وزارة التعليم العالي و البحث العلمي
جامعة وهران للعلوم و التكنولوجيا محمد بوضياف

THÈSE

En vue de l'obtention du

Diplôme de Doctorat en Sciences

Présentée par : Hichem Chaalal

***Intitulé*: Adaptation de la fragmentation verticale sur les bases de données Orientées lignes.**

Devant le Jury Composé de :

Faculté ***: Mathématique et Informatique***
Département ***: Mathématique et Informatique***
Spécialité ***: Informatique***
Option ***: Informatique***

Membres de Jury	*Grade*	*Qualité*	*Domiciliation*
Président	*Pr*	*Rahal Sid ahmed*	*USTO-MB*
Directeur de thèse	*Pr*	*Hafida Belbachir*	*USTO-MB*
Examinateur	*Pr*	*Nait Bahloul Safia*	*U-Oran1*
Examinateur	*Pr*	*Babahamed Latéfa*	*U-Oran1*
Examinateur	*Pr*	*Belkadi Khaled*	*USTO-MB*
Examinateur	*Dr*	*Sehaba Karim*	*Univ-Mosta*
Invité	*Pr*	*Nicolas Travers*	*ESILV Sorbonne*

Année Universitaire : 2020/2021

Dédicace

Je dédie ce présent mémoire à ma bien aimée mère Chergui Fadhila Houria, elle qui nous a quittée soudainement, elle qui m'a éduquée l'honnêteté, le respect et l'amour d'autrui, Allah yarhmak.

A mon père Nacer Edine, qui m'a appris tout ce que je suis, m'a appris le savoir et l'amour du savoir.

A ma femme Fatima, et a mes fils Omar et Mohamed, qui j'espère deviendrons meilleur que leur père des disciples et amoureux de la sagesse et la science.

A toute ma famille, mes frères et ma sœur Mahjouba.

Remerciements

Je tiens à remercier toute personne qui a contribué de prés ou de loin à l'aboutissement de ce modeste travail, ami, personnel de l'administration, spécialement à Madame Halima Maazouzi, qui on soutenus ma quête durant toutes ces années, à tous les faiseurs de miracles, et ceux qui les aident.

Un grand merci à Professeur Belbachir qui a mis toute sa confiance dans mes idées et imaginations, et à accepter de m'encadrer et de me soutenir durant toute cette thèse, et qui m'a permis de réaliser ce travail.

Professeur Nicolas Travers, qui grâce à lui et à travers son soutient, m'a orienter et a partagé toutes ses connaissances, je tiens à le remercier pour avoir investis beaucoup de temps à discuter et étudier toutes mes idées, et qui le fais toujours avec moi.

Sommaire

Chapitre I : Introduction

I-1. INTRODUCTION

A chaque instant on constate que tout existant fait l'objet d'un état ou d'une action, ces représentations peuvent être interprétées en un historique, un historique qui peut être traduit et décomposé en informations, qui peuvent être matérialisées et transformées en un ensemble de données.

La connaissance d'un sujet, ou d'un phénomène quelconque permet de comprendre son comportement passé, son évolution et surtout la prédiction à un certain degré de son état futur, ou celui d'un cas similaire.

Ainsi, se doter d'un système d'information ou d'une base de données est devenu une exigence pour la gestion et le pilotage des organisations et des sociétés. Conséquence de tout cela, le marché de l'information connait un énorme essor, et la demande sur les systèmes d'information ne cesse de croitre.

L'économie de l'information et les technologies de communications se développent très rapidement, ainsi l'utilisation des systèmes d'information et des systèmes de gestion de bases de données est devenue indispensable pour les différents acteurs de l'économie : organisations, structures publiques et scientifiques.

Aujourd'hui les systèmes d'information gérés par les applications dite systèmes de gestion de bases de données sont devenus indispensable pour toute entreprise, une existence virtuelle de l'organisme, un support permettant de disposer de l'information sous différentes formes indépendamment de l'espace temporel et géographique, voir même une source de richesse, et surtout une condition suffisante pour la subsistance, l'évaluation et le développement des entreprises.

Le marché de l'information connait une grande évolution, presque quasi présent dans tous les secteurs, il est devenu un facteur commun et critique, et la demande sur les systèmes d'information ne cesse de croitre, mais cette convoitise et cette demande croissante et pressante entraine derrière elle beaucoup de défis pour les concepteurs et éditeurs des systèmes d'information, et remet en question les solutions proposées. En effet les éditeurs SI (systèmes d'information) et des bases de données sont amenés à résoudre plusieurs contraintes et à proposer des solutions adéquates pour satisfaire la demande et les perspectives souhaitées par les usagers et consommateurs.

L'évolution et la diversification des besoins des utilisateurs et des sources de données a fait jaillir plusieurs orientations SGBD spécialisées sur plusieurs secteurs, les utilisateurs de SGBD se sont orientés sur plusieurs domaines comme : calcul scientifique, Big_data, data mining, modèle OLAP, modèle OLTP, etc. Conséquence de cela plusieurs variantes de SGBD ont émergé se spécialisant pour les besoins d'un domaine précis dont nous citons : SGBD orientés colonnes, SGBD distribués, SGBD orientés clé valeur, orientés document, NoSQL, in Memory, et bien d'autres, ces SGBD nécessitent des compétences techniques bien pointues et un équipement assez

couteux pour faire fonctionner ces solutions, ce qui n'est pas à la porté de la majorité des usagers.

Plusieurs secteurs, plusieurs utilisations et objectifs, il est devenu difficile de faire le bon choix du modèle SGBD correspondant le mieux aux besoins et objectifs de l'entreprise, et souvent les entreprises on un besoin occasionnel pour l'utilisation de certains SGBD, ce qui nécessite de se doter de beaucoup de technologies et du matériel et compétences techniques qui en suit, et bien souvent les entreprises ont tendance à utiliser plusieurs SGBD pour chaque besoin, ou sous traiter leurs travail à d'autres sociétés spécialisés, ce qui n'est pas à la porter de toute les collectivités et organisations, ni bien recommandé pour gérer les données confidentielles.

Le modèle SGBD le plus courant et le plus utilisé est le modèle SGBD relationnel orienté ligne classique, c'est un des modèles permettant de gérer les données d'une organisation moyenne pour des besoins bien limité, souvent non adapté aux besoins pointus comme OLAP ou Big Data. Mais nous avons l'intime conviction que Les SGBD classiques peuvent encore subvenir aux besoins d'une grande variante de domaine, sans recourir à de nouvelles technologies couteuses, car ils utilisent une grande variété de techniques leurs permettant d'augmenter leurs performances et un passage à l'échelle aisé, des techniques comme index, cluster, fragmentation verticale et horizontale, etc.

Dans cette étude nous allons proposer une technique innovante qui permet de casser l'aspect naïf des SGBD classiques, et leur donner une allure bien élégante et flexible, permettant d'élargir le champ d'action des SGBD classiques, en permettant de ne travailler que sur une seule plateforme. Notre objectif est d'ouvrir de nouvelles voies de recherches et des horizons applicables sur plusieurs types SGBD, ce qui permettra d'augmenter les performances des SGBD classiques.

Spécialement nous nous sommes intéressés à la technique de la fragmentation verticale, une technique existante, mais dont l'application reste restreinte à quelques essais et nouvellement proposée par les SGBD classiques avec modération, pourtant, l'efficacité de cette technique a permis de développer le secteur OLAP, les modèles analytiques, et les données massives. Cette technique a inspiré les concepteurs des bases de données, et a permis de créer une nouvelle tendance les SGBD orientés colonnes et In Memory, des SGBD qui ont réussi à faire leurs preuves sur le marché et un traitement bien au delà des espérances des SGBD classiques.

Comparé aux SGBD classiques, les SGBD orienté colonnes et In Memory effectuent un traitement très impressionnant en un temps éclair des requêtes OLAP sur des volumes de données massives, un traitement impossible à atteindre avec les modèles classiques. En comparant les deux modèles nous avons l'impression que les SGBD classiques ont pour objectif d'exécuter correctement les requêtes, sans se soucier du temps de traitement, un traitement avec une solution correcte point, qui dure pour des requêtes plusieurs heures, à l'inverse les nouvelles conceptions permettent un

traitement efficace sur des temps record en quelques minutes, et sur les mêmes plateformes, toutefois les SGBD classiques restent performant et efficace pour le traitement transactionnel.

Dans cette étude, nous nous somme fixés pour objectif de donner un nouveau souffle aux bases de données classiques, notre approche se base sur l'étude de l'impact de la fragmentation verticale sur les performances des bases de données, et d'adapter cette technique sur les bases de données classiques. Pour se faire nous avons étudié et testé différents algorithmes et approches, afin d'aboutir à un schéma de fragmentation verticale optimisé, et d'augmenter les performances de la base de données, et en second lieu nous proposons une nouvelles conception physique améliorée et mieux adaptée aux SGBD classiques.

L'optimisation du schéma FV « fragmentation verticale » est obtenue par l'application des algorithmes d'optimisation, nous avons implémenté les algorithmes Génétiques et l'algorithme Apriori et l'algorithme Affiner. Affiner est une nouvelle approche que nous avons développé, une méthode qui est simple à implémenter et ne nécessite aucun paramétrage, et permet de générer un schéma de fragmentation verticale optimisé.

Pour l'adaptation de la FV on s'est basé sur l'implémentation d'une fragmentation verticale classique et nous avons ajouté une nouvelles structure physique dont l'utilisation reste optionnelle, mais qui a fais ses preuves sur le modèle orienté lecture comme OLAP avec le benchmark TPCH.

En résumé, notre contribution s'est formalisée sur deux volets, le premier concerne une nouvelle approche dite Affiner permettant l'optimisation de la structure physique d'une base de données par la fragmentation verticale. Le deuxième volet concernera la nouvelle conception physique T-Plotter, une nouvelle technique physique permettant d'améliorer les performances d'une base de données verticalement fragmentée, et de combler les défaillances d'une fragmentation verticale classique.

Le présent mémoire est organisé en chapitre de la manière suivante :

Chapitre I État de l'art

Dans le chapitre II, nous développerons l'état de l'art sur la technique de la fragmentation verticale et les différentes conceptions physiques. Nous commençons par présenter les bases de données et les entrepôts de données où nous mettons l'accent sur la conception physique et les différentes techniques d'optimisation utilisées.

Nous nous concentrons ensuite sur une technique d'optimisation importante qui est la fragmentation verticale et les différents travaux effectués pour sélectionner un schéma de fragmentation verticale.

Contribution 1 : Algorithme Affiner

Le chapitre III représente la première partie de notre contribution. Nous commençons par présenter en détail les deux algorithmes de sélection d'un schéma de fragmentation verticale que nous avons implémenté : un algorithme génétique et un algorithme de fouille de données « data mining » : Apriori.

Nous présentons par la suite l'algorithme Affiner que nous avons proposé. L'explication de cet algorithme sera illustrée à travers un exemple détaillé.

Contribution 2 T-Plotter

Le chapitre IV traitera la deuxième partie de notre contribution, à savoir la technique de conception physique T-Plotter.

Nous présentons l'architecture générale de T-Plotter, les principales fonctionnalités ainsi que les résultats de tests effectués sur un benchmark.

Conclusion et perspectives

Enfin, la partie conclusion qui récapitule ce qui a été réalisé, et nous proposons quelques perspectives et proposition pour continuer sur la même voie.

Chapitre II : Etat de l'art

II-1. INTRODUCTION

Aujourd'hui les éditeurs SI (systèmes d'information) et des bases de données sont amenés à résoudre plusieurs contraintes et proposer des solutions pour satisfaire les demandes et les besoins des usagers de SGBD.

Sur le marché, on constate que les sources de données et le volume d'information exploité ne cesse de croitre, et que le coût de stockage des données ne cesse de baisser, ce qui rajoute plus de contraintes et de problèmes pour les concepteurs et éditeurs, comme le stockage des données, latences du système, réduction des performances.

Une des solutions qui se présente pour résoudre la latence du système et de recourir à un bon paramétrage de la conception physique de la base de données, en effet une bonne conception physique permet d'augmenter les performances du système.

La conception physique est une étape essentielle dans le cycle de vie d'une base de données, elle inclut la sélection d'index, fragments, clustering, etc. La conception physique permet la gestion des méthodes de stockage et d'accès physique des données sur disque, ce qui permet d'augmenter et d'optimiser les capacités d'une base de données, et permet d'optimiser les performances des applications fonctionnant dessus. En effet, si la conception physique n'est pas bien ajustée, les performances de la base de données peuvent chuter considérablement et vice versa.

La conception physique est une tache compliquée, c'est elle dépend de plusieurs paramètres, souvent difficiles à manipuler et à réguler. Il est bon à savoir que l'administration des SGBD diffère pour chaque système et nécessite une formation approfondie et une grande expérience pour en devenir expert et certifié.

L'ajustement de ces paramètres se répercute souvent sur les différentes architectures proposées et sur les performances de la base de données. Aussi, il est à noter que l'impact de la conception physique se fait ressentir plus sur les bases de données volumineuses que les bases de données non volumineuses.

Les chercheurs de diverses disciplines scientifiques se basent dans leurs études et recherches sur le regroupement du maximum d'information sur un phénomène ou une entité. L'étude, l'analyse et la compréhension de ces informations permet d'avoir une idée globale sur l'existence du phénomène, et sa compréhension. Cela permettra de développer le modèle mathématique ou des équations d'états de ce dernier, pour l'aide à la décision.

Autre facteur important qui a joué un très grand rôle dans le développement et l'expansion des systèmes d'informations et les bases de données, est celui de la baisse du coût du matériel informatique, précisément le coût de stockage a connus une décroissance ces dernières décennies.

L'étude faite par Lyman et Varian [43] à l'université de Berkley, nous dévoile la baisse du coût de stockage, et l'augmentation du volume de stockage (voir la Figure 2 :), il est à noter que cette étude date d'il y a dix ans.

Dans la Figure 1 nous voyons bien que durant ces dernières années le coût de stockage a baissé d'une manière impressionnante, et le volume d'informations utilisé ne cesse de croitre exponentiellement.

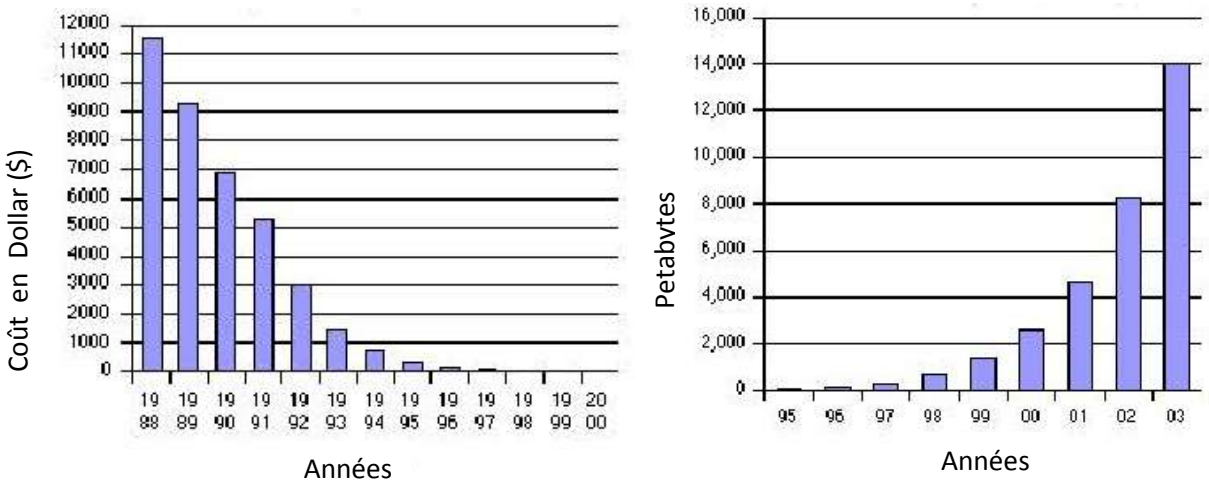


Figure 1: Réduction du coût de stockage, Croissance du taux de stockage 43

Dans le même contexte, le Tableau 1 exprime bien la croissance du volume d'information en chiffre pour l'année 1999, il faut s'attendre à ce que ce volume augmentera d'une façon exponentiel ces dernières années.

Cette croissance se fait ressentir plus dans les pays industrialisés là où les prises de décisions et la gestion des ressources se base sur des systèmes d'informations.

Selon la source web1 rien qu'EBay à lui seul traite un volume de données, plus de 50 pétaoctets2 d'informations quotidiennes tout en ajoutant 40 téraoctets de données chaque jour, tandis que des millions de personnes vendent et achètent des éléments dans plus de 50 000 catégories. S'ensuit les sites de réseaux sociaux comme facebook, et plein d'autres systèmes.

[1] http://www.decideo.fr/eBay-et-Teradata-s-allient-pour-developper-des-solutions-approfondies-en-termes-d-analyse-des-donnees_a3723.html.

[2] 1 Pétaoctet = 10^3 Téraoctet = 10^6 Gigaoctet.

Support de Stockage	Type du contenu	Estimation sup: Téraoctet / année	Estimation min: Téraoctet / année	Taux de croissance
Papier	Livres	8	1	2
	Journaux	25	2	-2
	Périodiques	12	1	2
	Documents officielles	195	19	2
	Sous-Total	240	23	2
Film	Photos	410,000	41,000	5
	Cinéma	16	16	3
	X-Rays	17,200	17,200	2
	Sous-Total	427,216	58,216	4
Optique	CD-Musique	58	6	3
	CD-Données	3	3	2
	DVD	22	22	100
	Sous-Total	83	31	70
Magnétique	Cassette Caméscope	300,000	300,000	100
	Disquette pilotes PC	766,000	7,660	100
	Serveurs départementaux	460,000	161,000	100
	Serveurs entreprise	167,000	108,550	55
	Sous-Total	2.120,539	635,480	50
Total		2.120,539	635,480	50

Tableau 1: Volume d'informations dans le monde, 1999. [43]

Il est devenu indispensable de se doter d'un système d'information ou d'une base de données qui gère l'ensemble des données, et c'est l'une des raisons qui a joué un très grand rôle dans la prolifération des systèmes d'informations, ce qui a ouvert un vaste marché fruiteux pour les concepteurs et éditeurs des systèmes d'information.

Les bases de données représentent la matérialisation des systèmes d'informations sur des supports électroniques.

Nous présentons dans la section suivante les bases de données et les systèmes logiciels qui les gèrent.

II -2. LES BASES DE DONNÉES

Par définition, une base de données est une collection ordonné et structuré représentant la mémorisation des données. Elle permet le stockage, la manipulation et la gestion des données, c'est une sorte de projection structurée et ordonnée d'une réalité organisationnelle qui évolue dans le temps.

Les bases de données peuvent avoir une architecture centralisée ou distribuée, selon les nécessités et les besoins de l'organisation.

« Historiquement, c'est en 1970, dans une thèse mathématique, qu'Edgar F.Codd (1923-2003), chercheur chez IBM, propose d'utiliser les informations sous forme d'enregistrements pour assurer les liens entre les informations et de regrouper les enregistrements dans des tables »[3].

Ce regroupement est motivé par le fait que le résultat de chaque recherche dans une base de données est une table. Le concept est repris par Lawrence Ellison qui créa une startup devenue Oracle Corporation. Cette proposition est la base des modèles de données relationnelles, modèles utilisés par la quasi-totalité des systèmes de gestion de base de données actuels. Dans ces modèles, tout comme dans les modèles réseaux, les entités sont reliées par des associations selon une organisation arbitraire.

Cependant contrairement à ce dernier, les informations d'associations ne sont pas des pointeurs mais des informations contenues dans les enregistrements que le système de gestion de base de données utilise pour effectuer des produits cartésiens.

Le modèle entité-association a été inventé par Peter Chen en 1975, il est destiné a clarifié l'organisation des données dans les bases de données relationnelles.

Dans le modèle relationnel, la relation désigne l'ensemble des informations d'une table, tandis que l'association, du modèle entité-association, représente le lien logique qui existe entre deux tables avec des informations en commun.

Les premières bases de données étaient calquées sur la présentation des cartes perforées : répartis en lignes et colonnes de largeur fixe.

Une telle répartition permet difficilement de stocker des objets de programmation. en particulier, elles ne permettent pas l'héritage entre les entités, caractéristique de la programmation orientée objet.

Apparues dans les années 1990, les bases de données objet-relationnel utilisent un modèle de données relationnel tout en permettant le stockage des objets.

L'un des modèles de base de données le plus utilisé est OLTP.

OLTP (On Line Transaction Processing)[4] :

OLTP est une variété de base de données dont le mode de fonctionnement est transactionnel. L'objectif d'OLTP est de pouvoir insérer, modifier et interroger rapidement et en sécurité la base. Ces actions doivent pourvoir être effectuées très rapidement par de nombreux utilisateurs simultanément.

[3] http://fr.wikipedia.org/wiki/Base_de_données

[4] http://datawarehouse4u.info/OLTP-vs-OLAP.html

Chaque transaction opère sur de faibles quantités d'informations, et toujours sur les versions les plus récentes des données. Les modifications et les suppressions des données sont fréquentes dans OLTP.

Dans ces bases de données les associations d'héritage des objets s'ajoutent aux associations entre les entités du modèle relationnel.

II- 2- A. SYSTEME DE GESTION DE BASES DE DONNEES

Le système de gestion de base de données (Data Base Management System) est une solution logicielle qui permet toutes les opérations sur les bases de données : nous citons par exemple la création de la base de données, le stockage, le déploiement, la sécurité, Tuning [5] et bien d'autres opérations.

Un SGBD rend les informations disponibles et partagées pour différents usagers, ces derniers accèdent aux sources d'informations sur le SGBD via des applications qui effectuent l'interrogation de la base par les requêtes (SQL ou autres), le SGBD devient en quelque sorte un serveur pour répondre aux requêtes des utilisateurs.

Les nouvelles stratégies et orientations des entreprises exigent un accès garanti et universel aux données de l'organisation.

La disponibilité de l'information représente un grand intérêt pour les utilisateurs, et permet un rapprochement entre la décision à l'action.

Un SGBD doit garantir la bonne mémorisation des données ainsi que leur cohérence, aussi il doit permettre aux usagers de retrouver aisément les informations qu'ils cherchent en un temps acceptable.

Gérer un grand volume de données entre des milliers d'usagers en simultanéité pour différents lieu et garantir la cohérence et l'efficacité n'est pas une chose évidente.

Chaque opération engagée par l'utilisateur peut représenter une transaction financière ou autre information d'une importance capitale, et malheur arrive si des erreurs ou des pertes se produisent.

Ce qui pourrait couter cher à l'organisme gérant, par conséquent, un SGBD ne doit tolérer aucune erreur, et garantir l'efficacité et la disponibilité des données.

De plus en plus les entreprises recourent pour le pilotage et la prise de décision aux systèmes d'aide à la décision basés sur l'analyse des données historiques de l'activité de l'entreprise, on appelle ces systèmes les entrepôts de données.

[5] Tuning: réglage des ressources et des paramètres physique de la base de données afin d'optimiser les performances de la base.

II- 2- B. ENTREPÔT DE DONNÉES (DATA WAREHOUSE)

Les entrepôts de données sont des bases de collections des données, qui stockent les données par rapport à la dimension historique. L'analyse de données permet de récolter des informations utiles pour la prise de décision de l'entreprise.

Le concept des entrepôts de données peut être représenté par une centrale de collections de données, dont le traitement et l'analyse permet d'en extraire des informations et des connaissances pour aide à la décision, au moyen d'outils d'analyse et de synthèse (*data mining*).

Le concept des entrepôts de données a vu le jour en 1980 quand les chercheurs d'IBM Barry Devlin et Paul Murphy ont développé le " business data warehouse ". Le concept d'entrepôt de données avait pour but de fournir un modèle pour enregistrer et traiter le flux de données provenant de systèmes opérationnels pour approvisionner l'environnement d'aide à la décision.

Les entrepôts de données sont orientés sujets, leurs objectifs est de servir les besoins de l'entreprise, en fournissant un support pour l'aide à la décision.

Ils sont composés de plusieurs tables de dimensions et une table centrale nommée table de faits, ces derniers sont accédés par des requêtes décisionnelles complexes caractérisées par de multiples opérations de sélection, de jointure et d'agrégation.

Il existe quatre modèles de schéma d'entrepôts de données le schéma étoile, schéma flocon de neige, schéma en galaxie, et schéma en constellation.

Souvent cité parmi les processus décisionnels les plus utilisés, les systèmes OLAP se basent et exploitent les entrepôts de données.

OLAP (On line Analitycal Processing) [13] est une couche qui se trouve généralement au dessus d'un entrepôt de données. L'entrepôt de données permet de fournir l'infrastructure permettant à OLAP de puiser les données analysées, il est possible qu'OLAP puisse puiser ces sources de données à partir des bases de données de type OLTP.

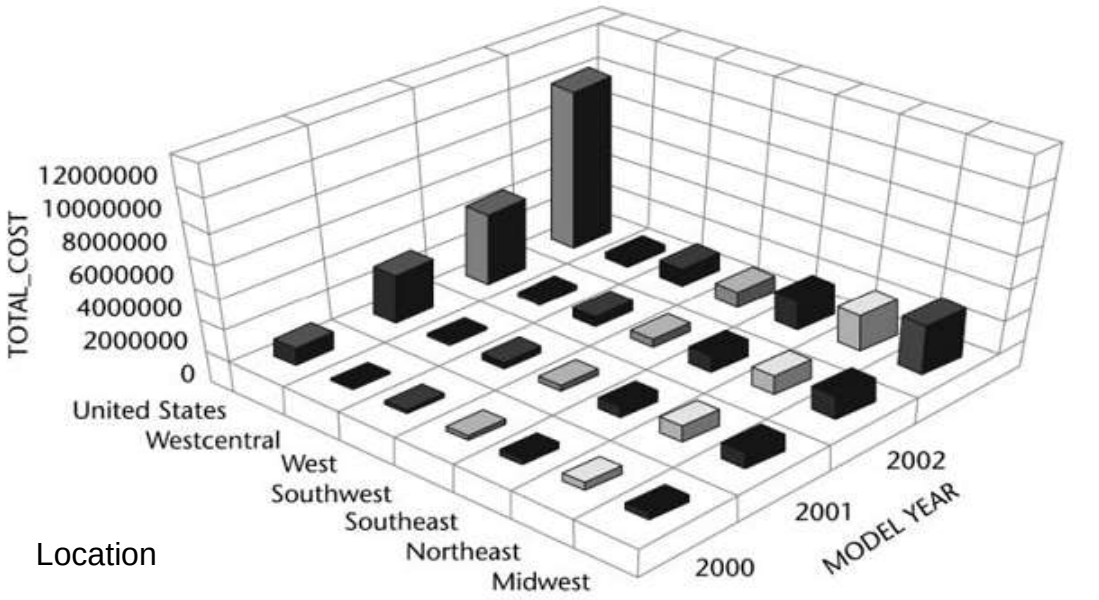
Les entrepôts de données contiennent souvent des centaines de millions de tuples de données de dates antérieur représentant des données historiques, Et pour répondre aux requêtes posées directement à l'encontre des données détaillées peut consommer des ressources informatiques importantes.

OLAP est caractérisé par de grands volumes d'informations traitées, peu de mise à jour, et des insertions fréquentes.

Le but d'OLAP est de répondre aux requêtes rapidement sur une grande quantité de données sous-jacentes. En général, les requêtes SQL posées sur les systèmes OLAP sont les clauses "group by" de certains attributs, et l'application des fonctions d'agrégation sur d'autres attributs.

Par exemple, un gérant peut-être intéressé par la recherche du coût total, regroupé par année et par région.

La plupart des systèmes OLAP offre une représentation graphique des résultats (voir l'exemple de la Figure 2 :).



[6]

Figure 2 : exemple d'un rapport obtenu à partir d'un processus OLAP [4]

Le graphe à trois dimensions dans la Figure 2, représente un rapport obtenu d'un traitement à partir d'un système OLAP.

Il est constitué de trois axes qui sont : l'axe [MODEL YEAR] qui indique les résultats calculés et regroupés par année, l'axe [LOCATION] indiquant les résultats calculé et regroupés par ville, et le troisième axe [TOTAL_COST] indiquant le coût total par ville et par année.

Les systèmes OLAP sont organisés d'une manière à faciliter la manipulation des données. OLAP facilite Les opérations CRUD[7].

Les systèmes OLAP permettent d'explorer de grandes quantités de données facilement et rapidement. La plupart des systèmes OLAP se basent sur l'enregistrement des résultats sommaires sur des vues matérialisées.

A partir des vues matérialisées on extrait des données sommaires, elles sont généralement beaucoup plus petites que les tableaux contenant les informations détaillées. Quand une requête est posée, les vues matérialisées sont utilisées pour

[6] Source: Cognos PowerPlay.

[7] CRUD : Create, Replace, Update, Delete.

obtenir une réponse rapide, en évitant le calcul des résultats sur d'énorme quantité de données sous-jacentes.

Il existe trois catégories générales de mécanismes de stockage dans les systèmes OLAP: OLAP relationnel (ROLAP), OLAP multidimensionnel (MOLAP) et OLAP hybride (HOLAP).

ROLAP utilise des tables relationnelles pour stocker des données, typiquement en utilisant les dimensions comme une clé primaire composite. Chaque ligne représente une cellule du cube de données. Un cube est la table qui contient les données sommaires (les vues matérialisées).

Les cellules vides ne sont pas explicitement représentées dans ROLAP. MOLAP stocke les données sur le disque d'une façon organisée par rapport à l'emplacement dans la table. Chaque cellule est représentée par un emplacement fixe calculable.

Les cellules vides consomment de l'espace dans les représentations MOLAP. Il est possible d'utiliser des techniques de compression pour réduire l'espace perdu dans MOLAP.

Il est généralement reconnu que ROLAP est plus adapté sur les SGBD dont les données sont espacées et éparses, et MOLAP est meilleur lorsque les données sont denses.

Une stratégie utile pourrait consister à stocker les données éparses utilisant ROLAP, et les données denses en utilisant MOLAP. Cette approche hybride est appelé HOLAP.

L'objectif des bases de données OLTP est de fournir des réponses rapides à des requêtes simples, exemple : les ventes des produits cosmétiques.

Les bases OLAP permettent des requêtes plus complexes à qui s'ajoute la notion de périodes: comme les ventes des produits Cosmétiques par vendeur, région et par mois.

La conception physique des bases de données représente l'un des maillons dans le cycle de vie d'une base de données, elle inclut la sélection d'index, de partition, de cluster, etc.

Le cycle de vie d'une base de données intègre quatre étapes : la définition des besoins ou cahier de charge, l'étape de la conception logique, l'étape de la conception physique, et enfin la mise en œuvre et la réalisation de la base de données.

La conception physique de base de données débute après l'étape de la définition et la normalisation des tables, et permet l'optimisation des performances de la base de données.

Une explication plus détaillée sur la conception physique est présentée dans la section qui suivra.

II -3. LA CONCEPTION PHYSIQUE

La conception physique des bases de données a pour rôle la gestion des méthodes de stockage et d'accès physique des tables sur disque, ce qui permet à la base de fonctionner avec une grande efficacité.

L'objectif de la conception physique consiste à optimiser les performances de la base de données et des applications fonctionnant sur cette base.

La conception physique sur les bases de données est une rude tâche, que les gestionnaires et les administrateurs de bases de données maitrisent peu. Elle implique des dizaines et souvent des centaines de variables et contraintes, qui sont difficiles à gérer et à régler, sachant bien que ces modifications sont très souvent liées aux différentes architectures proposées.

L'intervention sur les différentes architectures et configurations de la conception physique d'une base de données se répercute sur les performances de cette dernière.

Les calculs individuels de performance basée sur un mécanisme d'indexation donnée ou d'un algorithme de fragmentation peuvent prendre plusieurs heures à la main, et l'analyse de la performance est souvent basée sur la comparaison de nombreuses configurations différentes et des conditions de charge, ce qui nécessite beaucoup de calculs et le résultat n'est pas toujours garanti.

L'étude de Lyman et Varian [43] a révélé que le taux de stockage ne cesse de croitre, ce qui influe négativement sur les performances des bases de données, d'où la nécessité d'une intervention permanente du gestionnaire de la base de données pour optimiser la conception physique et les performances de la base.

Dans cette étude nous nous somme intéressés à l'optimisation des performances de la base de données par la conception physique.

II- 3- A. TECHNIQUES POUR LA CONCEPTION PHYSIQUE

Plusieurs techniques de conceptions physiques existent dont nous citons l'indexation, les vues matérialisées, le clustering, fragmentation, etc.

L'index est une organisation de données mis en place pour accélérer la recherche (requête) des données de tables. Dans les systèmes de gestion de base de données, les index peuvent être spécifiés par les programmeurs d'applications de base de données en utilisant les commandes SQL[8].

Il existe plusieurs sortes d'index tel que l'index unique, index binaire, index de jointure, index de jointure binaire, etc.

[8] Structured Query Language

Lorsque une ou plusieurs tables sont interrogés, le résultat peut être stocké dans ce qu'on appelle une **vue matérialisée**. Les vues matérialisées sont stockées sous forme de tables dans la base de données comme n'importe quelle autre table.

Dans les entrepôts de données, des vues matérialisées sont maintenus comme des agrégats de données dans les tables de base.

Le **Clustering** multidimensionnel (MDC) est une technique par laquelle les données peuvent être regroupés selon les dimensions, tels que l'emplacement, le calendrier, ou type de produit.

En particulier, le MDC permet aux données d'être regroupés selon plusieurs dimensions en même temps.

La technique de fragmentation détailler dans le prochain paragraphe car elle fait l'objet de notre étude.

Il existe d'autres techniques d'optimisation de la conception physique pour rendre l'accès aux données plus efficaces sur une base de données.

A titre d'exemple, la compression de données est une technique qui permet de stocker plus de données dans espace disque réduit, les données comprimées sont donc accessibles plus rapidement.

Dans ce qui suit nous présenterons les techniques de fragmentation sur les bases de données.

II- 3- B. TECHNIQUES DE FRAGMENTATION

La fragmentation est une technique qui consiste à décomposer les objets de la base (index, table, vue matérialisée, …) en sous ensembles pour permettre une exploitation meilleure des données et optimiser le temps d'exécution de la charge de travail qui représente l'ensemble des requêtes à laquelle la base de données elle est soumise.

Pour qu'une fragmentation soit correcte il faut qu'elle respecte les règles suivantes :

- Complétude : la fragmentation doit être complète, donc chaque élément du schéma doit au moins appartenir à un fragment du sous schéma.
- Reconstructible : après avoir fragmenté physiquement un schéma donné, on doit pouvoir recomposer le schéma initial à partir des fragments générés.

II.3.B.1. INTÉRÊTS DE LA FRAGMENTATION

La fragmentation sur les bases de données nous permet de gagner sur plusieurs plans, elle avantage une amélioration de la performance du système, la

fragmentation rend les données disponibles pour les bases de données distribuées, en favorisant les accès locaux.

Aussi, en appliquant la fragmentation, la requête n'accède qu'aux données requises pour son exécution, cela permet de réduire énormément le nombre d'accès physique aux fichiers de données et de réduire les traitements et les calculs au niveau du processeur, ce qui permet à la fois d'augmenter le débit du système et réduit l'exécution des requêtes.

