TP Optimisation de requêtes

1. Préambule

Le schéma comprend les tables Commune, Departement et Region. Le schéma relationnel est indiqué de manière succincte.

- COMMUNE (**CODEINSEE**, CODEPOSTAL, NUMDEP, NOMCOMMAJ, NOMCOMMIN, LATITUDE, LONGITUDE)
- DEPARTEMENT(**NUMDEP**, CHEFLIEU, NUMREG, NOMDEPMAJ, NOMDEPMIN)
- REGION(**NUMREG**, CHEFLIEU, NOMREGMAJ)

1.1 Disposer des trois tables et d'index uniques associés aux trois clés primaires

Vous disposez déjà des tables COMMUNE et DEPARTEMENT. Vous définirez la table REGION à partir de l'instruction :

```
create table REGION as select * from P00000009432.REGION ;
-- ne pas oublier de rajouter la contrainte de cle primaire et donc d'index unique
alter table REGION add constraint REGIONPK primary key(numreg) ;
```

Vérifiez que tous les index uniques associés aux contraintes de clé primaire COMMUNEPK, DEPARTEMENTPK et REGIONPK sont bien définis (consulter les vues USER_INDEXES et USER_CONSTRAINTS)

1.2 Exercices sur machine

Testez tous les plans d'exécution à partir de l'interpréteur sqlplus d'Oracle. Utilisez également les directives pour guider/forcer le plan d'exécution et ainsi évaluer le coût de différentes requêtes conduisant aux mêmes résultats. Des exemples d'utilisation de directives sont données dans le cours. Pour l'ensemble des exercices, exploitez les commandes :

1. set autotrace on (résultats, plan d'exécution et statistiques) ou set autotrace traceonly (seulement plan d'exécution)
2. explain plan for select ... puis
   set linesize 300
   set pagesize 100
   select * from table(dbms_xplan.display()) ;
3. SELECT /*+ GATHER_PLAN_STATISTICS */ ...
   puis
   SELECT * FROM
   TABLE(DBMS_XPLAN.display_cursor(format=>'ALLSTATS LAST +cost +outline')) ;

2. Exercice 1

Huit requêtes SQL sont données, et vous en obtiendrez les plans d'exécution.

```
-- query 1
select nomcommaj, codeinsee from commune, departement
where codeinsee = cheflieu
minus
select nomcommaj, codeinsee from commune, region
where codeinsee = cheflieu;

-- query 2
select nomcommaj, codeinsee from commune where
codeinsee in (select cheflieu from departement)
and codeinsee not in (select cheflieu from region);

-- query 3
select nomcommaj, codeinsee from commune where
exists (select null from departement where codeinsee=cheflieu)
and not exists (select null from region r where codeinsee=cheflieu);

-- query 4
select nomcommaj, codeinsee from commune, departement
where codeinsee = cheflieu
and codeinsee not in (select cheflieu from region);

--query 5
select nomcommaj, codeinsee from commune left join departement on codeinsee = cheflieu
where cheflieu is not null
and codeinsee not in (select cheflieu from region);

--query 6
select nomcommaj, codeinsee from commune join departement on codeinsee = cheflieu
minus
select nomcommaj, codeinsee from commune join region on codeinsee = cheflieu;

--query 7
select nomcommaj, codeinsee from commune left join departement on codeinsee = cheflieu
where decode(cheflieu,null,'non','oui') = 'oui'
and codeinsee not in (select cheflieu from region);

--query 8
select nomcommaj, codeinsee from commune, (select cheflieu from departement) d
where codeinsee = d.cheflieu
minus
select nomcommaj, codeinsee from commune, (select cheflieu from region) r
where codeinsee = r.cheflieu;
```

1. Est ce que ces requêtes vous semblent équivalentes (c.a.d. donnent le même résultat à partir des mêmes données) ? Donnez la sémantique naturelle associée à chacune de ces requêtes. Vous pouvez vous aider avec la construction d'un arbre algébrique.

Elles sont équivalentes et renvoient toutes le même résultat à partir des mêmes données : à savoir le code insee et le nom des communes qui sont chef-lieu de département mais pas de région.

---

2. Est ce que l'optimiseur fait des choix différents pour chaque requête concernant le plan d'exécution ? Comment le savoir ?

