HMIN328 : révision

1. Optimisation de requêtes

1.1 Plan d'exécution d'une requête

Expliquer la sémantique associée à la requête suivante, construisez l'arbre algébrique correspondant et décrivez le plan d'exécution obtenu (au moyen d'autotrace).

```
select /*+ use_merge(c p) */ p.codeinsee, val_population from population p, commune c
where c.codeinsee = p.codeinsee and numdep ='34' and annee = 2010;
```

Côté sémantique, il s'agit de retourner le code insee et la valeur de la population en 2010 des communes de l'Hérault (numdep='34'). Côté requête SQL, une jointure naturelle est opérée sur l'attribut codeinsee présent à la fois dans commune et population. Deux conditions de sélection complètent la requête, une sur population et l'année 2010, et l'autre sur commune et le numéro de département. Les attributs codeinsee et val_population de la table population, sont les seuls projetés. Un index nommé num_idx est exploité sur numdep pour une recherche sur intervalle sur la valeur de numdep (='34'). Les tuples de la table commune sont aussi atteints via l'index unique, s'appliquant à commune, et via leur valeur de Rowid.

Opérations d'accès aux tables

1. TABLE ACCESS FULL : parcourir toutes les lignes d'une table
2. TABLE ACCESS HASH : parcourir les lignes d'une table à partir des clés de l'index hash
3. TABLE ACCESS BY INDEX ROWID : parcourir les lignes d'une table à partir des valeurs d'index
4. TABLE ACCESS BY USER ROWID : parcourir les lignes d'une table identifiées par leur ROWID

```
Statistiques
----------------------------------------------------------
        196  recursive calls
          0  db block gets
         48  consistent gets
          0  physical reads
          0  redo size
       1073  bytes sent via SQL*Net to client
        396  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
         16  rows processed
```

L'addition de *consistent gets et db block gets* donnent le nombre de blocs logiques lus (blocs de données et d'index), *physical reads* donne le nombre de blocs physiques lus (donc chargés en mémoire centrale). *recursive calls* concerne les appels à des ordres SQL qui vont se réveler nécessaires pour l'exécution de la requête (tris de données par exemple). Redo size est utile dans les ordres d'écriture et donc n'est pas exploité dans le contexte d'une consultation. Les tris, en particulier au niveau des mémoires disques (sorts (disk)), sont particulièrement coûteux lorsque les volumes de données sont importants (opérations de jointure (sort/merge), partitionnement (group by), tri (order by, distinct)).