Aussi, cela permet à quelques transactions concurrentes de s'exécuter parallèlement, puisque chaque transaction accède à un fragment séparé physiquement de l'autre.

Grace à la fragmentation la capacité et le coût de stockage deviennent plus équilibrés et moins condensés, ce qui permet une plus grande disponibilité des données. Cela est particulièrement adaptée aux architectures de disques comme le RAID (réseaux redondants de disques indépendants) où les données peuvent être accessibles en parallèle sur plusieurs disques d'une manière organisée.

Un système fragmenté est beaucoup plus tolérant aux pannes, il est robuste et moins vulnérable, car si un fragment d'un objet de la base est endommagé, le reste des fragments n'en serra pas affecté.

La fragmentation offre un autre avantage pour le cas des bases de données distribuées, elle permet la réduction des transactions de communication entre les sites, qui est d'une manière générale assez considérable, ce qui signifie que les performances de la base et le temps de réponse seront meilleurs.

II.3.B.2. INCONVÉNIENTS DE LA FRAGMENTATION:

La fragmentation sur les bases de données nous apporte des avantages mais aussi des inconvénients, il est à considérer que le schéma de fragmentation se base sur l'analyse de la base de données mais aussi sur la charge de travail, et dans un environnement où les requêtes changent fréquemment il est déconseillé de modifier la conception physique de la base de données à chaque variation de la charge de travail.

La fragmentation d'une base de données est une arme à double tranchant, en effet, elle peut jouer un rôle positif sur la performance de la base si elle est bien implémentée, mais elle peut jouer un rôle négatif si la fragmentation n'est pas adéquate au système. C'est pour cette raison qu'il faut se doter d'outils performants pour l'assistance à la fragmentation.

Il faut prendre en considération qu'une fois la base de données fragmentés le schéma de la base sera changé, et le chemin pour l'accès aux données changera, c'est pour cette raison qu'il faut penser à reproduire les clés primaire des relations,

préserver l'intégrité référentielle entre les relations en dupliquant sur chaque fragment les clés étrangère, et enfin réécrire les requêtes selon la nouvelle conception.

Il est à noter que pour certaine bases de données supportant la fragmentation ces indications seront prises en charge par le SGBD, qui se chargera de ces indications.

Enfin, pour le cas des bases de données distribuées, il peut s'ajouter le problème d'allocation, le bon fragment au bon endroit (site), et là aussi il est conseillé de se doter d'algorithmes et des outils nécessaires pour mener à bien cette opération.

II.3.B.3. TYPES DE FRAGMENTATION

Il existe trois variantes de fragmentation sur les bases de données :

- La fragmentation horizontale : fragmenter la relation (table de la base) par rapport à ses lignes (enregistrement),
- La fragmentation verticale : fragmenter une relation par rapport à ses colonnes (attributs),
- Fragmentation mixte : une combinaison entre la fragmentation horizontale et la fragmentation verticale.

II.3.B.4. FRAGMENTATION HORIZONTALE

La fragmentation horizontale décompose un objet de la base de données (relation, vue matérialisée, index) en des sous ensembles d'une manière horizontale dans le sens des enregistrements, chaque fragment représente une sélection selon un prédicat.

Les fragments étant le regroupement d'un sous ensemble d'enregistrements de l'objet initial.

L'opération union permet de reconstruire le schéma à partir de fragments horizontaux.

Il existe deux variantes de fragmentation horizontale :

- Fragmentation horizontale primaire : on effectue la fragmentation horizontale sur la base de prédicats de sélection définis sur la relation ou l'objet lui même.
- Fragmentation horizontale dérivée : dans cette variante, la décomposition est effectuée sur une relation R en utilisant des prédicats définis sur une autre Relation R'.

L'exemple suivant nous permettra de bien comprendre la fragmentation horizontale d'une table.

ILLUSTRATION :

Soit la table Client (Code, Nom, Prénom, Genre, Coordonnés), sur la base du genre féminin ou masculin du client.

Soit les requêtes suivantes :

Q1: SELECT * FROM **CLIENTS** WHERE **GENRE = F;**

Q2: Select * FROM **CLIENTS** WHERE **GENRE = M;**

Pour exécuter ces deux requêtes le système doit parcourir toute la table client, et vérifier sur chaque table le genre F ou M, ce qui nécessite beaucoup d'opérations et d'accès physiques aux fichiers de la base.

Une des solutions qui se présente à l'administrateur de la base est de fragmenter horizontalement la relation Client selon les prédicats de sélection :

(GENRE = F) & (GENRE = M).

Clients

Code	Nom	Prénom	Genre	Téléphone
1	Ali	Noura	F	0772151201
2	Bacha	Khaled	M	0551232045
3	Chari	Safia	F	0660151520
4	Nouria	Abed	M	0778452115
5	Laribi	Mourad	M	0771528962

3_ a

Clients_H1

Code	Nom	Prénm	Genre	Tlphn
1	Ali	Noura	F	0772151201
3	Chari	Safia	F	0660151520

3_ b

Clients H2

Code	Nom	Prénm	Gnr	Tlphn
2	Bacha	Khaled	M	0551232045
5	Laribi	Mourad	M	0771528962
4	Nouadria	Abed	M	0778452115

Figure 3: illustration de la fragmentation Horizontale.

La Figure 3_a, représente une instance de la relation client initiale.

Dans la figure 3_b, les deux tables Clients_H1 et Clients_H2 représentent les deux fragments générés sur la base de la table Client.

Une fois la table partitionnée, le SGBD réécrit les requêtes pour qu'elles puissent s'exécuter correctement sur le nouveau schéma et les fragments qui lui sont valides[9], les deux requêtes seront réécrites comme suit :

Q1: **select** * **from** Clients_H1;

Q1: **select** Nom **from** Clients_H1;

[9] Un fragment est valide pour une requête Q si et seulement si Q accède au moins à un n-uplet du fragment.

De cette manière nous éviterons le parcours entier de la table client, pour l'exécution des requêtes, ce qui signifie un gain en temps de parcours de la table, ce qui réduit énormément le temps d'exécution des requêtes.

II.3.B.5. FRAGMENTATION VERTICALE

La fragmentation verticale est une technique utilisée pour exécuter efficacement le modèle OLAP et les requêtes qui invoquent certaines colonnes de la table. Les requêtes OLAP sont connues pour être complexes et nécessitent des opérations d'agrégat. En général, elles ne doivent analyser que quelques colonnes de la table. La technique de fragmentation verticale permet donc de travailler sur un ensemble de colonnes [14], au lieu d'appeler la table entière avec un SGBDR classique, et cette méthode de traitement naïve a un impact négatif sur le temps de traitement des requêtes, en particulier sur les grandes tables ou les méga données.

La fragmentation verticale a été largement étudiée et elle peut résoudre de nombreux problèmes auxquels les concepteurs et les utilisateurs de bases de données sont confrontés, rarement proposées par les SGBD classiques [1314], en particulier lorsqu'ils passent du modèle OLTP au modèle OLTP / OLAP, en raison des besoins d'entreprise en matière d'analyse et de Data Mining.

Nous avons pour motivation et objectif d'apporter une solution et d'aider à promouvoir et à faire revivre la technique de fragmentation verticale. Elle sera donc généralisée et davantage utilisée dans le monde des SGBDR et contribuera à combler certaines lacunes.

La fragmentation verticale d'une table T permet de décomposer la relation T en un ensemble de fragments (T1,…, Tn) regroupant chacun un ensemble d'attributs. Pour chaque fragment la clé primaire de la table T doit être reproduite (afin de repérer les tuples[10] dans les sous fragments). Les fragments verticaux sont obtenus en utilisant l'opération de projection de l'algèbre relationnelle.

L'opération de jointure permet de reconstruire le schéma à partir de fragments verticaux, mais il est nécessaire que pour chaque fragment vertical généré a partir d'une relation R il va falloir dupliquer la clé primaire de la relation R.

Pour mieux comprendre la fragmentation verticale sur les bases de données, nous vous proposons de consulter l'illustration suivante.

[10] Instance d'une relation.

A. *ILLUSTRATION :*

Dans cet exemple nous traiterons le concept de la fragmentation verticale sur la table Client, le but par l'application de la fragmentation verticale est de réduire les coûts (temps de réponse) des requêtes Q1' et Q2', avec :

Clients (Code, Nom, Prénom, Genre, Coordonnés)

Q1 ': **select** Code, Nom, Prénom **from** Clients ;

Q2' : **select** Code, Genre, Coordonnés **from** Clients ;

La Figure 4_a, représente une instance de la table Clients. Le SGBD lors de l'exécution des requêtes Q1' et Q2' parcours toute la table, et des colonnes non utile aux requêtes seront chargées.

En fragmentant la table Clients verticalement (Figure 4_b), le SGBD ne chargera que les données utiles à l'exécution des requêtes, ce qui réduit le temps d'exécution des requêtes et l'accès physique aux tables.

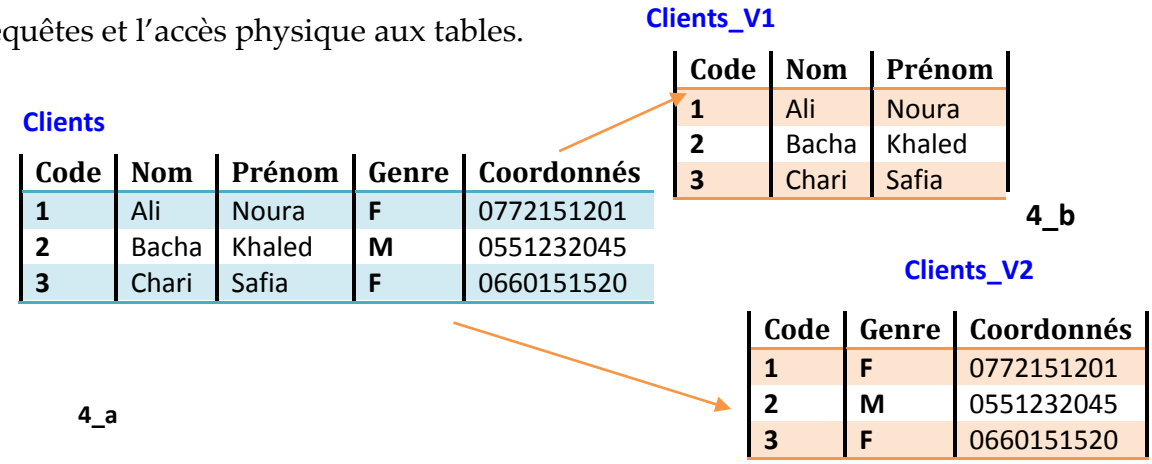


Figure 4 : Illustration de la fragmentation verticale

Nous devons réécrire les requêtes pour qu'elles puissent s'exécuter correctement sur le nouveau schéma.

Q1': **select** * **from** Clients V1;

Q2': **select** * **from** Clients V2;

La fragmentation verticale sur les tables relationnelles est complexe. Ceci est essentiellement dû au grand nombre d'alternatives possibles :

– En partitionnant horizontalement: Özsu et al [42] on montré que pour N prédicats simples, le nombre de fragments possibles est 2^N; dans le cas où les prédicats contradictoires existes ce nombre diminue par l'élimination de ces derniers.

– Par contre si une table R à m attributs non clés primaires, l'ensemble des possibilités de fragmentations verticales est calculé par $B(m) \approx m^m$, pour de grande valeur de m (Hammer & Niamir [33]).

B(M) étant le nombre de Bell, les valeurs suivantes nous permettent d'avoir une idée sur la grandeur du nombre de Bell où :

$m = 5 ; \quad B(m) \approx 52.$

$m = 10 ; \quad B(m) \approx 115{,}000.$

$m = 15 ; \quad B(m) \approx 1{,}38*10^{9}.$

$m = 22 ; \quad B(m) \approx 4{,}50*10^{15}.$

$m = 30 ; \quad B(m) \approx 8{,}46*10^{23}.$

Ainsi, pour un schéma de base de données composé de n relations, le nombre de fragments possibles devient très important, et il faut prendre en considération les fragments de chaque relation le nombre de possibilités est donné par: B(A1)*B(A2)… B(An), donc pour un schéma de 10 relations, composées de 15 attributs, le nombre possible de fragments est de $(10^{10})^9$.

Le nombre de fragments verticaux possibles augmente avec des grandeurs exponentielles, ce qui rend presque impossible l'évaluation du meilleur schéma de fragmentation pour les tables composées d'un nombre important de colonnes.

Par conséquent, obtenir le meilleur schéma de fragmentation verticale par le calcul et l'évaluation de toutes les combinaisons n'est pas évident, c'est pour cette raison que nous sommes obligés de recourir à des heuristiques pour obtenir des solutions performantes avec des temps raisonnables.

En pratique pour implémenter la fragmentation verticale sur les BD classiques Il existe trois façons : conception basée sur les vues matérialisées, conception basée sur l'index sur chaque attribut, conception basée sur des sous tables.

II.3.B.6. SIMULATION DE LA FRAGMENTATION VERTICALE SUR LES SGBD CLASSIQUES :

Dans cette section, nous allons discuter sur plusieurs techniques différentes qui peuvent être utilisées pour implémenter une simulation d'une fragmentation verticale sur un SGBD en ligne.

- La manière la plus simple de simuler une fragmentation verticale sur un SGBD classique consiste à partitionner verticalement une relation en sous-tables, chaque sous-table doit être composée d'au moins un attribut de la relation fragmentée.

Un mécanisme est nécessaire pour reconstruire les champs de même rang dans le bon ordre. Pour ce faire, la méthode la plus simple consiste à ajouter un attribut rang à chaque fragment vertical indiquant la position, qu'il est souvent préférable de considérer comme clé primaire car les clés primaires d'origine peuvent être grandes et parfois composites.

- Plans basés sur des index: L'idée est de créer un index Btree pour chaque fragment vertical de la table, de sorte que pour toute requête d'un attribut donné le compilateur doit passer par l'index, réduisant le temps d'accès aux attributs du fragment.

Bien que cette approche soit intéressante d'un point de vue théorique, elle pose quelques problèmes pratiques. Tout d'abord, l'attribut rang est requis pour chaque colonne afin de permettre la reconstruction de la table initiale, ce qui augmente l'espace de stockage et aura un impact négatif sur le chemin de l'index. Puisque les index ne stockent pas de valeurs ciblées cela implique que pour chaque prédicat ou accès à un attribut par index nous aurons un coût supplémentaire pour accéder à l'index et accéder à la valeur de l'attribut, cette approche nécessite l'analyse de l'index pour extraire les valeurs, qui peut être plus lent que le chemin direct d'un fichier (comme cela se produirait dans l'approche de la sous-table). Enfin, la reconstruction des tuples nécessite une jointure sur les index des attributs pertinents, et il a été remarqué que cette opération est très coûteuse par rapport à une jointure ordinaire sur des sous-tables.

- Vues matérialisées: La troisième approche est basée sur des vues matérialisées. Nous créons un ensemble optimal de vues matérialisées pour chaque type de requête dans la charge de travail. La vue matérialisée optimale peut être définie par une vue matérialisée avec seulement les colonnes pertinentes qui répondent à une requête définie du même type, ce qui nécessite une analyse de charge de travail et un classement des requêtes dans plusieurs catégories, cependant, cette méthode nécessite de connaître à l'avance la charge de travail des requêtes, rendant cette stratégie intéressante uniquement dans des situations de contraintes.

De plus, chaque vue matérialisée est adaptée à une requête ou à quelques requêtes, donc si dans notre système nous aurons un grand nombre de requêtes, nous aurons besoin d'un grand nombre de vues matérialisées, ce qui sature l'espace de stockage et surtout pour des masses de données volumineuses, et nécessite un grand nombre d'opérations pour maintenir une simple mise à jour ou insertion, et

enfin si, malheureusement, une nouvelle requête nécessite une jointure de deux vues, on peut s'attendre à un grosse chute de performance.

En conclusion, le moyen le plus efficace et le plus pratique de simuler la fragmentation verticale d'une base de données est la méthode des sous-tables, qui semble être la moins coûteuse en termes de stockage sur disque et la plus rapide en termes de traitement des requêtes. Dans ce contexte nous proposons la méthode affiner qui permet de choisir le schéma de fragmentation verticale le plus efficace.

Nous présentons dans la section suivante, une étude détaillée sur des travaux proposés pour sélectionner un schéma de fragmentation verticale, ainsi qu'une étude sur les conceptions physiques basée sur les sous table.

II.3.B.7. TRAVAUX SUR LA FRAGMENTATION VERTICALE

La fragmentation verticale sur les bases de données a fait l'objet de plusieurs travaux pour différents contextes.

Tous les algorithmes proposés dans ces études convergent vers l'optimisation des performances de la base de données, mais les approches et les orientations des chercheurs diffèrent, et peuvent être classées en trois types.

Algorithmes basés sur les affinités entre les attributs : Ces algorithmes calculent de l'affinité entre les attributs, et regroupent les attributs les plus proches dans la même partition, plusieurs auteurs ont étudié cette approche dont : Hoffer & al [34, 35] Navathe et al [40], Navathe et Ra [41].

Algorithmes basés sur le modèle de coût : ces algorithmes effectuent l'évaluation de chaque solution (schéma de fragmentation) selon un modèle de coût qui estime le temps de réponse des applications et requête s'exécutant sur la base de données, parmi ces travaux on cite: Hammer et Niamir [33] , Cornell et Yu [26], Chu [25].

Algorithmes basés sur la fouille de données : Les travaux de Cheng et al. [24], Travaux de Gorla [32], Song et Gorla [31] se sont basés sur cette méthode, il procède par la génération des partitions en se basant sur les méthodes de fouille de données.

A. APPROCHES BASEES SUR LA MATRICE D'AFFINITE

Nous allons développer dans la section suivante le principe de la matrice d'affinité et les quelques recherches qui s'y sont basées:

A.1. PRINCIPE DE LA MATRICE D'AFFINITE :

Plusieurs recherches dans le domaine de fragmentation verticale des bases de données proposent d'utiliser la matrice d'affinité entre les attributs, en commençant

par la matrice d'utilisation : pour le cas de n attributs, la matrice d'affinité (MA) est une matrice carré n x n, chaque valeur dans la matrice MAij représente le facteur d'affinité entre les attributs Ai et Aj, et qui est calculé par le nombre d'accès des transactions référençant les deux attributs Ai et Aj.

Les fréquences d'accès des requêtes sont prises en considération afin de regrouper les attributs dans les mêmes fragments accédés fréquemment et simultanément. Soit P_{ij} la probabilité que les attributs A_i et A_j soient nécessaires pour la même requête. Une fonction de coût fondée sur cette hypothèse est déduite, cette fonction reflète la quantité prévue de données qui doivent être transmises afin de répondre à la requête. L'objectif ici est de choisir une partition qui minimise cette fonction de coût.

Les algorithmes basée sur la MA commencent par une matrice d'usage des attributs (AUM). AUM est une matrice, qui a des attributs pour colonnes, et requêtes pour lignes et fréquence d'accès des requêtes comme valeurs dans la matrice. La plupart des algorithmes de fragmentation emploient une matrice d'affinité des attributs (AAM) dérivée de l'AUM fourni comme entrée, cette dernière est obtenue par l'application de l'équation 1.

Les résultats des différents algorithmes (Hoffer & al [34] ,Navathe et al [40]) sont parfois différents même pour la même matrice d'affinité des attributs puisque les fonctions objectives employées par ces algorithmes sont différentes.

La majeure partie des algorithmes de fragmentation verticale basés sur l'affinité n'ont pas de mécanisme pour évaluer la « qualité » des partitions générées [6].

L'affinité entre les attributs mesure le lien entre deux attributs d'une relation en prenant en compte comment ils sont accédés par l'application.

Exemple : Dans ce qui suit nous allons donnée un exemple sur la l'application de l'algorithme basé sur l'affinité sur une table (relation) d'une base de donnée.

Soit la table R composée de 10 attributs R : (A1, …,A10), les transactions (requêtes) (T1,…, T10) s'exécutent sur la base de données englobant la relation avec des fréquences indiquées sur la colonne fréquence sur le Tableau 2.

La matrice d'usage est représentée sur le Tableau 2, la valeur 1 sur la matrice indique que la transaction sur la ligne requiert l'attribut de la colonne, la valeur 0 indique que la transaction ne requiert pas l'attribut sur la colonne.

Transactions	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Fréquence
T1	1	0	0	0	1	0	1	0	0	0	25
T2	0	1	1	0	0	0	0	1	1	0	50
T3	0	0	0	1	0	1	0	0	0	1	25
T4	0	1	0	0	0	0	1	1	0	0	35
T5	1	1	1	0	1	0	1	1	1	0	25
T6	1	0	0	0	1	0	0	0	0	0	25
T7	0	0	1	0	0	0	0	0	1	0	25
T8	0	0	1	1	0	1	0	0	1	1	15

Tableau 2: Matrice d'usage

L'affinité entre deux attributs Ai et Aj est définie par :

$$Aff_{ij} = \sum_{t=1}^{T} q_{t,ij}$$ **Équation 1**

Où $q_{t,ij}$ représente le nombre d'accès des transactions t référençant à la fois i et j.

A titre d'exemple l'affinité entre les attributs A9 et A2 est calculée en appliquant la formule (1), application :

Aff (9,2) = Aff (2,9) = T2 + T5 = 50+ 25 =75.

Attributs	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
A1	75	25	25	0	75	0	50	25	25	0
A2	25	110	75	0	25	0	60	110	75	0
A3	25	75	115	15	25	15	25	75	115	15
A4	0	0	15	40	0	40	0	0	15	40
A5	75	25	25	0	75	0	50	25	25	0
A6	0	0	15	40	0	40	0	0	15	40
A7	50	60	25	0	50	0	85	60	25	0
A8	25	110	75	0	25	0	60	110	75	0
A9	25	75	115	15	25	15	25	75	115	15
A10	0	0	15	40	0	40	0	0	15	40

Tableau 3: Matrice d'affinité.

Une fois la matrice d'affinité obtenue, la prochaine étape consiste à dériver une nouvelle matrice ordonnée, sur la base de la matrice d'usage, et d'en dériver les ensembles conjoints et présentant une affinité proche. Ces derniers représenteront les partitions.

Il existe plusieurs algorithmes qui permettent de générer la matrice ordonnée, nous citons l'algorithme Bond Energie.

A. *L'ALGORITHME DE BOND ENERGIE (BEA) [40]:*

Cet algorithme est employé pour grouper les attributs d'une relation sur la base des valeurs d'affinité entre les attributs dans l'AAM. On le considère approprié pour plusieurs raisons.

- Il est conçu particulièrement pour déterminer des groupes d'items semblables (Il regroupe les attributs qui ont une grande d'affinité ensemble, et ceux avec de petites valeurs ensemble).
- Les groupements finaux sont peu sensibles à l'ordre dans lequel les items sont exposés à l'algorithme.
- AAM est symétrique, ce qui permet la permutation par paires des lignes et des colonnes, qui réduit la complexité.
- La complexité de l'algorithme est raisonnable, de l'ordre $O(n^2)$, où n est le nombre d'attributs.

Cet algorithme prend comme entrée la matrice d'affinité d'attribut, permute ses lignes et colonnes et produit une matrice d'affinité groupée (CAM). La permutation est faite de telle manière à maximiser la mesure globale d'affinité (AM).

La génération de la matrice d'affinité ordonnée se fait en trois étapes :

Initialisation: Placer et fixer une des colonnes AAM arbitrairement dans CAM.

Itération: Sélectionner chacune des colonnes n - i restantes (où i est le nombre de colonnes déjà placées dans CAM) dans les positions i + 1 restantes dans la matrice CAM. Choisir le positionnement qui apporte la plus grande contribution à la mesure globale d'affinité. Continuer ceci jusqu'à ce qu'il ne reste plus de colonnes à placer.

Tri des lignes : Une fois que le rangement des colonnes est déterminé, le placement des lignes devrait suivre les positions des colonnes, car la matrice est symétrique.

La formule qui permet d'évaluer la bonne position est la suivante :

$$\boldsymbol{cont(A_i, A_i, A_k) = 2bond(A_i, A_j) + 2bond(A_j, A_k) - 2bond(A_i, A_k)} \quad \textbf{Équation 2}$$

$A_1,\ldots, A_n$ **:** Attributs numéro « i » d'une relation.

Aff(A_1,A_n) : dite de la matrice d'affinité.

$$\boldsymbol{bond(A_x, A_y) = \sum_{z=1}^{n} aff(A_z, A_x) * aff(A_z, A_y)} \quad \textbf{Équation 3}$$

Après avoir appliqué l'algorithme Bond Energy sur la matrice du Tableau 3: Matrice d'affinité., on obtient la matrice ordonnée suivante :

Attributs	A5	A1	A7	A2	A8	A3	A9	A10	A4	A6
A5	75	75	50	25	25	25	25	0	0	0
A1	75	75	50	25	25	25	25	0	0	0
A7	50	50	85	60	60	15	25	0	0	0
A2	25	25	60	110	110	75	75	0	0	0
A8	25	25	60	110	110	75	75	0	0	0
A3	25	25	25	75	75	115	115	15	15	15
A9	25	25	25	75	75	115	115	15	15	15
A10	0	0	0	0	0	15	15	40	40	40
A4	0	0	0	0	0	15	15	40	40	40
A6	0	0	0	0	0	15	15	40	40	40

Tableau 4 : Matrice d'affinité ordonnée.

Une fois que la matrice ordonnée est obtenue, plusieurs ensembles d'attributs sont formés, et plusieurs partitions sont possibles, pour dérivé les meilleurs partitions possibles on peut utiliser l'algorithme de fragmentation verticale binaire [41].

B. L'ALGORITHME DE FRAGMENTATION VERTICALE BINAIRE [6]:

Le but de cet algorithme est de regrouper les attributs ayant une haute affinité. Le principe du regroupement consiste à trouver un point dans la matrice de façon à diviser la matrice en deux ensembles, par la permutation des colonnes et des lignes de la matrice, jusqu'à l'obtention d'une matrice ordonnée.

Si *At* est l'ensemble des attributs utilisés par la transaction *t*, il est alors possible de calculer les ensembles suivants:

$$\boldsymbol{T = (t \ / \ t \ est \ une \ transaction)}$$

$$\boldsymbol{LT = (t/At \in L)} \qquad \textbf{Équation 4}$$

$$\boldsymbol{UT = (t/At \in U)} \qquad \textbf{Équation 5}$$

$$\boldsymbol{IT = T - (LT \cup UT)} \qquad \textbf{Équation 6}$$

T représente l'ensemble de toutes les transactions. *LT* et *UT* représentent l'ensemble des transactions qui sont concernées par la fragmentation, comme elles peuvent être entièrement traitées en utilisant les attributs de la partie inférieur *L* ou supérieur *U, et qt représente les requêtes.*

$$\boldsymbol{CT = \sum t \in T \ qt} \qquad \textbf{Équation 7}$$

$$\boldsymbol{CL = \sum t \in LT \ qt} \qquad \textbf{Équation 8}$$

$$\boldsymbol{CU = \sum t \in UT \ qt} \qquad \textbf{Équation 9}$$

$$\boldsymbol{CI = \sum t \in IT \ qt} \qquad \textbf{Équation 10}$$

CT compte le nombre total d'accès des transactions à l'objet considéré. *CL* et *CU* compte le nombre total d'accès de transactions qui ont besoin d'un seul fragment; *CI* compte le nombre total d'accès de transactions qui ont besoin de deux fragments. D'une manière générale n-1 emplacements possibles du point (X) le long de la sont considérés, où n est la taille de la matrice (c.-à-le nombre d'attributs). Une partition non recouvrant est obtenue en sélectionnant le point (X) le long de la diagonale de telle sorte que la fonction objectif suivante z soit maximisée:

$$\max Z = CL * CU - CI^2 \qquad \text{Équation 11}$$

La partition qui correspond à la valeur maximale de la fonction (*Z*) est acceptée si la fonction (*Z)* est positive et rejetée dans le cas contraire. La fonction ci-dessus provient d'un jugement objectif empirique de ce qui devrait être considéré comme une "bonne" partitionnement. La fonction est croissante dans *CL* et *CU* et décroissante dans *CI*. Pour une valeur donnée de *CI*, il sélectionne *CL* et *CU* de telle façon que le produit *CL* * *CU* soit maximisé.

Cela se traduit par la sélection de valeurs pour *CL* et de *CU* qui sont aussi proches que possible. Ainsi, la fonction ci-dessus (*Z)* produira des fragments qui sont «équilibrés» à l'égard des transactions.

Cet algorithme pose l'inconvénient de ne pas être en mesure de partitionner un bloc en présence d'un sous bloc, a cause de la complexité des calculs.

Attributs	A5	A1	A7	A2	A8	A3	A9	A10	A4	A6
A5	75	75	50	25	25	25	25	0	0	0
A1	75	75	50	25	25	25	25	0	0	0
A7	50	50	85	60	60	15	25	0	0	0
A2	25	25	60	110	110	75	75	0	0	0
A8	25	25	60	110	110	75	75	0	0	0
A3	25	25	25	75	75	115	115	15	15	15
A9	25	25	25	75	75	115	115	15	15	15
A10	0	0	0	0	0	15	15	40	40	40
A4	0	0	0	0	0	15	15	40	40	40
A6	0	0	0	0	0	15	15	40	40	40

X

Tableau 5: Matrice d'affinité ordonnée divisée.

Le Tableau 5 représente la matrice d'affinité ordonnée divisée par le point X.

La matrice du Tableau 5 est divisée en deux parties par rapport au point X, deux fragments sont générés par cette opération : **{5, 1, 7, 2, 8, 3, 9}, {10, 4, 6}.**

L'objectif du partitionnement est de maximiser les nombre d'accès à un seul fragment, et minimiser les accès à plusieurs fragments pour la même transaction (requête).

Dans le cas où le nombre des attributs est important, il sera nécessaire de réitérer l'opération de regroupement sur les fragments obtenus, ou trouver plusieurs points de division sur la matrice.

Nous déduisons que pour une relation avec un nombre important d'attributs la génération des fragments verticaux par l'approche d'affinité, nécessite beaucoup de calculs.

C. *GRAPHE D'AFFINITÉ [6]:*

Nous présentons un autre algorithme qui permet de déduire le bon schéma de fragmentation : l'approche basée sur le graphe d'affinité débute par la construction de la matrice d'affinité, ensuite un graphe d'affinité est construit de la manière suivante :

Chaque arête du graphe est marquée par son poids, qui représente l'affinité entre ses sommets. Les sommets représentent les attributs de la relation, l'affinité entre les sommets représente le nombre de requêtes dans lequel les attributs sont requis simultanément.

La deuxième étape consiste à former un arbre couvrant linéaire. Les cycles représentent les partitions candidates.

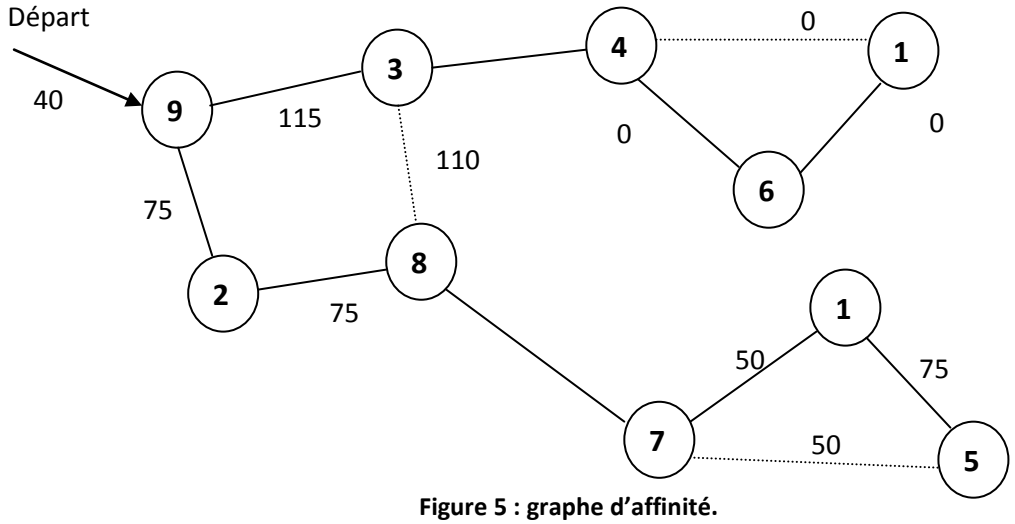


Figure 5 : graphe d'affinité.

Reprenons l'exemple du Tableau 4 : Matrice d'affinité ordonnée. La figure 5 montre le résultat de l'application de l'algorithme au graphe d'affinité. Dans la figure 5, les nœuds représentent les attributs de la relation.

On en déduit le schéma de fragmentation suivant :

1- (A1, A5, A7) 2- (A2, A3, A8, A9) 3- (A4, A6, A10)

Dans la section suivante nous allons présenter les travaux et études sur la fragmentation verticale basée sur l'approche de la matrice d'affinité.

D. APPLICATION DE LA MATRICE D'AFFINITE POUR LA FV :

Hoffer & al [34] ont été les premiers à utiliser la matrice d'affinité dans la fragmentation verticale des bases de données.

Les auteurs dans [34] ont employés l'affinité entre les attributs pour la fragmentation verticale, qui est basée sur le regroupement des attributs ayant un nombre d'accès disque rapproché.

Cette méthode se base sur le calcul de l'affinité entre les attributs, ensuite il effectue le regroupement des attributs dont l'affinité est la plus forte.

Cette méthode permet de réduire au minimum les coûts de récupération et de mise à jour, l'algorithme est assez simple à implémenter, il est constitué de trois étapes :

- Génération de la matrice d'usage,
- Génération la matrice d'affinité des attributs,
- Génération des fragments par l'application de l'algorithme de bond énergie.

Navathe et al [40] ont étendue l'approche BEA « Bond Energy Algortihm » en proposant un algorithme qui produit des fragments disjoints et non disjoints.

Cette approche améliore et réduit le nombre de fragments accédés par transaction en prenant en compte le coût de stockage.

Cette méthode consiste en deux étapes :

1. Dans la première étape les paramètres d'entrées sont rangés dans une matrice d'usage par laquelle on va construire la matrice d'affinité des attributs, utilisée dans les prochaines étapes.
2. Dans la seconde étape on prend en considération les coûts de stockage afin d'affiner les résultats.

L'étude [40] discute la fragmentation verticale dans trois contextes : bases de données centralisées stockées dans un seul niveau de mémoire. bases de données centralisées stockées dans différents niveaux de mémoire. et bases de données distribuées.

La première étape est identique pour les trois types de bases de données, pour la deuxième étape les modèles de coût seront calculés différemment pour les bases de données distribuées et les bases de données centralisées.

Le coût de transfert est pris en considération. En effet, le coût de transfert d'un bloc de données dans une base de données distribuées est plus important que celui d'une base de données centralisée.

Cependant, le coût de transfert entre deux nœuds est fixé pour toutes les transactions entre chaque paire de nœuds du réseau.

Navathe et Ra [41] étendent les travaux de Navathe [41] sur la fragmentation verticale en proposant une méthode de fragmentation verticale utilisant des techniques graphiques.

La majeure contribution de cette méthode est que tous les fragment sont générés en une seule itération avec une complexité de $O(n^2)$, ce qui est meilleur que $O(n^2 \log(n))$ qui représente la complexité de l'algorithme [40].

