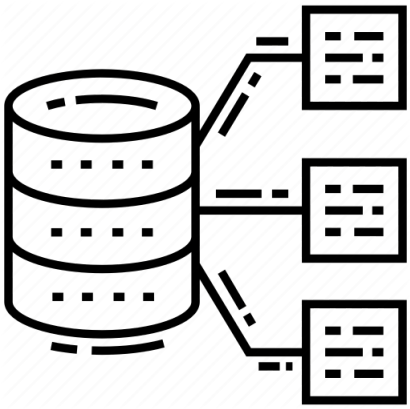
DEPARTEMENT INFORMATIQUE
DE LA FACULTE DES SCIENCES

Quentin Yeche (21520370), Yanis Allouch (21708237)

Rapport du TP N°2 : Optimisation de requête



HMIN122M — Entrepôts de données et Big-Data

Référent: Federico Ulliana et Anne-Muriel Chifolleau

2020

Table des matières

Introduction

L'objectif est de comprendre le fonctionnement d'optimisation d'un SGBD, au travers d'une étude attentive de la production et de l'estimation du coût de plans d'exécution logiques d'un SGBD théorique d'abords puis pratique avec ORACLE.

L'environnement de travail est composé de Draw.io pour la schématisation, ORACLE sur l'instance *pmaster* de la faculté des sciences pour le SGDB pour les tests.

Première Partie

1 Exercice

1.1 Coût de plans d'exécution logiques

Soit le modèle relationnel composé des relations suivantes :

ETUDIANTS(IDE, NOM, AGE) – la relation contenant tous les étudiants
MODULES(IDM, RESPONSABLE, INTITULE) – la relation contenant tous les modules
IP(#IDE, #IDM) – la relation contenant la liste des inscripions pédagogiques (inscription d'un étudiants à un module)
FORMATION(IDF, NOMF) – la relation contenant toutes les formations
IA (#IDE, #IDF) – la relation contenant la liste des inscriptions administratives (inscription d'un étudiants à une formation)

Question 1

Que permet d'obtenir la requête ?
— La liste des noms étudiants régulièrement inscrits au module ayant pour intitulé "EDBD" ;

Question 2

Pour chaque plan d'exécution logique, calculer le nombre de lignes intermédiaires créées.

Sachant les informations suivantes sur les tables : 200 étudiants, 70 modules, 4200 IP (inscriptions pédagogiques), 50 formations, 70 IA (des étudiants peuvent être inscrits à plusieurs formations), ainsi qu'un facteur de sélectivité de **30%**.

On notera f_s le facteur de sélectivité. Le nombre total de ligne intermédiaires générées pour chaque arbre (de gauche à droite) est le suivant :
— Premier arbre
1. Jointure IP et Modules $\rightarrow$ 4200 lignes
2. Jointure avec Etudiants $\rightarrow$ 4200 lignes
3. Selection $\rightarrow f_s * 4200$ lignes
4. Projection $\rightarrow f_s * 4200$ lignes

Nous avons donc un total de $2 * 4200 + 2 * f_s * 4200 = 10920$ lignes générées
— Deuxième arbre
1. Selection sur Modules $\rightarrow$ 1 ligne (avec l'hypothèse que les modules ont tous des noms uniques)
2. Jointure avec IP $\rightarrow f_s * 4200$ lignes
3. Jointure avec Etudiants $\rightarrow f_s * 4200$ lignes
4. Projection $\rightarrow f_s * 4200$ lignes

Nous avons donc un total de $3 * 4200 + 1 = 3781$ lignes générées
— Troisième arbre
Cette stratégie d'execution est très similaire à la stratégie 2. Le nombre de lignes générées est identique, 3781. La différence est que cette stratégie effectue les projections le plus tôt possible, ce qui réduit la taille de chaque ligne dans la suite de la stratégie.

— Quatrième arbre

1. Jointure Modules et Etudiants. Ces deux relations ne sont pas reliés par des clés, cette jointure est donc un simple produit cartésien $\rightarrow 70 * 200$ lignes
2. Jointure avec IP $\rightarrow$ 4200 lignes
3. Sélection $\rightarrow f_s * 4200$ lignes
4. Projection $\rightarrow f_s * 4200$ lignes

Nous avons donc un total de $70 * 200 + 4200 + 2 * f_s * 4200 = 20720$ lignes générées.

Question 3

Quel est le plan d'exécution logique optimal ? Pourquoi ?

— Le plan le plus optimisé est le plan n°3, parce qu'il restreint le nombre de modules et, par la suite il présente des jointures qui ne concernent que les colonnes nécessaires au calcul de la liste des étudiants, c'est à dire que le nombre de lignes et de colonnes sont minimum.

1.2 Définition de plans d'exécution logiques

Les plans d'exécution sont placés en annexe, parce qu'ils prennent une place conséquente dans la mise en page.
De plus, dans un souci de clarté et pour simplifier les arbres, nous avons souvent choisi d'omettre les projections intermédiaires qui pourraient être faites afin de réduire la taille des lignes à chaque étape.

1. Permet d'obtenir les noms des responsables des modules dont l'intitulé commence par "HMIN", et dans lesquels un étudiant nommé "Dupond" est inscrit.

```
SELECT RESPONSABLE
FROM ETUDIANTS E ,MODULES M ,IP I
WHERE E.IDE = I.IDE AND M.IDM=I.IDM
AND NOM = "DUPOND" AND INTULE LIKE "HMIN";
```

   Les stratégies sont ici classées de la plus coûteuse à la plus optimale. Cependant la stratégie 1(c) est peu susceptible (ou pas du tout) d'être utilisée par un SGBD puisque leur approche heuristique tend plutôt à éviter les approches parallélisées. Dans ce cas la stratégie 1(b) est probablement celle qui sera choisie à la place.

2. Permet d'obtenir les noms des étudiants de la formation "Master Aigle" qui ont une inscription administrative, et qui sont inscrits au module "EDBD".

```
SELECT NOM
FROM ETUDIANTS E ,FORMATION F ,IA A, IP I, MODULE M
WHERE E.IDE = A.IDE AND F.IDF=A.IDF AND I.IDE=E.IDE AND M.IDM=I.IDM,
AND NOMF = "MASTER AIGLE" AND INTULE = "EDBD" ;
```

   Le plan 2(a) semble le plus optimal. Cependant il est d'une part parallélisé, et d'autre part il effectue une jointure entre les tables IA et IP sur deux clés étrangères (qui pointent toutes les deux vers la clé primaire de ETUDIANTS), ce qui n'est pas un comportement intuitif. Le plan 2(c), est plutôt efficace, non parallélisé et effectue les sélections le plus tôt possible.

3. Affiche les intitulés des modules auxquels sont inscrits les étudiants les plus jeunes.

```
SELECT INTILULE
FROM ETUDIANTS E ,MODULES M ,IP I
WHERE E.IDE = I.IDE AND M.IDM=I.IDM
AND AGE = (select min (AGE) from ETUDIANTS) ;
```

Dans ce cas-là, l'approche parallélisée est pour une fois moins efficace que le traitement en série 3(b). L'approche en série 3(b) force à effectuer la fonction d'agrégation (min) au début de l'exécution, ce qui permet ensuite de faire les sélections très tôt et ainsi réduire considérablement le nombre de lignes intermédiaires.

1.3 Réécriture de plans d'exécution logiques

1. Les deux expressions retournent-elles le même résultat (sont-elles équivalentes) ? Justifiez votre réponse en indiquant les règles de réécriture que l'on peut appliquer.
   — Oui, elles sont équivalentes. On peut descendre ou faire monter les conditions sur les sélections et on retrouve les mêmes expressions.
2. Une expression vous semble-t-elle meilleure que l'autre si on les considère comme des plans d'exécution ?
   — La seconde expression est meilleure étant donnée que la jointure se fait sur des tables réduites.

1.4 Tous les plans d'exécution logiques

Soit le modèle relationnel suivant :

ACTEUR (idA, nom, prenom, nationalite) – la relation contenant tous les acteurs
FILM (idF, titre, annee, nbspectateurs, #idRealisateurs, #idGenre) – la relation contenant tous les films
JOUER (#idActeur,#idFilm, salaire) – la relation contenant la listes des acteurs et des films dans lesquels ils jouent
REALISATEUR (idR, nom, prenom, nationalite) – la relation contenant tous les réalisateurs
GENRE (idG, description) – la relation contenant tous les genres des films (horreur, comédie, ...)

```
SELECT acteur.nom,acteur.prenom
FROM acteur,jouer,film,genre, realisateur
WHERE (idA=idActeur) AND (idFilm=idF) AND (idGenre=idG) AND (idRealisateur=idR)
AND (nationalite='France') AND (description='comedie') AND (realisateur.nom = 'Les
     freres Coen') ;
```

1. Pour la requête ci-dessus, donner quelques plans d'exécution logique.
   — Voir de l'arbre 4(a) à l'arbre 4 (d) en annexe.
2. Parmi les plan d'exécution , quel est le plan optimal ? Justifier votre choix.
   — Les plans (c) et (d) sont les plus optimisés que nous ayons trouvé. Ils sont très proche tous les deux. Le (c) est meilleur puisque l'on fait descendre la sélection sur le genre plus bas, mais nous forçons donc une stratégie parallélisée, ce qui est généralement à éviter. Dans la suite on regardera donc plutôt la stratégie (d). Cette stratégie nous permet de bien majorer le nombre de lignes intermédiaires :
     i La sélection sur le réalisateur dès la première étape nous permet de nous restreindre sur leur oeuvre. Il est assez logique de penser que le nombre de

films qu'ils ont réalisé est très petit par rapport au nombre total de films, de même pour les acteurs qui ont joué dans leurs films par rapport au nombre total d'acteurs.

ii La sélection description="Comédie" permet de restreindre encore plus le domaine, puisque nous sommes maintenant restreints aux films des frères Cohen qui sont des comédies.

iii La sélection nationalité="France" peut sembler tardive, mais elle ne l'est pas, même pour le cas parallélisé (c). En effet le nombre d'acteurs qui ont joué dans les comédies des frères Cohen est probablement beaucoup plus petit que le nombre d'acteurs français. Si on considère que la jointure se fait sur un index (hypothèse assez faible), faire la sélection de la nationalité plus tard est plus avantageux.

