République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Mémoire de Licence

Domaine Mathématiques et Informatique

Spécialité

Ingénierie des Systèmes d'Information et des Logiciels

Thème

Conception d'un simulateur en ligne pour l'aide aux administrateurs et concepteurs de bases de données.

Présenté par :

Mr ADDI Ramzi

Mr BENAMARA Soufian Przemyslaw

Proposé par :

Mme KERKAD Amira

Devant le jury :

Mr A. HIMRANE (Président)

Mme N. OUROUA (Membre)

Projet N° : 041/2018

Remerciements

Nous commençons par exprimer notre gratitude à Dieu, tout puissant, qui nous a aidé à mener à bien ce travail.

Nous remercions ensuite notre promotrice Madame Amira KERKAD, qui nous a donné la chance de suivre ce sujet, et qui nous a soutenu, encouragé et qui nous a fait apprendre énormément de choses.

Nous remercions aussi nos parents qui ne nous laissent jamais tomber et qui ne cessent de nous aider, ainsi que tous ceux qui nous ont encouragés et motivés.

Ramzi et Soufian

Addi Ramzi : A tous ceux qui ont m'aidé dans mon parcours.

Benamara Soufian Przemyslaw : A mes chers parents,
ma Pologne bien-aimée.

Table de matières

Chapitre I : Introduction générale

Les informations nous entourent, on les trouve partout, elles sont devenues aujourd'hui le produit le plus chère. Au cours de quatre décennies, des concepts, méthodes et algorithmes ont été développés pour les gérer. Des outils d'hiérarchisation et accès instantané ont été conçues, c'est l'ère des Bases de Données (BD). Au début de leur popularisation, les BD faisaient leur travail proprement en toute efficacité et en respectant les délais, facilitant en conséquence la sauvegarde et garantissant un accès rapide dans le cas de modification ou consultation. Au fil des années, et suite au manipulation du grand volume de données ainsi que les usages complexes exigés par les utilisateurs dans différents domaines, ces BD (dites classiques ou traditionnelles) exécutaient les ordres (requêtes) de plus en plus lourds qui affectaient considérablement les critères phares de l'informatique : le temps et l'espace mémoire. Suite à ce nouvel problème, une architecture de stockage fraiche est apparue : les Entrepôts de Données (ED) (en Anglais : Data Warehouse), ils étaient adaptés aux attentes et offraient des techniques modernes pour le traitement de l'information dans un temps raisonnable, tels que : analyses de données et prise de décisions. L'évolution apporte des biens qui mènent à des défis, même avec des solutions aussi puissantes et sophistiquées, les ED commençaient à gouter les inconvénients de leurs ancêtres. Pour remédier à cet obstacle, les spécialistes ont dirigé leurs recherches aux techniques d'optimisation. Suite aux résultats de recherches, le milieu des méthodes d'optimisation des ED a connu un énorme progrès, ainsi qu'une diversité qui touchaient les différents supports physiques et logiques. De plus, elles étaient rapidement adoptées par les plus célèbres entreprises des BD.

La variété des structures d'optimisation a conduit à leur classement selon la redondance liée à leur implémentation sur le support physique. Ainsi, si une structure n'engendre pas un coût de stockage ou de maintenance supplémentaire, alors la structure est dite non-redondante (par exemple : la fragmentation horizontale, l'ordonnancement des requêtes et le traitement parallèle).

Dans le cas contraire, elle est dite redondante (par exemple : la fragmentation verticale, les index et les vues matérialisées). [1]

Cependant, la quantité impressionnante des techniques d'optimisation conduit au labyrinthe de choix qui doit impérativement prendre en considération les ressources matérielles des utilisateurs des BD, et puisque la sélection est une étape très délicate, elle peut conduire à l'élection d'une seule structure (sélection isolée) ou plusieurs (sélection multiple).

Ce mémoire traitera la problématique d'optimisation des requêtes de jointure en étoile dans les entrepôts de données en adoptant une sélection isolée d'une technique redondante : les vues matérialisées, en se basant sur le plan d'exécution estimé par le SGBD (Système de Gestion de Base de Données) hôte utilisé par le décideur (ou administrateur) à distance. Le résultat sera implémenté dans une interface web qui offrira, de plus, quelques outils supplémentaires aidant à mieux visualiser le plan d'exécution des requêtes définies.

L'organisation de ce mémoire sera comme suite : Dans le chapitre 2, nous ajustons le jargon technique du domaine des BD en définissant les concepts de base et présenter les différentes solutions possibles pour résoudre la problématique posée avec les illustrations et éclaircissements nécessaires. Le chapitre 3 touchera la partie théorique de notre travail en expliquant les difficultés déjà rencontrées par les pionniers du domaine d'optimisation des requêtes SQL, les arguments derrière notre choix et ses principes (fonctionnement) et les résultats obtenus suite à l'adoption de ce choix. Le chapitre 4 abordera tout ce qui concerne la pratique, notamment : mise en place de l'interface web, les outils utilisés, ses services, son mode d'emploi ainsi qu'une brève description des algorithmes implémentant cette interface. Enfin, le chapitre 5 sera une conclusion qui englobera la présentation du bilan du projet et les futures perspectives pour améliorer le travail réalisé.

Chapitre II : Etat de l'art et problématique

Sommaire

1 Introduction

L'étude de ce domaine informatique si vaste conduit impérativement vers l'ajustement du jargon qui sera utilisé tout au long de ce mémoire. Ainsi, il servira comme ouverture du problème posé en décrivant les différentes manières possibles pour le résoudre.

2 Définitions et notions

2.1 Base de données

2.1.1 Définition [2]

Une Base de données est un gros ensemble d'informations structurées mémorisées sur un support permanent.

2.1.2 Requête SQL

L'interaction avec la BD se fait par le biais d'une requête. Cette dernière est le moyen d'acheminement des ordres utilisateur vers les informations stockées dans la BD. Ainsi, il peut les consulter avec des critères (conditions) bien précises, les modifier (mettre à jour), supprimer ou d'en ajouter autres. Les requêtes sont généralement simples à comprendre et faciles à utiliser, ils constituent le point de départ vers le résultat (ou action) souhaité. L'exécution des requêtes SQL nécessite leur analyse et optimisation par la BD.

2.1.3 Système de gestion d'une base de données (SGBD) [9]

Un SGBD peut être perçu comme un ensemble de logiciels systèmes permettant aux utilisateurs d'insérer, de modifier et de rechercher efficacement des données spécifiques dans une grande masse d'informations partagée entre plusieurs utilisateurs. Un SGBD est un outil informatique qui permet la sauvegarde, l'interrogation, la recherche et la mise en forme de données

stockées sur mémoires secondaires. Ce sont des fonctions premières, complétées par des fonctions souvent plus complexes destinés par exemple à assurer le partage de données mais aussi à protéger les données contre tout incident et à obtenir des performances acceptables. La figure 1 illustre la composition d'un SGBD.

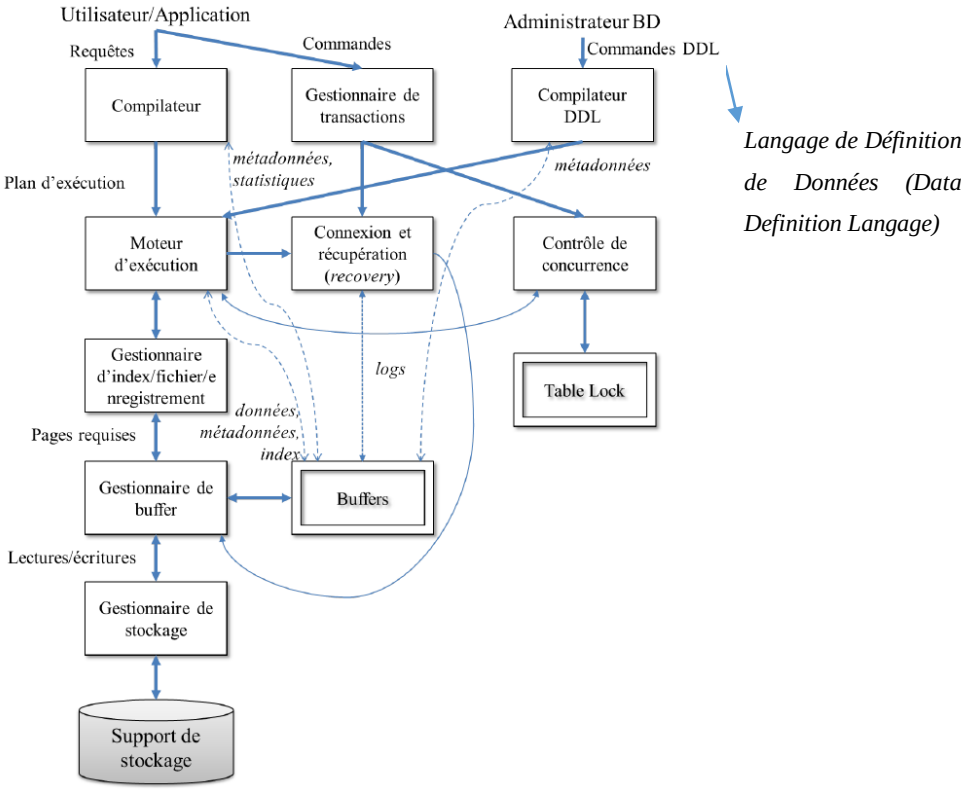


Figure 1 : Schéma détaillant les composants d'un SGBD relationnel

2.1.4 Plan d'exécution d'une requête

Le plan d'exécution montre les étapes détaillées nécessaires pour exécuter une requête SQL. Ces étapes sont générées par le traitement des opérateurs contenus dans la requête (sélection, jointure, projection, trie, …etc.) [3]. En fait, avant chaque exécution d'une requête SQL, l'optimiseur des requêtes fait une estimation des coûts des différentes façons d'exécuter une requête. Le plan d'exécution avec le coût estimé minimal de toutes les possibilités est choisi comme plan de réalisation.