Autre avantage de cette méthode, on n'a pas besoin d'une fonction objective empirique pour générer les partitions.

Il est à noter que le résultat des fragments générés peut être plus important que le nombre de nœuds dans un système distribué, si le cas se présente une opération de fusionnement des fragments sera nécessaire.

Lin et Zhang [39] a souligné que la restriction d'un cycle d'affinité résultant dans une formulation est un problème NP-difficile, et donc les propriétés revendiquées dans [41] ne peuvent être garantis. De ce fait, un nouveau graphe est proposé en utilisant 2-connectivité au lieu du cycle d'affinité pour construire des fragments disjoint, ce qui est plus tard attribué à des nœuds du réseau distribué.

Lin et Zhang. [39] ont poursuivi le travail [38] en présentant un algorithme pour la construction des fragments non disjoints.

Cheng et al. [24] ont formulé la problématique de fragmentation de base de données dans le contexte distribué comme un problème du voyageur de commerce (TSP), pour lequel une approche basée sur la recherche génétique pour la fragmentation est proposé pour réaliser des performances système élevées.

Encore une fois, cette approche est basée sur la matrice d'affinité, qui calcule l'affinité entre les attributs d'une partition en fonction de la distance entre les attributs.

L'objectif de cette approche est de regrouper les attributs tels que la différence entre la distance moyenne au sein des groupes et la distance moyenne entre les groupes soit réduite au minimum.

Cependant, il n'existe aucune preuve que cette approche peut en effet minimiser le coût total des requêtes.

Après avoir survolé quelques travaux sur la fragmentation verticale par la matrice d'affinité, nous étudierons dans la section suivante les travaux et approches basés sur le modèle de coût.

II.3.B.8. APPROCHE ORIENTÉE MODÈLE DE COÛT

Le principe de cette approche se base sur l'estimation du coût de chaque transaction. Le calcul du coût est calculé par l'estimation du nombre d'accès physique aux disques et l'estimation du temps de calcul du CPU, il existe une multitude de modèle de coût.

Ces algorithmes emploient des facteurs physiques spécifiques tels que le nombre d'attributs, leur longueur et sélectivité, cardinalité de la relation etc.…

L'un des travaux important dans le domaine est celui de Yao [52] qui présente une estimation rapprochée basée sur des fondements mathématiques.

Cardenas [0] a donné l'expression estimant le nombre de blocs accédés pour une requête donnée.

Le principe de la fragmentation verticale basé sur le modèle de coût et une analyse de l'approche sont détaillés dans la section suivante.

A. PRINCIPE DES APPROCHES BASEES SUR LE MODELE DE COUT

Lorsqu'une transaction ou une requête s'exécute sur un SGBD, plusieurs tâches seront déclenchées en arrière plan par le serveur, en l'occurrence l'analyse syntaxique et lexicale de la requête, l'optimisation de la requête, l'accès physique aux données :

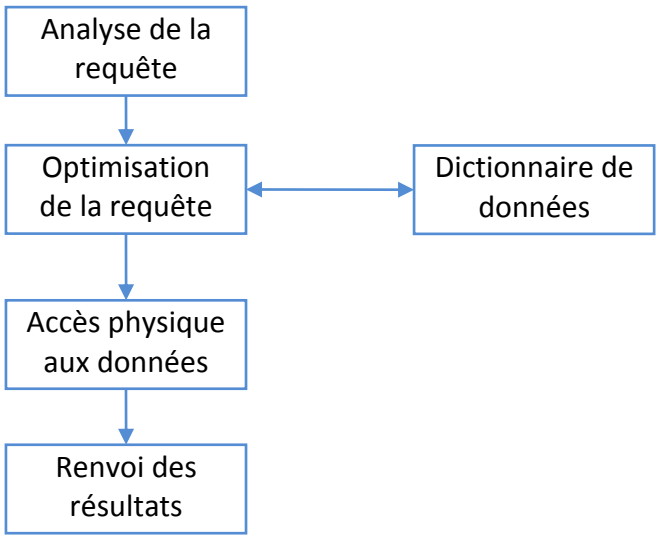


Figure 6 : exécution d'une requête sur un SGBD.

Toutes ces tâches consomment des ressources et du temps, et engendrent des calculs qui s'effectuent au niveau du processeur en arrière plan, mais l'accès physique aux fichiers de données sur disques déclenche le processus du pivotement des têtes pour effectuer la lecture sur disque afin d'atteindre l'adresse des blocs de données.

Le modèle de coût est une formule qui permet pour une requête donnée de rapprocher une estimation sur le nombre d'accès aux fichiers de données et le temps de calcul du CPU.

Comparé aux processus du pivotement des têtes de lecture, le temps que consomme le processeur (CPU) pour le calcul est négligeable. Dans la majorité des études on néglige le temps de calcul du CPU.

Une fonction de coût fondée sur cette hypothèse est déduite, cette fonction reflète le temps que consomment les ressources de la machine pour exécuter la requête.

Les facteurs sur lesquels se basent le modèle de coût sont : la quantité prévue de données qui doivent être transmises afin de répondre à la requête, et le nombre d'entrée/sortie (pivotement de la tête de lecture) pour lire les données sur disque.

Il est à noter que le modèle de coût ne représente qu'une estimation du coût d'exécution des requêtes, et ce modèle varie selon le contexte de la base (base de données distribuée, base de données centralisée).

La compréhension de l'accès physique fichiers de la base sur disque nous permettra de mieux comprendre le fondement du modèle de coût, nous vous proposons de vous reportée à la partie Annexe A consacrée à l'organisation physique des fichiers.

Nous présentons le modèle de coût de Yao [52], le modèle sur lequel se base la majorité des études orientées modèle de coût.

A.1. LITTERATURE SUR LE MODELE DE COUT

Hammer et Niamir, [33] L'objectif de ce travail est de concevoir des mécanismes qui permettent de sélectionner automatiquement une partition quasi-optimale des attributs, basé sur le modèle d'utilisation du fichier et sur les caractéristiques des données dans le fichier.

L'approche adoptée pour ce problème est basée sur l'utilisation d'un évaluateur de partition et d'une heuristique qui guide une recherche à travers le vaste espace des partitions possibles.

L'heuristique est de proposer un ensemble restreint de partitions prometteuses de soumettre à une analyse détaillée.

Pour ce faire les auteurs [33] ont développé deux heuristiques, de groupement et de regroupement, et les ont utilisées pour effectuer la fragmentation. Le groupement heuristique commence en assignant chaque attribut à un fragment différent.

Dans chaque itération, tous les groupements possibles de ces partitions sont considérés avec celle de l'amélioration maximum qui est choisie en tant que candidate pour le groupement pour la prochaine itération.

Pendant le regroupement, des attributs sont déplacés entre les fragments pour réaliser toutes les améliorations additionnelles possibles.

Cornell et Yu [26] ont discuté la fragmentation verticale pour les bases de données relationnelles et ont estimé que le temps de réponse des transactions est influencé par le nombre d'accès disque par transaction.

Considérant que l'utilité de la fragmentation verticale est de réduire au minimum le nombre d'accès disque, Cornell et Yu [26], ont proposé une méthode à deux étapes qui consiste en une étape d'analyse de requête pour estimer les paramètres, et une étape de fragmentation binaire qui peut être appliqué de manière récursive.

Avec le modèle de coût proposé, le résultat de [40] à été testé pour montrer que la solution à base d'affinité ne conduit pas toujours à une réduction du nombre d'accès disque.

Chu [25] présente deux méthodes pour résoudre le problème de fragmentation verticale: MAX et FORWARD SELECTION. Ces méthodes expriment l'idée simple que, pour minimiser les coûts, un segment doit contenir tous les attributs requis par une transaction donnée et excluent les autres attributs.

Contrairement à toutes les études antérieures, ces deux méthodes traitent une transaction comme une variable de décision et leur exécution n'est pas affectée par le nombre d'attributs. Les deux méthodes produisent des résultats excellents, sans l'aide de calculs mathématiques compliqués.

Ils prennent également en compte la complexité résultant de l'interaction entre l'attribut de partitionnement et de sélection du chemin d'accès.

Par ailleurs, les deux méthodes donnent des informations détaillées normalement non disponible à partir d'autres méthodes. Ces informations donnent un nouvel aperçu du processus de partitionnement attribut.

MAX est adapté pour les cas impliquant un petit nombre de transactions, FORWARD SELECTION pour les cas impliquant un grand nombre de transactions.

La caractéristique importante de ces deux procédures est qu'elles traitent les transactions (au lieu des attributs) en tant que variables de décision. Ainsi, le temps d'exécution de ces procédures ne dépend pas du nombre d'attributs et peut être efficacement exécuté lorsque le nombre d'attributs est très important.

La fonction objective ne compte que le nombre d'accès disque. Les deux approches [25] et [26] ne sont pas adaptées à des bases de données distribuées.

Golfarelli et al. [30] utilisent la fragmentation verticale pour partitionner des vues matérialisées définies sur un entrepôt [30]. Cette fragmentation est basée sur une charge de requêtes et un modèle de coût.

Selon les auteurs, la fragmentation verticale désigne deux opérations : d'une part la fragmentation d'une vue en plusieurs fragments et, d'autre part, l'unification en une seule vue de deux ou plusieurs vues ayant une clé commune.

L'unification respecte la règle de reconstruction d'une table fragmentée à partir de ses fragments verticaux [42] et vise à réduire la redondance des vues.

Les auteurs supposent que leur approche peut être bénéfique pour la distribution de l'entrepôt sur une architecture parallèle et proposent de combiner leur algorithme de fragmentation avec un algorithme d'allocation des fragments sur les nœuds distants.

Chakravarthy et al. [20] ont proposé un moyen de mesurer la qualité d'un schéma de fragmentation verticale.

À cet effet, ils ont créé une fonction objective, Partition Evaluator (PE), pour évaluer les différents algorithmes de fragmentation verticale en utilisant les mêmes critères.

Le PE est composé de deux éléments, coût d'accès aux attributs locaux et le coût d'accès aux attributs distants. Dans ce cas, le but est de minimiser les accès à distance et de maximiser les accès locaux sans tenir compte de la réplication.

En d'autres termes, l'objectif est que chaque site soit capable de traiter les transactions au niveau local avec un nombre minimum d'accès à des sites distants. L'idée est de parvenir à un coût minimum de traitement pour toute transaction, peu importe où il est situé.

Il est clair que le PE ne peut être utilisé dans les bases de données distribuées, car ni la taille ni les facteurs de coûts de transaction du réseau n'ont été pris en compte.

Son et Kim [48] ont supposé que le problème de fragmentation verticale doit prendre en compte le nombre de fragments finalement généré.

Ils ont discuté du problème de génération de n_fragments verticaux qui est plus souple que le problème de génération d'un partitionnement optimal.

Leur nouvelle contribution est une fonction objective qui vise à minimiser non seulement la fréquence des accès de requête pour différents fragments, mais aussi la fréquence des accès interférés entre les requêtes.

Dans la fonction objective, la localisation des données n'est pas considérée car les requêtes ne se distinguent pas entre les sites.

L'auteur ***Gorla [32]*** étend les recherches précédentes qui sont basées sur la fragmentation d'une seule relation, et effectue la fragmentation de plusieurs relations, en prenant en compte les contraintes d'intégrité référentielles, clefs étrangère et primaires.

Le temps d'accès est basé sur l'estimation du temps d'exécution, le calcul correspond à un modèle de coût complet qui prend en charge la plupart des types de transaction comprenant des mises à jour et les jointures.

L'auteur a employé un procédé heuristique pour résoudre le problème utilisant un index d'affinité à 2 attributs et un algorithme de groupement (clustering) à 2 étapes. L'application de cette méthodologie sur de petits problèmes a rapporté les solutions optimales obtenues par énumération approfondie.

Nous présentons dans la section suivante les approches basées sur les algorithmes génétiques pour la sélection d'un schéma de FV.

II- 3- c. ALGORITHME GÉNÉTIQUE:

C'est à la parution du livre intitulé « L'origine des espèces au moyen de la sélection naturelle ou la lutte pour l'existence dans la nature » que Charles Darwin en 1860 a bouleverser toute les théories sur la création et l'existence des espèces.

Bien qu'elle ne soit pas réellement basée sur des fondements scientifiques, et quelle soit très controversée par la communauté scientifique, le fondement de ses révélations, a pour hypothèse que les entités appartenant à un environnement subissant l'influence de facteurs externes peuvent changer de composition et de structure et s'adapte au fur et à mesure leurs environnement.

Les êtres d'un espace donné, évoluent en perpétuelle continuité au fil du temps. grâce à la mutation. et transmettent leurs acquis génétiquement par le procédé de la reproduction, créant ainsi de nouveaux individus plus forts. plus adaptés et plus forts des individus a plus de chance de survivre et de se reproduire, et les moins forts disparaissent ce procédé est la sélection naturelle.

Ces révélations ont inspiré plusieurs recherches, les algorithmes génétiques (AG) en sont un fruit.

Les algorithmes génétiques sont basés sur des concepts basiques de la génétique et des lois de la survie dans la nature énoncées par Darwin : croisements, mutations, sélection.

Ils représentent une variante des algorithmes évolutionniste, une variante des algorithmes métaheuristiques et des algorithmes d'optimisation stochastiques.

Historiquement les premiers algorithmes évolutionnistes et méta heuristiques ont vu le jour en 1965 à l'université technique de Berlin (Allemagne).

L'idée à été proposée par Ingo Rechenberg, quand aux algorithmes génétiques ils furent révélés grâce aux travaux de John Holland [36] sur les systèmes adaptatifs. John Holand a proposé le concept par lequel les programmes informatiques pouvaient évoluer et s'adapter en conséquence de l'environnement dont elles appartiennent.

L'objectif des algorithmes génétiques (AG) est de résoudre les problèmes combinatoires complexes PCD (ou d'en obtenir une solution rapprochée) dans un temps acceptable.

Les problèmes combinatoires difficiles (PCD) sont des problèmes n'ayant pas de solutions calculables en temps raisonnable, pour lesquels on ne connaît pas de solution exacte ou meilleur, là où souvent échouent d'autres méthodes classiques.

Les PCD sont caractérisées par ensemble de solutions possibles très vastes désigné par le terme espace de recherche, et l'exploration de tout cet espace peut se révéler très coûteuse, ce qui peut consommer beaucoup de temps et de ressources.

Il est clair que les méthodes de calcul classiques ne sont pas particulièrement bien adaptées aux espaces de recherche multimodale, un espace multimodale étant défini par une espace qui se compose de plusieurs extrema locaux.

Les extrema locaux multiples présentent des difficultés pour les méthodes classiques pour atteindre

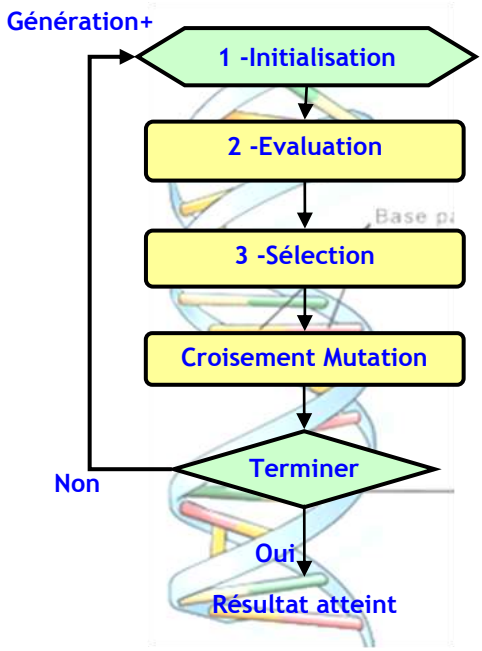


Figure 7 : Fonctionnement des AG

l'extrême global, les méthodes classiques peuvent se stabiliser sur l'un des extrema locaux, en ignorant totalement l'extrema global qui existe ailleurs dans l'espace.

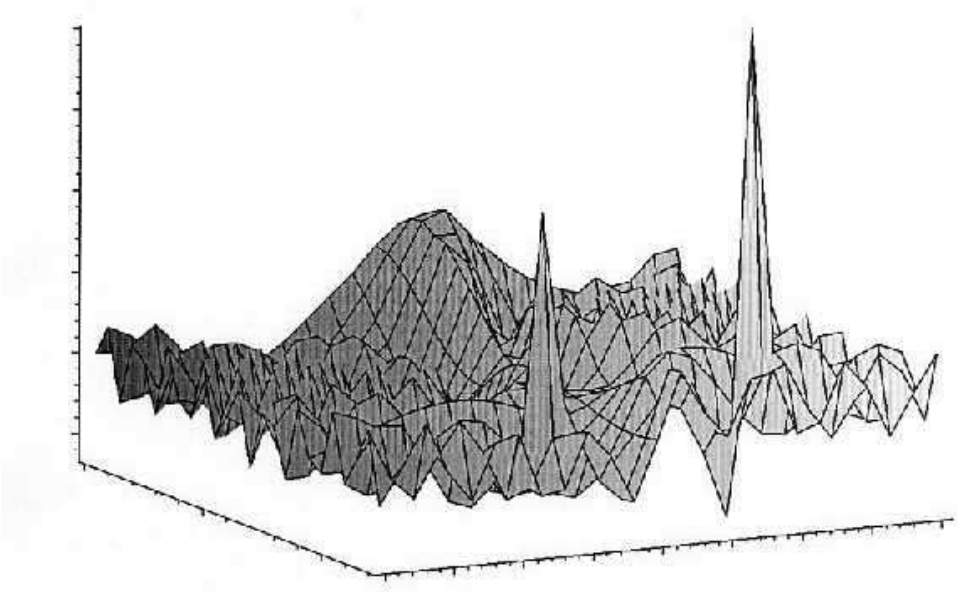


Figure 8 : Espace de recherche multimodal

La Figure ci-dessus illustre un espace de recherche très multimodal, cette figure reflète la forme d'un espace de recherche difficile contenant plusieurs extrema locaux. L'optimum sur cet espace est difficile à atteindre.

Dans la partie qui suit nous présenterons le mode de fonctionnement des algorithmes génétiques.

II.3.C.1. FONCTIONNEMENT DES ALGORITHMES GÉNÉTIQUES:

Le principe fondamental de la méthode est le suivant :

- Au début l'algorithme commence par générer aléatoirement une solution initiale (population);
- Chaque individu de la population est soumit à une évaluation par la fonction fitness afin d'en mesurer la qualité de la solution fournie par ce dernier (la force) ;
- Après l'évaluation intervient l'opération de sélection des individus pour en reproduire les nouveaux éléments de la population, selon Mendel la survie du plus adapté (la loi du plus fort) permet d'obtenir une progéniture forte.
- Ensuite l'opération de reproduction à base des individus sélectionnés au préalable en utilisant les opérations de mutation et de croisement qui nous permettent d'obtenir une nouvelle génération qui serra en principe mieux adaptés ou équivalente au fitness de la génération précédente.

En générale, à la fin de l'exécution de l'algorithme, nous obtenons une solution optimale ou proche de l'optimum pour le problème posé initialement.

II.3.C.2. OPÉRATEURS DES ALGORITHMES GÉNÉTIQUES

Dans la section suivante, une description détaillée des différents opérateurs et étapes de l'algorithme vous est présentée:

A. LE CODAGE

La mise en œuvre des AG débute par le codage ; en fait c'est le langage que manipulent les AG ; et c'est l'un des facteurs les plus importants pour le bon déroulement de l'algorithme.

Il est à rappeler que dans la biologie naturelle les scientifiques procèdent à un codage des gènes et chromosomes de l'ADN afin de les identifiés ; en suivant le même raisonnement, dans les AG.

On commencent par adopter un codage approprié au problème posé, opération dont l'objectif est de construire un modèle numérique et permet d'obtenir une projection de notre problème sur un modèle manipulable par les algorithmes génétiques.

Pour bien expliquer le codage, il faut avoir une idée générale sur les entités que manipule un AG. Les AG manipulent une population, cette population sera générée à chaque itération grâce aux procédés de la mutation et croisement et sélection.

La structure de la population est la suivante : une population est un ensemble d'individus, chaque individu est un code qui représente une solution potentielle pour notre problème, un individu est formé d'un ou plusieurs chromosomes.

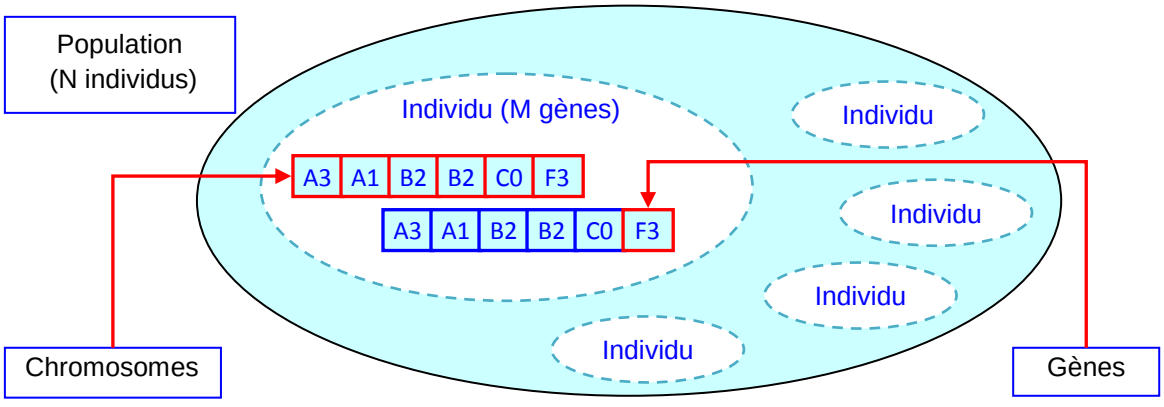


Figure 9: le codage

Un chromosome est une suite de gènes ; les gènes sont une suite de code et peuvent prendre une des différentes valeurs des allèles ; en quelque sorte un gène est une variable qui puise ses valeurs dans les allèles.

La récapitule la notion du codage sur les algorithmes génétiques.

Par exemple le problème du déplacement d'un robot dans un labyrinthe, un individu peut être codé comme suivant, {droite, gauche, haut, gauche, droite, bas}.

Il faut prendre en considération qu'un bon codage peut aider énormément dans la réussite de l'exploration efficace de l'espace de recherche, et en atteindre les optimums plus rapidement.

Il existe plusieurs variantes de codage nous en citons :

A.1. LE CODAGE BINAIRE :

C'est le type de codage le plus utilisé, chaque individu de la population est représenté par une suite de 1 et 0, 5, exemple : [0 0 1 0 1 0 1 0].

A.2. LE CODAGE DE GRAY :

Il ajoute le concept de distance entre deux individus, deux éléments voisins en termes de distance de Hamming peuvent ne pas coder nécessairement deux éléments proches dans l'espace de recherche. Cet inconvénient dans les autres types codage peut être évité en utilisant un codage de Gray.

La "distance de Hamming" est considérée comme mesure de la différence entre deux éléments de population, cette mesure compte les différences de bits du même rang de ces deux séquences.

B. LA SELECTION

L'étape de sélection est l'étape qui permet de choisir pour chaque génération les individus destinés à transmettre leurs gènes et à se reproduire, les individus qui seront soumis aux deux opérations de croisement et de mutation.

Il existe plusieurs algorithmes de sélection nous en citons les suivant :

B.1. ROUE DE LA FORTUNE (ROULETTE WHEEL).

Cet algorithme est l'une des variantes les plus intéressantes et les plus utilisées dans les applications recourant aux algorithmes génétiques.

Son principe est le suivant, un individu aura les chances de se reproduire proportionnellement par rapport à la valeur de son fitness. L'algorithme partage un

camembert sur les individus, chaque individu lui sera attribué une partie de la roue, une fois la roue est tournée, l'individu sur lequel s'arrêtera la roue sera élu.

Exemple : Soit la population composée de quatre individus avec les valeurs de leurs fitness représenté comme suit: P = {(Individu A, f=0.1), (Individu B, f=2.2), (Individu C, f=7), (Individu D, f=0.7)}.

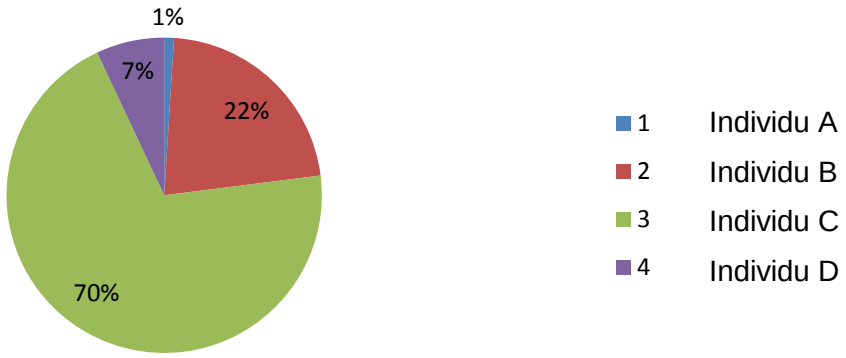


Figure 10 : Sélection par roulette de la fortune

Dans la Figure 10 on remarque que la roue est partagée entre les quatre individus qui composent la population en proportion de leur fitness, comme pour le cas de l'individu C, qui vraisemblablement occupe la plus grande partie de la roue, par ce fait, cet individu est le mieux positionné pour être sélectionner, néanmoins il faut prendre en considération que l'individu « A » a quand même une chance d'être sélectionner.

B.2. LA METHODE ELITISTE.

Le principe est le plus simple qu'il soit, choisir les meilleurs individus de la population vis à vis de leurs fitness.

Il est à noter que cette méthode n'est pas efficace, car elle peut converger vers un minimum local, et aboutir sur des solutions prématurées, et par conséquent, omettre une grande partie de l'espace de recherche.

B.3. LA SELECTION PAR TOURNOIS.

Le principe de cette méthode est le suivant : on effectue un tirage avec remise de deux individus de la population, on les compare, celui qui a le fitness le plus élevé l'emporte avec une probabilité p comprise entre 0.5 et 1.

B.4. LA SELECTION UNIVERSELLE STOCHASTIQUE.

Dans cette méthode la population est considérée comme l'image d'un segment, l'image est découpée en nombre de sous-segments autant de fois que le nombre

d'individus. Un ensemble de points équidistants sont choisis, ces points représentent les individus sélectionnés par cet algorithme.

B.5. SELECTION UNIFORME.

La sélection des individus se fait aléatoirement, uniformément et sans prise en compte du fitness. Chaque individu a donc une probabilité 1/P d'être sélectionné, où P représente le nombre total d'individus dans la population.

C. *LE CROISEMENT*

L'opération de croisement est le procédé par lequel s'effectue la production de nouveaux individus sur la base de deux autres individus, le croisement dans les algorithmes génétiques est inspiré de la reproduction dans la nature à partir d'un couple d'individu, le croisement des deux chromosomes des individus, nous permet d'en extraire deux nouvelles structures sur la base de la recombinaison des chromosomes parents, selon un schéma donné, avec une probabilité P renseignée au préalable dans l'algorithme.

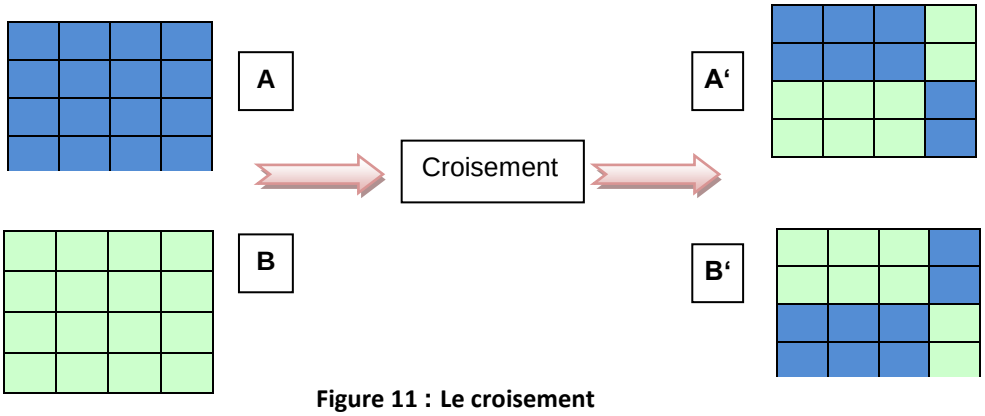


Figure 11 : Le croisement

Dans l'exemple de la Figure 11, le croisement entre deux individus permet de développer deux nouvelles combinaisons de chromosome A' et B' à partir de A et B, ces deux nouvelles combinaison sont en fait le croisement et la recombinaison des chromosomes de A et B.

D. *LA MUTATION*

Comme dans la nature, la mutation d'un chromosome est l'altération ou la modification d'un gène de la combinaison selon une probabilité donnée, (bien qu'en réalité cette altération ne peut être que néfaste dans la nature comme le cancer, effets des radiations).

Cette altération est modélisée par la modification de la valeur du gène choisi par une autre valeur prise aléatoirement parmi les valeurs possibles que peut prendre le gène.

La Figure 12 illustre brièvement l'opération de mutation, les cases colorées sont les cases mutées.

Figure 12 : La mutation

Utilité des opérateurs de croisement et de mutation

Ces deux opérateurs permettent aux algorithmes d'explorer et d'atteindre les optimums locaux et généraux dans l'espace de recherche.

Le croisement permet la transmission des caractères acquis d'un individu vers les descendants, ainsi les individus développés seront soit équivalents ou soit meilleurs que leurs ascendants, concrètement atteindre le minimum local.

La mutation nous offre la possibilité de trouver des individus qui appartiennent à une autre partie de l'espace de recherche et ainsi de ne pas tomber dans le piège de se contenté du minimum local, alors qu'il pourrait exister une solution meilleur que celle du minimum local.

Grâce à l'opérateur du croisement nous pouvons atteindre l'optimum pour un extrema local, mais cette solution ne représente guère la meilleure solution. Quand à l'application de l'opérateur de mutation, il nous permet de découvrir un nouvel espace de solution qui peut être meilleur et qui n'es pas évident à atteindre.

Les algorithmes génétiques sont assez souples et flexibles pour s'adapter à n'importe quel problème, et présentent de très bonnes aptitudes à résoudre les problèmes combinatoires difficiles.

Dans la section suivante nous détaillerons les principaux travaux réalisés sur la fragmentation verticale utilisant les algorithmes génétiques.

II- 3- D. APPROCHE BASEE SUR LES ALGORITHMES GENETIQUES

Les algorithmes génétiques sont des algorithmes métaheuristiques, qui permettent de résoudre les problèmes combinatoires. De nombreux travaux se sont basés sur les AG pour optimiser la conception physique par la technique de FV.

Puisque dans notre étude nous avons implémenté une approche basée sur les AG pour la fragmentation verticale, nous avons consacré toute une section pour détailler les études se référant de cette dernière.

Il est à considérer que la fonction objective ou fitness décrite dans toutes ces études, représente l'estimation du coût de la charge de travail appliquée sur la base de données.

Cheng et al. [24] ont proposé une recherche basée sur l'algorithme génétique pour le partitionnement de données pour atteindre de hautes performances de récupération dans les bases de données réparties, en considérant que le problème de la fragmentation verticale est formulé comme le problème du voyageur de commerce.

Song et Gorla [31] ont employé des algorithmes génétiques pour obtenir une solution simultanée pour générer les fragments verticaux et repérer le chemin d'accès aux fragments. Ils ont également utilisé le nombre d'accès de disque comme critère d'évaluation de la fragmentation.

Concernant la conception de base de données orientée objet, Gorla a employé l'algorithme génétique pour déterminer les variables d'instance qui devraient être stockées dans chaque sous-classe/classe dans une hiérarchie de sous-classes, de sorte que tous les coûts des opérations de base de données soient réduits au minimum.

Angel et Zevala [12] ; Dans le contexte des bases de données distribuées un nouveau problème est souvent traité en complément du problème de fragmentation. Il s'agit du problème d'allocation des fragments sur les nœuds du réseau, ça revient à allouer les fragments dans les différents sites du réseau de telle sorte à ce que l'accès à l'information soit le plus rapide que possible.

Dans ce contexte, les auteurs [12] proposent l'adaptation de l'algorithme génétique pour résoudre à la fois le problème de la fragmentation et de l'allocation des fragments sur les nœuds du réseau.

Un vecteur de nombre entier de longueur égale aux nombre d'attribut de la relation sera employé pour représenter les solutions possibles au problème.

Ce vecteur prendra les nombres entiers entre 1 et N, où N dénote le nombre d'emplacement dans le réseau.

Exemple :

Soit une base de données distribuée sur un site composé de 5 sites, et soit une relation R appartenant à la base composée de 10 attributs :

R (A, B, C, D, E, F, G, H I, J)

Selon l'approche d'Angel [12], le vecteur suivant est un codage d'une solution potentielle pour la fragmentation et l'allocation de la relation R:

(5, 2, 4, 3, 1, 3, 3, 5, 2, 4)

Ce vecteur sera interprété comme suit :

- La position dans le vecteur indique le numéro d'attribut,
- Les composants du vecteur indiquent les emplacements sur site.

Par conséquent, le vecteur donné en solution sera interprété comme suit :

Sites	Allocation des attributs		
1	E		
2	B	I	
3	D	F	G
4	C	J	
5	A	H	

Tableau 6 : Allocations des attributs par site.

Cette approche permet de donner une solution qui résout à la fois la fragmentation de la relation et l'allocation des fragments sur les sites du réseau.

Nous allons aborder dans la section suivante le sujet de la fragmentation verticale basée sur la fouille de données.

II- 3- E. FOUILLE DE DONNÉES (DATAMINING)

Considérée comme une discipline associée au domaine de l'aide à la décision, la fouille de donnée ou « datamining » est un procédé qui nous permet d'extraire les connaissances et les informations valides et exploitables à partir de sources de données.

Le terme Datamining à été lancé par le MIT « Massachusetts Institute of Technology » vers les années 1980.

L'idée principale est de *valoriser la mémoire et les données d'une entreprise ou d'une entité en les explorant au moyen de techniques de mathématiques appliquées*. Le MIT annonçait que cette technologie de l'information préparait la prochaine révolution du siècle.

A la différence des méthodes d'analyse de données ou de statistiques, la fouille de données permet l'extraction des connaissances sans établir une hypothèse au préalable ; en fait les connaissances extraites émergent.

La fouille de données se propose donc de transformer de grands volumes données en informations ou connaissances, les données analysées proviennent souvent de

sources diverses comme les entrepôts de données, ou autres comme Internet, ou source de temps réel (GAB, analyse des achats par téléphone ….).

Il est à noter qu'une réécriture des données s'impose dans le cas où la source de données n'est pas forcement une base de données, cela suppose s'être doté des outils nécessaires.

Souvent utilisé dans l'analyse du comportement des personnes : consommateur, clients et usager, la fouille de données peut aussi s'avérer utile dans la prédiction des comportements des gens ainsi que la détection des fraudes et la prédiction des zones et heures de pointes dans le transport.

II.3.E.1. PRINCIPE DE LA FOUILLE DE DONNEES

Plus qu'une théorie normalisée, la fouille de données est un processus d'extraction de connaissances métiers comportant les étapes principales suivantes :

- *Formaliser* un problème que l'organisation cherche à résoudre en termes de données
- *Accéder* aux données appropriées quelles qu'elles soient
- *Préparer* ces données en vue des traitements et utilisations futurs
- *Modéliser* les données en leur appliquant des algorithmes d'analyse
- *Évaluer* et valider les connaissances ainsi extraites des analyses
- *Déployer* les analyses dans l'entreprise pour une utilisation effective

Ce processus est cyclique et permanent; la fouille de données rend dès lors plus compréhensible, "visible", l'activité de l'organisation, et permet de rationaliser le stockage de l'information et des données.

