TP noté index (3h)

1. Préambule

1.1 Fichier textuel à rendre

Un squelette de fichier textuel (PRENOM_NOM_TP1.txt) contenant les questions pour lesquelles il vous faut apporter des réponses est donné. Il vous faudra le déposer sur Moodle dans l'espace de dépôt TpNote1.

1.2 Consultation des vues du méta-schéma relatives aux index

Les vues INDEX_STATS et USER_INDEXES aident à la compréhension des structures d'index manipulées par un serveur de base de données. L'ordre SQL donné ci-dessous exploite USER_INDEXES et permet par exemple de consulter l'ensemble des index définis sur le schéma utilisateur, le nom de l'index, le nom de la table impactée par l'index ainsi que la hauteur de l'arbre (sans le niveau des feuilles) [1].

```
SELECT index_name, blevel, table_name FROM USER_INDEXES;
```

La vue INDEX_STATS donne des informations complémentaires (parfois chevauchantes) à la vue USER_INDEXES. Il est ainsi possible de disposer d'informations sur la place mémoire occupée par l'index, le nombre de blocs occupés par les nœuds branches (BR_BLKS) et les nœuds feuilles de l'arbre (LF_BLKS). Il est cependant nécessaire de collecter les statistiques sur les index avant de consulter cette vue. La consultation ci-dessous retourne respectivement le nom de l'index, l'espace occupé en octets, le nombre de répétitions pour la valeur de clé la plus répétée, le nombre de tuples (clé, rowid, pointeur tuple gauche, pointeur tuple droit) au niveau feuille, le nombre de tuples (clé, pointeur) au niveau nœud des branches et la hauteur de l'arbre (avec le niveau feuille et donc égal à blevel+1).

```
-- mettre a jour les statistiques pour un index donné
ANALYZE INDEX <index_name> VALIDATE STRUCTURE;

-- ne renvoie des valeurs que pour l'index en cours d'analyse
-- ici renvoi du nom de l'index, espace total occupe en octets par index,
-- repetition de cle (1 si unique),
-- nombre de tuples feuilles, nombre de tuples branches et hauteur
SELECT name, btree_space, most_repeated_key, lf_rows, br_rows, height FROM INDEX_STATS;
```

La signification des principaux attributs de INDEX_STATS est donnée :

1. La valeur 0 indique que l'index n'est constitué que d'un niveau racine

— BLOCKS blocs alloués au segment d'index
— NAME nom de l'index
— LF_ROWS nombre de tuples feuilles (littéralement leaF_ROWS)
— LF_BLKS nombre de blocs feuilles
— LF_ROWS_LEN somme de la taille totale de tous les tuples feuilles en octets
— LF_BLK_LEN espace libre dans les blocs feuilles
— BR_ROWS nombre de tuples branches (littéralement BRanch_ROWS)
— BR_BLKS nombre de blocs branches
— BR_ROWS_LEN somme de la taille totale de tous les tuples branches en octets
— BR_BLK_LEN espace libre dans les blocs branches
— DISTINCT_KEYS nombre de valeurs d'entrée (clés) distinctes dans l'arbre
— MOST_REPEATED_KEY nombre de répétitions de valeurs d'entrée (clés) dans l'arbre

1.3 Rafraîchir les statistiques des données collectées

Il vous faut penser aussi à collecter les dernières statistiques sur les tables avant de faire appel aux vues USER_TABLES ou DBA_TABLES du dictionnaire.

```
-- pour une table en particulier ici
ANALYZE TABLE ABC COMPUTE STATISTICS;
-- pour le schema utilisateur dans son ensemble (appel de la procedure analyze_schema
--  du paquetage dbms_utility
EXEC dbms_utility.analyze_schema(USER,'COMPUTE')
```

Vous pourrez aussi faire appel aux vues USER_SEGMENTS (ou DBA_SEGMENTS), et USER_EXTENTS (ou DBA_EXTENTS) du dictionnaire pour avoir la taille en octets et en blocs des segments associés à la table ABC et à l'index posé sur l'attribut A.

2. Exercice 1 : Tests des calculs de l'exercice 4 de la feuille de TD

L'exercice 4 de la feuille de TD a été corrigé la semaine dernière en salle de cours (le corrigé vous est rappelé en annexe). Vous avez à en tester les résultats maintenant sur machine. L'idée est en effet d'évaluer les ordres de grandeur obtenus de part et d'autre.

