HMIN122M Rendu
TP2

Bachar Rima Joseph Saba Tasnim Muhammad

27 septembre 2018

Table des matières

1 Sélection

1.1 Question 1

@script_table.sql : création des tables de la base de données.

@script_remplissage.sql : remplissage des tables créées.

set autotrace on : faire des statistiques sur les requêtes exécutées.

set linesize 200 : étendre la taille des tables affichées pour une meilleure visibilité.

1.2 Question 2

```
1  explain plan for select nom from ville where
       insee='34172';
2  select plan_table_output from
       table(dbms_xplan.display());
```

Listing 1 – requêtes permettant d'expliquer le plan d'exécution affichant le nom des villes dont le numéro insee est 34172

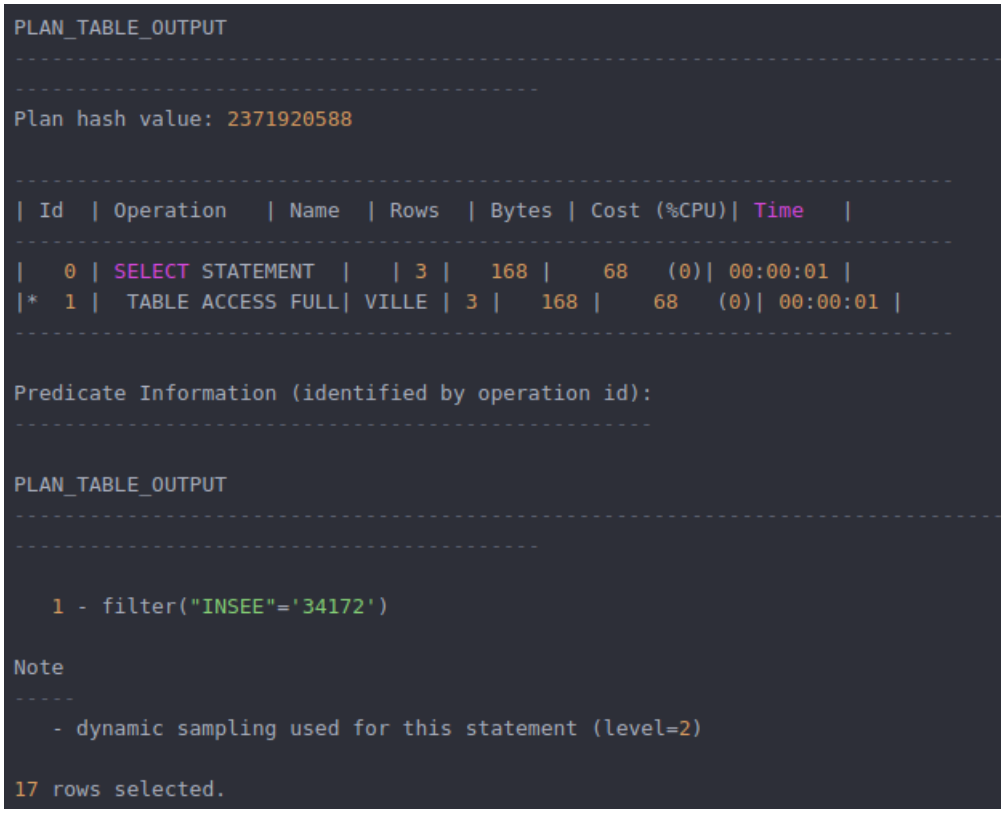


FIGURE 1 – Plan d'exécution

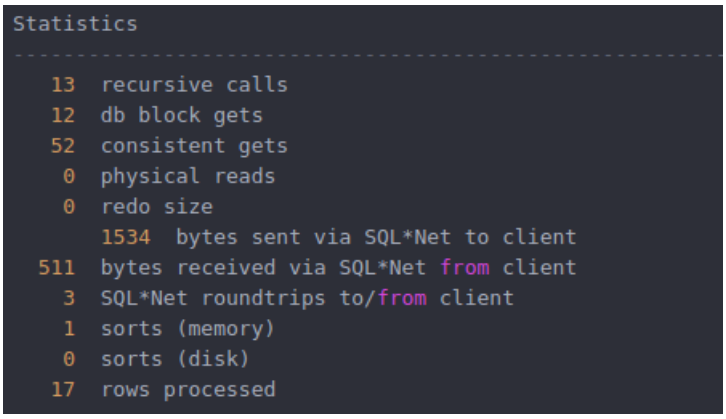


FIGURE 2 – Statistiques

1.3 Question 3

```
1   alter table ville add constraint PK_VILLE primary
        key(insee);
```

Listing 2 – ajout d'une clé primaire sur la table Ville en utilisant l'attribut insee

1.4 Question 4

Avant l'ajout d'une clé primaire à la table `Ville`, l'algorithme choisi par l'optimiseur pour la requête est *Table Access Full*. Ce dernier est généralement gourmand en termes de consommation de ressources, vu qu'il doit balayer la table entièrement, ligne par ligne, et attribut par attribut.

