

## TP2 - Optimisation de requêtes PARTIE 2

# 1 Les plans d'exécution sous ORACLE

## 1.1 Sélection

**Question 1 :** Examinez les scripts pour comprendre ce qu'il font.

Il y a un script qui crée les tables et l'autre qui les remplit.

**Question 2 :** Explicitiez 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 1293 : testez avec `insee=1293` et `insee='1293'`, puis comparez.

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

Figure 1: Conversion de Insee en number

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	19	69 (2)	00:00:01
* 1	TABLE ACCESS FULL	VILLE	1	19	69 (2)	00:00:01

Figure 2: Plan de la question 2a

On voit que sur les figures 1 et 2 page 1 que le système réalise une conversion de type de la variable Insee en nombre car on n'utilise pas les quotes.

Ainsi le système considère que la requête n'a pas de problème syntaxique, donc elle va chercher les différents plans et donc réalise une sélection sur la totalité de la table ville. On voit également que le système n'a sélectionné aucune ligne mais que cela ne l'empêche pas de réaliser un plan.

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

Figure 3: Aucune conversion

Sur la requête en rajoutant des quotes pour l'attribut Insee, on voit que sur les figures 3 et 4 aux pages 1 et 2 qu'il ne fait pas de conversion en nombre donc il considère bien la chaîne de caractères donc il sélectionne dans ce cas une ligne et utilise également une sélection sur la totalité de la table ville.

**Question 3 :** Ajoutez une clé primaire sur la table ville (utiliser l'attribut insee).

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	224	68 (0)	00:00:01
* 1	TABLE ACCESS FULL	VILLE	4	224	68 (0)	00:00:01

Figure 4: Plan de la question 2b

On a ajouté la contrainte primary key sur l'attribut insee dans la table ville.

**Question 4 :** Explicitez à nouveau le plan d'exécution choisi par l'optimiseur et les statistiques obtenues lors de l'exécution de la requête précédente (afficher le nom des villes dont le numéro insee est 1293 en testant insee=1293 et insee='1293'). Quelles sont les différences observées par rapport à la question 2 ?

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	19	2 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	VILLE	1	19	2 (0)	00:00:01
* 2	INDEX UNIQUE SCAN	SYS_C00360844	1		1 (0)	00:00:01

Figure 5: Plan de la question 4

On vient donc de rajouter un indice en ajoutant la contrainte de clé primaire sur Insee, ce qui permet au système de l'utiliser pour retrouver plus facilement la valeur à sélectionner comme on le voit dans la figure 5 à la page 2, cela ne marche que quand on met les quotes sur le numéro 1293.

En revanche quand on ne met pas les quotes on se retrouve le même cas de figure que précédemment. Le système va simplement chercher sur la table ville le nom qui correspond au nombre 1293 et non à la chaîne de caractères '1293'. On a donc pas réaffiché ce que fait le système.

## 1.2 Jointure

**Question 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 insee='1293'.

Id	Operation	Name	R
0	SELECT STATEMENT		
1	NESTED LOOPS		
2	TABLE ACCESS BY INDEX ROWID	VILLE	
* 3	INDEX UNIQUE SCAN	SYS_C00360844	
4	TABLE ACCESS BY INDEX ROWID	DEPARTEMENT	
* 5	INDEX UNIQUE SCAN	SYS_C00360846	

Figure 6: Plan de la question 5

On voit sur la figure 6 à la page 2 que l'optimiseur choisit d'utiliser deux index, un sur les villes pour trouver la ville avec l'indice '1293' et un sur la table département en prenant l'identifiant du département dans lequel la ville est.

Cette utilisation va donc plus vite sachant qu'il cherche directement un identifiant et non une sélection sur toute la table comme avant.

**Question 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 ?**

Id	Operation	Name
0	SELECT STATEMENT	
* 1	HASH JOIN	
2	TABLE ACCESS FULL	DEPARTEMENT
3	TABLE ACCESS FULL	VILLE

Figure 7: Plan de la question 6

On voit sur la figure 7 à la page 3 que les index ne sont pas utilisés car le SELECT ne porte pas sur des clefs primaires.

En revanche, l'optimiseur effectue une jointure par hachage, car on doit sélectionner un grand nombre de lignes, après les avoir en totalité toutes lues dans les tables département et ville.

### 1.3 Modification du comportement de l'optimiseur

**Question 7 : Essayez maintenant la requête de la question 6 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.**

Id	Operation	Name	Rows
0	SELECT STATEMENT		3
1	NESTED LOOPS		3
2	TABLE ACCESS FULL	DEPARTEMENT	
* 3	TABLE ACCESS FULL	VILLE	

Figure 8: Plan de la question 7

Sur la figure 8 à la page 3, on force l'optimiseur à utiliser des nested loops, ainsi on voit qu'il réalise bien des boucles imbriquées ce qui augmente la vitesse du SELECT et permettent de lire les premiers enregistrements sans avoir à attendre la fin de l'exécution.

## 1.4 Utilisation d'index

**Question 8** Créer un index secondaire sur l'attribut dep de la table ville : create index idx\_dep\_ville on ville (dep). Ré-exécutez les requêtes des questions 5 et 6 et explicitez le plan d'exécution choisi par l'optimiseur et les statistiques obtenues.

Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	DEPARTEMENT
* 3	INDEX UNIQUE SCAN	SYS_C00360846
4	TABLE ACCESS BY INDEX ROWID BATCHED	VILLE
* 5	INDEX RANGE SCAN	IDX_DEP_VILLE

Figure 9: Plan de la question 8

On voit sur la figure 9 à la page 4 que une fois que l'on a créé un autre index pour relier les départements et les villes, le système l'utilise pour aller plus facilement récupérer les villes d'un certain département auparavant choisi par un indice système.

**Question 9** Exécutez la requete 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.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		43176	5902K	75 (2)	00:00:01
* 1	HASH JOIN		43176	5902K	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	43176	2361K	69 (2)	00:00:01

Figure 10: Plan de la question 9

On voit sur la figure 10 à la page 4 que l'optimiseur réalise une jointure de type hash join entre les lignes des tables region et département pour refaire un hash join avec les lignes qu'on vient de voir et les lignes de la table ville.

On utilise donc pas les indices créés auparavant car on ne demande pas un département, une région ou une ville précise on veut simplement manipuler beaucoup de lignes en même d'où le fait que l'optimiseur utilise des hash join.

**Question 10** 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.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		43176	5902K	75 (3)	00:00:01
* 1	HASH JOIN		43176	5902K	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_DEP_REG	104		1 (0)	00:00:01
* 5	SORT JOIN		27	1080	4 (25)	00:00:01
PLAN_TABLE_OUTPUT						
6	TABLE ACCESS FULL	REGION	27	1080	3 (0)	00:00:01
7	TABLE ACCESS FULL	VILLE	43176	2361K	69 (2)	00:00:01

Figure 11: Plan de la question 10

Après avoir créé le nouvel index, on voit sur la figure 11 à la page 5 que le système utilise cet index pour récupérer les départements associés à chaque région ce qui permet de faire qu'un merge join est utilisé avec beaucoup moins de lignes. Un hash join est également utilisé pour lier la table ville.

**Question 11** 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.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3925	536K	21 (0)	00:00:01
1	NESTED LOOPS		3925	536K	21 (0)	00:00:01
2	NESTED LOOPS		3925	536K	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_C00362668	1		1 (0)	00:00:01
PLAN_TABLE_OUTPUT						
6	TABLE ACCESS BY INDEX ROWID BATCHED	DEPARTEMENT	5	220	1 (0)	00:00:01
* 7	INDEX RANGE SCAN	IDX_DEP_REG	5		0 (0)	00:00:01
* 8	INDEX RANGE SCAN	IDX_DEP_VILLE	785		1 (0)	00:00:01
9	TABLE ACCESS BY INDEX ROWID	VILLE	785	43960	6 (0)	00:00:01

Figure 12: Plan de la question 11

On voit, sur la figure 12 à la page 5, que l'optimiseur va pour cette requête utiliser les deux indices créés pour à la fois récupérer les départements de la région cherchée et les villes de chaque département associé. Par ailleurs, il utilise l'index du système sur les régions pour avoir la région que l'on demande directement. Ainsi au niveau des jointures on va pouvoir faire des nested loops qui sont utilisées pour beaucoup moins de lignes que le hash join par exemple.

**Question 12** Exécutez maintenant la requete suivante : afficher le nom des villes dont le numéro de département (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 ?

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	87	1 (0)	00:00:01
1	NESTED LOOPS		1	87	1 (0)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	DEPARTEMENT	1	31	1 (0)	00:00:01
* 3	INDEX UNIQUE SCAN	SYS_C00362669	1		1 (0)	00:00:01
4	TABLE ACCESS BY INDEX ROWID BATCHED	VILLE	1	56	0 (0)	00:00:01
* 5	INDEX RANGE SCAN	IDX_DEP_VILLE	1		0 (0)	00:00:01

Figure 13: Plan de la question 12

On voit sur la figure 13 à la page 6 que pour chercher l'indice du département il utilise l'index du système. Et pour récupérer toutes les villes qui sont associées au département on utilise l'index idx\_dep\_ville. Ainsi, l'utilisation de jointures nested loops est très efficace, car la requête n'a pas beaucoup de lignes à join.

## 1.5 Les statistiques des tables

**Question 13 :** Regardez les données disponibles dans la table `USER_TAB_COL_STATISTICS` pour les tables précédentes. Est-ce que les statistiques correspondent bien aux données présentes dans vos tables ?

On constate que ORACLE n'a pas fait de statistique sur les tables.

**Question 14 :** Demandez maintenant 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.

Regardez à nouveau les données disponibles dans la table `USER_TAB_COL_STATISTICS` pour les tables précédentes.

Ainsi après les avoir faites manuellement, on voit que les statistiques ont été faites sachant que les tables qu'on vient de voir sont dans la liste (figure 14 page 7).

```

TABLE_NAME
-----
APPAREIL_PHOTO
APPAREIL_PHOTO
APPAREIL_PHOTO
CONFIGURATION
CONFIGURATION
CONFIGURATION
CONFIGURATION
CONFIGURATION
CONFIGURATION
DEPARTEMENT
DEPARTEMENT

TABLE_NAME
-----
DEPARTEMENT
LICENCE
LICENCE
PHOTO
PHOTO
PHOTO
PHOTO
PHOTO
PHOTO
REGION
REGION

TABLE_NAME
-----
UTILISATEUR
UTILISATEUR
VILLE
VILLE
VILLE

27 lignes selectionnees.

```

Figure 14: Table dont les statistiques ont été faite