Entrepôts de données et Big-Data - TP2

Elliot DURAND

Question 1

- `script_table` permet de créer les tables `ville`, `region` et `departement`.
- `script_remplissage` permet d'insérer des données dans ces tables (régions, départements, villes).

Question 2

```
explain plan for select nom from ville where insee=1293;
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |     3 |   168 |    69   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| VILLE |     3 |   168 |    69   (2)| 00:00:01 |
-------------------------------------------------------------------------------

1 - filter(TO_NUMBER("INSEE")=1293)
```

```
explain plan for select nom from ville where insee='1293';
```

```
-------------------------------------------------------------------------------
| 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 |
-------------------------------------------------------------------------------

1 - filter("INSEE"='1293')
```

Dans le premier plan, `INSEE` est converti en nombre avec `TO_NUMBER` car Oracle ne peut pas comparer directement un nombre et une chaîne.

Question 3

```
alter table ville add primary key(insee);
```

Question 4

```
explain plan for select nom from ville where insee=1293;

-------------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |     1 |    56 |    69   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| VILLE |     1 |    56 |    69   (2)| 00:00:01 |
-------------------------------------------------------------------------------

1 - filter(TO_NUMBER("INSEE")=1293)

-----------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes |(%CPU)| Time    |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |    56 | 2(0) | 00:00:01 |
|   1 | TABLE ACCESS BY INDEX ROWID  | VILLE         |     1 |    56 | 2(0) | 00:00:01 |
|*  2 | INDEX UNIQUE SCAN            | SYS_C00525152 |     1 |       | 1(0) | 00:00:01 |
-----------------------------------------------------------------------------------

2 - access("INSEE"='1293')
```

Dans la deuxième requête (avec la chaine de caractère) il utilise bien l'index que nous avons créer avec la clé primaire (table access by index rowid), nous avons moins de bytes utilisé également. Par contre ce n'est pas le cas pour la première requête (avec le nombre) car il est obligé de reconvertir insee en nombre (TO_NUMER).

Question 5

```
explain plan for select d.nom from departement d join ville v on d.id = v.dep
where v.insee='1293';

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost | 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         | SYS_C00525152 |     1 |       |  1(0)| 00:00:01 |
|   4 |   TABLE ACCESS BY INDEX ROWID| DEPARTEMENT   |     1 |    31 |  1(0)| 00:00:01 |
|*  5 |    INDEX UNIQUE SCAN         | SYS_C00525124 |     1 |       |  0(0)| 00:00:01 |
-----------------------------------------------------------------------------------------

3 - access("V"."INSEE"='1293')
5 - access("D"."ID"="V"."DEP")
```

Question 6

```
explain plan for select d.nom from departement d join ville v on d.id = v.dep
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             | 36601 |   607K|    72   (2)| 00:00:01 |
|*  1 |  HASH JOIN         |             | 36601 |   607K|    72   (2)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTEMENT |   104 |  1456 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| VILLE       | 36601 |   107K|    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

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

Ce n'est plus une nested loop utilisé pour la jointure mais un hash join. On utilise également plus les index mais table access full. Le hash join est utilisé car on a pas de condition de sélection pour réduire le nombre de ville à 1 comme à la requête précédente, un nested loop serait trop coûteux, on utilise donc un hash join qui sera bien plus rapide.

Question 7

```
explain plan for select /*+ use_nl(v d) */ d.nom from departement d join vill
on d.id = v.dep;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             | 36601 |   607K|  6937   (1)| 00:00:01 |
|   1 |  NESTED LOOPS      |             | 36601 |   607K|  6937   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTEMENT |   104 |  1456 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| VILLE       |   352 |  1056 |    67   (2)| 00:00:01 |
-------------------------------------------------------------------------------
```

Le coup de la nested loop monte a 6937 alors que celui du hash join est à 72, soit une opération 95x plus élevé pour la même requête. Le nombre de ligne sélectionnées reste le même mais le coup est bien plus élevé car nested loop va parcourir pour chaque ville son département.

Question 7

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             | 36601 |   607K|  6937   (1)| 00:00:01 |
|   1 | NESTED LOOPS       |             | 36601 |   607K|  6937   (1)| 00:00:01 |
|   2 | TABLE ACCESS FULL  | DEPARTEMENT |   104 |  1456 |     3   (0)| 00:00:01 |
|*  3 | TABLE ACCESS FULL  | VILLE       |   352 |  1056 |    67   (2)| 00:00:01 |
-------------------------------------------------------------------------------
```

```
3 - filter("D"."ID"="V"."DEP")

Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 1 (U - Unused (1))

2 - SEL$58A6D7F6 / D@SEL$1
      U - use_nl(v d)
```

Le coup de la nested loop monte a 6937 alors que celui du hash join est à 72, soit une opération 95x plus élevé pour la même requête. Le nombre de ligne sélectionnées reste le même mais le coup est bien plus élevé car nested loop va parcourir pour chaque ville son département.

