Projet SGBD

JAVA - JDBC

Karim MOULAY

SOMMAIRE :

Rappel :

Cette partie du projet SGBD consiste à développer une application qui permet une recherche partielle des données sous oracle. Ceci veut dire que pour rechercher des données, on va remplir quelques champs pour voir les résultats.

Dans notre cas, on va laisser un ou plusieurs champs vides, qui seront dédiés aux résultats, donc pour toutes les valeurs possibles.

Pour récupérer les données, il est conseillé de profiter de l'indexation par cluster d'oracle.

Les choix :

Pour développer une telle application, il était important de définir comment procéder. Nous avons été conseillés et orientés vers JAVA et JDBC, mais nous avions toutefois la liberté total de choisir un autre langage de programmation.

Et sur ce, une idée m'a traversée l'esprit et j'ai pensé qu'il serait intéressant, de faire une interface homme/machine en JAVA, et un code en PL/SQL.
L'avantage de cette solution, est que l'utilisateur fait ces manipulations, et les triggers passent derrière pour mettre à jour, et faire les opérations qu'il faut.
Mais comme je l'ai dit, cette idée n'a fait que très vite me traverser l'esprit. J'ai oublié cette solution, car elle nécessite une bonne maitrise du PL/SQL pour ne pas ce noyer dans les bugs. Et vue le temps restreint que j'avais pour ce projet, et le planning chargé uniquement pour mon master, cela n'était pas faisable.

Ainsi, le projet est fait en JAVA, en utilisant les bibliothèques JDBC. Pour ce faire, j'ai mis en place une structure simple, il y a une TABLE **liste** dans la base de données. Chaque table ajoutée depuis mon interface, aura sa structure sauvegardée dans la table LISTE, y compris la valeur de la fonction modulo choisie pour chaque colonne.

Chaque table créée, ce voit ajoutée une colonne **code**. Celle ci n'apparaît nulle part sur l'interface JAVA. Elle contient le code généré par le codage des tuples selon leur modulo respectif choisi. Un index cluster sur CODE est créé pour chaque table créée. Une SEQUENCE est aussi créée pour numéroter les données.

La recherche ce fait en manipulant les bits directement, et consiste à régénérer le (ou les) code(s) correspondant.

Codes et algorithmes :

Je vais développer les points les plus pertinents dans cette partie.

Commençons par étape :

1. Création de table :

Comme je l'avais expliqué, pour chaque table créée, une colonne id est automatiquement ajoutée, et une séquence associée est aussi créer pour incrémenter automatiquement l'id.

Un cluster lui est aussi dédié. Lorsque l'on crée un CLUSTER, il y a deux données importantes à connaître, à savoir :

- le nombre de blocs à réserver, et la valeur de la fonction modulo choisie.

Dans cette application, j'ai choisi de définir la même valeur pour le nombre de bloc et pour le modulo.

Lors de la création du cluster, ORACLE arrondi automatiquement le nombre de bloc au nombre premier supérieur le plus proche. Du coup, lors de la création de la table, après l'étape de création de la structure de celle-ci; il est proposé à l'utilisateur de sélectionner le nombre de valeur approximatif de ligne qu'il va choisir. Ceci permettra de calculer le nombre de blocs nécessaires.

Comment ça se passe ?

Selon le nombre de colonne à partitionner sélectionnées, on va avoir un nombre minimum de bloc à produire.
En effet, pour exploiter ce mode de recherche, Il faut au minimum partitionner 2 colonnes. Sans quoi, l'application refuse de valider la structure. C'est pourquoi, il est impératif que chaque colonne ait un modulo au minimum supérieur ou égale à 2.

Exemple :

Soit 3 colonnes à partitionner, chaque colonne doit avoir logiquement un modulo au minimum de 2 (modulo 1 n'aura aucun sens). Donc si à chaque colonne on affecte un modulo 2, on aura une somme de minimum modulo 6. Il faudra donc réserver au moins 6 blocs. Hors, ORACLE va arrondir au chiffre premier le plus proche, soit donc 7.

