

**Département d’informatique et mathématiques**

**8TRD157 – Base de données avancées**

Travail pratique 4 : Optimisation de requêtes et analyse de la performance

|  |  |
| --- | --- |
| **Cours** | 8TRD157 – Base de données avancées |
| **Étudiant 1 (nom et CP)** | Valentin BAILLEUL (BAIV04059700) |
| **Étudiant 2 (nom et CP)** | Alexandre COX (…) |

|  |  |
| --- | --- |
| **Rapport** | **(100 pts)** |
| **Code SQL pour le schéma interne**  **(création d'index et CLUSTER)** | **/ 15** |
| **Reformulation des requêtes avec indices**  **(et nouvelles tables)** | **/ 15** |
| **Méthodologie expérimentale (plans d'exécution et statistiques)** | **/ 15** |
| **Analyse des résultats** | **/ 40** |
| **Choix de la configuration (jointure double et sélection) et justification** | **/ 15** |
|  |  |
| **Total** | **/ 100** |

# Opérations de sélection

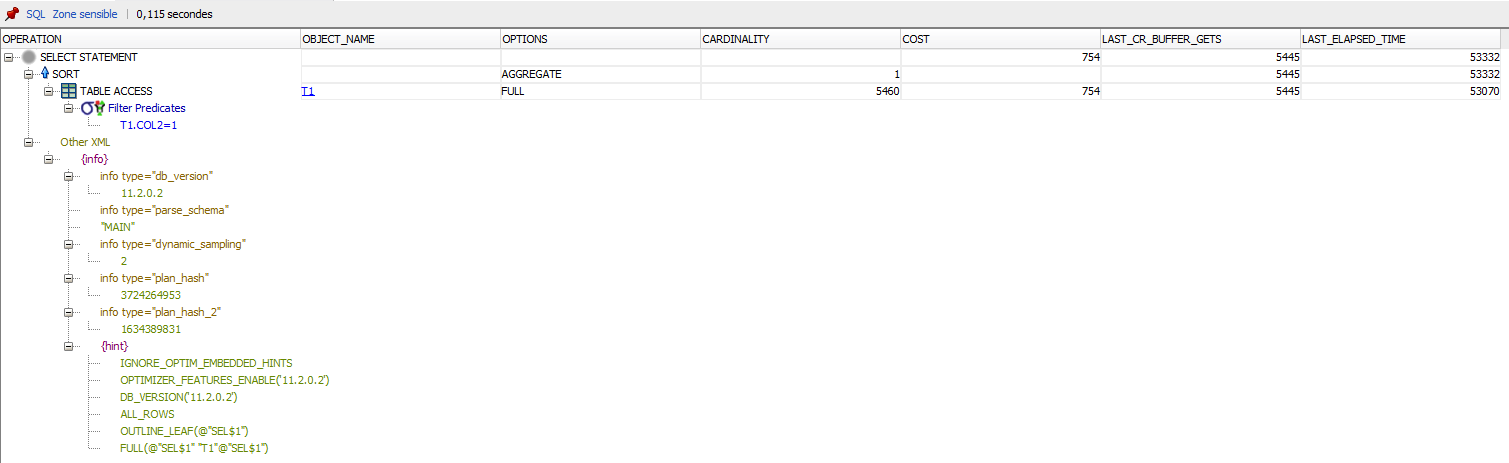
## Sélectivité petite

### Évaluation des optimisations

[Pour chacune des configurations suivantes, donnez 1) le code SQL employé pour créer les indexes ou CLUSTER requis, 2) la requête testée comprenant les indices et, possiblement, les nouvelles tables insérées dans les CLUSTER, 3) le plan d'exécution et les statistiques obtenues lors de l'exécution de la requête]

