TP2 : Indexation, Optimisation et Transaction

Ce TP propose de découvrir quelques mécanismes fournis par Oracle pour optimiser les requêtes et gérer des transactions. Il comprend de nombreuses questions où vous devez expliquer et/ou analyser les résultats obtenus. Il est important de ne pas passer à la question suivante sans être sûr d'avoir bien répondu à la question. N'hésitez donc pas à vous en assurer auprès de votre enseignant.

A - Organisation physique des données

Comme expliqué en cours et TD, les données d'une base de données sont réparties dans des *blocs* (aussi appelés *paquets* ou *pages*) d'un ou plusieurs fichiers.

A-1. Quelle est la taille en Ko d'un bloc (paramètre `DB_BlOCK_SIZE`) ? Utilisez l'aide ci-dessous.

A-2. Sous Oracle, les données d'une table sont stockées dans un *segment* qui correspond généralement à plusieurs blocs. Sachant que la table `user_segments` contient les segments créés par l'utilisateur courant, écrivez une requête permettant de trouver la taille en Mo de la table `Joueur` et le nombre de blocs correspondants. Déduisez-en la taille moyenne d'une ligne de cette table.

A-3. Lors d'une lecture séquentielle, plusieurs blocs peuvent être lus avec une seule entrée/sortie (E/S). Le nombre maximum de blocs lus en une E/S pendant une lecture séquentielle est défini par le paramètre `DB_FILE_MULTIBLOCK_READ_COUNT`. Quelle est la valeur de ce paramètre ? Lorsque l'optimiseur va devoir parcourir cette table pour faire un filtre, est-ce qu'il aura toujours intérêt à utiliser un index pertinent pour faire ce filtre ? Attention, les blocs contenant les données d'une table ne sont pas forcément placés les uns après les autres sur le disque (car Oracle alloue un bloc pour une table dès qu'il y en a besoin).

A-4. La table `user_indexes` contient des informations sur les index. Écrivez une requête permettant de trouver les index qui ont été créés en même temps que les tables de votre base de données ? Pour chaque index, vous afficherez son nom, son type (b-tree, bitmap) et la hauteur de l'index si c'est un b-tree. Vous afficherez aussi la colonne `clustering_factor` et, en vous appuyant sur la documentation en ligne de la table `user_indexes`, indiquerez ce que cela signifie pour la table `Joueur` et l'intérêt de l'index associé à cet table. Remarque : si les valeurs ne sont pas indiquées (null), faites la question B-1 avant celle-ci.

A-5. Sachant qu'un index est stocké dans un segment, donnez la taille en Mo de l'index créé sur la table `Joueur` ainsi que le nombre de blocs correspondants. En utilisant la colonne `leaf_blocks` de la table `user_indexes`, expliquez comment ces blocs sont répartis dans les différents niveaux de l'index. Enfin, comparez la taille de l'index à celle de la table `Joueur` et expliquez pourquoi vous obtenez ce résultat.

Aide Affichage de la valeur d'un paramètre :

```
SHOW PARAMETER parametre;
```

Obtenir les colonnes d'une tables :

```
DESC nom_table;
```

B - Gestion des statistiques sur la base de données

Pour exécuter une requête SQL, un optimiseur génère les différents plans d'exécution possibles et choisit celui dont il estime que le coût est minimum. Pour faire cette estimation, il se base sur des statistiques qu'il maintient sur les tables et index de la base de données. Pour ne pas ralentir des instructions SQL, les statistiques ne sont pas toujours mises à jour lors d'insertions, mises à jour ou suppressions de données. Il est donc important de mettre à jour manuellement ces statistiques lorsque de grosses modifications sont faites sur les tables (sinon, l'optimiseur risque de choisir un mauvais plan d'exécution car il se sera basé sur des estimations des coûts erronés).

B-1. Mettez à jour les statistiques sur tous les objets (tables, index) contenus dans votre schéma (c'est-à-dire créés avec votre compte) avec la commande suivante :

```
EXEC dbms_stats.gather_schema_stats('nom_user');
```

B-2. En utilisant la table `user_tab_statistics`, vérifiez que les statistiques (nombre de lignes, nombre de blocs et taille moyenne d'une ligne) sur la table `Joueur` sont correctes.