C'est pourquoi, j'ai écrit deux petites équations mathématiques. :

- La première, calcul le poids maximal d'une ligne, le multiplie par le nombre de ligne que l'on veut saisir et divise le tout par **8192**octets : **(taille_ligne*nombre_ligne)/8192**.
- La seconde équation, qui est en fait plus une fonction ; consiste selon la valeur obtenue par l'opération précédente à déterminer si le nombre est premier ou pas. Si le nombre n'est pas premier, il est incrémenté jusqu'à le devenir.

```
   Soit nb=nombre de blocs nécessaire ;
   Repeter nb++ jusqu'à Premier= vrai
        Premier=vrai;
        POUR i = 1 ; i < nb/2, i+2
(on avance de 2 car les nombres pairs sont inutiles et ne font jamais de nombre premier.)
                si nb mod i ≠ 0 alors i=nb (pour sortir de la boucle) premier=faux
                fin si
        fin pour
   fin repeter
```

Après cette étape, Nous avons le nombre de bloc et de ce fait, la valeur de la fonction modulo, et ce proportionnellement au nombre de ligne à saisir.
Il faut ensuite affecter à chaque colonne à indexer une valeur de modulo, de telle sorte que leurs sommes soient égales aux nombres de blocs à réserver. Si la valeur est supérieure ou inférieur, il est impossible de finaliser la création.

Quand on valide toutes les étapes, avant de commencer la création de la table et de faire toutes les étapes associées, on fait un **COMMIT** sur la base données. Si les opérations n'aboutissent pas, et que durant la création il y a une erreur, un **ROLLBACK** est exécuté.

2. Insertion de données :

Un COMBOBOX permet de choisir parmi une liste la table dans laquelle on veut insérer les données. On insert les données dans les champs respectifs. On ajoute si nécessaire des lignes. Et on valide. Les lignes seront lues une à une, et envoyées à la base de données une à une après avoir généré le code correspondant. Chaque tuple partitionné est codé selon son type et le modulo qui lui est associé.

Mais comment ce fait le codage ?

Pour les nombres (INT, FLOAT, DATE,…), rien de particulier, c'est un simple modulo. Pour ce qui est des chaines de caractères, c'est un codage qui dépend du code ASCII du premier caractère.

Exemple : soit le nom "**M**OULAY"
le cryptage ce fera par rapport au code ASCII du **M**; qui vaut 77. Donc si on va appliquer un modulo 5 par exemple, on aura : **M mod 5 = 2**

Mais pour le moment, nous n'avons codé que les tuples partitionnés, mais on n'a pas généré le code de la ligne.

Comment ça cela passe ?

Pour ce faire, j'ai écrit quelques fonctions. Parmi ces fonctions, j'ai noté la fonction **log2**.
C'est simple, elle prend en paramètre une valeur (entier généralement).
Celle-ci va retourner la longueur de sa valeur en bits.

Exemple :
3 en entrée retourne 2 pour 2 bits. 11 en entrée, retourne 4…
Une autre fonction que j'utilise, c'est la fusion binaire.

Elle prend en entrée deux valeurs à fusionner, et un modulo pour les séparer.

Exemple :
soient v1=3, v2=1 et mod=2; v1=11 est décaler de 2 bits à gauche. Ce qui donne 12.
On ajoute v2=1 pour obtenir le résultat, qui est 13.
Pour faire un décalage binaire à gauche, il y a 2 solutions, soit on multiplie par 2 pour chaque bit, soit on utilise les opérateurs binaires **<<** (choix que j'ai fait).

L'opération de fusion ce fait au fur et à mesure que l'on récupère les données.

Exemple :

Dans le cas de 3 valeurs (v1, v2 et v3) à saisir, f=0, on fusionne :
f=(f, v1, mod1), puis f=(f, v2, mod2), et enfin f=(f, v3, mod3)
Au final, on aura le code pour cette ligne.
Chaque ligne finie est insérée à la base.