La consultation du schéma du meilleur (le moins chère) plan d'exécution d'une requête peut se faire avec l'utilisation du package DBMS_XPLAN (sous le SGBD du Oracle). La représentation obtenue est tabulaire, facile à interpréter. Elle montre les indexes qui seront utilisés, les tables qui seront scannées entièrement, l'enchaînement d'exécution de la requête…etc. Cependant, elle ne permet pas de comparer le temps réel d'exécution avec le temps estimé.

```
EXPLAIN PLAN FOR
    SELECT p.nom, p.prenom FROM PERSONNE p, PASSER a
        WHERE a.numero=p.numero ORDER BY p.nom;

SELECT * FROM TABLE (dbms_xplan.display);
```

Figure 2 : Exemple d'une demande de consultation du plan d'exécution

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 546615235
------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     6 |   120 |     6  (34)| 00:00:01 |
|   1 |  SORT ORDER BY      |          |     6 |   120 |     6  (34)| 00:00:01 |
|*  2 |   HASH JOIN         |          |     6 |   120 |     5  (20)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| PERSONNE |     5 |    85 |     2   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| PASSER   |     6 |    18 |     2   (0)| 00:00:01 |
------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("A"."NUMERO"="P"."NUMERO")
16 rows selected.
```

Figure 3 : Exemple d'un plan d'exécution sur la ligne de commandes Oracle

2.1.5 Types de bases de données

Dans le domaine d'organisation et traitement d'informations. On distingue deux types de bases de données : [4]

● OLTP (On-Line Transaction Processing) : Utilisées pour des besoins opérationnels, optimisés pour traitement d'un petit volume de données. Ce type de systèmes permet uniquement la création

des rapports standards pour les documents, sans analyse. Ce type est adopté par les BD classiques.

● OLAP (On-Line Analytical Processing) : Permettent de traiter un grand volume de données, ainsi leur historisation pour l'analyse des anomalies (problèmes) ou autres situations (schémas de comportement par exemple). Ce type est adopté par les entrepôts de données.

Critère	OLTP (BD Classiques)	OLAP (Entrepôt de Données)
Détails des données	Grand	Moyen ou petit
Type d'accès aux données	Lecture ou écriture	Lecture uniquement
Consolidation des données	Petite	Grande
Répétition des données	Grande	Petite
Fréquence d'opérations sur les données	Très grande	Moyenne ou petite
Horizon temporel	Très court ou court	Long (données historiques)
Taille des données	Mo, Go	Go, To
Nombre des tuples touchés par une opération	Dizaines	Millions

Table 1 : Tableau comparatif des caractéristiques entre OLTP et OLAP. [5]

2.2 Entrepôts de données

2.2.1 Définition [6]

Un entrepôt de données est une collection de données intégrées, non volatiles et historiées. Dédiés aux applications d'analyse et de prise de décision. Le processus d'analyse est réalisé à l'aide de requêtes complexes comportant de multiples jointures et des opérations d'agrégation sur des tables volumineuses induisant un temps de réponse très élevé. Il permet de fournir un accès permanent aux données même lorsque les bases de données individuelles sont inaccessibles.

2.2.2 Schéma en étoile [6]

En général, les ED sont conçus comme des bases relationnelles, sous forme du schéma en étoile. Dans ce schéma, les mesures sont représentées par une table de faits (étant la base des analyses) et chaque dimension par une table de dimensions. La table des faits référence les tables de dimensions en utilisant une clé étrangère pour chacune d'elles. Autour de cette table des faits figurent les tables de dimensions qui regroupent les caractéristiques des dimensions. La table des faits peut atteindre une taille importante par rapport au nombre de n-uplets.

Ce type schéma se caractérise par : [7]

- Structure simple, alors le schéma est plus simple à comprendre.
- Grande efficacité des requêtes, grâce au petit nombre de liaisons entre les tables.
- Structure dominante pour les ED.

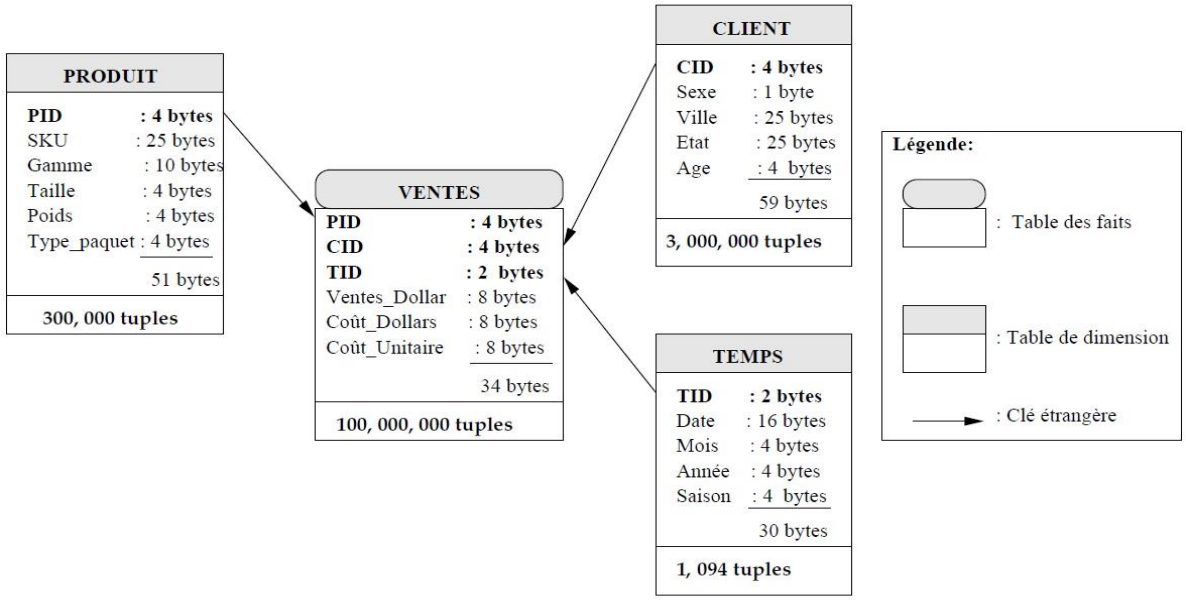


Figure 4 : Un exemple d'un schéma en étoile

2.2.3 Requêtes de jointure en étoile

Les requêtes typiques de ce schéma sont appelées les requêtes de jointure en étoile (star-join queries) qui ont les caractéristiques suivantes :

1. Il y a des jointures multiples entre la table des faits et les tables de dimension.

2. Il n'y a pas de jointure entre les tables de dimensions.

3. Chaque table de dimension impliquée dans une opération de jointure a plusieurs prédicats de sélection sur ses attributs descriptifs.

La syntaxe générale de ces requêtes est la suivante :

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

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

WHERE <Liste de prédicats de sélection & jointure>

[GROUP BY <Liste des attributs de tables de dimension>]

[ORDER BY <Liste des attributs de tables de dimension>]

2.3 Structures d'optimisation

2.3.1 Définition

Les structures d'optimisation (SO) sont des techniques utilisées dans le but de réduire (minimiser) le coût (temps d'exécution) de la requête. Pour chaque structure il y en a une large panoplie d'algorithmes et approches adoptées. Du coup, une SO peut différer d'un SGBD à autre, elle est définie selon le type d'implémentation sur le support de stockage et l'algorithme utilisé pour cette approche.

2.3.2 Catégories des structures d'optimisation

2.3.2.1 Techniques d'optimisation matérielles

Comme leur nom indique, ces techniques sont souvent offertes par les plateformes de déploiement et leurs systèmes de stockage. On peut citer par exemple les optimisations suivantes :

- Utilisation des supports de stockage plus rapides (Disque-dur).
- L'adoption du traitement parallèle en utilisant des processeurs supplémentaires.
- L'augmentation de la taille des caches à tous les niveaux.

2.3.2.2 Techniques d'optimisation logicielles

Elles aussi, sont classées selon la redondance liée à leur implémentation sur le support physique. Ainsi, si une structure n'engendre pas un coût de stockage ou de maintenance supplémentaire, alors la structure est dite non-redondante. Dans le cas contraire, elle est dite redondante.