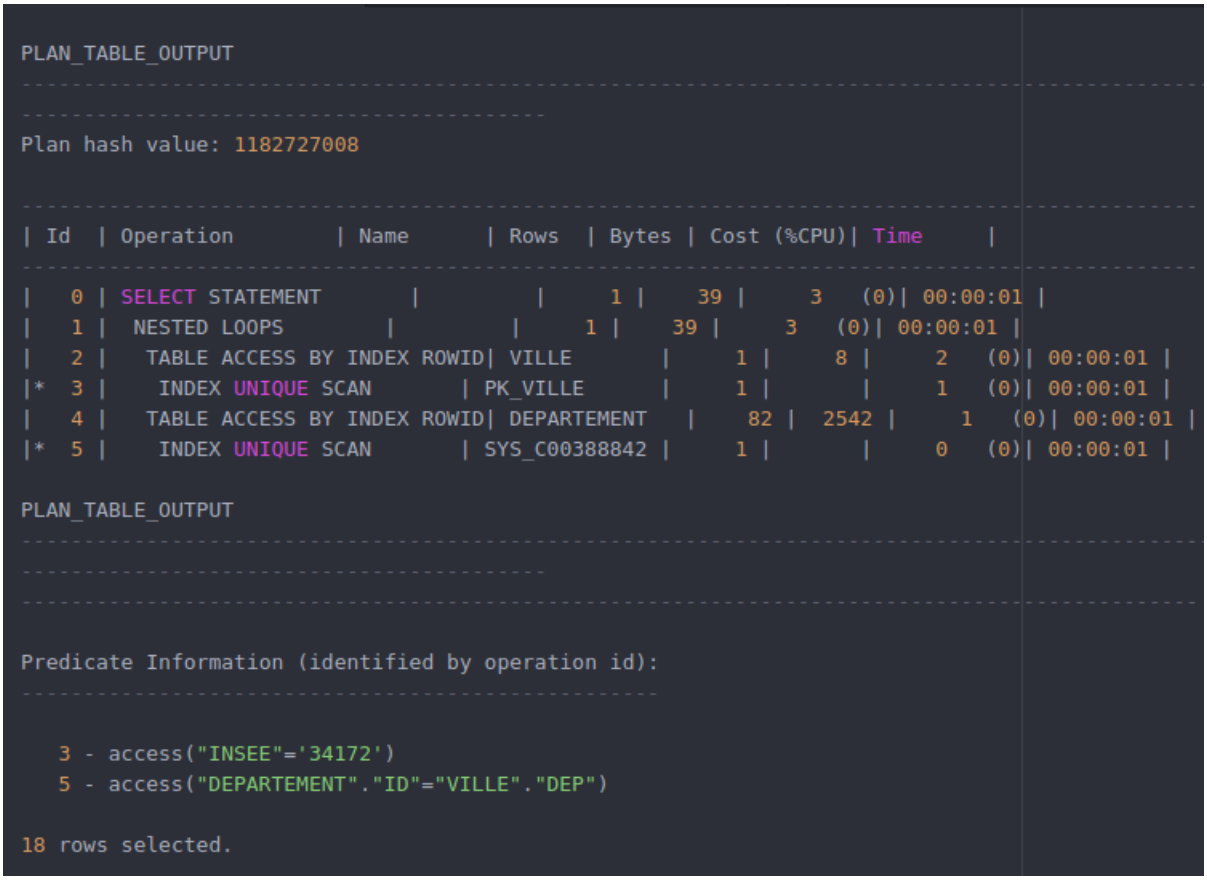
Après l'ajout d'une clé primaire à la table `Ville`, notamment sur l'attribut **insee**, l'algorithme choisi par l'optimiseur sera *Index Unique Scan*. Ce dernier permet de parcourir la table selon l'index placé sur la clé primaire et est ainsi moins gourmand en termes de consommation de ressources.

2 Jointure

2.1 Question 5

```
1   explain plan for select departement.nom from
        departement, ville where insee='34172' and
        departement.id = ville.dep;
2   select plan_table_output from
        table(dbms_xplan.display());
```

Listing 3 – requêtes permettant d'expliquer le plan d'exécution affichant le nom du département pour la ville dont le numéro insee est 34172



```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------
Plan hash value: 1182727008

--------------------------------------------------------------------------------
| Id  | Operation         | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |              |     1 |    39 |     3   (0)| 00:00:01 |
|   1 |  NESTED LOOPS     |              |     1 |    39 |     3   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| VILLE       |     1 |     8 |     2   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN       | PK_VILLE     |     1 |       |     1   (0)| 00:00:01 |
|   4 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT  |    82 |  2542 |     1   (0)| 00:00:01 |
|*  5 |    INDEX UNIQUE SCAN       | SYS_C00388842 |     1 |       |     0   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------
--------------------------------------------------------------------------------

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

   3 - access("INSEE"='34172')
   5 - access("DEPARTEMENT"."ID"="VILLE"."DEP")

18 rows selected.
```

FIGURE 3 – Plan d'exécution

```
Statistics
----------------------------------------------------------
   17  recursive calls
   12  db block gets
   50  consistent gets
    0  physical reads
    0  redo size
       2008 bytes sent via SQL*Net to client
  511  bytes received via SQL*Net from client
    3  SQL*Net roundtrips to/from client
    1  sorts (memory)
    0  sorts (disk)
   18  rows processed
```