Deuxième partie

2 Exercice

2.1 Les plans d'exécution sous ORACLE

Les plans d'exécutions sont disponibles en pièce jointe de ce rapport.

Sélection

1. Examinez les scripts pour comprendre ce qu'il font.
   — Il y a 2 scripts SQL, l'un nommé *script_table.sql*, chargé de créer les trois tables suivantes : ville, region, et departement.
   Le second, nommé *script_remplissage.sql*, insère des tuples dans les tables précédemment créées.
2. Explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues lors de l'exécution d'une requête permettant d'afficher le nom des villes dont le numéro insee est 34172.

```
1 SELECT nom
2 FROM ville
3 WHERE insee='34172';
```

   La sortie de console est disponible en Figure 13. Pour une seule table, et sans index sur la colonne *insee*, la seule méthode d'accès logique est TABLE ACCESS FULL. Cela demande un accès disque important (12 blocs). Le nombre d'appels récursif est lui aussi conséquent.
3. Ajoutez une clé primaire sur la table ville (utiliser l'attribut insee).

```
1 ALTER TABLE ville
2 ADD CONSTRAINT PK_ville PRIMARY KEY (insee);
```

4. Explicitez à nouveau le plan d'exécution choisi par l'optimiseur et les statistiques obtenues lors de l'exécution de la requete précedante (afficher le nom des villes dont le numéro insee est 34172). Quelles sont les différences observées par rapport à la question 2 ?

```
1 select nom from ville where insee='34172';
```

   Une coquille s'est introduite dans les scripts de création des tables. En effet, la clé primaire n'est pas un entier mais une chaîne de caractères. Cette question perd légèrement de son sens étant donnée que l'optimisation de recherche utilisant l'index serait plus adaptée pour un type INT. L'indexation sur VARCHAR est néanmoins possible.
   Voir Figure 14.
   On est toujours en TABLE ACCESS FULL, puisque la colonne filtrée *insee* n'est toujours pas indexée. On voit néanmoins le *db block gets* qui tombe à 0, et les *consistent get* qui chutent également, probablement dû à un système de mise en cache des résultats précédents.

Jointure

5. Explicitez maintenant le plan d'exécution choisi par l'optimiseur et les statistiques obtenues lors de l'exécution d'une requête permettant d'afficher le nom du département pour la ville dont le numéro insee est 34172.

```
SELECT d.nom FROM ville v, departement d WHERE v.dep = d.id AND insee='34172';
```

Voir Figure 15.
Les NESTED LOOPS sont ce à quoi on peut s'attendre dans le cas général. Cependant on constate que, bien que nous n'effectuons qu'une seule jointure nous avons deux NESTED LOOPS, donc deux jointures. La première jointure (la plus profonde dans l'arborescence) est une jointure entre la table VILLE et SYS_C00164756, qui est le nom de l'index relatif à la clé primaire de la table DEPARTEMENT. Le SGBD effectue donc une première jointure entre VILLE et l'index de DEPARTEMENT, la jointure externe en TABLE ACCESS BY INDEX ROWID est donc la jointure entre la première jointure et le contenu de la table DEPARTEMENT.
En terme de statistiques, le nombre de blocs est important mais l'essentiel du coût concerne le TABLE ACCESS FULL sur la table VILLE. L'accès à DEPARTEMENT est presque nul puisqu'il ne concerne qu'une seule ligne.

6. Faites de même avec la requête permettant d'afficher le nom des départements de toutes les villes. Quelles sont les différences observées par rapport à la question 5 ?

```
SELECT d.nom FROM ville v, departement d WHERE v.dep = d.id;
```

Voir Figure 16.
Le SGBD choisit ici un HASH JOIN. C'est une méthode qui est souvent choisie lorsque les tailles des tables sont très disparates, ce qui est le cas ici. Dans ce cas la table DEPARTEMENT est bien évidemment celle qui sert à créer la table de hachage. Le nombre de *consistent gets* reste conséquent, vu la taille des tables.

7. Exécutez la requête suivante et explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues : afficher le nom des villes et du département dont le numéro est 91 (id). Utilisez 91 ou '91' et comparez les résultats.

```
select v.nom, d.nom from ville v join departement d on v.dep = d.id
WHERE id='91';
```

Voir Figure 17.
Cette question est l'une des victimes de l'erreur de typage pour la colonne *insee*. En effet la requête avec 91 renvoie une erreur. Un test sur une colonne de type NUMBER nous a montré que le SGBD effectue la conversion VARCHAR → NUMBER sans renvoyer d'erreur et sans rien changer au plan d'exécution. Concernant, ce dernier, le SGBD accède par index (valeur '91') à la table DEPARTEMENT puis effectue la jointure avec VILLE filtrée par dep='91'. Les sélections sont donc faites avant la jointure lorsqu'elles concernent la colonne sur laquelle la jointure s'effectue.

Modification du comportement de l'optimiseur

8. Essayez maintenant les mêmes requêtes mais en forçant l'utilisation de boucles imbriquées (nested loops par la directive /*+ use_nl(table1 table2)*/) et explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues.

```
SELECT /*+ use_nl(ville departement) */  d.nom FROM ville v, departement d
    WHERE v.dep = d.id;
```

Requête Q6 On passe ici d'un HASH JOIN à un NESTED LOOPS. Le nombre de *consistent gets* est multiplié par 10, on comprend donc pourquoi le HASH JOIN était préférable. Voir Figure 18.

Requête Q7 Aucune différence avec la requête précédente puisque NESTED LOOPS était déjà le comportement adopté par le SGBD. L'usage d'autres hints, telles que use_hash, n'a pas non plus eu d'effet. Il semblerait donc qu'il s'agisse bien seulement de conseils, et non pas d'instructions fortes, et que le SGBD se réserve donc le choix d'ignorer les mauvais conseils.
Voir Figure 19.

Utilisation d'index

9. Créer un index secondaire sur l'attribut dep de la table ville : create index idx_dep_ville on ville (dep). Ré-exécutez les requêtes précédentes et explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues.

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

Requête Q5 La stratégie ne change pas. La jointure entre VILLE et l'index de DEPARTEMENT est optimal, et l'ajout de l'index sur dep ne semble pas changer la donne.
Voir Figure 20.

Requête Q6 Le HASH JOIN est encore présent. Cependant le TABLE ACCES FULL sur VILLE devient un INDEX FAST FULL SCAN. Le nombre de *consistent gets* est similaire mais le coût en %CPU est bien plus faible. Le nombre d'appels récursifs est aussi bien plus faible. Une des raisons est qu'un bloc peut contenir beaucoup plus de valeurs d'index que de lignes entières.
Voir Figure 21.

Requête Q7 Le nombre de *consistent gets* est plus faible. Le nouvel index est utilisé au lieu d'un TABLE ACCES FULL. On note également un TABLE ACCES BY INDEX ROWID BATCHED. Le SGBD récupère un batch de paires (index,bloc) et les ordonne par bloc afin de regrouper les accès aux lignes situées dans un même bloc.
Voir Figure 22.

10. Exécutez la requête suivante et explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues : afficher le nom des villes, de leurs départements et de leurs régions.

```
1 SELECT v.nom, d.nom, r.nom
2 FROM ville v, region r, departement d
3 WHERE v.dep=d.id AND r.id=d.reg;
```

Voir Figure 23.
Le SGBD effectue deux HASH JOIN pour les deux jointures demandées. Ici encore les tailles des tables jointes sont très différentes, ce qui motive ce choix. Au niveau des statistiques nous voyons un nombre de blocs utilisés assez important.

11. Créer un index secondaire sur l'attribut reg de la table departement. Ré-exécutez la requête précédente et explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues.

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

Voir Figure 24.
Encore une fois le hash join est intéressant puisque la table ville est bien plus grande que la jointure de DEPARTEMENT et REGION. Le MERGE JOIN est un peu plus mystérieux. La documentation d'Oracle nous donne un indice possible : Sort Merge Join est préférable lorsque la mémoire n'est pas suffisante pour contenir la taille d'une table de hash qui devrait alors nécessiter des écritures sur disque. Etant donnée la taille de VILLE, cela semble bien possible. Un autre détail est que le SORT JOIN (tri) n'est effectué que sur la deuxième table. Là encore la documentation Oracle nous dit : "If an index exists, then the database can avoid sorting the first data set. However, the database always sorts the second data set, regardless of indexes." Les raisons pour lesquelles un tri peut être évité ne sont malheureusement pas expliquées...