L'optimiseur construit un plan d'exécution différent pour les 5 premières requêtes. Les requêtes 6 et 8 sont des variantes de 1 avec un usage des différentes syntaxes SQL 89, 92 et 99 pour la jointure (même plan) et la requête 7 est une variante de 5 exploitant une jointure externe gauche (même plan). Pour exemple de query 2 et query 3, la valeur du "plan hash value" est différente même si certains plans sont quasi identiques. A mon avis, ces deux plans sont vus comme différents juste parce que les valeurs non renseignées (null) ne sont pas traitées de la même manière au travers d'un anti-jointure mettant en œuvre les prédicats not in et not exists. "Not exists" renvoie les tuples pour lesquelles la valeur de la colonne impliquée dans l'anti-jointure est non renseignée alors que ce n'est pas le cas pour "not in" (voir `http://ahmedaangour.blogspot.com/2011/06/anti-joins-not-in-vs-not-exists.html`

---

3. Renvoyez le plan d'exécution qui vous semble le plus performant et commentez le. À cet effet, décrivez les opérations dans l'ordre d'enchaînement de ces opérations. Quelle est la table exploitée en premier lieu. Citez un opérateur physique mobilisé et indiquer son rôle. Est ce que les estimations en terme de tuples résultants sont correctes ?

Le plan le plus coûteux est de loin celui de query 1 qui fait appel à une différence. Les plans des query 2 et 3 sont les meilleurs d'un point de vue optimisation. Au regard du plan de la requête 3, l'optimiseur fait un tri unique (SORT UNIQUE) sur les valeurs toutes distinctes de cheflieu puis exploite l'index unique sur codeinsee (ici commune_pk) de Commune avant d'exploiter une opération de jointure de type boucles imbriquées (nested loops). Les tuples des communes satisfaisant la condition de jointure (codeinsee = cheflieu de departement) sont ensuite accédés via les blocs feuilles de l'index et leurs valeurs rowid (identifiants physiques). L'estimation de 1 seul tuple satsifaisant le filtre est totalement erronée. Ensuite le résultat de cette première jointure qui est une semi-jointure est joint (sous forme d'anti-jointure) avec la table region (filtre exluant les tuples satisfaisant codeinsee = cheflieu de region). L'opérateur physique choisi est une table de hachage en mémoire vive (hash join). Là encore, l'estimation est fausse car 101 tuples est trop optimiste sachant que certains chefs lieux de departement sont chef lieux de region.

```
explain plan for select nomcommaj, codeinsee from commune where
  exists (select null from departement where codeinsee=cheflieu)
  and not exists (select null from region where codeinsee=cheflieu);

Explicite.

select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
Plan hash value: 250767370

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
```

```
|   0 | SELECT STATEMENT              |             |   101 |  3131 |    65   (2)| 00:00:01 |
|*  1 |  HASH JOIN ANTI               |             |   101 |  3131 |    65   (2)| 00:00:01 |
|   2 |   NESTED LOOPS                |             |   101 |  2525 |    62   (2)| 00:00:01 |
|   3 |    NESTED LOOPS               |             |   101 |  2525 |    62   (2)| 00:00:01 |
|   4 |     SORT UNIQUE               |             |   101 |   606 |     3   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL        | DEPARTEMENT |   101 |   606 |     3   (0)| 00:00:01 |
|*  6 |     INDEX UNIQUE SCAN         | COMMUNE_PK  |     1 |       |     0   (0)| 00:00:01 |
|   7 |    TABLE ACCESS BY INDEX ROWID| COMMUNE     |     1 |    19 |     1   (0)| 00:00:01 |
|   8 |   TABLE ACCESS FULL           | REGION      |    18 |   108 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - access("CODEINSEE"="CHEFLIEU")
   6 - access("CODEINSEE"="CHEFLIEU")
```

4. Que pourriez vous ajouter au schéma pour améliorer les performances ? Tester vos ajouts en matière d'index.

Définition d'index uniques pour cheflieu sur les tables departement et region ou bien des contraintes d'unicité sur cheflieu qui conduiraient aussi à la définition d' index uniques. Par exemple :
create unique index cl_idx on departement (cheflieu) ;
alter table region add constraint reg_unique unique (cheflieu) ;
voir les index construits avec : select index_name from user_indexes ;

```
set linesize 200
set pagesize 200
select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------------
Plan hash value: 49651575

-------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |   101 |  3131 |   104   (1)| 00:00:01 |
|   1 |  NESTED LOOPS ANTI  |            |   101 |  3131 |   104   (1)| 00:00:01 |
|*  2 |   HASH JOIN         |            |   101 |  2525 |   104   (1)| 00:00:01 |
|   3 |    INDEX FULL SCAN  | CL_IDX     |   101 |   606 |     1   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| COMMUNE    | 35358 |   656K|   103   (1)| 00:00:01 |
|*  5 |   INDEX UNIQUE SCAN | REG_UNIQUE |     1 |     6 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - access("CODEINSEE"="CHEFLIEU")
   5 - access("CODEINSEE"="CHEFLIEU")

Note
-----
   - this is an adaptive plan
```