A noter que cette propriété de redondance n'est en aucun cas le critère décisif pour le choix de la technique d'optimisation.

2.3.2.2.1 Structures redondantes

● **Index :** Un objet complémentaire à la base de données permettant d'indexer certaines colonnes dans le but d'améliorer l'accès aux données par le SGBD, au même titre qu'un index dans un livre permet souvent d'économiser du temps lorsqu'on recherche une partie spécifique dans ce dernier. En absence de l'index utilisable, une requête portant sur une table nécessitera le balayage total de la table. Toutefois les index sont très gourmands en espace de stockage.

● **Vues matérialisées (VM) :** Une vue ordinaire stocke le résultat d'une requête en mémoire

centrale comme une table virtuelle. Par contre, une vue matérialisée est une vue stockée d'une façon permanente sur un support de stockage non volatile (disque-dur). Les vues matérialisées sont très utilisées dans le cadre des entrepôts de données. Les requêtes peuvent accéder indirectement aux données de l'entrepôt via le mécanisme de réécriture de requêtes en sollicitant ces vues. Les vues matérialisées nécessitent un coût supplémentaire pour leur maintenance. Une bonne conception des vues peut améliorer considérablement le temps d'accès des requêtes utilisateurs.

● **Fragmentation verticale :** Elle consiste à diviser un schéma de données S en un sous-ensemble de schémas. Les sous-schémas sont obtenus par des opérations de projection selon un ou plusieurs attributs. L'objectif principal de la fragmentation verticale est de réduire la taille des données chargées en mémoire centrale pour répondre à une requête en regroupant les attributs fréquemment utilisés ensemble dans le même fragment. La fragmentation verticale, crée des fragments avec des tuples de longueur réduite ce qui permet de lire un nombre minime de page disque pour exécuter une requête.

● **Gestion du Buffer :** Consiste à exploiter l'espace dédié au SGBD, contenu dans la RAM (nommé : result-cache) en stockant temporairement des sous-requêtes élues. Le cache garantit un accès instantané (compté en nanosecondes) à son contenu. Vu que la mémoire primaire est volatile, cette technique reste limitée et appliquée uniquement quand la machine est sous tension.

2.3.2.2.2 Structures non-redondantes

● **Fragmentation horizontale :** Consiste à découper une table en plusieurs sous-ensembles disjoints de tuples. Cette technique vise à optimiser les opérations de sélection en réduisant ainsi la taille des résultats intermédiaires qui sont souvent coûteuses dans les bases de données volumineuses.

● **Ordonnancement des requêtes et parallélisme :** En analysant les corps d'une série des requêtes SQL, on identifie et réordonne les requêtes qui peuvent être exécutées dès que leurs opérandes sont

disponibles. Autrement dit, Si une requête A n'utilise pas le résultat d'une requête B et vice-versa, alors A et B peuvent être exécutées en parallèle.

2.3.3 Problème de sélection

La variété des structures d'optimisation a mis les concepteurs devant un problème du choix des SO. Est-ce que ces techniques sont indépendantes, ou il existe une façon de combiner plusieurs au bénéfice du coût des requêtes. Autrement dit, sa résolution serait la réponse à la question : *"De quelle manière choisir l'ensemble des SO pour tirer le meilleur d'elles ?"*

2.3.3.1 Sélection isolée

Constitue le cas le plus simple, il s'agit d'utiliser (sélectionner) **une seule** structure pour l'optimisation. Ainsi, toutes les SO citées ci-dessus, prises individuellement, forment une sélection isolée.

2.3.3.2 Sélection multiple

Elle est définie par l'utilisation d'**au moins deux** structures dans la procédure de l'optimisation. Cependant, l'élection de l'ensemble candidat de ces structures est un problème en lui-même (pour plus de détails, voir le chapitre 3).

3 Problématique

Ce travail se concentre sur une sélection isolée redondante d'une structure particulière : la VM. Pour cela, le prochain chapitre sera consacré à la partie théorique liée à ce choix, contenant des explications, arguments et principes de fonctionnement.

Chapitre III : Etude de la problématique et propositions

Sommaire

1 Introduction :

Dans le but de faciliter l'utilisation des SGBD, ces derniers exécutent les commandes reçues de façon transparente à l'utilisateur. Cependant, derrière cette transparence se cachent des algorithmes implémentés dans le but de résoudre une liste de problèmes assez complexes. En effet, l'exécution d'une requête passe par un moteur d'optimisation permettant de trouver –selon lui– la "route" la plus rapide et la moins coûteuse pour arriver aux résultats attendues, ainsi cela diffère d'un moteur à autre. Cette route est le plan d'exécution estimé par le SGBD utilisé.

2 Solution proposée :

2.1 Problème NP-complet :

En parlant dans le domaine de la complexité, les spécialistes ont monté que le choix du plan d'exécution d'une requête est un problème NP-complet [1].

Selon Stephen Cook, un problème NP-complet est un problème de décision vérifiant les propriétés suivantes : [8]

● Il est possible de vérifier une solution efficacement (en temps polynomial) ; la classe des problèmes vérifiant cette propriété est notée NP.

● Est un problème NP-difficile, i.e. le problème est au moins aussi difficile que tous les autres problèmes de la classe NP.

Ainsi, tous les algorithmes connus pour résoudre des problèmes NP-complets ont un temps d'exécution exponentiel en la taille des données d'entrée dans le pire des cas.

2.2 Optimisation des requêtes SQL :

Vu que la résolution du problème de choix de la meilleure façon d'exécuter une requête SQL prend un temps exponentiel, du coup, le choix de la structure d'optimisation adéquate (la plus

performante, la moins coûteuse en espace et temps d'exécution) est lié à un problème NP-complet. L'approche proposée dans ce projet se penche sur une sélection isolée (une seule SO) d'une technique redondante (occupante un espace disque) : la sauvegarde des jointures en étoiles communes entre les requêtes en utilisant les vues matérialisées (VM).

2.3 Arguments de choix des VM :