La fouille de données ne consiste pas en une succession d'études ad hoc mais a pour objectif de capitaliser des connaissances acquises sous forme de connaissances explicites.

Elle conduit donc à mieux structurer les contenus nécessaires à l'ingénierie des connaissances.

Nous pouvons distinguer trois grandes familles dans les algorithmes de fouille de données:

A. METHODES NON SUPERVISES :

Cette catégorie ne considère en aucun cas qu'un ensemble de données ou de variables soit plus important que les autres.

Parmi les techniques les plus connues nous citons : réseau de neurones, k-means, règles d'association, etc.…

B. METHODES SUPERVISEES :

Contrairement aux méthodes non supervisées, cette catégorie est adaptée plus particulièrement à un groupe de variables définis comme étant l'objectif de l'analyse.

Les méthodes supervisées ont donc pour objectif d'expliquer un comportement précis comme l'augmentation des ventes d'un produit par rapport à un autre, ou les raisons de la réussite d'une technique de marketing par rapport à une autre.

Les techniques suivantes appartiennent à cette catégorie : techniques statistiques de régressions linéaires et non linéaires, techniques à base d'arbres de décision, raisonnement par cas ….

C. LES METHODES DE REDUCTION DES DONNEES :

Permettent de compresser et de réduire et de filtrer un ensemble de données volumineux, et d'en extraire les informations les plus pertinentes et significatives.

Elles sont souvent utilisées comme approches complémentaires des techniques supervisées ou non supervisées et dans le domaine de la statistique. Parmi ces techniques nous citons : techniques d'analyse factorielle et technique de positionnement.

Dans notre étude nous nous intéressons plus aux techniques de génération des règles d'association, les règles d'association sont traditionnellement liées au secteur de la distribution car leur principale application est « l'analyse du panier de la ménagère » qui consiste en la recherche d'associations entre produits sur les tickets de caisse.

Le but de la méthode est l'étude de ce que les clients achètent pour obtenir des informations sur qui sont les clients et pourquoi ils font certains achats. La méthode recherche **quels produits tendent à être achetés ensembles.**

La méthode peut être appliquée à tout secteur d'activité pour lequel il est intéressant de rechercher des groupements potentiels de produits ou de services : services bancaires, services de télécommunications, par exemple.

Elle peut être également utilisée dans le secteur médical pour la recherche de complications dues à des associations de médicaments ou à la recherche de fraudes en recherchant des associations inhabituelles.

Un attrait principal de la méthode est la clarté des résultats générés. En effet, le résultat de la méthode est un ensemble **de règles d'association.**

Des exemples de règles d'association sont :

- si un client achète des plantes alors il achète du terreau,
- si un client achète du poisson et du citron alors il achète un jus.
- si un client achète une télévision, il achètera un magnétoscope dans un an.

Ces règles sont intuitivement faciles à interpréter car elles montrent comment des produits ou des services se situent les uns par rapport aux autres. Ces règles sont particulièrement utiles en marketing.

Les règles d'association produites par la méthode de réduction des données peuvent être facilement utilisées dans le système d'information de l'entreprise.

Cependant, il faut noter que la méthode, si elle peut produire des règles intéressantes, peut aussi produire des règles triviales (déjà bien connues des intervenants du domaine) ou inutiles (provenant de particularités de l'ensemble d'apprentissage).

La recherche de règles d'association est une méthode non supervisée car on ne dispose en entrée que de la description des achats.

Dans le paragraphe suivant nous allons définir la terminologie exploitée dans le domaine.

II.3.E.2. DÉFINITIONS

- ***Les items*** décrivent l'ensemble des articles ou d'éléments sujet de l'étude, par exemple dans l'application du panier ménagère la liste décrivant la liste des items corresponds à l'ensemble des articles disponibles dans le magasin (pain, lait, café, friandises). Il est à noter que les items ne sont pas dupliqués dans le même ensemble.
- ***Les règles d'association,*** est une association entre plusieurs articles qui permet de découvrir à partir d'un ensemble de transactions, un ensemble de règles qui exprime une possibilité d'association entre différents items.
- ***Une transaction*** est une succession d'items exprimée selon un ordre donné ; de même, l'ensemble de transactions contient des transactions de longueurs différentes.

Un exemple classique sur l'application des règles d'association est le panier de ménagère qui décrit un ensemble d'achats effectués dans un supermarché ; les règles d'association permettent de découvrir des régularités dans l'ensemble de transactions comme par exemple : Si champoing alors gèle douche, etc.

Ces règles permettent par exemple au gérant de proposer des bons de réductions significatifs sur les achats futurs des clients.

Une *règle d'association* est une règle de la forme : Si **condition** alors **résultat**. Dans la pratique, on se limite, en général, à des règles où la **condition est une conjonction d'apparition d'articles et le résultat est constitué d'un seul article.**

Par exemple, une règle à trois articles sera de la forme :

Si X et Y alors Z : (X et Y) → Z ; règle dont la sémantique peu être énoncée : Si les articles X et Y apparaissent simultanément dans un achat alors l'article Z apparaît.

Pour choisir une règle d'association, il nous faut définir les quantités numériques qui vont servir à valider l'intérêt d'une telle règle.

Pour faire, les règles d'associations sont vérifiées sur la base de la mesure des deux valeurs du Support et de la Confiance définies dans la section suivante.

A.1. SUPPORT ET CONFIDENCE D'UNE REGLE :

Le **support d'une règle** indique le pourcentage d'enregistrements qui vérifient la règle, il indique le nombre de fois, en termes de pourcentage, où la loi est applicable.

Considérons l'ensemble des paniers d'achats T ; soit R un groupe de produits et U l'ensemble des paniers d'achat contenant le sous-groupe R, on a :

$$Support(R) = \left(\frac{|U|}{|T|} \right) * 100\%$$

Avec $|U|$ et $|T|$ le nombre d'éléments de U et de T.

Un ensemble d'items est dit fréquent s'il correspond à un motif fréquent dans la base de transactions. Un seuil minimal de support min_Supp est fixé à partir duquel un ensemble d'items est dit fréquent.

Exemple

Si S = {café, lait}, si un magasin reçoit 210 consommateurs dans la journée et que 45 d'entre eux achètent du café et du lait, alors :

Support ({café, lait}) = (45 / 210) * 100% = 21.4 %.

Support (S1 -> S2) = support (S1 & S2).

Reprenons notre exemple : la loi « café -> lait » n'est applicable que si le consommateur qui a acheté du lait, a acheté du café, c'est-à-dire dans 24,8% des cas.

Plus le support est important, et plus la fréquence avec laquelle on a pu vérifier que la loi était vraie ou fausse l'est aussi.

La confiance est le rapport entre le nombre de transactions où tous les items figurant dans la règle apparaissent, et le nombre de transactions où les items de la partie condition apparaissent.

Soit la règle d'association R = S1 -> S2, avec S1 et S2 des groupes de produits.

$$confiance = \frac{Support(S1, S2)}{Support(S1)} * 100\%$$

Exemple

Si S1 = {café} et S2 = {lait}, et que sur les 210 consommateurs 52 achètent du café, 108 achètent du lait, parmi lesquels 45 achètent ces deux produits simultanément:

*Confiance(R) = (support ({café, lait}) / support ({café}))*100%*

Confiance(R) = 21,4% / 24,8% = 86,3%.

Cela signifie que si le consommateur achète du café, il achète également du lait dans 86,3% des cas.

A noter que la loi S1 -> S2 est différente de la loi S2 -> S1. Ainsi, dans notre exemple, si le consommateur achète du lait, il n'achète du café que dans 41,6% des cas.

Une règle est dite solide si son support est supérieur ou égal à un support minimal fixé et si sa confiance est supérieure à une confiance minimale donnée.

Enfin nous présenterons les principaux travaux utilisant le principe de la fouille de données afin de réaliser une fragmentation verticale.

II- 3- F. LITTERATURE SUR LA FOUILLE DE DONNEES (DATAMINING)

La fouille de données est une technique qui permet d'extraire des connaissances et des règles à partir d'un important volume de données et d'informations. La fouille de données est utilisée dans plusieurs domaines dont nous citons les statistiques, le commerce et marketing, les systèmes experts, les applications industriels, profilage des clients et consommateurs.

Le travail de Gorla et Pang [31] s'est distingué par son originalité, les détails fournis, et les bons résultats obtenus.

Ce travail fera l'objet d'une analyse et étude détaillée dans le prochain chapitre, avec une implémentation logicielle et plusieurs tests.

Gorla & Pang Wing [31] proposent une nouvelle approche pour fragmenter verticalement les relations d'une base de données en utilisant des règles d'association, une technique de datamining qui consiste à adapter l'algorithme Apriori sur le problème de la fragmentation.

La méthode se résume en trois étapes : estimation de la charge de travail, calcul des formules de coût pour l'exécution de transaction dans un environnement fragmenté, et un algorithme pour générer les fragments.

L'approche [31] a été expérimentée sur deux bases de données réelles, et les résultats obtenus sont assez concluants.

II -4. BILAN ET DISCUSSION:

Aboutir à un schéma de fragmentation verticale optimale sur une base de données est une opération complexe qui nécessite beaucoup de calculs, ceci se fait ressentir plus sur les relations composées d'un nombre important d'attributs.

Les travaux que nous avons cités proposent des méthodes différentes dans des contextes différents pour aboutir à un schéma acceptable ou optimal de fragmentation verticale, nous avons synthétisé les différentes approches et études dans le Tableau 7.

L'application des approches et algorithmes d'optimisation afin de déduire le meilleur schéma de fragmentation verticale devient indispensable. Ceci a donné naissance aux outils d'assistance des gestionnaires et administrateurs de bases de données pour la conception physique.

Ces outils offrent la possibilité d'optimiser la conception physique d'une base de données sur la base de plusieurs techniques, avec la possibilité de simuler l'impact de l'application d'une conception sur les performances de la base de données.

Travaux	Matrice d'affinité	Modèle de coût	Algorithmes et contribution	Contexte	
				BD centralisée	BD distribuée
Hoffer & al [34]	X		Bond énergie	X	
Navathe et al [40]	X		Bond énergie, allocation, réplication	X	X
Navathe et Ra [41]	X		Graphe d'affinité	X	X
Lin et Zhang [39]	X		Graphe d'affinité amélioré	X	X
Cheng et al. [24]	X		Voyageur de commerce, Algorithme génétique	X	X
Hammer et Niamir [33]		X	Groupement regroupement récursive	X	
Cornell et Yu [26]		X	Fragmentation binaire récursive	X	
Chu [25]		X	MAX et FORWARD SELECTION	X	
Golfarelli et al. [30]		X	Fragmentation, unification	X	X
Chakravarthy et al[20]		X	Partition Evaluator, maximiser les accès locaux	X	X
Son et Kim [48]		X	Nombre de fragments peut être pré-renseigné.	X	
Gorla [32]	X	X	Prise en compte intégrité référentielle, plusieurs relations.	X	
Song et Gorla [31]		X	Fragmentation et chemin d'accès aux fragments, Algorithme génétique	X	
Angel et Zevala [12]		X	Fragmentation et allocation, Algorithme génétique	X	X
Gorla & Pang Wing [31]		X	Datamining : Apriori	X	

Tableau 7 : sommaire des travaux sur la fragmentation verticale

II-5. CONCLUSION:

Les techniques de conception physique nous permettent d'optimiser les performances des bases de données, et de remédier au problème de la latence du système.

Aujourd'hui, Tous les éditeurs et concepteurs de SGBD ont pris conscience du fait que la gestion de la conception physique et l'autorégulation des ressources de la base doivent être incluse et prise en charge par le système.

Nous pouvons bien comprendre l'ampleur de l'assistance des administrateurs de base de données dans la conception physique. Ceci n'est plus devenu qu'une simple option pour les administrateurs et gestionnaires de bases de données, mais une exigence, surtout pour les SGBD gérant de grands volumes de données, pour le compte d'utilisateurs et de clients de plus en plus impatient et exigeant.

L'Etude de l'impact de la fragmentation verticale sur les performances des bases de données, et l'analyse de la littérature sur les travaux et recherches réalisés, nous permettra d'implémenter et de réaliser des algorithmes permettant de sélectionner un schéma de fragmentation verticale optimisé.

Sur le marché des SGBD, la technique de la fragmentation verticale ne connait pas une grande popularité, surtout pour les SGBD commerciaux, d'ailleurs, nous pouvons remarquer que rare sont les SGBD qui ont mis au point un dispositif pour intégrer et supporter la fragmentation verticale, et aucun des outils d'assistance n'intègre des recommandations sur la fragmentation verticale.

Dans le prochain chapitre nous vous proposons des tests et simulations des différents algorithmes, ce programme offre des recommandations sur la fragmentation verticale, se basant sur les techniques des algorithmes génétiques, le datamining, et l'approche Affiner que fait l'objet de notre contribution.

Chapitre III : Optimisation de la fragmentation verticale.

III -1. INTRODUCTION:

Après avoir fait un survol sur le contexte de notre travail, et eu un aperçu général sur les travaux phares réalisés sur la fragmentation verticale, nous abordons dans cette partie les approches et contributions de notre travail.

Ce chapitre est divisé en trois parties.

La première partie concernera l'algorithme Apriori, une variante optimisée basée sur les algorithmes Datamining ou fouille de données, suivi d'une illustration sur le fonctionnement d'Apriori. Enfin, nous clôturons cette partie par la manière avec laquelle on va utiliser Apriori pour procéder à la fragmentation verticale.

La deuxième partie traitera le sujet des algorithmes génétiques, nous présenterons des définitions et analyses sur les notions théoriques de l'algorithme, le mode de fonctionnement des algorithmes génétiques, suivi par le procédé d'utilisation des algorithmes génétiques pour la fragmentation verticale.

La troisième partie traite l'approche "Affiner" que nous avons proposée, avec une explication détaillée de cette approche, suivie d'une illustration de l'approche pour la fragmentation verticale. Enfin, nous clôturons ce chapitre par une conclusion.

Une variante des plus intéressante des algorithmes d'extractions des règles d'associations et qui à fait l'objet de notre étude, c'est l'algorithme dit Apriori, développé par Agrawal et al [8], ce qui nous a intéressé dans cette approche c'est la complexité réduite des opérations pour aboutir aux même résultats que les autres algorithmes.

III -2. ALGORITHME APRIORI

Apriori est le fruit des travaux réalisés par Agrawal & al [8], c'est un algorithme permettant de repérer les lois d'association. L'induction des lois d'association, également appelée analyse des paniers d'achats, est une méthode qui a pour objectif de trouver des régularités dans le comportement des consommateurs.

Ces régularités se caractérisent par des groupes de produits qui sont régulièrement achetés. Ainsi, la présence d'un ou plusieurs produits dans un panier d'achat peut entraîner la présence d'un ou plusieurs autres produits.

L'Algorithme Apriori est basé sur le principe que tout ensemble d'un sous-ensemble non fréquent est non fréquent ce qui implique une limitation de l'espace de recherche.

Déroulement de l'algorithme A Priori:

- Génération des ensembles d'items
- Calcul des fréquences des ensembles d'items
- Obtention des ensembles d'items fréquents : avec un support minimum: les ensembles d'items fréquents
- Génération à partir de l'ensemble des items fréquents des règles d'associations solides : ayant une confiance suffisante.

Code de l'algorithme : Nous adoptons dans la suite la terminologie suivante :

- Ck est l'ensemble des items de taille k,
- Lk est l'ensemble d'items fréquents de longueur k.

L'algorithme Apriori est énoncé dans l'algorithme 1 :

```
Calculer L1 (l'ensemble des items fréquents)
K ← 2
Tan que  L(k-1) ≠ ∅ faire
Ck ←apriori-gen (L(k-1))
Pour chaque transaction t
Faire
    Ct ←sous-ensemble (Ck, t)
    Pour tout les candidats c ∈ Ct :
    Faire
                Count[c] ++;
    Fin faire
Fin faire
Lk ← {c∈ Ck| count[c]≥ minSup};
K++;
Return ⋃k Lk;
```

Algorithme 1 : Algorithme Apriori. [23]

Où count [c] représente la fréquence de l'élément « c » dans les transactions ou le panier.

L'algorithme Apriori permet de découvrir les sous-ensembles d'items fréquents en partant de ceux dont la longueur est 1 et en augmentant la longueur au fur et à mesure.

Cet algorithme est fondé sur la propriété des sous-ensembles d'items fréquents. Il fait appel à deux algorithmes :

Apriori-gen : L'algorithme Apriori-gen est constitué de deux phases la première phase nommé Joindre trouve tous les candidats possibles de longueur k à partir de l'ensemble Lk−1,

La deuxième phase efface Ck les éléments qui ne vérifient pas la propriété des sous-ensembles fréquents.

Sous-ensemble : L'algorithme sous-ensemble calcule le sous ensemble Ct ⊆ Ck qui correspond aux sous-ensembles présents dans les transactions contenues dans D.

Par exemple si L3 = {{123}, {124}, {134}, {135}, {234}},

La phase joindre donne comme résultat C4 = {{1234}, {1345}}, ensuite la phase effacer donne le résultat :

C4 = {1234} car l'élément {145} n'est pas dans L3 et donc {1345} est effacé.

Exemple d'application : avec minSup =2

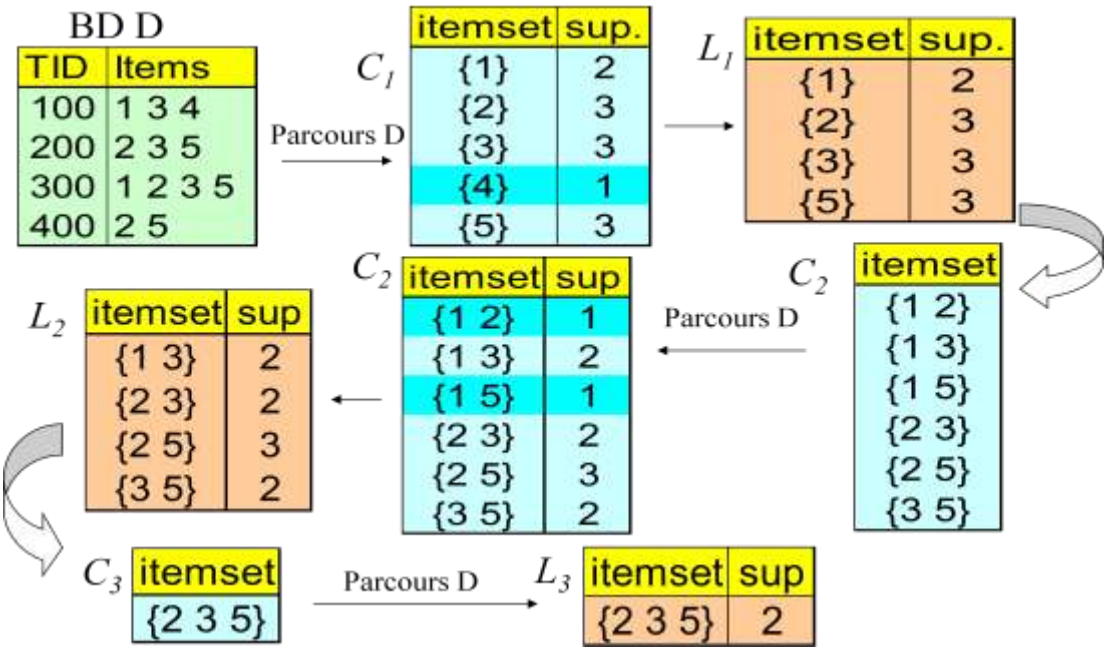


Figure 13 : Illustration de l'exécution d'Apriori

L'algorithme Apriori nous permet d'extraire les règles d'associations entre les différents items, l'intérêt majeur que nous apporte l'algorithme c'est la complexité réduite de ce dernier, il apporte des solutions en un temps efficace.

Nous avons vu dans le chapitre I que la fragmentation verticale sur les entrepôts de données est un problème difficile, et les nombres de propositions possibles est en

nombre de Bell, la complexité réduite de l'algorithme nous à motiver à appliquer Apriori pour apporter une réponse au problème de la fragmentation verticale des bases de données.

III- 2- A. UTILISATION D'APRIORI POUR LA FRAGMENTATION VERTICALE [14]

L'utilisation d'Apriori pour générer des fragments verticaux suit le raisonnement suivant.

Soit A et B deux attributs dans une relation. Si la valeur de la confiance de la règle A→B est plus grande que le minimum prédéfini (min_conf), alors l'association entre les attributs A et B est assez forte pour qu'ils puissent être groupés dans la même partition.

Puisque l'ordre des attributs dans la même partition n'est pas significatif, les valeurs de confiances pour A→B et pour B→A sont calculées et la plus faible valeur sera choisie pour représenter l'association entre les attributs A et B.

Ainsi, la confiance de (A, B) = freq (A, B)/max (freq(A), freq(B)).

La valeur de confiance est plus haute dans deux situations :

– Quand il existe peu de transactions requérant l'attribut A seulement ou B seulement,

– Quand les transactions requièrent à la fois A et B. Ainsi, la confiance fournit davantage de justification, en ajout pour le support minimum (minSup) du stockage des attributs A et B dans la même partition.

L'algorithme proposé regroupe les étapes suivantes :

– découvrir les grands ensembles d'item,

– filtrer les ensembles d'item,

– produire les partitions de données, et choisir le meilleur schéma de fragmentation.

III.2.A.1. DECOUVERTE DES GRANDS ENSEMBLES D'ITEM (ITEMSET)

Les grands ensembles d'item sont les combinaisons d'attributs qui ont un support au-dessus du support minimum prédéfini minSup.

Pour une transaction de récupération, l'ensemble des attributs dans les clauses select est considéré ; pour une transaction d'INSERT/DELETE, tous les attributs dans la relation sont employés.

De grands ensembles d'items (itemsets) peuvent être découverts en adaptant l'algorithme « Apriori » (Agarwal et Srikant, 1994 [98]) et en ajoutant des fréquences de transaction plutôt que de compter les transactions.

Le support des candidats en C_k est calculé par la somme de la fréquence des requêtes qui contiennent l'ensemble des candidats. On dérive l'ensemble des grands items L_K de telle façon qu'il atteint le niveau minimum prédéfini de support (minSup).

Ainsi, les entrées pour cette étape sont un ensemble de transactions de base de données (des récupérations et des mises à jour) $T_1 \ldots T_m$, les fréquences de transaction $Freq_1 \ldots Freq_m$, et le niveau de support prédéterminé minSup.

Les sorties de ce module représentent les grands ensembles d'items $L_1 \ldots L_K$.

III.2.A.2. FILTRAGE DES GRANDS ENSEMBLES D'ITEMS

L'ensemble d'Items dont la confiance est plus petite que la confiance minimum prédéterminée (min_conf) est rejeté.

Pour chaque ensemble d'Items des grands itemsets (ensembles d'items) L_K, toutes les règles d'association possibles sont produites et les niveaux de confiance correspondants sont calculés.

Soit les règles d'association LHS→RHS, où LHS, RHS représentent des ensembles d'items, le niveau de confiance est calculé comme la fréquence totale des transactions qui requièrent à la fois LHS et RHS, divisés par la fréquence totale des transactions requérant seulement LHS.

Cette étape prend en entrées les grands Itemsets ($L_1 \ldots L_K$) et le niveau de confiance minimum prédéterminé (min_conf) et en sortie génère de grands Itemsets filtrés ($L'_1 \ldots L'_k$).

III.2.A.3. DÉRIVATION DES PARTITIONS VERTICALES

Après avoir déduit les grands itemsets filtrés $L'_1 \ldots L'_k$, nous commençons à partir des k-itemsets pour déterminer les partitions.

Créer un schéma de fragmentation en sélectionnant un itemset à partir de L'_k, et puis en sélectionnant d'autres itemset de $L'_{k-1} \ldots L'_1$ dans cet ordre de telle sorte qu'elles soient disjointes.

Réitérer la même opération de sélection des itemset, jusqu'à ce que les derniers items soient sélectionnés à partir L'_1, ceci produit un schéma de fragmentation.

Répéter le processus ci-dessus jusqu'à ce que tous les schémas de fragmentation possibles soient dérivés. Ainsi, ce module a pour entrée les grands Itemsets filtré $L'_1 \ldots L'_k$ et retourne en sortie les schémas de partitions P1…Ps.

Il convient noter de que chaque schéma de fragmentation peut être constitué de plusieurs partitions.

III.2.A.4. DÉDUCTION DU SCHÉMA FINAL

Les entrées pour ce module est l'ensemble des schémas de fragmentations P1…Ps et l'ensemble des transactions $T_1 \ldots T_M$ et en sortie le schéma résultant P_{res}.

Une fois la dernière étape de l'algorithme est exécutée, nous obtenons en résultat au moins un schéma de fragmentation proposée par la méthode.

En calculant les coûts d'exécutions pour les schémas de fragmentations P (par le calcule de la sommes des coûts pour chaque transaction en utilisant le modèle de coût de), on peut déduire le schéma de fragmentation P_{res} avec le meilleur coût d'exécutions.

III.2.A.5. ILLUSTRATION:

Dans ce qui suit nous allons illustrer le fonctionnement de l'algorithme Apriori pour la fragmentation verticale.

Considérons l'ensemble de transactions T1… T5 et leurs fréquences opérant sur une relation avec six attributs R (A, B, C, D, E, F).

	Transactions	Attributs	Fréquences
T1	SELECT A, B, E FROM R ;	A, B, E	1
T2	SELECT B, E FROM R ;	B, E	3
T3	SELECT A, D, F FROM R ;	A, D, F	3
T4	INSERT FROM R ;	A, B, C, D, E, F	2
T5	DELETE FROM R;	A, B, C, D, E, F	1

Tableau 8 : ensembles de transactions

Soit le support minimum de 40% (c.-à-d., le min_supp est 4), les grands itemsets LK sont produit en adaptant l'algorithme Apriori.

- ***Première étape : dérivation des 1-itemsets :***

Calcul du support : la somme de la fréquence d'apparition de l'attribut de la requête
Elimination des attributs qui ne présente pas un support ≥ 4

D'abord, le candidat 1-itemsets C1 est dérivé en ajoutant les fréquences des transactions dans lesquelles un attribut est employé.

Pour les transactions d'insertions et de suppression, on suppose que tous les attributs sont employés puisque la rangée entière doit être accédée.

Ensemble d'item	Support
A	7
B	7
D	6
E	7
F	6

Tableau 9 : dérivation des 1- fréquent itemsets

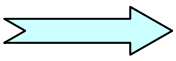
Par exemple, l'attribut A est employé dans les transactions T1, T3, T4, et T5 avec les fréquences 1, 3, 2, et 1 respectivement. Ainsi le support pour l'attribut {A} = 7. L'opération est reproduite pour chaque attribut.

Le candidat 1-itemset L1 est dérivé du candidat 1-itemset C1 en éliminant les itemsets dont le support est inférieur à 4.

- ***Deuxième étape : dérivation des 2-itemsets***

  Création de l'association d'ensemble de deux attributs,
  Calcul du support par la somme de la fréquence de l'apparition des ensembles dans les requêtes.
  Élimination des ensembles qui ne présentent pas un support ≥ 4

Ensemble d'item	Support
A, B	4
A, D	6
A, E	4
A, F	6
B, D	3
B, E	7
B, F	3
D, E	3
D, F	6
E, F	3



Ensemble d'item	Support
A, B	4
A, D	6
A, E	4
A, F	6
B, E	7
D, F	6

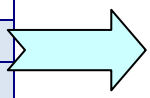
Tableau 10 : dérivation des 2- fréquent itemsets

Le candidat 2-itemset C2 est dérivé en concaténant L1 avec L1 et en cherchant les niveaux de support en balayant l'ensemble de transaction, il faut noter que les doublons sont éliminés.

- *Troisième étape : dérivation des 3-itemsets*

Création des groupes de trois attributs,
Calcul du support des ensembles d'attributs,
Elimination des ensembles qui ne satisfont pas la condition support ≥ 4.

Ensemble d'item	Support
A, B, D	3
A, B, E	4
A, B, F	3
A, D, E	3
A, D, F	6
A, E, F	3



Ensemble d'item	Support
A, B, E	4
A, D, F	6

Tableau 11 : dérivation des 3- fréquent itemsets

Par exemple, à partir de L1, en concaténant {A} et {B}, nous trouvons {A B} en tant qu'un des 2-itemsets C2. Le niveau de support pour {A B} est obtenu en ajoutant les fréquences des transactions dans lesquelles A et B sont employés.

Ces transactions sont T1, T4, et T5 avec des fréquences de 1, 2, et 1, respectivement, ayant pour résultat le niveau de support 4 pour {AB}.

Le processus ci-dessus est répété jusqu'à ce que tous les grands itemsets soient dérivés.

A. *FILTRAGE AVEC DES NIVEAUX DE CONFIANCE PREDETERMINES :*

Chaque ensemble d'items dans les grands ensembles d'items est maintenu seulement si l'ensemble d'items à un niveau de confiance supérieur ou égale à min_conf.

Par exemple, considérons l'association {A, D, F}. La valeur de confiance pour A→DF est calculée par le nombre de transactions dans lesquelles tous les attributs A, D, et F sont sollicités (c.-à-d., 6) divisé par le nombre de transactions dans lesquelles A est employé (c.-à-d., 7).

Ainsi, les valeurs de confiance pour A→DF, D →AF et DF→A sont 6/7, 6/6, et 6/6, respectivement. De même la confiance pour DF→A, AF→D, AD→A est 6/6, 6/6, et 6/6, respectivement. Puisque la valeur la plus basse (6/7 ou 86%) est plus haute que le niveau de confiance minimum prédéfini (disons que min_conf = 30%), l'association {A, D, F} est maintenue dans l'ensemble des grands items filtrés L3.

B. *DEDUCTION DES FRAGMENTS VERTICAUX :*

L'ensemble d'items {A, D, F} est pris de L3 en tant qu'une partition ; alors nous allons sur L2 balayer l'ensemble 2-items et constater qu'il y a seulement {B E} qui ne recouvre pas avec la partition existante. L'attribut restant {C} est pris de L1, ayant pour résultat le schéma de fragmentation, (C, B E, A D F).

De la même façon, en considérant l'autre ensemble 3-items {A, B, E} de L3, nous obtenons (C, D F, A B E) comme autre solution.

Parmi ces deux schémas de fragmentation, celui qui a le moindre coût d'exécution selon le modèle de coût est choisi comme le schéma de fragmentation.

III -3. LES ALGORITHMES GENETIQUES POUR LA FV:

Nous présentons dans cette section comment nous avons utilisé les algorithmes génétiques pour sélectionner un schéma de fragmentation verticale.

III- 3- A. LE CODAGE

Nous avons adopté un codage assez simple, les individus représentent la manière dont laquelle la table sera fragmentée verticalement, en d'autre terme les groupes d'attributs que nous allons formés. Le chromosome est une collection d'entiers, la longueur du chromosome est égale aux nombres de d'attributs dans la table, chaque entier exprime la partition à laquelle l'attribut sera affecté.

Exemple : Table avec 10 attributs, le codage suivant : chromosome {5, 0, 5, 5, 0, 1, 1, 2, 3, 4} sera interprété comme suit :

- *Partition 0* : {1, 4} ; *Partition 1* : {5, 6} ; *Partition 2* : (1) ;
- *Partition 3* : {8} ; *Partition 4* : (1) ; *Partition 5* : {0, 2, 3}.

III.3.A.1. LA POPULATION

L'initialisation de la population s'effectue d'une manière totalement aléatoire. La taille de la population peut être ajustée par l'utilisateur, et le nombre de génération peut être aussi renseigné par l'utilisateur.

III.3.A.2. LE CROISEMENT

Un point est choisi au hasard dans le premier chromosome, avec une certaine probabilité, il sera actif pour le croisement selon une probabilité Pc de croisement (renseigné au préalable par l'utilisateur).

Un autre point est choisi au hasard sur le second chromosome en utilisant le même procédé, mais le point choisi doit être dans le même ordre que le point choisi dans le premier chromosome.

Si le point correspondant dans le second chromosome n'a pas pu être trouvé, alors les chromosomes ne seront pas croisés, voir l'exemple dans la Figure 14.

Exemple :

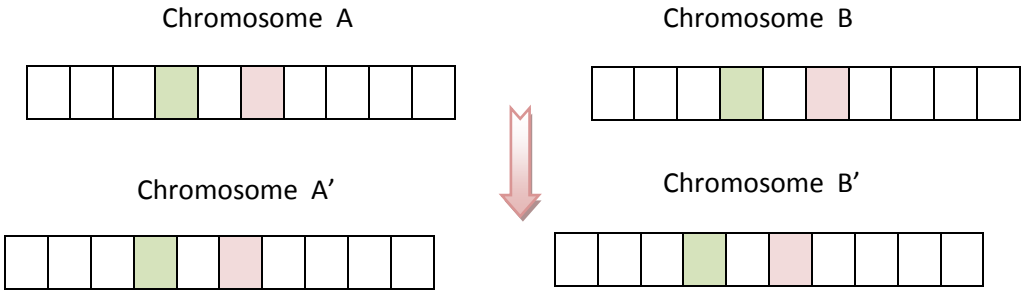


Figure 14 : Exemple de croisement

III.3.A.3. LA MUTATION

L'opérateur de mutation parcourt chaque gène des chromosomes candidat retenu par le procédé de la sélection dans la population, et les mute selon un taux de mutation donnée (renseigné au préalable par l'utilisateur).

III.3.A.4. LA SÉLECTION

Dans notre approche nous offrons la possibilité de procéder par les différents modes de sélections présentées dans les paragraphes précédents.

Le mode de sélection standard permet de sélectionner un nombre donné de chromosomes de la population pour leur permettre de continuer à survivre.

Cette sélection devrait être guidée par les valeurs de fitness, mais la fonction fitness doit être considérée comme une probabilité statistique de survie, non pas comme le seul facteur déterminant pour la survie. En d'autres termes, les chromosomes avec des valeurs de fitness plus élevées sont plus susceptibles d'être sélectionnés que ceux avec des valeurs fitness inférieure, sauf que ce n'est pas une chose toujours garantie.

III.3.A.5. LA FONCTION FITNESS

Concernant la fonction fitness, l'évaluation est basée sur le modèle de coût de Yao [52]. L'objectif de l'algorithme sera de chercher les solutions avec le moindre coût, les solutions dont le temps d'exécution sera le plus réduit.

La fonction coût (fitness) est estimée sur la base de deux opérations:

• le temps de d'accès de données dans le support de stockage des données, qui dépend de la quantité de données transférées.

• Temps de traitement des données dans le CPU, mais dans notre cas, le temps de traitement du CPU est négligé par rapport au temps de transfert des données car il est assez faible.