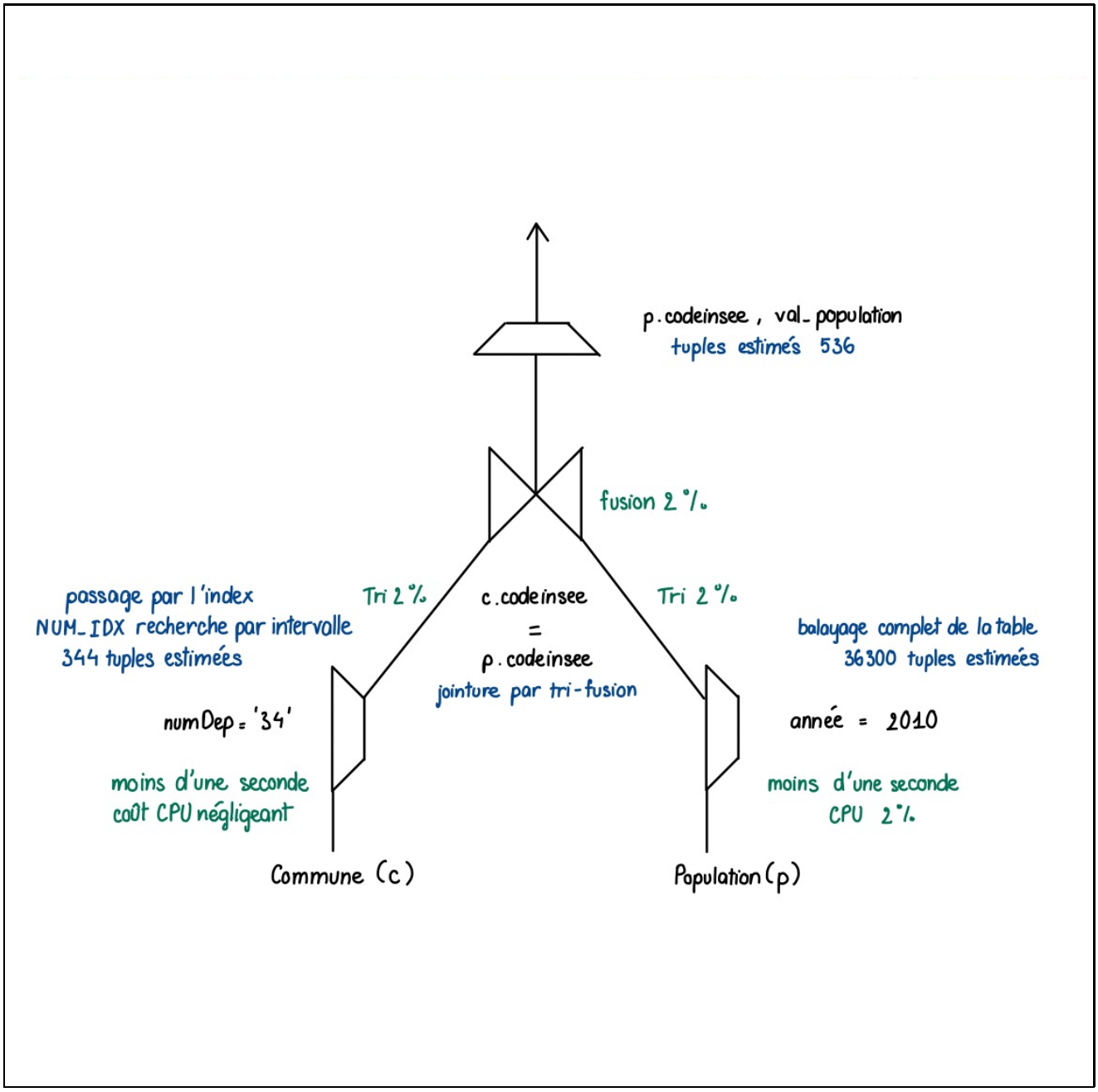

FIGURE 1 – Arbre algébrique et annotations

```
Plan d'execution
----------------------------------------------------------
Plan hash value: 696708763

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |            |   536 | 12864 |       |  1210   (2)| 00:00:01 |
|   1 |  MERGE JOIN                           |            |   536 | 12864 |       |  1210   (2)| 00:00:01 |
```

```
|   2 |   SORT JOIN                           |            |   344 |  3440 |       |    55   (2)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| COMMUNE    |   344 |  3440 |       |    54   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN                  | NUMD_IDX   |   344 |       |       |     1   (0)| 00:00:01 |
|*  5 |   SORT JOIN                           |            | 36300 |   496K|  1720K|  1155   (2)| 00:00:01 |
|*  6 |    TABLE ACCESS FULL                  | POPULATION | 36300 |   496K|       |   972   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

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

   4 - access("NUMDEP"='34')
   5 - access("C"."CODEINSEE"="P"."CODEINSEE")
       filter("C"."CODEINSEE"="P"."CODEINSEE")
   6 - filter("ANNEE"=2010)


Statistiques
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       3541  consistent gets
       3430  physical reads
          0  redo size
       8757  bytes sent via SQL*Net to client
        719  bytes received via SQL*Net from client
         24  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
        343  rows processed
```