***Sans optimisation***

SELECT /\*+ FULL\*/ COUNT(\*) FROM T1 WHERE T1.col2=1

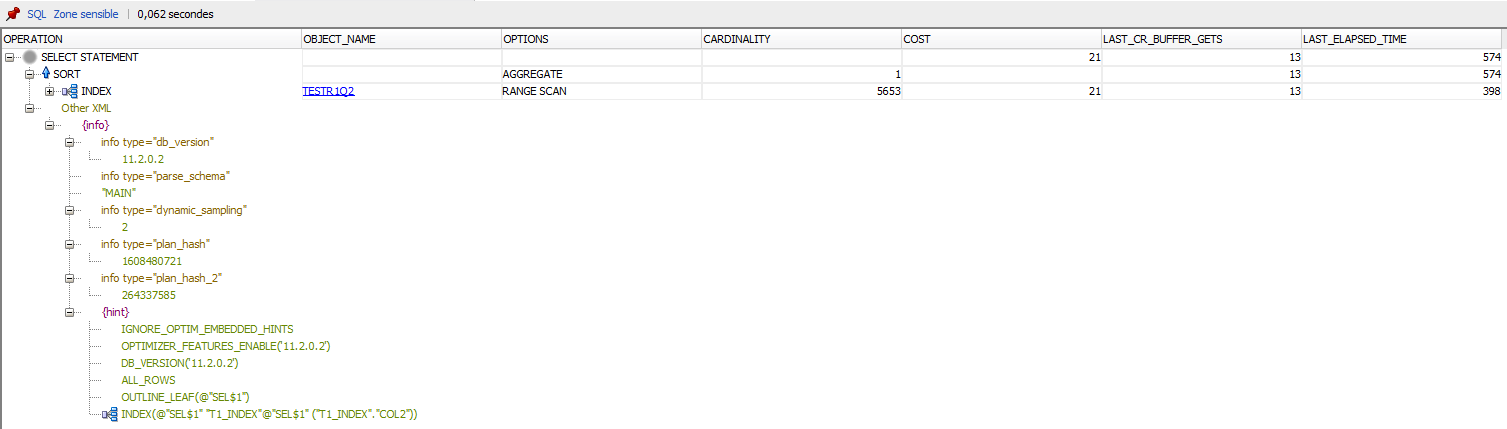


***Avec index secondaire et indice INDEX ou INDEX\_FFS (précisez lequel a été choisi)***

CREATE TABLE T1\_INDEX AS SELECT \* FROM T1;

CREATE INDEX testR1Q2 ON T1\_INDEX(col2);

SELECT /\*+ INDEX(T1\_INDEX col2)\*/ COUNT(\*) FROM T1\_INDEX WHERE T1\_INDEX.col2=1



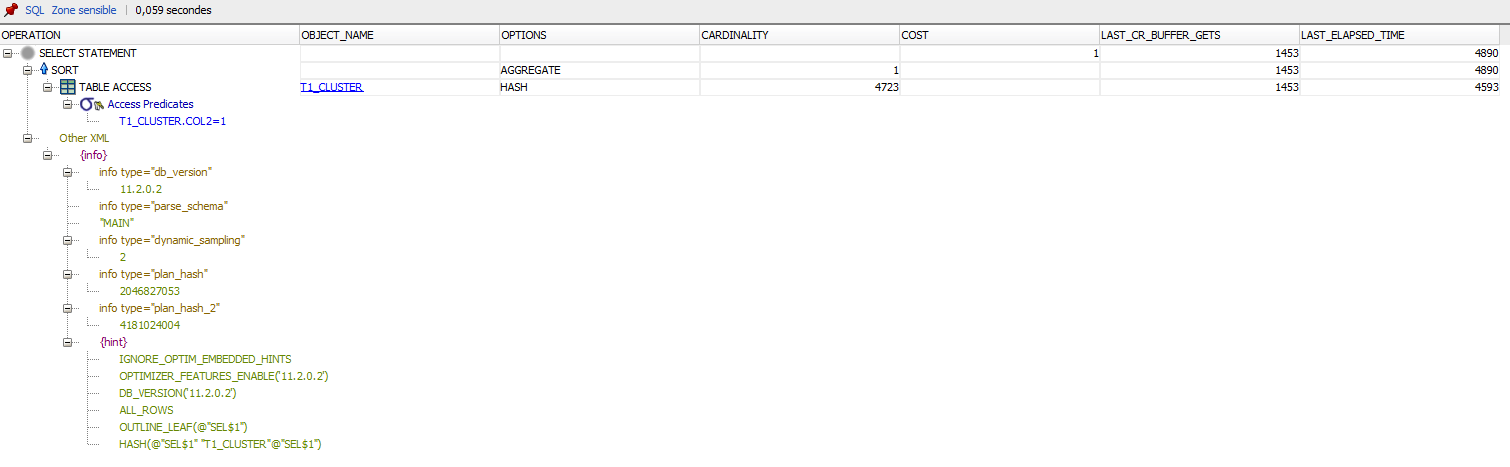
***Avec HASH CLUSTER***

CREATE CLUSTER Grappe1 (col2 INTEGER)

