

COMPTE RENDU BDA

TP06

Plan d'exécution des
requêtes «explain
plan»

Sommaire

Table des matières

1-Introduction.....	3
2-Les requêtes.....	4
1) Requête avec sélection et projection.....	4
2) Requête avec jointure.....	8
3-Conclusion :.....	16

1-Introduction

Le sujet de ce TP concerne **l'étude de la façon dont Oracle Sql developer traite les requêtes.**

L'objectif est de **générer et interpréter les arbres logiques et physiques de différentes requêtes.**

Pour ce faire nous avons utilisé un logiciel : *Oracle SQL Developer.*

2-Les requêtes

Ce TP vise à comprendre le fonctionnement interne du calcul des requêtes LID. Le SGBD construit un plan d'exécution des requêtes optimal afin de calculer cette dernière le plus rapidement possible.

1) Requête avec sélection et projection

Oracle permet la consultation du plan d'exécution (explain plan) des requêtes. Pour ce faire, il convient d'interroger la table PLAN_TABLE après avoir soumis la requête à l'explain plan.

Voici les commandes :

1. // suppression de l'ancien plan d'exécution

```
DELETE FROM plan_table;
```

2. // construction du plan d'exécution

```
EXPLAIN PLAN
```

```
SET statement_id = 'Q1'
```

```
FOR SELECT numCli, nom, prenom FROM Client WHERE Pays = 'FRA' AND DateN >  
TO_DATE('01/01/1999','DD/MM/YYYY');
```

3. // affichage du plan d'exécution

```
SELECT id, parent_id, operation, options, object_name, filter_predicates, access_predicates,  
projection
```

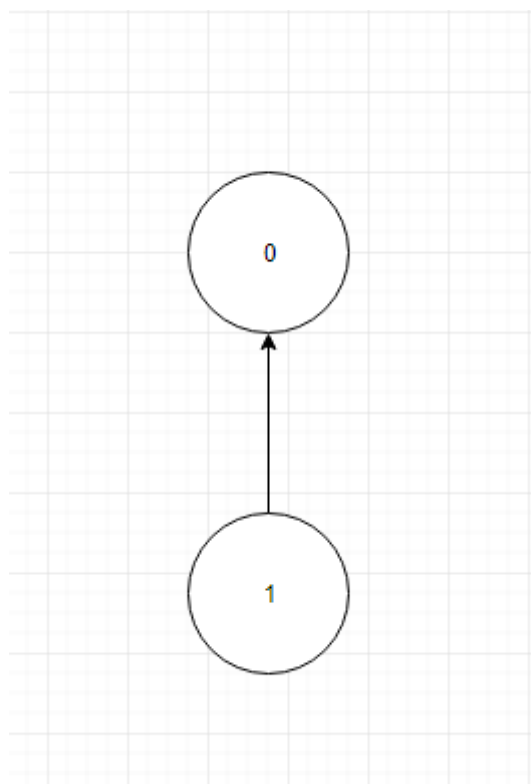
```
FROM PLAN_TABLE
```

```
WHERE statement_id = 'Q1';
```