1.2 Questions complémentaires

1. À quoi correspond la directive `use_merge(c p)` ?
   La directive force l'optimiseur à évaluer le coût de la requête en exploitant l'algorithme de tri-fusion pour le calcul de la jointure entre commune aliasée par c, et population aliasée par p
2. Donner les différents opérateurs physiques disponibles sous Oracle pour le calcul d'une jointure.
   Ils sont au nombre de 3 : tri-fusion (use_merge), boucles imbriquées (use_nl) et jointure par hachage (use_hash). Selon le contexte (volumétrie des tables, présence d'index, ...), l'un ou l'autre de ces opérateurs va être plus avantageux et donc préféré par l'optimiseur.
3. Est ce que le serveur est allé lire des blocs de données, en mémoire secondaire, pour satisfaire le calcul de la requête ? La lecture des statistiques montre que oui, 3430 blocs ont été chargés en mémoire principale depuis le disque (physical reads) pour être parcourus.
4. Combien de blocs de données sont parcourus au total ? La lecture des statistiques montre 3430 (depuis le disque) + 3541 (déja dans le database buffer cache), soit 6971 blocs ont été traités pour les besoins de la requête
5. Est ce qu'il y a une différence, et de combien, entre le nombre de tuples réellement retournés par la requête et le nombre de tuples estimés ?
   L'optimiseur estime à 536 le nombre de tuples répondant à la requête. En réalité, seuls 343 tuples font partie du résultat. Cette surestimation est un peu étonnante, d'autant qu'après sélection des communes de l'Hérault, l'estimation de 344 tuples est plutôt très correcte. La surestimation ne semble pas provenir du nombre inégal de communes par département, mais semble plutôt apparaître au moment de la fusion des tuples triés à joindre de part et d'autre.

6. La requête correspond à une semi-jointure. Donnez en deux écritures SQL équivalentes.
   semi-jointure avec test d'appartenance
   select codeinsee, val_population from population where annee = 2010 and codeinsee in (select codeinsee from commune where numdep='34') ;
   semi-jointure avec test de non vacuité
   select codeinsee, val_population from population p where annee = 2010 and exists (select null from commune c where p.codeinsee = c.codeinsee and numdep='34') ;

2. Index

1. Quel est le rôle d'un index ? Quelle est la structure d'index la plus exploitée dans les SGBDR ? Le rôle d'un index est d'améliorer les accès aux données afin de satisfaire les besoins des usagers de la manière la plus efficace possible. Plus l'index est discriminant et plus il va se révéler d'intérêt pour rendre le requêtage efficace (en temps et en coût de calcul). Tout est affaire de compromis, et il faut évaluer le coût supplémentaire de définition, réorganisation et stockage d'un index par rapport à ses rôles. Le B+Tree (ou arbre équilibré) est la structure la plus exploitée, mais on peut aussi retrouver des index de type bitmap ou table de hachage au sein des SGBDR. L'intérêt du B+Tree est d'être tout aussi efficace pour une recherche sur valeur exacte, ou sur une recherche par intervalle. Il peut de plus ne nécessiter q'une profondeur toute relative, et va se révéler très performant lorsque les tables comportent des milliers voire des millions de tuples.
2. Vous expliquerez l'ordre suivant :

   ```
   set verify off
   select index_name from dba_indexes where upper(table_name) = upper(&matable);
   ```

   retourne le nom des index s'appliquant à une table dont le nom est saisi par l'usager au moment de la consultation (usage d'une variable paramétrée)
3. Vous disposez d'une table EMP qui comporte un attribut NUM sur lequel est posé une contrainte de clé primaire.
   (a) EMP se voit appliquer un index unique ? (justifier votre réponse)
      i. OUI bonne réponse : un index unique de type B+Tree est défini automatiquement en complémentarité avec la définition de la contrainte de clé primaire
      ii. NON
   (b) Des requêtes de consultation vous sont données. Lesquelles vont bénéficier de l'index ?
      i. select num from emp ;
      oui : il suffit juste d'explorer l'index puisque seul num est demandé
      ii. select nom from emp where num = 20 ;
      oui, une recherche classique sur une valeur exacte sur la base de l'attribut indexé
      iii. select nom, fonction from emp ;
      non, l'attribut indexé n'est pas du tout mobilisé dans la requête
      iv. select nom from emp where num > 10 ;
      oui, une recherche classique sur une collection de valeurs sur la base de l'attribut indexé