SIZE 160 HASHKEYS 100000;

CREATE TABLE T1\_CLUSTER CLUSTER Grappe1(col2) AS SELECT \* FROM T1;

SELECT /\*+ HASH(T1\_CLUSTER col2)\*/ COUNT(\*) FROM T1\_CLUSTER WHERE T1\_CLUSTER.col2=1



### Analyse globale

[Faites une courte analyse (10-15 lignes) expliquant les résultats obtenus à la lumière de la matière vue en classe]

Après avoir obtenu le résultat des trois configurations différentes à tester, nous pouvons les comparer. Tout d’abord, nous pouvons remarquer que le temps d’exécution de la requête avec le FULL est certes faible mais reste deux fois plus grand que celles des deux configurations suivantes (à savoir INDEX et CLUSTER). De plus, on peut remarquer dans la colonne LAST\_ELAPSED\_TIME qu’il y a également une large différence entre cette configuration et les deux autres (pour cette colonne, la configuration avec INDEX obtient la plus petite valeur, cependant certaines valeurs données aux paramètres de la configuration du CLUSTER aurait permis d’obtenir une plus petite valeur). En ce qui concerne le coût de l’opération, on peut observer que celui du CLUSTER est nettement plus inférieur à celui des deux autres configurations. Une autre remarque peut être observer au niveau des options. Effectivement, on peut remarquer que l’option créé sur la configuration INDEX est RANGE SCAN, les index sont donc traités en série et non en parallèle. Pour terminer, nous avons remarqué que les résultats récupérées grâce à la trace sur les CLUSTER étaient différentes lorsque nous avons changé les valeurs des paramètres de SIZE et de HASHKEYS (plus on augmentait la valeur de HASHKEYS, plus la performance augmentait).

Voici les résultats retournés par le traçage en fonction des paramètres donnés :

|  |  |  |
| --- | --- | --- |
| **SIZE** | **HASHKEYS** | **LAST\_ELAPSED\_TIME** |
| 160 | 100 | 310 985 |
| 160 | 10 000 | 88 000 |
| 700 | 100 | 300 457 |
| 160 | 100 000 | 4 593 |
| 700 | 10 000 | 4 749 |

La problématique est donc devenue la suivante : quels paramètres choisir ? Nous avions 1 000 000 d’entrées avec 10 000 valeurs possibles. Nous pourrions utiliser 10 000 HASHKEYS (une pour chaque valeur possible) mais nous nous sommes retrouvé avec beaucoup trop de ligne pouvant avoir la même valeur ce qui demanderait un SIZE trop grand qui serait non supporté par la base de données (16 000). Il faudrait assumer le fait que seulement 1 à 3 % des lignes similaire pour ne pas subir un trop grand nombre de débordement. Nous avons donc décidé d’augmenter notre nombre de HAHKEYS pour compenser cette limite du SIZE. Nous avons donc décidé de continuer la suite du TP avec des valeurs que nous avons choisi en fonction de la cohérence des résultats (160 pour la SIZE et 100 000 pour HASHKEYS).