Nous retrouvons bien les mêmes erreurs d'estimation, par contre les nouveaux index construits (cl_idx et reg_unique) sont bien exploités. Le plan compte moins d'opérations et est donc moins coûteux au final. Certaines opérations sont cependant plus chères comme le balayage complet de commune ou l'anti-jointure via des boucles imbriquées.

5. Que pourraient apporter les directives exploitées ci-dessous à la requête ?

```
set pagesize 100
set linesize 200
select nomcommaj, codeinsee from commune where
exists (select /*+ hash_sj */ null from departement where codeinsee=cheflieu)
and not exists (select /*+ nl_aj */ null from region where codeinsee=cheflieu);
```

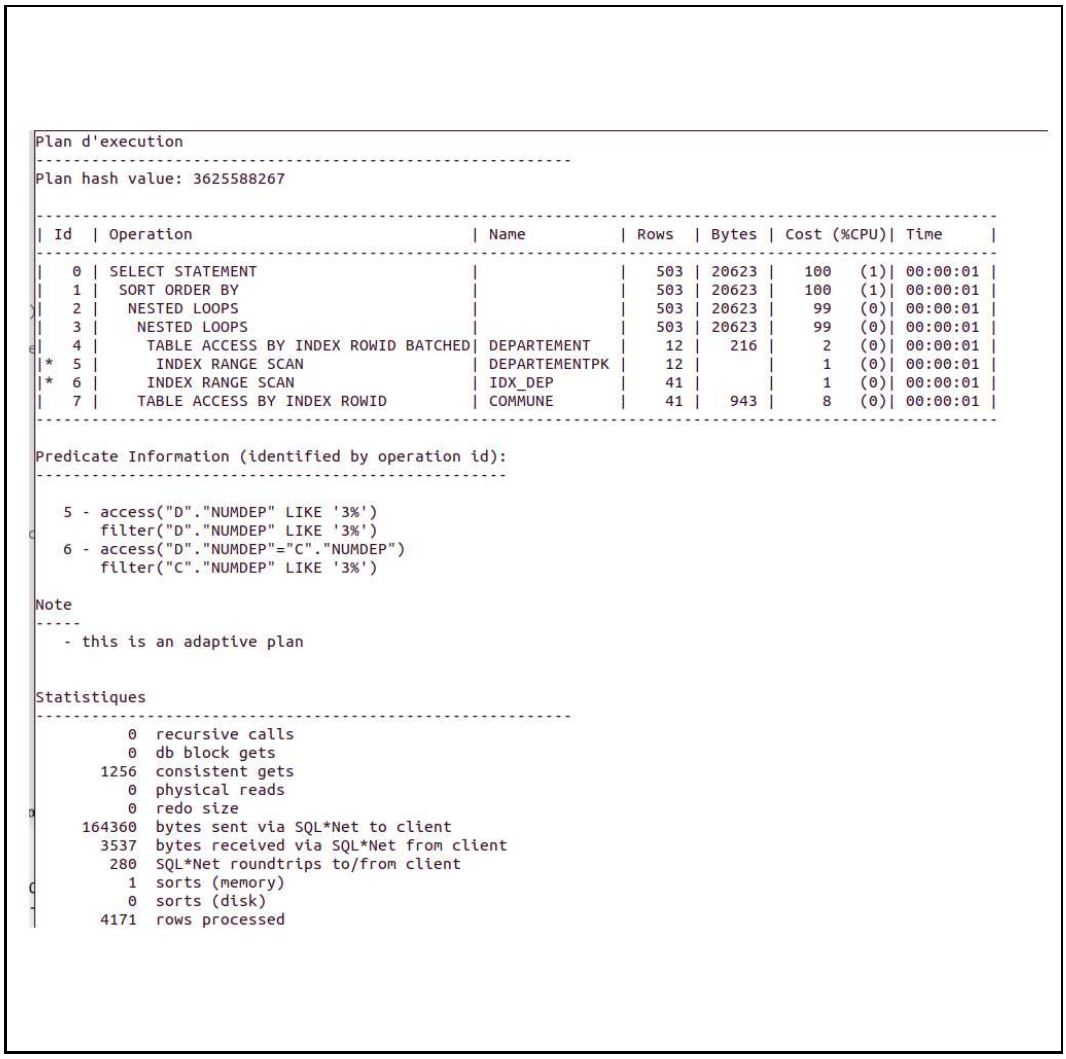
Les directives ajoutées permettent de traiter la semi jointure avec un opérateur de semi-jointure mobilisant une jointure par hachage pour le premier test d'appartenance et une anti-jointure via des boucles imbriquées pour le test de vacuité mais ce dernier point est déjà pris en charge dans le dernier exemple.