12. Exécutez maintenant la requete suivante : afficher le nom des villes, de leurs départements et de la région pour la région dont le numéro (id) est 91. Explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues.

```
select v.nom, d.nom, r.nom
from ville v JOIN departement d ON v.dep=d.id JOIN region r on r.id=d.reg
WHERE r.id=91;
```

Voir Figure 22.
Uniquement des NESTED LOOPS. Cependant tous les accès se font sur des index. Les INDEX RANGE SCAN correspondent aux index secondaires, qui n'ont pas de garantie d'être uniques. Ce n'est d'ailleurs pas le cas ici puisque les départements et les régions ne sont pas uniques dans les tables des index. D'après certaines sources, il semblerait que tant que la cardinalité est assez faible (moins de 4% des lignes pour chaque valeur d'index) INDEX RANGE SCAN reste plus efficace qu'un FULL TABLE ACCES. Étant donnée la distribution des villes dans les département il est bien possible que ce chiffre de 4% soit dépassé dans certains cas.

13. Exécutez maintenant la requete suivante : afficher le nom des villes dont le numéro de department (dep) commence par '7'. Explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues. Qu'en est-il de l'utilisation de l'index secondaire ?

```
SELECT v.nom
FROM ville v, departement d
WHERE v.dep=d.id
AND v.dep LIKE '7%';
```

Voir Figure 26.
Pas de soucis avec une chaîne dont le début est fixé, l'index peut être utilisé. Par contre si on voulait LIKE '%7', cela forcerait un TABLE ACCESS FULL puisque l'index ne pourrait pas être utilisé correctement. Cela est assez logique si l'on pend en compte le fonctionnement des arbres B, qu'Oracle utilise en général pour ses index. En pratique utiliser un index et son REVERSE peut être utile pour pouvoir gérer le cas ou une *wildcard* doît être uitilisée en début de chaîne.

Les statistiques des tables

14. Regarder les données disponibles dans la table USER_TAB_COL_STATISTICS pour les tables précédentes. Puis demander à Oracle de recalculer les statistiques sur les tables précédentes en utilisant la commande suivante :

```
exec dbms_stats.gather_table_stats('login','nom_table') ;
```

sur une table ou

```
exec dbms_stats.gather_schema_stats('login') ;
```

sur votre schéma.

Regarder à nouveau les données disponibles dans la table USER_TAB_COL_STATISTICS pour les tables précédentes. Ces informations correspondent aux données utilisées par Oracle pour évaluer le cout des différents plans d'éxécution afin de choisir le moins couteux.

```
exec dbms_stats.gather_table_stats('e20170005405', 'ville') ;
exec dbms_stats.gather_table_stats('e20170005405', 'departement') ;
exec dbms_stats.gather_table_stats('e20170005405', 'region') ;
exec dbms_stats.gather_schema_stats('e20170005405');
```

Avant l'exécution de la commande il n'y a pas de statistiques pour les tables crées récemment. Après l'exécution il y a des statistiques sur les minimas et maximas des colonnes, la densité, le nombre de buckets (probablement en rapport avec les index), et d'autres statistiques.

3 Annexe

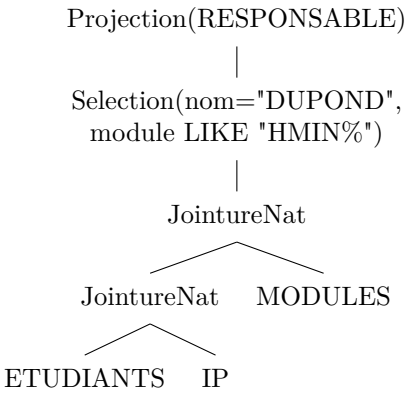


FIGURE 1 – Plan d'exécution logique n°1 (a)

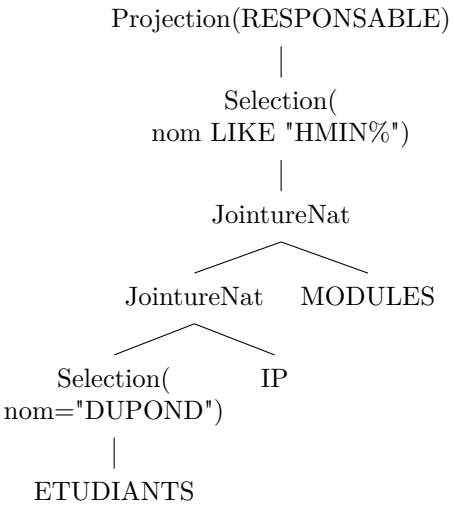


FIGURE 2 – Plan d'exécution logique n°1 (b)

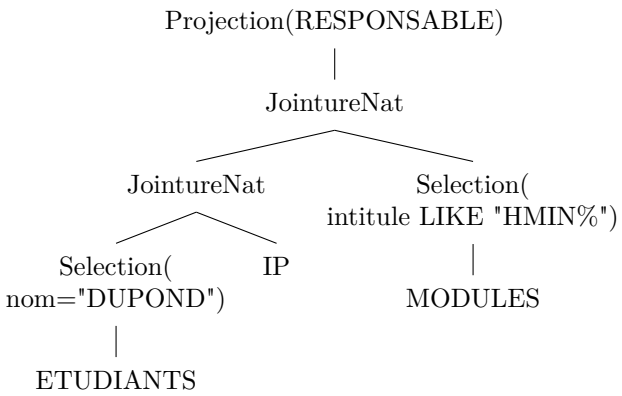


FIGURE 3 – Plan d'exécution logique n°1 (c)

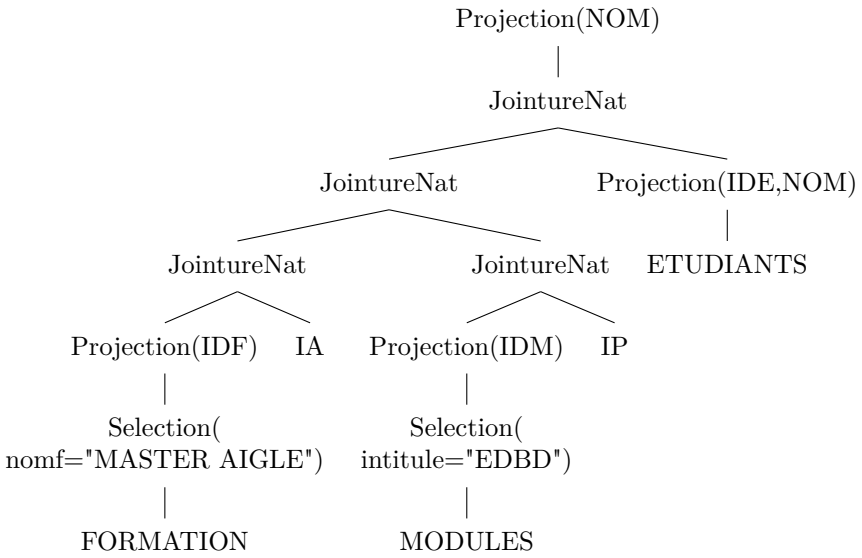


FIGURE 4 – Plan d'exécution logique n°2 (a)

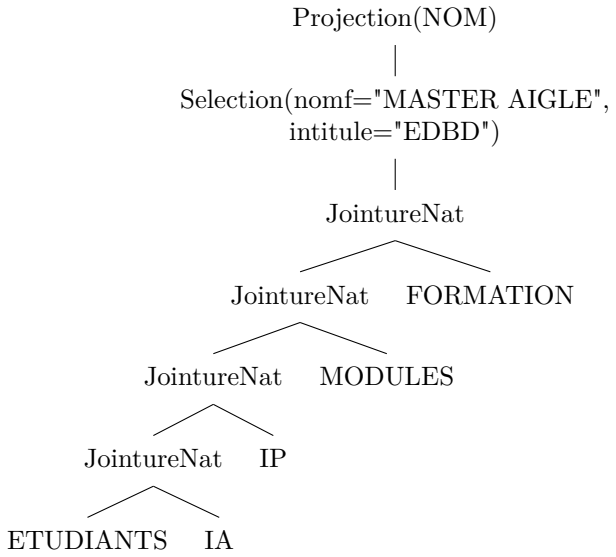


FIGURE 5 – Plan d'exécution logique n°2 (b)

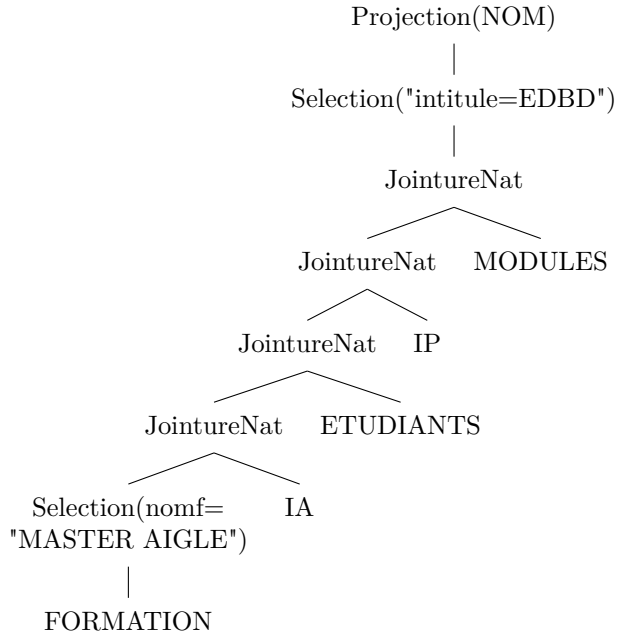


FIGURE 6 – Plan d'exécution logique n°2 (c)

