'un site WEB, les règles d'associations permettent d'entrevoir quelles modifications rendraient le site plus convivial et permettant ainsi aux internautes de trouver rapidement les informations recherchées. Les règles d'associations sont également utiles dans l'étude des comportements d'achats des consommateurs, car elles permettent de mettre en lumière les comportements d'achats à travers leurs habitudes dans le temps.

L'extraction des règles d'association est une technique très répandue dans le domaine de fouille de données, il s'agit d'implication conditionnelle permettant de trouver des corrélations entre des éléments qui sont liés par une relation. Dans le cas des bases de données transactionnelles, étant donné un ensemble d'articles, le but est de découvrir si l'occurrence de cet ensemble dans une transaction est associée à l'occurrence d'un autre ensemble d'articles qui est dans notre contexte l'apparition des attributs dans une charge de requête.

Dans notre approche, le but est de trouver une configuration d'index par la recherche de l'ensemble de règles d'association à partir d'un ensemble de motifs fréquents afin de ressortir les attributs les plus corrélés à partir d'un contexte d'extraction qui reflète l'ensemble des requêtes supposées représentatives du système ce qui nous donne une configuration d'index.

Nous présentons, dans ce chapitre, le principe de cette nouvelle approche. Nous commençons en premier temps par montrer la motivation de notre choix de sélection des index par les règles d'association ; par la suite, nous présentons la définition du cadre formel d'extraction des règles d'association à savoir la définition d'une règle d'association ainsi que ces éléments constitutifs comme les motifs fréquents ; les mesures telle que *support* et *confiance*, nous présentons l'algorithme de base *Apriori* ; et en fin nous détaillerons la mise en œuvre de notre approche avec explication des différents étapes.

III.2 Les règles d'association

L'extraction des règles d'association est une technique de fouille de données qui permet de découvrir des relations significatives entre attributs ou événements qui occurrent de manière simultanée dans une base de données transactionnelles. Une règle d'association décrit une cor-

rélation entre des ensembles d'items dans une base de transaction. Autrement dit, étant donné un ensemble d'items, le but est de découvrir si l'occurrence de cet ensemble dans une transaction est associée à l'occurrence d'un autre ensemble d'items. Actuellement avec un nombre croissant de grandes bases de données et pour déterminer la façon dont les données sont organisées et en extraire des informations utiles ; les règles d'association représentent un outil efficace et performant pour ce type de problème. Il a été introduit en 1993 par Agrawal et al. dans [22]. L'une des premières applications des règles d'association a fait l'étude du panier de la ménagère ; elle permet d'analyser les achats des clients à travers les enregistrements des tickets de caisses afin de comprendre leurs habitudes de consommation ce qui conduit à mieux organiser les promotions, gérer les stocks, etc. La technique consiste à mettre en évidence des règles de la forme :$\{prémisse \rightarrow conclusion\}$. C'est à dire de type $\{\ IF\ C\ THEN\ P\ \}$ où C est la condition de la règle et P sa prédiction. L'objectif est de découvrir les règles d'association les plus pertinentes, les plus explicatives. Dans la section suivante quelques définitions très utiles avant de présenter les concepts des règles d'association.

III.2.1 Définitions

Cette section définit les principaux concepts liés à la tâche d'extraction des règles d'association.

Item :

Un item est un objet, article ou attribut, appartenant à un ensemble fini d'éléments distincts $I = \{i_1; i_2; ...; i_n\}$ i_i est un item.

Transaction :

On considère un ensemble $I = \{i_1; i_2; ...; i_n\}$ de n éléments (items) distincts. On appelle une transaction T le sous ensemble $I' \subset I$.

ItemSet

Un itemset est tout sous-ensemble d'items de I.

K-Itemset

Un K-Itemset est un itemset contenant K-items.

Support d'un itemset

Le support d'un itemset ***X*** est le nombre de transactions de la base des transactions ***T*** contenant ***X*** sur le nombre total des transactions.

$$\boxed{sup(X) = \frac{Card\{t \in T / X \subseteq t\}}{Card(T)}}$$

Itemset fréquent

Etant donné un seuil de support minimum S_0, un itemset X est dit fréquent dans une base de transactions, si son support dépasse le seuil fixé S_0.

III.2.2 Formalisation d'une règle d'association

Si on considère :

– $I = \{i_1; i_2; ...; i_n\}$ un ensemble de ***n*** items

– X et Y deux sous-ensembles d'éléments qui appartiennent à un ensemble $I = \{i_1; i_2; ...; i_n\}$.

On appelle une règle d'association entre X et Y la relation de dépendance entre les items de X et ceux de Y. Une règle d'association est de la forme $X \rightarrow Y$ telle que :

$$X \subseteq I \text{ et } Y \subseteq I$$

$$X \cap Y = \phi$$

Une règle d'association comporte donc une partie **prémisse** composée d'un ensemble d'items X et une partie **conclusion** également composée d'un ensemble d'items Y, disjoint de X.

III.2.3 Mesures d'une règle d'association

Les mesure de qualité des règles servent à déterminer la qualité individuelle d'une règle selon différents aspects orientés utilisateur. Les mesures de base utilisés sont le support et la confiance [23]. qui sont définie dans ce qui suit :

Support d'une règle d'association :

Le support de la règle $X \longrightarrow Y$ définit le pourcentage des transactions qui contiennent $\mathbf{X} \cup \mathbf{Y}$ par rapport au nombre total des transactions dans la base. Le support indique la portée de la règle ou aussi la proportion des transactions où les items $\mathbf{X} \cup \mathbf{Y}$ se trouvent ensemble, par rapport à l'ensemble des transactions.

$$\boxed{sup(X \longrightarrow Y) = \frac{Card\{t \in T / X \cup Y \subseteq t\}}{Card(T)}} \quad \text{(III.1)}$$

Plus le seuil de support est élevé, on obtient moins de règles, par contre les règles obtenues sont plus solides.

Confiance d'une règle d'associations :

La confiance d'une règle est la proportion des transactions dont les items de $X \cup Y$ se trouvent ensemble, par rapport aux transactions dont les items de $\mathbf{X}$ se trouvent aussi ensemble.

$$\boxed{conf(X \longrightarrow Y) = \frac{Card\{t \in T / X \cup Y \subseteq t\}}{Card\{t \in T / X \subseteq t\}}} \quad \text{(III.2)}$$

Une règle est de meilleure qualité quand sa confiance est plus grande.

III.2.4 Construction des règles d'association

L'extraction des règles d'association consiste à trouver l'ensemble des règles intéressantes, respectant un seuil minimal de support *minsup* et à un seuil minimal de confiance *minconf* prédéfinis par l'utilisateur.

La recherche des règles d'association considère en entrée :

- Un ensemble T avec $Card(T) = m$ dont les éléments sont appelés : *transactions*.
- Un ensemble I avec $Card(I) = n$ dont les éléments représentent les *items*.

Le problème de la découverte de toutes les règles d'association consiste dans un premier temps à trouver tous les ensembles (*itemsets*) qui ont le support des transactions au-dessus de la valeur *minsup* (défini par l'utilisateur) ; par un parcours de tous les itemsets de la base transactionnelle(exemple III.1), ce sont donc les *itemsets* fréquents ; les autres *itemsets* sont appelés peu

fréquents. L'ensemble des motifs fréquents trouvés sont utilisés pour générer les règles d'association souhaitées ayant les confiances au-dessus de la valeur *minconf* (défini par l'utilisateur).

Le tableau suivant illustre un exemple d'une base de transaction constituée d'un ensemble I de huit items et un ensemble T de six transactions où chacune est identifié par un numéro de transaction.

Items
A B C D E F G H

N Transaction	**Items**
1	A B D F
2	B C E
3	A B C D
4	A E G
5	B D F
6	B C D F G

TABLE III.1 – Base de transactions

Pour la génération de toutes les règles d'association intéressantes, plusieurs algorithmes ont été proposés. Nous avons choisi d'appliquer l'algorithme *Apriori* proposé par Agrawal dans [22], qui est l'algorithme de base de tous les autres algorithmes.

Algorithme *Apriori*

L'algorithme *Apriori* est l'un des algorithmes de recherche de règles d'association les plus connus ; il s'applique à des bases de données contenant des transactions où *Apriori* est capable de sortir des règles liant les transactions entre elles. La façon de définir les règles intéressantes est de spécifier les seuils minimums S_0 sur le *support* et C_0 sur la *confiance*. L'algorithme *Apriori* fonctionne en deux étapes :

Etape 1 : Recherche des sous ensemble fréquents ayant un support supérieur à S_0.

Etape 2 : Construction de toutes les règles ayant une confiance supérieure à C_0.

Algorithm 1: Algorithme Apriori pour la recherché des K-itemsets fréquents

ENTREE : T Ensemble des transactions ; S_0 : Support minimum;
SORTIE : Ensemble des k-itemsets fréquents.;
$L_1 \longleftarrow \{1 - itemsfrquents\}$;
$K \longleftarrow 2$;
tant que $(L_{k-1} \neq \emptyset)$ **faire**
    $C_k \longleftarrow Apriori\text{-}Gen(L_k)$;
    pour *(chaque* $t \in T$*)* **faire**
        $C_t \longleftarrow Subst(C_k, t)\{les\ candidats\ contenus\ dans\ C_k\}$;
        pour *(chaque* $t \in T$*)* **faire**
            c.count++;
    $L_k \longleftarrow \{c \in C_t / c.count \geq S_0\}$;
    K++;
$Return \cup L_k$;

Pour réduire le coût de recherche d'ensemble d'items fréquents, l'algorithme *Apriori* repose sur les propriétés suivantes :

Propriété 1 : Les itemsets construits à partir d'itemsets non fréquents sont non fréquents.

Propriété 2 : Un sous-itemsets d'un itemset fréquent est fréquent.

Propriété 3 : Si Les itemsets sont triés et si **Z** est un K-itemsets candidat, alors il existe deux (k-1)-itemsets X et Y partageant leurs (k-2) premiers items tel que $Z = X \cup Y$.

Exemple d'application

Dans cet exemple la base de transaction représente les requêtes et les items représentent les attributs ; supposons que les requêtes sont numérotées $1, 2, 3\ldots$ et que les attributs sont désignés par $A, B, C, \ldots$ la figure III.4 est un exemple de contexte d'extraction pour la recherche de règles d'association. Appliquons maintenant l'algorithme *Apriori* à cet exemple :

N° Requête	Attributs
1	{ A , B , C , G }
2	{ A , B }
3	{ A , C , D }
4	{ C , D , E , G }
5	{ C , E , F }
6	{ A , C , F }
7	{ C , D , E }
8	{ B , D , H}
9	{ B , D , E }
10	{ B , E }

TABLE III.2 – Ensemble d'une base de transactions

Recherche des sous-ensembles fréquents :

Cette étape consiste à rechercher tous les sous-ensembles d'attributs apparaissant simultanément dans au moins $< S_0 \times m >$ requêtes où m est le nombre de requêtes à analyser (10 dans notre exemple), selon le principe suivant :

On note L_k l'ensemble des sous-ensembles de I de taille k et de support $sup \geq S_0$ dans T.

1. Rechercher dans un premier temps l'ensemble L_1 de tous les attributs apparaissant dans au moins $< S_0 \times m >$ requêtes.
2. L'algorithme se poursuit d'une manière itérative, en construisant l'ensemble C_{k+1}, des candidats pour former l'ensemble L_{k+1}. C_{k+1} est construit à partir de L_k par une jointure de L_k sur lui même sur les colonnes ($Item_1$,$Item_2$,...,$Item_{k-1}$). L_{k+1} est obtenu par élimination de C_{k+1} des $(k+1)$-uplets ne vérifiant pas la contrainte du support minimum en deux temps :
   - En éliminant d'abord les $(k+1)$-uplets dont un sous-ensemble de k éléments n'est dans L_k.

- En parcourant l'ensemble des transactions pour éliminer les $(k+1)$-uplets restant ne vérifiant pas la contrainte de support minimum.

3. L'algorithme s'arrête lorsque l'ensemble L_k est vide.

On commence par construire l'ensemble L_1. Pour ce faire, on compte pour chaque attribut le nombre de requêtes dans lesquelles il apparaît et on élimine les attributs en dessous du seuil fixé. Ainsi la table L_1 contient au plus n éléments, où n est le nombre des attributs.

Si on fixe le support minimum à 20%, c'est-à-dire un minimum de $0.2x10 = 2$ requêtes dans lesquelles un attribut doit apparaître, alors l'ensemble L_1 est représenté par le tableau III.3. L'attribut H n'a pas été retenu car il n'est apparus que dans une seule requête.

Items	Fréquences
A	4
B	5
C	6
D	5
E	5
F	2
G	2

TABLE III.3 – Ensemble L1

Pour construire l'ensemble L_2, on commence par la construction d'un ensemble C_2 contenant des candidats pouvant appartenir à L_2.C_2 comme L_2 comporte deux colonnes et son contenu est obtenu en faisant le produit cartésien de L_1 sur lui-même, en enlevant les couples correspondant à des sous-ensembles équivalents par symétrie. L'ensemble C_2 est représenté par le tableau III.1.

Ensemble C_2

Attribut 1	Attribut 2	Fréquence
A	B	2
A	C	3
A	D	1
A	E	0
A	F	1
A	G	1
B	C	1
B	D	2
B	E	2
B	F	0
B	G	1
C	D	3
C	E	3
C	F	2
C	G	2
D	E	3
D	F	0
D	G	1
E	F	1
E	G	1
F	G	0

Ensemble L_2

Attribut 1	Attribut 2	Fréquence
A	B	2
A	C	3
B	D	2
B	E	2
C	D	3
C	E	3
C	F	2
C	G	2
D	E	3

FIGURE III.1 – Construction de l'ensemble L2

L'ensemble L_2 (Figure III.1) est obtenu en éliminant de C_2 les sous-ensembles qui n'ont pas pu apparaître que dans une seule requête ou aucune.

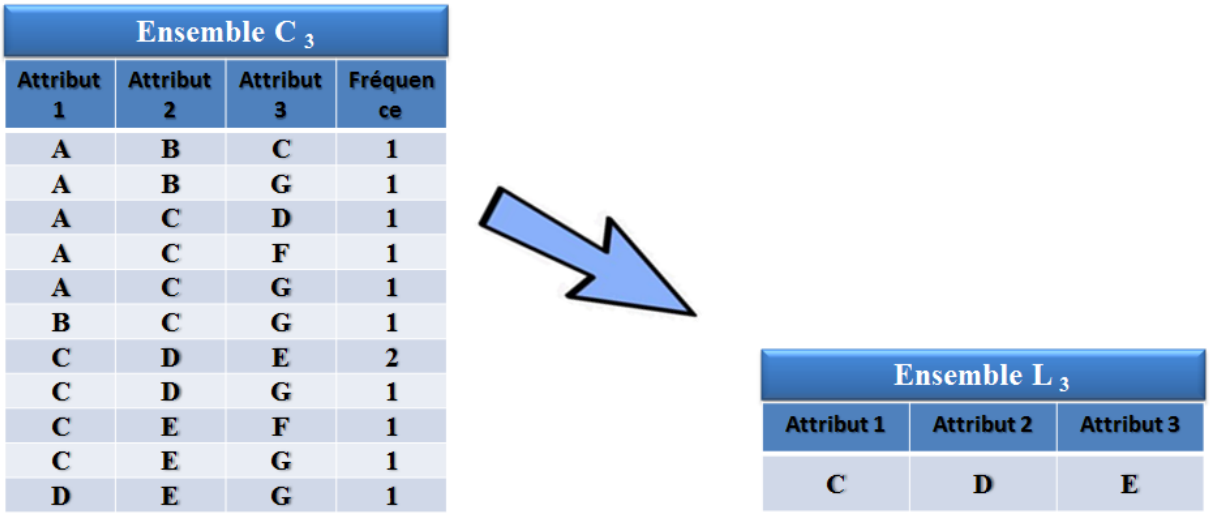
D'une manière itérative, on construit les ensembles L_k définies par :

$$L_K = \{\{i_1,...,i_k\} \in P_k / Car(\{t \in T / \{i_1,...,i_k\} \subseteq t\} \geq 2)\}$$

Avec

$$P_k = \{A \subseteq I / Card(A) = k\}$$

Pour construire l'ensemble L_3. On commence par la construction d'un ensemble C_3 contenant les éléments candidats pour appartenir à L_3. L'ensemble C_3 comporte trois colonnes (Attribut1,Attribut2,Attribut3) et son contenu est obtenu en faisant une jointure de L_2 sur lui-même, sur la colonne (Attribut1). Ce qui donne pour notre exemple : la figure III.2.