3. Exercice 2 (vu en TD)

Un plan d'exécution est donné. Il a été obtenu après définition d'un index non unique sur l'attribut numdep de la table Commune.

```
create index idx_dep on commune (numdep);
```

Les attributs projetés sont codeinsee, nomcommaj, numdep (de la table Departement), nomdepmaj, numreg.



```
Plan d'execution
----------------------------------------------------------
Plan hash value: 3625588267

-------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |               |   503 | 20623 |   100   (1)| 00:00:01 |
|   1 |  SORT ORDER BY                         |               |   503 | 20623 |   100   (1)| 00:00:01 |
|   2 |   NESTED LOOPS                         |               |   503 | 20623 |    99   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                        |               |   503 | 20623 |    99   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| DEPARTEMENT   |    12 |   216 |     2   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                  | DEPARTEMENTPK |    12 |       |     1   (0)| 00:00:01 |
|*  6 |     INDEX RANGE SCAN                   | IDX_DEP       |    41 |       |     1   (0)| 00:00:01 |
|   7 |    TABLE ACCESS BY INDEX ROWID         | COMMUNE       |    41 |   943 |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   5 - access("D"."NUMDEP" LIKE '3%')
       filter("D"."NUMDEP" LIKE '3%')
   6 - access("D"."NUMDEP"="C"."NUMDEP")
       filter("C"."NUMDEP" LIKE '3%')

Note
-----
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       1256  consistent gets
          0  physical reads
          0  redo size
     164360  bytes sent via SQL*Net to client
       3537  bytes received via SQL*Net from client
        280  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
       4171  rows processed
```

FIGURE 1 – Plan d'exécution

1. Quels sont les index mobilisés ?

   Les index departementpk et idx_dep l'index non unique nouvellement défini sont mis à contribution. Le balayage se fait par intervalle sur les deux index (range scan) en démarrant par la table departement. Les tuples de table satisfaisant le filtre sur numdep sont accédés via l'index (les blocs feuilles et les valeurs rowid)

2. Quel est l'opérateur physique de jointure choisi par l'optimiseur ? Discutez le choix de cet opérateur. Est ce que les estimations sur le nombre de tuples retournés sont correctes ? Comment faire pour vérifier que certaines de ces estimations sont incorrectes ?

   L'opérateur physique = boucles imbriquées. Cet opérateur est typiquemet utilisé quand des index sont présents sur les attributs impliqués dans la jointure et qu'une des deux tables présente une faible volumétrie. Les estimations sont incorrectes par exemple pour le nombre de départements satisfaisant le filtre (cela devrait etre 10) et le nombre de communes associées

(beaucoup plus que 41) et beaucoup plus que 501 au final (résultat dans les statistiques de 4171 tuples véritablement renvoyés)
Pour le vérifier (tuples reels A-Rows pour Actual Rows)
set pagesize 200
set linesize 200
SELECT /*+ GATHER_PLAN_STATISTICS */ codeinsee, nomcommaj, d.numdep, nomdepmaj, numreg from departement d, commune c where d.numdep = c.numdep and d.numdep like '3%' order by numreg ;
SELECT * FROM TABLE(DBMS_XPLAN.display_cursor(format=>'ALLSTATS LAST +cost +outline')) ;

3. Donner la sémantique naturelle de la requête SQL qui a donné lieu à ce plan

Retourne le code insee, le nom de la commune, le num du département, le nom du département et le numéro de region pour les communes qui sont localisées dans un département dont le numéro commence par 3, le tout trié sur le numéro de région (le tri peut se deviner (sort by) mais l'attribut sur lequel s'effectue le tri ne peut pas se deviner (cependant obligatoirement un attribut projeté).