Inspiré des travaux de Gorla & al [32], dans cette étude le coût d'accès à l'index et le coût de stockage ne sont pas pris en compte, l'estimation se base sur le temps d'accès aux blocs de données pertinents qui est donnée par Yao [53] comme base de calcul du coût du traitement des requêtes. S'il y a *"n"* enregistrements divisés en *"m"* blocs et si *"k"* enregistrements pertinents répondent à une requête sont répartis uniformément entre les *"m"* blocs, alors le nombre de blocs accessibles pour la requête est égal à : *"m (1 - (1 - 1 / m) k)"*, Yao [53]. Dans la base de données partitionnée, nous appliquons cette formule pour chaque segment *i* auquel accède la requête q. Ainsi, le coût de traitement d'une requête (de récupération) q est estimé comme indiqué dans la formule 1, où la première partie calcule le nombre total de blocs auxquels accéder à partir de chacune des partitions du segment qj. La deuxième partie indique la surcharge pour concaténer les partitions de segment qj dans la mémoire. Où:

Freq_i = fréquence de la requête q_i
Segment qj = nombre de partitions j requise par la requête q.
$M_i = [T * L_i/BS]$
K_q *: nombre de tuples satisfaisant la requête q.*
T : nombre total de tuples.
Li : taille d'une partition.
BS : taille de bloque.

$$\mathrm{freq_q} \times \left(\left(\sum_{i=1}^{\mathrm{Segment\ qj}} \left[\mathrm{M_i} \left(1 - \left(1 - {}^{1}/_{\mathrm{M_i}} \right)^{\mathrm{Kq}} \right) \right] \right) \times \left(1 + \left(0.1 \times \left(\mathrm{Segment_{qj}} - 1 \right) \right) \right) \right)$$

Équation 12 : Modèle de cout de Yao [53].

III.3.A.6. ILLUSTRATION DE L'ALGORITHME GÉNÉTIQUE:

Dans notre exemple nous allons illustrer l'optimisation de la fragmentation verticale par les algorithmes génétiques de la table personnes composé de 11 attributs dont l'attribut code est la clé primaire de la table :

Personnes : {***Code*** ; Nom ; Prénom ; Date-N ; Lieu-N ; Taille ; Adresse ; Photo ; Empreinte ; Code-ADN ; Remarque}.

L'algorithme sera chargé de fragmenter la table verticalement par rapport à une charge de travail, cette fragmentation aura pour objectif de réduire le temps d'exécution de la charge de travail.

L'algorithme génétique que nous allons appliquer aura les caractéristiques suivante : la taille de la population P est 100, l'initialisation de la population sera générés aléatoirement, la probabilité de croisement sera Pc, la probabilité de mutation sera Pm.

Sur la table personnes, la taille des chromosomes sera 10, puisque l'attribut Code ne sera pas pris en compte, pour raison de contraintes sur la clé primaire.

- 1: création de la génération 0.

Dans ce qui suit nous allons simuler le déroulement de l'algorithme génétique sur cet exemple : P : {A1 ; A2 ; … ; A100} ;

Supposons que la longueur de chaque chromosome est égale à 10,

Soit : A1 = {5, 0, 5, 5, 0, 1, 1, 2, 3, 4} ; A2 = {2, 1, 1, 1, 0, 1, 1, 2, 3, 4} ; A3 = {….} ; etc.

- 2: évaluation de la qualité de chaque individu.

Les chromosomes de la population seront évalués et leurs valeurs fitness correspondantes sera calculé par l'application de la formule du modèle de coût de Yao [52], supposons que les valeurs de fitness seront : *f*-A1 : 14520 ; *f*-A2 : 54620 ; *f*-A3 : 1458720 ; etc.…

- 3: sélection des futurs parents, croisement et mutation.

Selon le mode de sélection plusieurs chromosomes seront choisis pour la reproduction, supposons que A1 et A2 seront choisies. L'algorithme procèdera au croisement et mutation, le schéma suivant illustrera le procédé de mutation et croisement sur A1 et A2.

Sur la Figure 15, la procédure de croisement parcoure chaque gène du chromosome A1, et sélectionne des gènes selon la probabilité Pc (si Pc = 0.8, 80% seront sélectionnés), le même nombre de gènes sera sélectionné sur le chromosome A2, sur des points différents, une fois les emplacements du croisement définis, l'algorithme échangera les gènes marqués sur les chromosomes.

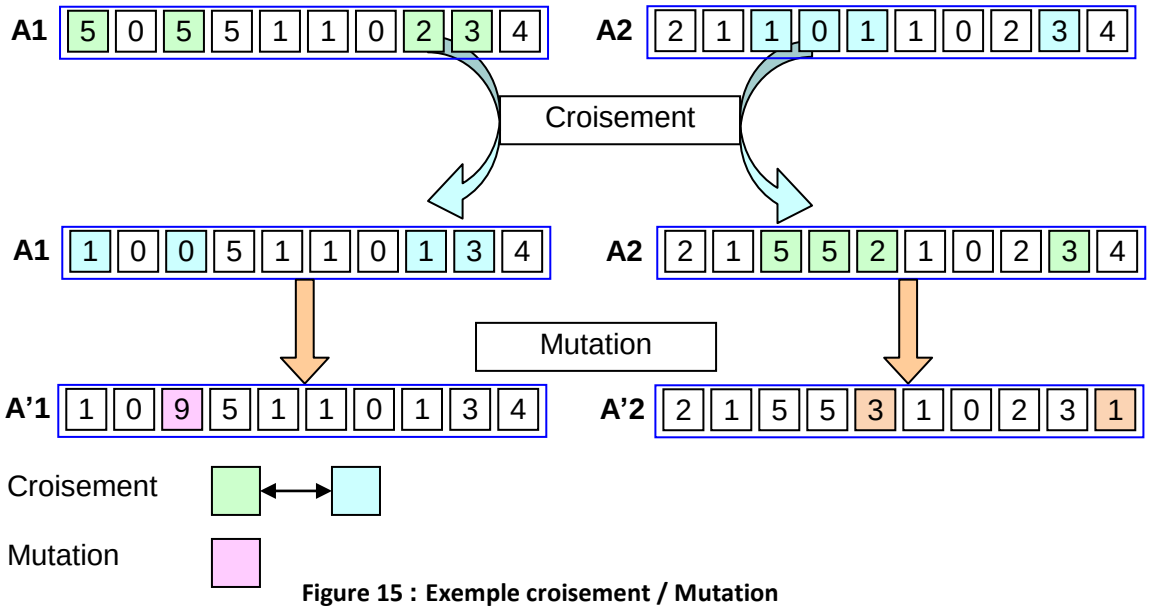


Figure 15 : Exemple croisement / Mutation

- 4: obtention de la nouvelle génération.

En suivant le même concept, une proportion de gènes sera sélectionnée selon la probabilité Pm. Les points marqués seront altérés avec une autre valeur aléatoire comprise entre 0 et 9. Nous remarquons que le chromosome n'a pas été altéré par la mutation car la probabilité Pm est assez faible.

Les chromosomes A1 et A2 ont générés deux nouveaux chromosomes A'1 et A'2, ces deux derniers formeront la nouvelle génération, et une évaluation du fitness de chaque chromosome nous permettra d'évaluer la qualité des solutions proposées par l'algorithme, et une sélection des chromosomes avec le meilleur fitness nous permettra de généré la nouvelles génération selon la méthode élitiste.

Le procédé décrit si dessus sera appliqué sur tous les chromosomes choisis par l'algorithme de sélection, jusqu'à atteindre le nombre de générations prédéfinis au préalable, dans notre cas, nous avons choisis le nombre de 100 population.

III.3.A.7. DISCUSSION

Dans cette partie nous avons donné un aperçu global sur les algorithmes génétiques, leurs fonctionnements, et les différents modules qui les composent.

Les algorithmes génétiques ont fait leurs preuves dans la résolution d'un grand nombre de problèmes, comme le problème du voyageur de commerce, ou problèmes

non linéaires, comme les systèmes adaptatifs, problèmes d'optimisation multi-objectifs, comme pour les systèmes industriel, ou de prise de décision. [11].

Les algorithmes génétiques présentent une grande souplesse et aisance dans leurs mise en œuvre et implémentation, c'est pour cette raison qu'un grand nombre de chercheurs et développeurs de systèmes industriels, informatiques, automatique et autres leurs portent un grand intérêt et les ont adopté.

Néanmoins, le bon choix du codage et des paramètres (probabilité de croisement, de mutation, nombre de générations et taille de la population) n'est pas une chose facile, car rien ne garantie que le codage soit approprié à notre problème, puisqu'un le codage représente un paramètre important et significatif pour le succès de l'algorithme et permet d'avoir de meilleurs résultats, de plus le modèle de coût présente une grand enjeux pour que la recherche converge vers un optimum globale, et fais ressortir la meilleur combinaisons représentons le meilleur partitionnement.

III- 3- B. OPTIMISATION SCHÉMA FV: AFFINER

L'analyse et l'adaptation des algorithmes génétique et fouille de données, pour fragmenter verticalement une base de données par rapport à une charge de travail définie, nous a permis de déduire l'algorithme Affinité, un algorithme original conçu par nos soins, et que nous avons adapté pour la fragmentation verticale.

La motivation qui nous a poussés à proposer cette approche, est l'intuition sur le fait qu'il pouvait exister une approche plus adaptée au problème de la fragmentation verticale.

L'idée principale se base sur l'analyse et l'évaluation de l'affinité entre chaque attribut, puis le regroupement des attributs fortement affine.

Un des atouts majeur de cet algorithme est qu'il effectue une recherche sur l'ensemble des attributs et en déduit les partitions possibles sans utilisé un modèle de coût ce qui réduit énormément le temps de calcul, sachant que l'estimation du modèle reste théorique et peut changer d'un SGBD à un autre, d'une version à une autre, et d'une architecture à une autre.

En fait, après l'analyse de chaque attribut séparément, seul les partitions les plus intéressantes seront retenues pour la composition du schéma de fragmentation final.

[11] http://fr.wikipedia.org/wiki/Algorithme_génétique.

Les attributs seront comparés un par un, par rapport à l'ensemble des autres attributs, en prenant en compte la charge de travail, en fait le calcul de l'affinité s'effectuera par rapport à chaque requête. Les attributs présentons une plus grande affinité seront regroupés dans la même partition, cette opération sera réitérée jusqu'à ce que l'ensemble des attributs soit évalué.

Dans ce qui suit nous allons donner une explication plus détaillée sur l'algorithme Affiner.

III.3.B.1. TERMINOLOGIE

- S ensemble des éléments candidats construits à partir des attributs de la table, exemple : S = {Nom, Prénom, …},
- Ai attribut numéro i,
- Ai(k) = 1, si l'attribut i est requis pour la requête k,
- Ai(k) = 0, si l'attribut i n'est pas requis pour la requête k,
- P = est l'ensemble des partitions, générés à partir de la combinaison des éléments S,
- Aff(i,j) = Affinité entre l'attribut i et j .
- Partition(i) : ensemble d'attribut représentant la partition numéro i
- Partition(i).add(Ai) : fonctionnalité d'ajout de l'attribut Ai à la partition.
- B l'ensemble des partitions retenues.

Les termes que nous avons définis seront utilisés dans l'algorithme Affiner. L'algorithme se base sur l'évaluation de l'affinité entre tous les attributs par rapport aux autres attributs, si l'indice d'affinité entre deux attributs est élevé, un regroupement entre les attributs sera effectué, chaque regroupement présentera une partition, la partition retenue sera retiré de l'ensemble S, car il serra considéré comme un élément arrangé.

Le schéma de fragmentation sera la combinaison des éléments de « F » avec l'ensemble des éléments restant en une seule partition.

L'algorithme évalue le schéma de fragmentation afin qu'il puisse être comparé aux autres algorithmes implémenté dans notre étude.

L'algorithme sera détaillé et déroulé avec un exemple dans la section suivante :

III.3.B.2. DÉROULEMENT DE L'ALGORITHME AFFINER

- L'ensemble des candidats est généré par la décomposition élémentaire des attributs du problème,
- Génération des partitions candidates F, en combinant les éléments de S,
- à partir de l'ensemble des éléments B déduire le schéma de fragmentation.

L'énoncé de l'algorithme :

- P_0 : $Tableintialesanspartition$;
- P_k: $Partitionnumérok$;
- Ai: $Attributnumeroi$;
- $Affinity(A_x, A_y)$: Valeur de l'affinité entre l'attribut Ax et Ay;

Begin :

$Var\ k = 0;$

$for\ each\ element\ (A_x, A_y) \in \{Ordre_{Affinity}\} do$

$if\ \Big((A_x \in P_0) and (A_y \in P_0)\Big)\ then$

$$\begin{cases} P_k \leftarrow (A_x, A_y)\ ; (new\ partition) \\ P_0 \leftarrow P_0 - (A_x, A_y); \\ k \leftarrow k + 1; \end{cases}$$

$elseif\ \Big((A_x \in P_0) and (A_y \in P_z)\Big)\ then$

$$\begin{cases} if (\forall A_i \in P_z \Rightarrow Affinity(A_i, A_x) \geq 0.5) \\ \qquad then \begin{cases} P_z \leftarrow P_z + (A_i); \\ P_0 \leftarrow P_0 - (A_i); \end{cases} \end{cases}$$

$$elseif\ \Big((A_x \in P_z) and (A_y \in P_r) and (P_z \neq P_t)\Big)\ then \begin{cases} if \begin{pmatrix} (\forall A_i \in P_z \Rightarrow Aff(A_i, A_y) \geq 0.5) \\ and (\forall A_z \in P_y \Rightarrow Aff(A_z, A_j) \geq 0.5) \end{pmatrix} \\ \qquad then \begin{cases} P_x \leftarrow (P_x \cup P_y); \\ P_y \leftarrow \emptyset; \end{cases} \end{cases}$$

End.

Algorithme 2 : Algorithme Affiner.

Déroulement d'affiner L'algorithme Affiner est énoncé comme suit ;

a. Génération de la matrice d'usage,
b. calcul de la matrice d'affinité,
c. tri de la matrice d'affinité dans l'ordre descendant selon les valeurs d'affinités,
d. déduire le schéma de fragmentation à partir de l'ensemble ordonné.

Génération de la matrice d'usage : Pour une relation donnée, la matrice d'usage représente une matrice qui trace le lien entre les requêtes et les attributs, dans cette matrice si la requête fait appel à l'attribut la case correspondante lui sera affecté « 1 » sinon « 0 ».

Illustration : Soit la relation T1 composée des attributs {A_1, A_2, A_3, A_4, A_5, A_6}, et soit les requêtes suivantes faisant appels aux attributs.

R_1 = {A_7}
R_2 = {A_1, A_8}
R_3 = {A_2, A_3, A_4}
R_4 = {A_2}

R_5 = {A_1, A_2, A_3}
R_6 = {A_2, A_3, A_6, A_8}

La matrice d'usage, et matrice d'affinité sont présentées dans le tableau suivant.

	A1	A2	A3	A4	A5	A6	A7	A8
R1	0	0	0	0	0	0	1	0
R2	1	0	0	0	0	0	0	1
R3	0	1	1	1	0	0	0	0
R4	0	1	0	0	0	0	0	0
R5	1	1	0	1	0	0	0	0
R6	0	1	1	0	0	1	0	1

Tableau 12 : Matrice d'usage

	A1	A2	A3	A4	A5	A6	A7	A8
A1	**1**	**2/6**	**0**	**2/4**	**0**	**0**	**0**	**2/4**
A2		**1**	**4/6**	**4/6**	**0**	**2/5**	**0**	**2/6**
A3			**1**	**2/4**	**0**	**2/3**	**0**	**2/4**
A4				**1**	**0**	**0**	**0**	**0**
A5					**1**	**0**	**0**	**0**
A6						**1**	**0**	**2/3**
A7							**1**	**0**
A8								**1**

Tableau 13 : Matrice d'affinité

Calcul de l'affinité: Dans notre étude nous avons pu constater que l'affinité entre les attributs est principalement régie par le nombre de requêtes faisant appel aux attributs. Si deux attributs sont invoqués par les mêmes requêtes il serait bénéfique de les placer dans la même partition, sinon il est plus intéressant de les placer dans des partitions séparées.

Sur la base de ces hypothèses, nous proposons la formule suivante qui permet le calcul de l'affinité entre deux attributs :

Ai *: colonne numéro i de la matrice d'usage, représentant un attribut,*

Aik *: élément de la matrice d'usage colonne n° i et ligne n° k,*

t *: nombre total de requêtes.*

$$\text{Affinity}(A_i\ ;\ A_j) = = \frac{nbr\acute{d}\ invocation\ de\ A_i\ \ et\ A_j en\ commun}{nbr\ total\ d\acute{}invocations\ de\ A_i\ et\ de\ A_j \hbar\ \hbar\ \hbar}$$

$$= \frac{2 \times \sum_{k=0}^{t} \left((A_{ik} \times A_{jk}) \right)}{\left(\sum_{k=0}^{t} A_{ik} + \sum_{k=0}^{t} A_{jk} \right)}$$

La formule d'affinité présente les propriétés suivantes :

- la formule est symétrique :
$$\text{Affinity}(A_i\ ;\ A_j) = \text{Affinity}(A_j\ ;\ A_i)$$
- $0 \leq \text{Affinity}(A_i\ ;\ A_j) \leq 1$;
- $\text{Affinity}(A_i\ ;\ A_j) = 1$;

Toutes les requêtes qui font appel à Ai et les requêtes qui font appel à Aj sont identiques, on déduit que chaque attribut est totalement affine avec lui-même, ainsi :

$$\text{Affinity}(A_i\ ;\ A_i) = 1;$$

- $\text{Affinity}(A_i\ ;\ A_j) = 0.5;$ la moitié des requêtes qui font appel à Ai font appel à la moitié des requêtes qui font appel à Aj ;
- $\text{Affinity}(A_i\ ;\ A_j) = 0;$ toutes les requêtes qui font appel à Ai ne font aucun appel à Aj.

Le Tableau 12 représente un exemple de matrice d'affinité calculé selon la matrice d'usage du Tableau 13.

Illustration:

$$\text{Affinity}(A_3\ ;\ A_4) = \frac{2 \times \big((0 \times 0) + (0 \times 0) + (1 \times 1) + (0 \times 1) + (1 \times 0)\big)}{\big((0+0+1+0+0+1) + (0+0+1+0+1+0)\big)}$$

$$\text{Affinity}(A_3\ ;\ A_4) = \frac{2 \times 1}{(2+2)} = \frac{1}{2}$$

$$\text{Affinity}(A_2\ ;\ A_3) = \frac{2 \times \big((0 \times 0) + (0 \times 0) + (1 \times 1) + (1 \times 0) + (1 \times 1)\big)}{\big((0+0+1+1+1+1) + (0+0+1+0+0+1)\big)}$$

$$\text{Affinity}(A_2\ ;\ A_3) = \frac{2 \times 2}{(4+2)} = \frac{2}{3}$$

De la même façon nous calculerons la matrice d'affinité, vu le caractère symétrique de la formule d'affinité, nous obtiendrons une matrice symétrique composée de valeurs comprises entre 1 et 0.

Tri de la matrice d'affinité : Avant de générer les partitions nous passons par l'étape de tri de la matrice, qui consiste à trier les affinités des pairs d'attributs dans un ordre descendant selon leur affinité dans l'ensemble qu'on nommera Ordre_Affinity, ce tri est utile pour la génération de partitions regroupant les attributs les plus affines.

Dans l'ensemble Ordre_Affinity ne seront retenus que les attributs d'une affinité supérieure ou égale à 0.5 (dans le tableau 13 case grise), valeur qui signifie que les attributs sont référencés au moins par la même moitié de requêtes, la raison qui nous a porter à choisir cette condition (affinité ≥ 0.5), est le besoin de trouver un compromis entre trop de partitions donc beaucoup de les jointures entre les partitions (pour la reconstruction de la table) ce qui est assez couteux ; et le parcours de tous les attributs dont une majorité seront inutile pour la requête, ce qui occupera inutilement les ressources. Sachant qu'un schéma de partition qui satisfait une requête peut dégrader les performances pour d'autres requêtes.

Pour le classement des attributs de même affinité nous nous référons aux nombres d'appels des attributs dans la matrice d'usage($\sum_{k=0}^{t} A_{ik}$), ainsi nous obtiendrons un

ensemble ordonné des affinités ce qui nous permettras une classification des attributs et la génération de partitions.

Illustration: Pour l'exemple précédent on obtiendra L'ensemble Ordre_Affinity suivant:

Ordre_Affinity = {(A3;A2), (A2;A4), (A3;A6), (A6;A8), (A3;A4), (A3;A8), (A1;A4), (A1;A8)}.

Génération des partitions : le processus de génération des partitions (ensemble d'attributs) est formulé comme suit, l'algorithme effectue une analyse de chaque élément (Ai, Aj) de l'ensemble Ordre_Affinity en commençant par l'ordre descendant, pour chaque élément de Ordre_Affinity(Ai, Aj) l'algorithme considérera l'un des trois cas suivant :

1. Si les deux attributs n'appartiennent à aucune partition (ensemble d'attributs) alors une nouvelle partition sera générée regroupant les deux attributs Px = {Ai, Aj} ;
2. Si l'un des attributs n'appartient à aucune partition et l'autre appartient à une partition alors →soit : $\{A_i \in \emptyset \; et \; A_j \in P_y\}$ où Py désigne une partition donnée; alors on étudie la possibilité de l'affectation de Ai à Py, dans ce cas, Ai doit satisfaire une affinité acceptable (≥0.5) avec tous les éléments de Py, formule : $(\forall A_i \in P_z \Rightarrow Affinity(A_i, A_x) \geq 0.5)$, sinon Ai ne sera affecté à aucune partition;
3. Enfin Si les deux attributs appartiennent à deux ensembles distincts → Par exemple pour l'élément (Ai, Aj) si $(Ai \in Px)$ et $(Aj \in Py)$ et $(Px \neq Py)$ on étudie la possibilité de fusionner les deux ensembles Px et Py, pour faire, if faut que l'affinité entre tous les éléments des deux partitions soit satisfaisante (≥0.5), sinon les deux partitions Px et Py resteront disjointes.

III- 3- c. TESTS ET RÉSULTATS 1 :

Sur les différents bancs de tests ou benchmarks existants nous avons opté pour l'utilisation du Benchmark OLAP APB1 Release II [13], un benchmark offrant la possibilité de créer des schémas de bases de données et des table assez volumineuses ce qui nous permet de mener à bien notre recherche. Aussi APB1 Release II est très facile à implémenter, aussi le mode de génération de la base de données et du script ne nécessite pas de très grandes connaissances dans le domaine des bases de données, en plus nous avons noté que le benchmark APB1 est universelle par rapport à tous les SGBD (Oracle, IBM, …).

Pour notre application nous avons généré le schéma APB1 R II, et nous avons choisi de fragmenter la table des faits de HistInventory composé de 19 attributs contenant 27.700.646 enregistrements :

Hist_Inventory { customer_level char(12), product_level char(12), inv199501 Integer, inv199502 Integer, inv199503 Integer, inv199504 Integer, inv199505 Integer, inv199506 Integer, inv199507 Integer, inv199508 Integer, inv199509 Integer, inv199510 Integer, inv199511 Integer, inv199512 Integer, inv199601 Integer, inv199602 Integer, inv199603 Integer, inv199604 Integer, inv199605 Integer}.

Le SGBD ou moteur de base de données sur lequel nous avons effectué nos tests est Oracle 11G, une version très stable de SGBD, largement utilisée dans les entreprises manipulant les bases de données de type OLAP.

Sachant que le SGBD Oracle ne supportant pas la fragmentation verticale nous avons été contraint d'ajouter une clé additionnelle de type entier auto incrémentée à la table Hist_Inventory, afin que l'on puisse effectuer une jointure entre les différentes partitions générées.

Une fois les recommandations énoncées par l'application que nous avons développé, on procède à la génération des vues matérialisées (snapshot) représentant les partitions recommandées par FrgaMan, puis on effectue une réécriture de toutes les requêtes composant la charge de travail pour qu'elles puissent s'exécutées selon les vues matérialisées générées.

Dans cette partie on a effectué plusieurs tests sur la base de données qui sont illustrés ci-dessous :

III.3.C.1. PREMIER TEST:

Dans ce premier test, on a choisi une charge de travail contenant quatre requêtes s'exécutant spécialement sur la table HistInventory, après l'exécution des tests, on obtient les recommandations qu'on implémente manuellement sur la base de données, et on obtient les temps d'exécution réels de la charge de travail sur le nouveau schéma partitionné.

Charge de travail:

Q1 :
```
select  product_level, inv199503, inv199509
from histinventory
where inv199503> 10 and inv199509 < 15;
```

Q2 :
```
select  product_level, inv199503, inv199505, inv199509, inv199602
from histinventory
where inv199506> 10 and inv199511< 12;
```

Q3 :
```
select  inv199503, inv199509
from histinventory
where product_level like '%M';
```

Q4 :
```
select  inv199503, inv199505, inv199506, inv199511
from histinventory
where product_level like '%K' and inv199509> 10 and inv199602< 12;
```

Avec les fréquences suivantes : fréquence (Q1) = 3, fréquence (Q2) = 3, fréquence (Q3)=3, fréquence (Q4)=3. On obtient les résultats suivants :

```
Genetic Algorithm:

  Total evolution time: 1234 ms

  Partition : CUSTOMER_LEVEL ; INV199501 ; INV199502 ; INV199504 ; INV199505
  Partition : PRODUCT_LEVEL ; INV199503 ; INV199509
  Partition : INV199506 ; INV199510 ; INV199603 ; INV199604
  Partition : INV199507
  Partition : INV199508
  Partition : INV199511
  Partition : INV199512 ; INV199601
  fitness : 1458.0 ;

Apriori Algorithm:

  Execution time is: 0.002 seconds.

  Partition : INV199506 ; INV199510 ; INV199603 ; INV199604
  Partition : PRODUCT_LEVEL ; INV199503 ; INV199509
  Partition : CUSTOMER_LEVEL ; INV199501 ; INV199502 ; INV199504 ; INV199505 ; INV199507 ;
              INV199508 ; INV199511 ; INV199512 ; INV199601 ; INV199602 ; INV199605
  fitness : 1458.0 ;


Affiner Algorithm

  Execution time is: 0.001 seconds.

  Partition : CUSTOMER_LEVEL ; INV199501 ; INV199502 ; INV199504 ; INV199505 ; INV199507 ;
              INV199508 ; INV199511 ; INV199512 ; INV199601 ; INV199602 ; INV199605
  Partition : PRODUCT_LEVEL ; INV199503 ; INV199509
  Partition : INV199506 ; INV199510 ; INV199603 ; INV199604
  fitness : 1458.0
```

Les temps d'exécutions des requêtes sont représentés sur la figure suivante :

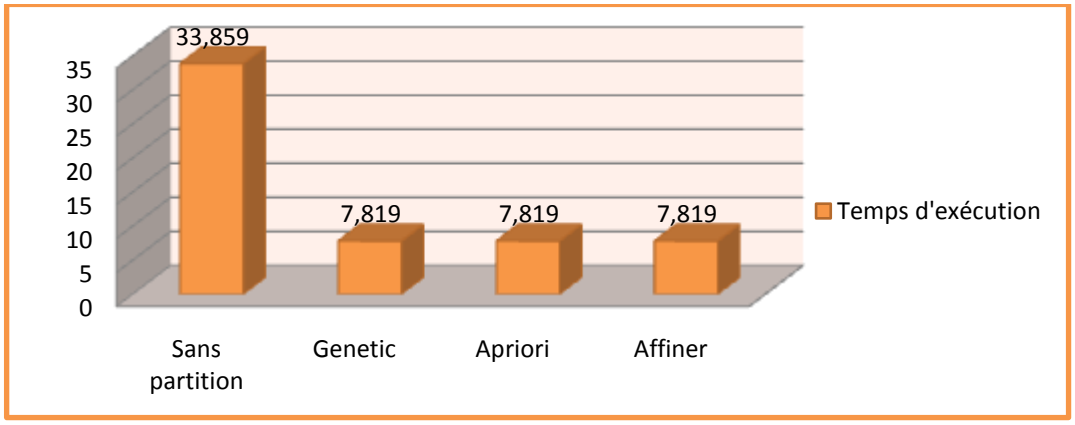


Figure 16 : Temps d'exécution Test 1.

Le gain en performance en temps d'exécution de la charge de travail à diminuer d'une manière significative avec un taux de 80%.

III.3.C.2. DEUXIÈME TEST:

Charge de travail :

```
Q1 :   Select a.inv199603
         from histinventory a
         where inv199604>20
Q2 :   Select INV199603 , INV199604
         From histinventory a
         Where INV199603 >5;
```

Avec les fréquences suivantes :

Fréquence (Q1) = 1, fréquence (Q2) = 2,

Une fois le logiciel de test exécuté, les recommandations appliquées sur la base de données, les temps d'exécutions des requêtes obtenues sur le nouveau schéma sont représentés sur la figure suivante :

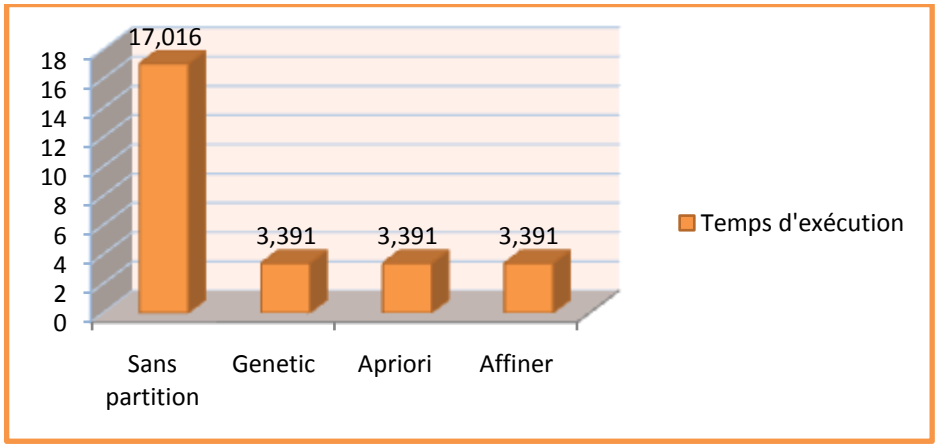


Figure 17 : temps d'exécution Test 2

III.3.C.3. TROISIÈME TEST :

Charge de travail:

```
Q1 :   select  product_level,inv199503,inv199505,inv199506
       from histinventory
       where inv199509 > 2 and inv199511 <2 and inv199602 >2;
Q2 :   select  product_level,inv199503,inv199511,inv199602
       from histinventory
       where Inv199509 > 2 and inv199603>15 and inv199604 >10;
Q3 :   select  product_level,inv199503,inv199505, inv199603, inv199602
       from aahistinventory
       where inv199509 > 2 and inv199511 <2 and inv199604 >2 and inv199506=5;
Q4 :   select  product_level, Inv199509 ,inv199503,inv199511,inv199602
       from histinventory
       where product_level Like '%T'and inv199603>15 and inv199604 >10;
Q5 :   select  inv199509,inv199503,inv199505,inv199506
       from histinventory
       where product_level Like '%T' and inv199511 <2 and inv199602 >2;
Q6 :   select  product_level,inv199503,inv199505,inv199506, inv199603,inv199604
       from histinventory
       where inv199509 > 2 and inv199511 <2 and inv199602 >2;
```

Avec les fréquences suivantes : fréquence (Q1) = 1, fréquence (Q2) = 1, fréquence (Q3) = 1, fréquence (Q4) = 1, fréquence (Q5) = 1, fréquence (Q6) = 1,

Une fois le programme exécuté, on obtient les recommandations suivantes :

```
Genetic Algorithm:

  Total evolution time: 1454 ms

  Partition : CUSTOMER_LEVEL ; INV199501 ; INV199502 ; INV199504
  Partition : PRODUCT_LEVEL ; INV199503 ; INV199509 ; INV199511 ; INV199602
  Partition : INV199505 ; INV199506
  Partition : INV199507 ; INV199508 ; INV199510 ; INV199512 ; INV199601
  Partition : INV199603 ; INV199604
  Partition : INV199605
  fitness : 3904.0


Apriori Algorithm:

  Execution time is: 0.047 seconds.
  Solution générée
  Partition : PRODUCT_LEVEL ; INV199503 ; INV199509 ; INV199511 ; INV199602 ; INV199603 ;
              INV199604
  Partition : INV199505 ; INV199506
  Partition : CUSTOMER_LEVEL ; INV199501 ; INV199502 ; INV199504 ; INV199507 ; INV199508 ;
              INV199510 ; INV199512 ; INV199601 ; INV199605
  fitness : 4456.0

Affiner Algorithm

  Execution time is: 0.016 seconds.
  Partition : CUSTOMER_LEVEL ; INV199501 ; INV199502 ; INV199504 ; INV199507 ;
              INV199508 ; INV199510 ; INV199512 ; INV199601 ; INV199605
  Partition : PRODUCT_LEVEL ; INV199503 ; INV199509 ; INV199511 ; INV199602
  Partition : INV199505 ; INV199506
  Partition : INV199603 ; INV199604
  fitness : 3896.0
```

Les recommandations appliquées sur la base de données, on obtient les temps d'exécutions des requêtes sur le nouveau schéma qui sont représentés sur la figure suivante :

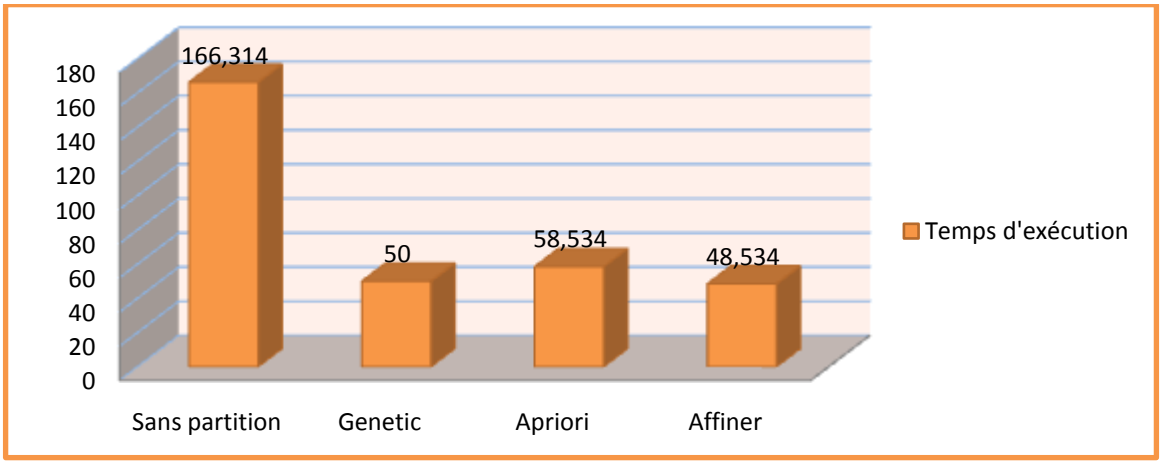


Figure 18 : temps d'exécution Test 3

Sur la Figure 18, on remarque que l'algorithme Affiner nous a procuré les meilleures recommandations, en deuxième position l'algorithme Génétique a donné des résultats presque aussi similaires, et en troisième position l'algorithme Apriori a donné des solutions plus couteuses en temps d'exécution par rapport à Affiner et au Génétique.

III- 3- D. TESTS ET RÉSULTATS 2:

Les algorithmes implémentés dans l'outil développé que nous avons nommé FragMan se basent sur une estimation théorique et offre la possibilité de fragmenter un schéma virtuelle, afin de valider cette étude nous avons procéder à des tests dont nous allons présenter les différents résultats obtenus, les résultats représentent la fragmentation verticale réel de la base de données (benchmark) sur la base des recommandations de FragMan.