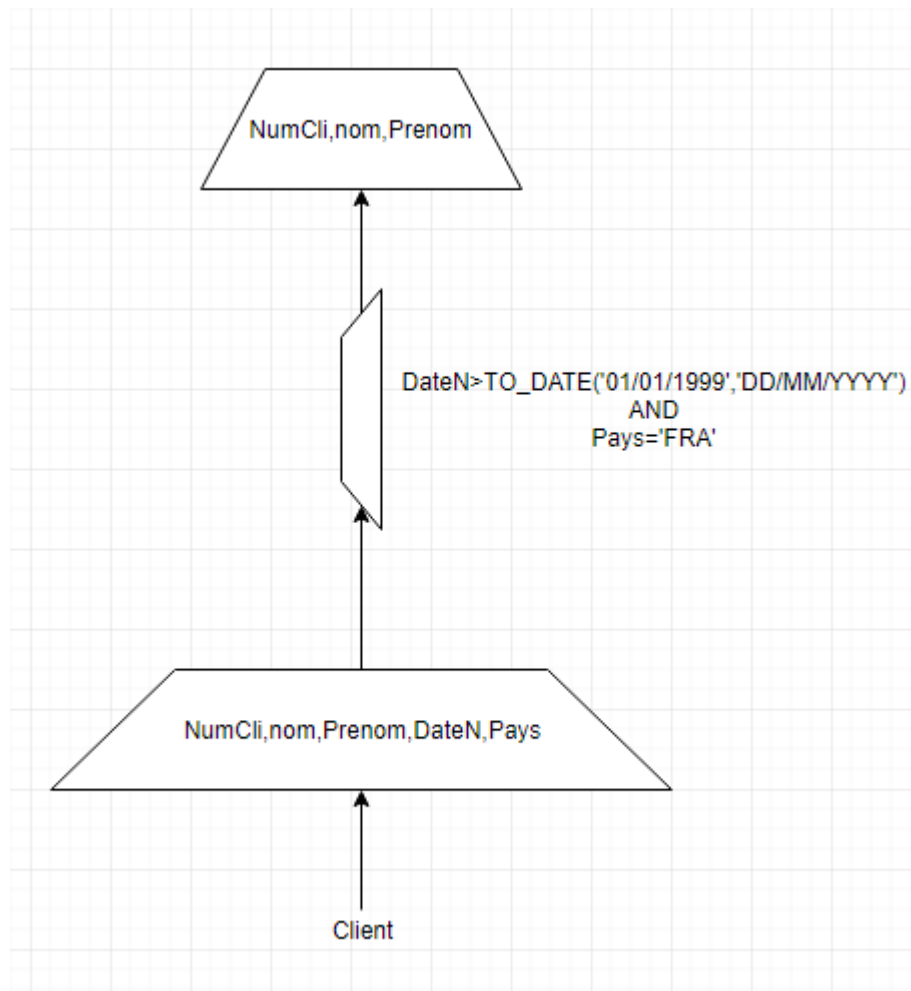
1. Donnez l'arbre algébrique physique issu de l'explain plan d'Oracle.

ID	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME	FILTER
1	0	SELECT STATEMENT	(null)	(null)	(null)
2	1	TABLE ACCESS	FULL	CLIENT	'DATE'

Arbre physique :



2. Représentez l'arbre algébrique logique optimal qui lui correspond en utilisant les notations étudiées en cours (algèbre relationnelle).



3. Reprenez la requête précédente sous SQL Developer directement.

SELECT numCli, nom, prenom

FROM Client

WHERE Pays = 'FRA' AND DateN > TO_DATE('01/01/1999','DD/MM/YYYY');

The screenshot shows the SQL Developer interface. The top pane displays the following SQL query:

```
SELECT numCli, nom, prenom
FROM Client
WHERE Pays = 'FRA'
AND DateN > TO_DATE('01/01/1999','DD/MM/YYYY');
```

The bottom pane shows the execution plan for the query. The top bar indicates the execution time is 8,229 secondes. The execution plan table is as follows:

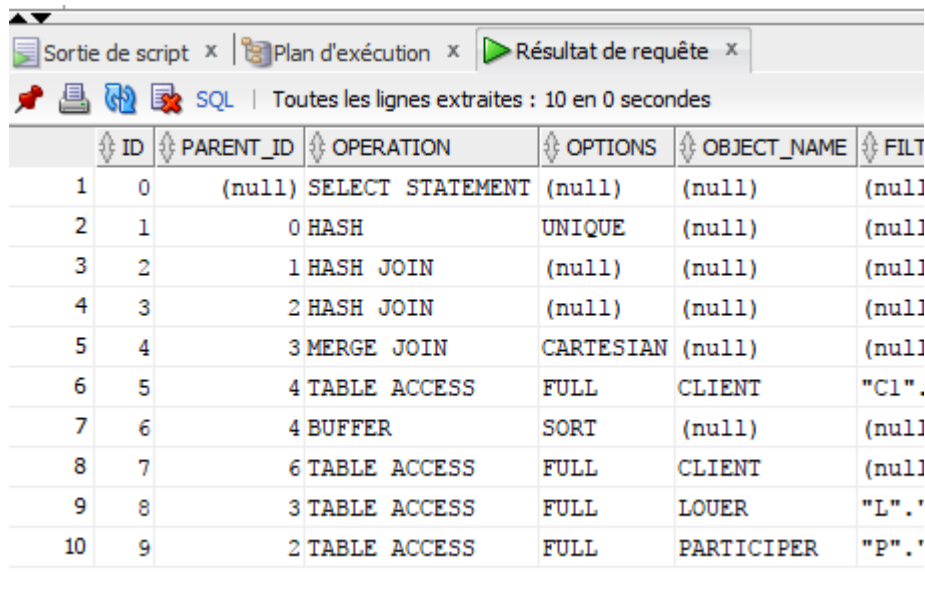
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	CC
SELECT STATEMENT				81
TABLE ACCESS	CLIENT	FULL		81