3. Recherche de données :

Le but de cette application est d'utiliser cette fonctionnalité. Dans le cadre de mon développement du projet, je me suis limité à un seul champ vide. Comment ça ce passe ? L'utilisateur choisi une table dans la liste. Automatiquement, il aura un tableau où seul les colonnes partitionnées pour cette table s'afficheront.

Il faut remplir les champs que l'on a choisis de fixer, et en laisser vide un seul.
Après, c'est bon, on lance la machine.

Je vais encore une fois simplifier les choses ici aussi.
Comme toujours, on récupère les données colonne par colonne, on fait tout comme dans le cas de l'insertion, une fusion des données au fur et à mesure de leur récupération.

Mais ici, il y a deux choses particulières que l'on va faire.

- La première, est que lorsque l'on doit fusionner avec la case vide, on fusionne par la valeur 0, et le modulo qu'aura pris cette colonne.
- La seconde, est que l'on a deux nouvelles variables que l'on va appeler mod et somme. La variable mod, prendra la valeur du modulo de la colonne qui sera vide. La seconde variable, contiendra la somme des modulos à partir de la colonne vide.

Pour plus de clarté dans mon explication, je vais développer un exemple mais avant celui-ci, je vais commencer par l'algorithme :

```
somme=0, mod=0, code=0
Répéter récupérer cellule suivant jusqu'à plus de cellules
    Si case vide alors mod=modulo_tuple, somme=0
    Sinon somme= somme+log2(modulo_tuple)
    Fin si
    Code=Fusion(code, modvaleur_cellule, modulo_tuple)
finrepeter
```

Déroulons à présent l'algorithme avec un exemple simple :
Soit : les données/modulo suivent : 34/4, 22/7, NULL/3 et 45/5
On commence :

Somme=0, mod=0, code=0
On récupère 34 ≠ NULL
somme=0+log2(4)= 0+3=3
code=fusion(0, 34%4 , 4)= fusion (0, 2, 4)=2
On récupère 22 ≠ NULL
somme=3+log2(7)=3+3=6
code=fusion(2, 22%7, 7)=fusion(2, 1, 7)= 7
On récupère NULL
mod=3, somme=0
code=fusion(7, 0, 3)= 14
On récupère 45 ≠ NULL
somme=0+5=5
code=fusion(14, 45%5, 5)=fusion(14, 0, 5)=42
plus de cellule
Résultat, à ce niveau, on a l'adresse 42, mod=3, et somme=5

On continue donc la recherche :

J'ai ensuite écrit une nouvelle fonction recherche. Elle prend en paramètres d'entrée valeur, modulo et decale. Et en sortie, on a un tableau d'entier

```
Tab[log2(mod)] (tableau d'entier de taille = mod)
Pour i de 0 à log2(mod) faire
    Tab[i]=val+(i << log2(mod))
Fin pour
Retourne tab
```

Exemple :
Si je déroule l'algorithme avec les valeurs récupérées du précédant exemple, ça va donner ça :

val=42, modulo= 3, decale=5
pour i de 0 à 3 faire
i=0, tab[0]=42+(0 << 2)=42
i=1, tab[1]=42+(1 << 2)=42+ 4=46
i=2, tab[2]=42+(2 << 2)=42+8=50
i=3, tab[3]=42+(3 << 2)=42+12=54
fin pour

Donc pour les valeurs choisies, au début, on va avoir comme résultat, toutes les lignes qui auront comme valeur de code 42, 46, 50 et 54.

Déroulement du développement :

J'espère que ces exemples et explications sont assez clairs pour expliquer les fonctions principales du programme.