FIGURE 4 – Statistiques

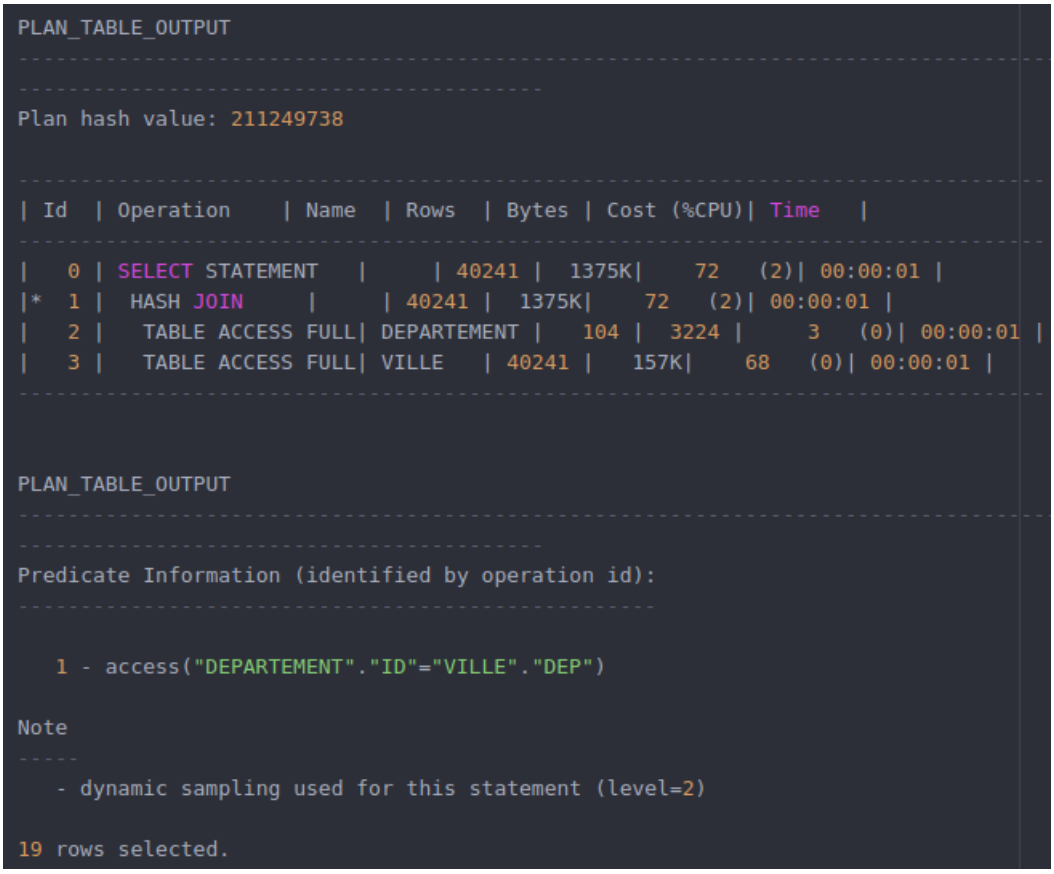
2.2 Question 6

Dans la requête précédente, l'optimiseur a choisi les algorithmes *Index Unique Scan* puis *Nested Loops* lors de la jointure des tables `Ville` et `Departement`, vu que cette dernière se fait sur leurs attributs clés primaires, respectivement **insee** et **id**, après avoir sélectionné les tuples vérifiant une condition de sélection (notamment, la ville ayant le code **insee** 34172).

D'autre part, la requête courante ne spécifie aucune condition de sélection précédant la jointure sur les deux tables. Par conséquent, l'optimiseur a choisi les algorithmes *Table Access Full* pour balayer les tables et *Hash Join* pour faire la jointure.

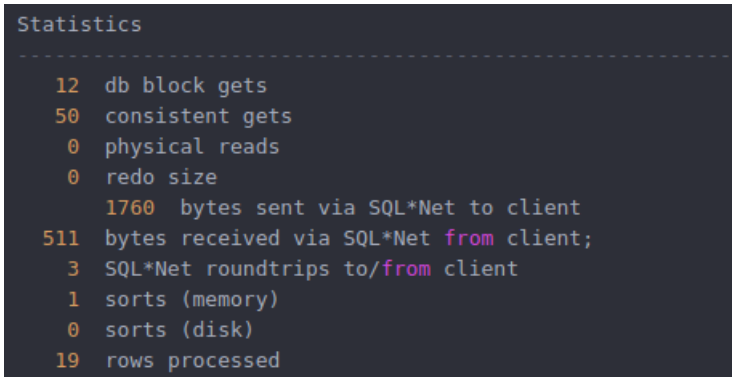
```
1  explain plan for select departement.nom from
       departement, ville where departement.id = ville.dep;
2  select plan_table_output from
       table(dbms_xplan.display());
```

Listing 4 – requêtes permettant d'expliquer le plan d'exécution affichant le nom du département pour toutes les villes



```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------
Plan hash value: 211249738

--------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            | 40241 |  1375K|    72   (2)| 00:00:01 |
|*  1 |  HASH JOIN         |            | 40241 |  1375K|    72   (2)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTEMENT |   104 |  3224 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| VILLE      | 40241 |   157K|    68   (0)| 00:00:01 |
--------------------------------------------------------------------------------


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

   1 - access("DEPARTEMENT"."ID"="VILLE"."DEP")

Note
-----
   - dynamic sampling used for this statement (level=2)

19 rows selected.
```

FIGURE 5 – Plan d'exécution



```
Statistics
----------------------------------------------------------
         12  db block gets
         50  consistent gets
          0  physical reads
          0  redo size
       1760  bytes sent via SQL*Net to client
        511  bytes received via SQL*Net from client;
          3  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
         19  rows processed
```

FIGURE 6 – Statistiques

2.3 Question 7

Pour la requête du listing 5, l'optimiseur a choisi les algorithmes *Index Unique Scan* pour sélectionner le département ayant un **id** (clé primaire de la table `Departement`) 91 et *Table Access Full* pour balayer la table `Ville`, vu que la jointure se fait sur un attribut non clé primaire de celle-ci. Suite à cette opération, l'algorithme *Nested Loops* est utilisé lors de l'opération de jointure sur ces deux tables.

Pour la requête du listing 6, par contre, en absence d'une opération de jointure, l'optimiseur choisit l'algorithme *Index Unique Scan* sur l'attribut clé primaire **id** de la table `Departement` pour la valeur 92.

```
1  explain plan for select ville.nom, departement.nom
      from ville, departement where departement.id='91'
      and ville.dep=departement.id;
2  select plan_table_output from
      table(dbms_xplan.display());
```

Listing 5 – requêtes permettant d'expliquer le plan d'exécution affichant le nom des villes et du département dont le numéro est 91 (id)

```
1  explain plan for select departement.nom from
      departement where departement.id='92';
2  select plan_table_output from
      table(dbms_xplan.display());
```

Listing 6 – requêtes permettant d'expliquer le plan d'exécution affichant le nom du département dont le numéro est 92

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------
Plan hash value: 3197113083

-------------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |   759 | 66033 |    69   (0)| 00:00:01 |
|   1 |  NESTED LOOPS          |          |   759 | 66033 |    69   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT   |     1 |    31 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN       | SYS_C00388842 |     1 |       |     1   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL      | VILLE       |   759 | 42504 |    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------

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

   3 - access("DEPARTEMENT"."ID"='91')
   4 - filter("VILLE"."DEP"='91')

Note
-----
   - dynamic sampling used for this statement (level=2)