Les données considérées correspondent à la table suivante :
ABC(A number, B varchar(20), C varchar(20))

2.1 En pratique

Un programme principal alimente une table nommée ABC avec 1 000 000 de tuples de 50 octets. Le paquetage DBMS_RANDOM est mobilisé pour générer des valeurs textuelles aléatoirement. La fonction string de ce paquetage va permettre de générer des chaînes de caractères d'une taille spécifiée en minuscules (L pour Lowercase) ou en majuscules (U pour Uppercase).
Vous avez à créer la table ABC, puis à l'alimenter avec le programme principal PL/SQL qui vous est donné.

```
drop table abc;
```

```
create table ABC (A number, B varchar(20), C varchar(20)) ;

 declare i number;
  begin
  for i in 1..1000000
  loop
  insert into abc values (i,dbms_random.string('L', 20),dbms_random.string('U', 20)) ;
  end loop ;
  commit ;
 end ;
/
```

Un index nommé ABC_PK (de type arbre B+) est ensuite créé lors de la définition de la contrainte de clé primaire sur l'attribut A (voir l'ordre de définition ci-dessous). Vous comparerez et commenterez les résultats obtenus par les requêtes portant sur user_tables et index_stats avec les calculs de l'exercice de TD (correction en annexe).

```
alter table abc add constraint abc_pk primary key (a);

analyze table abc compute statistics ;

select avg_row_len, num_rows, blocks, avg_space, empty_blocks
from user_tables where table_name = 'ABC';

analyze index abc_pk validate structure ;

set linesize 180
select height, blocks, lf_rows, lf_rows_len, lf_blks, br_rows, br_blks, br_rows_len
from index_stats;

select segment_name, blocks, bytes/1024/1024 from dba_segments where owner = 'E20....';
```

2.2 Partie à rendre

Toute la partie de questions en rouge est à rendre.

Des questions très précises auquelles il vous faut répondre, à travers l'écriture de requêtes SQL, sont listées. Il vous faut à la fois donner l'ordre SQL exploité, et le résultat de la requête :

— Pour la table (1 point par question)

1. vérifier la taille d'un tuple de table et vérifier la cardinalité de la table (nombre de tuples).
2. donner le nombre total de blocs alloués à la table ABC en indiquant le nombre de blocs ayant fait l'objet d'écriture et le nombre de blocs vides
3. donner le nombre d'extents (et leur taille en blocs) contenus dans le segment de table associé à la table ABC
4. donner la taille en octets de l'espace de stockage qui a été réservé pour la table ABC

5. commenter les résultats obtenus par rapport à ce que vous en saviez après calculs. Est ce cohérent ?
6. donnez un exemple d'utilisation d'une vue qui pourrait être consultée pour connaître le nombre de blocs parcourus lors de l'exécution d'une requête (utilisant ou non l'index) ?

— Pour l'index (1 point par question)

1. comment savoir si l'index ABC_PK est unique et dense ?
2. donner la taille en octets d'un tuple de branche d'index et la taille d'un tuple de feuille d'index (en expliquant la différence de taille)
3. donner le nombre de blocs branche d'index et le nombre de blocs feuille d'index
4. donner le nombre total de blocs alloués à l'index ABC_PK, ainsi que la hauteur de l'index
5. donner la taille en octets de l'espace de stockage qui a été réservé à l'index ABC_PK
6. commenter les résultats obtenus par rapport à ce que vous en saviez après calculs. Est ce cohérent ?
7. donnez un exemple d'utilisation d'une vue qui pourrait être consultée pour savoir si tous les blocs de l'index ABC_PK sont présents dans le cache de données

3. Exercice 2 : De manière plus générale

3.1 Question 1 (0.5 points par question)

Pensez vous que l'index est utilisé pour les ordres de consultation suivants ? (justifiez) :

1. `select * from ABC where A = 10001 ;`

2. `select A from ABC ;`

3. `select A, B from ABC ;`

4. `select C from ABC where A >=20 AND A <=40`

5. `select * from ABC where C like 'ABC%';`

6. `select A from ABC where A != 10001 ;`

4. Question 2 (notée sur 4 points)

Construisez une procédure PL/SQL nommée InfosTable qui exploite au moins deux vues du méta-schéma parmi lesquelles USER_TABLES, USER_INDEXES, USER_SEGMENTS et USER_EXTENTS, qui soit à même de renvoyer les informations les plus importantes concernant l'organisation logique et physique d'une table et de ses index. La définition de la procédure au sein d'un paquetage et la gestion des exceptions pourront faire l'objet d'un point bonus.