3. Surcouche procédurale PL/SQL

3.1 Déclencheurs

Trois ordres de déclencheurs vous sont donnés. Un seul ordre est correct. Vous indiquerez lequel et vous expliquerez pourquoi les deux autres ne le sont pas.

```
create or replace trigger t1 before update
on commune
begin
:new.nom_com := lower(:old.nom_com) ;
end t1;
/
```

incorrect, ce déclencheur est déclaré comme opérant au niveau global (et non pas pour chaque tuple : absence de for each row). De ce fait, les prefixes :new et :old qui correspondent aux valeurs après ou avant d'attributs de tuples impactés par l'évènement déclenchant, ne peuvent pas être mobilisés.

```
create or replace trigger t1 before update
on commune
for each row
begin
:new.nom_com := upper(:old.nom_com) ;
end t1;
/
```

correct : le nom de la commune va être modifiée et persistée en majuscules avant (before) la mise à jour

```
create or replace trigger t1 after update
on commune
for each row
begin
:new.nom_com := upper(:old.nom_com) ;
end t1;
/
```

incorrect : le nom de la commune ne peut pas être modifiée et persistée en majuscules après (after) la mise à jour

3.2 Fonctions et procédures

Expliquez la différence entre une procédure et une fonction ?
Répondez au préalable aux questions suivantes (plusieurs réponses possibles)

1. Les procédures et fonctions doivent retourner des valeurs
2. Les procédures et fonctions peuvent retourner des valeurs
3. Les procédures peuvent retourner des valeurs
4. Les fonctions doivent retourner des valeurs

les réponses correctes sont les réponses 3 et 4

4. Architecture Oracle

4.1 Eléments mis en jeu

Une figure incomplète de l'organisation physique d'une base de données Oracle vous est donnée. La vision client-serveur est celle d'un serveur dédié. Vous compléterez les informations manquantes en ce qui concerne les structures, fichiers et processus mis en jeu.

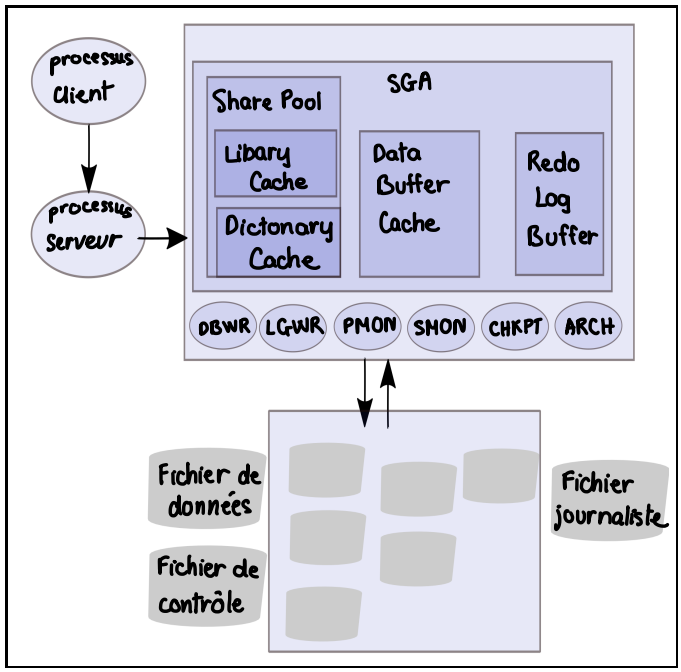

FIGURE 2 – Architecture physique Oracle (structures et processus)

Vous aurez à positionner en complément des composants généraux (serveur, instance, base de données) les éléments suivants :

— structures mémoire de l'instance
  1. SGA
  2. shared pool
  3. database buffer cache
  4. redo log buffer
  5. library cache
  6. dictionary cache

— fichiers de la base de données
  1. fichiers de données
  2. fichiers de contrôle
  3. fichiers journaux

— processus

1. db writer
2. log writer
3. chkpt (checkpoint)
4. smon
5. pmon

voir le cours

4.2 Superviser les structures et éléments mis en jeu

La même figure vous est à nouveau proposée. Vous indiquerez maintenant les vues du méta-schéma qui servent à renseigner sur les différentes structures mémoire et processus d'arrière plan (ou autres processus). Pour vous faciliter la tâche, une liste de vues vous est fournie.