Question 8

```
-------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost | Time    |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |     1 |   23 | 3(0) | 00:00:01 |
|   1 | NESTED LOOPS                  |               |     1 |   23 | 3(0) | 00:00:01 |
|   2 | TABLE ACCESS BY INDEX ROWID   | VILLE         |     1 |    9 | 2(0) | 00:00:01 |
|*  3 | INDEX UNIQUE SCAN             | SYS_C00525152 |     1 |      | 1(0) | 00:00:01 |
|   4 | TABLE ACCESS BY INDEX ROWID   | DEPARTEMENT   |     1 |   14 | 1(0) | 00:00:01 |
|*  5 | INDEX UNIQUE SCAN             | SYS_C00525124 |     1 |      | 0(0) | 00:00:01 |
-------------------------------------------------------------------------------------

3 - access("V"."INSEE"='1293')
5 - access("D"."ID"="V"."DEP")

-------------------------------------------------------------------------------------
| Id  | Operation             | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |               | 36601 |   607K|    25   (4)| 00:00:01 |
|*  1 | HASH JOIN             |               | 36601 |   607K|    25   (4)| 00:00:01 |
|   2 | TABLE ACCESS FULL     | DEPARTEMENT   |   104 |  1456 |     3   (0)| 00:00:01 |
|   3 | INDEX FAST FULL SCAN  | IDX_DEP_VILLE | 36601 |   107K|    21   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

Pour le plan de la question 5 rien n'a changé

Pour le plan de la question 6, on reste sur un hash join mais on voit que le coup n'est que de 25 comparé à 72. Cela est du au fait qu'il utilise l'index mis dans la table ville avec un coup de 21 au lieu de 68 avec l'opération "index fast full scan" pour la table ville. L'index que l'on à mis est donc efficace car il divise par 3 le coup sur notre même requête.

Question 9

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

```
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               | 36601 |  1751K|    75   (3)| 00:
|*  1 | HASH JOIN                     |               | 36601 |  1751K|    75   (3)| 00:
|   2 | MERGE JOIN                    |               |   104 |  3432 |     6  (17)| 00:
|   3 | TABLE ACCESS BY INDEX ROWID   | REGION        |    27 |   432 |     2   (0)| 00:
|   4 | INDEX FULL SCAN               | SYS_C00525123 |    27 |       |     1   (0)| 00:
|*  5 | SORT JOIN                     |               |   104 |  1768 |     4  (25)| 00:
|   6 | TABLE ACCESS FULL             | DEPARTEMENT   |   104 |  1768 |     3   (0)| 00:
|   7 | TABLE ACCESS FULL             | VILLE         | 36601 |   571K|    68   (0)| 00:
-------------------------------------------------------------------------------------

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

Note:
- this is an adaptive plan
```

Dans un premier temps, Oracle récupère la table region et departement en faisant un sort join sur departement pour ensuite faire un merge join entre les deux. Le sort join permet au merge join d'être très rapide car tout est déjà ordonné entre departement et region. Il fait ensuite un hash join entre la jointure de departement / region et ville (qu'il aura préalablement lu séquentiellement). Le cout total est de 75 pour 36K lignes sélectionnées, ce qui est très optimisé. On peut également voir qu'Oracle précise que ce plan est adaptatif en fonction du nombre réel de ligne qu'il va devoir sélectionnées, puisque les lignes qu'il nous affiches sont une estimation.

Question 10

```
-------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             | 36601 |  1751K|    75   (3)| 00:00
|*  1 | HASH JOIN                     |             | 36601 |  1751K|    75   (3)| 00:00
|   2 | MERGE JOIN                     |             |   104 |  3432 |     6  (17)| 00:0
|   3 | TABLE ACCESS BY INDEX ROWID   | DEPARTEMENT |   104 |  1768 |     2   (0)| 00:00
|   4 | INDEX FULL SCAN               | IDX_REG_DEP |   104 |       |     1   (0)| 00:00
|*  5 | SORT JOIN                     |             |    27 |   432 |     4  (25)| 00:00
|   6 | TABLE ACCESS FULL             | REGION      |    27 |   432 |     3   (0)| 00:00
|   7 | TABLE ACCESS FULL             | VILLE       | 36601 |   571K|    68   (0)| 00:00
-------------------------------------------------------------------------------------

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

Oracle applique la meme logique mais cette fois en "inversant" departement et region. Il utilise cette fois le sort join sur region et merge join entre le sort join de region et departement, avec lequel il aura acceder via l'index que l'on viens de créer. Un hash join à la fin mais cela n'affecte pas le coup ou bien le nombres de lignes. Cela montre que les index sont parfois pas plus efficace.