Sinon, pour le développement, il y a eu naturellement une phase réflexion. Le mode opératoire que j'ai choisis pour faire le programme n'a pas été très difficile. J'ai commencé par écrire les fonctions dont j'avais besoin, telles que les fonctions « coder mot, coder nombre, fusion, recherche… » . Mais, la galère était pour développer une interphase homme/machine dynamique.
En effet, cette étape m'a pris beaucoup de temps, et en particulier, l'interphase de saisie de la structure de la nouvelle table. Il fallait trouver le moyen de construire une interphase qui puisse évoluer selon le nombre de tuple que l'on veut ajouter. Je suis passé par une multitude de solutions, l'avant dernière fut de faire une interface de saisie, qui va saisir les tuples un par un. Ce qui impliquait de trouver une solution pour stocker les variables temporairement, avant de les exécuter, et ceci sans connaître leur nombre final.
Je suis passé par plusieurs alternatives. Mais après plusieurs jours de traquas, j'ai vu le projet d'un autre groupe de la promo durant le TP. Ils m'ont dit qu'ils utilisaient les tables. Je me suis donc penché sur cette possibilité, et là, enfin, j'ai trouvé une solution qui pourrait convenir.
En effet, une variable qui ne nécessite pas que je face une liste chainée, ni une classe particulière, ni rien pour conserver les données « **UN TABLEAU DYNAMIQUE** ».

Après plusieurs recherches, je suis arrivé à maitriser au moins les fonctionnalités dont j'avais besoin. Le type de variable JTABLE m'a beaucoup servie aussi pour la suite, à savoir, après la création, de l'utiliser pour l'insertion, et la sélection.

Le second point, qui fût une patinoire, c'est le codage du modulo. Bien que tout parait clair dans ma tête, après quelques nuits blanches, on s'emmêle les pinceaux, et plus vraiment moyen de comprendre ce qu'il ce passe…

Autre point de mystère, mais j'en suis convaincu. Pour la création de table, j'effectue un COMMIT, et en cas de défaut, un ROLLBACK. J'ai fait ce choix, plutôt que de mettre des verrous ; par sécurité, il ne vaut mieux pas trop tester des choses trop nouvelles, et que je n'ai pas bien maitrisé. Pourtant, j'ai eu beaucoup de problèmes avec certaines requêtes, à certains moments de la journée. D'après mon analyse personnelle, il doit y avoir au moins deux bases de données qui ce synchronisent. L'accès en SSH à BUTOR peut donner accès à une autre base de données que celle de UFRSCIENCESTECH. Pourquoi ? La première fois, une nuit blanche pour accéder à une table que JAVA me dit ne pas la trouver. Alors qu'en SSH, elle est bien là.
J'y ai passé une nuit blanche car je pensais que l'erreur venait de moi. Comme qui dirait, la nuit porte conseil, et pour preuve, le lendemain, miracle, sans rien changer, ça marche !!!
Par la suite, je me suis rendu compte que les données supprimées depuis BUTOR, existaient encore sur UFRSCIENCESTECH. J'ai effectué des DROP TABLE et des DELETE, mais les données ressortaient toujours, ou encore pire, réapparaissaient, après avoir disparues (après une nouvelle connexion). Il est du coup possible qu'en cas de surcharge du serveur principal, que la requête soit dirigée vers un serveur secondaire… Plusieurs hypothèses sont possibles en tout cas.

Conclusion :

Les deux parties du projet ont été très intéressantes. Mais je regrette que pour la seconde partie, l'interphase graphique m'a pris autant de temps, car dans le cas contraire, j'aurai pu passer plus de temps sur les options, et les compléments du projet. Parmi les choses que j'aurai pu ajouter, la saisie avec possibilité de laisser plusieurs champs vide, et générer l'adresse, l'optimisation du code, l'amélioration des fonctions modulo mot (comme par exemple faire un modulo de la somme des caractères par exemple), mais aussi la fonction HASHCODE qui comporte quelques défauts de la façon dont celle-ci est faite.
Car si par exemple, je choisie pour une table la fonction modulo 37, et que je partitionne 2 colonnes, la première en modulo 32, et la seconde en modulo 5. Si je cherche des valeurs sur modulo 5, je vais avoir des adresses telles que 38 et 39, qui ne donneront aucun résultat, mais qui vont obliger ORACLE à lire des blocs qui ne le concernent pas…