Sur les différents bancs de tests ou benchmarks existants nous avons opté pour l'utilisation du benchmark TPCH, un benchmark offrant la possibilité de créer des schémas de bases de données et des table assez volumineuse ce qui nous permet de mener à bien notre recherche. Aussi TPCHest très facile à implémenter, et le mode de génération de la base de données et du script ne nécessite pas de très grandes connaissances dans le domaine des bases de données, en plus nous avons noté que le benchmark TPCH représente une référence pour les tests des différents SGBD (Oracle, IBM, …).

Pour notre application nous avons générer le schéma TPCH, et nous avons choisi de fragmenter la table de faits dite *LineItem* composé de 16 attributs contenant 6001215 enregistrements avec une taille globale de la base de 1GB:

LineItem {l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_linestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_comment}.

Le SGBD ou moteur de base de données sur lequel nous avons effectué nos tests est Oracle 11G, une version très stable de SGBD, largement utilisé dans les entreprises manipulant les bases de données de type OLAP, sur une machine Sony Vaio, dotée d'un processeur Intel Core i3, avec 4 GO de RAM, et un espace disque largement suffisant.

Sachant que le SGBD Oracle ne supportant pas la fragmentation verticale, plusieurs choix se sont présenter à nous pour simuler la fragmentation verticale, dont nous citons création de vue matérialisée, génération d'index composé, ou des sous tables, nous avons optés pour le choix de génération de sous tables ce qui semble le plus stable des choix, et celui qui consomme moins de ressources.

Afin d'optimiser l'espace de stockage nous avons été contraint de créer une nouvelle clé primaire basée sur la clé primaire composée de la table initiale *LineItem*, et la dupliquer sur les différents fragments, afin que l'on puisse reconstituer la table *LineItem* a partir des différents fragments ou partitions générées.

Une fois les recommandations énoncées par *FragMan*, on procède à la génération de tables représentant les partitions recommandées par *FrgaMan*, puis on effectue une réécriture de toutes les requêtes composant la charge de travail pour qu'elles puissent s'exécutées selon les partitions générées.

La charge de travail (requête) a été sélectionnée à partir des 22 requêtes, présente dans le benchmark TPCH avec les fréquences respectives de 1 occurrence pour chaque requête.

Table Lineitem	Q3	Q5	Q6	Q12	Q14	Temps Secondes	Réduction %
Non fragmentée	14,086	17,144	13,810	16,870	14,280	76,190	0%
Fragmentée Par Apriori	2,565	17,725	6,970	15,845	29,735	65,870	13.54%
Fragmentée par AG	5,741	8,807	14,605	17,595	14,640	52,581	30.99%
Fragmentée par Affiner	0.593	7,191	2,184	12,558	7,176	29,702	70.298%

Tableau 14 : Coût d'exécution des algorithmes

Le tableau 14 présente les résultats de l'exécution des requêtes sur une base de données réelle, inspiré du benchmark TPCH, le temps d'exécution des requêtes est estimé en secondes.

Classé par ordre décroissant Les valeurs obtenues présentent le coût d'exécution respectivement sur un schéma dont :

- la table lineItem n'est pas partitionnée,
- lineItem partitionnée selon les recommandations d'Apriori,
- lineItem partitionnée selon les recommandations du Génétique,
- lineItem partitionnée selon les recommandations d'Affiner,

Selon les résultats réels obtenus nous le remarquons que l'algorithme Affiner présente les meilleurs résultats comparé aux algorithmes génétiques et Apriori, le gain de performance en temps d'exécution de la charge de travail à diminuer d'une manière significative avec un taux de 59%, ce qui est considérable.

III -4. CONCLUSION

Dans cette étude nous nous sommes intéressés au sujet de la fragmentation verticale sur les bases de données orientées ligne, notre choix s'est porté sur l'utilisation de deux variantes algorithmiques basées sur l'évaluation par le modèle de coût, qui sont l'algorithme génétique et l'algorithme Apriori, et comme contribution nous avons proposé un troisième algorithme que nous l'avions baptisé Affiner.

Cette étude nous a permis de simuler l'application de la fragmentation verticale sur les bases et entrepôts de données orientés ligne.

L'algorithme Affiner semble bien répondre à la résolution du problème de la Fragmentation Verticale, cet algorithme peut être généralisé sur d'autres techniques de conception physique, comme la fragmentation horizontale.

Bien que les résultats obtenus semble être convainquant du point de vue expérimentale, mais en pratique ce n'es pas convainquant ; car la forme la plus connue pour implémenter la FV sur les BD consiste à créer des vues matérialisées ou des sous tables représentant chacune une partition.

Nous estimons que cette forme ne répond pas aux attentes des gestionnaires de bases de données, et que la fragmentation verticale trouvera un sens seulement si nous trouverons la solution pour appliquer facile la fragmentation verticale sur les bases de données orienté ligne, comme c'est le cas pour la fragmentation horizontale, nous voulons continuer dans cette voie et contribuer afin de proposer une architecture adéquate pour l'application de la FV sur les bases de données.

Chapitre IV : T-Plotter Nouvelle Conception Physique

IV-1. INTRODUCTION

Aujourd'hui la quantité d'informations et le volume qu'elles représentent a littéralement explosé. Les systèmes d'information traditionnels doivent faire face à d'énormes problèmes de performances liés à des modélisations et des structures parfois inadaptées à leurs besoins pour les analyses et la prise de décisions, traitant des requêtes lourdes englobant agrégation et jointures très complexes, lourdes à exécuter voire impossible, notamment, lors de l'exécution de requêtes de type OLAP (On Line Analytical Processing), sur des masses de données volumineuses, ou sur des tables surdimensionnées avec une centaine de colonnes. Et le constat est, l'utilisation des bases de données orientées lignes (OL) ne répond plus à la demande du marché de l'information.

En réponse à ce problème, de nouvelles solutions ont été proposées, s'orientant vers des architectures distribuées ou orientées colonnes (OC) [16]. De nouvelles tendances plus spécialisées au traitement des données volumineuses et requêtes OLAP émergent répondant mieux aux attentes des utilisateurs.

D'autre part, les bases de données OC s'adressent tout particulièrement a cette demande croissante de calculs complexes avec des temps de réponses devenant acceptables (e.g., C-Store [5]). Malgré cette nouvelle approche, cela ne répond qu'au problème des applications OLAP et non OLTP (On line Transaction Processing). Les solutions traditionnelles OL continuent à perdurer à cause de leur mal adaptation pour traité les volumes de données important, pourtant les bases de données classiques restent les plus utilisées sur le marché, car elle traite mieux tous les autres modèles de données.

Les solutions OC ne répondent que peu à ces besoins d'applications sur OLAP et OLTP. Ainsi, les architectures des systèmes d'information d'entreprise s'orientent alors sur une architecture hybride, intégrant un choix multiple de technologies permettant de répondre à l'ensemble des besoins de l'entreprise, que ce soit orienté OLTP ou OLAP.

Hormis les besoins d'expertise dans chaque type de solution, les problèmes de cohérence, de modélisation, de gestion de flux, de sécurité sont omniprésents dans ces situations.

Dans notre étude nous nous intéressons à apporter une solution bénéficiant à la fois des avantages des architectures OLTP et OLAP. Nous proposons une approche

permettant de séparer les colonnes pour faciliter l'agrégation, mais également capable de les reconstruire de manière efficace en profitant des propriétés inhérentes au stockage des données. Ainsi, sans avoir à passer par des vues matérialisées, des reconstructions multiples ou de mises à jour multiples d'index, nous apportons une solution capable de répondre aussi bien au transactionnel qu'aux agrégations.

L'architecture T-Plotter s'inspire des besoins des utilisateurs des bases de données classiques orientée lignes « OL » pour la manipulation, la modélisation et l'optimisation, mais également des structures de stockage des bases de données orientées colonnes « OC » pour optimiser les requêtes lourdes de type OLAP. Cette approche apporte une solution unique pour les problèmes d'un système d'information, sans avoir à changer la modélisation des données, ni à optimiser le stockage des données, ni la structure des requêtes d'interrogation.

Pour mieux comprendre la différence entre les modèles de bases de données nous allons essayer de comprendre le fonctionnement et l'architecture de chaque type de base de données.

IV -2. ARCHITECTURES DE BASES DE DONNEES :

Bien que plusieurs de modèles de bases de données BD ont émergé, Dans notre étude nous nous somme concentrés sur deux architecture classique orientée lignes, la plus utilisée sur le marché, et l'architecture orientée colonne la plus optimisée pour traiter les structure données de type OLAP et volumineuse.

La figure suivante illustre la différence entre les architectures de chaque modèle de bases de données, et l'impact des requêtes OLAP et LOTP sur les performances de chaque modèle de base de données.

IV- 2- A. ARCHITECTURES ORIENTÉES LIGNE:

Les bases de données traditionnelles optimisent le traitement de requêtes sur un mode de stockage "orienté-lignes" (OL). Tout tuple est stocké dans un bloc de données unique et ce qui permet d'effectuer des requêtes riches et complexes. Le but de ce type de système est de gérer un grand nombre de requêtes d'interrogation de type OLTP et des requêtes de mises à jour.

La principale problématique est ainsi de concilier aussi bien entre les accès pour interrogation que pour les mises à jour. Toutefois, l'évaluation de requêtes lourdes impliquant de gros volumes de données (i.e., de nombreux tuples) font chuter les

performances de la base de données, ceci est dû au grand nombre d'entrées/sorties effectué sur le disque (lecture et écritures temporaires).

L'optimisation de ce type de système repose essentiellement sur l'amélioration des accès aux données : organisation des données (clustered index, hachage), indexation (BTree, Bitmap, etc.). Cette étape essentielle permet de se concentrer sur les informations utiles, mais toutefois, ce n'est pas toujours une bonne solution. L'optimiseur s'attaque également à la réécriture de plans d'exécution, pour permettre de réduire les accès globaux via réduction d'ensembles, bénéficier d'effets cache (éviter l'accès successif à un même bloc), créer des prétraitements optimisés...

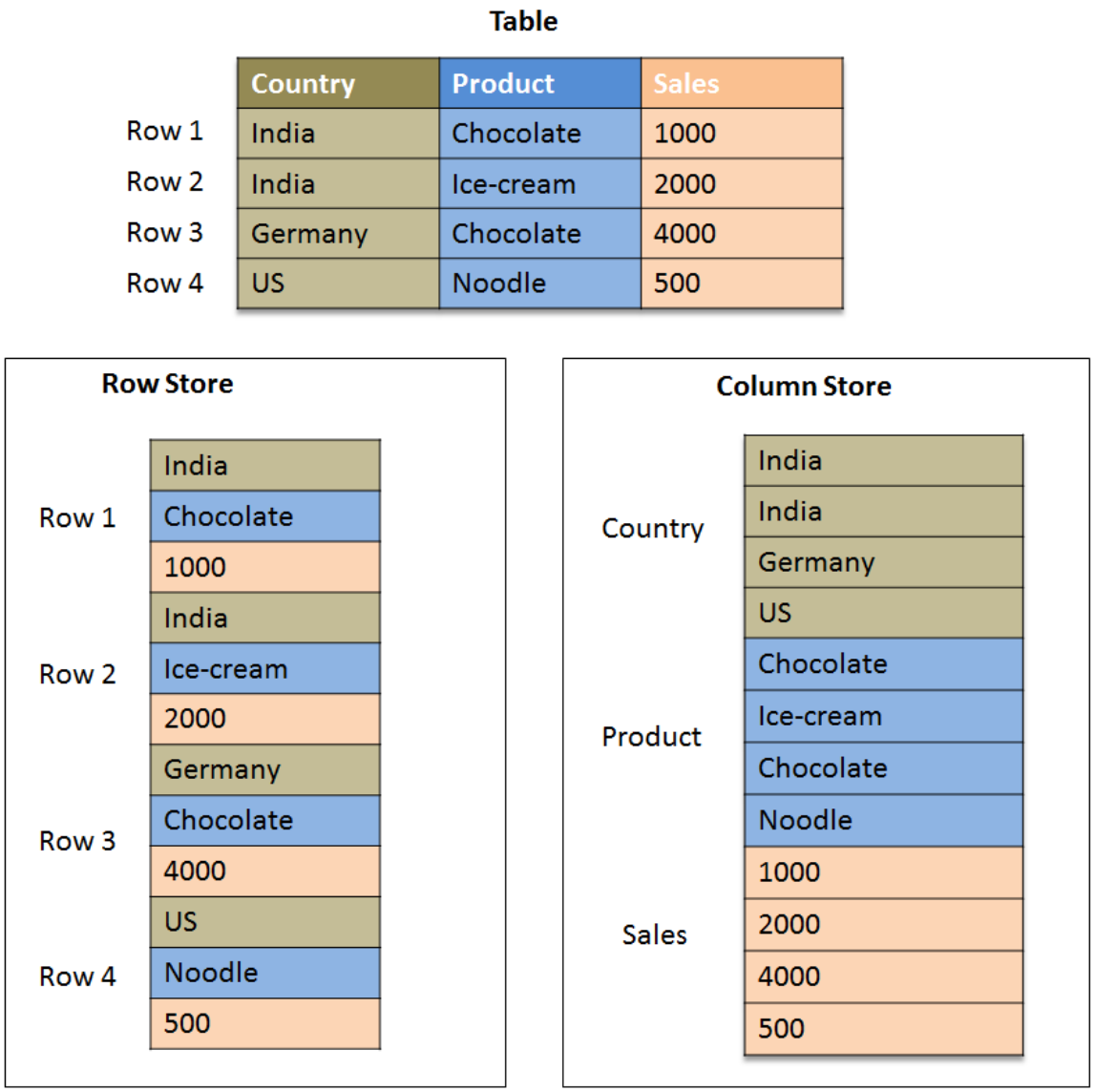


Figure 19 : Bases de données Orientée ligne et Orientée Colonne

D'un autre côté, le stockage orienté ligne a le désavantage non négligeable de forcer le traitement d'attributs parfois inutiles pour la requête. Même si l'utilisation de projections dans les opérateurs réduit la consommation mémoire, le nombre d'entrées/sorties générées pour la simple consultation d'un attribut coûte cher du fait de la forte distribution de cet attribut sur les blocs de données. C'est à cette fin que les solutions orientées colonnes (OC) proposent de regrouper les valeurs d'un attribut dans une structure homogène et ainsi de réduire les I/O.

Nous pouvons également citer la technique de vues matérialisées permettant de créer des résultats temporaires, pouvant être interrogées par la suite. Ces vues matérialisées permettent d'éviter de faire des calculs lourds. Toutefois, celles -ci posent le gros problème de fraicheur des données, de stockage, et d'optimisation de la mise à jour de la vue. De plus, une vue matérialisée ne répond qu'à un seul type de requête, il est alors nécessaire de produire une vue pour chaque type de requêtes et de multiplier les problèmes de mises à jour.

IV- 2- B. ARCHITECTURES ORIENTÉES COLONNE

Les bases de données orientées colonnes dites "OC" répondent mieux au problème des requêtes lourdes OLAP. Le but est d'optimiser le traitement d'un très grand nombre de tuples devant être manipulés et agrégés. L'idée du stockage OC vient du fait que tous les attributs d'un même n-uplet ne sont pas forcément utiles dans une requête. Ainsi, ces données inutiles sont tout de même accédées dans les BD classiques et alourdissent les opérateurs, malgré les projections. Les Bases de données OC répondent à cette problématique en fragmentant de manière verticale le schéma relationnel. Ces colonnes sont alors stockées dans des blocs distincts et optimisées pour pouvoir récupérer les informations d'un attribut.

Les avantages d'une telle approche sont doubles. Les requêtes ne faisant appel qu'à une seule colonne sont extrêmement efficaces, en particulier pour les agrégations. De plus, du fait d'un stockage sur un type d'attribut, cela réduit la redondance des valeurs et favorise le taux de compressions des données, la taille prise par les données et par conséquent, diminue les I/O, et le temps d'exécution des requêtes.

Ce type de solution pose toutefois le problème de reconstruction de tuples, l'opération qui a pour but de fusionner les résultats provenant de plusieurs colonnes afin de donner une seule réponse à une requête donnée. C'est pour cette raison que

ce type de solution est plus propice aux requêtes OLAP qu'OLTP. Les solutions MonetDB et C_Store sont deux approches distinctes sur la manière d'implémenter le stockage orienté colonnes, et sont considérées comme références dans le domaine. Leurs approches optimisent la reconstruction des tuples grâce à un concept commun : un ordonnancement unique pour toutes les colonnes. Chaque colonne conserve un ordre particulier (déterminé par le système) pour l'insertion des valeurs d'attribut. Cet ordre unique favorise la recherche de valeurs grâce à un positionnement facilement identifiable sur les blocs de données lors de la reconstruction de tuples.

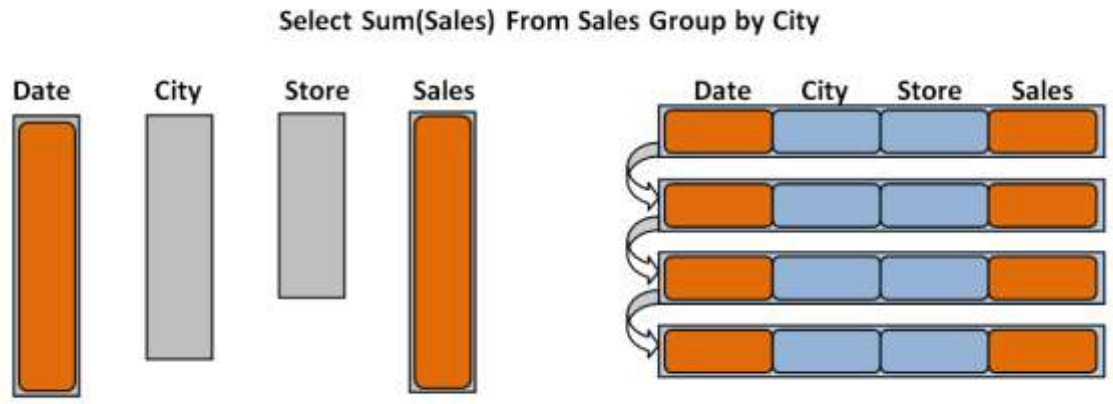


Figure 20 : exécution d'une requête OLAP.

La Figure 20 illustre le besoin de reconstruction des architectures OC. Supposons une requête contenant à la fois une projection et une sélection. La sélection extrait 3 tuple, tandis que la projection ne garde que les attributs B et C. Le stockage orienté lignes montre le fait qu'il est nécessaire d'extraire les attributs nécessaires au résultat, tout en récupérant les attributs A et D inutiles. Tandis que pour l'approche orientée colonnes, seuls les attributs nécessaires sont accédés, par contre, une reconstruction est ici nécessaire pour fusionner les attributs B et C, ceci grâce à leur position dans la colonne.

MonetDB et C_Store ref peuvent être considérés comme les pionniers des nouvelles générations OC. Nous allons détailler ici leur fonctionnement et les différentes techniques d'optimisation pour pallier aux problèmes de mises à jour et de reconstruction de tuple.

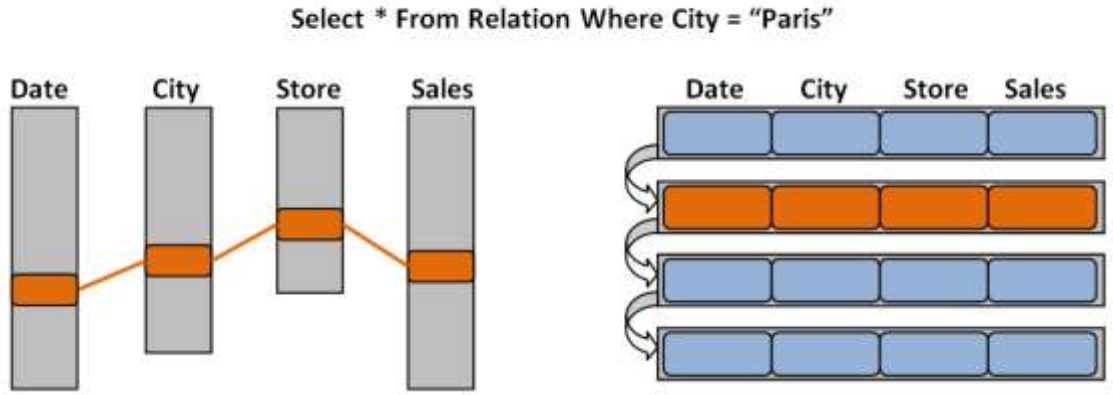


Figure 21 : Exécution d'une Requête OLTP.

MonetDB [25] est un SGBD open source orienté colonnes et In Memory développé chez « CWI database architectures research group » depuis 1993, MonetDB est désigné pour le traitement des masses de données orientées OLAP, et a prouvé de hautes performances dans le traitement des données et requêtes OLAP sur des volumes de données importants. Nous allons nous concentrer sur les techniques d'optimisation qui ont été proposées dans MonetDB :

Late Materialization [2,4]. Le fait que les données soient stockées en colonnes, les données de projection dans une requête sont accédées le plus tard possible (difficile voire impossible en orienté ligne). Ce qui réduit le coût d'exécution en n'allouant que les données nécessaires au traitement.

Fichiers BAT (Binary Association Table). Ce mode de stockage a pour but de reconstituer les tuples de manière efficace. Chaque colonne est alors stockée dans un fichier BAT associant le rang du tuple (OID), à sa valeur. Ce rang dans la table d'origine permet de recoller les morceaux de la table fragmentée. Pour ce faire, MonetDB a recourt à la jointure par hachage sur ce rang, ce qui peut être coûteux en ressources pour de gros volumes de données.

Cracking index [16]. Cet index adaptatif permet d'optimiser en mémoire les données accédées pour une colonne. À chaque interrogation de celle-ci, une réorganisation des colonnes est effectuée pour classer les données fréquemment accédées. A chaque colonne, un "index cracker" est associé optimisant toute requête fréquente sur des données identiques. Là aussi, les ressources sont importantes. MonetDB repose principalement sur le fait qu'une bonne partie de la base de données doit rester en mémoire pour être efficace. Le Cracking index reflète

clairement ce concept en tentant d'organiser les données fréquemment accédées en mémoire.

C_Store [5] est un SGBD orienté colonnes open source développé au MIT, et les universités de Boston, Brown et Brandeis. Le système a évolué jusqu'en 2006 pour devenir Vertica (version commerciale). Les concepts spécifiques à cette solution sont les suivants :

Une projection de C_Store est un ensemble de colonnes stockées dans un fichier unique (séparé des autres colonnes non prises en compte) et ordonné selon un des attributs de la projection. Cette projection peut également intégrer des attributs provenant d'une autre table, favorisant les opérations de jointure. Le but de la projection est de se focaliser sur les requêtes qui accèdent le plus fréquemment à la base de données et ainsi de ne matérialiser que les données nécessaires.

L'exemple ci-dessous illustre la création de quatre projections sur les tables Personne et Département. La première projection regroupe les attributs nom, prénom et age, la projection est alors organisé sur l'attribut "age" ("| Per.Age"). La seconde est ordonnée sur le bureau produit par la jointure entre les deux tables d'origine. La projection 3 s'oriente sur le numéro de département, tandis que la projection 4 sur le responsable.

Tables d'origine

Personne (id, Nom, Prenom, Age, Adresse, Dept_num)

Departement (Num , Section, Bureau, Responsable)

Projection1 (Per.Nom, Per.Prenom, Per.Age | Per.Age)

Projection2 (Per.Nom, Per.Prenom, Dept.bureau | Dept.bureau)

Projection3 (Per.id, Per.Adresse, Per.Dept_num | Per.Dept_Num)

Projection4 (Dept.num, Dept.Section, Dept.Responsable | Dept.Responsable)

Une projection C_Store peut être considérée comme une vue matérialisée; une projection d'attributs, voire via une jointure, dont le résultat est stocké avec une organisation optimisée. Cette technique a toutefois le désavantage de reposer essentiellement sur les requêtes, donc peut subir des problèmes d'évolutions des utilisateurs, de plus, le nombre de projections fait croître les problèmes de mises à jour associées.

C_Store utilise également les index de jointure pour reconstruire la table d'origine. Pour cela, il suffit de prendre deux projections (ou plus) dont l'union recouvre l'ensemble des attributs nécessaires. Cet index lie chaque rang de la première projection vers le rang de la seconde projection (organisation différente). Toutefois, dans le cas où la reconstruction nécessiterait M projections pour reconstruire la table, cela produirait M-1 indexes de jointures. Pour finir, il est important de rappeler que ces index de jointure sont unidirectionnels et imposent donc un ordre de reconstruction. L'exemple de la table 1 peut ainsi reproduire les deux tables d'origine en combinant les projections 1, 2 et 3 pour la table Personne, et les projections 2 et 4 pour la table Département. Les index de jointure utilisés devront préciser l'ordre dans lesquels les jointures seront effectuées.

Les stockages RS & WS ont pour but de favoriser les lectures et écritures en les découplant dans deux stockages distincts. Ces fichiers contiennent les projections et les index de jointure et dupliquent chacune dans ces deux structures.

WS (Write Store) est une structure de données parallèle à RS dédiée à l'écriture des données, WS regroupe les mêmes projetions présentent sur RS, et ces projections sont rangées dans le même ordre que celle de RS, sauf que les projections de WS ne sont pas compressées et les données y sont stockées explicitement, aussi le numéro de tuple est stocké physiquement pour chaque données de la projection, WS est optimisé et destinée aux écritures et aux mises à jour.

RS (Read Store) est une structure de données optimisée essentiellement pour la lecture de données, RS regroupe les mêmes projections présentent dans WS, ces projections sont rangées dans le même ordre que ceux de WS, les données dans WS sont compressés selon la techniques RLE (Run-length encoding – RLE [46]), et les numéros de tuples ne sont pas stockés physiquement dans les projections, ce qui réduit les nombre de I/O pour une lecture de données, et favorise la balayage rapide des données.

Toute mise à jour d'une projection doit être effectuée en premier lieu sur RS, puis impactée dans WS tout en utilisant l'organisation des données de la projection ; Bien entendu, C_Store use également du principe de late materialization pour pouvoir réduire le coût d'allocation de ressources lors de l'exécution de requêtes.

IV -3. APPROCHE PROPOSÉ E T-PLOTTER

T-Plotter est l'abréviation de tuple plotter ou tuple dessinateur; reflétant la notion d'indexation multiple de tuples sur un fragment séparé, aidant à reconstruire des tuples stockés dans un fragment séparé, cette solution répond aux besoins des bases de données OLAP et des requêtes orientées OLAP, applicables sur une architecture fragmentée verticalement, cette technique pouvant être appliquée non seulement pour les bases de données classiques, mais peut être appliqué sur différentes bases de données de conception physique.

IV- 3- A. CONCEPT DE T-PLOTTER.

Notre approche consiste à ajouter une couche sur une table fragmentée verticalement. Pour chaque fragment (sous-table), nous ajoutons une colonne avec le numéro d'attribut faisant référence au numéro d'enregistrement Tuple_id permettant la reconstruction de tuple en utilisant l'opération de jointure naturelle sur la correspondance de tuple_id, où tuple_id reflète le rang du tuple dans la table logique.

T-Plotter offre la possibilité d'accéder rapidement aux données des différents fragments triés dans le même ordre ou dans un ordre différent, ce qui réduit considérablement le coût d'exécution des requêtes, contrairement à une reconstruction classique qui est très coûteuse en termes de traitement de la CPU et de nombre d'entrées-sorties , mais avec notre approche, il suffit de parcourir T-Plotter en fonction de la liste des tuple_id pertinents, cela suffit pour obtenir directement les données stockées sur plusieurs fragments, et l'opération de reconstruction des tuples est terminée, sans opération de jointure, ni d'accès avec autres index.

IV- 3- B. ARCHITECTURE DE T-PLOTTER:

T-Plotter repose sur une architecture très simple, un regroupement d'adresses physique qui garantissent la possibilité un accès rapide basé sur l'identificateur tuple_id aux données des différents fragments triés dans un ordre différent, ce qui réduit considérablement le coût d'exécution des requêtes et permet à un grand nombre de données de répondre au résultat de la requête qui ne peut être conservé en mémoire.

Dans une fragmentation verticale classique et naïve on aura recourt a l'utilisation de plusieurs index pour récupérer les données en fonction du fragment et de l'opération de jointure par hachage, etc. ; et a la fin il est nécessaire de reconstruire le

résultat final, qui est très coûteux termes de traitement CPU et nombre d'entrées sorties, mais avec notre approche, il suffit de parcourir T-Plotter selon la liste des tuple_id pertinents, cela suffit, et nous obtenons directement les données stockées sur plusieurs fragments, et l'opération de reconstruction est terminée.

T-Plotter est une technique flexible et dynamique, elle peut être appliquée à différents types de conceptions physiques, mais T-Plotter ne restreint pas l'accès aux attributs physiquement stockés séparément, car les attributs de T-Plotter permettent de référencer différents supports de stockage. Les structures, dont nous citons des vues matérialisées, des tableaux, des fragments stockés sur une structure distribuée, des projections C-Store, ouvrant les perspectives à d'autres implémentations et à d'autres applications, et ne réduisant pas leur application à des structures spécifiques, nous pensons que cette fonctionnalité favorisera et facilitera l'intégration de T-Plotter à différents SGBD.

Ces fragments sont généralement des sous-tables d'une table donnée; Cependant, nous pouvons imaginer une composition entre des fragments de divers tableaux, sources et types.

Pour comprendre le T-Plotter en profondeur, nous expliquerons comment ce dernier est structuré, comment il peut être utilisé pour reconstruire des n-uplets à la demande et comment interroger des données à ce sujet.

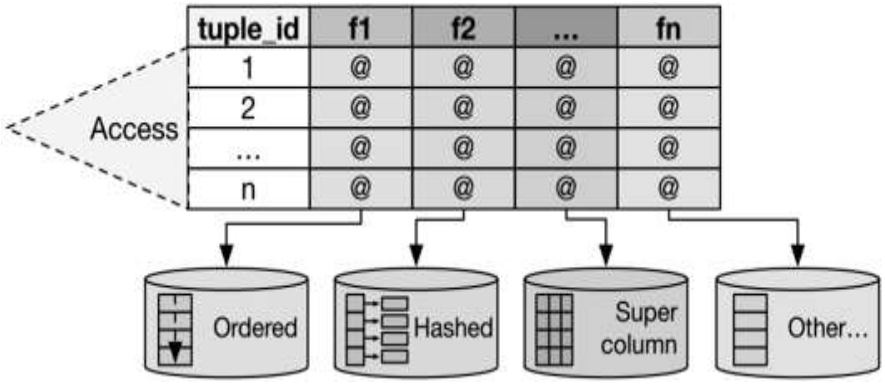


Figure 22 - T-Plotter structure de données.

IV- 3- c. STRUCTURE DE T-PLOTTER

La structure de T-Plotter est représentée à la Figure 22. T-Plotter est une structure de données ayant pour fonction de lier les différentes sources de données et de reconstruire des lignes ou des tuples à partir des sources de données requises. La structure physique de T_Plotter est présentée comme suit: il s'agit d'un tableau composé de plusieurs attributs (tuple_id, @ 1, @ 2 ... @n); où l'attribut tuple_id représente le rang de l'enregistrement dans le fragment et correspond à son rang d'origine dans la table avant sa fragmentation. Les attributs @V représentent des attributs de pointeur sur des adresses physiques souvent appelées ROWID, ce qui permet un accès direct et rapide à un attribut du fragment souhaité, de la même manière qu'un accès d'index.
T-Plotter présente plusieurs avantages:

• Accès aux données à la demande. Il permet une matérialisation tardive et une compression des données.

• Chaque fragment est indépendant l'un de l'autre. Des organisations physiques spécifiques peuvent être définies pour chaque fragment, puis optimiser les traitements (ordonné, haché, super-colonnes, etc.).

• Le résultat obtenu en accédant à T-Plotter représente directement un tuple reconstruit.

Rowid est une structure de données spécifique qui stocke un pointeur logique sur une position de tuple dans un fichier de données. Il permet à la base de données d'accéder directement à un enregistrement donné, ce qui constitue le moyen le plus rapide de le faire (sauf la mise en cache).

Nous détaillerons dans la suite la base de la reconstruction des tuples, ce qui conduit à des gains substantiels en traitement de requête, en particulier pour une projection multi-colonnes qui devrait nécessiter plusieurs opérations de jointure.

Pour construire T-Plotter, une seule requête est requise:

```
CREATE TABLE TP AS
SELECT tuple_id, F1.rowid AS f1, F2.rowid as f2,
F3.rowid AS f3, F4.rowid as f4
FROM F1, F2, F3, F4
WHERE F1.tuple_id = F2.tuple_id
AND F1.tuple_id = F3.tuple_id
AND F1.tuple_id = F4.tuple_id.
```

IV- 3- D. MATÉRIALISATION DE T-PLOTTER

Notre stratégie d'accès aux fragments repose sur des tuples de T-Plotter stockés dans une structure unique. On accède à chaque prédicat filtre pour réduire cet ensemble de tuples de T-Plotter. Et en cas d'accès multiples, les identifiants de ligne peuvent être remplacés par les valeurs de filtrage (et les attributs liés du fragment), cette caractéristique correspond à la matérialisation précoce [4]; Lorsque tous les prédicats de filtrage sont traités, nous accédons directement (via des ID de ligne connexes) aux attributs projetés afin de compléter les attributs requis, correspondant à la matérialisation tardive. La Figure 23 donne une vue globale du processus de matérialisation du T-Plotter.

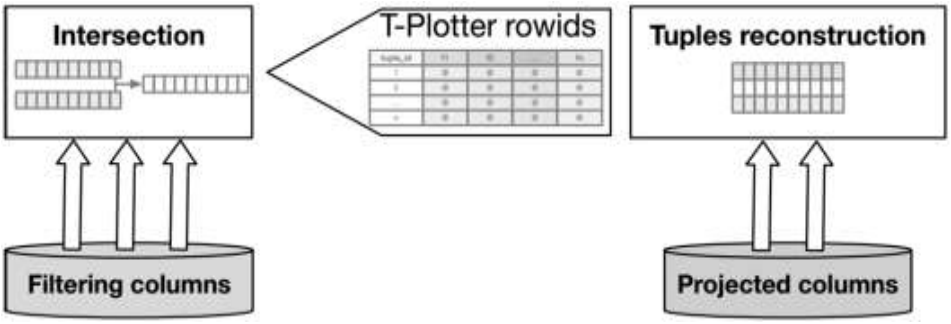


Figure 23. T-Plotter traitement de requête

Notez que ce processus ne nécessite aucun ordre ou tri commun sur les fragments, ce qui est un facteur positif pour la compression. De plus, cette stratégie bénéficie à la fois de la matérialisation précoce et tardive, en fonction des fragments requis de la requête. Ainsi, nous réduisons la quantité de données à conserver dans la mémoire principale pour calculer une requête en ne conservant que les attributs obligatoires, mais également en conservant les attributs ré-accédés (au lieu de combiner des bitmaps) [44];

Pour obtenir efficacement les tuples de T-Plotter, la structure est indexée afin d'optimiser les accès sur tuple_id (attribut commun). La structure de T-Plotter est également conservée en mémoire principale car elle est requise pour chaque opération, ainsi, un seul index doit être mis en cache de manière permanente pour rendre le système efficace pour chaque plan d'exécution.