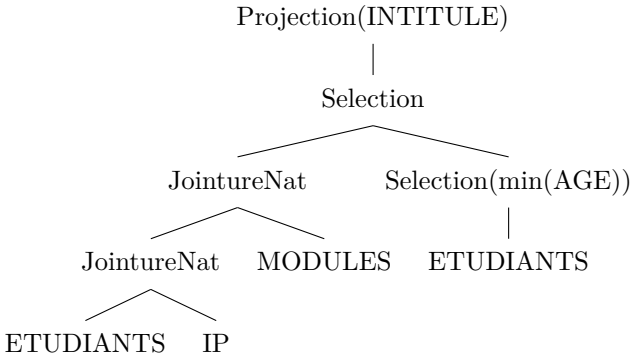


FIGURE 7 – Plan d'exécution logique n°3 (a)

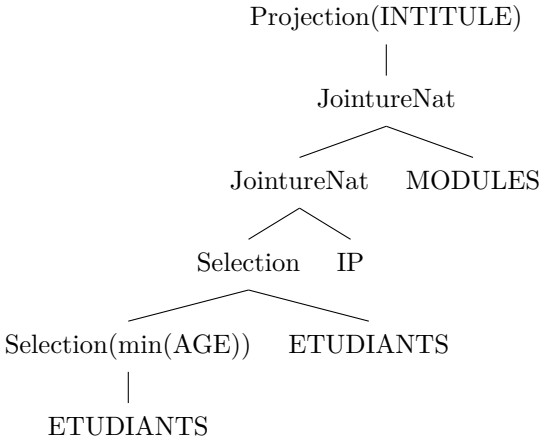


FIGURE 8 – Plan d'exécution logique n°3 (b)

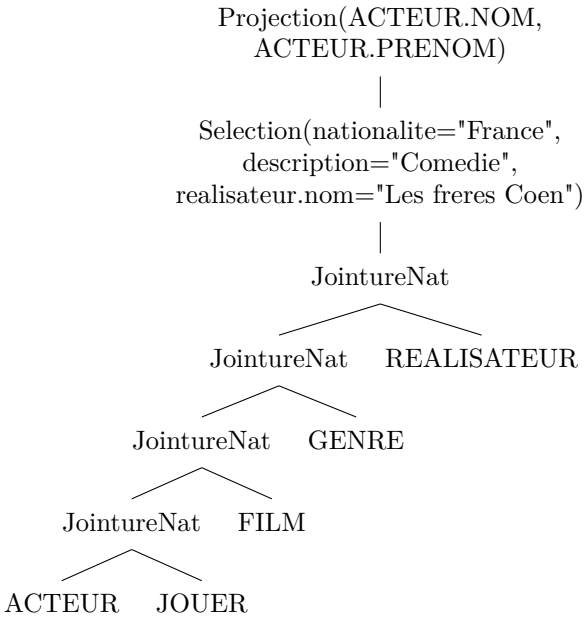


FIGURE 9 – Plan d'exécution logique n°4 (a)

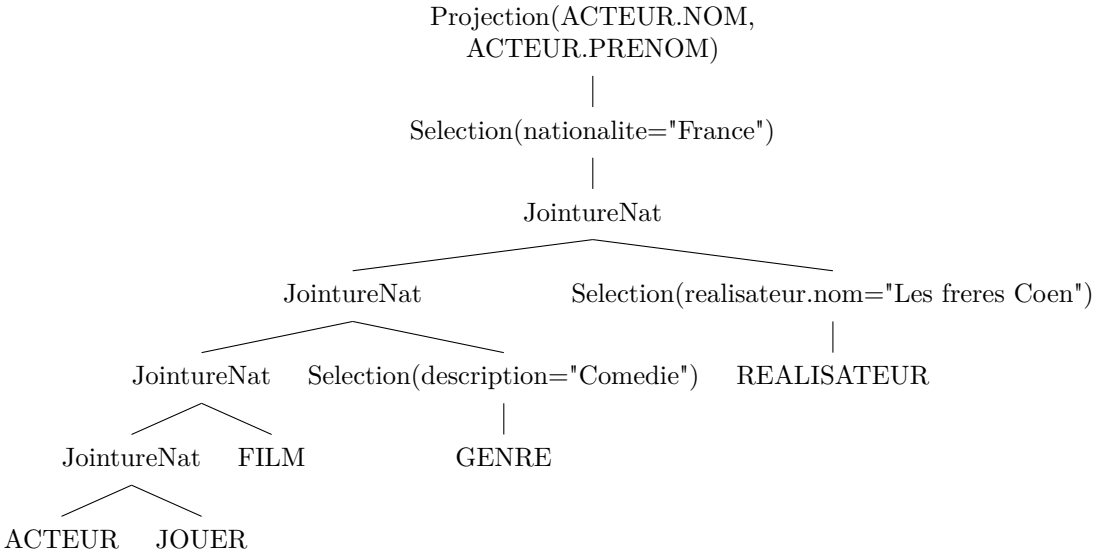


FIGURE 10 – Plan d'exécution logique n°4 (b)

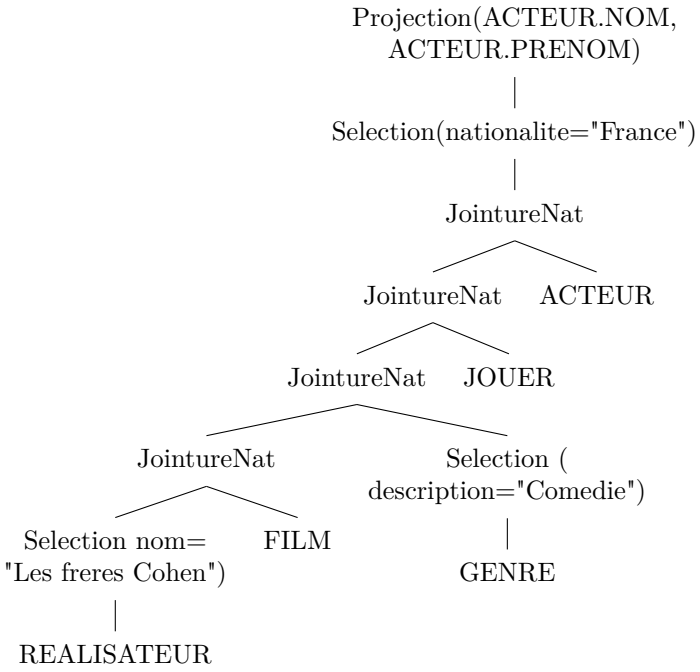


FIGURE 11 – Plan d'exécution logique n°4 (c)