Chaque idée, approche ou solution proposée possède ses motifs. Ainsi, le choix des VM comme structure d'optimisation pour ce cas d'étude n'est pas fait au hasard.

En effet, les mémoires secondaires (par exemple : disque-dur, flash-disk, carte mémoire ou CD) sont non-volatiles gardant ainsi les données reçues même si le périphérique (le matériel équipé par cette mémoire) est éteint. Et, c'est justement la caractéristique la plus importante à considérer dans les BD en général et les ED en particuliers puisqu'ils possèdent des tables avec des millions de tuples. Ainsi, pour optimiser l'accès régulier aux informations stockées dans les ED c'est nécessaire de choisir une solution durable (dans le temps), qui accélère les accès et prend en considération les différents changements liés aux tuples, apportées dans l'ED (ajouts, suppressions ou modifications), et justement, les VM sont des structures flexibles (qui s'adaptent aux changements).

Cependant, il est clair que le stockage d'une partie des requêtes peut diminuer considérablement l'espace de stockage disponible dans le périphérique utilisant le SGBD (et souhaitant bénéficier de la solution proposée dans ce projet). Mais, de l'autre côté, l'espace vient le deuxième à considérer dans les choix des techniques d'optimisation, c'est le temps d'exécution (accès) qui est le plus important. De plus, l'espace ne pose pas un grand problème dans nos jours (en comparant au dizaines d'années passées).

Enfin, il est sûr que le choix d'une seule SO n'est pas la meilleure approche à proposer, en la comparant par le choix d'au moins deux SO, mais ça reste toujours un cas intéressant à étudier.

2.4 Conventions adoptées :

La définition de l'approche adoptée pour optimiser les requêtes ne suffit pas. En effet, la diversité des schémas des BD et les différentes façons d'écrire la même requête est un facteur assez compliqué à prendre en compte, pouvant augmenter rapidement la complexité de l'approche proposée. Pour cela, une liste de conventions (hypothèses de travail) a été établie afin de cibler un cas particulier dans ce projet.

● Toutes les requêtes introduites sont des requêtes de jointure en étoile (consultation d'un ED avec schéma en étoile).

● Pas de sélections imbriquées. i.e. Une requête contient un seul "SELECT" (les jointures se trouvent dans la section "WHERE").

● Deux prédicats sont équivalents si et seulement s'ils sont égaux (la même chaine de caractères). Ainsi, le prédicat "num IN (1, 2, 3)" est différent du prédicat "num BEWEEN 1 AND 3".

2.5 Principes de fonctionnement :

L'approche proposée se base sur l'exploitation du plan d'exécution du SGBD. Ce dernier, il est libre (en fonction des algorithmes implémentés) de choisir les opérations (actions : jointure, sélection, projection, …) à effectuer pour exécuter la requête entrée. Ainsi, une requête peut contenir une jointure qui sera omise dans le plan d'exécution proposé car elle est jugée "inutile" (n'a pas un impact sur le résultat, de plus, elle augmente le temps d'exécution estimé) (Figure 5)

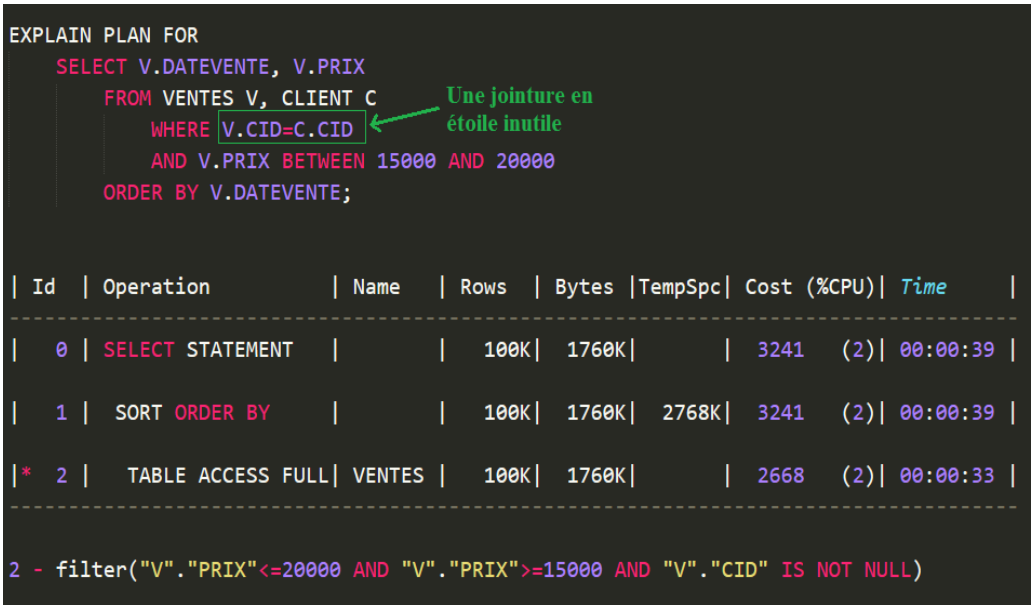


Figure 5 : Exemple d'une jointure en étoile non prise en compte

Par conséquent, l'identification des jointures en étoile est faite directement depuis le plan d'exécution et **non pas** du format textuel (commande : SELECT … FROM …) de la requête concernée (Figure 6)

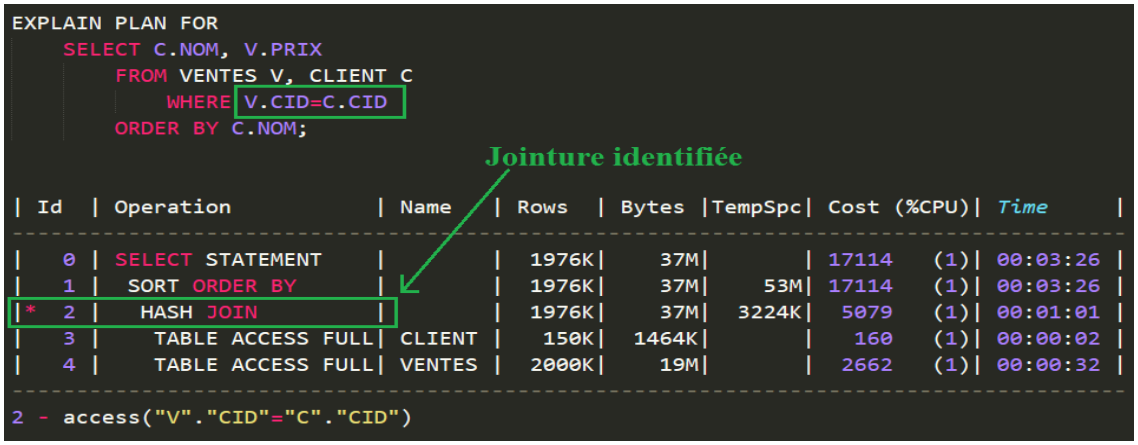


Figure 6 : Identification d'une jointure en étoile à partir du plan d'exécution

Donc, la VM définie contiendra les tables impliquées dans la jointure identifiée. Ses attributs seront les identifiants de ces tables et l'ensemble des attributs sélectionnés par les requêtes impliquées dans cette jointure (qui ont la même jointure) appartenant à une des tables de la VM définie (Figure 7) (Pour plus de détails : voir Chapitre 4).

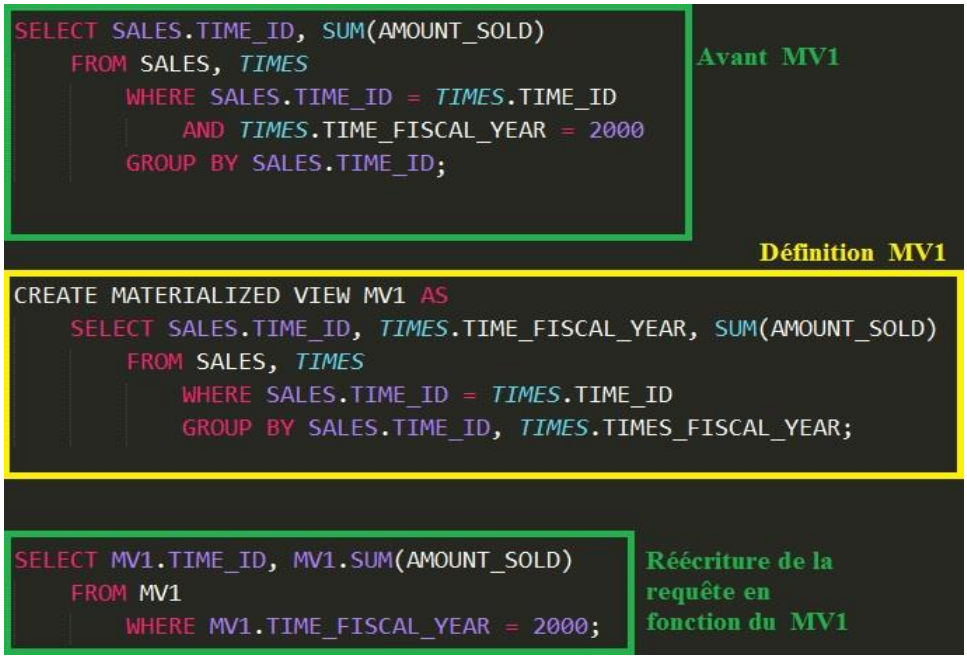


Figure 7 : Exemple d'optimisation d'une requête de jointure en étoile avec une VM

Chapitre IV : Outil d'aide à l'optimisation

Sommaire

1 Présentation des logiciels et langages utilisés :

Le côté pratique de ce projet est basé sur une interface web, permettant aux administrateurs et concepteurs de se connecter à distance à leur base de données, en passant par une petite configuration du modem et le SGBD lui-même (voir Figure 10), d'entrer les requêtes de jointure en étoile (voir Figure 12), visualiser le plan d'exécution proposé par le moteur du SGBD ainsi que l'arbre algébrique (voir Figure 13 et 14) et, surtout, optimiser les requêtes SQL en utilisant les vues matérialisées (voir figure 15).

Le diagramme UML : use-cases (Figure 8) explique toutes les possibilités et options réalisées pour cette interface.

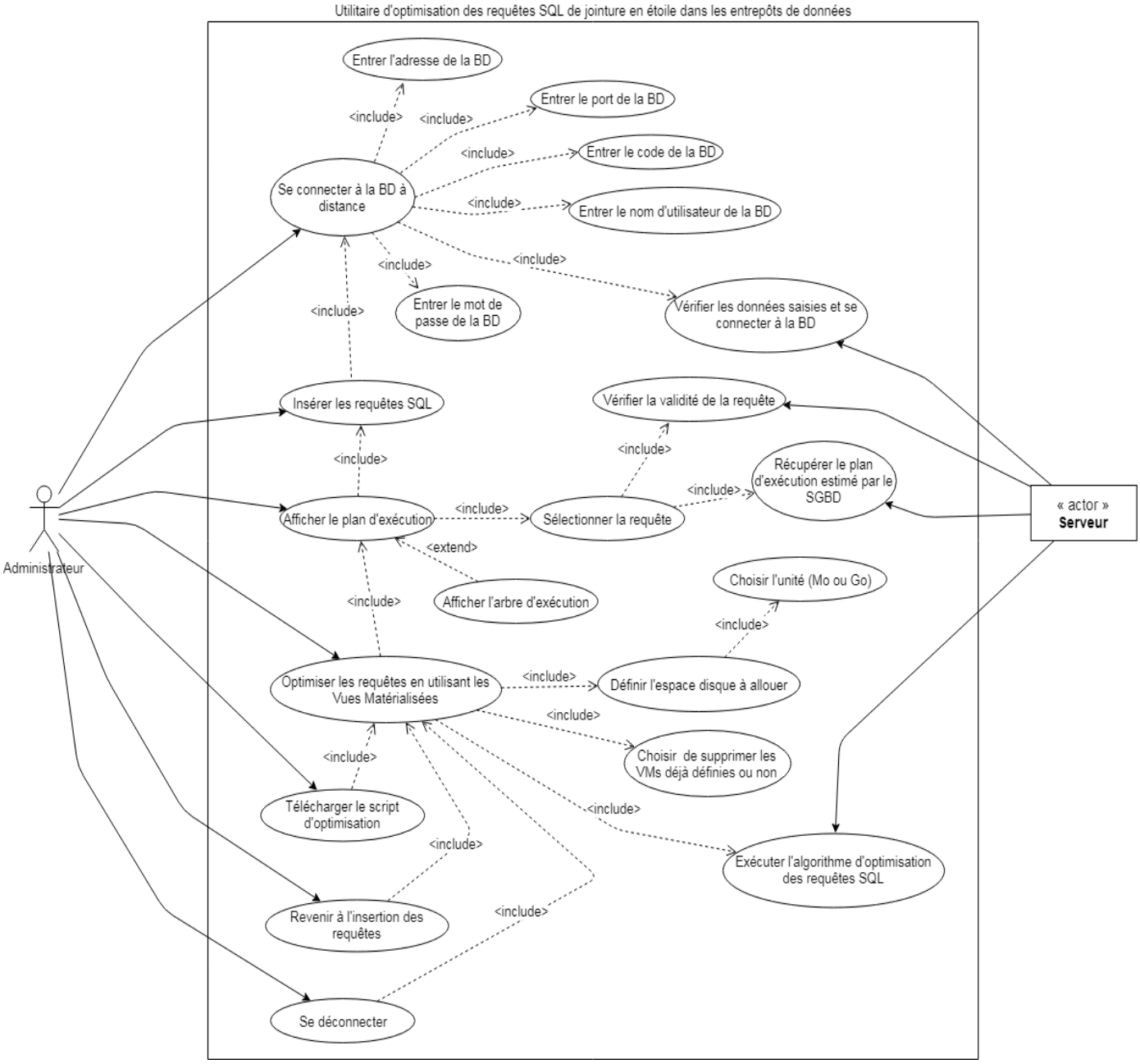


Figure 8 : Diagramme use-cases de l'outil d'aide conçu

En ce qui concerne les logiciels, langages, Frameworks ou APIs utilisées, le tableau ci-dessous met en évidence tout ce qui a servi à réaliser à cette interface web.

Nom	Description
HTML, CSS et JavaScript	Pour la structuration de la page WEB et animations (côté client).
PHP	Pour les algorithmes écrits, c'est la base de ce projet (côté serveur).
Oracle 11g2 XE (Express Edition)	Pour la connexion à distance, extraction des plans d'exécution et évaluation des performances des requêtes SQL.
WampServer	Pour simulation d'un serveur local, et donc exécution du code PHP.
Google Chrome et Mozilla Firefox	Pour les tests réels de l'interface web réalisée. La recherche des erreurs, bugs ou anomalies.

2 Outil d'optimisation

2.1 Configuration de la base de données :

L'utilisation des services offerts par le site web nécessite une étape primaire qui consiste à configurer son SGBD et PC pour qu'il puisse se communiquer à travers l'internet.

Premièrement, l'utilisateur doit installer le client (listener) offert par le constructeur de son SGBD qui va permettre d'écouter le port défini pour recevoir les requêtes SQL et transmettre le résultat.

Dans le cas d'Oracle, c'est "instantclient". Pour commencer l'écoute du port spécifié dans l'installation, il faudrait taper la commande "lsnrctl start" dans l'invite de commandes.

