TP noté index (3h)

1. Préambule : à lire avec attention

1.1 Fichier textuel à rendre

Un squelette de fichier textuel (PRENOM_NOM_TP_24.txt) contenant les questions pour lesquelles il vous faut apporter des réponses est donné. Il vous faudra le déposer , après l'avoir renommé, sur MOODLE dans l'espace de dépôt **TpNote1_24**.

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].

Listing 1 – ordre sur la vue USER_INDEXES

```
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.

Listing 2 – ordre de collecte des statistiques sur l'index

```
ANALYZE INDEX <index_name> VALIDATE STRUCTURE;
```

La consultation dans le listing 6 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).

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

TABLE 1 – Attributs d'intérêt

Attribut	Description
NAME	nom de l'index
HEIGHT	hauteur de l'index
BLOCKS	blocs alloués au segment d'index
LF_ROWS	nombre de tuples feuilles
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
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

Listing 3 – Exemple d'ordre sur INDEX_STATS

```
-- ne renvoie des valeurs que pour l'index en cours d'analyse

SELECT name, btree_space, most_repeated_key, lf_rows, br_rows, height
FROM INDEX_STATS;
```

Les principaux attributs de **INDEX_STATS** sont donnés dans le tableau 1. Vous pouvez toutefois explorer l'ensemble des attributs de cette vue au moyen de **DESC INDEX_STATS**

1.3 Rafraîchir les statistiques collectées sur les tables

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.

Listing 4 – Statistiques sur les tables

```
-- pour une table en particulier
ANALYZE TABLE <table_name> 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** et **USER_EXTENTS** (ou encore **DBA_SEGMENTS** et **DBA_EXTENTS**) du dictionnaire pour avoir la taille en octets et en blocs des segments associés à la table et à l'index posé (associé à la table) que vous manipulerez dans ce TP.

1.4 Correction de l'exercice 4 de la feuille de TD

Vous aviez à corriger par vous même l'exercice 4 de la feuille de TD (reste de la feuille corrigé la semaine dernière en salle de cours). Le corrigé de cet exercice vous est toutefois donné en annexe (en fin de ce document).

2. Construire une table et un index et explorer le coût de stockage et d'accès aux données

Vous avez à tester en pratique les résultats pouvant être obtenus sur machine. L'idée est en effet d'évaluer et de comparer les ordres de grandeur obtenus de part et d'autre (d'un point de vue théorique dans l'exercice 4 et d'un point de vue pratique).

2.1 Préliminaires

Les données considérées correspondent à la table suivante :
ABC(A number, B varchar(20), C varchar(20))
Un programme principal (fourni) alimente la table **ABC** avec 1 000 000 de tuples de 50 octets.

La table **ABC** possède également une contrainte de clé primaire **ABC_PK** qui s'applique à l'attribut **A** et qui va donner implicitement lieu à un index de même nom **ABC_PK**. 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).

Listing 5 – En pratique

```
DROP TABLE ABC;
CREATE TABLE ABC (A number, B varchar(20), C varchar(20)) ;

--programme principal PL/SQL
 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 ordres 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 4 du TD (correction annexe section 3).

Listing 6 – En pratique suite

```
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 Exercice 1

Des questions très précises auquelles il vous faut répondre, à travers l'écriture de requêtes SQL, sont listées. Les vues citées précédemment (USER_INDEXES, USER_TABLES, USER_SEGMENTS, INDEX_STATS, ...) sont à exploiter pour la construction des requêtes. Pour chacune des questions, vous donnerez la requête construite, ainsi que son résultat.

2.2.1 Côté table ABC (5 points)

— Pour la table (1 point par question numérotée)

1. donner la taille d'un tuple de table et donner 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. donnez le nom 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), ainsi qu'un exemple de requête associée à cette vue ?

2.2.2 Côté index ABC_PK (6 points)

— Pour l'index (1 point par question numérotée)

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. donner le nom de la vue qui pourrait être consultée pour savoir si tous les blocs de l'index ABC_PK sont présents dans le cache de données

2.3 Comparaison pratique / théorie (2 points)

Commenter les résultats obtenus par rapport à ce que vous en saviez après calculs (exercice TD). Est ce cohérent ?

2.4 Exercice 2

2.4.1 Côté accès aux données de la table ABC (3 points)

Pensez vous que l'index est utilisé pour les deux ordres de consultation suivants ? Justifiez votre réponse et donnez le nombre de blocs à parcourir (de données et possiblement d'index dans les deux cas) pour satisfaire la requête.

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

2. select A, B from ABC where C like '%ABC%';
```

2.5 Exercice 3

2.5.1 Question ouverte PL/SQL (4 points)

Construisez une procédure PL/SQL qui vous semble à même de renvoyer les informations les plus importantes concernant l'organisation logique et physique d'une table et de ses index. Un plus sera de traiter les exceptions possibles et de construire un paquetage. Un exemple d'exécution et d'affichage des résultats est également attendu.

3. Annexe : questions de l'exercice 4 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). L'espace de bloc utilisable (8192-819.2=7372.8) est de 7372 octets.

3.1 Rappel des questions de l'exercice 4

3.1.1 Questions 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

3.1.2 Questions 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

Les résultats obtenus sont les suivants :

3.1.3 Correction pour la table

— facteur de blocage table = 7372 / 50 octets par tuple = 147,44 soit 147 tuples pouvant être pris en charge au sein d'un bloc
— Taille en blocs de la table 1000000/147 = 6802,72 soit 6803 blocs nécessaires
— 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

3.1.4 Correction pour l'index

— facteur de blocage de l'index = 491 (7372/15) : 491 clés d'index peuvent êtres stockées au sein d'un bloc/nœud.
— nombre de blocs nécessaires pour stocker l'index : au niveau feuilles 1000000/491 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). Au niveau intermédiaire : 2037/491 = 4.14 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 (feuilles) +5 (branches) +1 (racine) = 2043 blocs * 8192 octets (par bloc) = 16736256 octets soit environ 15,96 Mo (16736256/1024/1024)
— 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).