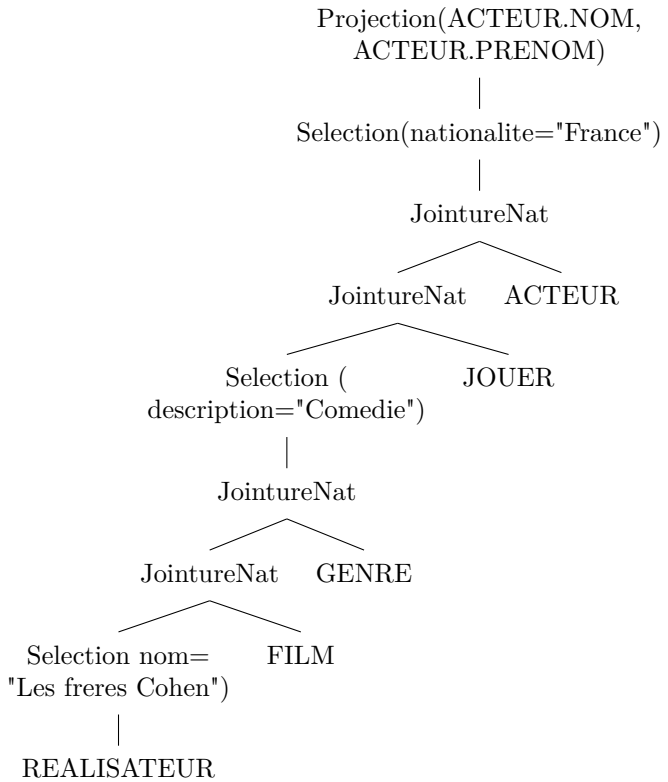


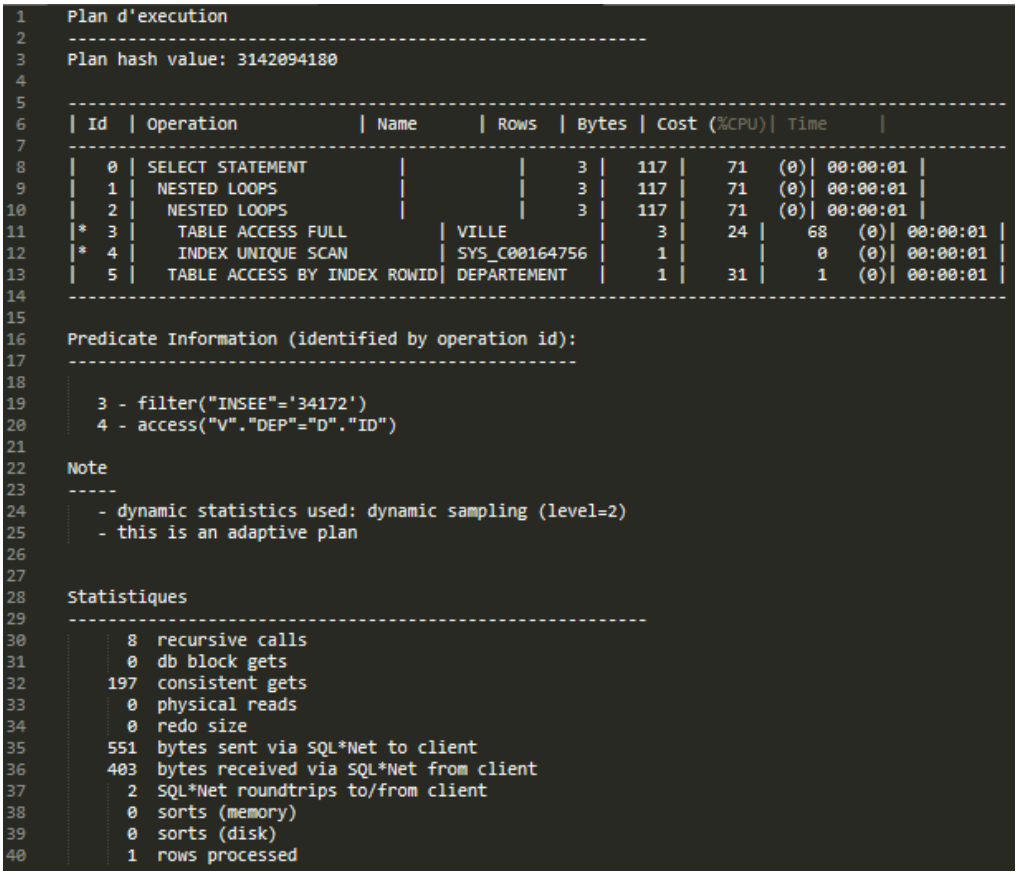
FIGURE 12 – Plan d'exécution logique n°4 (d)

```
Plan d'execution
----------------------------------------------------------
Plan hash value: 2371920588

--------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       | 3 |   168 |    68   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| VILLE | 3 |   168 |    68   (0)| 00:00:01 |
--------------------------------------------------------------------------------

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

   1 - filter("INSEE"='34172')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
         73  recursive calls
         12  db block gets
        327  consistent gets
          0  physical reads
       2072  redo size
        554  bytes sent via SQL*Net to client
        367  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

FIGURE 13 – Sortie de console Question 2

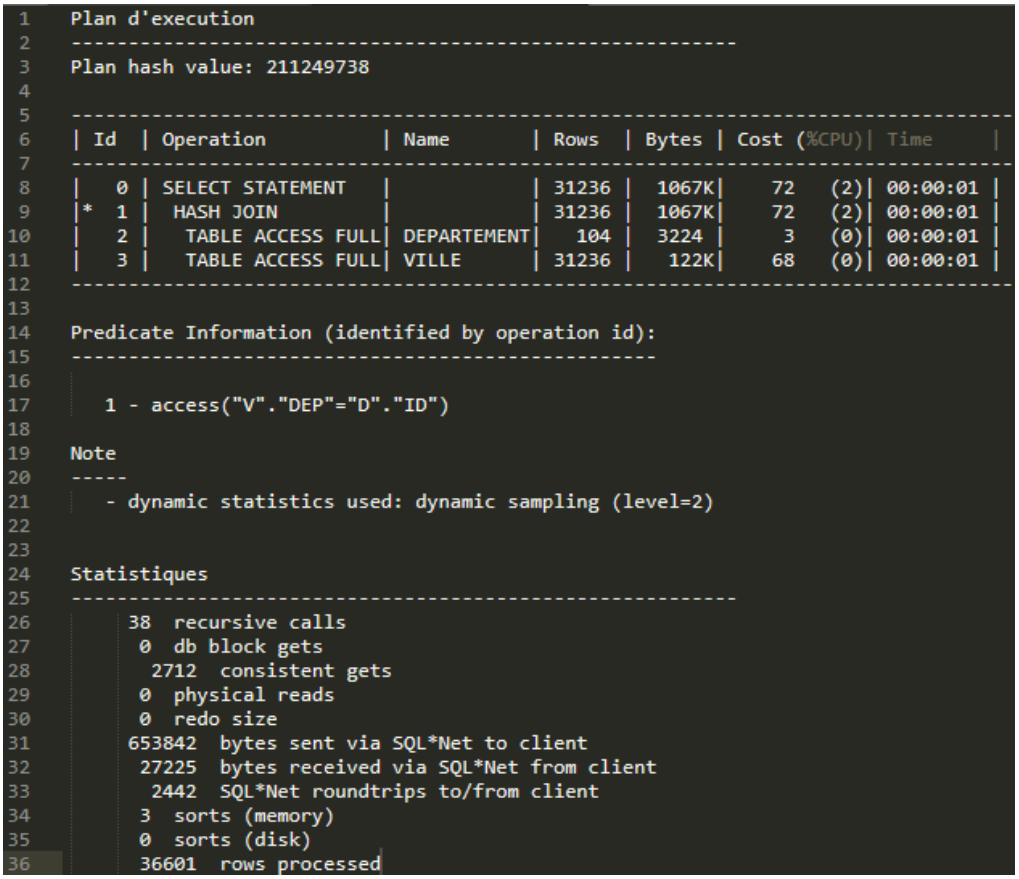
```
Plan d'execution
----------------------------------------------------------
Plan hash value: 2371920588

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       | 3 |   168 |    68   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| VILLE | 3 |   168 |    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("INSEE"='34172')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        192  consistent gets
          0  physical reads
          0  redo size
        788  bytes sent via SQL*Net to client
        367  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

FIGURE 14 – Sortie de console Question 4



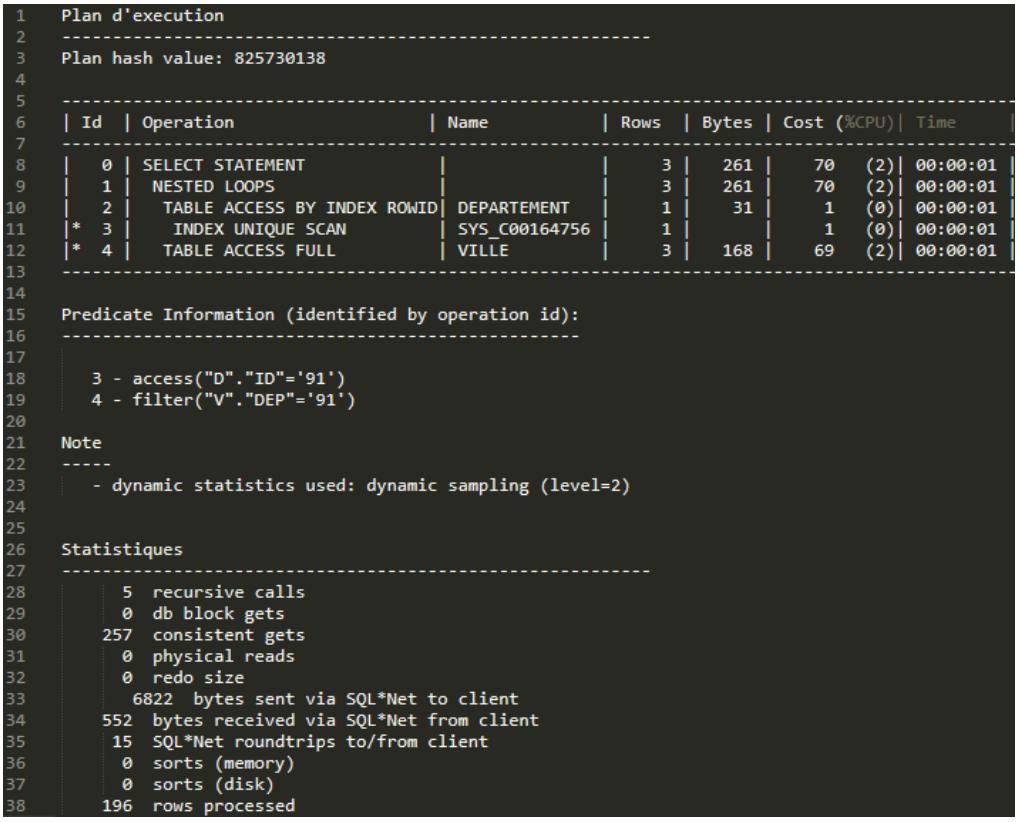
```
Plan d'execution
----------------------------------------------------------
Plan hash value: 3142094180

-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     3 |   117 |    71   (0)| 00:00:01 |
|   1 |  NESTED LOOPS        |           |     3 |   117 |    71   (0)| 00:00:01 |
|   2 |   NESTED LOOPS       |           |     3 |   117 |    71   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL | VILLE     |     3 |    24 |    68   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN | SYS_C00164756 |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT |     1 |    31 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   3 - filter("INSEE"='34172')
   4 - access("V"."DEP"="D"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
          8  recursive calls
          0  db block gets
        197  consistent gets
          0  physical reads
          0  redo size
        551  bytes sent via SQL*Net to client
        403  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

FIGURE 15 – Sortie de console Question 5



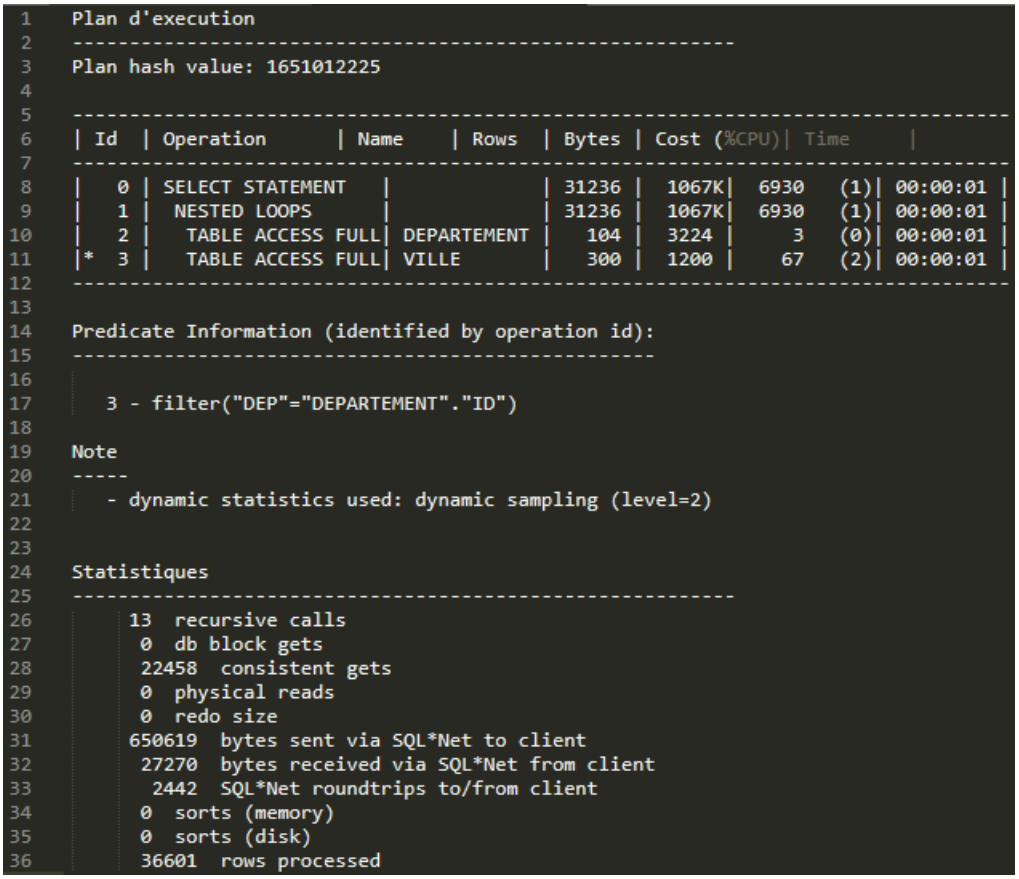
```
Plan d'execution
----------------------------------------------------------
Plan hash value: 211249738