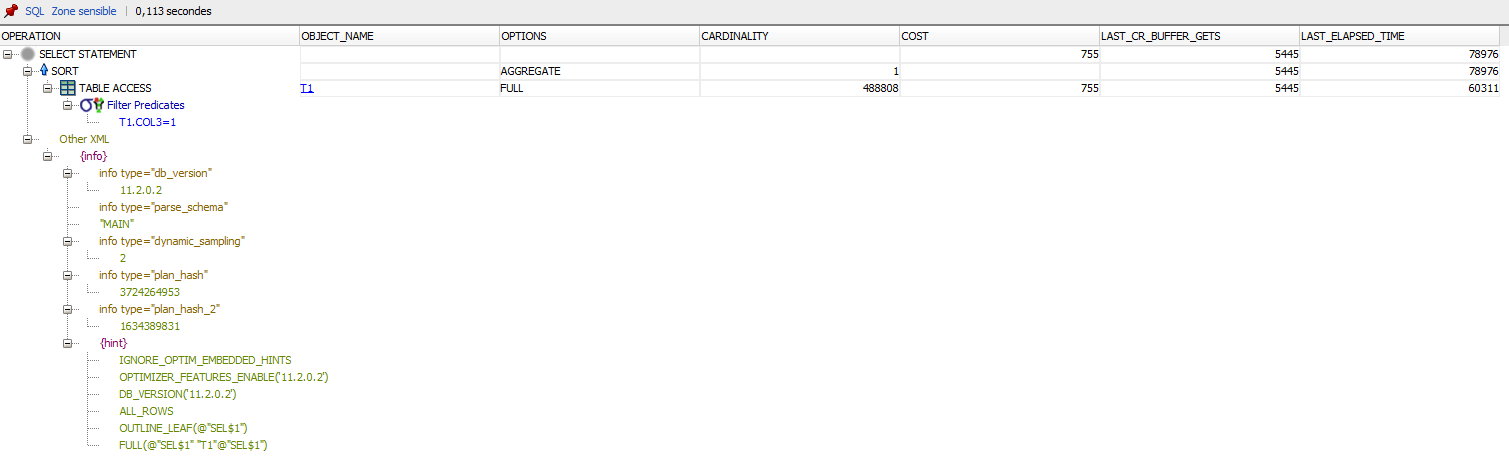
## Sélectivité grande

### Évaluation des optimisations

[Pour chacune des configurations suivantes, donnez 1) le code SQL employé pour créer les indexes ou CLUSTER requis, 2) la requête testée comprenant les indices et, possiblement, les nouvelles tables insérées dans les CLUSTER, 3) le plan d'exécution et les statistiques obtenues lors de l'exécution de la requête]

***Sans optimisation***

SELECT /\*+ FULL \*/COUNT(\*) FROM T1 WHERE T1.col3=1

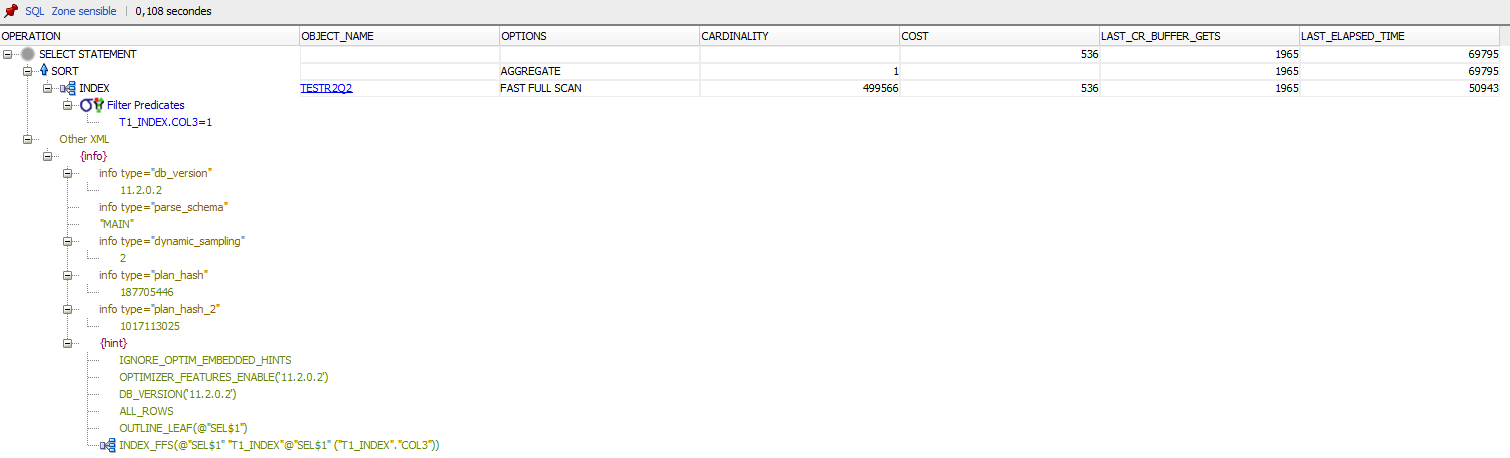


***Avec index secondaire et indice INDEX***

CREATE TABLE T1\_INDEX AS SELECT \* FROM T1;