21 rows selected.
```

FIGURE 7 – Plan d'exécution

```
Statistics
----------------------------------------------------------
          20  recursive calls
          12  db block gets
          54  consistent gets
           0  physical reads
           0  redo size
        1979  bytes sent via SQL*Net to client
         511  bytes received via SQL*Net from client
           3  SQL*Net roundtrips to/from client
           1  sorts (memory)
           0  sorts (disk)
          21  rows processed
```

FIGURE 8 – Statistiques

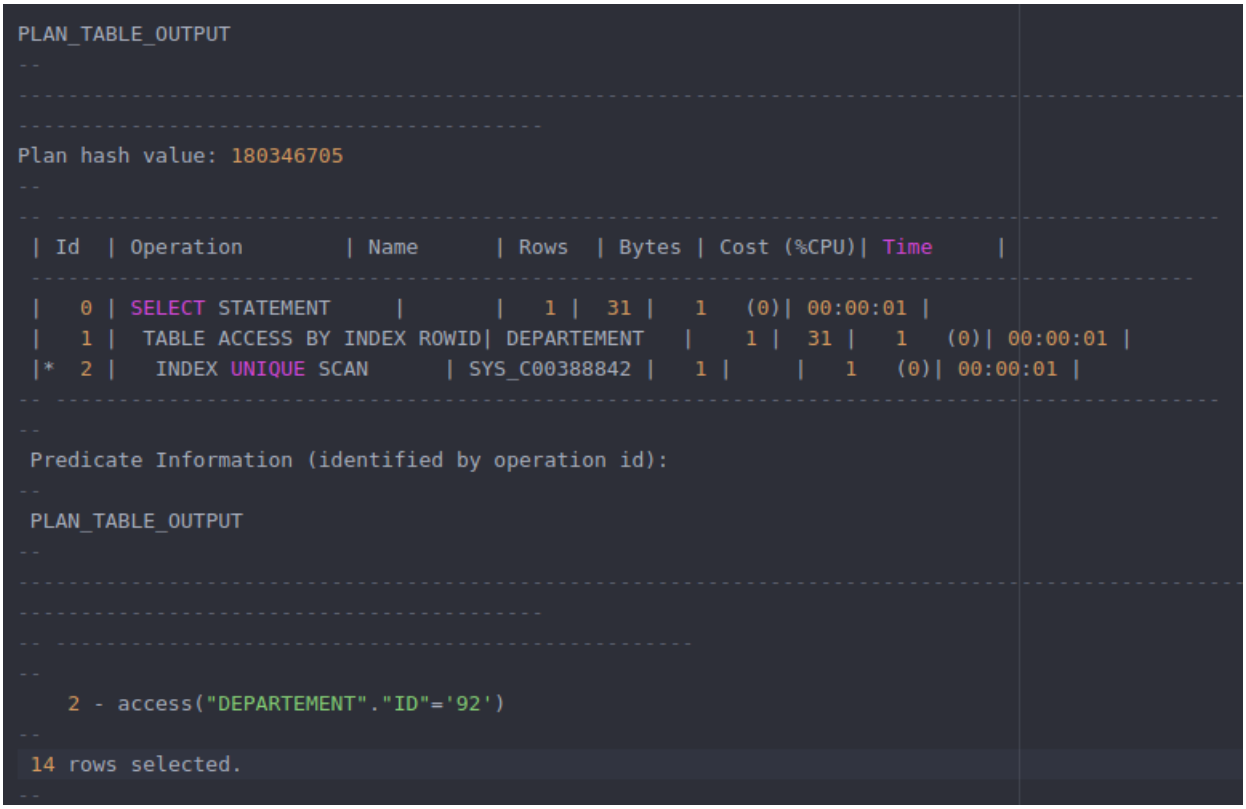


FIGURE 9 – Plan d'exécution

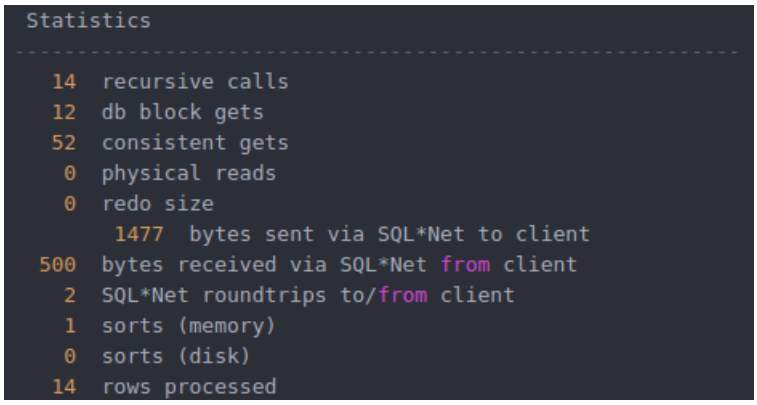


FIGURE 10 – Statistiques

3 Modification du comportement de l'optimiseur

3.1 Question 8

En choisissant explicitement l'algorithme *Nested Loops* pour les requêtes des listings 3 et 5, on obtient les mêmes résultats.

D'autre part, quand on force l'optimiseur à utiliser l'algorithme *Nested Loops* pour la requête du listing 4 (cf. listing 7), le nombre d'appels récursifs [1] augmente considérablement (cf. Figure 12) par rapport à l'algorithme *Hash Join* utilisé par défaut par l'optimiseur pour cette requête.

```
1  explain plan for select /*+ use_nl(departement
       ville)*/ departement.nom from departement, ville
       where departement.id = ville.dep;
2  select plan_table_output from
       table(dbms_xplan.display());
```

Listing 7 – requêtes permettant d'expliquer le plan d'exécution affichant le nom du département pour toutes les villes, en forçant l'utilisation de l'algorithme de jointure Nested Loops

1. *recursive calls*

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------
Plan hash value: 1651012225

-------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            | 40241 |  1375K|  6908   (1)| 00:01:23 |
|   1 |  NESTED LOOPS      |            | 40241 |  1375K|  6908   (1)| 00:01:23 |
|   2 |   TABLE ACCESS FULL| DEPARTEMENT |   104 |  3224 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| VILLE      |   387 |  1548 |    66   (0)| 00:00:01 |
-------------------------------------------------------------------------------


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

   3 - filter("DEPARTEMENT"."ID"="VILLE"."DEP")

Note
-----
   - dynamic sampling used for this statement (level=2)

19 rows selected.
```

FIGURE 11 – Plan d'exécution

```
Statistics
----------------------------------------------------------
         15  recursive calls
         12  db block gets
         54  consistent gets
          0  physical reads
          0  redo size
       1761  bytes sent via SQL*Net to client
        511  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
         19  rows processed
```

FIGURE 12 – Statistiques

4 Utilisation d'index

4.1 Question 9

```
1   create index idx_dep_ville on ville(dep);
```

Listing 8 – ajout d'un index secondaire sur l'attribut dep de la table Ville

L'ajout d'un index secondaire sur l'attribut **dep** de la table `Ville` n'affecte pas les plans d'exécution des requêtes des listings 1 et 3, vu que cet attribut ne figure pas dans ces requêtes.

D'autre part, cet ajout change le plan d'exécution de la requête du listing 4, vu que celle-ci contient bien cet attribut. En effet, l'ajout d'un index sur cet attribut permet à l'optimiseur de choisir l'algorithme *Index Fast Full Scan* (cf. Figure 13) pour balayer la table selon cet index, au lieu d'utiliser *Table Full Scan* choisi précédemment.