L'Algorithme 3: T-Plotter détaille le processus de reconstruction des tuples. Nous pouvons voir que tous les prédicats filtre sont traités en première étape (lignes 2 à 16)

et les projections à la deuxième étape (lignes 17 à 24). Pour filtrer les tuples de T-Plotter, nous devons vérifier si les attributs requis sont déjà matérialisés (ligne 3), sinon, un accès à ce prédicat est effectué (ligne 4) en fonction de l'organisation sous-jacente et des statistiques (index clustérisé, table de hachage, B-Tree ou accès complet si nécessaire).

Ensuite, si c'est le premier accès à T-Plotter, nous obtenons toute instance qui correspond au premier filtre (ligne 4 à 9) en fusionnant les tuples utilisés avec T-Plotter (via tuple_id), sinon nous produisons une intersection entre le résultat des étapes précédentes et le nouveau filtre (ligne 10) grâce aux valeurs de tuple_id. Si les attributs requis sont déjà matérialisés (grâce à la fragmentation), nous pouvons accéder au filtre directement dans la mémoire principale (ligne 13).

```
Input:
-    ℱ set filtering predicates (ordered by selectivity)
-    𝒫 set of projected attributes
-    𝒯 set of T-Plotter tuples
-    Output: Result set
-    𝒯 := ϕ
1.   For each f in ℱ do
2.        if !materialized(𝒯, f.attributes) then
3.             S := access(f)
4.             if 𝒯 = ϕ then
5.                  for each s in S do
6.                       t := s ⋈ TP(s.tuple_id)
7.                       𝒯 := 𝒯 ∪ t
8.                  end
9.             else
10.                 𝒯 := 𝒯 ∩ S
11.            end
12.       else
13.            filter(𝒯, f)
14.       end
15.  For each p in 𝒫 do
16.       if !materialized(𝒯, p) then
17.            for each t in 𝒯 do
18.                 m := access(rowid(t, p))
19.                 t := t ⋈ m
20.            end
21.       end
22.  return t
23.  End.
```

Algorithme 3: **T-Plotter matérialisation**

Pour matérialiser les attributs nécessaires à la projection, nous devons vérifier si ces attributs ont déjà été matérialisés (ligne 16). Sinon, nous accédons aux fragments

correspondants en utilisant les ID de ligne présents dans les tuples de T-Plotter (ligne 18), puis une jointure de fusion entre eux est appliquée (ligne 19).

Nous devons noter que l'étape de matérialisation bénéficie de l'effet de facteur de clustering. En fait, les lignes de calcul des tuples T-Plotter ont une forte probabilité de pointer sur des blocs de données continus et, par conséquent, peu d'accès à ces blocs sont nécessaires car ils sont déjà disponibles en mémoire principale.

Pour maximiser cette étape, un tri sur les sous-ensembles de Rowid est appliqué (ce sous-ensemble peut tenir en mémoire).

Puisque la matérialisation tardive peut être insuffisante dans le traitement de prédicats a faible sélectivité, nous devons l'écarter. Un algorithme alternatif est ensuite appliqué où l'étape 1 est remplacée par un accès global à T-Plotter.

Le prédicat est appliqué lors de la matérialisation initiale en accédant aux fragments requis (deuxième étape). Nous montrerons le gain de cet algorithme alternatif dans les expériences.

IV- 3- E. T-PLOTTER TUPLE RECONSTRUCTION

T-Plotter Effectue une reconstruction de tuple d'une manière très simple, cette reconstruction peut être basée sur des fragments non nécessairement ordonnés sur la même clé, un simple accès d'index à la structure de T-Plotter permet d'obtenir l'ensemble des n-uplets reconstruit, T-Plotter est stocké sur la clé primaire tuple_id qui permet un accès indexé et direct aux données pertinentes sans recourir à plusieurs index, une fois que les positions de tuples pertinentes sont obtenues, il suffit d'accédez simplement à T-Plotter pour récupérer les adresses de données stockées sur les autres fragments, le résultat obtenu est un tuple reconstruit, sans autre opération nécessaire.

T-Plotter prend en charge la matérialisation précoce et la matérialisation tardive, sur des structures de données non nécessairement organisées sur la même clé de tri, ainsi que sur des données compressées et des données non compressées.

Considéré comme le meilleur moyen d'effectuer des requêtes OLAP, nous expliquons ci-après le processus de matérialisation tardive sur T-Plotter:

1. Au début, apporte uniquement les attributs cités dans la partie prédicats, par exemple: (A, B, C)

2. Filtrez (A, B, C) sur les prédicats pour obtenir des fichiers bitmap (A ', B', C ') indiquant les positions des tuples pertinents,

3. Construisez le fichier bitmap Bmp1 en vous basant sur l'opération (A '^ B' ^ C ') pour obtenir un fichier bitmap unique Bmp1, note; Si les attributs (A, B, C) ne sont pas triés dans le même ordre, il sera nécessaire de trier les résultats (A', B', C ') dans le même ordre pour obtenir un résultat correct.

4. Utilisez la structure T-Plotter pour accéder aux tuples d'attributs pertinents (D, E) en fonction des positions de Bmp1 et obtenir (D', E') et obtenir le résultat final.

IV- 3- F. ANALYSE MODÈLE DE COÛTS

Pour calculer ou estimer le temps de traitement d'une requête, il est nécessaire de détailler chaque opération du plan d'exécution de la requête. cette opération s'appelle le calcul du modèle de coût; L'analyse du modèle de coût repose essentiellement sur le calcul du nombre d'accès au disque entrants et sortants, car c'est l'opération la plus chère par rapport aux autres opérations comme le coût du traitement du processeur ou l'accès de coût à la mémoire principale; Afin d'étudier l'impact de T-plotter «TP» sur le fonctionnement des jointures, nous nous concentrerons sur deux conceptions physiques importantes: la fragmentation verticale classique «VF» et T_Plotter T «TP», en référence aux modèles de coûts que nous pouvons comparer les performance de TP avec la fragmentation verticale classique; Pour l'optimisation, nous avons limité cette comparaison au coût d'accès d'un seul tableau fragmenté verticalement et nous avons concentré notre étude sur la partie jointe pour la reconstruction de tuples, qui représente la principale différence entre FV et TP et le gain de performance souhaité obtenu par TP.

Notations:

- Rtp: le fragment de T-Plotter.
- $|R|$: nombre de blocs dans la relation R,
- $\|R\|$: nombre de tuples dans R, le nombre de tuples est identique pour tous les fragments ainsi $\|R\_i\| = \|R\_tp\|$,
- $\|R'\|$: nombre de tuples pertinents dans R, de tel façon $\|R'\| = Sel*\|R\|$,
- $|Index|$: hauteur d'index,
- $0 \leq Sel \leq 1$: sélectivité de la requête.

Hauteur d'index: les index sont basés sur les attributs rang, de sorte que les index de chaque fragment sont de taille identique et que chaque index est de type B-Tree, l'attribut rang est un entier de taille 4 octets.

$Index_i$: représente l'index basé sur le fragment I, dans notre cas la principale table Lineitem est tous les fragments verticaux dérivés de Lineitem contiennent $6*10^6$ tuple, la hauteur de l'index référençant une table de la même taille est

$$\text{Height index}_i = \log_{\text{ordre } i}(\|R_i\|) = \log_{200}(6 \times 10^6) = 2.9457 \approx 3 ,$$

Résultat, pour accéder à une donnée de chaque fragment à l'aide de l'index, nous devons lire trois blocs,

Nous calculons le modèle de coût pour la partie de reconstruction de tuples, où {R_1,…, R_n}, représente le fragment nécessaire à l'attribut de tri de la requête.

Conception VF: est calculée en fonction du coût d'accès aux fragments et de leurs index;

$$\begin{aligned} \boldsymbol{Cost_{VF}} &= \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left((|\boldsymbol{Index_i}| \times \boldsymbol{Sel} \times \|\boldsymbol{R_i}\|) + \|\boldsymbol{R'_i}\| \right) \\ &= \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left((\boldsymbol{3} \times \boldsymbol{Sel} \times \|\boldsymbol{R_i}\|) + \|\boldsymbol{R'_i}\| \right) \\ &= \textstyle\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{3} \times \boldsymbol{Sel} \times \|\boldsymbol{R_i}\|) + \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \|\boldsymbol{R'_i}\| \quad \textbf{Équation 13} \end{aligned}$$

Conception TP: Pour calculer le modèle de coût du TP, nous distinguons deux cas, le premier cas est lorsque la sélectivité est très faible, avec un accès a TP par un balayage complet « full scan », le second cas lorsque la sélectivité est élevée, ici il est plus intéressant d'utiliser un index pour accéder a TP.

La sélectivité faible est vérifiée lorsque: Sel> |R_tp| /‖R_tp‖

Dans ce cas, le coût est composé d'un accès scan complet de TP, plus l'accès aux fragments d'autres tables Ri.

$$Cost_{TP} = |R_{tp}| + \sum_{i=0}^{n} (\|R'_i\|)$$

Une sélectivité élevée est vérifiée pour: Sel <|R_tp| /‖R_tp‖

Ici, le coût est composé du coût d'un balayage indexé sur TP, plus l'accès à d'autres fragments.

$$\begin{aligned} Cost_{TP} &= \left(|Index_{tp}| \times Sel \times \|R_{tp}\| \right) + \|R'_{tp}\| + \sum_{i=0}^{n} (\|R'_i\|) \\ &= \left(\boldsymbol{3} \times \boldsymbol{Sel} \times \|\boldsymbol{R_{tp}}\| \right) + \|\boldsymbol{R'_{tp}}\| + \textstyle\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\|\boldsymbol{R'_i}\|) \quad \textbf{Équation 14} \end{aligned}$$

Pour comparer les deux conceptions, nous étudions le comportement de l'équation CostVF - CostTP, lorsque CostVF - CostTP > 0, cela signifie que le TP est moins coûteux en temps que VF et lorsque la formule est négative CostVF - CostTP < 0, le design VF sera le meilleur choix:

- Sélectivité faible: $\boldsymbol{Sel} > (|\boldsymbol{R\_tp}| / \|\boldsymbol{R\_tp}\|)$

$$Cost_{VF} - Cost_{TP} = \left(\sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) + \sum_{i=0}^{n}\|R'_i\|\right) - \left(|R_{tp}| + \sum_{i=1}^{n}(\|R'_i\|)\right)$$
$$= \sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) + \sum_{i=0}^{n}(\|R'_i\|) - |R_{tp}| - \sum_{i=1}^{n}(\|R'_i\|)$$
$$= \sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) - |R_{tp}|.$$

- Sélectivité forte: $\boldsymbol{Sel} < (|\boldsymbol{R\_tp}| / \|\boldsymbol{R\_tp}\|)$

$$\boldsymbol{Cost_{VF} - Cost_{TP} = \left(\sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) + \sum_{i=0}^{n}\|R'_i\|\right) - \left(\left(Sel \times \|R_{tp}\| \times 3\right) + \|R'_{tp}\| + \sum_{i=1}^{n}(\|R'_i\|)\right) = \sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) + \sum_{i=0}^{n}(\|R'_i\|) - \left(Sel \times \|R_{tp}\| \times 3\right) - \|R'_{tp}\| - \sum_{i=1}^{n}(\|R'_i\|) = \sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) + \sum_{i=0}^{n}(\|R'_i\|) - (Sel \times \|R_i\| \times 3) - \|R'_{tp}\| - \sum_{i=1}^{n}(\|R'_i\|) = \sum_{i=1}^{n-1}(3 \times Sel \times \|R_i\|) - \|R'_{tp}\|.}$$ **Équation 15**

Les principales variables influentes dans ces équations sont le nombre de fragments "n" et la sélectivité de la requête "*Sel*", ainsi :

- *Sélectivité faible:*

$$Sel > \left(|R_{tp}| / \|R_{tp}\|\right) : Cost_{VF} - Cost_{TP} > 0 \implies$$
$$\implies \sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) - |R_{tp}| > 0 \Rightarrow \sum_{i=1}^{n}(3 \times Sel \times \|R_i\|) > |R_{tp}|$$
$$\Rightarrow Sel > \frac{|R_{tp}|}{\sum_{i=1}^{n}(3 \times \|R_i\|)} \Rightarrow Sel > \frac{|R_{tp}|}{3n \times \|R_i\|}$$
$$\Rightarrow \boldsymbol{Sel > \frac{|R_{tp}|}{3n \times \|R_{tp}\|}}. \quad \textbf{Équation 16}$$

Et on a : $\forall n \geq 1 : \left(|R_{tp}| / \|R_{tp}\|\right) > \left(|R_{tp}| / (3n \times \|R_{tp}\|)\right)$

Donc, l'

$$\Rightarrow \boldsymbol{Sel > \frac{|R_{tp}|}{3n \times \|R_{tp}\|}}. \quad \textbf{Équation 16}$$

L'équation 16 est toujours vérifiée, cela signifie que la conception TP est toujours meilleure que la conception de VF pour une faible sélectivité.

- *Sélectivité forte:*

$$0 \leq Sel < {|R_{tp}|}/{\|R_{tp}\|} : Cost_{VF} - Cost_{TP} > 0 \Rightarrow \sum_{i=0}^{n-1}(3 \times Sel \times \|R_i\|) - \|R'_{tp}\| > 0$$

$$\Rightarrow \sum_{i=0}^{n-1}(3 \times Sel \times \|R_i\|) > \|R'_{tp}\| \Rightarrow (n-1) \times 3 \times Sel \times \|R_i\| > \|R'_{tp}\|$$

$$\Rightarrow (n-1) \times 3 \times Sel \times \|R_{tp}\| > \|R'_{tp}\| \Rightarrow Sel > {\|R'_{tp}\|}/{(n-1) \times 3 \times \|R_{tp}\|}$$

$$\Rightarrow Sel > {Sel \times \|R_{tp}\|}/{\|R_{tp}\| \times (n-1) \times 3} \Rightarrow 1 > {1}/{(n-1) \times 3}$$

$$\forall n > 1 \Rightarrow 1 > {1}/{(n-1) \times 3}$$

$$\Rightarrow \boldsymbol{Cost_{VF} - Cost_{TP} > 0 \Rightarrow Cost_{VF} > Cost_{TP}}. \quad \textbf{Équation 17}$$

On déduit de l'$\Rightarrow \boldsymbol{Cost_{VF} - Cost_{TP} > 0 \Rightarrow Cost_{VF} > Cost_{TP}}$. Équation 17 est toujours vérifiée, cela signifie pour une forte sélectivité la conception TP est toujours meilleure que la conception de VF.

Conclusion: sur la base du calcul théorique du modèle de coût, nous en déduisons que le T-Plotter est plus efficace que la fragmentation verticale classique, que ce soit pour une sélectivité faible ou élevée.

IV -4. IMPLEMENTATION

Pour mettre en pratique notre solution, nous avons choisi le benchmark bien connu TPCH, qui propose des tests destiné aux modèle OLAP. Il est composé d'une suite de 22 requêtes ad-hoc destinées aux entreprises [50], disponible sur l'appendice_2. Composé de huit tables: part, partsupp, lineitem, orders, customer, supplier, nation, et region. Le benchmark TPCH est conçu pour insister sur les performances du SGBDR et du système. Les 22 requêtes reflètent des implémentations spécifiques et une interrogation relative combinant des agrégations, des jointures et des opérations de tri, avec une sélectivité variable. La table de faits Lineitem contient 13 colonnes et est interrogée par 16 des 22 requêtes. Pour mettre en valeur les performances du concept T-Plotter, nous nous concentrerons uniquement sur ces requêtes. Puisque Lineitem est la plus importante des relations et tables du schéma (grand volume d'attributs). il est clair que cette optimisation peut être mise en pratique sur chaque table ou même d'avantage sur différents types d'architecture. Afin de produire des plans d'exécution efficaces basés sur des sous-tables, on doit décomposer Lineitem en fragments, pour y parvenir, nous nous appuyons sur l'algorithme Afinner [19] qui propose un bon compromis entre une décomposition

complète et une reconstruction des tuples. Pour cela, il calcule un ensemble de requêtes fréquentes. Dans notre cas, nous avons utilisé les 16 requêtes TPCH.

Le Tableau 15 montre la fragmentation verticale (VF) de Lineitem en 7 fragments distincts. Nous pouvons voir que l'attribut tuple_id est présent dans chaque fragment afin de reconstruire des tuples. De plus, à l'exception de tuple_id, aucun attribut n'est dupliqué dans les fragments.

Fragments	Attributes
F1	Tuple_id, L_ORDERKEY, L_PARTKEY, L_SUPKEY
F2	Tuple_id, L_LINENUMBER, L_COMMENT
F3	Tuple_id, L_QUANTITY
F4	Tuple_id, L_EXTENDPRICE, L_DISCOUNT
F5	Tuple_id, L_TAX, L_RETURNFLAG, L_LINESTATUS
F6	Tuple_id, L_COMMITDATE, L_RECIEPTDATE, L_SHIPINSTRUCT, L_SHIPMODE.
F7	Tuple_id, L_SHIPDATE

Tableau 15: Fragmentation vertical de Lineitem.

Pour mieux faire, des index sur tuple_id ont été choisis afin d'améliorer le facteur de clustering pour la reconstruction de n-uplets. Bien entendu, d'autres stockages physiques auraient dû être choisis en fonction d'un ensemble de requêtes différent.

IV- 4- A. PROCESSUS DE REECRITURE DE REQUETES

En raison de la fragmentation de Lineitem, les requêtes TPCH doivent être réécrites manuellement pour une évaluation correcte en fonction de la nouvelle conception physique. Pour chaque attribut de la requête, le fragment associé est ajouté dans la clause FROM. Cette tâche simple nécessite, dans les étapes suivantes, de s'en tenir à l'Algorithme 3. Pour réaliser la reconstruction des n-uplets avec T-Plotter; Les jointures en boucle imbriquées sont traitées entre les fragments et la structure de T-Plotter. Grâce à ce dernier balayage, les tuples nécessaire sont déjà disponibles et aucun balayage d'index n'est nécessaire pour chaque tuple (c'est-à-dire moins d'E/S).

Pour illustrer l'étape de réécriture, nous avons choisi de nous concentrer sur deux requêtes TPCH: Q1 et Q14. Elles permettent de mettre en lumière deux types de comportement distincts sur le plan d'exécution de T-Plotter.

La requête Q1 a deux caractéristiques principales. Il ne traite qu'un prédicat avec une faible sélectivité de 56,86%, mais doit également projeter plusieurs attributs avant de procéder à l'agrégation. Ces caractéristiques sont présentées dans le Tableau

15, qui indique pour chaque requête le nombre de fragments requis pour filtrer et projeter les attributs du projet, mais également la sélectivité des prédicats.

Par conséquent, nous pouvons voir que la requête Q1 ne nécessite qu'un seul fragment (F1) pour filtrer la structure T-Plotter, et trois fragments (F3, F4, F5) pour la projection. Comme indiqué à la section 3.3, la matérialisation de T-Plotter se comporte différemment avec des prédicats de sélectivité médiocres. La requête Q1 doit ensuite être réécrite comme si elle ne contenait aucun prédicat, mais uniquement des projections. En fait, la requête réécrite suivante utilise T-Plotter pour accéder aux projections (F3, F4, F5), mais aussi aux filtres (F1) avec rowid. Les Hint seront détaillés dans ce qui suit.

La requête Q14 contient un fragment (F1) pour le filtrage et un autre pour la projection (F5). L'attribut de jointure avec la table des pièces est déjà contenu dans F1, avec une sélectivité de 0,013%, nous pouvons appliquer l'Algorithme 3 (comme pour les 11 autres requêtes). Ainsi, nous accédons au fragment F1 et obtenons la valeur tuple_id pour obtenir les tuples de T-Plotter dans TP. Ensuite, on accède à F5 lorsque cela est nécessaire pour appliquer la matérialisation tardive.

TPCH: Q14:

```
SELECT 100.00 * sum(case when p_type like 'PROMO%'
then l_extendedprice * (1 - l_discount) else 0 end)
/ sum(l_extendedprice * (1 - l_discount)) as promo_revenue
FROM Lineitem, Part
WHERE l_partkey = p_partkey AND
l_shipdate>= '1995-08-01' AND l_shipdate< '1995-09-01';
```

TPCH: Q14 Réécrite:

```
SELECT /*+index(TP) leading(f1 TP f5 part) use_nl(f5 TP)*/
100.00 * sum(case when p_type like 'PROMO%'
then l_extendedprice * (1 - l_discount) else 0 end)
/ sum(l_extendedprice * (1 - l_discount)) as promo_revenue
FROM F1, F5, TP, Part
WHERE F1.tuple_id = TP.tuple_id AND TP.f5 = F5.rowid AND
l_shipdate>= '1995-08-01' AND l_shipdate< '1995-09-01'
AND l_partkey = p_partkey;
```

IV.4.A.1. PLAN D'EXÉCUTION:

Enfin, des HINT doivent être ajoutées à la requête pour que l'optimiseur s'exécute correctement. En fait, l'optimiseur ne sait pas comment traiter la structure de T-Plotter de manière naturelle avec une sélection de plusieurs Rowid et l'ordre approprié pour traiter les fragments. En réalité, la plupart du temps, le plan d'exécution proposé donne des résultats faibles. En introduisant des astuces, le plan d'exécution produit correspond aux exigences de l'Algorithme 3.

La requête Q1 utilise le Hint use_nl pour accéder aux fragments avec un NESTED LOOPS avec un ordre spécifique; TP pour l'initialisation de T-Plotter, F1 pour le filtrer (intersection) avec une faible sélectivité, puis d'autres fragments pour la matérialisation tardive.

Le plan d'exécution résultant de la Figure 24 pour la requête Q1 montre que les boucles NESTED sont utilisées avec l'ordre requis. On peut noter que les accès aux fragments se font avec le Hint TABLE ACCESS BY USER ROWID donné par la table TP. Le plan d'exécution se comporte de manière classique pour que les fonctions d'agrégat soient traitées à la fin de la requête.

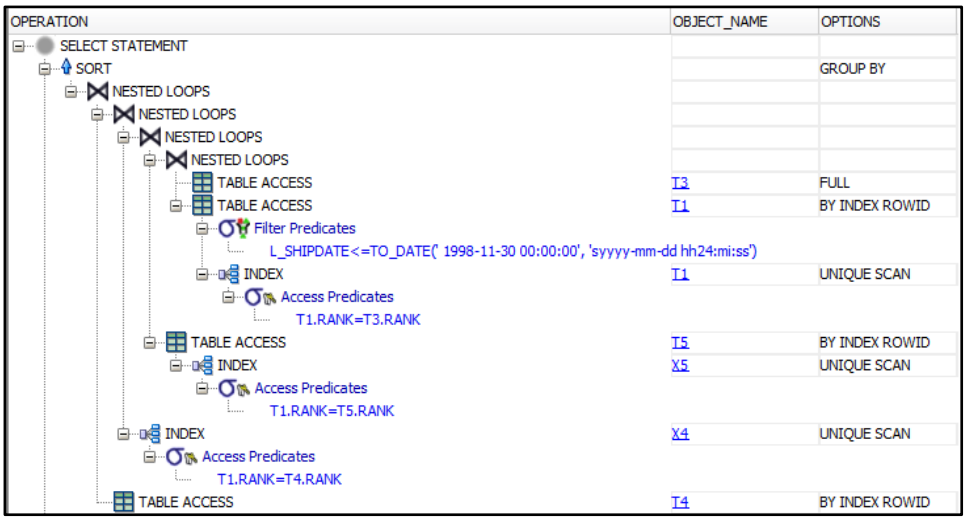


Figure 24 - T-Plotter plan d'exécution Q1

La requête Q14 force l'utilisation de l'index sur TP afin de s'assurer que le tuple_id est correctement utilisé pour l'étape de filtrage. Ensuite, un ordre précis est donné pour le processus; F1 pour obtenir les tuples de filtrage, TP pour obtenir les tuples de T-Plotter correspondants, F5 pour les attributs de projet, puis une partie pour l'opération de jointure requise. Pour finir, un indicateur Use_nl est utilisé pour imposer l'accès par Rowid's pour TP et F5. Aucune astuce n'est donnée pour traiter la jointure finale avec Part.

La Figure 25 illustre le plan d'exécution de la requête Q14. Nous pouvons suivre le processus de l'algorithme pour le fragment de filtrage et la matérialisation tardive de F5. Nous pouvons noter que F1 est accessible avec un index standard; il peut être échangé par un autre type d'index en fonction des caractéristiques du prédicat impliqué (par exemple, un index clustérisé ou une table de hachage).

Figure 25. T-Plotter Plan d'exécution Q14

IV- 4- B. EXPÉRIENCES:

L'index T-Plotter a été intégré à une base de données Oracle 11gR2 dans laquelle nous avons simulé le comportement d'une fragmentation verticale. Afin de montrer l'efficacité de notre système, nous l'avons testé dans un environnement contraint avec une mémoire principale de 4 Go seulement (partagée avec le système d'exploitation) et un processeur Intel i5 2430M à 2,4 GHz. Un disque dur classique à 7 200 tr / min a été utilisé. Ce paramètre veut prouver que nous pouvons stresser une base de données avec des requêtes OLAP même dans un environnement médiocre.

La référence TPCH a été définie sur un facteur d'échelle de 1, ce qui produit 6 millions de n-uplets. Ce facteur d'échelle produit une base de données de 1 Go difficile à conserver en mémoire principale. Les tables résultantes ont été stockées dans trois bases de données différentes:

RS: Une base de données Row-Store traditionnelle avec des index classiques pour optimiser les requêtes pertinentes.

VF: Fragmentation verticale de la table Lineitem comme expliqué précédemment.

T-Plotter: notre index a été placé au-dessus de la base de données VF afin de montrer le gain de performances et les avantages des stockages RS et CS.

L'index de T-Plotter qui trace sur les 6 fragments peut être mis en cache dans Oracle PGA pour des performances évidentes. Sa taille est de 0,34 Go qui se loge plus facilement dans la mémoire principale

IV.4.B.1. TEMPS D'EXÉCUTION

Nous avons exécuté les 16 requêtes de TPCH impliquant la table Lineitem. Rappelons que les caractéristiques présentées dans le Tableau 15 indiquent le nombre de fragments utilisés pour le filtrage (première étape de l'Algorithme 3) et pour la projection (deuxième étape). Il fournit également la sélectivité des prédicats pour chaque requête. Ces caractéristiques aideront à comprendre ce qui se passe pendant le processus d'évaluation.

Le Tableau 17 donne le temps résultant pour chaque requête en fonction de la structure physique. Nous pouvons identifier deux types différents de requêtes Q1 et autres. La requête Q1 est très différente des autres. En fait, comme nous l'avons vu à la section 4.2, trois fragments sont utilisés pour la projection et un seul pour le filtrage avec une sélectivité médiocre (56,86%). Cela a un impact énorme sur la mise en œuvre de la FV. En fait, le temps est quatre fois plus grand que celui de RS. Cela est dû au fait que 4 fragments sont nécessaires, avec plus de la moitié du nombre de n-uplets, ce qui ne rentre pas dans une opération HASH JOIN. Par conséquent, une latence élevée est obtenue pour traiter les trois jointures.

T-Plotter exploite ces caractéristiques pour fournir un plan d'exécution alternatif. Il accède aux fragments requis par le biais de rowid, filtre les lignes à la volée, puis rassemble les attributs restants. Les effets de facteur de regroupement permettent d'éviter l'accès à un bloc de données deux fois au cours du processus d'évaluation. Etant donné que le filtre réduit le nombre de blocs de données sur les trois fragments, même s'il est supérieur à 56%, le temps global est considérablement réduit. Le résultat final est meilleur que le stockage RS tout en restant proche de celui-ci. Ce résultat montre que notre stratégie bénéficie des avantages de la RS avec un nombre élevé de fragments en projection et une faible sélectivité.

La requête Q3 a une sélectivité élevée et nécessite la projection d'un seul fragment. En conséquence, le temps de traitement est divisé par quatre. Nous pouvons voir que T-Plotter est légèrement meilleur que VF, cela est dû au fait qu'aucun index n'est nécessaire pour reconstruire les tuples puisqu'il est déjà en mémoire principale avec l'index T-Plotter tandis que VF traite par un INDEX UNIQUE SCAN pour y accéder aux tuples référencés par la valeur tuple_id.

La requête Q14 a exactement le même type de résultats. La différence de temps est due au fait qu'une seule opération de jointure est nécessaire, ce qui entraîne un temps de traitement intéressant. Les requêtes Q5, Q7 et Q8 ont des comportements similaires mais avec plus de tables à joindre.

	Filter Frag	Selectivity %	Project Frag
Q 1	1	56.86	3
Q 3	1	0.005	1
Q 4	2	0.009	0
Q 5	1	0.001	1
Q 6	3	0.019	0
Q 7	1	0.001	1
Q 8	1	0.001	1
Q 9	1	0.054	2
Q 10	2	0.038	1
Q 12	2	0.005	0
Q 14	1	0.013	1
Q 17	2	0.0001	1
Q 18	1	0.1647	1
Q 19	3	0.0001	1
Q 20	2	0.00003	1
Q 21	2	0.000001	0

Tableau 16. Fragments par requêtes

	RS	VF	TP
Q 1	15.87	25.40	13.62
Q 3	24.25	11.50	10.61
Q 4	22.60	9.03	8.21
Q 5	74.25	6.26	6.36
Q 6	19.89	1.43	1.57
Q 7	24.00	8.79	8.76
Q 8	23.25	4.26	4.42
Q 9	25.81	5.37	5.17
Q 10	24.28	9.36	12.34
Q 12	20.60	2.46	11.12
Q 14	0.53	0.81	1.00
Q 17	19.17	2.42	2.70
Q 18	52.29	28.65	29.45
Q 19	19.85	2.32	2.39
Q 20	21.78	4.79	4.42
Q 21	53.43	10.98	11.42

Tableau 17.Temps d'exécution.

Puisque la requête Q4 contient une requête imbriquée "EXIST" sur Lineitem, elle requiert le traitement du fragment autant que la table "orders" en a besoin. Par conséquent, la mise en œuvre de RS prend un certain temps. Dans les implémentations T-Plotter et VF, seuls deux fragments sont nécessaires: un pour la jointure (imbriqué) et un pour le prédicat sur Lineitem. Ces deux étapes réduisent considérablement le nombre d'accès à effectuer. Là encore, les analyses d'index sont évitées dans l'implémentation de T_Plotter.

Requête Q10 se comporte de la même manière est un filtre préliminaire dans une table externe, puis passe à un deuxième filtre. Cependant, il faut ici encore un fragment de plus à accéder pour la projection et une sélectivité plus grande.

La requête Q6 est vraiment intéressante du point de vue de la fragmentation. En fait, trois fragments sont utilisés pour le filtrage et chaque attribut est déjà disponible pour la projection. VF affiche de très bonnes performances. Le traitement temporel de T-Plotter est légèrement supérieur, car il nécessite l'utilisation du CPU pour accéder aux index dans la mémoire principale, puis calcule les intersections, tandis

que VF calcule uniquement les intersections. Cet effet est minimisé dans les autres requêtes car aucune projection n'est requise dans la requête Q6.

La requête Q12 peut être comparée à Q6 pour les fragments de filtrage multiples, aucune projection n'est requise. Même avec une sélectivité plus forte, le temps de traitement est plus long. Ceci est dû à une caractéristique très particulière de cette requête qui implique deux attributs présents dans deux fragments différents (l_shipdate, l_commitdate). Cette particularité implique un second traitement qui doit être effectué après l'accès aux fragments (donc moins de filtrage que prévu). Enfin, le filtre est appliqué dans la mémoire principale.

La requête Q9 contient plusieurs opérations de jointure avec une sélectivité de 0,054% (très sélective). Cependant, un seul fragment est nécessaire pour ces jointures, mais deux fragments sont nécessaires pour la projection. La fragmentation divise par deux le temps de traitement en réduisant la quantité de données consultées (sauf en cas de matérialisation tardive).

La requête Q19 est une requête disjonctive avec trois prédicats similaires dans lesquels un seul prédicat est en train de changer. Cette fonctionnalité est très intéressante dans notre cas puisque tous les filtres sont identiques dans un premier temps, et le dernier fragment est appliqué pour produire la disjonction. Même si la sélectivité est vraiment intéressante, ce multi-filtre, associé à une jointure, fournit un temps de traitement moyen.

En fin de compte, TP ne remplace pas la conception fragmentation verticale, mais il est complémentaire à cette conception. Il permet de comblez les lacunes en cas d'échec de la FV naïve. Nous proposons donc de combiner les solutions de remorquage VF et TP afin de trouver une solution générale à l'ensemble des requêtes, pour être le meilleur, cette combinaison est présentée dans le tableau 18.

	RS	VF	TP	VF++
Q 1	15.87	25.40	13.62	13.62
Q 3	24.25	11.50	10.61	11.50
Q 4	22.60	9.03	8.21	8.21
Q 5	74.25	6.26	6.36	6.26
Q 6	19.89	1.43	1.57	1.43
Q 7	24.00	8.79	8.76	8.76
Q 8	23.25	4.26	4.42	4.26
Q 9	25.81	5.37	5.17	5.17
Q 10	24.28	9.36	12.34	9.36
Q 12	20.60	2.46	11.12	2.46
Q 14	0.53	0.81	1.00	0.81
Q 17	19.17	2.42	2.70	2.42
Q 18	52.29	28.65	29.45	28.65
Q 19	19.85	2.32	2.39	2.32
Q 20	21.78	4.79	4.42	4.42
Q 21	53.43	10.98	11.42	10.98
Total	441.95	133.8	133.56	119,7

tableau 18 : Temps d'exécution

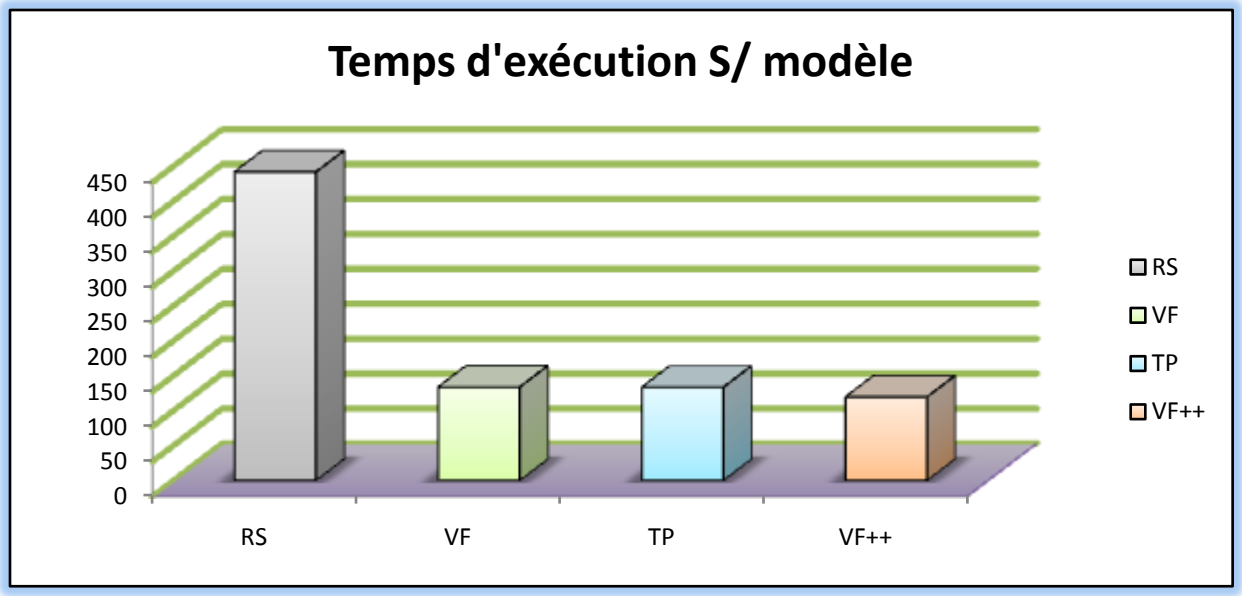


Figure 26. Temps d'exécutions des requêtes sur les différents modèles.