Ensemble C_3

Attribut 1	Attribut 2	Attribut 3	Fréquence
A	B	C	1
A	B	G	1
A	C	D	1
A	C	F	1
A	C	G	1
B	C	G	1
C	D	E	2
C	D	G	1
C	E	F	1
C	E	G	1
D	E	G	1

Ensemble L_3

Attribut 1	Attribut 2	Attribut 3
C	D	E

FIGURE III.2 – Construction de l'ensemble L3

A la différence de l'itération précédente, pour former L_3 l'épuration de C_3 se fait en deux temps :

- On supprime de C_3 les triplets pour lesquels le couple (Attribut2,Attribut3) n'est pas dans L_2. Cette suppression peut se faire sans avoir parcourir l'ensemble des requêtes, mais seulement L_2.
- On parcourt l'ensemble des requêtes pour éliminer les triplets qui ne vérifient pas la condition de *support minimum*.

On obtient ainsi l'ensemble L_3 (figure III.2). L'algorithme s'arrête car on voit facilement que L_4 est vide. Les sous-ensembles fréquents sont représenté par le tableau III.4.

Sous-ensemble	Support
{ A }	4 / 10
{ B }	5 / 10
{ C }	6 / 10
{ D }	5 / 10
{ E }	5 / 10
{ F }	2 / 10
{ G }	2 / 10

Sous-ensemble	Support
{ A , B }	2 / 10
{ A , C }	3 / 10
{ B , D }	2 / 10
{ B , E }	2 / 10
{ C , D }	3 / 10
{ C , E }	3 / 10
{ C , F }	2 / 10
{ C , G }	2 / 10
{ D , E }	3 / 10

Sous-ensemble	Support
{ C, D, E }	2 / 10

TABLE III.4 – Les sous-ensembles fréquents

Construction des règles

Pour chaque sous-ensemble fréquent de support supérieur ou égal à $< S_0 \ x \ m >$ trouvé dans l'étape précédente, on construit des règles vérifiant la contrainte de seuil sur la confiance.

Soit f un sous-ensemble fréquent d'attributs, c'est-à-dire tel que son support $sup(f)$ soit supérieur à S_0. Soit g un sous-ensemble de f. On peut produire la règle $(g \rightarrow (f-g)$, à condition que la confiance de la règle soit supérieure au seuil C_0. C'est à dire $conf(g \rightarrow (f-g)) \geq C_0$.

$$\textbf{conf}(g \rightarrow (f-g)) = \frac{Card(\{t \in T / g \cup (f-g) \subseteq t\}}{Card(\{t \in T / g \subseteq t\}} \tag{III.3}$$

d'où

$$\textbf{conf}(g \rightarrow (f-g)) = \frac{Card(\{t \in T / f \subseteq t\}}{Card(\{t \in T / g \subseteq t\}} \tag{III.4}$$

et donc

$$\textbf{conf}(g \rightarrow (f-g)) = \frac{Sup(f)}{Sup(g)} \tag{III.5}$$

Pour produire des règles à partir d'un sous-ensemble fréquent f consiste donc à considérer tous

les sous-ensembles possibles g de f et de produire une règle $g \rightarrow (f-g))$ si la contrainte sur la confiance est vérifiée.

Ensemble	Règle	Sup.	Conf.
{ A , B }	A →B	0,2	0,5
	B →A	0,2	0,4
{ A , C }	A →C	0,3	0,75
	C →A	0,3	0,5
{ B , D }	B →D	0,2	0,4
	D →B	0,2	0,4
{ B , E }	B →E	0,2	0,4
	E →B	0,2	0,4
{ C , D }	C →D	0,3	0,5
	D →C	0,3	0,6
{ C , E }	C →E	0,3	0,5
	E →C	0,3	0,6
{ C , F }	C →F	0,2	0,33
	F →C	0,2	1
{ C , G }	C →G	0,2	0,33
	G →C	0,2	1
{ D , E }	D →E	0,3	0,6
	E →D	0,3	0,6
{ C , D , E }	C →{ D , E }	0,2	0,33
	D →{ C , E }	0,2	0,4
	E →{ C , D }	0,2	0,4
	{ D , E } →C	0,2	0,66
	{ C , E } → D	0,2	0,66
	{ C , D } → E	0,2	0,66

⇒

Règle	Sup.	Conf.
A →B	0,2	0,5
A →C	0,3	0,75
C →A	0,3	0,5
C →D	0,3	0,5
D →C	0,3	0,6
C →E	0,3	0,5
E →C	0,3	0,6
F →C	0,2	1
G →C	0,2	1
D →E	0,3	0,6
E →D	0,3	0,6
{ D , E } →C	0,2	0,66
{ C , E } → D	0,2	0,66
{ C , D } → E	0,2	0,66

TABLE III.5 – L'ensemble des règles d'association

si une règle $g \rightarrow (f-g))$ vérifie la contrainte de confiance, alors pour toute partie g' de g, la règle $g' \rightarrow (f-g'))$ vérifie nécessairement aussi la contrainte de confiance. D'où l'algorithme produit d'abord toutes les règles avec un seul conséquent qui vérifient la contrainte de confiance, puis à partir de celles-ci, toutes les règles avec deux conséquents,etc. En appliquant l'algorithme à notre exemple, nous obtenons le tableau III.5(première table).

Par exemple la confiance de la règles $A \rightarrow B$ s'obtient en divisant le support de l'ensemble A , B par le support de A .

Si on choisit un seuil de $C_0 = 0.5$ pour la confiance des règles, on obtient le résultat final représenté par le tableau III.5 (deuxième table).

III.3 Démarche générale de la solution proposée

La figure III.3 représente l'architecture générale de notre approche qui exploite une charge de requête pour extraire une meilleure configuration d'index multi-attributs. Le schéma met en évidence les principaux axes de notre solution qui sont :

- Sélection et préparation des données.
- Recherche des règles d'association.
- Génération de la configuration finale des index.

Dans notre approche, les transactions sont des requêtes et les items sont les attributs extraits de ces requêtes ; pour la rechercher les règles d'association. Dans ce qui suit nous montrons en détail chacun de ces axes.

III.3.1 Sélection et préparation des données

Extraction de la charge des requêtes

Le système décisionnel sauvegarde dans le journal des transactions toutes les opérations réalisées durant une période déterminée par l'administrateur ; généralement cette période est suffisante pour enregistrer les requêtes les plus fréquentes exécutées par les utilisateurs durant l'activité du système. Dans cette étape nous procèdent à l'extraction de la charge des requêtes à partir du journal des transactions ; cette charge représente les requêtes les plus utilisée dans le système décisionnel.

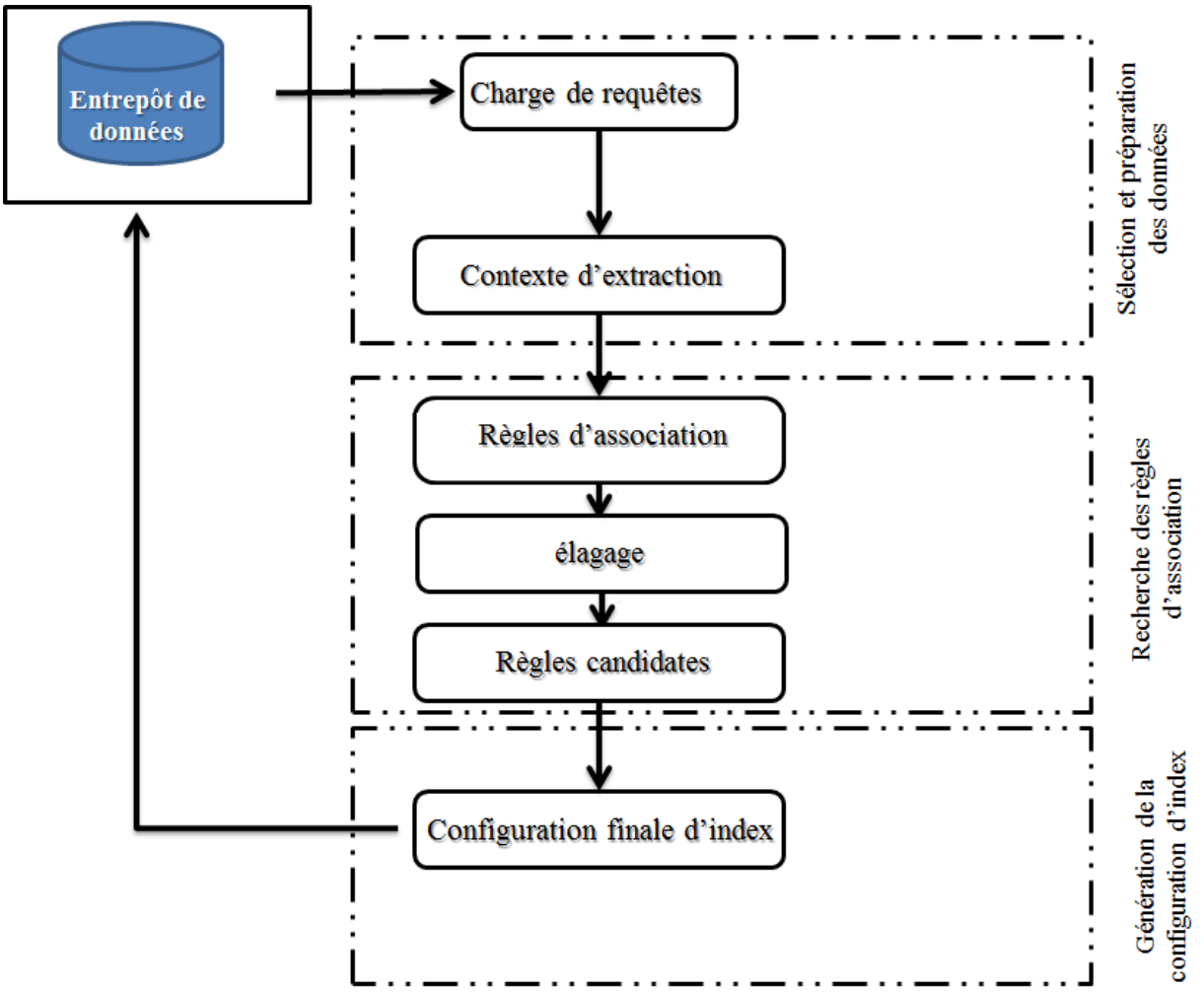


FIGURE III.3 – Architecture générale de la solution

Analyse de la charge

Après extraction de la charge de requêtes du journal de transaction, nous procédons à une analyse de cette charge par un parcourt séquentiel de toutes les requêtes SQL afin d'en extraire tous les attributs susceptibles d'être des supports d'index. Pour cela on ne considère que les attributs présents dans les clauses **WHERE**, **GOUPED By** et **ORDER BY** des requêtes ; dans le but de choisir les meilleurs index pour chaque requête [8].

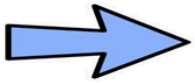
Construction du contexte d'extraction

Notre contexte d'extraction est représenté sous forme d'une matrice dont les lignes représentent les requêtes Q_i et les colonnes représentent tous les attributs indexables A_j par l'ensemble des requêtes. Pour construire cette matrice nous procédons selon l'algorithme suivant :

Algorithm 2: Algorithme de la construction du contexte d'extraction

ENTREE : Charge de requêtes ;Ensemble d'attributs;
SORTIE : Contexte d'extraction.;
pour *(Chaque ligne I)* **faire**
  pour *(Chaque colonne J)* **faire**
    si *l'attribut* A_J *est utilisé par la requête* Q_I **alors**
      Alors $C_{IJ} \longleftarrow 1$
    sinon
      Sinon $C_{IJ} \longleftarrow 0$

N° Requête	Attributs
1	{A, B, C, G}
2	{A, B}
3	{A, C, D}
4	{C, D, E, G}
5	{C, E, F}
6	{A, C, D, F}
7	{C, D, E}
8	{B, D}
9	{B, D, E}
10	{B, E}



N° Requête	A	B	C	D	E	F	G
1	1	1	1	0	0	0	1
2	1	1	0	0	0	0	0
3	1	0	1	1	0	0	0
4	0	0	1	1	1	0	1
5	0	0	1	0	1	1	0
6	1	0	1	1	0	1	0
7	0	0	1	1	1	0	0
8	0	1	0	1	0	0	0
9	0	1	0	1	1	0	0
10	0	1	0	0	1	0	0

FIGURE III.4 – Exemple de contexte d'extraction

III.3.2 Recherche des règles d'association

Cette étape représente le cœur de notre approche, l'objectif dans cette phase est la détection des liens entre attributs dans la base de transaction, ce problème nous ramène à l'extraction de toutes les règles de type **SI** *A* **ALORS** *B*. La *condition A* est un ensemble d'attributs et la *pré-*

diction B de la règle est également un ensemble d'attributs. Notre objectif étant d'extraire toutes les règles d'association, pour cela nous avons appliqué l'algorithme *Apriori* car ce dernier représente la base de tous les algorithmes de recherche de règles d'association ; dans un premier temps, la recherche de tous les motifs fréquents, et ensuite, d'en déduire, les règles d'association en se basant sur les deux valeurs qui sont le *support* qui indique le pourcentage des transactions (les requêtes dans notre cas) qui vérifient la règle et la *confiance* qui mesure la validité de la règle par le pourcentage de transactions qui vérifient la conclusion parmi ceux qui vérifient la prémisse. Ces deux paramètres mesurent la force d'une règle d'association et permettre d'éliminer la majorité des règles afin d'obtenir que les meilleur résultats.

Une fois toutes les règles d'association sont générées et afin de ne laisser que les règles intéressante un élagage est nécessaire pour réduire le nombre de règles par l'élimination des redondances.