En outre, cet ajout change le plan d'exécution de la requête du listing 5, vu que celle-ci contient bien cet attribut. En effet, l'ajout d'un index sur cet attribut permet à l'optimiseur de choisir l'algorithme *Index Range Scan* (cf. Figure 15) pour balayer la table selon cet index, au lieu d'utiliser *Table Full Scan* choisi précédemment.

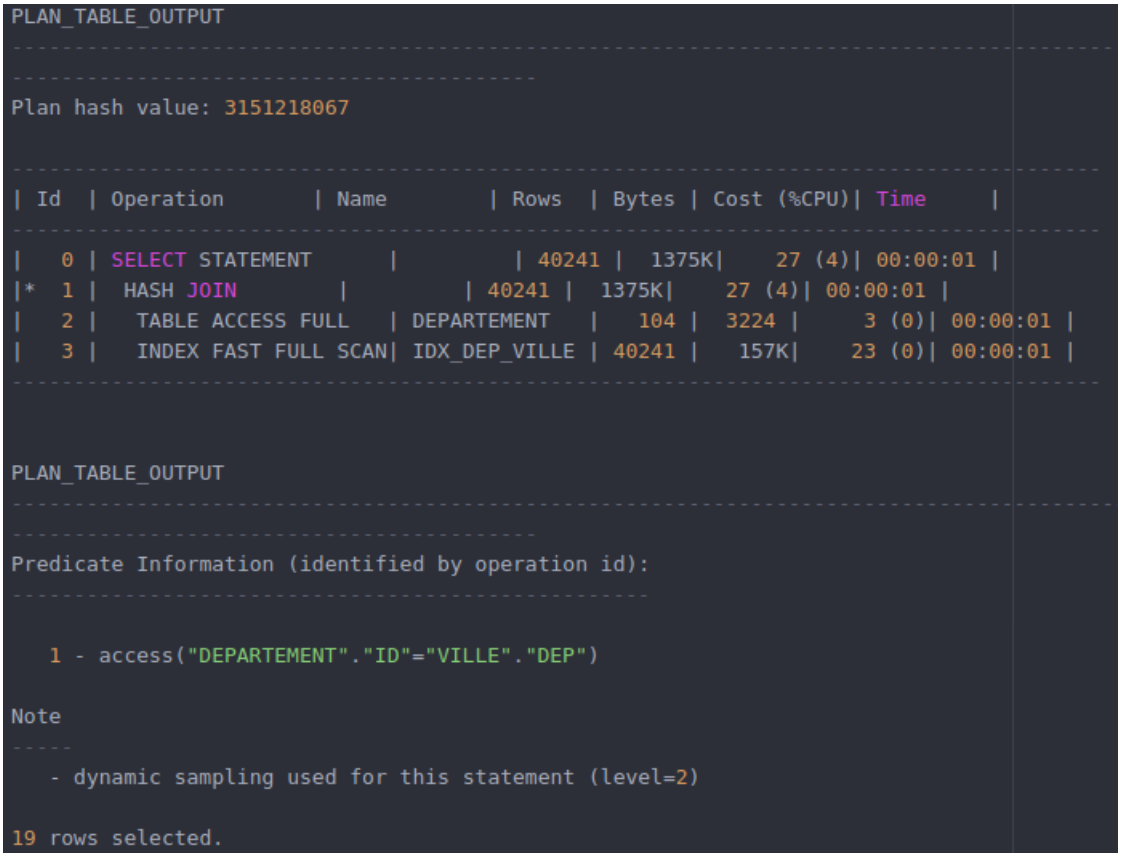


FIGURE 13 – Plan d'exécution

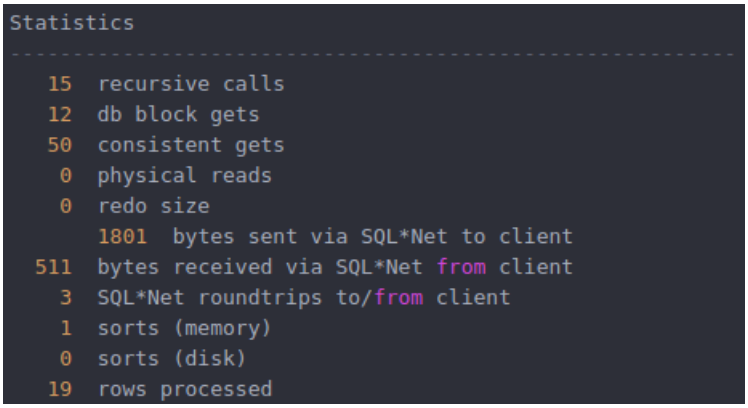


FIGURE 14 – Statistiques

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------
Plan hash value: 2966802174

--------------------------------------------------------------------------------
| Id  | Operation         | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |               |   196 | 16660 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS     |               |   196 | 16660 |     4   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT    |     1 |    31 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN       | SYS_C00388842 |     1 |       |     1   (0)| 00:00:01 |
|   4 |   TABLE ACCESS BY INDEX ROWID| VILLE         |   196 | 10584 |     3   (0)| 00:00:01 |
|*  5 |    INDEX RANGE SCAN        | IDX_DEP_VILLE |   196 |       |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------
--------------------------------------------------------------------------------

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

   3 - access("DEPARTEMENT"."ID"='91')
   5 - access("VILLE"."DEP"='91')

Note
-----
   - dynamic sampling used for this statement (level=2)

22 rows selected.
```

FIGURE 15 – Plan d'exécution

```
Statistics
----------------------------------------------------------
         17  recursive calls
         12  db block gets
         50  consistent gets
          0  physical reads
          0  redo size
       2075  bytes sent via SQL*Net to client
        511  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
         22  rows processed
```

FIGURE 16 – Statistiques

4.2 Question 10

```
1  explain plan for select ville.nom, departement.nom,
      region.nom from ville, departement, region where
      ville.dep = departement.id and departement.reg =
      region.id;
2  select plan_table_output from
      table(dbms_xplan.display());
```

Listing 9 – requêtes permettant d'expliquer le plan d'exécution affichant le nom des villes, de leurs départements et de leurs régions

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------
Plan hash value: 424771235

--------------------------------------------------------------------------------
| Id  | Operation     | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |     | 40241 |  5501K|    75   (2)| 00:00:01 |
|*  1 |  HASH JOIN      |     | 40241 |  5501K|    75   (2)| 00:00:01 |
|*  2 |   HASH JOIN     |     |   104 |  8736 | 7  (15)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| REGION    |    27 |  1080 | 3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| DEPARTEMENT |   104 |  4576 | 3   (0)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | VILLE   | 40241 |  2200K|    68   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------
--------------------------------------------------------------------------------

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

   1 - access("VILLE"."DEP"="DEPARTEMENT"."ID")
   2 - access("DEPARTEMENT"."REG"="REGION"."ID")

Note
-----
   - dynamic sampling used for this statement (level=2)

22 rows selected.
```