Question 11

```
-------------------------------------------------------------------------------------
| Id  | Operation                             | Name          | Rows  | Bytes | Cost (%
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |               |  1830 | 89670 |    21
|   1 | NESTED LOOPS                           |               |  1830 | 89670 |    21
|   2 | NESTED LOOPS                           |               |  1830 | 89670 |    21
|   3 | NESTED LOOPS                           |               |     5 |   165 |     2
|   4 | TABLE ACCESS BY INDEX ROWID            | REGION        |     1 |    16 |     1
|*  5 | INDEX UNIQUE SCAN                      | SYS_C00525123 |     1 |       |     0
|   6 | TABLE ACCESS BY INDEX ROWID BATCHED    | DEPARTEMENT   |     5 |    85 |     1
|*  7 | INDEX RANGE SCAN                       | IDX_REG_DEP   |     5 |       |     0
|*  8 | INDEX RANGE SCAN                       | IDX_DEP_VILLE |   366 |       |     1
|   9 | TABLE ACCESS BY INDEX ROWID            | VILLE         |   366 |  5856 |     4
-------------------------------------------------------------------------------------

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

Comme on a une condition qui réduit le nombre de région à 1, Oracle utilise une triple nested loop pour la jointure. En tout premier il filtre la table region (avec index unique scan) mais également la table departement (avec index range scan) avec l'index que l'on a créer sur la clé étrangère de département précédement. Il fait ensuite la jointure entre ville et département, pour chaque département on cherche les villes associées. Le cout monte à 21 pour 1830 ligne estimé, ce qui est très optimisé.

Question 12

```
-------------------------------------------------------------------------------------
| Id  | Operation                             | Name          | Rows  | Bytes | Cost (
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |               |   647 | 31703 |    33
|   1 | NESTED LOOPS                           |               |   647 | 31703 |    33
|   2 | NESTED LOOPS                           |               |   658 | 31703 |    33
|   3 | MERGE JOIN                             |               |    14 |   462 |     5
|   4 | TABLE ACCESS BY INDEX ROWID            | REGION        |    27 |   432 |     2
|   5 | INDEX FULL SCAN                        | SYS_C00525123 |    27 |       |     1
|*  6 | SORT JOIN                              |               |    14 |   238 |     3
|   7 | TABLE ACCESS BY INDEX ROWID BATCHED    | DEPARTEMENT   |    14 |   238 |     2
|*  8 | INDEX RANGE SCAN                       | SYS_C00525124 |    14 |       |     1
|*  9 | INDEX RANGE SCAN                       | IDX_DEP_VILLE |    47 |       |     1
|  10 | TABLE ACCESS BY INDEX ROWID            | VILLE         |    47 |   752 |     2
-------------------------------------------------------------------------------------

6 - access("D"."REG"="R"."ID")
    filter("D"."REG"="R"."ID")
8 - access("D"."ID" LIKE '7%')
    filter("D"."ID" LIKE '7%')
```

```
9 - access("V"."DEP"="D"."ID")
    filter("V"."DEP" LIKE '7%')
```

Ici, le nombre de département sortant étant plus élevé que 1 région comme précédement, Oracle fait un sort join sur departement avant de faire un mere join sur les deux, avant de faire une nested loop pour la jointure entre departement et ville. Cette approche combine les deux vu précédement et à un coup de 33 pour 647 lignes. Cela est forcément moins optimisé car Like et bien plus couteûx que de rechercher un id de région par exemple.

Question 13

Table 1: Statistiques des tables

Tables	Attribut	Nombres de lignes distinct estimé	Densité
VILLE	insee	36601	0,00002732
VILLE	nom	33772	0,00002961
VILLE	dep	100	0,00001366
DEPARTEMENT	id	104	0,00961538
DEPARTEMENT	nom	104	0,00961538
DEPARTEMENT	reg	27	0,00480769
REGION	id	27	0,03703704
REGION	nom	27	0,03703704

Il existe d'autres statistiques comme la valeur minimal / maximal, les histogrammes etc mais j'ai pris les valeurs qui me semblait les plus intéréssantes.

Après vérification approximatives toutes les statistiques sont vrais. Il y a effectivement plusieurs ville avec le même nom mais pas le même insee comme par exemple "chevry" avec l'insee 1103 et 50134.

Question 14

Après exécution de :

```
exec dbms_stats.gather_table_stats('e20210009747', 'ville');
```

La seule statistique changée est la date de mise à jour (23/09/2025 pour `VILLE` contre 22/09/2025 pour les autres tables).

Après execution de la commande sur les autres tables les statistiques n'ont pas changées non plus.