Élagage

Un problème important concernant la recherche des règles d'association est la pertinence des informations extraites ; car les algorithmes appliqué produisent souvent de très nombreuses règles qui présentent des redondances syntaxiques ou sémantiques [25], en effet pour un ensemble de k attributs : Le nombre total des règles d'association :

$$\text{Nombre de règles} = 3^k - 2^{k-1} + 1$$

Étant donné le nombre élevé de règles d'association extraites à partir de l'ensemble de données en entrée, un élagage doit être effectué pour éliminer toutes les redondances exprimant des connaissances déjà connues, ce qui permet de réduire le nombre de règles d'association qu'aux règles intéressantes ; pour l'élagage appliquer afin écarter les règles d'associations redondantes nous procédons comme suit :

- Consiste à un parcours dans l'ensemble des règles résultantes pour la suppression des règles redondantes (c.à.d. génèrent le même index). Si dessous un exemple de deux règles redondantes :

  $A,B,C \rightarrow D$ ..... **Règle** R_1

  $A,B \rightarrow C,D$ ..... **Règle** R_2

  La règle R_1 nous génère la configuration A,B,C et la règle R_2, génère aussi A,B,C, Il s'agit

donc de deux règles différentes mais qui génèrent la même configuration. Pour cette phase d'élagage une seule règle sera retenue.

- Afin de réduire d'avantage le nombre de règles candidates et vu l'efficacité des motifs fréquents maximaux à réduire le nombre de motifs générés tout en préservons tous les attributs fréquents, nous avons opté pour cette stratégie pour faire l'élagage des règles générées. Pour cela, nous avons défini un élagage dit : par inclusion dans lequel on élimine toute règle construite sur un ensemble d'attributs constituant un sous-ensemble d'attributs d'une autre règle. Dans ce qui suit un exemple d'élagage :

  $A,B \rightarrow C,D,F$ ..... **Règle** R_1

  Cette règle génère le motif : A,B,C,D,F

  $A,B \rightarrow C$ ..... **Règle** R_2

  Cette règle génère le motif : A,B,C

  le motif A,B,C est inclut dans le motif A,B,C,D donc la règle R_2 est incluse dans la règle R_1 d'où la règle R_2 est éliminée. A cet effet nous avons développé un algorithme d'élagage comme suit :

Algorithm 3: Algorithme d'élagage

```
regles_resultat ⟵ {} ;
Fichier ⟵ ensemble_regles;
tant que (Fichier ≠ Fin_fichier) faire
    R ⟵ Lire_regle(Fichier);
    Supprimer_regle(Fichier,R);
    si (Inclu(regles_resultat,R) = Faux) alors
        Ajouter(regles_resultat,R)
```

III.3.3 Génération de la configuration finale des index

Après la détermination de l'ensemble des règles intéressante et avant de générer la configuration d'index finale, nous avons remarqué que étant donné qu'une règle d'association par sa définition génère un motif contenant au minimum deux attributs :

$A \rightarrow B$ cette règle génère le motif A,B

le motif A,B contient les deux attributs A et B.

Où les index mono attribut ne sont pas générés dans notre processus de recherche alors nous savions que ce type d'index permet une amélioration du coût d'exécution de ce fait pour améliorer notre solution, nous avons ajouté les attributs mono fréquents. Donc pour construire la configuration finale d'index pour chaque règle candidate :

$A,B \rightarrow C$ l'index généré par cette règle correspond à :A,B,C

Notre configuration d'index finale contient tous les index transformés à partir de toutes les règles trouvées avec tous les attributs mono fréquents, pour mesurer l'intérêt de notre solution nous avons implémenté un modèle de coût qui sera présenté dans la section suivante.

III.4 Modèle de coût

Pour évaluer notre solution, nous avons utilisé un modèle de coût pour la comparaison des résultats obtenus avec une charge sans index et pour mesurer le gain en nombre d'entrée sortie, pour cela nous nous sommes basé sur le modèle de coût dans les travaux de (Aouiche et al.)[8] où deux variantes du modèle sont présentées par les auteurs : un modèle de coût qui ne prend pas en considération l'implémentation physique des index et un modèle qui considère des IJB accédés par B-Arbre. Dans notre cas, nous allons considérer le premier modèle.

Le Tableau III.6 résume les notations adoptées dans le modèle que nous avons choisit :

Notations	Description
F	Table des faits
$\|\| F \|\|$	Nombre de pages nécessaires pour stocker F
$\| RowId \|$	Taille de l'identificateur de ligne en bits
$\| A_j \|$	Cardinalité de l'attribut A_j.
$\| X \|$	Nombre de pages nécessaires pour stocker la table X ou cardinalité de l'attribut X
S_p	Taille en octets d'une page système
d	Nombre de bitmaps utilisés pour évaluer une requête donnée

TABLE III.6 – Paramètres du modèle de coût

Taille de l'espace de stockage d'un index de jointure binaire

L'espace nécessaire pour stocker un index *bitmap* simple dépend linéairement de la cardinalité de l'attribut indexé et du nombre de lignes de la table concernée. la taille d'un index de jointure binaire est donnée par la formule suivante :

$$Taille(I) = \frac{\left(|RowId| + \sum_{j=1}^{n} |A_j|\right) \times |F|}{8}$$

où F représente la table de fait et A l'attribut de dimension.

Coût d'accès aux données par IJB

L'accès aux données dans un index de jointure binaire se fait en deux phases :

- La phase de parcours des *bitmaps* de l'index : Dans le pire des cas ; tous les *bitmaps* nécessaires doivent être parcourus pour rechercher le *bitmap* correspondant à une valeur de l'attribut indexé. Si S_p est la taille d'une page disque ; alors le nombre des pages parcourues pour lire un seul *bitmap* est $\frac{|F|}{8S_p}$. Le coût en terme d'entrées/sorties nécessaires pour rechercher un seul bitmap est $\frac{|A||F|}{8S_p}$. Lorsque la lecture de d *bitmaps* est nécessaire, donc le coût de la phase de parcours des *bitmaps* de l'index est :

  $$C_{parcours} = d\frac{|A|(|F| + |RowId|)}{8S_p}.$$

  La valeur de d est égale au nombre de prédicats appliqués sur l'attribut indexé et liés par l'opérateur **or** ou la cardinalité de la liste d'une clause *in*.
- La phase de lecture des n-uplets de la table indexé :

  le nombre total de n-uplets lus pour une requête utilisant d ; est peut être donné par :

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

  Étant donné le nombre de n-uplets lus, défini par la formule précédente, le nombre d'entrées/sorties de la phase de lecture est :

  $$C_{lecteur} = || F || \left(1 - e^{\frac{N_r}{||F||}}\right)$$

Donc Le coût d'accès aux données à travers un IJB est égal à :

$$C_{Acces} = d\frac{|A|(|F| + |RowId|)}{8S_p} + || F || \left(1 - e^{\frac{N_r}{||F||}}\right)$$

Chapitre IV

Étude expérimentale

IV.1 Introduction

Devant la complexité de la tâche de l'administration des entrepôts de données et la nécessité de l'application des méthode d'optimisation afin d'assister l'administrateur pour améliorer la qualité des réponse aux requêtes décisionnelles.

Beaucoup de travaux de recherches qui concernent le problème de sélection d'une configuration d'index dans les entrepôts des données relationnels. Le bût est d'améliorer le coût en temps de réponse et d'avoir une meilleure taille d'index qui respecte la contrainte de l'espace de stockage. Dans ce contexte de travail, nous nous sommes intéressés par l'utilisation des règles d'association qui l'une technique de fouille de données afin de bénéficie de la force des résultats obtenu par les règles d'association pour avoir une meilleure configuration d'index ou chaque index regroupe les attributs les plus corréler entre eux dans une charge de requête.

Dans la première partie de ce chapitre nous allons présenté brièvement l'outil *ISARules* (*Index Selection by Associate Rules*) que nous avons élaboré pour valider notre approche ; la deuxième partie est consacré à la présentation de l'environnement expérimentale sous lequel nous avons réaliser les différents tests. Dans la dernière partie nous présentons les différentes expériences menées afin de montrer l'intérêt de notre approche.

IV.2 Outil de sélection d'index *ISARules*

ISARules est notre outil de recherche des règles pour la sélection des index multi-attribut dans le cadre de validation de notre approche présenté dans le chapitre précédent, notre outil graphique ISARules réalisé en java (NetBeans IDE 7.3 Beta 2) dont l'architecture et l'interface principale sont présentées respectivement dans les figures IV.1 et IV.2.

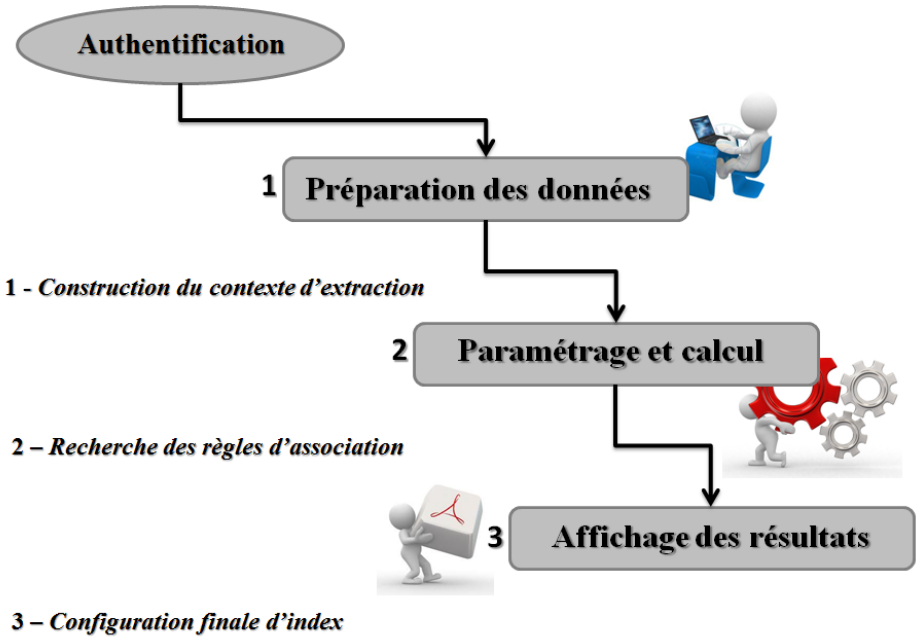


FIGURE IV.1 – Architecture de l'outil ISARules

ISARule se compose de trois modules :

- Préparation des données.
- Recherche des règles d'association.
- Affichage des résultats.

FIGURE IV.2 – Interface principale de l'outil ISARules

IV.2.1 Préparation des données

On prépare les données avant le lancement de la recherche des règles d'association, pour ce faire on distingue trois modules : Visualisation de l'entrepôt de données, Chargement du contexte d'extraction, et enfin Introduction des paramètres.

- **Visualisation de l'entrepôt de données**

  Ce module donne les détails caractérisant l'entrepôt de données, notamment les attributs des différentes tables et leurs cardinalités, avec la possibilité de visualisé la charge des requêtes.

- **Chargement du contexte d'extraction**

  Ce module est consacré à la construction du contexte d'extraction par un parcourt séquentiel de toutes les requêtes SQL notamment dans les clauses **WHERE**, **GOUPED By** et **ORDER BY** des requêtes dans la charge des requêtes pour extraire tous les attributs utiles qui peuvent être des support d'index dans la configuration d'index. après détermination des attributs indexables, le contexte d'extraction est crée sous forme d'une matrice dont les lignes représentent les requêtes Q_i et les colonnes représentent tous les attributs indexables A_j trouvés.

- **Introduction des paramètres**

  Dans ce module l'utilisateur donne les valeurs des paramètres essentiels : le support minimal ; la confiance minimale et le nombre maximal des règles d'association.

IV.2.2 Recherche des règles d'association

L'objectif de module calcul étant d'extraire les règles d'association, pour cela nous avons opté pour l'algorithme *Apriori* à travers l'appel du l'outil weka (voir annexe) ; pour lancer la recherche des règles des règles d'association l'outil à besoin d'un fichier spécifique en entré qui crée par notre module de calcul, un fichier résultat est généré après le lancement de la recherche ; ce fichier sera analysé par notre module pour ressortir les règles candidates.
Après détermination de toutes les règles candidate ; le module calcul procède à l'application de l'élagage vu dans le chapitre précédent pour éliminer toute les règles redondantes ; la dernière étape de ce module est la construction de la configuration d'index finale après l'ajout de tout les attributs fréquent comme des mono index.

IV.2.3 Affichage des résultats

Le module affichage des résultats montre la configuration d'index finale avec le coût d'entrée sortie et le coût de stockage.

IV.3 Expérimentation

IV.3.1 Environnement expérimental

Entrepôt de données

Notre étude utilise un entrepôt de données réel issu du benchmark APB1 (Council, 1998). Sur cet entrepôt de données. Le schéma en étoile est donné par la figure IV.3. Il est constitué d'une table de faits Actvars (24 786 000 tuples) et de quatre dimensions, Prodlevel (9 000 tuples), Custlevel (900 tuples), Timelevel (24 tuples) et Chanlevel (9 tuples).

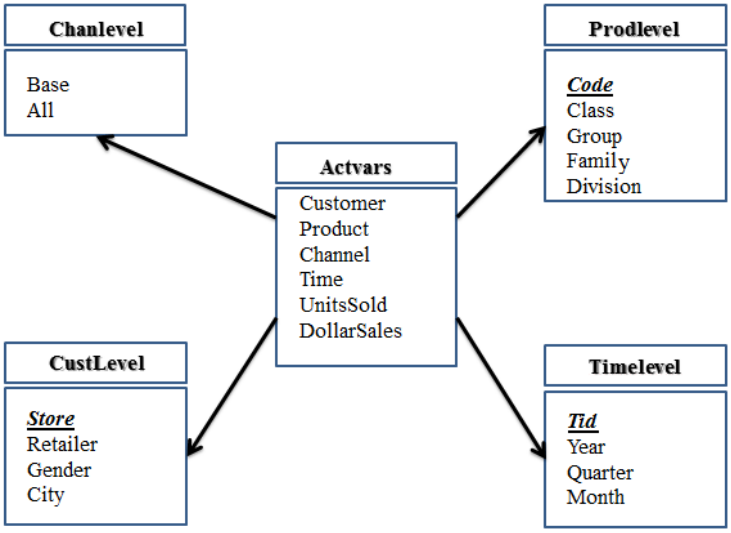


FIGURE IV.3 – Schéma de l'entrepôt expérimental

Table	Nbre. d'enregistrements	Taille de l'enregistrement
Actvars	24 786 000	74
ProdLevel	9	24
TimeLevel	900	24
CustLevel	9000	72
ChanLevel	24	36

TABLE IV.1 – Caractéristiques des tables de l'entrepôt expérimental

Charge de requêtes

La Charge de requêtes que nous avons considéré contient 60 requêtes indexables sur 12 attributs présentés dans le tableau suivant IV.2 :.

IV.3.2 Description de l'expérimentation

Pour évaluer notre approche nous avons procédé à quatre expériences qui nous permettent de mesurer l'intérêt de notre solution qui sont :

1. Effet de la technique d'élagage proposée.
2. Effet de la variation de la confiance minimale

Attribut	Code	Type	Taille	Cardinalité
CLASS_LEVEL	1	Char	12	605
QUARTER_LEVEL	2	Char	6	4
GROUP_LEVEL	3	Char	12	300
FAMILY_LEVEL	4	Char	12	75
LINE_LEVEL	5	Char	12	15
DIVISION_LEVEL	6	Char	12	4
YEAR_LEVEL	7	Char	4	2
MONTH_LEVEL	8	Char	6	12
RETAILER_LEVEL	9	Char	12	99
GENDER_LEVEL	10	char	12	2
ALL_LEVEL	11	char	12	5
CITY_LEVEL	12	char	12	4

TABLE IV.2 – Table des attributs

3. Effet du rajout des index mono attributs fréquents
4. Comparaison avec les motifs maximaux

Tout au long de ces expériences nous avons fixé la valeur du support minimal à 5%, la valeur retenu par les approches antérieures.