-------------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            | 31236 |  1067K|    72   (2)| 00:00:01 |
|*  1 |  HASH JOIN         |            | 31236 |  1067K|    72   (2)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTEMENT|   104 |  3224 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| VILLE      | 31236 |   122K|    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("V"."DEP"="D"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
     38  recursive calls
      0  db block gets
     2712  consistent gets
      0  physical reads
      0  redo size
     653842  bytes sent via SQL*Net to client
      27225  bytes received via SQL*Net from client
       2442  SQL*Net roundtrips to/from client
      3  sorts (memory)
      0  sorts (disk)
      36601  rows processed
```

FIGURE 16 – Sortie de console Question 6



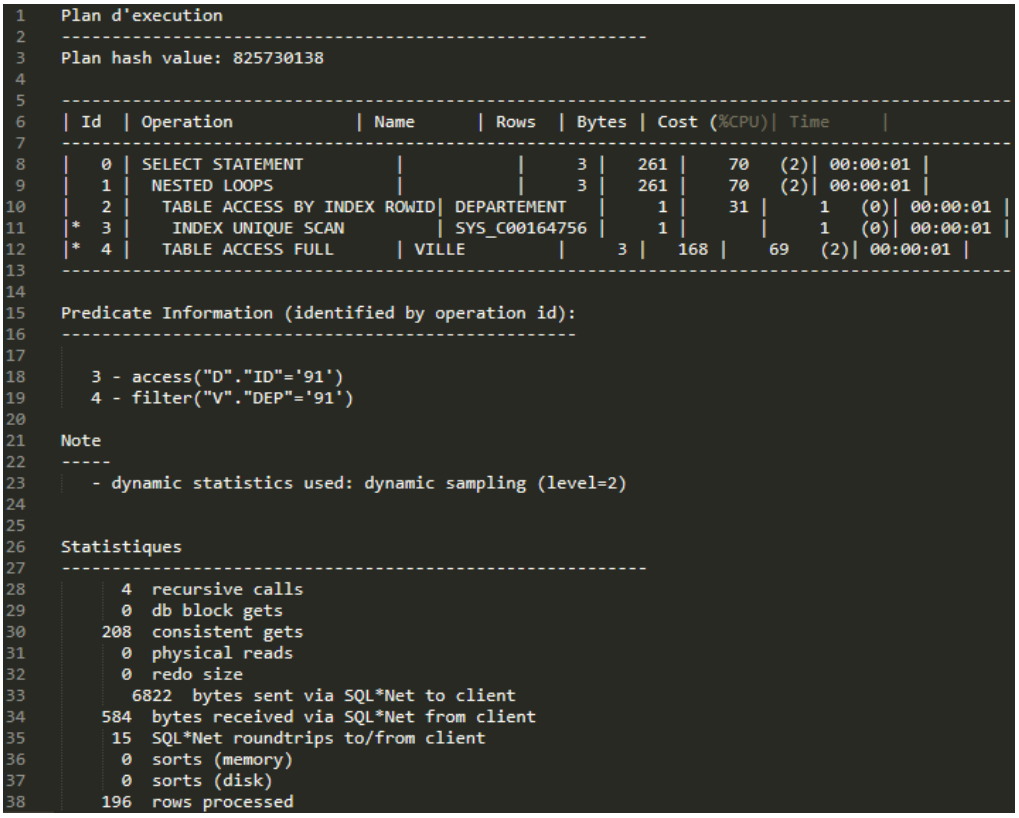
```
Plan d'execution
----------------------------------------------------------
Plan hash value: 825730138

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     3 |   261 |    70   (2)| 00:00:01 |
|   1 |  NESTED LOOPS                |                |     3 |   261 |    70   (2)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT    |     1 |    31 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | SYS_C00164756 |     1 |       |     1   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL          | VILLE          |     3 |   168 |    69   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   3 - access("D"."ID"='91')
   4 - filter("V"."DEP"='91')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
      5  recursive calls
      0  db block gets
    257  consistent gets
      0  physical reads
      0  redo size
      6822  bytes sent via SQL*Net to client
    552  bytes received via SQL*Net from client
     15  SQL*Net roundtrips to/from client
      0  sorts (memory)
      0  sorts (disk)
    196  rows processed
```

FIGURE 17 – Sortie de console Question 7



```
Plan d'execution
----------------------------------------------------------
Plan hash value: 1651012225

-------------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            | 31236 |  1067K|  6930   (1)| 00:00:01 |
|   1 |  NESTED LOOPS      |            | 31236 |  1067K|  6930   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTEMENT |   104 |  3224 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| VILLE      |   300 |  1200 |    67   (2)| 00:00:01 |
-------------------------------------------------------------------------------------

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

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

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
         13  recursive calls
          0  db block gets
      22458  consistent gets
          0  physical reads
          0  redo size
     650619  bytes sent via SQL*Net to client
      27270  bytes received via SQL*Net from client
       2442  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      36601  rows processed
```

FIGURE 18 – Sortie de console Question 8 requête 6

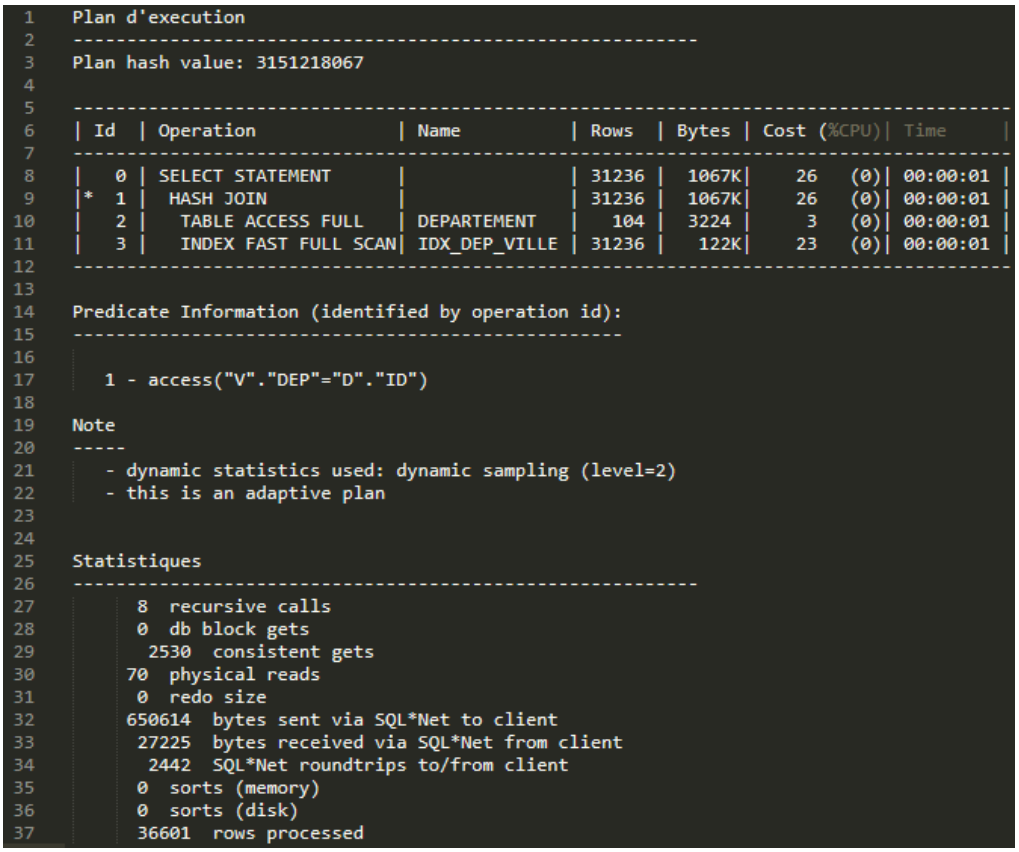