CREATE INDEX testR2Q2 ON T1\_INDEX(col3);

SELECT /\*+ INDEX(T1\_INDEX col3)\*/ COUNT(\*) FROM T1\_INDEX WHERE T1\_INDEX.col3=1



***Avec HASH CLUSTER***

CREATE CLUSTER Grappe2 (col3 INTEGER)

SIZE 160 HASHKEYS 100000;

CREATE TABLE T1\_CLUSTER CLUSTER Grappe2(col3) AS SELECT \* FROM T1;

SELECT /\*+ HASH(T1\_CLUSTER col3)\*/ COUNT(\*) FROM T1\_CLUSTER WHERE T1\_CLUSTER.col3=1

***Avec BITMAP INDEX***

### CREATE TABLE T1\_BITMAP AS SELECT \* FROM T1;

### CREATE BITMAP INDEX bitmap1 ON T1\_BITMAP(col3);

### SELECT /\*+INDEX(T1\_BITMAP col3) \*/ COUNT(\*) FROM T1\_BITMAP WHERE T1\_BITMAP.col3 = 1;

### Analyse globale

[Faites une courte analyse (10-15 lignes) expliquant les résultats obtenus en lien avec la matière vue en classe]

Mise à part le temps passé dans l’opération du FULL, les configurations demandent beaucoup plus de temps à passer pour chaque opération comparée aux différentes requêtes effectuées dans la sélectivité petite. Effectivement, en ce qui concerne la configuration du FULL, le temps est quasiment le même pour les deux sélectivités. En ce concerne les options, on ne constate aucune différence significative de performance (cela s’explique par le faible nombre de valeur possible pour la colonne 3, ce qui n’est pas une condition avantageuse pour l’utilisation d’un INDEX ou d’un HASH). Dans cette situation, l’index bitmap pourrait permettre de créer une colonne pour chaque valeur possible (il n’existe que deux types de valeurs pour la colonne 3 de T1), les sélections peuvent être donc plus rapide que les trois configurations effectuées précédemment. Pour finir, on peut remarquer qu’il y a très peu de différence entre les résultats obtenus pour le balayage, l’indexage et le hachage.