FIGURE 17 – Plan d'exécution

```
Statistics
----------------------------------------------------------
  17  recursive calls
  12  db block gets
  54  consistent gets
   0  physical reads
   0  redo size
        1988  bytes sent via SQL*Net to client
 511  bytes received via SQL*Net from client
   3  SQL*Net roundtrips to/from client
   1  sorts (memory)
   0  sorts (disk)
  22  rows processed
```

FIGURE 18 – Statistiques

4.3 Question 11

```
1   create index idx_reg_departement on departement(reg);
```

Listing 10 – ajout d'un index secondaire sur l'attribut reg de la table Departement

La ré-exécution de la requête du listing 9, suite à l'ajout de l'index secondaire sur l'attribut **reg** de la table `Departement`, fournit les résultats affichés dans les figures 19 et 20.

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
----------------------------
Plan hash value: 3309180011

------------------------------------------------------------------------------------------------
| Id  | Operation          | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                    | 40241 |  5501K|  74   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |                    | 40241 |  5501K|  74   (0)| 00:00:01 |
|   2 |   NESTED LOOPS     |                    |       |       |          |          |
|   3 |    NESTED LOOPS    |                    |   104 |  8736 |   6   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL       | REGION        |  27 |  1080 |   3   (0)| 00:00:01 |
|*  5 |     INDEX RANGE SCAN        | IDX_REG_DEPARTEMENT |   4 |     |   0   (0)| 00:00:01 |
|   6 |    TABLE ACCESS BY INDEX ROWID| DEPARTEMENT     |   4 | 176 |   1   (0)| 00:00:01 |
|   7 |   TABLE ACCESS FULL        | VILLE         | 40241 |  2200K|  68   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------

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

   1 - access("VILLE"."DEP"="DEPARTEMENT"."ID")
   5 - access("DEPARTEMENT"."REG"="REGION"."ID")

Note

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
----------------------------
-----
   - dynamic sampling used for this statement (level=2)

24 rows selected.
```

FIGURE 19 – Plan d'exécution

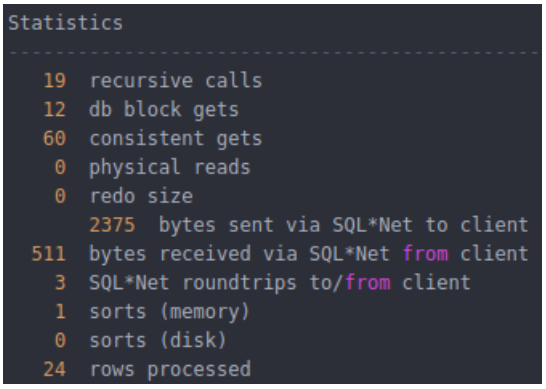
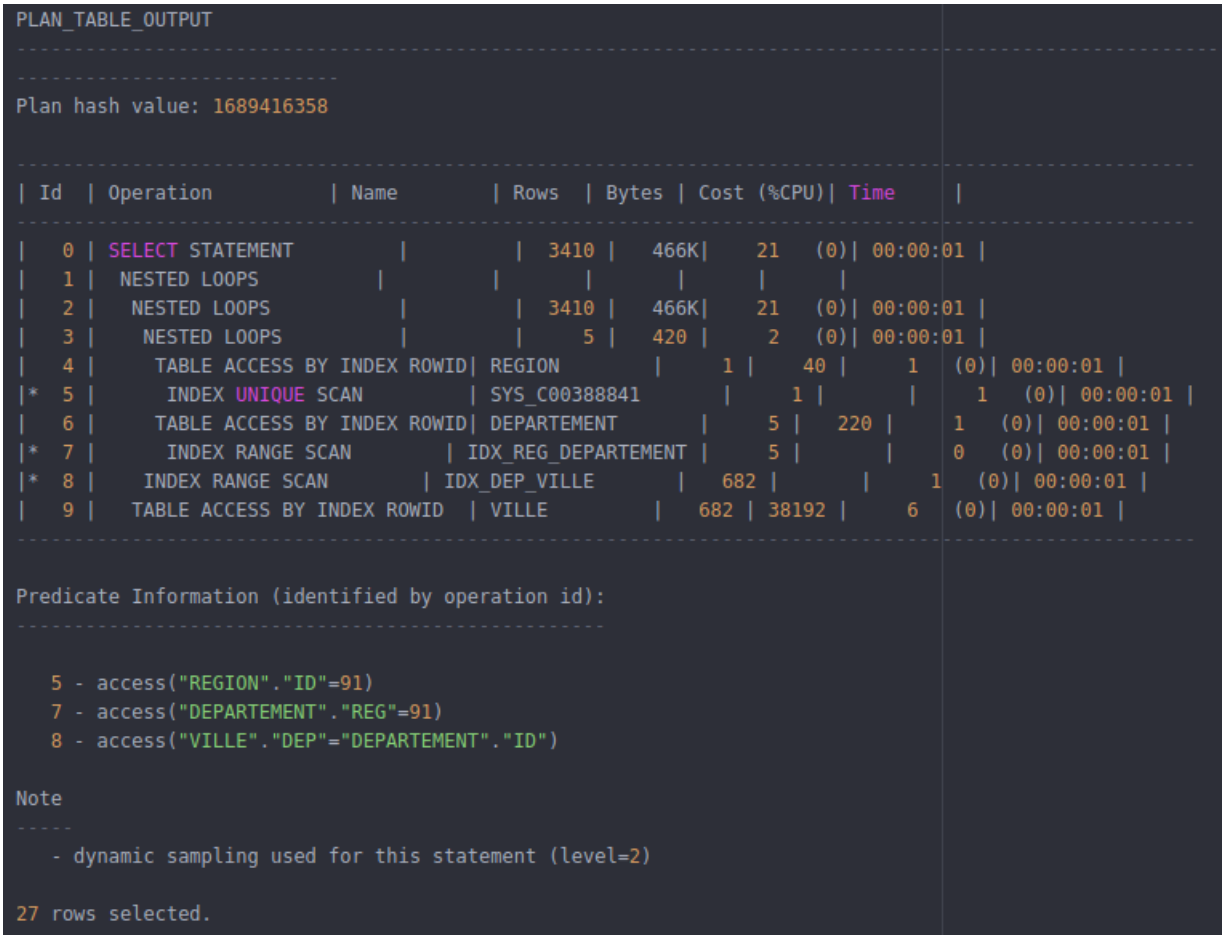


FIGURE 20 – Statistiques

4.4 Question 12

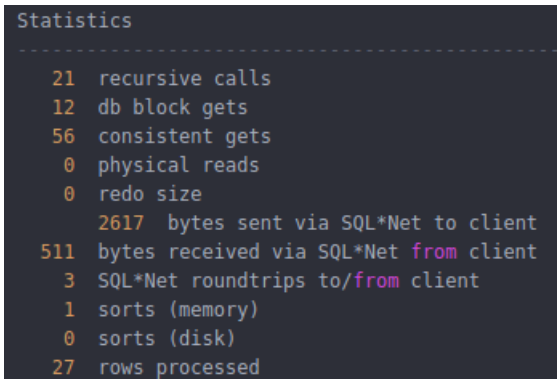
```
1  explain plan for select ville.nom, departement.nom,
      region.nom from ville, departement, region where
      region.id=91 and ville.dep=departement.id and
      departement.reg=region.id;
2  select plan_table_output from
      table(dbms_xplan.display());
```

Listing 11 – requêtes permettant d'expliquer le plan d'exécution affichant le nom des villes, de leurs départements et de la région pour la région dont le numéro (id) est 91