```
Plan d'execution
----------------------------------------------------------
Plan hash value: 825730138

-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     3 |   261 |    70   (2)| 00:00:01 |
|   1 |  NESTED LOOPS        |           |     3 |   261 |    70   (2)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT   |     1 |    31 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN | SYS_C00164756 |     1 |       |     1   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL  | VILLE         |     3 |   168 |    69   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   3 - access("D"."ID"='91')
   4 - filter("V"."DEP"='91')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
          4  recursive calls
          0  db block gets
        208  consistent gets
          0  physical reads
          0  redo size
       6822  bytes sent via SQL*Net to client
        584  bytes received via SQL*Net from client
         15  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        196  rows processed
```

FIGURE 19 – Sortie de console Question 8 requête 7

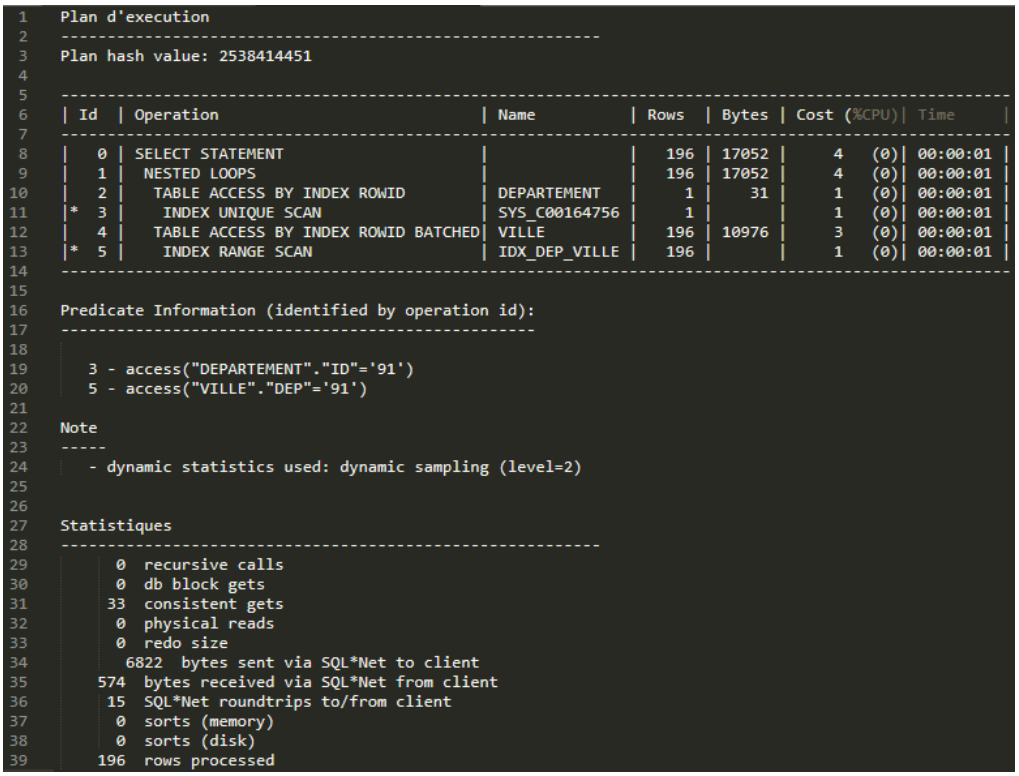
```
Plan d'execution
----------------------------------------------------------
Plan hash value: 3142094180

-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |     3 |   117 |    71   (0)| 00:00:01 |
|   1 |  NESTED LOOPS            |           |     3 |   117 |    71   (0)| 00:00:01 |
|   2 |   NESTED LOOPS           |           |     3 |   117 |    71   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL         | VILLE         |     3 |    24 |    68   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | SYS_C00164756 |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT   |     1 |    31 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   3 - filter("INSEE"='34172')
   4 - access("V"."DEP"="D"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
          8  recursive calls
          0  db block gets
        197  consistent gets
          0  physical reads
          0  redo size
        551  bytes sent via SQL*Net to client
        403  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

FIGURE 20 – Sortie de console Question 9 requête 5



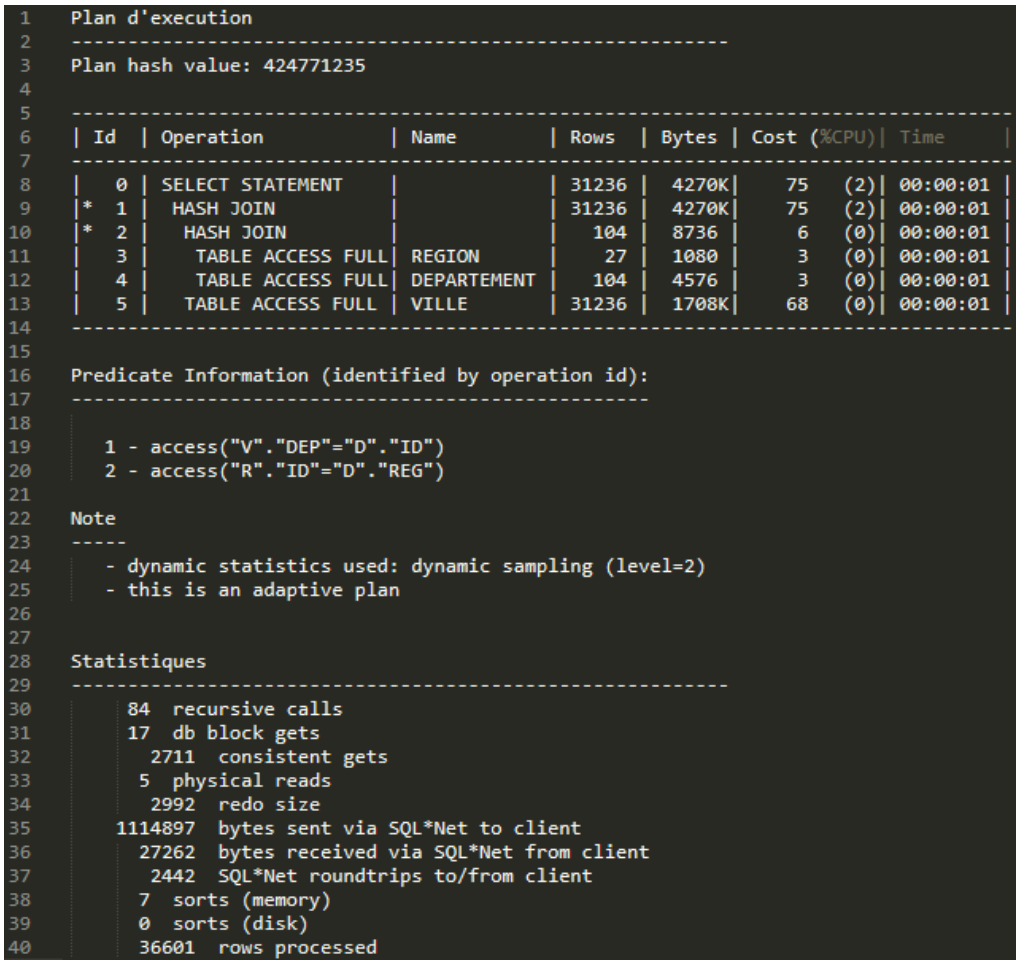
```
Plan d'execution
----------------------------------------------------------
Plan hash value: 3151218067

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |               | 31236 |  1067K|    26   (0)| 00:00:01 |
|*  1 |  HASH JOIN            |               | 31236 |  1067K|    26   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL   | DEPARTEMENT   |   104 |  3224 |     3   (0)| 00:00:01 |
|   3 |   INDEX FAST FULL SCAN| IDX_DEP_VILLE | 31236 |   122K|    23   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - access("V"."DEP"="D"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
         8  recursive calls
         0  db block gets
      2530  consistent gets
        70  physical reads
         0  redo size
    650614  bytes sent via SQL*Net to client
     27225  bytes received via SQL*Net from client
      2442  SQL*Net roundtrips to/from client
         0  sorts (memory)
         0  sorts (disk)
     36601  rows processed
```

FIGURE 21 – Sortie de console Question 9 requête 6



```
Plan d'execution
----------------------------------------------------------
Plan hash value: 2538414451

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |   196 | 17052 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                       |               |   196 | 17052 |     4   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID       | DEPARTEMENT   |     1 |    31 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN                | SYS_C00164756 |     1 |       |     1   (0)| 00:00:01 |
|   4 |   TABLE ACCESS BY INDEX ROWID BATCHED| VILLE         |   196 | 10976 |     3   (0)| 00:00:01 |
|*  5 |    INDEX RANGE SCAN                 | IDX_DEP_VILLE |   196 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

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

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         33  consistent gets
          0  physical reads
          0  redo size
       6822  bytes sent via SQL*Net to client
        574  bytes received via SQL*Net from client
         15  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        196  rows processed
```