1. v$instance : vue portant sur l'instance de la base de données (l'instance étant composée des structures mémoires et des processus du serveur de données)
2. v$database : vue portant sur la base de données (la base de données étant composée des fichiers (données, journaux, contrôles, ...) sur disque)
3. v$bgprocess : vue portant sur les processus d'arrière-plan (background ou bg)
4. v$sgainfo : vue portant sur la mémoire nommée System Global Area qui va se décomposer en différentes sous-structures
5. v$sgastat : vue portant sur la mémoire nommée System Global Area qui va se décomposer en différentes sous-structures (complémentaire à la vue précédente)
6. v$log : vue portant sur les fichiers journaux (dits log files)
7. vsql, vsqlarea, v$sqltext : vues portant sur la sous-structure library cache de la shared pool. Ces vues renseignent sur les dernières requêtes exécutées par les différents usagers du système
8. v$datafile : vue portant sur les fichiers de données, dans lesquelles sont persistées toutes les données de la base de données
9. v$logfile : vue portant sur les fichiers journaux qui sont importants pour le recouvrement de la base de données lors de pannees diverses
10. v$controlfile : vue portant sur les fichiers de contrôle qui vont consigner toutes les informations essentielles au bon fonctionnement de la BD
11. v$bh : vue portant sur le cache de données (database buffer cache) et renseignant sur les blocs montés en mémoire vive, consultés et/ou modifiés
12. v$cache : vue portant également sur le cache de données (database buffer cache)

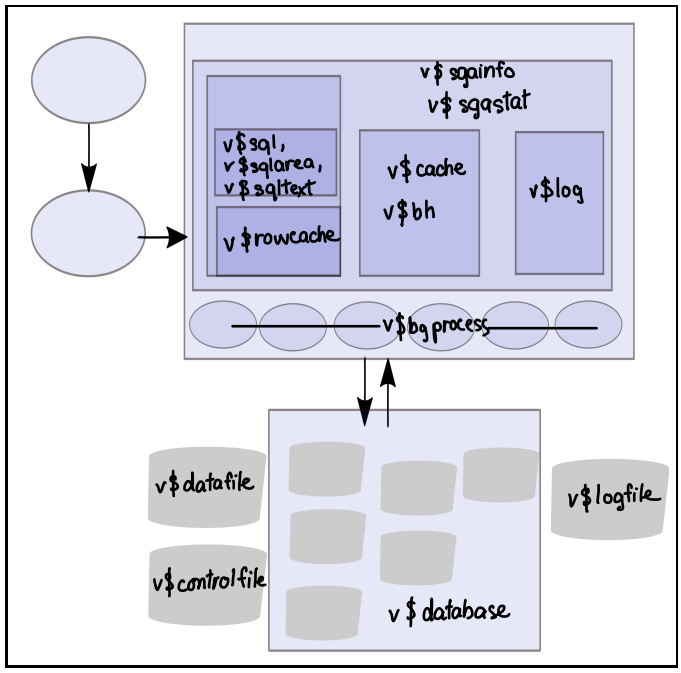

FIGURE 3 – Architecture physique Oracle (vues du méta-schéma associées)

4.3 Organisation logique

1. A quoi correspond la notion d'espace de table (tablespace) ? En quoi se décompose t'elle ? C'est un notion importante qui permet d'organiser "logiquement" les tables et de leur contenu, et de s'affranchir de l'organisation physique tout en permettant au système d'être très efficace. Ainsi une table se situe dans un seul espace de table et les blocs de données correspondant à ces tables vont se situer dans un même segment composé de blocs de données. Un espace de tables se compose de multiples segments. Chaque segment est ensuite constitué de un à plusieurs extents (extensions) qui sont des blocs de données contigûs. Ainsi, il est simple d'aller chercher tous les blocs correspondant à une table, en allant d'abord au niveau de l'espace de table, puis du segment, puis des extents et enfin des blocs. La notion d'extent permet d'avoir de la flexibilité dans la manière de gérer la volumétrie changeante de la table.
2. Comment connaître le nom de l'espace de table dans lequel les tables de votre schéma utilisateur sont organisées ? idem pour les index ? en consultant les vues user_tables et user_indexes
   par exemple : select tablespace_name from user_indexes ;
3. Quels sont les types de segments les plus habituels ?
   avec la requête :
   select distinct segment_type from dba_segments ;
   on pourra noter que les types sont par exemple : INDEX, TABLE, ROLLBACK, INDEX PARTITION (table de hachage), ...
4. Comment savoir à quel fichier de données correspond un espace de table ? avec la requête :
   select file_name, tablespace_name from dba_data_files ;
5. Quel est le paquetage qui vous renseigne sur le bloc de données qui contient un tuple précis d'une table en particulier ? le paquetage dbms_rowid