```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------

------------------------------
Plan hash value: 1689416358

-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                     |  3410 |   466K|    21   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                     |       |       |            |          |
|   2 |   NESTED LOOPS               |                     |  3410 |   466K|    21   (0)| 00:00:01 |
|   3 |    NESTED LOOPS              |                     |     5 |   420 |     2   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| REGION            |     1 |    40 |     1   (0)| 00:00:01 |
|*  5 |      INDEX UNIQUE SCAN       | SYS_C00388841       |     1 |       |     1   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID| DEPARTEMENT       |     5 |   220 |     1   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN        | IDX_REG_DEPARTEMENT |     5 |       |     0   (0)| 00:00:01 |
|*  8 |    INDEX RANGE SCAN          | IDX_DEP_VILLE       |   682 |       |     1   (0)| 00:00:01 |
|   9 |   TABLE ACCESS BY INDEX ROWID  | VILLE             |   682 | 38192 |     6   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------

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

   5 - access("REGION"."ID"=91)
   7 - access("DEPARTEMENT"."REG"=91)
   8 - access("VILLE"."DEP"="DEPARTEMENT"."ID")

Note
-----
   - dynamic sampling used for this statement (level=2)

27 rows selected.
```

FIGURE 21 – Plan d'exécution



```
Statistics
----------------------------------------------------------
   21  recursive calls
   12  db block gets
   56  consistent gets
    0  physical reads
    0  redo size
       2617  bytes sent via SQL*Net to client
  511  bytes received via SQL*Net from client
    3  SQL*Net roundtrips to/from client
    1  sorts (memory)
    0  sorts (disk)
   27  rows processed
```

FIGURE 22 – Statistiques

4.5 Question 13

L'ajout d'un index secondaire sur l'attribut **dep** de la table `Ville` permet à l'optimiseur de choisir l'algorithme *Index Range Scan* pour balayer cette table selon cet index, et permet par conséquent d'optimiser les accès aux tuples et moins de consommation en termes de temps d'exécution et de mémoire.

```
1  explain plan for select ville.nom from ville where
       ville.dep LIKE '7%';
2  select plan_table_output from
       table(dbms_xplan.display());
```

Listing 12 – requêtes permettant d'expliquer le plan d'exécution affichant le nom des villes dont le numéro de département (dep) commence par 7

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------
Plan hash value: 428725444

--------------------------------------------------------------------------------
| Id  | Operation         | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |  4183 | 220K|  35   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| VILLE     |  4183 | 220K|  35   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN  | IDX_DEP_VILLE |  4183 |     |  10   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------
--------------------------------------------------

   2 - access("VILLE"."DEP" LIKE '7%')
       filter("VILLE"."DEP" LIKE '7%')

Note
-----
   - dynamic sampling used for this statement (level=2)

19 rows selected.
```