Below the table, a tree view shows the query structure: SELECT STATEMENT, TABLE ACCESS, Filter Predicates, AND, DATEN > TO_DATE('1999-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss'), PAYS='FRA', and Other XML.

2) Requête avec jointure

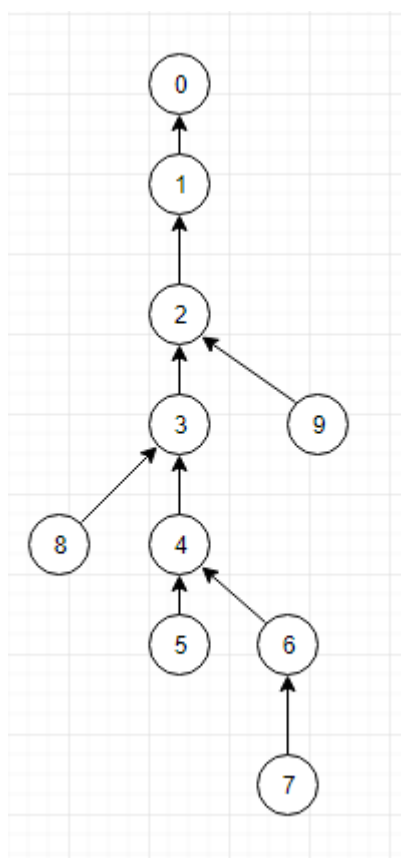
1. Soit la requête qui calcule la liste des clients (nom, prenom) présents au camping durant la haute saison, pour les locations effectuées par Morrel Cristina. Donnez l'arbre algébrique physique issu de l'explain plan d'Oracle.

```
SELECT DISTINCT C2.nom, C2.prenom
FROM Client C1, Client C2, Louer L, Participer P
WHERE C1.numCli = L.numCli
AND C1.numCli = P.numCli
AND C2.numCli = P.numCliP
AND L.numCli = P.numCli
AND L.numLoc = P.numLoc
AND L.dateDeb = P.dateDeb
AND C1.nom = 'Morrell'
AND C1.prenom = 'Cristina'
AND P.dateDeb BETWEEN TO_DATE('14/07/2017','DD/MM/YYYY')
AND TO_DATE('15/08/2017','DD/MM/YYYY'))
```



ID	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME	FILTER
1	0	(null) SELECT STATEMENT	(null)	(null)	(null)
2	1	0 HASH	UNIQUE	(null)	(null)
3	2	1 HASH JOIN	(null)	(null)	(null)
4	3	2 HASH JOIN	(null)	(null)	(null)
5	4	3 MERGE JOIN	CARTESIAN	(null)	(null)
6	5	4 TABLE ACCESS	FULL	CLIENT	"C1".
7	6	4 BUFFER	SORT	(null)	(null)
8	7	6 TABLE ACCESS	FULL	CLIENT	(null)
9	8	3 TABLE ACCESS	FULL	LOUER	"L".
10	9	2 TABLE ACCESS	FULL	PARTICIPER	"P".