Résultat avec élagage et sans élagage

Afin de montrer la nécessité d'un élagage pour réduire la taille de la configuration d'index générée et de mesurer l'efficacité d'élagage, nous avons fixé la valeur de la confiance minimal à 50% et nous avons calculé le coût d'exécution et le coût de stockage dans les cas suivant :

1. Sans index.
2. Avec la totalité des index générés.
3. Avec index mais après l'élagage des index par notre approche.

Les résultats ont montré qu'avec les index générés nous pouvons avoir une réduction de 40% par rapport au coût d'exécution sans index, cela est dû aux index créés qui permettent d'éviter les jointures coûteuses entre la table de faits et les tables de dimensions figure IV.4.

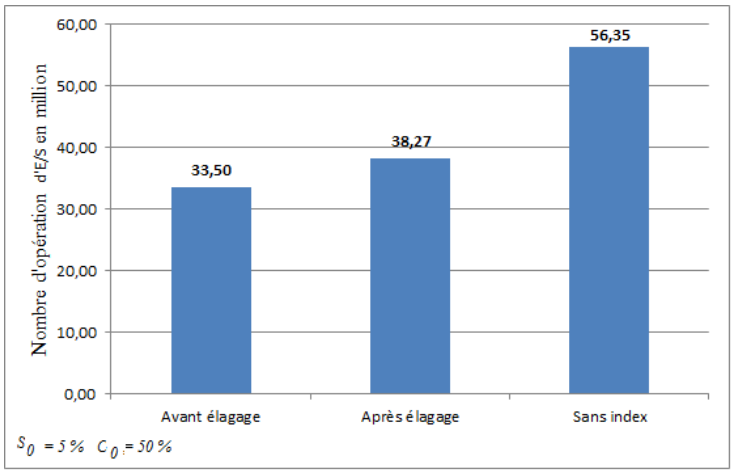


FIGURE IV.4 – Effet de l'élagage sur le nombre d'opérations d'E/S

Par contre la taille de la configuration des index est de 108,95 GO ce qui est énorme. Cela s'explique par le nombre d'index généré qui est de 74 index (Figure IV.5). Pour réduire ce nombre nous avons appliqué notre approche d'élagage qui a permis de réduire le nombre d'index de 74 index à 5 index et du fait la taille de la configuration d'index a diminué jusqu'à 8,32 GO soit une réduction de 92,35%. Par contre le coût d'exécution n'a diminué que de 32% par rapport à la configuration sans index (Figure IV.4). Cela s'explique par la diminution du nombre d'index qui ne permet pas pour certaines requêtes à être optimisées. Cette diminution sera acceptable en tenant compte du gain de 92,35% dans la taille de l'espace de stockage (Figure IV.4).

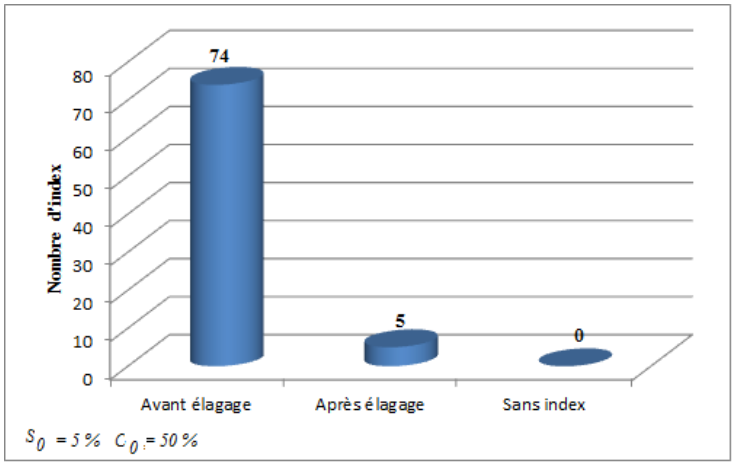


FIGURE IV.5 – Effet de l'élagage sur le nombre d'index

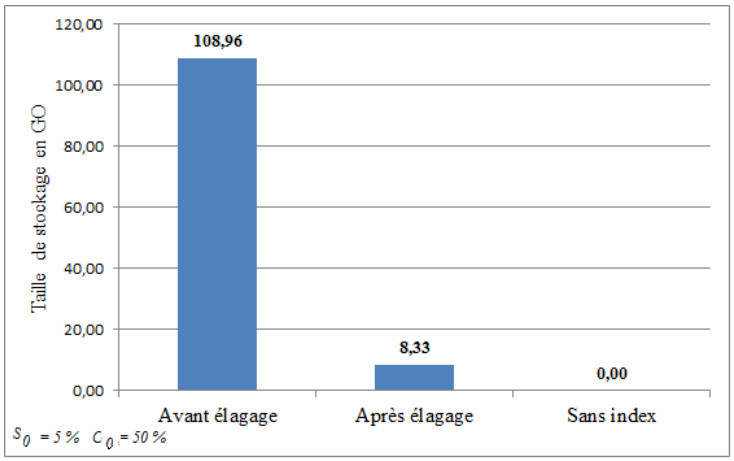


FIGURE IV.6 – Effet de l'élagage sur l'espace de stockage

Effet de la variation de la confiance minimale

Afin de connaitre l'effet de la variation de la confiance minimale sur le coût d'entrée sortie et le coût de stockage, nous avons mené une expérience dans laquelle nous calculons le coût de stockage pour les valeurs de la confiance minimale de $0,10$ à $0,90$ avec un pas de $0,10$. La figure IV.7 montre que l'augmentation de la valeur de la confiance minimale apporte une diminution de la taille de stockage, cela est dû à la réduction du nombre des index générées ce qui est prévisible grâce à la propriété de la confiance minimale qui exige plus de corrélation entre les attributs constituants l'index. Tandis que le coût d'exécution a augmenté à cause de la diminution du nombre d'index figure IV.8.

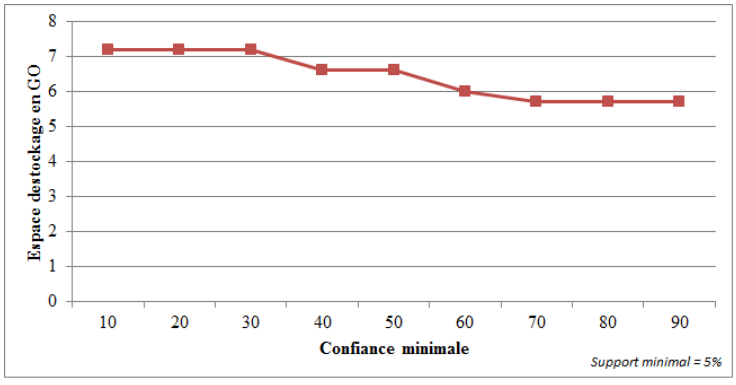


FIGURE IV.7 – Effet de la variation de la confiance minimale sur le coût de stockage

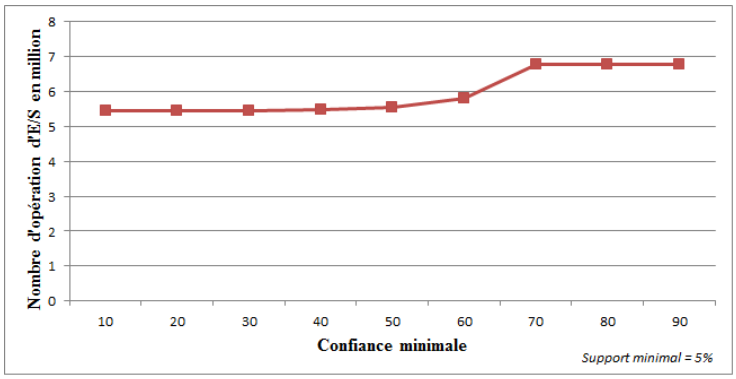


FIGURE IV.8 – Effet de la variation de la confiance minimale sur le coût d'exécution

Effet du rajout des index mono attributs fréquents

Vu les propriétés des règles d'association telle que chaque règles génère au minimum un index à deux attributs ce qui pénalise les index mono attributs, par conséquent les requêtes utilisant un seul attribut ne serons pas optimisées ; pour cela nous avons rajouté à notre configuration les index mono attribut (uniquement les attributs fréquents), la figure IV.9 montre la configuration résultante qui donne un gain de 6,95% en temps d'exécution par rapport à la configuration ne contenant pas des index mono attribut, avec une légère augmentation du coût de stockage qui est dû à l'augmentation du nombre d'index IV.10.

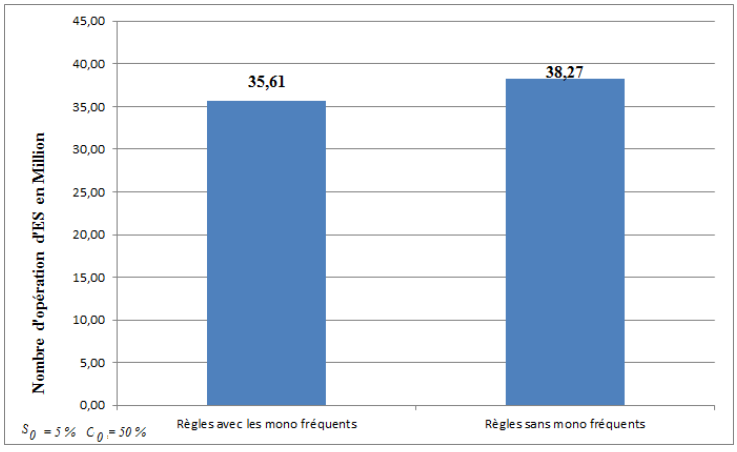


FIGURE IV.9 – Effet du rajout des index mono attributs fréquents sur le coût d'E/S

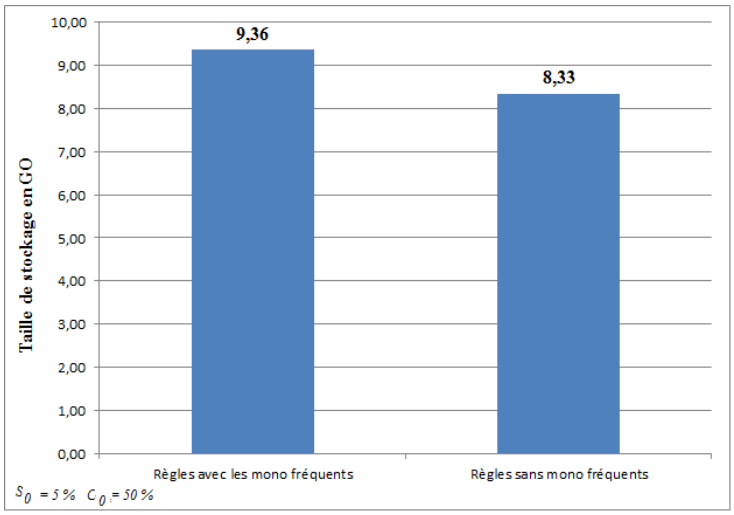


FIGURE IV.10 – Effet du rajout des index mono attributs fréquents sur le coût stockage

Etude comparative

Pour situer notre approche par rapport aux autres approches de sélection d'index par des techniques de datamining, nous avons effectué une expérimentation dans laquelle nous avons comparé nos résultats avec ceux de l'approche des motifs maximaux. Nous avons calculé le coût d'exécution et l'espace de stockage. La figure IV.11 montre que le coût de stockage de notre approche à diminuer de 2.84% par rapport à l'approche des motifs maximaux qui représente un gain en espace de stockage, par contre le coût d'exécution de notre approche est légèrement supérieur ; cela est dû à la diminution du nombre d'index ce qui pénalise quelques requêtes d'être optimisées figure IV.12.

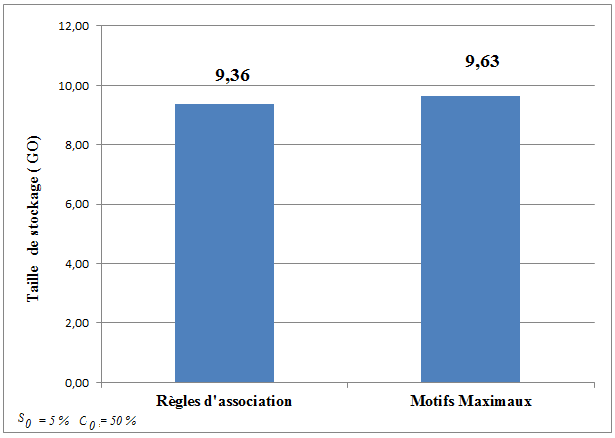


FIGURE IV.11 – Maximaux Vs Règles d'association(coût de stockage)

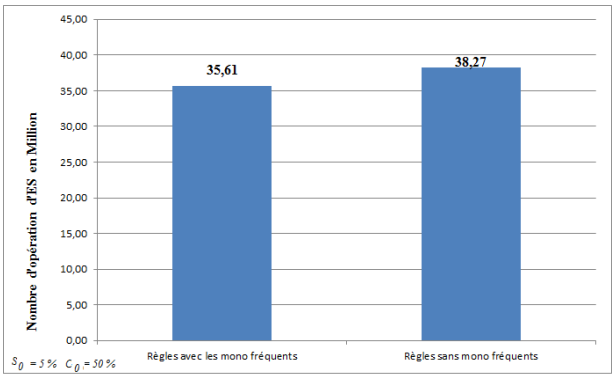


FIGURE IV.12 – Maximaux Vs Règles d'association(coût d'exécution)

Conclusion générale

Les entrepôts de données étendent les concepts et la technologie traditionnelle des systèmes de gestion des bases de données pour la création des systèmes d'aide à la décision où un nombre important de requêtes à exécuter et qui peuvent également prendre un temps d'exécution important. Il est alors crucial pour l'administrateur de pouvoir gérer la performance de l'entrepôt par l'optimisation des requêtes décisionnelles exécutées sur entrepôt de données ; Nous avons présenté dans ce mémoire notre approche basée sur la recherche des règles d'associations pour la sélection d'index multi-attributs. Pour montrer l'intérêt de notre recours aux règles d'association pour une meilleur configuration d'index, nous avons élaboré un outil logiciel *ISARules* qui prend en entrée le contexte d'extraction et génère la configuration d'index en sortie ; Pour montrer l'intérêt de notre approche, nous avons effectué une série de test ainsi qu'une comparaison avec l'approche des motifs maximaux ; Les résultats nous ont permis de constater :

- Un gain de 32% en temps d'exécution entre la configuration sans index et la configuration d'index de notre approche.
- Un coût d'exécution proche de celui de l'approche des motifs maximaux mais cette dernière à un coût de stockage supérieur.
- Vu que les règles d'association ne permettent pas de générer des index mono-attributs, nous avons intégré des index mono-attributs à la configuration d'index. Ceci a permis un gain de 37% par rapport à la configuration sans les mono-index.