B-3. Faites de même pour l'index sur la table `Joueur` en utilisant la table `user_ind_statistics`. Vous regarderez les statistiques sur : le nombre de niveau, le nombre de blocs feuilles, le nombre de clés distinctes et le `clustering_factor`.

B-4. Ajoutez une ligne à la table joueur et regardez à nouveau les statistiques sur la table `Joueur` et son index. Faites en sorte que les statistiques sur cette table soient à jour en utilisant la commande suivante :

```
EXEC dbms_stats.gather_table_stats('nom_user', 'nom_table');
```

B-5. En consultant la table `user_tab_col_statistics`, donnez des statistiques sur la taille des joueurs : nombre de tailles distinctes, tailles mininale et maximale, nombre de tailles non renseignées (valeur null) et vérifiez les par des requêtes. Pour décoder les valeurs minimales et maximales, vous utiliserez l'expression : `to_char(utl_raw.cast_to_number(colonne))`. Pensez bien à définir dans le `WHERE` la table utilisée et la colonne utilisée (sinon la requête sera extrêmement longue).

B-6. L'optimiseur maintient, en plus des statistiques précédentes, un histogramme décrivant la fréquence d'apparition cumulative de chaque valeur distincte de taille. Faites une requêtes sur la table `user_tab_histograms` pour afficher cet histogramme dans l'ordre croissant des fréquences d'apparition cumulative. Avant de faire cela vous exécuterez une requête recherchant les joueurs de plus de deux mètres et mettrez à jour les statistiques (ceci indique à Oracle de faire un histogramme sur la colonne taille puisqu'elle a été utilisée comme filtre dans une requête).

C - Analyse du plan d'exécution d'une requête

Les commandes suivantes permettent d'afficher le plan et le temps d'exécution d'une requête :

```
SET AUTOTRACE ON EXPLAIN;
SET TIMING ON;
```

Le plan d'exécution permet notamment de connaître (1) la méthode d'accès utilisée pour chaque table (accès sequentiel : `TABLE ACCESS FULL` ou accès via un index : `TABLE ACCESS BY INDEX`) de la requête (2) les algorithmes utilisés pour faire les jointures (boucle imbriquée : `NESTED LOOP`, tri-fusion `SORT-MERGE`, hachage : `HASH JOIN`) et (3) l'ordre des jointures. En plus de ces informations, le plan d'exécution contient des informations sur chaque opération effectuée : nombre de lignes retourné, taille en octet de ces lignes, coût estimé de l'opération (une valeur qui prend en compte les E/S, le côut CPU et l'utilisation de la mémoire centrale).

Questions

Pour chacune des requêtes suivantes, relevez leur plan d'exécution et indiquez la méthode d'accès utilisée pour chaque table et, s'il y a une jointure, l'algorithme utilisé et l'ordre des tables suivi pour faire cette jointure. Vous indiquerez également si les méthodes d'accès vous semblent logique (est-ce normal de faire un accès séquentiel ou un accès par index ?) en vous appuyant sur l'estimation des coûts indiqués (l'optimiseur choisi le plan dont le coût estimé est le minimum). Pour vérifier votre explication, vous forcerez l'optimiseur à utiliser une autre méthode d'accès (quand c'est possible) avec les *hints* suivants (ce sont des commentaires spéciaux interprétés par l'optimiseur) :

```
-- forcer l'utilisation d'un index
SELECT /*+ INDEX(nom_table nom_index) */ col1, col2 FROM ...
-- empêcher d'utiliser un index
SELECT /*+ NO_INDEX(nom_table nom_index) */ col1, col2 FROM ...
```

C-1. description des joueurs de plus de 2 mètres.

C-2. description du joueur d'identifiant `46509` ;

C-3. dates des matchs avec, pour chaque match, le nom de l'équipe locale qui l'a joué.