Arbre physique :

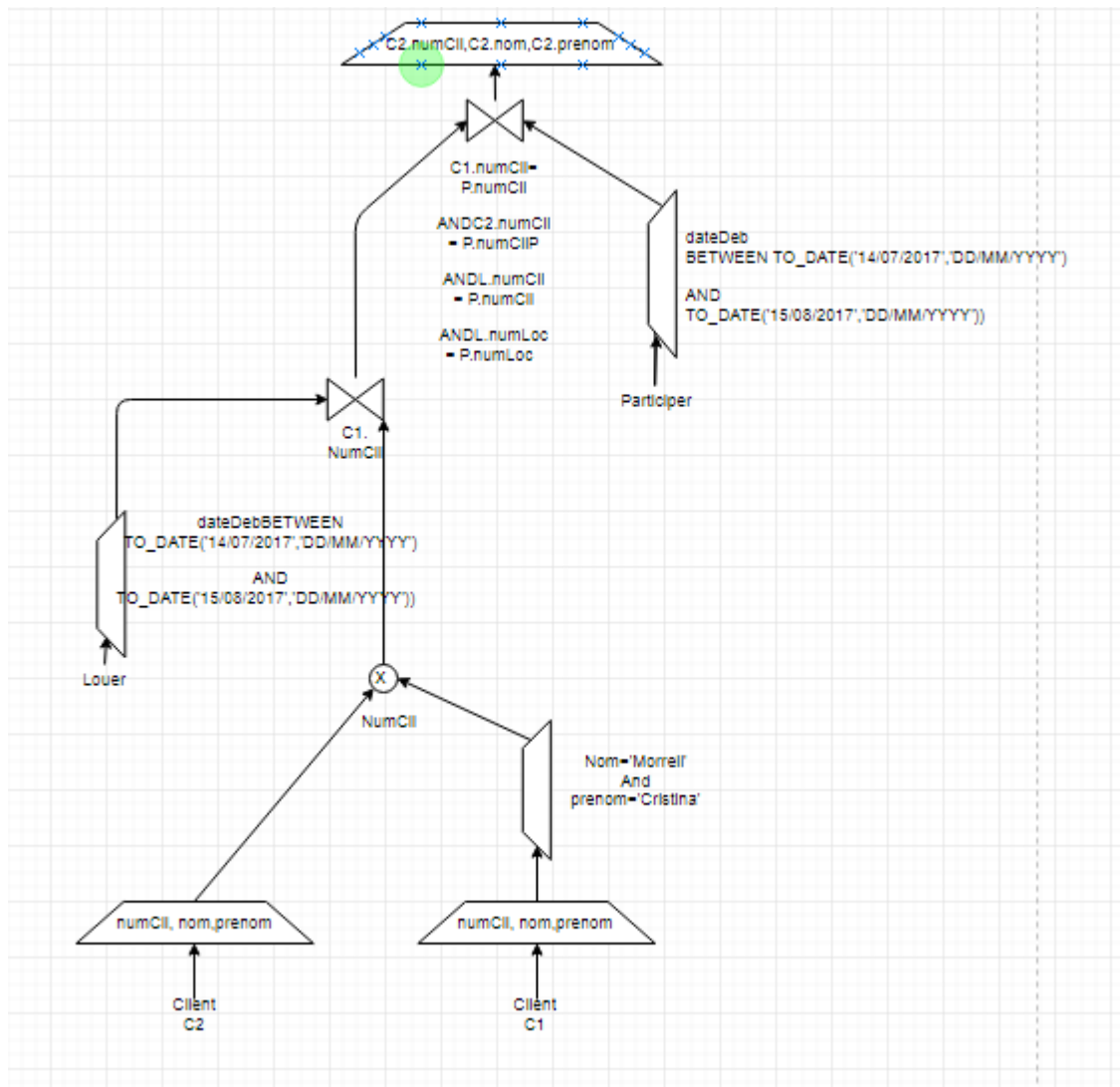


2. Relevez le coût théorique de ce plan d'exécution.

Sortie de script x Plan d'exécution x Résultat de requête x			
SQL 0,375 secondes			
OPTIONS	CARDINALITY	COST	
		1	41
UNIQUE		1	41
		1	40

Coût théorique : 41

3. Représentez l'arbre algébrique logique optimal qui lui correspond en utilisant les notations étudiées en cours (algèbre relationnelle).