5. Annexe

5.1 Rappel des questions de l'exercice de TD

5.1.1 Question pour la table

- facteur de blocage pour la table
- nombre de blocs requis pour stocker les 1 000 000 de tuples de la table
- Taille de l'espace de stockage en Mo
- Nombre de blocs à parcourir pour une recherche sélective si tuple existe ou si tuple n'existe pas

5.1.2 Question pour l'index BTree unique et dense

- facteur de blocage pour l'index
- nombre de blocs requis pour stocker les 1 000 000 de tuples de l'index au niveau feuilles (autant de tuples d'index que de tuples de table)
- hauteur de l'index
- Nombre de blocs total et taille de l'espace de stockage en Mo
- Nombre de blocs à parcourir pour une recherche sélective si tuple existe ou si tuple n'existe pas

5.2 Questions de l'exercice de TD corrigées

Vous ferez les calculs suivants pour une table nommée ABC de 1 000 000 de tuples de 50 octets chacun, une capacité du bloc de 8192 octets privés de 10% et un index dense de type arbre B+ avec des tuples de 15 octets (s'appliquant à l'attribut A).
Les résultats obtenus sont rappelés :

5.2.1 Correction pour la table

- Espace bloc utilisable (8192-819.2=7372.8) arrondi à 7372 octets utilisables
- facteur de blocage table = 7372 / 50 octets par tuple = 147,44 soit 147 tuples pouvant être pris en charge au sein d'un bloc (les tuples sont stockés dans leur intégralité au sein d'un bloc)
- Taille en blocs de la table 1000000/147 = 6802,72 soit 6803 blocs nécessaires (arrondi au nombre de blocs supérieur)
- Espace de stockage requis pour la table ABC : 6803 * 8192 = 55730176 octets soit 55730176/1024/1024 Mo (environ 53.15 Mo)
- Nombre de blocs à parcourir si le tuple existe : 6803/2 en moyenne ; et si le tuple n'existe pas : la totalité des blocs soit 6803

5.2.2 Correction pour l'index

- facteur de blocage de l'index = 491 (7372/15 = 491,467) : 491 clés d'index (tuples) peuvent êtres stockées au sein d'un bloc/nœud.
- nombre de blocs nécessaires pour stocker l'index : au niveau feuilles 1000000/491 (=2036,66) soit 2037 blocs. Il faut un niveau intermédiaire pour pointer sur autant de nœuds feuilles (le nœud racine ne pouvant pointer que sur au plus 492 nœuds (ou blocs) fils (491 + 1 pointeurs). Un second niveau ne suffira pas, puisque le nombre total d'entrées d'index y sera de 491 * 492 entrées soit 241572 tuples d'index, alors que nous avons 1000000, il faut donc prévoir un niveau supplémentaire. Au niveau intermédiaire : 2037/492 = 4.1402 soit 5 blocs nécessaires

pour pointer sur les nœuds feuilles. Il faut enfin le nœud racine pour pointer sur les nœuds intermédiaires.

— hauteur d'index 3 (ou 2 si la racine est omise). En théorie la hauteur est déterminée approximativement par l'ordre (nombre de clés d'index par bloc divisé par deux) qui est ici de 246 (491/2) et le nombre de tuples. La hauteur est comprise entre ln(1000000)/ln(491) et ln(1000000)/ln(246), ce qui donne des valeurs au dessus de 2

— L'espace de stockage supplémentaire pour l'index : 2037+5+1 = 2043 blocs * 8192 octets (par bloc) = 16736256 octets soit 15,9609375 Mo (16703488/1024/1024)

— Nombre de blocs à parcourir si le tuple existe : 3 blocs d'index pour traverser l'arbre de la racine aux feuilles + 1 bloc (de table avec une opération d'entrée/sortie si le bloc est en mémoire secondaire) ; si le tuple n'existe pas : 3 blocs d'index (index dense donc on sait si la valeur de la clé sur laquelle se fait la consultation est présente ou non).