Différentes perspectives sont envisagées. Dans un premier temps il est intéressant d'améliorer notre outil logiciel par l'ajout d'un module très utile pour analyser le journal de transaction et d'extraire le contexte d'extraction automatiquement. Également pour la suite de nos travaux nous comptons inclure d'autres paramètres des règles d'association comme le lift et finalement combiner notre approche avec d'autres techniques d'optimisation tel que les vues matérialisées et la fragmentation.

Références et bibliographie

[1] M. ROSS R. KIMBALL. The Data Warehouse Toolkit : The complete guide to dimentional modeling. In *John Wiley Sons, 2002.*, 2002.

[2] Olivier T. Modélisation et manipulation d'entrepôts de données complexes et historisées. In *Thèse de doctorat,Université PAUL SABATIER.*, 2000.

[3] Bellatreche L. Utilisation des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données . In *Thèse de Doctorat en Informatique, Université de Clermond Ferrand.*, 2000.

[4] Ziani B. Bellatreche L., Boukhalfa K. De la conception physique aux outils d'administration et de tuning des entrepôts de donnés. In *Thèse Doctorat*. Ecole nationale supérieure de mécanique et d'aérotechnique Poitier, Juillet 2009.

[5] Bellatreche L. Techniques d'optimisation des requêtes dans les data warehouses. In *Sixth International Symposium on Programming and Systems* , 2003.

[6] Encinas M. Entrepôts de données pour l'aide à la décision médicale : conception et expérimentation. In *Thèse de doctorat, Université Joseph Fourier.*, June 1978.

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

[8] Aouiche K. Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données. In *Thèse Doctorat*, Dćembre 2005.

[9] S. Rizzi. M. Golfarelli, D. Maio. Conceptual design of data warehouses from e/r schemes. In *the 31th Hawaii Conference on System Sciences.*, January 1998.

[10] B. Ziani K. Boukhalfa, L. Bellatreche. Index de Jointure Binaires : Stratégies de Sélection et Etude de Performances. In *revue des Nouvelles Technologies De L'information RNTI (EDA'2010), éditions cepadues*, Juin 2010.

[11] Darmont J. Optimisation et évaluation de performance pour l'aide à la conception et à l'administration des entrepôts de données complexes. In *Thèse de doctorat, Université Lumière - Lyon II , Laboratoire ERIC.*, November 2011.

[12] H. Mahboubi. Optimisation de la performance des entrepôts de données XML par fragmentation et répartition. In *Thèse de Doctorat en Informatique, Université de Université Lumière Lyon 2*, December 2008.

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

[14] D. Comer. The dificulty of optimum index selection. In *ACM Transactions on Database Systems (TODS).*, November 1978.

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

[16] Chaudhuri S. et Narasayya V. Index Selection For Databases : A Hardness Study and a Principaled Heuristic Solution. In *IEEE Transactions on Knowledge and Data Engineering*, 2008.

[17] Zilio G. Valentin G., Zuliani M. Db2 advisor : An optimizer smart enough to recommend its own indexes. In *16th International Conference on Data Engineering (ICDE 00),San Diego,USA*, page 273–284, 2000.

[18] Y. Feldman and J. Reouven. A knowledge based approach for index selection in relational databases. In *In Expert System with Application*, page 15–37, 2003.

[19] Tosic D. Kratica J., Ljubic I. A Genetic Algorithm for the Index Selection Problem . In *in Applications of Evolutionary Computing, EvoWorkshops 2003 : EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM, volume 2611 de LNCS*, pages 281–291, 2003.

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

[21] Missaoui R. et Necir H. Bellatreche L. A Data Mining Approach for Selecting Bitmap Join Indices. In *Journal of Computing Science and Engineering* ., pages 177–194, 2007.

[22] Swami A. Agrawal R., Imielinski T. association rules between sets of items in large databases,. In *in P. Buneman & S. Jajodia, eds, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993.

[23] Martine CADOT. Extraire et valider les relations complexes en sciences humaines : statistiques, motifs et règles d'association. In *Thèse de Doctorat en Informatique, Université de Franche-Comté – Besançon.*, 2006.

[24] Srikant R. Agrawal R. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[25] Jamil Stéphan. Extraction des règles de corrélation pour la décision dans une base de données relationnelle. 2006.

[26] Ouniten Y. Ziani B. Vers l'auto-sélection des index dans les entrepôts de données : une approche basé sur la recherche des motifs fréquents maximaux. In *10e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées CARI'2010*, 2010.

[27] Benameur Ziani and Youcef Ouinten. Combining data mining technique and query frequencies for automatic selection of indexes in data warehouses. In *DB&IS*, pages 71–83, 2012.

[28] Benameur Ziani, François Rioult, and Youcef Ouinten. A constraint-based mining approach for multi-attribute index selection. In *ICEIS (1)*, pages 93–98, 2012.

[29] R. Bouchakri. Une approche dirigée par la classification des attributs pour fragmenter et indexer les entrepôts de données. In *Thèse de Magister*, 2009.

[30] Darmont J. Performance des entrepôts de données. In *Rapport de cours université de Lyon 2.*, November 2009.

[31] Barr M. Approche dirigée par les fourmis pour la fragmentation horizontale des entrepôts de données relationnels. In *Thèse de Magister , E.S.I (Ecole nationale Supérieure d'Informatique) (ex. INI)*, 2009.

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

[33] Edward R. Omiecinski Martin R. Frank and Shamkant B. Navathe. Adaptive and Automated Index Selection in RDBMS. In *in 3rd International Conference on Extending Database Technology, EDBT 1992*, Vienna, Austria, volume 580 de Lecture Notes in Computer Science,, page 277 292, 1992.

[34] Gundem T.I. Near optimal multiple choice index selection for relational databases. In *Computers & Mathematics with Applications,*, pages 37(2) :111–120, 1999.

[35] Benoît Vaillant. Mesurer la qualité des règles d'association : études formelles et expérimentales. In *Thèse de Doctorat de l'ENST Bretagne « Mathématiques et Informatique »*, 2006.

[36] Georges H. Introduction de la recherche de règles d'associations et de séquences fréquentes. In *Ecole Modulad "Bases de données et statistiques"*, Croisic, 1999.

[37] Inmon B. Building the data warehouse. In *1st Edition, Wiley and Sons.*, 1992.

[38] FAVRE C. Utilisation des index bitmap pour la fouille de données. In *Equipe de Recherche en Ingénierie des Connaissances, rapport de recherche, Laboratoire ERIC, Université Lyon 2.*, 2003.

[39] Saporta G. Plasse M., Niang N. and Leblond L. Une comparaison de certains indices de pertinence des règles d'association. In *In Extraction et Gestion des Connaissances*, page 561–568, 2006.

[40] Boussaid O. et Bentayeb F. Aouiche K. Automatic Selection of Bitmap Join Indexes in Data Warehouses. In *7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK05),*, pages 64–73, 2005.

[41] GASMI G., Ben Yahia S., Mephu Nguifo E., and Y. Slimani. Igb : une nouvelle base générique informative des règles d'association. pages 31–67, oct 2006.

[42] Nicolas Pasquier. *Data Mining : algorithmes d'extraction et de réduction des règles d'association dans les bases de données*. PhD thesis, Université Blaise Pascal - Clermont-Ferrand II, January 2000.

[43] M. Plasse and G. Saporta. Utilisation conjointe des règles d'association et de la classification de variables. In *Journées de Statistique, Pau*, January 2005.

[44] Marie Plasse, Ndeye Niang, Gilbert Saporta, and Laurent Leblond. Une comparaison de certains indices de pertinence des règles d association. In *Extraction et gestion des connaissances (EGC 2006), Actes des sixièmes journées Extraction et Gestion des Connaissances, Lille, France, 17-20 janvier 2006, 2 Volumes*, pages 561–568, 2006.

[45] Saïd Taktak and Jamel Feki. A maintenance approach of a bji index configuration. In *conference ICSEA 2011, The Sixth International Conference on Software Engineering Advances*, pages 221–226, 2011.

[46] Cecile Favre, Fadila Bentayeb, and Omar Boussaid. Modèle d'entrepôt de données à base de règles. In *3ième atelier Fouille de Données Complexes dans un processus d'extraction de connaissances, (FDC06) en conjonction avec EGC 2006, Lille*, pages 39–50, January 2006.

[47] Soumaya Ben Hassine-Guetari and Delphine Clément. Règles d'association pour la qualité des données. In *Actes du 5e Atelier Qualité des Données et des Connaissances*, pages 33–39. association Extraction et Gestion des Connaissance, January 2009.

Annexe A

Annexes

ANNEXES

Liste des requêtes

Req 1 SELECT Time_level,count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.Class_LEVEL='PO0HV1RICH5W' group by Time_level	**Req 2** SELECT line_level,sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.Class_LEVEL='CI493YZ9KZUJ' group by line_level
Req 3 SELECT count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.Class_LEVEL='FDXAQ1N5U026'	**Req 4** SELECT Time_level,Avg(UNITSSOLD) FROM ACTVARS A,Timelevel T WHERE A.TIME_LEVEL=T.TID AND T.Quarter_level='Q1' group by Time_level
Req 5 SELECT division_level,count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.group_LEVEL='E4NJTW0ZR9FN' group by division_level	**Req 6** SELECT Max(UNITSSOLD) FROM ACTVARS A,Timelevel T WHERE A.TIME_LEVEL=T.TID AND T.Quarter_level='Q2'
Req 7 SELECT Time_level,count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.family_LEVEL='BEMFVK0N8125' group by Time_level	**Req 8** SELECT year_level,sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P, Timelevel T WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.time_level=T.TID AND P.family_LEVEL='UJHZ4TZMJT6V' group by year_level
Req 9 SELECT month_level,count(*) FROM ACTVARS A,Timelevel T WHERE A.time_level=T.TID AND T.Quarter_level='Q3' group by month_level	**Req 10** SELECT Sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.LINE_LEVEL = 'MJ1F1U1EG009'
Req 11 SELECT Product_level,count(*) FROM ACTVARS A,Timelevel T WHERE A.TIME_LEVEL=T.TID AND T.Quarter_level='Q4' group by Product_level	**Req 12** SELECT retailer_level,Avg(unitssold) FROM ACTVARS A,PRODLEVEL P ,Custlevel C WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.customer_level=C.Store_level AND P.DIVISION_LEVEL = 'BCR2T4K2K9D3' group by retailer_level

Req 13 SELECT count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.DIVISION_LEVEL = 'XRLXY6H61SLC'	**Req 14** SELECT Customer_level,Sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.DIVISION_LEVEL = 'RC5406URP1IE' group by Customer_level
Req 15 SELECT all_level,count(*) FROM ACTVARS A,PRODLEVEL P ,Chanlevel H WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.channel_level=H.Base_level AND P.DIVISION_LEVEL = 'G4HA5YITG3H7' group by Channel_level	**Req 16** SELECT count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.year_LEVEL = '1995'
Req 17 SELECT Product_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.year_LEVEL = '1996' group by Product_level	**Req 18** SELECT division_level,Avg(Unitssold) FROM ACTVARS A,TIMELEVEL T ,Prodlevel P WHERE A.TIME_LEVEL=T.TID AND A.product_level=P.Code_level AND T.month_LEVEL = '1' group by division_level
Req 19 SELECT count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '2'	**Req 20** SELECT Product_level,count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '3' group by Product_level
Req 21 SELECT division_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T ,Prodlevel P WHERE A.TIME_LEVEL=T.TID AND A.product_level=P.Code_level AND T.month_LEVEL = '4' group by division_level	**Req 22** SELECT Avg(Unitssold) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '5'
Req 23 SELECT Product_level,count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '6' group by Product_level	**Req 24** SELECT division_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T ,Prodlevel P WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '7' AND A.product_level=P.Code_level group by division_level

Req 25 SELECT Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '8'	**Req 26** SELECT Customer_level,count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '9' group by Customer_level
Req 27 SELECT year_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '10' group by year_level	**Req 28** SELECT count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '11'
Req 29 SELECT Product_level,Time_level,Avg(unitssold) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '12' group by Product_level,Time_level	**Req 30** SELECT year_level,month_level, Max(unitssold) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'Z6OFS4YAAD4J' AND A.time_level=T.TID group by year_level,month_level
Req 31 SELECT Product_level,Time_level,Avg(unitssold) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '12' group by Product_level,Time_level	**Req 32** SELECT year_level,month_level, Max(unitssold) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'Z6OFS4YAAD4J' AND A.time_level=T.TID group by year_level,month_level
Req 33 SELECT count(*) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'RQJNEN0UPKMQ'	**Req 34** SELECT Product_level,Time_level, Min(unitssold) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'NXEYFSIQE3JM' group by Product_level,Time_level
Req 35 SELECT year_level,Sum(Dollarcost) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.time_level=T.TID AND C.GENDER_LEVEL='M' group by year_level	**Req 36** SELECT count(*) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='ABCDEFGHIJKL'

Req 37 SELECT Channel_level,Time_level, Sum(Dollarcost) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='BCDEFGHIJKLM' group by Channel_level, Time_level	**Req 38** SELECT class_level,year_level, Time_level, Avg(Unitssold) FROM ACTVARS A,CHANLEVEL CH ,Prodlevel P ,Timelevel T WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND A.product_level=P.Code_level AND A.time_level=T.TID AND CH.ALL_LEVEL ='CDEFGHIJKLMN' group by class_level,year_level, Time_level
Req 39 SELECT count(*) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='DEFGHIJKLMNO'	**Req 40** SELECT Time_level,count(*) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='EFGHIJKLMNOP' group by Time_level
Req 41 SELECT year_Level,month_level, sum(dollarcost) FROM ACTVARS A,CUSTLEVEL C,PRODLEVEL P ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.time_level=T.TID AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.CLASS_LEVEL='CI493YZ9KZUJ' AND C.RETAILER_LEVEL='RQJNEN0UPKMQ' AND C.CITY_LEVEL='Bordeaux' group by year_level,month_level	**Req 42** SELECT sum(dollarcost) FROM ACTVARS A, PRODLEVEL P,TIMELEVEL T WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.TIME_LEVEL=T.TID AND T.QUARTER_LEVL='Q2' AND T.YEAR_LEVEL='1996' AND P.CLASS_LEVEL='FDXAQ1N5U026'
Req 43 SELECT Customer_Level, Time_level, Avg(Unitssold) FROM ACTVARS A, CHANLEVEL H,PRODLEVEL P, TIMELEVEL T,CUSTLEVEL C WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.TIME_LEVEL=T.TID AND A.CUSTOMER_LEVEL=C.STORE_LEVEL A.CHANNEL_LEVEL=H.BASE_LEVEL and P.CLASS_LEVEL='FDXAQ1N5U026' AND H.ALL_LEVEL ='ABCDEFGHIJKL' AND T.QUERTER_LEVEL='Q1' AND C.GENDER_LEVEL='F' group by Customer_level, Time_level	**Req 44** SELECT year_Level, division_level, Max(Unitssold) FROM ACTVARS A, CUSTLEVEL C,TIMELEVEL T ,Prodlevel P WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.product_level=P.Code_level AND A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='1' AND C.RETAILER_LEVEL ='RQJNEN0UPKMQ' AND C.GENDER_LEVEL='F' AND C.CITY_LEVEL='Poitiers' group by year_level, division_level