```
C:\Windows\system32>lsnrctl start

LSNRCTL for 32-bit Windows: Version 11.2.0.2.0 - Production on 08-MAI  -2018 21:
34:31

Copyright (c) 1991, 2010, Oracle.  All rights reserved.

TNS-01106: Listener using listener name LISTENER has already been started
```

Figure 9 : Démarrage de l'écouteur (listener) du port du SGBD

Ensuite, vient l'étape de la configuration des ports TCP/UDP pour que le modem redirige automatiquement les accès extérieurs (avec l'adresse IP publique) au port défini dans le "listener" vers l'adresse IP locale du SGBD de l'administrateur.

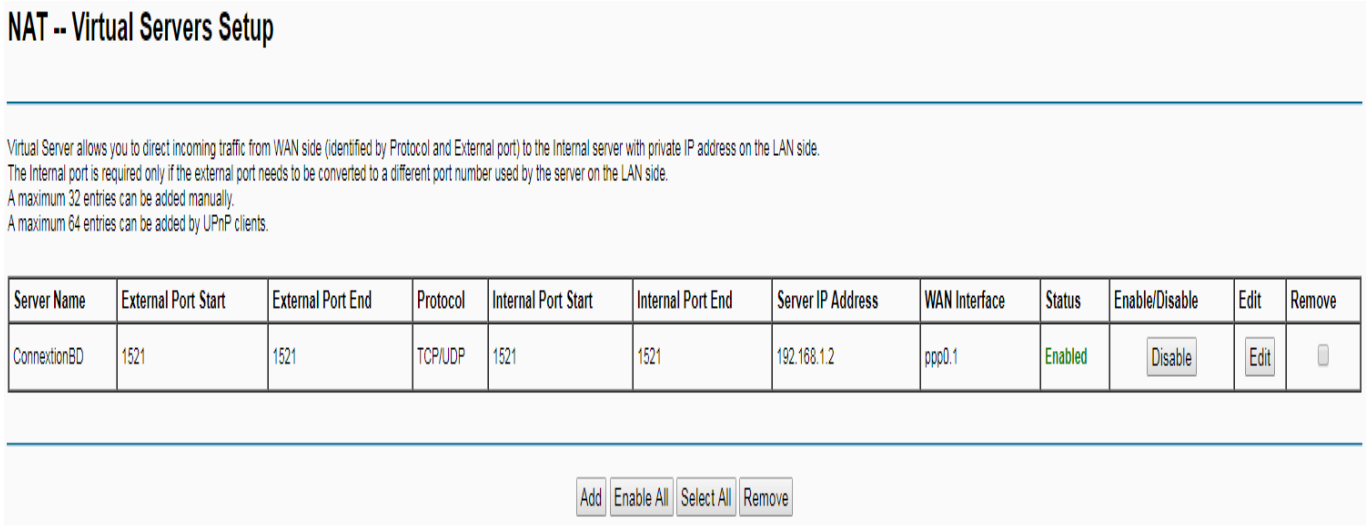


Figure 10 : Configuration des ports TCP/UDP dans le modem

2.2 Connexion avec la base de données :

La page principale (accueil) du site web se présente en forme d'un formulaire invitant l'utilisateur d'entrer les informations nécessaires pour établir la connexion avec sa base de données (à distance ou en local [uniquement pour les tests]).

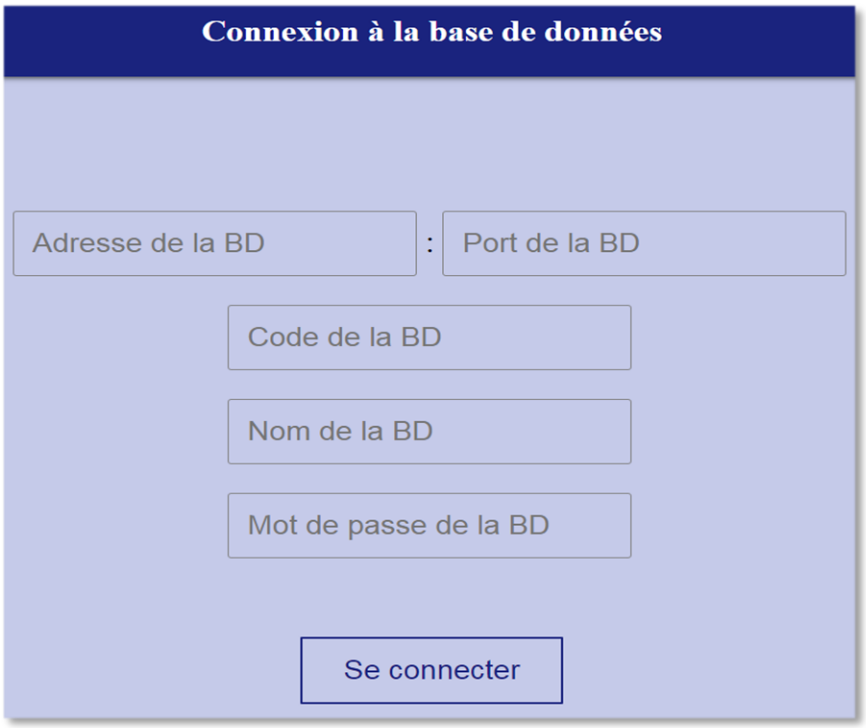


Figure 11 : Connexion à la base de données

Pour les informations demandées, nécessaires pour l'établissement de la connexion, sont :

● **Adresse de la BD :** C'est la donnée la plus importante, elle indique le chemin permettant d'accéder à la BD via internet. Le décideur est invité à saisir l'adresse IP publique (version 4) statique ou dynamique de son réseau (établie par son Fournisseur d'Accès Internet). Pour les tests sur un serveur local, il est possible d'entrer le mot "localhost".

● **Port de la BD :** Il s'agit du canal de communication entre le site web et le SGBD, la valeur par défaut est "1521" (qui peut être changée).

● **Code de la BD :** Indique le préfixe établi par le constructeur du SGBD, dans le cas d'Oracle, pour les éditions gratuites (avec différentes limitations) c'est "XE", et pour les versions business c'est 'ROOT'.

● **Nom de la BD :** Le nom d'utilisateur de la BD (qui doit déjà exister parmi la liste des utilisateurs du SGBD).

● **Mot de passe de la BD :** comme le nom l'indique, c'est le code secret défini pour l'utilisateur entré.

2.3 Insertion des requêtes SQL :

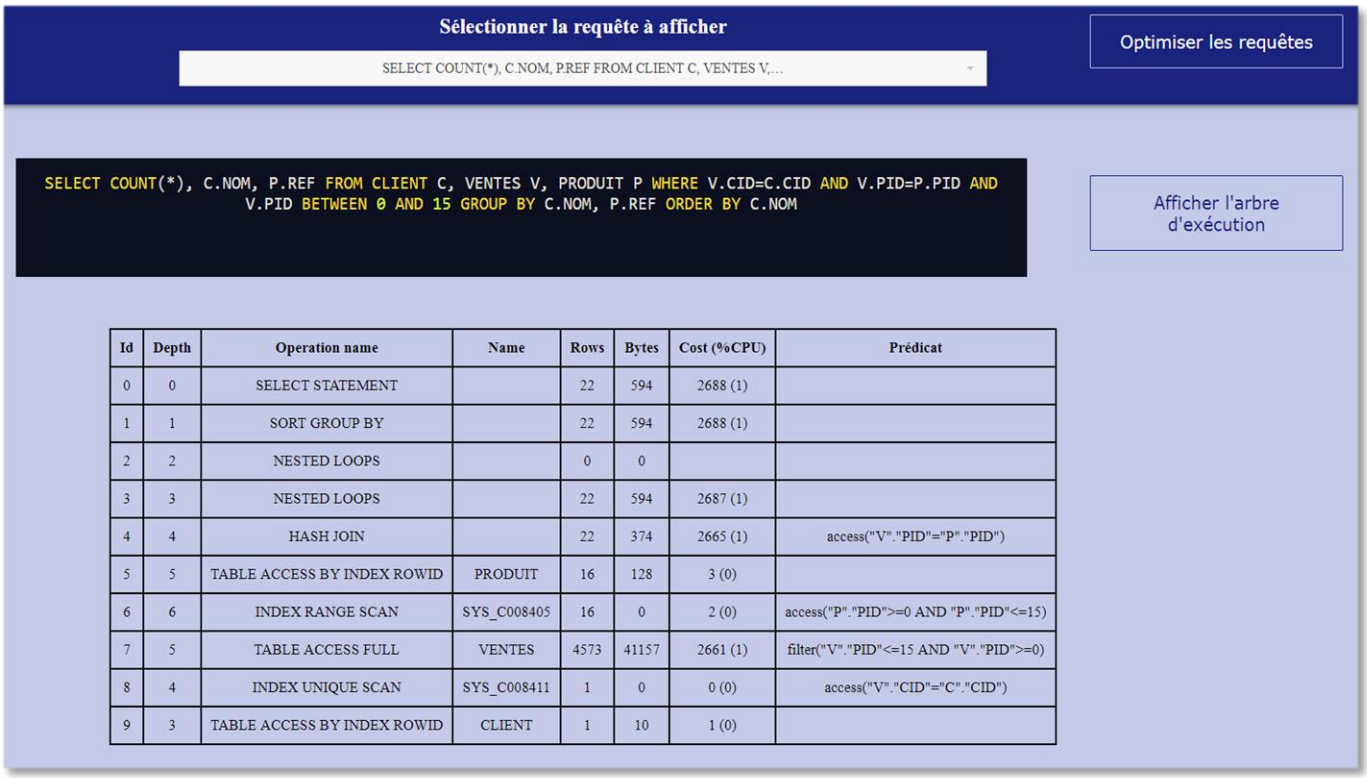
Après la vérification des informations saisies et l'établissement de la connexion avec succès au SGBD, une nouvelle page d'insertion des requêtes SQL et affichée. Les requêtes entrées au format textuel, sont séparées avec des points-virgules.

Figure 12 : Insertion des requêtes SQL

2.4 Affichage du plan et l'arbre d'exécution :

La validation des requêtes insérées conduit à la page de l'affichage du plan d'exécution. Ce dernier est récupéré directement depuis le SGBD hôte utilisé, mis dans une forme tabulaire afin de mieux visualiser le plan.

Pour bénéficier de cette fonctionnalité, l'utilisateur est invité à sélectionner la requête (identifiée par le numéro ou son corps) dans la liste déroulante. Ainsi, un algorithme est chargé de faire la récupération du plan et sa mise dans un tableau (Figure 14).



Id	Depth	Operation name	Name	Rows	Bytes	Cost (%CPU)	Prédicat
0	0	SELECT STATEMENT		22	594	2688 (1)	
1	1	SORT GROUP BY		22	594	2688 (1)	
2	2	NESTED LOOPS		0	0		
3	3	NESTED LOOPS		22	594	2687 (1)	