4. Proposer la requête SQL qui vous semble à l'origine de ce plan.

select codeinsee, nomcommaj, d.numdep, nomdepmaj, numreg from departement d, commune c where d.numdep = c.numdep and d.numdep like '3%' order by numreg

5. Proposer une écriture de la requête intégrant des directives permettant de changer d'opérateur physique de jointure, et de table guidant la jointure.

explain plan for select /* use_hash(c d) ordered */ codeinsee, nomcommaj, d.numdep, nomdepmaj, numreg from commune c, departement d where d.numdep = c.numdep and d.numdep like '3%' order by numreg ;

4. Exercice 3

Une requête de consultation est donnée sous sa forme algébrique.
$\Pi_{nomCommaj,nomDepMaj,nomRegMaj,d.chefLieu,r.chefLieu}$ (Commune c $\bowtie$ Departement d $\bowtie$ Region r)

1. donner la sémantique naturelle de cette requête

renvoie le nom de la commune, le nom du département, le nom de la région et le code insee des chefs lieux de département et de région

2. donner une écriture SQL de cette requête

Des écritures possibles
–1
select nomcommaj, nomdepmaj, nomregmaj, d.cheflieu, r.cheflieu from commune c, departement d, region r

where c.numdep = d.numdep and d.numreg = r.numreg ;
– 2
select nomcommaj, nomdepmaj, nomregmaj, d.cheflieu, r.cheflieu from commune c join departement d on c.numdep = d.numdep join region r on d.numreg = r.numreg ;

explain plan for select nomcommaj, nomdepmaj, nomregmaj, d.cheflieu, r.cheflieu from commune c, departement d, region r
where c.numdep = d.numdep and d.numreg = r.numreg ;
set linesize 300
set pagesize 100
select * from table(dbms_xplan.display()) ;

---

3. évaluer et expliquer le plan d'éxecution proposé par l'optimiseur (tracer les opérations réalisées, indiquer si les index sont mobilisés et lesquels, indiquer si les tables sont parcourues dans leur globalité)

Comme les tables doivent être parcourues dans leur globalité, finalement l'optimiseur n'utilise que regionpk et une opération de tri-fusion entre region et departement, ensuite opération de hachage en mémoire vive entre le résultat de cette première jointure et commune sur la base de numdep.

---

4. exploiter autotrace ou gather_plan_statistics pour exécuter la requête et non pas seulement estimer son exécution. Donnez quelques explications des statistiques obtenues : nombre de blocs parcourus depuis le cache de données, nombre de blocs parcourus depuis le disque, temps de calcul, temps total pour la restitution des résultats

---

Exemple d'actions à commenter
select /*+ GATHER_PLAN_STATISTICS */ nomcommaj, nomdepmaj, nomregmaj, d.cheflieu, r.cheflieu from commune c, departement d, region r
where c.numdep = d.numdep and d.numreg = r.numreg ;
SELECT * FROM TABLE(DBMS_XPLAN.display_cursor(format=>'ALLSTATS LAST +cost +outline')) ;
Valeur de hachage du plan ici 2064521880

select cpu_time/1000000 cpuSecs, elapsed_time/1000000 elapsedSecs, executions, sql_id, disk_reads, buffer_gets from v$sql where plan_hash_value = 2064521880 ;

---

5. expliquer pourquoi la requête de consultation suivante, également donnée sous sa forme algébrique, pourrait donner lieu à un plan d'exécution moins coûteux
$\Pi_{nomCommaj,d.numDep,r.numReg}$ (Commune c $\bowtie$ Departement d $\bowtie$ Region r)

---

explain plan for select nomcommaj, d.numdep, r.numreg from commune c, departement d, region r where c.numdep = d.numdep and d.numreg = r.numreg ;
le système n'a pas à parcourir Region et d'ailleurs nous n'avions pas besoin de Region pour construire cette requête puisque numreg est dans la table Departement. L'optimiseur pallie en partie notre surcoût d'ecriture, en utilisant l'index regionpk de region. C'est cependant plus

coûteux que si nous avions écrit la requête de manière plus concise car il faut cependant balayer departement pour numreg et commune pour nomcommaj.
explain plan for select nomcommaj, d.numdep, numreg from commune c, departement d where c.numdep = d.numdep ;