Req 45 SELECT sum(dollarcost), Avg(Unitssold) FROM ACTVARS A, CUSTLEVEL C,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='12' AND C.RETAILER ='RQJNEN0UPKMQ' AND C.CITY_LEVEL='Dijon'	**Req 46** SELECT Customer_Level, Time_level,Min(unitssold) FROM ACTVARS A, CHANLEVEL H,TIMELEVEL T, CUSTLEVEL C WHERE A.CHANNEL_LEVEL=H.BASE_LEVEL AND A.CUTOMER_LEVEL=C.STORE_LEVEL AND A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1996' AND T.QUARTER_LEVEL='Q3' AND H.ALL_LEVEL='DEFGHIJKLMNO' AND C.GENDER_LEVEL='M' group by Customer_level, Time_level
Req 47 SELECT month_Level, all_level, Time_level, Sum(Dollarcost) FROM ACTVARS A, CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T ,Chanlevel H WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.channel_level=H.Base_level AND A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1996' AND C.RETAILER_LEVEL ='NXEYFSIQE3JM' AND C.CITY_LEVEL='Paris' AND P.LINE_LEVEL='MJ1F1U1EG009' group by month_level,all_level, Time_level	**Req 48** SELECT Avg(unitssold) FROM ACTVARS A, CHANLEVEL H,PRODLEVEL P,TIMELEVEL T WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='1' AND P.DIVISION_LEVEL='XRLXY6H61SLC' AND H.ALL_LEVEL='BCDEFGHIJKLM'
Req 49 SELECT Customer_Level, Product_level, Channel_level, sum(dollarcost) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and P.FAMILY_LEVEl='AGG214DG271Q' AND C.RETAILER_LEVEL='NXEYFSIQE3JM' AND C.GENDER_LEVEL='M' AND H.ALL_LEVEL='DEFGHIJKLMNO' group by Customer_level, Product_level, Channel_level	**Req 50** SELECT division_Level, year_level, Max(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,TIMELEVEL T ,Prodlevel P WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.product_level=P.Code_level AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='4' AND C.RETAILER_LEVEL='NXEYFSIQE3JM' AND C.CITY_LEVEL='Poitiers' AND C.GENDER8LEVEL='F' AND H.ALL_LEVEL='BCDEFGHIJKLM' group by division_level, year_level

Req 51	**Req 52**
SELECT Min(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='7' AND P.GROUP_LEVEL='E4NJTW0ZR9FN' AND C.RETAILER_LEVEL='Z6OFS4YAAD4J' AND H.ALL_LEVEL='EFGHIJKLMNOP'	SELECT Customer_Level, Channel_level, sum(dollarcost) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='11' AND P.CLASS_LEVEL ='FDXAQ1N5U026' AND C.RETAILER_LEVEL ='RQJNEN0UPKMQ' AND C.CITY_LEVEL='Poitiers' AND H.ALL_LEVEL='DEFGHIJKLMNO' group by Customer_level, Channel_level
Req 53	**Req 54**
SELECT line_Level, month_level, Avg(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL AND A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1995' AND T.QUARTER_LEVEL='Q1' AND P.GROUP_LEVEL='E4NJTW0ZR9FN' AND C.RETAILER_LEVEL='RQJNEN0UPKMQ' AND H.ALL_LEVEL='BCDEFGHIJKLM' group by line_level, month_level	SELECT Min(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1995' AND T.QUARTER_LEVEL='Q1' AND T.MONTH_LEVEL in('2','3','4') AND P.CLASS_LEVEL='CI493YZ9KZUJ' AND C.RETAILER_LEVEL in ('Z6OFS4YAAD4J','RQJNEN0UPKMQ') AND C.GENDER_LEVEL='F' AND C.CITY_LEVEL='Poitiers' AND H.ALL_LEVEL ='ABCDEFGHIJKL'

Req 55 SELECT Customer_Level, Product_level, Time_level, Channel_level, Max(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1996' AND T.MONTH_LEVEL in('5','6','7') AND P.CLASS_LEVEL='FDXAQ1N5U026' AND C.GENDER_LEVEL='M' AND C.RETAILER_LEVEL='RQJNEN0UPKMQ' AND H.ALL_LEVEL ='DEFGHIJKLMNO' group by Customer_level, Product_level, Time_level, Channel_level	**Req 56** SELECT Sum(Dollarcost) FROM ACTVARS A,Timelevel T WHERE A.time_level=T.TID AND C.GENDER_LEVEL='F'
Req 57 SELECT month_level,Sum(Dollarcost) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.time_level=T.TID AND C.City_LEVEL='Poitiers' group by month_level	**Req 58** SELECT time_level,avg(Dollarcost) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.City_LEVEL='Bordeaux' group by time_level
Req 59 SELECT avg(Dollarcost) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.City_LEVEL='Paris'	**Req 60** SELECT year_level,max(Dollarcost) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.City_LEVEL='Dijon' group by year_level

Les fréquences des requêtes de la charge

Requête	Fréquence
Q 1	**3**
Q 2	**7**
Q 3	**10**
Q 4	**10**
Q 5	**12**
Q 6	**3**
Q 7	**3**
Q 8	**16**
Q 9	**4**
Q 10	**12**
Q 11	**4**
Q 12	**21**
Q 13	**21**
Q 14	**21**
Q 15	**3**
Q 16	**7**
Q 17	**10**
Q 18	**10**
Q 19	**20**
Q 20	**4**
Q 21	**12**
Q 22	**4**
Q 23	**21**
Q 24	**21**
Q 25	**21**
Q 26	**3**
Q 27	**7**
Q 28	**10**
Q 29	**10**
Q 30	**20**

Requête	Fréquence
Q 31	**12**
Q 32	**4**
Q 33	**12**
Q 34	**4**
Q 35	**21**
Q 36	**3**
Q 37	**15**
Q 38	**10**
Q 39	**3**
Q 40	**12**
Q 41	**12**
Q 42	**10**
Q 43	**18**
Q 44	**11**
Q 45	**12**
Q 46	**5**
Q 47	**21**
Q 48	**12**
Q 49	**12**
Q 50	**10**
Q 51	**15**
Q 52	**30**
Q 53	**5**
Q 54	**20**
Q 55	**18**
Q 56	**3**
Q 57	**4**
Q 58	**10**
Q 59	**2**
Q 60	**5**

Présentation de l'outil *weka*

Weka (Waikato Environment for Knowledge Analysis) a été développé à l'Université deWaikato en Nouvelle-Zélande :Waikato Environment for Knowledge Analysis , Le système est écrit en Java et distribué sous les termes de la licence GNU General Public. Weka est un ensemble d'outils permettant de manipuler et d'analyser des fichiers de données, implémentant la plupart des algorithmes d'intelligence artificielle, entre autres, les arbres de décision ; les réseaux de neurones et les règles d'association. Il fonctionne sur presque n'importe quelle plate-forme : Linux,Windows et les systèmes d'exploitation Macintosh. Il fournit une interface uniforme aux nombreux algorithmes d'apprentissage, ainsi que des méthodes de pré-et post-traitement et pour évaluer les résultats de l'apprentissage sur tout ensemble de données. Il se compose principalement :

- De classes Java permettant de charger et de manipuler les données.
- De classes pour les principaux algorithmes de classification supervisée ou non supervisée.
- D'outils de sélection d'attributs, de statistiques sur ces attributs.
- De classes permettant de visualiser les résultats.

On peut l'utiliser à trois niveaux :

- Via l'interface graphique, pour charger un fichier de données, lui appliquer un algorithme, vérifier son efficacité.
- Invoquer un algorithme sur la ligne de commande.
- Utiliser les classes définies dans ses propres programmes pour créer d'autres méthodes, implémenter d'autres algorithmes, comparer ou combiner plusieurs méthodes.

WEKA peut importer différents formats de fichier. Mais il propose surtout un format propriétaire (*.**ARFF**) qui est un format texte. Le format **ARFF** est composé de 3 parties. La première partie , initiée par le mot clé « @relation », correspond au dictionnaire des variables ou le nom de la base de données ;la deuxième partie commence par le mot clé « @attribute » liste les variables ou les attributs dans notre cas et Enfin, la troisième partie du fichier, commence par le mot clé « @data » correspond à la description du contexte d'extraction ou chaque ligne correspond à une requête ; la valeur 1 indique l'existence de l'attribut dans la requête et l'absence indique le contraire la disposition des attribut est selon la deuxième partie.