4	4	HASH JOIN		22	374	2665 (1)	access("V"."PID"="P"."PID")
5	5	TABLE ACCESS BY INDEX ROWID	PRODUIT	16	128	3 (0)	
6	6	INDEX RANGE SCAN	SYS_C008405	16	0	2 (0)	access("P"."PID">=0 AND "P"."PID"<=15)
7	5	TABLE ACCESS FULL	VENTES	4573	41157	2661 (1)	filter("V"."PID"<=15 AND "V"."PID">=0)
8	4	INDEX UNIQUE SCAN	SYS_C008411	1	0	0 (0)	access("V"."CID"="C"."CID")
9	3	TABLE ACCESS BY INDEX ROWID	CLIENT	1	10	1 (0)	

Figure 13 : Affichage du plan d'exécution d'une requête choisie

De plus, le site web offre la possibilité de visualiser l'arbre d'exécution de la requête courante (Figure 15). Cet arbre est construit en suivant les différentes profondeurs de nœuds qui composent le plan d'exécution. La racine étant le SELECT.

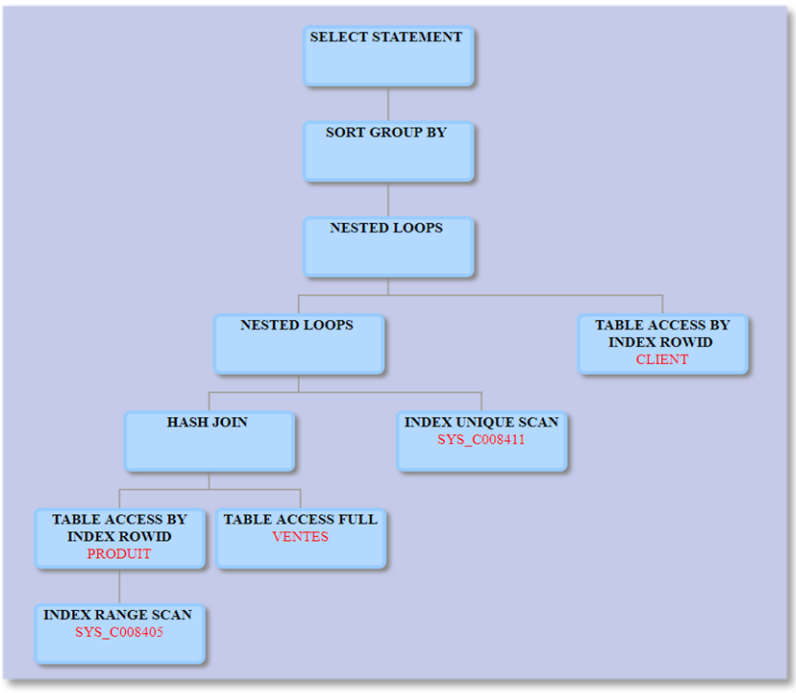


Figure 14 : Affichage de l'arbre d'exécution d'une requête choisie

2.5 Optimisation des requêtes :

Finalement vient la dernière étape, et la plus importante, il s'agit du l'optimiseur des requêtes SQL.

Avant de passer à la définition des VM, l'utilisateur doit spécifier l'espace alloué aux VM (au choix : en Mo ou Go) ainsi que la suppression des VM déjà définies ou non.

La définition des VM prend quelques instants (elle dépend du nombre des requêtes entrées et les nombre de sous arbres de jointure dans chaque requête). Le résultat d'optimisation est affiché sous forme de barres statistiques (Figure 16) qui comparent le coût d'exécution de la requête (à partir du plan d'exécution estimé par le SGBD hôte) avant et après l'optimisation.

A noter que la définition d'une VM est faite si et seulement si :

- Sa taille ne dépasse pas la taille disque définie par l'utilisateur.
- Il existe **au moins** deux sous arbres de jointures (issus de deux requêtes) communs.

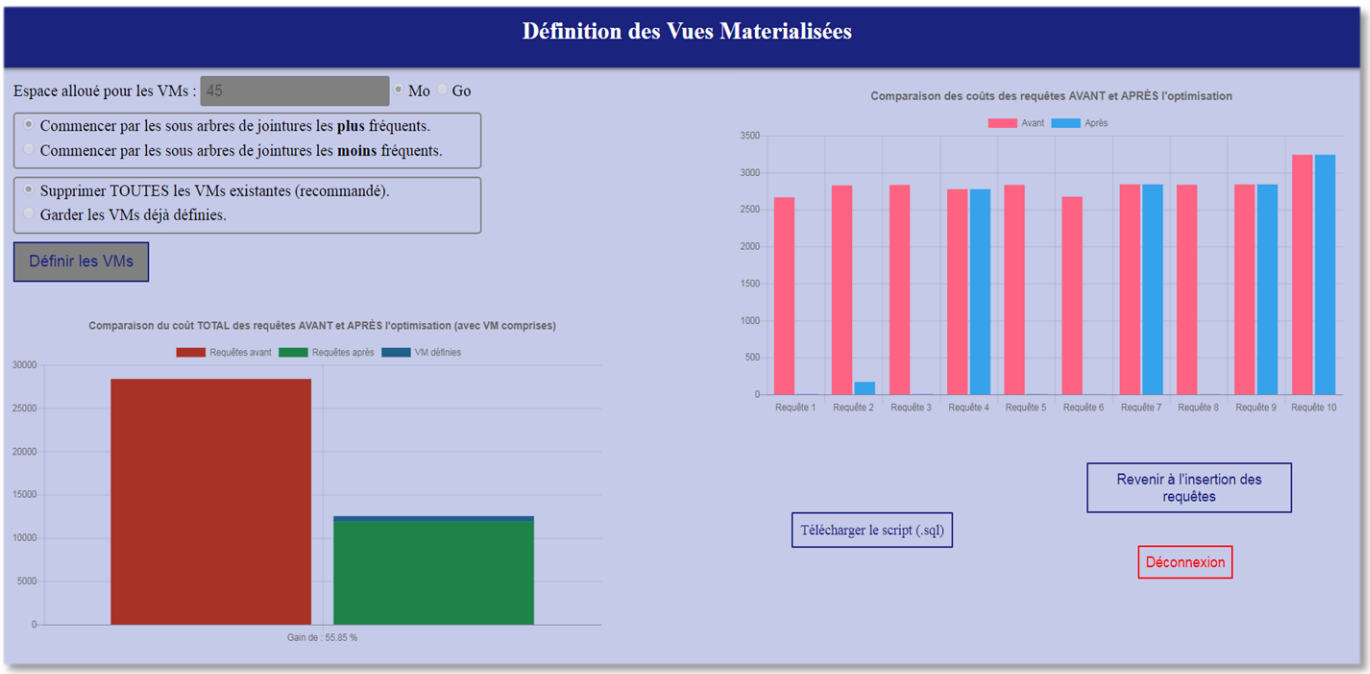


Figure 15 : Optimisation des requêtes avec les VM

Après l'optimisation, trois boutons s'affichent :

- **Télécharger le script (.sql) :** Obtenir le script d'optimisation. Un fichier texte contenant les

requêtes entrées par l'utilisateur (avant l'optimisation), les VM définies (durant l'optimisation) et les requêtes réécrites en fonction de ces VM (après l'optimisation).

● **Revenir à l'insertion des requêtes :** Pour optimiser davantage de requêtes.

● **Déconnexion :** Revenir à la page d'accueil (terminer la session actuelle).

L'algorithme de l'optimisation constitue la plus grande partie de cette interface web. Il utilise le plan d'exécution estimé offert par le SGBD hôte utilisé, ainsi, tous les traitements se font sur ce dernier, il constitue l'entrée la plus importante (avec la requête au format textuel). Les instructions de ce traitement sont divisées en trois grandes parties :

● **Récupération des sous arbres de jointure de chaque requête valide :** Une requête de jointure en étoile peut contenir plusieurs jointures (puisqu'un ED contient plusieurs tables de dimension). De ce fait, il est impératif de récupérer chaque jointure et ses composants (tables et prédicats) en groupe, nommé : sous arbre de jointure. A la fin de cette étape, on obtient pour chaque requête valide (par le SGBD), tous les sous arbres de jointure candidats (qui peuvent être optimisés par la suite). Le pseudo-algorithme associé :