4.4 Library Cache

1. Donnez votre compréhension de la requête qui suit

```
set linesize 300

SELECT substr(sql_text,1,60), (cpu_time/100000) "Cpu Time (s)",
(elapsed_time/1000000) "Elapsed time (s)",
fetches, buffer_gets, disk_reads, executions
FROM v$sqlarea
WHERE Parsing_Schema_Name =USER AND ROWNUM < 50
ORDER BY 3 DESC;
```

la requête en question renseigne sur les différents ordres passés par l'usager (USER). Les 60 premiers caractères, ainsi que le coût en temps CPU, en temps écoulé, en nombre de blocs parcourus en mémoire et à partir du disque, ainsi que le nombre d'exécutions de chaque ordre sont renvoyés. Il est à noter que les différents coûts sont une addition des coûts des différentes exécutions.

4.5 Vues statiques et dynamiques du dictionnaire de données

Vous expliquerez, dans un premier temps, les différences entre les vues statiques et les vues dynamiques. Pour ce qui concerne les vues statiques, vous expliquerez dans un second temps, la différence entre les vues préfixées respectivement par USER_, ALL_ et DBA_. Les vues statiques, aussi appelées vues structurelles renseignent sur l'organisation des objets (tables, contraintes, attributs, index, usagers, procédures, fonctions, déclencheurs, ...) organisés au sein de la BD. Les vues dynamiques renseignent sur l'état du système à l'état présent. Il est ainsi possible de savoir qui est connecté (v$session) et depuis quand, ou encore de savoir si des transactions sont en conflit (v$lock). Les vues préfixées par USER concernent les objets de notre schéma, les vues préfixées par ALL concernent les objets avec lesquels nous pouvons interagir, les vues préfixées par DBA concernent tous les objets de la base.

— Vous souhaitez connaître les tables qui vous sont disponibles en consultation. Quelle est la vue qu'il vous faut interroger à ce sujet ?

1. USER_TABLES
2. DBA_TABLES
3. ALL_TABLES

il faut consulter ALL_TABLES qui contient les informations sur les tables de notre schéma utilisateur mais aussi sur les tables sur lesquelles nous avons des droits (au moins de consultation) .

— Vous souhaitez connaître les tables qui vous sont accessibles en consultation mais qui ne vous appartiennent pas. Donner un exemple de requête qui vous permettrait d'avoir cette information select table_name from all_tables minus select table_name from user_tables ;

— A quoi correspond la notion de session ? Quelles sont les notions qui vous semblent associées à session ? La notion de session correspond à l'intervalle de temps pendant lequel l'usager est connecté et va interagir avec le serveur de données. Une session débute par une ouverture de connexion et se termine par une fermeture de connexion. De manière générale, l'usager est identifié via un identifiant et un mot de passe, et possède un certain nombre de privilèges système qui vont lui permettre de disposer de différents droits sur le système. La session peut se retrouver en conflit d'écriture avec d'autres sessions. La session est donc associée aux notions de connexion, d'usager, de droits d'usager, de contrôle de concurrence et de transaction.

5. Requêtes SQL

Un ensemble de questions vous sont posées. Vous chercherez à construire les requêtes apportant des éléments de réponse à ces questions.