D - Optimisation de requêtes en créant des index

Une technique d'optimisation de requêtes consiste à définir des index sur les colonnes des tables. Voici quelques commandes pour créer des index btree ou bitmap :

```
-- index btree
CREATE INDEX nom_index ON nom_table(nom_colonne)
-- index btree sur l'application d'une fonction à une colonne
CREATE INDEX nom_index ON nom_table(fonction(nom_colonne))
-- index bitmap
CREATE BITMAP INDEX nom_index ON nom_table(nom_colonne)
```

Questions

On rappelle qu'un index n'est utilisé pour faire un filtre sur une table que lorsque l'optimiseur estime que c'est moins coûteux que de faire un parcours séquentiel de la table. C'est souvent le cas si les conditions suivantes sont réunies : 1) le coût de faire un parcours séquentiel est important (donc la table a une taille importante), 2) le filtre est très sélectif et 3) seule une petite partie de l'index doit être parcouru.

Pour chacune des requêtes suivantes, indiquez si un (ou plusieurs) index peut permettre théoriquement de l'optimiser (c'est-à-dire qu'avec ce ou ces index, les 3 conditions évoquées précédemment seraient satisfaites). Testez ensuite votre proposition en créant le ou les index proposés et en regardant, à l'aide du plan d'exécution, s'ils sont utilisés. Vous essaierez à chaque fois d'expliquer les résultats obtenus (voir aide ci-dessous).

D-1. liste des joueurs dont le nom se trouve après la chaîne `'von'` dans l'ordre alphabétique ;

D-2. liste des joueurs dont le nom se trouve avant la chaîne `'von'` dans l'ordre alphabétique ;

D-3. liste des joueurs dont le nom se trouve après la chaîne `'van'` dans l'ordre alphabétique ;

D-4. liste des joueurs dont le prénom est inconnu ;

D-5. liste des joueurs dont le nom commence par `Ib` ;

D-6. liste des joueurs dont le nom termine par `oz` ;

D-7. nombre de joueurs dont le nom termine par `oz` ;

D-8. liste des joueurs dont le nom en majuscule est `MACDONALD` (utilisez la fonction `upper`) ;

D-9. liste des joueurs qui font une des tailles suivantes : `1,57m`, `1,6m`, `1,63m` ;

D-10. liste des joueurs qui font `2,03m` et ont un nom qui commence par `'M'` ;

D-11. liste des joueurs qui font `1,96m` et ont un nom qui commence par `'Z'` ;

D-12. la requête présente dans le fichier `requete.sql` sur `https://www.lias-lab.fr/ftppublic/enseignement/Oracle/L3/`. Comment réécrire cette requête pour avoir une exécution plus efficace en terme de coût ?

Remarque : ici nous ne regarderons que le plan d'exécution et non le temps d'exécution car la base de données est trop petite (quelques Mo) pour avoir des temps significatifs.

Aide

Si vous n'obtenez pas le résultat attendu, vous pourrez forcer l'optimiseur à utiliser l'index (voir section C) pour essayer de comprendre pourquoi il n'est pas utilisé (en regardant les coûts estimés). Si vous constatez que cela vient du fait que la table `Joueur` est trop petite), vous pourrez faire croire à l'optimiseur qu'il y a plus de lignes et blocs dans la table avec la commande ci-dessous. Il faudra aussi changer les statistiques de l'index créé :

```
EXEC dbms_stats.set_table_stats(USER, 'JOUEUR',
                                NUMROWS=>1000000, NUMBLKS=>100000);
EXEC dbms_stats.set_index_stats(USER, 'nom_index',
             NUMROWS=>1000000, NUMLBLKS=>40000,NUMDIST=>1000000);
```

Un `INDEX FAST FULL SCAN` est un parcours complet d'un index.

Optimisation par le choix d'un algorithme

On peut forcer Oracle a utiliser un algorithme de jointure particulier en utilisant la syntaxe suivante :

```
SELECT /*+ USE_NL(nom_table1 nom_table2 ...) */
       nom_colonne1, nom_colonne2 FROM ...
-- USE_NL : algorithme nested loop (force brute)
-- USE_MERGE : algorithme merge join (tri fusion)
-- USE_HASH : algorithme hash join (hachage)
```

Questions

D-13. Notez l'algorithme de jointure utilisé par Oracle pour exécuter la requête suivante :

```
SELECT *
  FROM Equipe, MATCH
 WHERE Equipe.equipeid = MATCH.equipelocale;
```

D-14. Quelles performances obtenez-vous si vous forcez Oracle à utiliser un autre algorithme ?