```
POUR CHAQUE requête valide FAIRE
        Créer une liste des ensembles (vide);
        POUR CHAQUE nœud du plan d'exécution FAIRE
                SI nœud de jointure ALORS
                        Récupérer l'ensemble des nœuds descendants;
                        Ajouter l'ensemble récupéré à la liste des ensembles;
                FSI;
        FAIT;
FAIT;
```

● **Création de la liste des sous arbres de jointure communes :** Comme mentionné dans le chapitre 3, la SO en utilisant les VM se base sur les jointures communes entre plusieurs requêtes (au moins deux). Du coup, cette phase est une continuation de celle précédente. Autrement dit, on récupère les ensembles (sous arbres de jointure) qui apparaissent dans au moins deux requêtes. On juge que deux sous arbres de jointures sont communs si et seulement si : ils ont les mêmes nœuds au mêmes endroits (même profondeur) avec le même nom de table et même prédicat.

A la fin, on aura comme résultat toutes les jointures avec l'ensemble des requêtes dans lesquelles cette jointure apparaisse. Le pseudo-algorithme :

```
POUR CHAQUE requête (r1) valide FAIRE
    POUR CHAQUE j1←sous arbre de jointure de (r1) FAIRE
        POUR CHAQUE requête (r2) différente de la requête courante FAIRE
            POUR CHAQUE j2←sous arbre de jointure de (r2) FAIRE
                SI j1 et j2 sont égaux ALORS
                    Ajouter (r2) à une liste temporaire;
                    Supprimer (j2); /* Pour ne pas la parcourir une autre fois */
                FSI;
            FAIT;
        FAIT;
    FAIT;
    SI la liste temporaire n'est pas vide ALORS /* (j1) existe au moins dans une autre
requête */
        Ensemble de jointures communes [sous arbres] ← j1;
        Ajouter (r1) à la liste temporaire;
        Ensemble de jointures communes [requêtes] ← liste temporaire;
    FSI;
FAIT;
```

● **Définition des VM :** Suite à la construction de l'ensemble des sous arbres de jointures communes et les requêtes associées, il est temps de construire les VM en prenant chaque jointure commune (avec les requêtes impliquées) et l'analyser pour définir la VM. A noter qu'une VM peut ne pas être définie si l'espace occupé par elle est supérieur à l'espace utilisé (et celui-ci doit être inférieur à l'espace disque alloué, défini précédemment par l'utilisateur). Ainsi, si l'espace utilisé atteint l'espace défini, la création des VM s'arrête. Le pseudo-algorithme associé :