Code source du « Modèle de coût »

```
package firstclass;
import java.io.*;
import java.util.*;
public class Firstclass {
/**
* @param args
* @return
*/
//Fonction qui permet de Faire le mapping String /Integer en vue d'utiliqer les attributs comme indices
public static Map<String, Integer> mapStringInt(String[] table) {
Map<String, Integer> myMap = new HashMap<String, Integer>();
for(int i = 0; i < table.length; i++){
myMap.put(table[i], new Integer(i));
}
return myMap;
}
//Fin mapping
//Recherche indice element Max
public static int max(int[] x, float[] tailleIndex)
{
if (x.length < 1)
{
throw new IllegalArgumentException("Empty input array, need at least one element.");
}
int highest = x[0];
int maxIndex = 0;
for (int i = 1; i < x.length; i++)
{
if (x[i] > highest)
{
highest = x[i];
maxIndex = i;
}
if (x[i] == highest && tailleIndex[maxIndex] > tailleIndex[i] )
{
maxIndex = i;
}
}
return maxIndex;
}
//Fin mapping
public static float strToFloat(String s)
{
float f = Float.valueOf(s.trim()).floatValue();
return f;
}
public static int tamponDispo(float tailleResultat, float tailleBuffer)
{
if (tailleResultat > tailleBuffer )
{
return 1;
}else
{
return 0;
}
}
public static void main(String[] args) {
//Déclaration des differents variables
```

```
float pagesTFait=0;
float pageSysteme =0;
float bufferSizePS =0;
float coutChargeRequete=0;
float coutChargeRequeteSansIndex=0;
double calcul=0;
int indice =0;
int nbrmotifs=0;
int nombreRequete =0;
int nUpletsTFait=0;
int nbrReqNonOptimise=0;
int nbrReqCouvertes=0;
int nbrReqPartielOptimise=0;
Map<String, Integer> mapStrToInt;
Map<String, Integer> mapSelectivity;
String[] titreVal={"titre","valeur"};
String[] tAttributs=null;
String[] tCard=null;
String[] tTailleAttribut=null;
String[] tTailleTableDim=null;
String[] tTableOfAttribut=null;
String[] tDim =null;
String[] tSelectivityF=null;
String[] tTailleTupleDim=null;
String[] tMotifs=null;
String[] tContexte=null;
float[]coutRequete;
float[]coutRequeteSansIndex;
/*************************************************************************************************
**********************
* TRAITEMENT DU FICHIER DES VALEURS
*************************************************************************************************
**********************/
//Chargement du fichier contenant les valeurs: cardinalités, tailles des tables, appartenance des attributs
String valFichier = "tabVal.txt" ;
try{
FileReader valFile = new FileReader ( valFichier ) ;
BufferedReader br = new BufferedReader (valFile ) ;
//Lire la 1ere ligne contenant nUplets Table De Fait
String ligne = br.readLine() ;
titreVal = ligne.split(" ");
nUpletsTFait= new Integer (titreVal[1]);
//Lire la 2eme ligne contenant taille Page Systeme:
ligne = br.readLine( ) ;
titreVal = ligne.split(" ");
pageSysteme = new Integer (titreVal[1]);
//Lire la 3eme ligne contenant taille TFait en octet
ligne = br.readLine( ) ;
titreVal = ligne.split(" ");
// calcul du nombre de pages systeme nécéssaires pour stocker la table des faits
pagesTFait = new Float (titreVal[1])/pageSysteme;
System.out.println(" pagesTFait= " + pagesTFait ) ;
//Lire la 4eme ligne contenant la taille du buffer exprimé en nombre de pages systeme
ligne = br.readLine( ) ;
titreVal = ligne.split(" ");
bufferSizePS = new Float (titreVal[1]);
//Lire la 5eme ligne contenant les attributs
ligne = br.readLine( ) ;
tAttributs = ligne.split(" ");
//Lire la 6eme ligne contenant les cardinalités des attributs
```

```
ligne = br.readLine( ) ;
tCard = ligne.split(" ");
//Lire la 7eme ligne contenant les tailles des attributs
ligne = br.readLine( ) ;
tTailleAttribut = ligne.split(" ");
//Lire la 8eme ligne contenant les tables contenant chaque attribut
ligne = br.readLine( ) ;
tTableOfAttribut = ligne.split(" ");
//Lire la 9eme ligne contenant les TABLES DE DIMENSION
ligne = br.readLine( ) ;
tDim = ligne.split(" ");
//Lire la 10eme ligne contenant les tailles des tables de dimension
ligne = br.readLine( ) ;
tTailleTableDim = ligne.split(" ");
//Lire la 11eme ligne contenant les tailles de tuples de chaque table de dimension
ligne = br.readLine( ) ;
tTailleTupleDim = ligne.split(" ");
//Lire la 12eme ligne contenant les facteurs de selectivité de jointura avec la table des faits
ligne = br.readLine( ) ;
tSelectivityF = ligne.split(" ");
//Fermeture du fichier
br.close() ;
}
//Gérer les exceptions
catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
}
catch ( IOException e ) {
System.out.println( "Problème à la lecture du fichier" ) ;
}
/**********************************************************************************************
********** TRAITEMENTS EN UTILISANT LES VALEURS LUES A PARTIR DU
FICHIER DE VALEURS
**********************************************************************************************
**********/ mapStrToInt= mapStringInt(tAttributs);
mapSelectivity= mapStringInt(tDim);
//Calcul de la taille en nombre de page systeme de chaque table de dimension
float[] tTailleTableDimPageSysteme= new float[tDim.length];
for (int i = 0; i < tDim.length; i++)
{
tTailleTableDimPageSysteme[i]= Float.parseFloat(tTailleTupleDim [i]) *
Float.parseFloat(tTailleTableDim[i])/(float)pageSysteme;
//System.out.println(" tTailleTableDimPageSysteme: " + i + " = " + tTailleTableDimPageSysteme[i]);
}
/**********************************************************************************************
**********
* TRAITEMENT DU FICHIER CONTENANT LE CONTEXTE D'EXTRACTION (ISSU DE LA CHARGE
DE REQUETE)
**********************************************************************************************
**********/ String contexteFichier = "contexteDExtraction.txt";
//calcul du nombre de requetes
try{
FileReader contexteFile = new FileReader ( contexteFichier ) ;
BufferedReader brContexte = new BufferedReader (contexteFile ) ;
String ligneContexte;
while ((ligneContexte = brContexte.readLine())!=null){
nombreRequete++;
}
//Fermeture du fichier
brContexte.close() ;
```

```
}
//Gérer les exceptions
catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
}
catch ( IOException e ) {
System.out.println( "Problème à la lecture du fichier" ) ;
}
coutRequete = new float[nombreRequete];
coutRequeteSansIndex = new float[nombreRequete];
/**********************************************************************************************
********************** TRAITEMENT DU FICHIER CONTENANT LES MOTIFS (INDEX)
**********************************************************************************************
**********************/
String motifsFichier = "motifs.txt";
//Chargement du fichier contenant les motifs et faire un premier parcours pour calculer le nbr de motifs= nbr
lignes
try{
FileReader motifsFile = new FileReader ( motifsFichier ) ;
BufferedReader br = new BufferedReader (motifsFile ) ;

while ((ligne=br.readLine())!=null){
nbrmotifs++;
}
//Fermeture du fichier
br.close() ;
}
//Gérer les exceptions
catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
}
catch ( IOException e ) {
System.out.println( "Problème à la lecture du fichier" ) ;
}
// initialisation des tables contenant taille Index, cout Acces, cout Chargement
float[] tailleIndex= new float[nbrmotifs];
float[] coutAcces= new float[nbrmotifs];
float[] coutChargement= new float[nbrmotifs];
float[] nR= new float[nbrmotifs];
float tailleConfigIndex=0;
double exponotiel=0;
int sommeCard;
//RE-Chargement du fichier contenant les motifs
try{
FileReader motifsFile = new FileReader ( motifsFichier ) ;
BufferedReader br = new BufferedReader (motifsFile ) ;
String ligne;
nbrmotifs = 0;
while ((ligne=br.readLine())!=null){
//System.out.print(ligne);
//détecter les attributs séparés par de blancs constituants le motif
tMotifs = ligne.split(" ");
//Calcul de la taille de la configuration d'index ET DU COUT D'ACCES
calcul=0;
sommeCard = 0;
exponotiel= 0;
//Calcul de la somme des cardinalités pour l'utiliser dans le calcul de la taille de l'index et du cout D'ACCES
for (int i = 0; i < tMotifs.length; i++)
{
indice= mapStrToInt.get(tMotifs[i]);
sommeCard+= Float.parseFloat(tCard[indice]);
```

```
}
calcul = (float)(sommeCard+80) * (float)nUpletsTFait /8 ;
tailleIndex[nbrmotifs]= (float) calcul ;
tailleConfigIndex+= tailleIndex[nbrmotifs];
//Calcul de cout DE Chargement de l'index
coutChargement[nbrmotifs]= tailleIndex[nbrmotifs]/pageSysteme;
//Calcul de cout D'Acces de l'index
nR[nbrmotifs]= (float)nUpletsTFait / (float)sommeCard;
exponotiel= Math.exp(-((double)nR[nbrmotifs]/(double)pagesTFait));
coutAcces[nbrmotifs] = (float) (pagesTFait *(1-exponotiel)) ;
System.out.print (" *- index: "+ nbrmotifs +" : sommeCard "+ sommeCard +" *nR = "+nR[nbrmotifs]+"
*exponotiel = "+exponotiel System.out.println ( " coutAcces = "+ coutAcces[nbrmotifs] );
nbrmotifs++;
}
System.out.println (" *- Nombre d'index: "+ nbrmotifs);
System.out.println (" *- Taille de la Configuration d'Index: "+ tailleConfigIndex/1000000000 +" GO");
//Fermeture du fichier
br.close() ;
}
//Gérer les exceptions
catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
}
catch ( IOException e ) {
System.out.println( "Problème à la lecture du f ichier" ) ;
}
/**********************************************************************************************
********** creation d'un fichier résumant les résultats
**********************************************************************************************
**********/
ry {
File file = new File("resultatsExecution.txt");
FileWriter outFile = new FileWriter(file);
PrintWriter outputFile = new PrintWriter(outFile);
int j=0;
outputFile.println( " ********************************************************************** " );
outputFile.println( " * Tailles des Tables de Dimension en Pages Systeme ");
outputFile.println( " ********************************************************************** " );
for (int i = 0; i < tDim.length; i++)
{
outputFile.println(" Taille de la Table de Dimension: " + tDim[i] + " = " + tTailleTableDimPageSysteme[i]+ " }
outputFile.println( "
**********************************************************************************************
" outputFile.println( " * Tailles des index en GO, leurs cout de charement et d'acces en nombre de pages systeme
");
outputFile.println( "
**********************************************************************************************
" outputFile.println (" *- Nombre d'index: "+ nbrmotifs);
FileReader motifsFile = new FileReader ( motifsFichier ) ;
BufferedReader br = new BufferedReader (motifsFile ) ;
String ligne;
for (int i = 0; i < nbrmotifs; i++) {
ligne=br.readLine();
outputFile.println ("*- index Numero: "+i+ ", construit sur les attributs: "+ligne);
outputFile.print (" - taille de l'index: " + i + " = " );
outputFile.printf("%.2f",tailleIndex[i]/1000000000 );
outputFile.println (" GO");
outputFile.print (" - cout de Chargement = ");
outputFile.printf("%.6f\n",coutChargement[i]);
outputFile.print (" - cout d'Acces = ");
```

```
outputFile.printf("%.6f\n",coutAcces[i]);
}
br.close() ;
outputFile.println();
//-----------------------------------------------------------------------------------------------------------------
//Calcul du cout de chaque requete sans index
//-----------------------------------------------------------------------------------------------------------------
try{
FileReader contexteFile = new FileReader ( contexteFichier ) ;
BufferedReader brContexte = new BufferedReader (contexteFile ) ;
String ligneContexte;
int requeteNumber=-1;
Collection requete;
Collection motifSet = new ArrayList();
float coutJointure=0;
int indiceTable=0;
//lire requete
j=0;
while ((ligneContexte = brContexte.readLine())!=null)
{ tContexte = ligneContexte.split(" ");
requete = new ArrayList(Arrays.asList(tContexte));
Set<String> tableSet = new HashSet<String>();
coutJointure=0;
requeteNumber++;
// Calcul du cout de jointure 3* (pages tabFait + somme pag sys tab dimention)
coutJointure = pagesTFait;
Iterator<String> it = requete.iterator();
while (it.hasNext()) {
tableSet.add(tTableOfAttribut[mapStrToInt.get(it.next())]);
}
String[] tableToJoin = tableSet.toArray(new String[0]);
// Calcul du cout de jointure 3* (pages tabFait + somme pag sys tab dimention)
coutJointure = pagesTFait;
for (int ii = 0; ii < tableToJoin.length; ii++)
{
indiceTable = mapSelectivity.get(tableToJoin[ii]);
coutJointure += tTailleTableDimPageSysteme[indiceTable];
}
coutJointure = coutJointure *3;
coutRequeteSansIndex[requeteNumber]= coutJointure;

coutChargeRequeteSansIndex+=coutRequeteSansIndex[requeteNumber];
}
//Fermeture du fichier
brContexte.close() ;
}
//Gérer les exceptions
catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
}
catch ( IOException e ) {
System.out.println( "Problème à la lecture du fichier" ) ;
}
outputFile.println( );
outputFile.println("-----------------------------------------------------------------------------------------------------------------"
outputFile.println( " *- REQUETE | COUT | senario | Tables Joint | index utilises | COUT outputFile.println("----
-----------------------------------------------------------------------------------------------------------------" try{
FileReader contexteFile = new FileReader ( contexteFichier ) ;
BufferedReader brContexte = new BufferedReader (contexteFile ) ;
String ligneContexte;
motifsFile = new FileReader ( motifsFichier ) ;
```

```
BufferedReader brMotif = new BufferedReader (motifsFile ) ; boolean[] indexSelected;
//System.out.println( " *- Nombre de Requetes: " + nombreRequete ) ;
int requeteNumber=-1;
//lire requete
j=0;
while ((ligneContexte = brContexte.readLine())!=null){
tContexte = ligneContexte.split(" ");
requeteNumber++;
j++;
outputFile.print( " - Q" + j + " " );
//choisir les index
String ligneMotif=null;
int[] nbrAttribIndex=new int[nbrmotifs];
indexSelected=new boolean[nbrmotifs];
Collection requete = new ArrayList(Arrays.asList(tContexte));
Collection motifSet = new ArrayList();
Collection temp_req = new ArrayList();
Collection senario1 = new ArrayList(requete);
Collection[] attributJoint;
Set<String> tableSet = new HashSet<String>();
int indiceMax;
int nbrIndexAvec0attributCommun;
int indiceTable=0;
float coutJointure=0;
for (int iteration = 1; iteration <= nbrmotifs+1; iteration++)
{
//System.out.println( "iteration: " + iteration ) ;
motifsFile = new FileReader ( motifsFichier ) ;
brMotif = new BufferedReader (motifsFile );
nbrIndexAvec0attributCommun=0;
//Rechercher le motif -index- contenant le plus d'attribut
for (int i = 0; i < nbrmotifs; i++)
{
ligneMotif= brMotif.readLine();
tMotifs = ligneMotif.split(" ");
motifSet = new ArrayList(Arrays.asList(tMotifs)); //créer une collection avec le motif
temp_req = new ArrayList(requete);
//System.out.println( " *- requete: " + requete ) ;
temp_req.retainAll( motifSet );
//System.out.println( " *- temp_req: " + temp_req ) ;
nbrAttribIndex[i]= temp_req.size();
if (temp_req.isEmpty()) // Si le nbr d'attribut commun = 0 incrémenter
{
nbrIndexAvec0attributCommun++;
//System.out.println( " *- nbrIndexAvec0attributCommun: " + nbrIndexAvec0attributCommun ) ;
if (nbrIndexAvec0attributCommun == nbrmotifs) // Si aucun index ne contien d'attributs, tester si senario 1 ou 3
{
iteration = nbrmotifs;
senario1.removeAll(requete);
if (senario1.isEmpty())
{
//*****************************************************************
// Fin du traitement de la requete, Calculer: sénario 1;
//*****************************************************************
//Recherche des tables à joindre, des attributs de la meme table peuvent exister dans m requete
Iterator<String> it = requete.iterator();
while (it.hasNext()) {
tableSet.add(tTableOfAttribut[mapStrToInt.get(it.next())]);
}
//System.out.println(" tableSet: " + tableSet);
```

```
String[] tableToJoin = tableSet.toArray(new String[0]);
// Calcul du cout de jointure 3* (pages tabFait + somme pag sys tab dimention)
coutJointure = pagesTFait;
for (int ii = 0; ii < tableToJoin.length; ii++)
{
indiceTable = mapSelectivity.get(tableToJoin[ii]);
coutJointure += tTailleTableDimPageSysteme[indiceTable];
}
coutJointure = coutJointure *3;
//System.out.println(" *- cout de la requete: " + requeteNumber + " = cout de jointure = " + coutJointure);
coutRequete[requeteNumber]= coutJointure;
outputFile.print(coutRequete[requeteNumber]+ "\t" );
outputFile.print("jointure seulement \t \t" );
outputFile.print(tableSet + "\t\t\t\t\t ----------");
outputFile.println(" " + coutRequeteSansIndex[requeteNumber] );
nbrReqNonOptimise++;
iteration = nbrmotifs+1;
}else{
//*****************************************************************************
//System.out.println(" Fin du traitement de la requete, Calculer: sénario 3");
*****************************************************************************
for (int iii = 0; iii < nbrmotifs; iii++)
{
if (indexSelected[iii]==true)
{
coutRequete[requeteNumber]+= coutChargement[iii] + coutAcces[iii];
}
}
Iterator<String> it = requete.iterator();
while (it.hasNext()) {
tableSet.add(tTableOfAttribut[mapStrToInt.get(it.next())]);
}
String[] tableToJoin = tableSet.toArray(new String[0]);
coutJointure = nR[0]*74/pageSysteme;
for (int ii = 0; ii < tableToJoin.length; ii++)
{
indiceTable = mapSelectivity.get(tableToJoin[ii]);
coutJointure += tTailleTableDimPageSysteme[indiceTable];
}
coutJointure = coutJointure *3;
coutRequete[requeteNumber]+= coutJointure;
outputFile.print(coutRequete[requeteNumber] + " ");
outputFile.print("index + jointure\t\t" );
outputFile.print(tableSet + "\t\t\t\t\t" );
for (int iii = 0; iii < nbrmotifs; iii++)
{
if (indexSelected[iii]==true)
{
outputFile.print(iii + " " );
}
}
outputFile.println(" " + coutRequeteSansIndex[requeteNumber] );
nbrReqPartielOptimise++;
iteration = nbrmotifs+1;
}
}
}
if (iteration <= nbrmotifs)
{
indiceMax = max (nbrAttribIndex, tailleIndex);
```

```
indexSelected[indiceMax]=true;
motifsFile = new FileReader ( motifsFichier ) ;
brMotif = new BufferedReader (motifsFile );
for (int i = 0; i <= indiceMax; i++)
{
ligneMotif= brMotif.readLine();
}
tMotifs = ligneMotif.split(" ");
motifSet = new ArrayList(Arrays.asList(tMotifs));
requete.removeAll( motifSet );
if (requete.isEmpty()) // Si y a plus d'attributs sortir
{
iteration = nbrmotifs+1981;
//*********************************************************************************************
**
//System.out.println(" Fin du traitement de la requete, Calculer: sénario 2");
*********************************************************************************************
*
for (int i = 0; i < nbrmotifs; i++)
{
if (indexSelected[i]==true)
{
coutRequete[requeteNumber]+= coutChargement[i] + coutAcces[i];
}
}
outputFile.print(coutRequete[requeteNumber] );
outputFile.print(" index seulement " );
outputFile.print(" ------- " );
for (int iii = 0; iii < nbrmotifs; iii++)
{
if (indexSelected[iii]==true)
{
outputFile.print(iii + " " );
}
}
outputFile.println(" " + coutRequeteSansIndex[requeteNumber] );
nbrReqCouvertes++;
//outputFile.println();
}
}
}
coutChargeRequete+=coutRequete[requeteNumber];
}
//Fermeture du fichier
brContexte.close() ;
brMotif.close() ;
}
//Gérer les exceptions
catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
}
catch ( IOException e ) {
System.out.println( "Problème à la lecture du fichier" ) ;
}
outputFile.println( );
outputFile.println( "
*********************************************************************************************
" outputFile.println( " *- Nombre de Requetes: " + nombreRequete ) ;
outputFile.println();
outputFile.print (" *- Taille de la Configuration d'Index = ");
```

```
outputFile.printf("%.6f",tailleConfigIndex/1000000000);
outputFile.println (" GO");

outputFile.println();
outputFile.print( " * cout des requetes sans index: ");
outputFile.printf("%.2f\n",coutChargeRequeteSansIndex);
outputFile.println();
outputFile.print( " * cout des requetes avec index: " );
outputFile.printf("%.2f\n",coutChargeRequete);
outputFile.println( "
**********************************************************************************************
");
System.out.print( " *- cout de la Charge de Requete = " );
System.out.printf("%.2f\n",coutChargeRequete);
outputFile.println( );
System.out.print( " *- coutChargeRequeteSansIndex = " );
System.out.printf("%.2f\n",coutChargeRequeteSansIndex);
System.out.println("nbrReqNonOptimise = "+ nbrReqNonOptimise);
System.out.println("nbrReqCouvertes = "+ nbrReqCouvertes);
System.out.println("nbrReqPartielOptimise = "+ nbrReqPartielOptimise);
//Fermeture des fichiers
outputFile.close();
//Gérer les exceptions
} catch( FileNotFoundException e ) {
System.out.println(" Fichier non trouvé " ) ;
} catch ( IOException e ) {
System.out.println( "Problème à la lecture du fichier" ) ;
}
//Fin du création du fichier de sortie
}
}
```