1. Donner les espaces mémoires exploités par utilisateur, triés du plus petit consommateur au plus grand (dba_segments)
   select owner, sum(bytes)/1024/1024 as usedSpace from dba_segments where tablespace _name ='DATA_ETUD' group by owner order by 2 ;
2. Quel utilisateur consomme l'espace de stockage mémoire le plus important (dba_segments et tablespace DATA_ETUD) ?
   select owner, sum(blocks) as nbBlocks, count(segment_name) as NbObjets, sum(blocks)*8192/1024/1024 as Mo from dba_segments where tablespace_name = 'DATA_ETUD' group by owner having sum(blocks) >= ALL (select sum(blocks) from dba_segments where tablespace_name = 'DATA_ETUD' group by owner) ;
3. Qui possède le plus d'index dans son schéma utilisateur (dba_segments) ?
   select owner, count(distinct segment_name) as NbreIndex from dba_segments where tablespace_name = 'DATA_ETUD' and segment_type = 'INDEX' group by owner having count(segment_name) >= ALL (select count(distinct segment_name) from dba_segments where tablespace_name = 'DATA_ETUD' and segment_type = 'INDEX' group by owner) ;
4. Qui n'a pas créé d'objets dans son schéma utilisateur depuis plus de 3 mois (dba_objects)
   select owner from dba_objects where created <sysdate-90 minus select owner from dba_objects where created >=sysdate-90 ;
5. Quels sont les objets et leurs propriétaires qui n'ont pas connu d'évolutions depuis plus de 3 mois (dba_objects)
   select owner, object_name, last_ddl_time from dba_objects
   where last_ddl_time<sysdate-90 and owner like 'E%' minus select owner, object_name, last_ddl_time from dba_objects where last_ddl_time>=sysdate-90 ;
6. Quels sont les usagers qui écrivent respectivement sur le tablespace SYSTEM, SYSAUX et UNDOTBS1 (dba_segments)
   select distinct owner, tablespace_name from dba_segments where tablespace_name in ('SYSTEM','SYSAUX','UNDOTBS1') ;
7. Quel(s) usager(s) a deux sessions (voire plus ouvertes (v$session))
   select username, osuser from v$session group by username, osuser having count(*) >=2 ;
8. Quel(s) usager(s) a une session, qui a posé des verrous bloquants pour une autre session (v$lock et v$session)
   select username, s.sid from v*sessions*, *v*lock l where s.sid = l.sid and block = 1 ;
9. Quelles sont les requêtes que j'ai exécutées qui sont encore prises en charge au niveau de la library cache (v$sql)
   select sql_id, substr(sql_text,1,40), cpu_time/1000000, disk_reads, buffer_gets from v$sql where parsing_schema_name = USER order by 3 ;
10. Quelles sont les requêtes que j'ai exécutées plusieurs fois qui sont encore prises en charge au niveau de la library cache (v$sql)
    select sql_id, substr(sql_text,1,20), executions as rejouees, cpu_time/1000000, disk_reads, buffer_gets from v$sql where parsing_schema_name = USER and executions > 2 order by 3 ;
11. Quels sont les objets de la base qui sont invalides et qui peuvent entrainer des points de contention au niveau du dictionary cache (dba_objects et status)
    select object_name, owner from dba_objects where status ='INVALID' ;

12. Quels sont mes privilèges utilisateur (user_tab_privs)
    select grantee, grantor, owner, table_name, privilege from user_tab_privs ;
13. Quels sont mes privilèges systèmes (user_sys_privs)
    select username, privilege from user_sys_privs ;
14. Quels sont les rôles qui m'ont été attribués (dba_roles et dba_sys_privs)
15. Quels sont les privilèges que me donnent ces rôles (dba_role_privs)
    incomplet
    select distinct username, granted_role from user_role_privs ;
16. Quel objet, et de quel utilisateur, utilise le plus de blocs du cache de données (dba_objects, v$bh)
    incomplet
    select objd, count(block#), object_name, owner from v$bh, dba_objects where objd = object_id and owner not in ('SYS', 'SYSMAN', 'XDB') group by objd, object_name, owner order by 2 ;
17. Comment savoir si tous les blocs de ma table EMP sont dans le cache de données (user_tables et v$bh) (rafraîchir les statistiques en analysant la table) ?
18. Quelles sont mes dix requêtes récentes les plus coûteuses