FIGURE 23 – Plan d'exécution

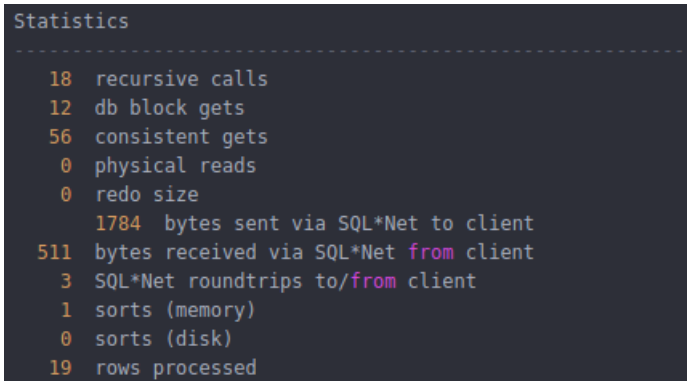


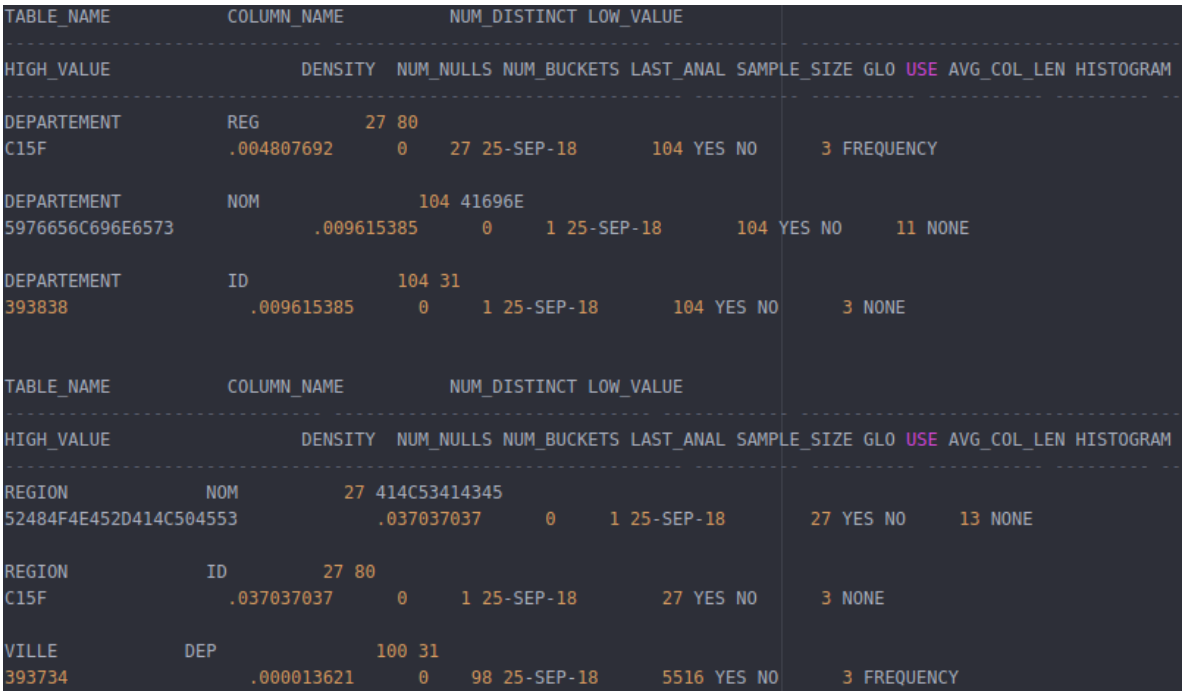
FIGURE 24 – Statistiques

5 Les statistiques des tables

5.1 Question 14

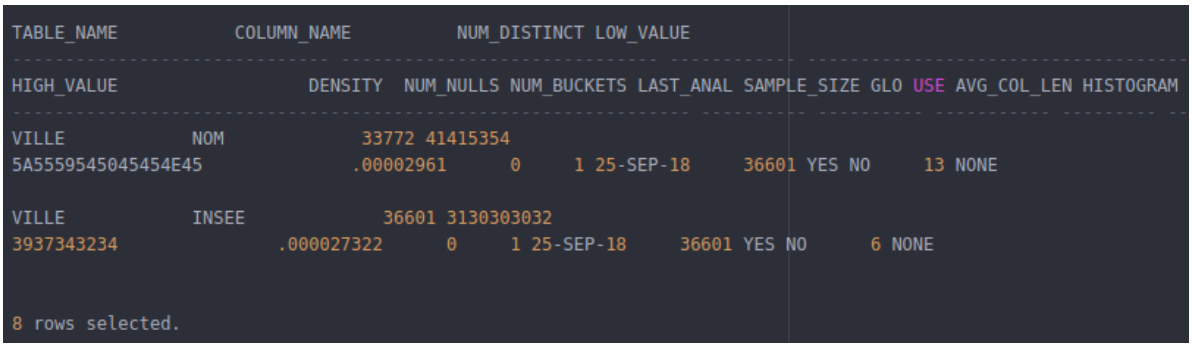
```
1  exec dbms_stats.gather_table_stats('brima','ville');
      --adds the cost details for table 'ville'
2  exec
      dbms_stats.gather_table_stats('brima','departement');
      --adds the cost details for table 'departement'
3  exec dbms_stats.gather_table_stats('brima','region');
      --adds the cost details for table 'region'
4  SELECT * FROM USER_TAB_COL_STATISTICS; --visualize the
      obtained tables describing these costs
```

Listing 13 – requêtes permettant de recalculer les statistiques sur les tables Ville, Departement et Region afin de visualiser les coûts des différents plans d'exécution qui y sont associés



```
TABLE_NAME           COLUMN_NAME          NUM_DISTINCT LOW_VALUE
-------------------- -------------------- ------------ ----------------------------------------
HIGH_VALUE                 DENSITY  NUM_NULLS NUM_BUCKETS LAST_ANAL SAMPLE_SIZE GLO USE AVG_COL_LEN HISTOGRAM
-------------------------------------------------- ---------- ---------- ---------- ---------- --

DEPARTEMENT          REG                27 80
C15F                 .004807692          0         27 25-SEP-18        104 YES NO       3 FREQUENCY

DEPARTEMENT          NOM                104 41696E
5976656C696E6573                .009615385          0          1 25-SEP-18        104 YES NO      11 NONE

DEPARTEMENT          ID                 104 31
393838                 .009615385          0          1 25-SEP-18        104 YES NO       3 NONE


TABLE_NAME           COLUMN_NAME          NUM_DISTINCT LOW_VALUE
-------------------- -------------------- ------------ ----------------------------------------
HIGH_VALUE                 DENSITY  NUM_NULLS NUM_BUCKETS LAST_ANAL SAMPLE_SIZE GLO USE AVG_COL_LEN HISTOGRAM
-------------------------------------------------- ---------- ---------- ---------- ---------- --

REGION               NOM              27 414C53414345
52484F4E452D414C504553                .037037037          0          1 25-SEP-18         27 YES NO      13 NONE

REGION               ID               27 80
C15F                   .037037037          0          1 25-SEP-18         27 YES NO       3 NONE

VILLE                DEP                100 31
393734                  .000013621          0         98 25-SEP-18       5516 YES NO       3 FREQUENCY
```

FIGURE 25 – Coûts des différents plans d'exécution des tables Departement et Region



```
TABLE_NAME           COLUMN_NAME          NUM_DISTINCT LOW_VALUE
-------------------- -------------------- ------------ ----------------------------------------
HIGH_VALUE                 DENSITY  NUM_NULLS NUM_BUCKETS LAST_ANAL SAMPLE_SIZE GLO USE AVG_COL_LEN HISTOGRAM
-------------------------------------------------- ---------- ---------- ---------- ---------- --

VILLE                NOM              33772 41415354
5A5559545045454E45                .00002961          0          1 25-SEP-18      36601 YES NO      13 NONE

VILLE                INSEE            36601 3130303032
3937343234                .000027322          0          1 25-SEP-18      36601 YES NO       6 NONE


8 rows selected.
```

FIGURE 26 – Coûts des différents plans d'exécution de la table Ville