Code source du module « Génération du fichier ARFF »

```
package extract;

import java.io.*;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

/**
 *
 * @author dell
 */
public class Extract {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundException, IOException {

      String NomReq = "c:\\contexte_extraction\\requetes.txt";
      BufferedReader Req  = new BufferedReader(new FileReader(NomReq));
      String line_Requetes;
      Set les_attrib =new HashSet(); // on crée notre Set
      while ((line_Requetes = Req.readLine())!= null)
      {     String[] Liste_attributs = line_Requetes.trim().split("\\s");
           for (int x2=0;x2<Liste_attributs.length;x2++)
           { //9
                les_attrib.add(new String(Liste_attributs[x2].toUpperCase()));

           } //9

      };
      Req.close();

      String Fich_out = "c:\\contexte_extraction\\attributs.txt";
      PrintWriter out  = new PrintWriter(new FileWriter(Fich_out));

      PrintWriter mon_f;
      mon_f = new PrintWriter(new FileWriter("c:\\contexte_extraction\\ozb.arff"      ));
      mon_f.println("@relation Entrepot");
      mon_f.println(" ");


      Iterator i=les_attrib.iterator(); // on crée un Iterator pour parcourir notre HashSet
      String Table_att = ""  ;int x2=1 ;
while(i.hasNext()) // tant qu'on a un suivant
{
    //out.print (i.next());
    Table_att =  Table_att + " " +(String) i.next();


}
out.println ( Table_att );
out.close();
 String[] Table_att_x  = Table_att.trim().split("\\s");
       for (x2=0;x2 < Table_att_x.length;x2++){
       System.out.print(x2);
       System.out.print(" ");
       System.out.println(Table_att_x[x2]);
```

```
      mon_f.println("@attribute "+Table_att_x[x2] + " "+"{1}");
      }

//  Construction de mon fichier
     String Ligne_result="";String [] Liste_attributs;
     NomReq = "c:\\contexte_extraction\\requetes.txt";
     Req  = new BufferedReader(new FileReader(NomReq));
     line_Requetes=""; int ind = 0;int cpt ;int x1;       boolean bool;

                 mon_f.println("");
                 mon_f.println("@data");
     String verg = ",";
     while ((line_Requetes = Req.readLine())!= null)
     {     Liste_attributs = null;
          Liste_attributs = line_Requetes.trim().split("\\s");
          bool = false; cpt = 0;verg= ",";
          Ligne_result = "";
          for (x2=0;x2<Table_att_x.length;x2++)
            { //90
                cpt = 0;
                for ( x1=0;x1<Liste_attributs.length;x1++)
                { //9
                   if (Table_att_x[x2].toUpperCase().equals(Liste_attributs[x1].toUpperCase()))
                      {  bool = true; cpt = 1;                          }
                } //9
                if (x2 == Table_att_x.length-1)
                {verg = "";}
                if (cpt == 1)
                   {   Ligne_result= Ligne_result+ "1" + verg ;                    }
                else
                   {   Ligne_result= Ligne_result+ "?" + verg ;                    }
                bool = false;
             } //90

      System.out.println(Ligne_result);
      mon_f.println(Ligne_result);
      ind = ind + 1;
     };

     Req.close();

// Fin Construction de mon fichier

                                  mon_f.close();
    }
}
```

Code source du « Construction de la configuration d'index »

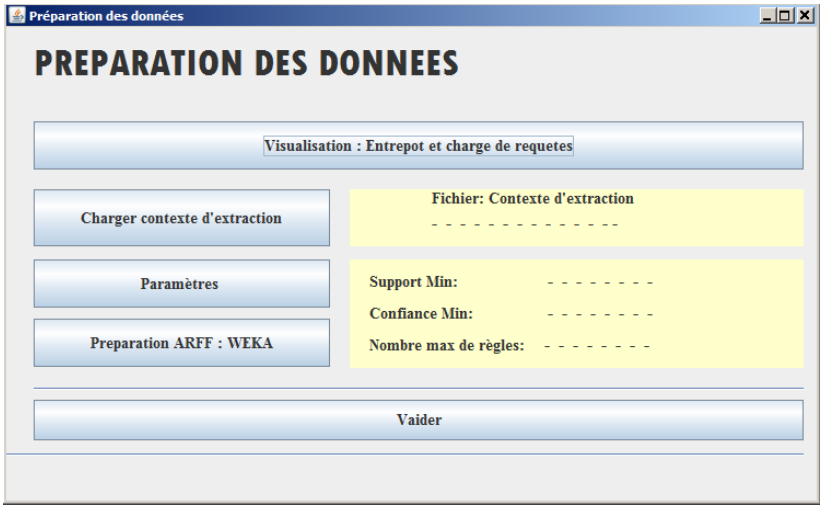
```
package motifs;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

/**
 *
 * @author dell
 */
public class Motifs {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundException, IOException {
        // TODO code application logic here


      String NomReq = "c:\\motifs\\0.50.txt";
      BufferedReader Mot  = new BufferedReader(new FileReader(NomReq));
      String line_motifs,lesmotifs;
      Set les_attrib =new HashSet(); // on crée notre Set
      boolean trouve = false;String permut = "";
      while (((line_motifs = Mot.readLine())!= null) && (trouve == false))
      {

       if (line_motifs.trim().equalsIgnoreCase("Best rules found:")  )
       { trouve = true;
       }
      }
      int nbligne = 0;
      while ((line_motifs = Mot.readLine())!= null)
      {
        nbligne = nbligne + 1 ;
        lesmotifs = ""; permut="";
         String[] Liste_attributs = line_motifs.trim().split("\\s");
           for (int x2=0;x2<Liste_attributs.length;x2++)
           { //9
                 if (Liste_attributs[x2].indexOf("=1")>0)
                 {

                 lesmotifs = lesmotifs + Liste_attributs[x2].substring(0, Liste_attributs[x2].indexOf("=1"))+" " ;
                 }
           } //9
       Liste_attributs = lesmotifs.trim().split("\\s");
       for (int cpt=1;cpt<=12;cpt++)
       {
         for (int x2=0;x2<Liste_attributs.length;x2++)
         {
```

```
            if  (Liste_attributs[x2].equals(Integer.toString(cpt)))
                  {permut = permut + Integer.toString(cpt)+" ";}
          }
       }
       System.out.print  (nbligne + " : ")       ;
       System.out.println(lesmotifs) ;
       les_attrib.add(permut.trim());

     };
      Iterator itr = les_attrib.iterator();
       while(itr.hasNext()) {

        Object element = itr.next();
        System.out.println(element + " ");

        }


     les_attrib.clear();
     Mot.close();



   }
}
```

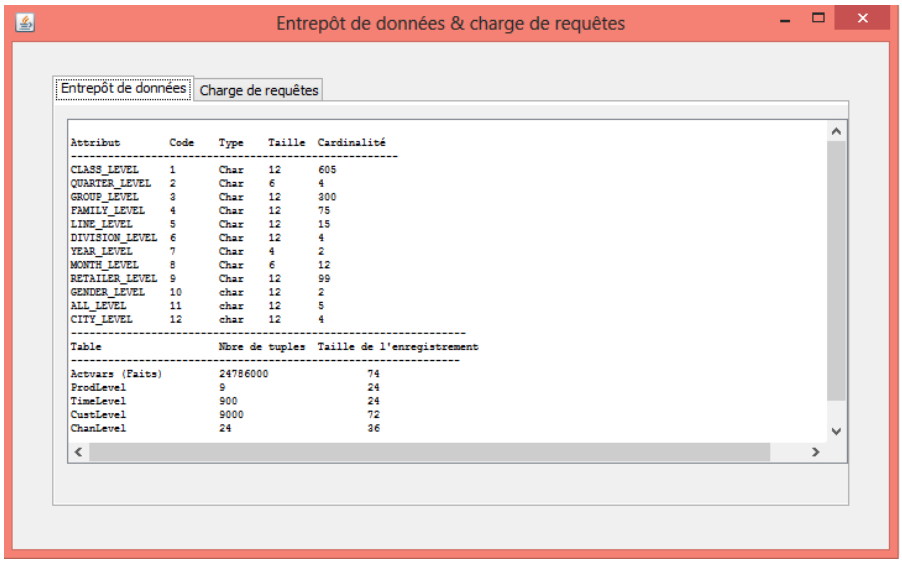
Présentation des interfaces de l'outil « *ISARules* »

1 Interface principale

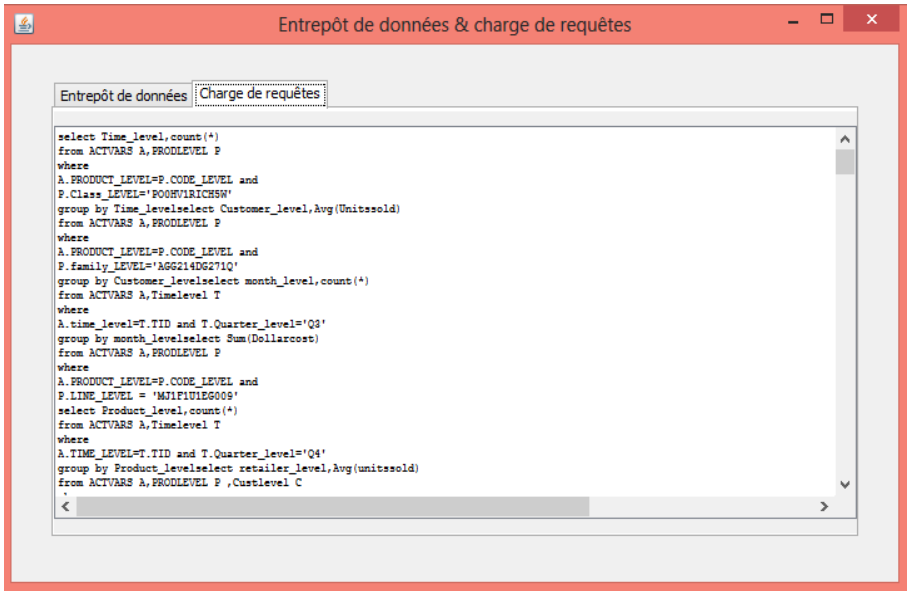
2 Chargement des données



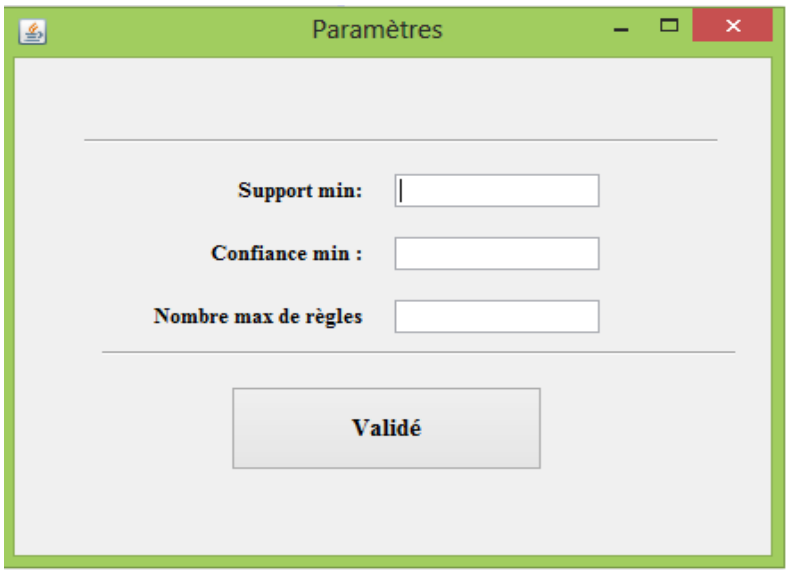
3 Visualisation de l'entrepôt de données



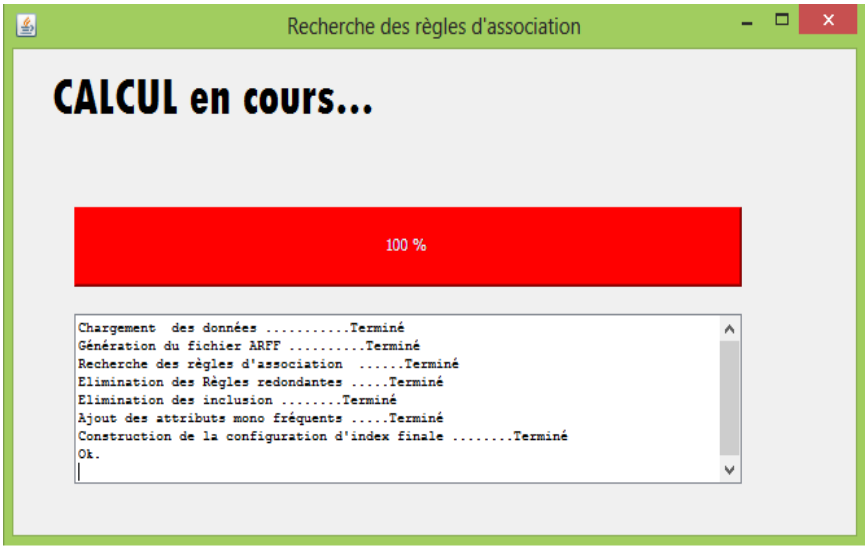
4 Visualisation de la charge des requêtes



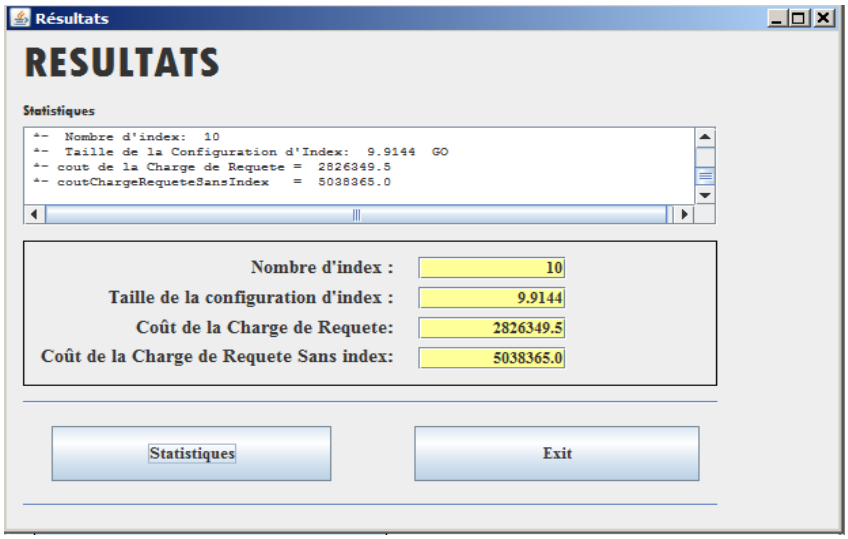
5 Paramétrage



6 Module « Calcul »



7 Résultats


Résultats
RESULTATS
Statistiques
*-- Nombre d'index: 10
*-- Taille de la Configuration d'Index: 9.9144 GO
*-- cout de la Charge de Requete = 2826349.5
*-- coutChargeRequeteSansIndex = 5038365.0
Nombre d'index :
10
Taille de la configuration d'index :
9.9144
Coût de la Charge de Requete:
2826349.5
Coût de la Charge de Requete Sans index:
5038365.0
Statistiques
Exit