D-15. Décrivez l'algorithme qui est utilisé lorsque vous forcez Oracle à utiliser une force brute (dans quel ordre les tables sont-elles parcourues ?). Peut-on améliorer cet algorithme en créant un index ? Testez votre proposition et expliquez les résultats.

E - Gestion des transactions

SQL Developer et les transactions

Dès qu'une connexion à une base de données est effectuée par SQL Developer, une transaction débute. Elle se termine soit en se déconnectant, soit en exécutant une des commandes `COMMIT` ou `ROLLBACK`. L'option `AUTOCOMMIT` placée à `ON` permet de modifier ce comportement. Dans ce cas, chaque opération de mise à jour de la base est traitée dans une transaction individuelle.

Questions

E-1. Quel est le comportement de SQL Developer sur la transaction en cours lorsqu'une déconnexion est effectuée (commit ou rollaback) ? Vous testerez le cas où l'utilisateur clique sur la croix pour fermer le logiciel et celui où il clique droit sur la connexion et sélectionne `déconnexion` ;

E-2. Que se passe-t-il lorsque vous exécutez les commandes suivantes sous SQL Developer lorsque l'option `AUTOCOMMIT` est placée sur `OFF` puis sur `ON` ?

```
INSERT INTO Ligue (ligueId, nom_ligue, pays)
       VALUES(2, 'National␣3', 'France');
ROLLBACK;
INSERT INTO Equipe (equipeId, nom_equipe, nom_court, ligueId)
       VALUES (1, 'Poitiers␣Football␣Club','PFC', 2);
COMMIT;
```

Oracle supporte également la segmentation de transaction. Ainsi, la commande :

```
SAVEPOINT nom_savepoint
```

définit un instant dans une transaction. Il est alors possible d'annuler toutes les mises à jour qui ont été effectuées à partir de cet instant par la commande :

```
ROLLBACK TO nom_savepoint
```

E-3. En utilisant le mécanisme de segmentation, faites en sorte que le code suivant insère le match sans que son score ne soit indiqué.

```
INSERT INTO MATCH (matchId, equipeLocale, equipeVisiteur)
       VALUES (26000, 7794, 7896);
UPDATE MATCH SET dateMatch = to_date('01/01/18', 'DD/MM/YY')
       WHERE equipeLocale = 7794 AND equipeVisiteur = 7896;
UPDATE MATCH SET scoreLocale = 1, scoreVisiteur = 2
       WHERE equipeLocale = 7794 AND equipeVisiteur = 7896;
```

E-4. Un rollback effectué sur une transaction annule-t-il la création d'un savepoint dans celle-ci ? Justifiez votre réponse par un scénario.

Oracle implémente les mécanismes de verrous de lecture et d'écriture. Un verrou d'écriture est implicitement défini sur un tuple modifié par une des commandes `INSERT`, `UPDATE` ou `DELETE` dans une transaction.

Oracle permet également de placer des verrous explicitement sur des tables par les commandes :

```
LOCK TABLE nom_table IN EXCLUSIVE MODE -- Ecriture
LOCK TABLE nom_table IN SHARE MODE -- lecture
```

Pour illustrer ces mécanismes, lancez deux sessions SQL Developer avec deux utilisateurs différents. Vous placerez l'option `AUTOCOMMIT` à `OFF`.

E-5. Que se passe-t-il lorsque deux utilisateurs souhaitent modifier le même match ? Expliquez pourquoi.

E-6. Vérifiez qu'un verrou de lecture ne peut pas être accordé sur une ligne bloquée en écriture.

E-7. Indiquez puis vérifiez les types de verrous que peut obtenir une transaction sur une ligne lorsqu'une autre transaction détient un verrou de lecture sur celui-ci.

E-8. Peux-t-il se produire des situations d'interblocage ? si oui, comment Oracle résout-il ces cas ?

E-9. Que se passe-t-il si vous placez un verrou exclusif sur une vue ? Tester avec une vue simple et une vue portant sur une requête avec jointures.