IV.4.B.2. CONCLUSION:

Selon ce panel de requêtes, nous pouvons conclure que le nombre de fragments accédé pour la projection a un impact plus important sur le temps de traitement que les fragments de filtrage. La sélectivité l'influence aussi mais surtout celles les plus hautes. De plus, l'utilisation d'un fragment pour joindre deux tables est également intéressante, car elle contient moins de données dans l'opérateur de jointure.

Nous pouvons remarquer que T-Plotter évite de passer du temps dans les index pour la reconstruction des tuples, ce qui offre de meilleures performances que la fragmentation verticale simple pour certains types de requêtes.

Comme on peut le voir sur la Figure 26, le temps a été divisé par trois. Selon RS, l'essentiel du gain est obtenu par des requêtes à forte sélectivité. À l'inverse, pour la fragmentation verticale, le gain est réalisé exclusivement sur une sélectivité médiocre (Q1).

IV- 4- C. REQUETES DE MISES A JOUR :

La deuxième série d'expériences tente d'éclairer les gains de performances en termes de mises à jour.

En fait, les bases de données classiques de orienté colonne reposent sur des vues matérialisées et des structures temporaires pour optimiser les traitements, mais ont

un impact sur les mises à jour. Nous montrons ici qu'avec un seul index, ce temps est considérablement réduit, ce qui conduit à des résultats intéressants.

Le Tableau 19 montre trois requêtes de mise à jour qui montrent en particulier l'impact d'un nombre différent de fragments. Respectivement un pour U1, deux pour U2 et trois pour U3. Nous avons également exécuté ces trois requêtes sur une sélectivité variable (0,00001%, 0,16% et 13,76%) pour déterminer l'impact de la sélectivité sur le temps de mise à jour.

Le Tableau 16 donne les résultats pour chaque requête, sélectivité et stratégie de stockage physique.

Nous pouvons voir que VF et T-Plotter surpassent le Row Store. Le résultat est évident pour un fragment, mais ce gain est similaire lors de la mise à niveau du nombre de fragments mis à jour.

Ce problème majeur correspond au fait que les n_uplets sont stockés dans des blocs de données qui sont gérés en cache lors de la mise à jour. Étant donné que les données sont fragmentées en fonction d'un sous-ensemble d'attributs, davantage de n_uplets sont stockés dans un même bloc de données, de sorte que la probabilité que deux prédicats mis à jour soient réalisés sur un même bloc de données est plus grande et maximise par conséquent l'utilisation du cache. Ce gain est plus faible pour U3, même si VF et T-Plotter sont meilleurs comparés aux bases de données orientées ligne.

Cet effet provient de la spécificité du fragment F6 (contenant 1 fragment commitdate). Ce fragment contient des attributs textuels (l shipinstruct et l shipmode), ce qui réduit considérablement la probabilité d'augmenter la longueur des n-uplets fragmentés. Par conséquent, la quantité de blocs de données mis à jour est plus élevée.

Requête	
U1	UPDATE Lineitem SET l_quantity=l_quantity+10 WHERE l_shipdate BETWEEN date1 AND date2
U2	UPDATE Lineitem SET l quantity=l quantity+10, l_extendedprice=l_extendedprice+100 WHERE l_shipdate between date1 and date2
U3	UPDATE Lineitem SET l_quantity=l_quantity+10, l_extendedprice=l_extendedprice+100, l_commitdate=l_commitdate+100 WHERE l_shipdate BETWEEN date1 AND date2

Tableau 19. Requêtes de mise à jour sur Lineitem.

	Sélectivité	Orientée ligne	VF	T-Plotter
U1	0.00001%	0.359	0.031	0.062
	0.16%	0.986	0.063	0.064
	13.76%	38.251	2.605	4.050
U2	0.00001%	0.343	0.037	0.036
	0.16%	0.960	0.390	0.406
	13.76%	42.62	3.212	4.742
U3	0.00001%	0.375	0.047	0.041
	0.16%	1.103	0.515	0.577
	13.76%	34.991	13.526	18.392

Tableau 20. Updates processing time (in seconds)

IV-5. CONCLUSION

Dans ce travail, nous soulignons le fait que toutes les études précédentes se sont concentrées sur la manière de fragmenter la base de données car il s'agit d'un problème difficile et complet sur NP, mais pas sur le modèle physique de la fragmentation verticale. En effet, nous avons identifié le point faible de la fragmentation verticale, nous avons fourni une solution pour facilement adapter et utiliser cette technique sur des bases de données traditionnelles.

T-Plotter est une structure de données unique pour améliorer la reconstruction de tuples et en même temps un algorithme pour le traiter efficacement en fonction de son modèle de coût et de la couche de fragmentation. Cette structure simple mais puissante aide à composer des tables fragmentées en n_uplets originaux. Il cible les avantages de la matérialisation tardive et précoce, pour pouvoir accéder aussi tard que possible aux données afin de réduire la consommation de mémoire, tôt pour éviter les accès multiples à un attribut.

Il permet également de stocker des fragments dans des ordres physiques indépendants, ce qui peut constituer un besoin crucial pour des requêtes de filtrage spécifiques. Il réduit le temps de traitement des requêtes qui exploitent une grande quantité de données, en particulier pour les tables étendues orientées colonne. Il maximise l'utilisation de filtres et la matérialisation tardive qui s'attaque au problème de la reconstruction des tuples.

Cette solution n'est pas un concurrent des bases de données orienté colonne; En fait, elle vise à améliorer chaque type de base de données en tant que couche supérieure pour recomposer des données fragmentées. Elle propose donc un index complémentaire à d'autres types de bases de données. Il peut être exploité comme techniques d'optimisation souvent utilisées dans le stockage en colonne, telles que la compression [15], la matérialisation ou le fichier BAT pour MonetDB [15, 16, 52].

CHAPITRE V : CONCLUSIONS & PERSPECTIVES.

Dans cette étude nous nous somme intéressés au sujet de la fragmentation verticale sur les bases de données classiques orientées ligne, dans le premier volet on s'est concentré sur l'optimisation du schéma de la fragmentation verticale, et nous avons comparé deux algorithmes de la littérature, notre choix s'est porté sur l'utilisation de deux variantes algorithmiques basés sur l'évaluation par le modèle de coût, l'algorithme génétique et l'algorithme Apriori, et comme contribution nous en avons proposé un troisième algorithme que nous l'avions baptisé Affiner.

Cette étude nous à permis de simuler la fragmentation verticale sur les bases et entrepôts de données. Nous somme parvenus à proposer dans ce travail, un algorithme appelé Affiner. Affiner, est une approche qui joue le rôle d'éclaireur pour trouver le meilleur schéma pour la fragmentation verticale.

L'algorithme Affiner pourrait servir pour assister les administrateurs de bases de données dans l'optimisation des performances de la base de données, et pourrait servir comme un banc d'essais offrant la possibilité de fragmenté verticalement la base de données.

Les tests ont démontré l'efficacité et le gain apporté quand à l'utilisation de la fragmentation verticale sur une base de données, bien que ces tests ne représentent que quelques formes de requêtes, néanmoins la fragmentation verticale semble bien répondre à l'optimisation des performances d'une base de données assez volumineuse (entrepôts de données) traitant des requêtes de type OLAP.

Mais nous avons constaté que la fragmentation verticale est limitée, car il y a des cas où une base de données fragmentée verticalement présente des chutes de performances par rapport à une bases de données ordinaire non fragmentée, même si elle est fragmentée d'une manière optimale, ce qui nous amène déduire que l'architecture de la fragmentation verticale elle-même doit être révisée et amélioré pour quelle soit efficace et pouvoir rivaliser avec les autres architecture SGBD récemment apparue sur le marché, c'est pour cette raison nous avons proposé une toute nouvelle architecture permettant de booster les performances de la fragmentation verticale et proposer la nouvelle conception physique de la fragmentation verticale qu'on nommera T-Plotter.

T-Plotter est une nouvelle architecture qui a permis de combler les lacunes dans la mise en œuvre de la fragmentation verticale sur des bases de données classiques fragmentées verticalement pour le traitement de modèles OLAP. La reconstruction des tuples représente toujours un coût important pour les bases de données orientées colonne [15, 45], même pour celles In Memory [44]. Nos expériences ont montré que T-Plotter peut tenir dans la mémoire principale même si les bases de données standard n'exploitent pas pleinement les emplacements disponibles (par exemple, la mémoire PGA doit être maximisée). De même, les bases de données orientées colonne tirent pleinement parti des paramètres de la machine pour le processeur et la mémoire 45].

Nous estimons que la taille du T-Plotter peut être réduite à un cinquantième de sa taille initiale, ce qui représente un gain intéressant. Même si la compression de la structure du T-Plotter peut réduire sa taille, cette fonctionnalité nivellera la charge du processeur et réduira les gains de performances. Cela devrait avoir un impact négatif sur les performances de la base de données.

Une autre possibilité est de changer la façon de reconstruire les tuples. L'idée clé est de stocker T-Plotter dans un fragment avec des valeurs distinctes, et les tuples de T-Plotter pointent vers ces valeurs (plusieurs tuples sur une même valeur). La fonction de reconstruction se concentre sur la reconstruction des valeurs et non sur les valeurs d'attribut. Cela peut réduire la taille de chaque fragment et améliorer chaque opération de regroupement pouvant être effectuée sur les valeurs ROWID plutôt que sur les valeurs. Nous pouvons voir cette fonctionnalité comme une matérialisation ultérieure.

D'autres points faibles de T-Plotter peuvent être améliorés dans les travaux futurs: 1) Les jointures utilisées lors de la reconstruction des tuples peuvent être optimisées car elles ne prennent pas en compte les cardinalités de la jointure. Les index de jointure aussi peuvent être utilisés pour résoudre ce problème. 2) Les environnements distribués, comme pour les bases de données larges orientées colonnes, proposent une approche complémentaire en parallélisant le traitement. Notre structure de données pourrait être répartie sur un cluster afin d'optimiser la reconstruction des tuples dans un contexte distribué. 3) Un dernier problème concerne l'évolution de l'affinité des fragments avec les requêtes sur la base de données, cela nécessite d'ajouter ou de supprimer des fragments en mémoire de T-Plotter.

Nous espérons ouvrir une nouvelle voie pour développer de nouveaux concepts et techniques permettant d'améliorer les performances des bases de données, afin de réussir la gestion des volumes de données d'une manière efficace sans recourir à élever les performances des machines et serveurs sur lesquels ils sont installées, ce qui a un impact positif sur les coûts de gestion, et l'environnement.

Cette étude à pour but de booster les performances des bases de données classiques, mais nous avons l'intime conviction que notre étude aura un impact positif sur toute les autres formes de bases de données non classiques, car nous allons proposé une nouvelle technique de conception physiques, et si cette approche réussi sur une forme classique, elle sera surement adaptable sur d'autres formes de bases de données, et c'est d'ailleurs le cas des techniques de conception physique.

BIBLIOGRAPHIES

1. Abadi D, Boncz PA, Harizopoulos S, Idreos S, Madden S. The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends in Databases. 2013;5(3):197-280.
2. Abadi D, Madden S, Ferreira M, editors. "Integrating Compression and Execution in Column-oriented Database Systems". Proceedings of the 2006 International Conference on Management of Data (SIGMOD'06); 2006 ACM.
3. Abadi DJ, Madden S, Hachem N, editors."Column-stores vs. row-stores: how different are they really?" Proceedings of the ACM International Conference on Management of Data (SIGMOD'08); 2008 2008.
4. Abadi DJ, Myers DS, DeWitt DJ, Madden S, editors." Materialization Strategies in a Column-Oriented DBMS". Proceedings of the 23rd International Conference on Data Engineering, (ICDE'07); 2007 2007.
5. Abadi DJ, Stonebraker M, Batkin A, Chen X, Cherniack M, Ferreira M, et al., editors. "C-store: A Column-oriented DBMS". Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05); 2005 2005: VLDB Endowment.
6. Abdalla HI and Marir F. "Vertical Partitioning Impact", in 18th National Computer Conference 2006, Saudi Computer Society.
7. Agrawal S, Surajit Chaudhuri and Vivek R. Narasayya, "Automated Selection of Materialized Views and Indexes in SQL Databases", Proceedings of 26th International Conference on Very Large Data Bases, 2000: 496-505.
8. Agrawal S, Vivek R. Narasayya, Beverly Yang, "Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design". SIGMOD Conference 2004.
9. Agrawal, R and Srikant, R. "Fast algorithms for mining association rules in large databases". in 20th International VLDB, pages 487-499, Santiago, Chile, September 1994.
10. Agrawal, S. et al. "Database Tuning Advisor for Microsoft SQL Server 2005". In Proceedings of the 30th VLDB, Toronto, Canada, 2004.
11. Agrawal, S. et al. "Database Tuning Advisor in SQL Server 2005". White paper. http://www.microsoft.com/technet/prodtechnol/sql/2005/sql2005dta.mspx
12. Angel, F. & al. Taddei-Zavala "Simultaneous Vertical Fragmentation and Segment Assignment in Distributed Data Bases using Genetic Algorithms".
13. APB1 OLAP Council, "APB-1 olap benchmark, release II," http://www.olapcouncil.org/research/bmarkly.htm.
14. Avnur, R., and Hellerstein, J. Eddies: "Continuously Adaptive Query Processing". In Proceedings of ACM SIGMOD Conference 2000.
15. Boissier M, Spivak A, Meyer C, editors. "Improving Tuple Reconstruction for Tiered Column Stores: A Workload-aware Ansatz Based on Table Reordering". Proceedings of the Australasian Computer Science Week Multiconference; 2017 2017: ACM.
16. Boncz PA, Zukowski M, Nes N, editors. "MonetDB/X100: Hyper-Pipelining Query Execution". Second Biennial Conference on Innovative Data Systems Research (CIDR'05); 2005
17. Cardenas, A. F. "Analysis and performance of inverted data base structures". Comm. ACM 18, 5 (Mai 1975); 253 -263.

18. Ceri, S. & Pernici,S. and Weiderhold,G. "Optimization Problems and Solution Methods in the Design of Data distribution". Information Sciences Vol 14, No. 3, p 261-272, 1989.
19. Chaalal H, Belbachir H. "An optimized vertical fragmentation approach". International Journal of Innovative Technology and Exploring Engineering (IJITEE'13). 2013;3(4):33-9.
20. Chakravarthy, S., Muthuraj, J., Varadarajan, R., and Navathe, S. B. "An objective function for vertically partitioning relations in distributed databases and its analysis". Distributed and Parallel Databases 2, 2 (1994), 183–207.
21. Chaudhuri, S. And Vivek R. Narasayya, "Microsoft Index Tuning Wizard for SQL Server 7.0", Proceedings ACM SIGMOD International Conference on Management of Data, 1998: 553-554.
22. Chaudhuri, S. and Narasayya, V. An Efficient, "Cost-Driven Index Selection Tool for Microsoft SQL Server". In Proceedings of the VLDB, Athens, Greece, 1997.
23. Chaudhuri, S. and Narasayya, V. AutoAdmin "What-If" Index Analysis Utility. In Proceedings of ACM SIGMOD, Seattle, WA, 1998. Pages 367–378
24. Cheng. C.H; & Lee, W-K; Wong, K-F, "A Genetic Algorithm-Based Clustering Approach for Database Partitioning" IEEE Transactions on Systems, Man, and Cybernetics, 32(3), 2002, 215-230. 33.
25. Chu, P.-C. "A transaction oriented approach to attribute partitioning". Information Systems 17, 4 (1992), 329–342.
26. Cornell, D., and Yu, P. "A vertical partitioning algorithm for relational databases". In International Conference on Data Engineering (1987), pp. 30–35.
27. Dageville, B., Das, D., Dias, K., Yagoub, K., Zaït, M., Ziauddin, M. "Automatic SQL Tuning in Oracle 10g". In Proceedings of VLDB 2004.
28. Goldberg D. "Genetic Algorithms in Search, Optimization and Machine Learning". Reading MA Addison Wesley, 1989.
29. Goldberg D., "Algorithmes génétiques, Adisson Wesley, France", 1994.[Gol89c]
30. Golfarelli, M. & Maio, D. & Rizzi, S. (1999). "Vertical Fragmentation of Views in Relational Data Warehouses". Settimo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati (SEBD 99), Como, Italy (pp. 19–33).
31. Gorla, N. & Pang Wing "vertical fragmentation in Databases Using Data-Mining technique", IGI Global Vol.4, Issue 3. 2008.
32. Gorla, N. "A Methodology for Vertically Partitioning in a Multi-Relation Database Environnement", JCS&T Vol.7 No. 3 October 2007.
33. Hammer, M. & Niamir, B. "A heuristic approach to attribute partitioning. In Proceedings ACM SIGMOD Int. Conf. on Management of Data", (Boston, Mass., 1979), ACM, New York, pages 93-101.
34. Hoffer, J. & Severance,D. "The Uses of Cluster Analysis in Physical Database Design", Proc in 1st International Conference on VLDB, Framingham, MA, 1975.
35. Hoffer, J. "An integer programming formulation of computer database design problems". Inf. Sci., 11(July 1976), 29-48.
36. Holland J.H. "Adaptation in natural and artificial system", Ann Arbor, The University of Michigan Press, 1975.
37. Lightstone, S., Teorey, T., and Nadeau, T. "Physical Database Design : the database professional's guide to exploiting indexes, views, storage, and more", Morgan Kaufmann Press, 2007. ISBN:

0123693896.
38. Lin, X., and Zhang, Y. "A new graphical method of vertical partitioning in database design". In Proceedings of the 4th Australian Database Conference (ADC) (1993), M. P. P. Maria E. Orlowska, Ed., World Scientific, pp. 131–144.
39. Lin, X., Orlowska, M., and Zhang, Y. "A graph based cluster approach for vertical partitioning in database design". Data and Knowledge Engineeering 11, 2 (1993), 151–169.
40. Navathe, S. & Ceri,S. & Weiderhold,G. and Dou,J. "Vertical Partitioning Algorithms for Database Design" ACM Transactions on Database Systems, Vol. 9, No. 4, 1984.
41. Navathe, S. & Ra, M. "Vertical Partitioning for Database Design: A Graphical Algorithm". ACM SIGMOD, Portland, Juin 1989.
42. Özsu .M. & Valduriez, P "Principles of Distributed Database Systems", 2nd edition (1 st edition 1991), New Jersey, Prentice-Hall, 1999.
43. P. LYMAN, H. VARIAN, J. DUNN, A. STRYGIN & K. SWEARINGEN, "How Much Information? ", School of Information Management and Systems, University of California, Berkeley, 2000 et 2003 (http ://www2.sims.berkeley.edu/research/projects/how-much-info-2003).
44. Petraki E, Idreos S, Manegold S, editors. "Holistic Indexing in Main-memory Column-stores. Proceedings of International Conference on Management of Data" (SIGMOD'15); 2015: ACM.
45. Plattner H. "The Impact of Columnar In-memory Databases on Enterprise Systems: Implications of Eliminating Transaction-maintained Aggregates". Proc VLDB Endow. 2014;7(13):1722-9.
46. Robinson AHC, C "Results of a prototype television bandwidth compression scheme". Proceedings of the IEEE 1967;55(3):356–64.
47. Sam S. Lightstone, Toby J. Teorey, Tom Nadeau "Physical Database Design, The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More", Morgan Kaufmann Publishers Mars 2007, ISBN 13: 978-0-12-369389-1.
48. Son, J. H., and Kim, M. H. "An adaptable vertical partitioning method in distributed systems". Journal of Systems and Software 73, 3 (2004), 551–561.
49. Song, S.K. & Gorla, N., "A genetic Algorithm for Vertical Fragmentation and Access Path Selection," The Computer Journal, vol. 45, no. 1, 2000, pp 81-93.
50. TPCH: ad-hoc, decision support benchmark. Transaction Processing Performance Council (TPC), http://www.tpc.org/tpch.
51. Valentin, G., Zuliani, M., Zilio, D., Lohman, G. and Skelley, A. DB2 Advisor: "An Optimizer Smart Enough to Recommend its Own Indexes". In Proceedings of ICDE, San Diego, USA, 2000.
52. Vermeij M, Quak W, Kersten M, Nes N, editors. Monetdb, a novel spatial columnstore dbms. Academic Proceedings of the 2008 Free and Open Source for Geospatial (FOSS4G) Conference, OSGeo; 2008 2008.
53. Yao, S. B. (1977). "Approximating block access in data-base organization. Communications of the ACM", 20(4), 260-261.

Appendice A : organisation physique des fichiers

Un SGBD stocke et gère des fichiers, l'organisation de ces fichiers fait partie du domaine du système d'exploitation st diffère pour chaque système d'exploitation, les fichiers seront stockés en vrac sur un disque, l'organisation d'un disque mémoire est décrite comme suit.

L'espace de stockage Le disque est lui-même composé de plusieurs plateaux ou disques, chaque plateaux et constitué de plusieurs pistes, les pistes de plusieurs plateaux interposées et sur lesquelles sont posées les têtes de lecture sont appelés cylindre, chaque piste peut être divisées en plusieurs blocs ou secteurs, le schéma suivant récapitule ces notions :

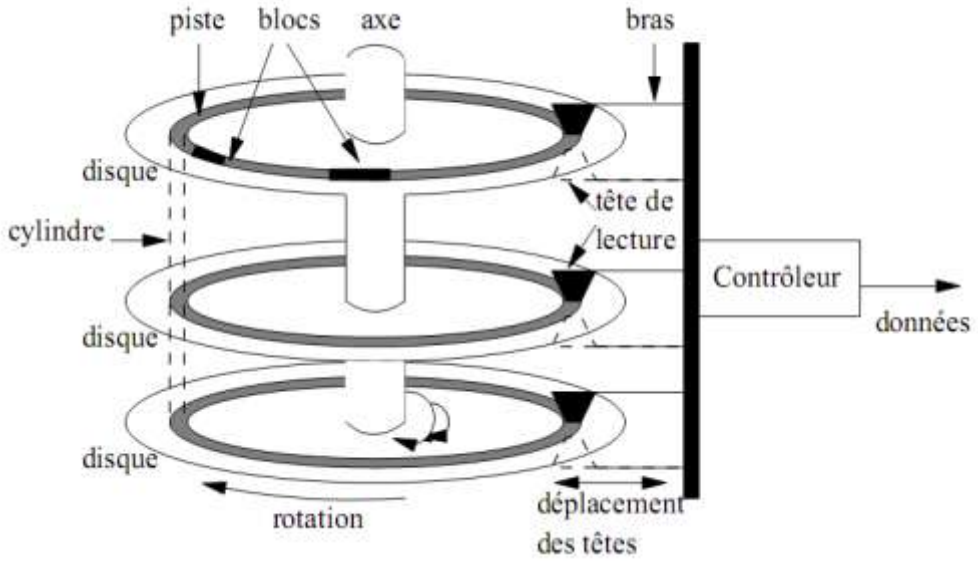


Figure 27 : Structure d'un disque.

Le système repère une donnée sur disque par sa mémoire, l'adresse est un code composé du numéro de plateaux (numéro de disque), la piste ou se trouve le bloc, et le numéro de bloc sur la piste.

Le temps nécessaire pour récupérer une information sur disque comporte le :

- Délai de positionnement : temps nécessaire à la tête de lecture pour qu'elle soit positionnée au dessus de la bonne piste.
- Délais de latence : temps nécessaire pour que la tête de lecture passe au dessus du bloc contenant la donnée.
- Délai de transfert : temps pour que le ou les blocs contenant les données soient lus et transférés.

Il est à noter que la plus petite unité de lecture pour une tête est de 1 bloc, c'est l'unité d'entrée sortie.

Grâce à ces données on déduit le temps nécessaire pour faire une lecture sur disque, il est important de noter que l'organisation des blocs peut avoir une grande influence sur le temps de lecture des données.

Appendice B : Requêtes TPCH.

```
/* TPC_H Query 1 - Pricing Summary Report */
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS
SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM LINEITEM
WHERE L_SHIPDATE <= dateadd(dd, -90, cast('1998-12-01' as date))
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS

/* TPC_H Query 2 - Minimum Cost Supplier */
SELECT TOP 100 S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
P_TYPE LIKE '%%BRASS' AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND
R_NAME = 'EUROPE' AND
PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST) FROM PARTSUPP, SUPPLIER, NATION, REGION
 WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
 AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY

/* TPC_H Query 3 - Shipping Priority */
SELECT TOP 10 L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE, O_ORDERDATE,
O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING' AND C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND
O_ORDERDATE < '1995-03-15' AND L_SHIPDATE > '1995-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE

/* TPC_H Query 4 - Order Priority Checking */
SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT FROM ORDERS
WHERE O_ORDERDATE >= '1993-07-01' AND O_ORDERDATE < dateadd(mm,3, cast('1993-07-01' as date))
AND EXISTS (SELECT * FROM LINEITEM WHERE L_ORDERKEY = O_ORDERKEY AND L_COMMITDATE <
L_RECEIPTDATE) GROUP BY O_ORDERPRIORITY ORDER BY O_ORDERPRIORITY

/* TPC_H Query 5 - Local Supplier Volume */
SELECT N_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA' AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < DATEADD(YY, 1, cast('1994-01-01' as date))
GROUP BY N_NAME  ORDER BY REVENUE DESC

/* TPC_H Query 6 - Forecasting Revenue Change */
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE L_SHIPDATE >= '1994-01-01' AND L_SHIPDATE < dateadd(yy, 1, cast('1994-01-01' as date))
AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01 AND L_QUANTITY < 24
```

```
/* TPC_H Query 7 - Volume Shipping */
SELECT SUPP_NATION, CUST_NATION, L_YEAR, SUM(VOLUME) AS REVENUE
FROM ( SELECT N1.N_NAME AS SUPP_NATION, N2.N_NAME AS CUST_NATION, datepart(yy, L_SHIPDATE) AS
L_YEAR,
 L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
 FROM SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2
 WHERE S_SUPPKEY = L_SUPPKEY AND O_ORDERKEY = L_ORDERKEY AND C_CUSTKEY = O_CUSTKEY
 AND S_NATIONKEY = N1.N_NATIONKEY AND C_NATIONKEY = N2.N_NATIONKEY AND
 ((N1.N_NAME = 'FRANCE' AND N2.N_NAME = 'GERMANY') OR
 (N1.N_NAME = 'GERMANY' AND N2.N_NAME = 'FRANCE')) AND
 L_SHIPDATE BETWEEN '1995-01-01' AND '1996-12-31' ) AS SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR

/* TPC_H Query 8 - National Market Share */
SELECT O_YEAR, SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0 END)/SUM(VOLUME) AS
MKT_SHARE
FROM (SELECT datepart(yy,O_ORDERDATE) AS O_YEAR, L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME,
N2.N_NAME AS NATION
 FROM PART, SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2, REGION
 WHERE P_PARTKEY = L_PARTKEY AND S_SUPPKEY = L_SUPPKEY AND L_ORDERKEY = O_ORDERKEY
 AND O_CUSTKEY = C_CUSTKEY AND C_NATIONKEY = N1.N_NATIONKEY AND
 N1.N_REGIONKEY = R_REGIONKEY AND R_NAME = 'AMERICA' AND S_NATIONKEY = N2.N_NATIONKEY
 AND O_ORDERDATE BETWEEN '1995-01-01' AND '1996-12-31' AND P_TYPE= 'ECONOMY ANODIZED STEEL') AS
ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR

/* TPC_H Query 9 - Product Type Profit Measure */
SELECT NATION, O_YEAR, SUM(AMOUNT) AS SUM_PROFIT
FROM (SELECT N_NAME AS NATION, datepart(yy, O_ORDERDATE) AS O_YEAR,
 L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
 FROM PART, SUPPLIER, LINEITEM, PARTSUPP, ORDERS, NATION
 WHERE S_SUPPKEY = L_SUPPKEY AND PS_SUPPKEY= L_SUPPKEY AND PS_PARTKEY = L_PARTKEY AND
 P_PARTKEY= L_PARTKEY AND O_ORDERKEY = L_ORDERKEY AND S_NATIONKEY = N_NATIONKEY AND
 P_NAME LIKE '%%green%%') AS PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC

/* TPC_H Query 10 - Returned Item Reporting */
SELECT TOP 20 C_CUSTKEY, C_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE, C_ACCTBAL,
N_NAME, C_ADDRESS, C_PHONE, C_COMMENT
FROM CUSTOMER, ORDERS, LINEITEM, NATION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE>= '1993-10-01' AND
O_ORDERDATE < dateadd(mm, 3, cast('1993-10-01' as date)) AND
L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS, C_COMMENT
ORDER BY REVENUE DESC

/* TPC_H Query 11 - Important Stock Identification */
SELECT PS_PARTKEY, SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000
```

```
 FROM PARTSUPP, SUPPLIER, NATION
 WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY')
ORDER BY VALUE DESC

/* TPC_H Query 12 - Shipping Modes and Order Priority */
SELECT L_SHIPMODE,
SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY = '2-HIGH' THEN 1 ELSE 0 END) AS
HIGH_LINE_COUNT,
SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH' THEN 1 ELSE 0 END )
AS LOW_LINE_COUNT
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE AND L_SHIPDATE < L_COMMITDATE AND L_RECEIPTDATE >= '1994-01-
01' AND L_RECEIPTDATE < dateadd(mm, 1, cast('1995-09-01' as date))
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE

/* TPC_H Query 13 - Customer Distribution */
SELECT C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY)
 FROM CUSTOMER left outer join ORDERS on C_CUSTKEY = O_CUSTKEY
 AND O_COMMENT not like '%%special%%requests%%'
 GROUP BY C_CUSTKEY) AS C_ORDERS (C_CUSTKEY, C_COUNT)
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC, C_COUNT DESC

/* TPC_H Query 14 - Promotion Effect */
SELECT 100.00* SUM(CASE WHEN P_TYPE LIKE 'PROMO%%' THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM LINEITEM, PART
WHERE L_PARTKEY = P_PARTKEY AND L_SHIPDATE >= '1995-09-01' AND L_SHIPDATE < dateadd(mm, 1, '1995-
09-01')

/* TPC_H Query 15 - Create View for Top Supplier Query */
CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
SELECT L_SUPPKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) FROM LINEITEM
WHERE L_SHIPDATE >= '1996-01-01' AND L_SHIPDATE < dateadd(mm, 3, cast('1996-01-01' as date))
GROUP BY L_SUPPKEY

 /* TPC_H Query 15 - Top Supplier */
SELECT S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
FROM SUPPLIER, REVENUE0
WHERE S_SUPPKEY = SUPPLIER_NO AND TOTAL_REVENUE = (SELECT MAX(TOTAL_REVENUE) FROM REVENUE0)
ORDER BY S_SUPPKEY
DROP VIEW REVENUE0

/* TPC_H Query 16 - Parts/Supplier Relationship */
SELECT P_BRAND, P_TYPE, P_SIZE, COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM PARTSUPP, PART
WHERE P_PARTKEY = PS_PARTKEY AND P_BRAND <> 'Brand#45' AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'
AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9) AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY FROM SUPPLIER
 WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
GROUP BY P_BRAND, P_TYPE, P_SIZE
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE
```

```
/* TPC_H Query 17 - Small-Quantity-Order Revenue */
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = P_PARTKEY)

/* TPC_H Query 18 - Large Volume Customer */
SELECT TOP 100 C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE, SUM(L_QUANTITY)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE O_ORDERKEY IN (SELECT L_ORDERKEY FROM LINEITEM GROUP BY L_ORDERKEY HAVING
 SUM(L_QUANTITY) > 300) AND C_CUSTKEY = O_CUSTKEY AND O_ORDERKEY = L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE

/* TPC_H Query 19 - Discounted Revenue */
SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM LINEITEM, PART WHERE (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12' AND P_CONTAINER IN
('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10 AND P_SIZE
BETWEEN 1 AND 5 AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR (P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23' AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED
PKG', 'MED PACK') AND L_QUANTITY >=10 AND L_QUANTITY <=10 + 10 AND P_SIZE BETWEEN 1 AND 10
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34' AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG
PACK', 'LG PKG') AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10 AND P_SIZE BETWEEN 1 AND 15
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'

/* TPC_H Query 20 - Potential Part Promotion */
SELECT S_NAME, S_ADDRESS FROM SUPPLIER, NATION
WHERE S_SUPPKEY IN (SELECT PS_SUPPKEY FROM PARTSUPP
 WHERE PS_PARTKEY in (SELECT P_PARTKEY FROM PART WHERE P_NAME like 'forest%%') AND
 PS_AVAILQTY > (SELECT 0.5*sum(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = PS_PARTKEY AND
  L_SUPPKEY = PS_SUPPKEY AND L_SHIPDATE >= '1994-01-01' AND
  L_SHIPDATE < dateadd(yy,1,'1994-01-01'))) AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME

/* TPC_H Query 21 - Suppliers Who Kept Orders Waiting */
SELECT TOP 100 S_NAME, COUNT(*) AS NUMWAIT
FROM SUPPLIER, LINEITEM L1, ORDERS, NATION WHERE S_SUPPKEY = L1.L_SUPPKEY AND
O_ORDERKEY = L1.L_ORDERKEY AND O_ORDERSTATUS = 'F' AND L1.L_RECEIPTDATE> L1.L_COMMITDATE
AND EXISTS (SELECT * FROM LINEITEM L2 WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
 AND L2.L_SUPPKEY <> L1.L_SUPPKEY) AND NOT EXISTS (SELECT * FROM LINEITEM L3 WHERE L3.L_ORDERKEY =
L1.L_ORDERKEY AND  L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE) AND
S_NATIONKEY = N_NATIONKEY AND N_NAME = 'SAUDI ARABIA'
GROUP BY S_NAME ORDER BY NUMWAIT DESC, S_NAME

/* TPC_H Query 22 - Global Sales Opportunity */
SELECT CNTRYCODE, COUNT(*) AS NUMCUST, SUM(C_ACCTBAL) AS TOTACCTBAL
FROM (SELECT SUBSTRING(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL
 FROM CUSTOMER WHERE SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17') AND
 C_ACCTBAL > (SELECT AVG(C_ACCTBAL) FROM CUSTOMER WHERE C_ACCTBAL > 0.00 AND
  SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')) AND
 NOT EXISTS ( SELECT * FROM ORDERS WHERE O_CUSTKEY = C_CUSTKEY)) AS CUSTSALE
GROUP BY CNTRYCODE ORDER BY CNTRYCODE
```

Liste des Tableaux :

Liste des Equations :

Liste des Figures :