FIGURE 22 – Sortie de console Question 9 requête 7

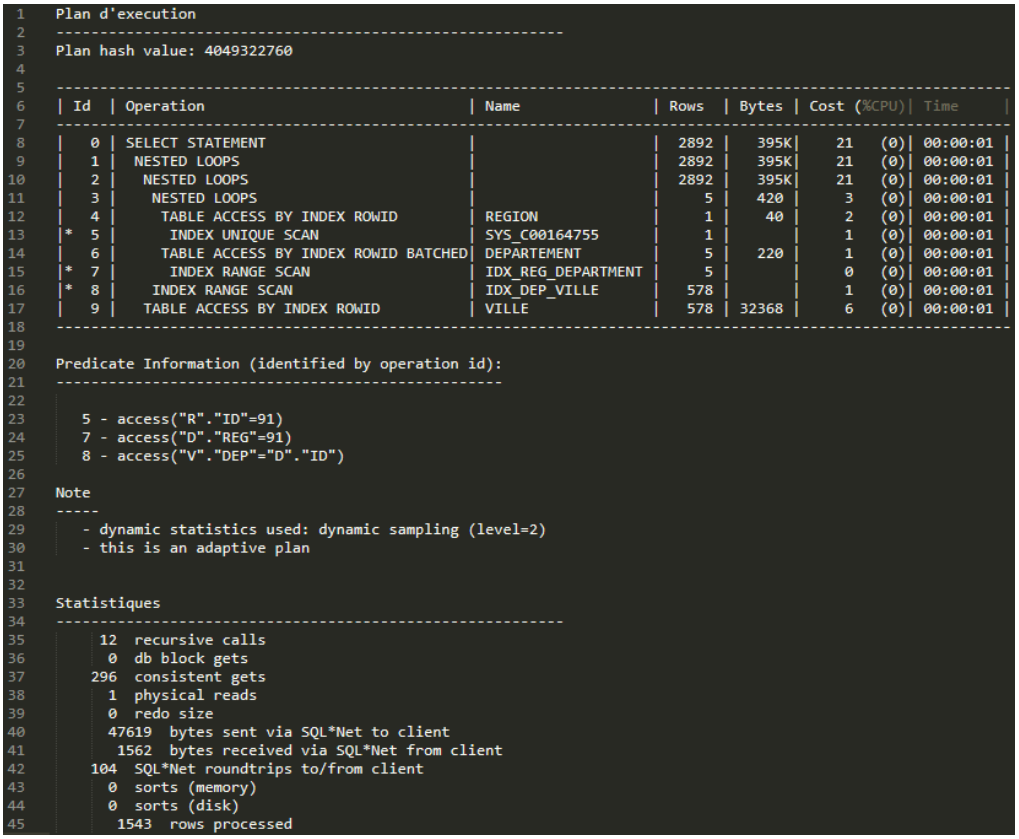


```
Plan d'execution
----------------------------------------------------------
Plan hash value: 424771235

-------------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             | 31236 |  4270K|    75   (2)| 00:00:01 |
|*  1 |  HASH JOIN          |             | 31236 |  4270K|    75   (2)| 00:00:01 |
|*  2 |   HASH JOIN         |             |   104 |  8736 |     6   (0)| 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       | 31236 |  1708K|    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("V"."DEP"="D"."ID")
   2 - access("R"."ID"="D"."REG")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
      84  recursive calls
      17  db block gets
    2711  consistent gets
       5  physical reads
    2992  redo size
 1114897  bytes sent via SQL*Net to client
   27262  bytes received via SQL*Net from client
    2442  SQL*Net roundtrips to/from client
       7  sorts (memory)
       0  sorts (disk)
   36601  rows processed
```

FIGURE 23 – Sortie de console Question 10

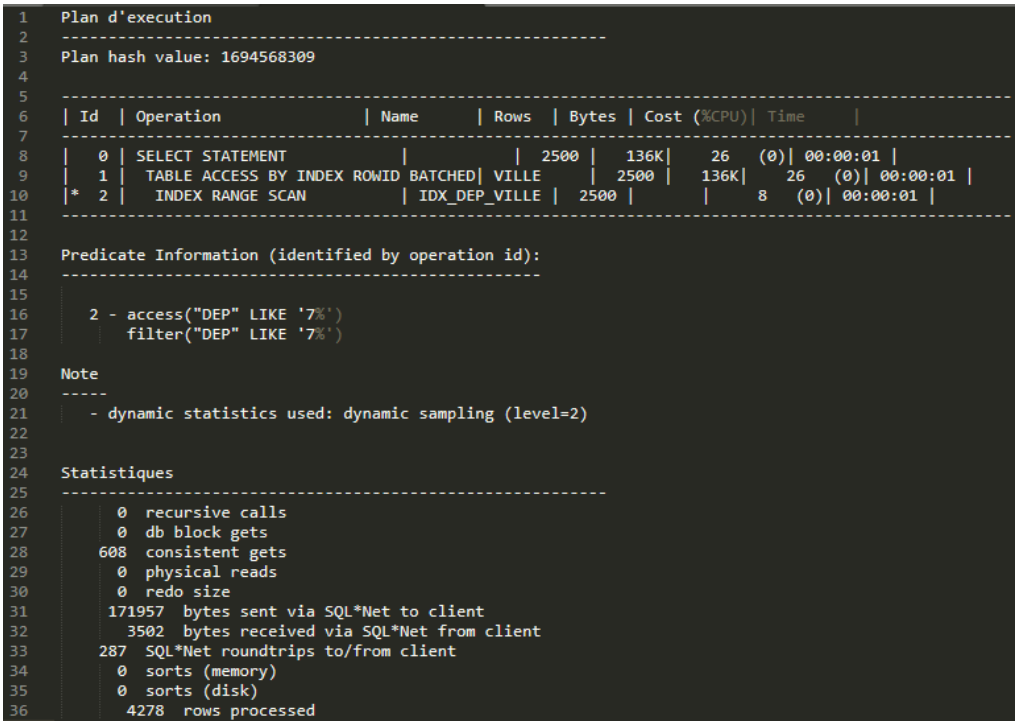
```
Plan d'executiPlan d'execution
----------------------------------------------------------
Plan hash value: 3225228285

-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     | 31236 |  4270K|    75   (3)| 00:00:01 |
|*  1 |  HASH JOIN                    |                     | 31236 |  4270K|    75   (3)| 00:00:01 |
|   2 |   MERGE JOIN                  |                     |   104 |  8736 |     6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTEMENT         |   104 |  4576 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | IDX_REG_DEPARTMENT  |   104 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |                     |    27 |  1080 |     4  (25)| 00:00:01 |
|   6 |     TABLE ACCESS FULL         | REGION              |    27 |  1080 |     3   (0)| 00:00:01 |
|   7 |   TABLE ACCESS FULL           | VILLE               | 31236 |  1708K|    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

   1 - access("V"."DEP"="D"."ID")
   5 - access("R"."ID"="D"."REG")
       filter("R"."ID"="D"."REG")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
         11  recursive calls
          0  db block gets
       2642  consistent gets
          0  physical reads
          0  redo size
    1114897  bytes sent via SQL*Net to client
      27262  bytes received via SQL*Net from client
       2442  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
      36601  rows processed
```

FIGURE 24 – Sortie de console Question 11



```
Plan d'execution
----------------------------------------------------------
Plan hash value: 4049322760

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                     |  2892 |   395K|    21   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                         |                     |  2892 |   395K|    21   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                        |                     |  2892 |   395K|    21   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                       |                     |     5 |   420 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID       | REGION              |     1 |    40 |     2   (0)| 00:00:01 |
|*  5 |      INDEX UNIQUE SCAN                | SYS_C00164755       |     1 |       |     1   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID BATCHED| DEPARTEMENT        |     5 |   220 |     1   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN                 | IDX_REG_DEPARTMENT  |     5 |       |     0   (0)| 00:00:01 |
|*  8 |    INDEX RANGE SCAN                   | IDX_DEP_VILLE       |   578 |       |     1   (0)| 00:00:01 |
|   9 |   TABLE ACCESS BY INDEX ROWID         | VILLE               |   578 | 32368 |     6   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

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

   5 - access("R"."ID"=91)
   7 - access("D"."REG"=91)
   8 - access("V"."DEP"="D"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan


Statistiques
----------------------------------------------------------
     12  recursive calls
      0  db block gets
    296  consistent gets
      1  physical reads
      0  redo size
    47619  bytes sent via SQL*Net to client
     1562  bytes received via SQL*Net from client
    104  SQL*Net roundtrips to/from client
      0  sorts (memory)
      0  sorts (disk)
     1543  rows processed
```

FIGURE 25 – Sortie de console Question 12



```
Plan d'execution
----------------------------------------------------------
Plan hash value: 1694568309

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |  2500 |   136K|    26   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| VILLE     |  2500 |   136K|    26   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_DEP_VILLE |  2500 |       |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

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

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistiques
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        608  consistent gets
          0  physical reads
          0  redo size
     171957  bytes sent via SQL*Net to client
       3502  bytes received via SQL*Net from client
        287  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
       4278  rows processed
```

FIGURE 26 – Sortie de console Question 13