```
POUR CHAUQE sous arbre de jointure commune FAIRE
    POUR CHAQUE requête ∈ sous arbre actuel FAIRE
        Analyser requête;
    FAIT;
    SI espace VM ≤ espace utilisé ALORS
        Définir la VM ;
        Espace utilisé += Espace VM définie;
        SI espace utilisé > espace défini ALORS
            Espace utilisé -= Espace VM définie;
            Supprimer VM ;
        SINON SI espace utilisé = espace défini ALORS
                Arrêter la création des VM;
            FSI;
        FSI;
    FSI;
FAIT;
```

● **Réécriture des requêtes en fonction des VM définies :** La dernière étape de l'optimisation est le fruit des VM définies. La réécriture des requêtes consiste à remplacer les tables utilisées dans chaque requête appartenant au sous arbre de jointure possédant une VM définie avec cette dernière en supprimant les prédicats (qui appartiennent à la VM) de cette requête. Ceci est assuré par un traitement de chaines de caractères. Une requête peut ne pas être réécrite si elle n'a pas une jointure en commun avec au moins une autre requête. Le pseudo-algorithme :

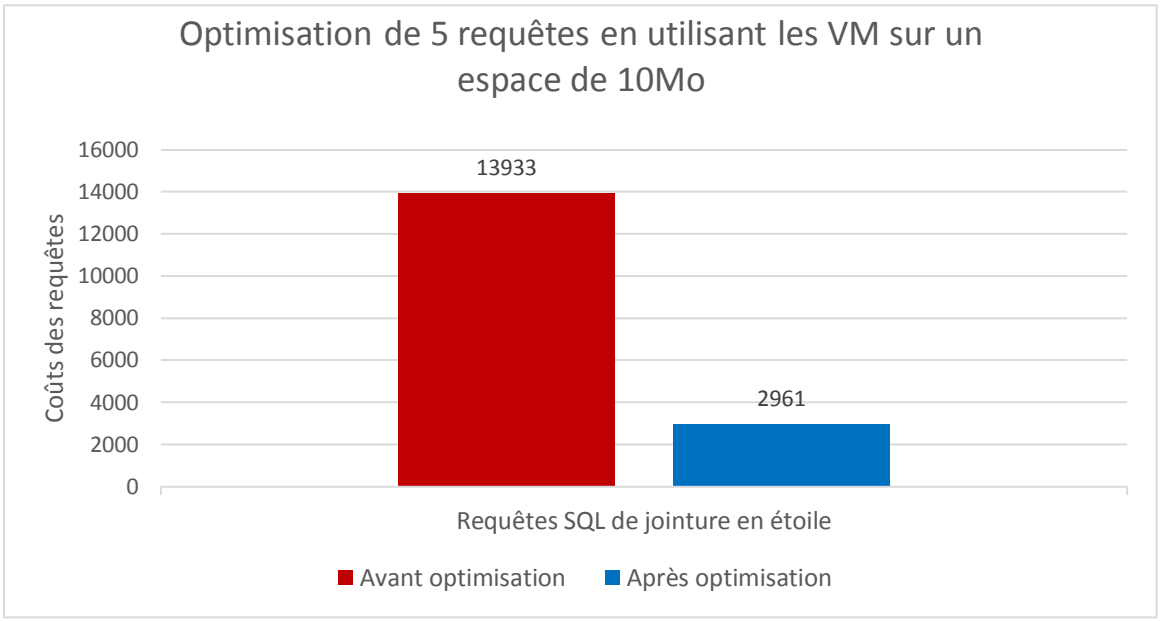
POUR CHAQUE sous arbre de jointure commune FAIRE
    Récupérer la VM définie;
    SI VM non-définie ALORS passer à la jointure suivante;
    SINON
        POUR CHAQUE requête ∈ sous arbre actuel FAIRE
            Remplacer les tables consultées de cette requête par la VM définie;
            Supprimer de la requête courante les prédicats qui apparaissent dans la VM définie ;
        FAIT;
    FSI;
FAIT;

A la fin, il ne reste que de comparer le coût de chaque requête introduite par l'utilisateur avant et après l'optimisation.

A noter que les pseudo-algorithmes ci-dessus sont très simplifiés, ils servent uniquement à comprendre l'idée derrière chaque étape. Dans la pratique, le programme d'optimisation est beaucoup plus complexe.

3 Résultats de l'approche :

Les résultats obtenus avec l'approche proposée dans ce projet montrent l'efficacité de cette dernière. En effet, on remarque la diminution considérable des coûts d'un ensemble de requêtes après leur réécriture en fonction des VM définies, soit une diminution de ~78%, ~51% du coût total des requêtes (en se basant sur le plan d'exécution estimé par le SGBD avant l'utilisation des VM et après) (Figure 8).



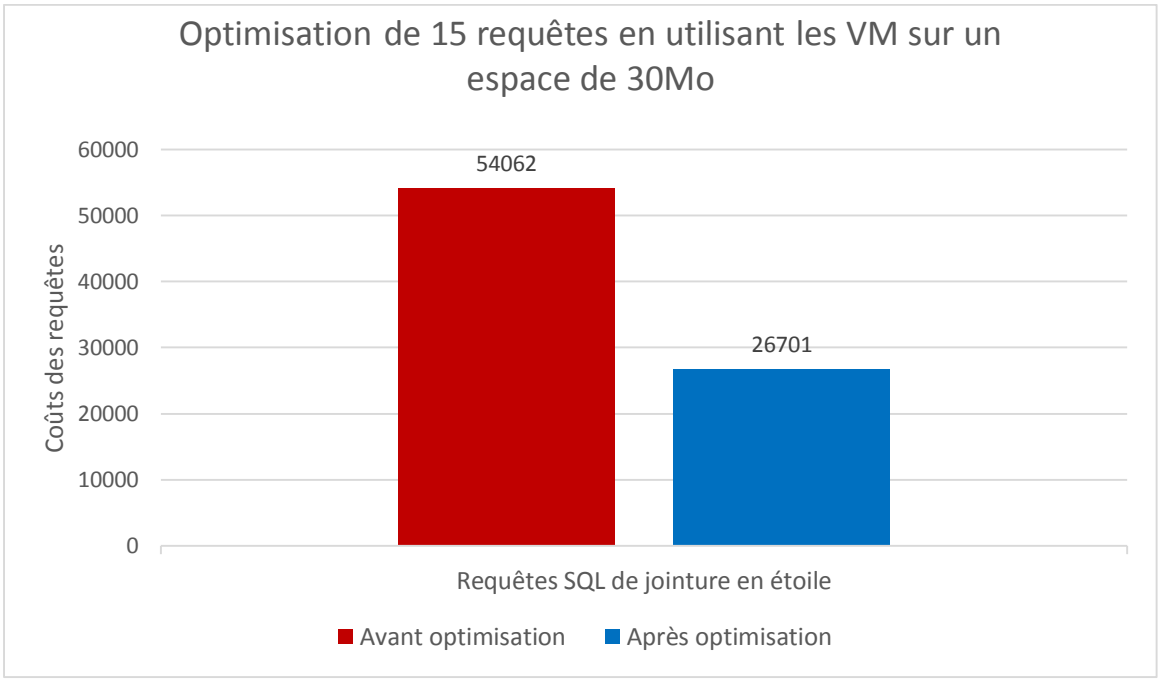


Figure 16 : Comparaison des coûts de 5 requêtes et 15 requêtes avant et après leur optimisation avec les VM

Bien que ces résultats diffèrent d'un ensemble des requêtes utilisées à autre. Dans la plupart des cas de test, une différence remarquable est notée.

4 Récapitulatif :

L’interface web conçue, est un outil à deux fonctionnalités principales :

● Se connecter à son SGBD à distance et visualiser le plan et l’arbre d’exécution de chaque requête introduite par l’utilisateur.

● Optimiser les requêtes SQL de jointure en étoile en utilisant les VM en définissant l’espace disque souhaité.

Chapitre V : Conclusion générale

Sommaire

1 Conclusion :

Nous avons traité, à travers ce travail, la problématique de l'optimisation des requêtes SQL dans les entrepôts de données avec un schéma en étoile, une structure connue d'être très volumineuse et donc coûteuse en temps d'accès, en proposant une approche isolée redondante, en utilisant les VM. On a réussi, par conséquent, à diminuer le coût, un facteur décisif de la performance, des requêtes en se basant sur leur plan d'exécution, un schéma établi pour trouver la meilleure façon –estimée– d'exécuter une requête. Tout cela, grâce à un outil d'aide destiné aux administrateurs et concepteurs, hébergé sur un site web en utilisant les différentes technologies et algorithmes. Les fonctionnalités phares de cet outil se résument en :

a) Affichage du plan et l'arbre d'exécution : Les requêtes entrées passent directement au moteur d'optimisation du SGBD choisi. Ainsi, un affichage tabulaire du plan d'exécution proposé est fourni, plus claire et lisible en le comparant à l'affichage direct depuis la ligne de commandes qui peut être parfois décalé ou mal lisible. De plus, pour plus de clarté, il est possible de visualiser l'arbre d'exécution, qui est une implémentation du plan au format tabulaire et qui prenne en considération la hauteur de chaque nœud du plan d'exécution.

b) Regroupent des jointures communes : L'optimisation nécessite, avant de l'effectuer, l'identification des sous arbres de jointure communes, il s'agit de récupérer, directement depuis le plan d'exécution, les jointures en étoile contenues dans les requêtes avec leur prédicats et tables impliquées qui peuvent servir à d'autres structures d'optimisation (isolée ou multiple).

c) Définition des VM et réécriture des requêtes : Les jointures communes sont prêtes à être mises, chacune d'elles, dans des VM. Cela est le premier pas pour que les requêtes soient optimisées, cependant, cet effet sera visible après la réécriture automatique de ces dernières en fonction des vues matérialisées définies précédemment.

d) Affichage des résultats : Le travail se termine par la récupération des coûts estimés des requêtes

réécrites dans un graphe statistique afin de mieux, encore une fois, visualiser les données-clé relatives à l'optimisation. De plus, nous donnons à l'utilisateur la possibilité de télécharger le script utilisé pour l'optimisation (les VM définies et les requêtes traitées avant et après l'optimisation) pour garantir la transparence.

Remarque : Dans ce mémoire, les termes utilisateur, concepteur, décideur et administrateur ont été utilisés dans le même contexte. Il s'agit d'une personne utilisant un SGBD.

2 Perspectives :

Pour les travaux futurs, il serait intéressant d'étendre notre travail aux cas suivants :

- Lier le choix des requêtes à optimiser à d'autres critères pouvant améliorer le coût de l'optimisation ou l'espace mémoire utilisé (par exemple : la taille ou la profondeur).

- Coupler l'approche réalisée avec une autre structure d'optimisation (sélection multiple) afin d'exploiter au mieux les ressources offertes par le matériel équipé du SGBD.

- Estimer automatiquement l'espace à allouer pour l'optimisation.

- Ajouter des fonctionnalités, autres que l'optimisation, pour transformer l'outil d'aide en un gestionnaire complet des SGBD à distance.

Bibliographie

[1] : Amira KERKAD. L'interaction au service de l'optimisation à grande échelle des entrepôts de données relationnels. ISAE-ENSMA - Poitiers, 2013.

[2] : Filippe Rigaux. Cours de bases de données. 13 Juin 2001.

[3] : The Oracle Optimizer Explain the Explain Plan. Avril 2017.

[4] : Marian Kopczewski. Data warehousing tool for decision – Making in managements. WSB Poznań. 2014.

[5] : Vidette Poe, Patricia Klauser, Stephen Brobst. Création d'un entrepôt de données. WNT Varsovie 2000.

[6] : Ladjel Bellatreche. La conception physique des data warehouses. LISI / ENSMA. 2006.

[7] : Andrzej Ptasznik. Entrepôts de Données – Comment garantir l'accès aux données. Ecole supérieure de l'informatique, Varsovie. 2015.

[8] : Stephen Cook, The complexity of theorem proving procedures, Proceedings of the Third Annual ACM Symposium on Theory of Computing. 1971.

[9] : H. Garcia-Molina, J. D. Ullman, and J.Widom. Database Systems : The Complete Book, Prentice Hall Press, Upper Saddle River, NJ, USA, 2éme édition, 2008.

Conception d'un simulateur en ligne pour l'aide aux administrateurs et concepteurs de bases de données en optimisant les requêtes de jointure en étoile : Exploitation du mémoire secondaire.

Présenté par :

Addi Ramzi et **BENAMARA Soufian Przemyslaw**

Encadré par :

Dr. KERKAD A.

Résumé : Un entrepôt de données est une structure contenant un grand volume de données. L'accès à ces données est assuré par des requêtes qui sont généralement très couteuses en temps de réponse. Dans notre travail, nous allons présenter un site web avec connexion à distance à une base de données qui permet de diminuer le coût de ces requêtes, en utilisant une technique : vues matérialisées.

Mots-clés : Base de données, Entrepôt de données, Optimisation des requêtes, Site web, Vues Matérialisées.

Abstract : A data warehouse is a structure which contains a large volume of data. Accessing this data is insured by queries which are generally very expensive in response time. In our work, we will introduce a website with a remote database connection which allow to reduce the cost of these queries, using a technique : materialized views.

Keywords : Databases, Data warehouses, Query optimization, Website, Materialized Views.