# Opérations de jointure

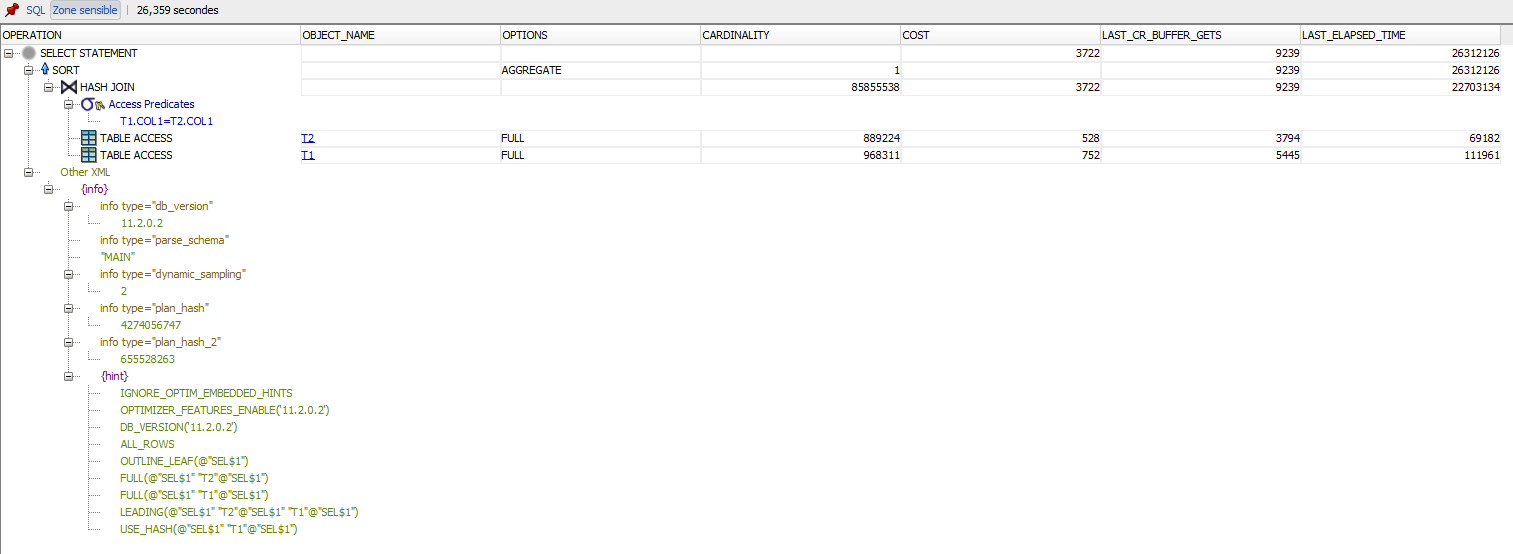
## Jointure simple

### Évaluation des optimisations

[Pour chacune des configurations suivantes, donnez 1) le code SQL employé pour créer les indexes ou CLUSTER requis, 2) la requête testée comprenant les indices et, possiblement, les nouvelles tables insérées dans les CLUSTER, 3) le plan d'exécution et les statistiques obtenues lors de l'exécution de la requête]

***Sans optimisation***

SELECT /\*+ FULL\*/ COUNT(\*) FROM T1, T2 WHERE T1.col1 = T2.col1

******

***Avec indexes secondaires et indice USE\_NL\_WITH\_INDEX ou USE\_HASH ou USE\_MERGE (pécisez lequel a été choisi)***

***Avec un HASH CLUSTER contenant les 2 tables (hétérogène)***

Après avoir attendu beaucoup de temps, il s’avère que notre requête ne se « termine » jamais. Voici tout de même le code concernant le HASH CLUSTER sans indice.

CREATE CLUSTER Grappe3 (col1 INTEGER)

SIZE 160 HASHKEYS 100;

CREATE TABLE T1\_CLUSTER CLUSTER Grappe3(col1) AS SELECT \* FROM T1;

CREATE TABLE T2\_CLUSTER CLUSTER Grappe3(col1) AS SELECT \* FROM T2;

SELECT COUNT(\*) FROM T1\_CLUSTER, T2\_CLUSTER WHERE T1\_CLUSTER.col1 = T2\_CLUSTER.col1

***Avec deux INDEX CLUSTER, chacun contenant une des tables (homogène)***

create cluster clusterT1(var1 INTEGER)

SIZE 250 INDEX;

create INDEX indexclusterT1 on cluster clusterT1;

create cluster clusterT2(var1 INTEGER)

SIZE 250 INDEX;

create INDEX indexclusterT2 on cluster clusterT2;

create table T1\_CLUSTER cluster clusterT1(col1) as select \* from T1;

create table T2\_CLUSTER cluster clusterT2(col1) as select \* from T2;

select count(\*) from T1\_CLUSTER, T2\_CLUSTER where T1\_CLUSTER.col1 = T2\_CLUSTER.col1;

### Analyse globale

[Faites une courte analyse (10-15 lignes) expliquant les résultats obtenus]

## Jointure double avec sélection

### Choix de la configuration

[Décrivez et justifiez birèvement le choix de la configuration retenue pour l'optimisation de cette requête.]

### Analyse des résultats avec différents ordres de jointure

[Faites une courte analyse (10-15 lignes) expliquant les résultats obtenus en changeant l'ordre des jointures]

Note :

An index fast full scan reads the ENTIRE index, unsorted, as it exists on disk. It is basically using the index as a "skinny" version of the table. The query in question would only be accessing attributes in the index (we are not using the index as a way to get to the table, we are using the index INSTEAD of the table) We use multiblock IO and read all of the leaf, branch and the root block. We ignore the branch and root blocks and just process the (unordered) data on the leaf blocks.  
  
An index full scan is when we read the index a block at a time - from start to finish. We'll read the root block, navigate down the left hand side of the index (or right if we are doing a descending full scan) and then when we hit the leaf block - we'll read across the entire bottom of the index - a block at a time - in sorted order. We use single block IO, not multiblock IO for this operation.