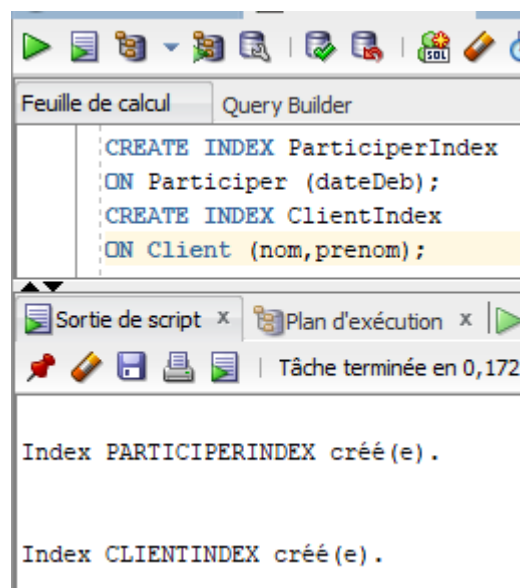
4. Modifiez votre vue afin d'ajouter les jours en tant que participant à une location (dans PARTICIPER).

Indication. Rappelons qu'un UNIQUE INDEX est automatiquement créé avec les contraintes de clés primaires. Les commandes pour ajouter des index sur d'autres attributs sont les suivantes :

```
CREATE UNIQUE INDEX nomIndex ON nomTable (ListeAttr);
```

```
CREATE INDEX nomIndex ON nomTable (ListeAttr);
```

- Sur Participer (dateDeb)
- Sur Client (nom, prenom)



5. Vérifiez la présence des index (USER_INDEXES).

Les index de manière générale permettent un accès plus rapide aux tables. Il s'agit au niveau physique de fichiers annexes ajoutés et utilisés par le système pour accéder plus rapidement aux données. Leur présence associée aux tables influence donc l'optimiseur d'Oracle qui établit des plans d'exécution différents.

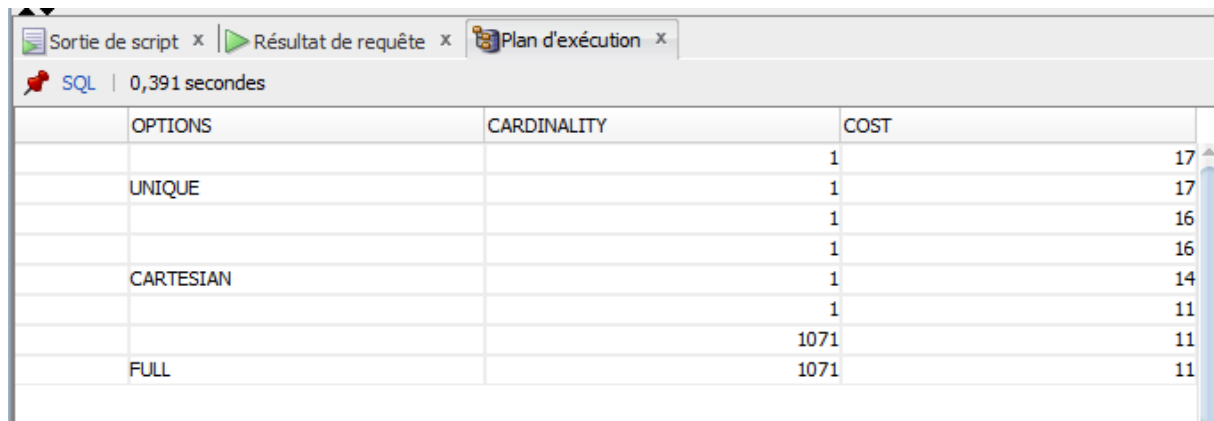
The screenshot shows the Oracle SQL Developer interface with the 'Query Builder' tab. The SQL editor contains the query:

```
select * from USER_INDEXES;
```

The 'Résultat de requête' (Query Result) window displays the results of the query in a table format. The table has 8 columns: INDEX_NAME, INDEX_TYPE, TABLE_OWNER, TABLE_NAME, TABLE_TYPE, UNIQUENESS, and COMPRESSION. There are 2 rows of data.

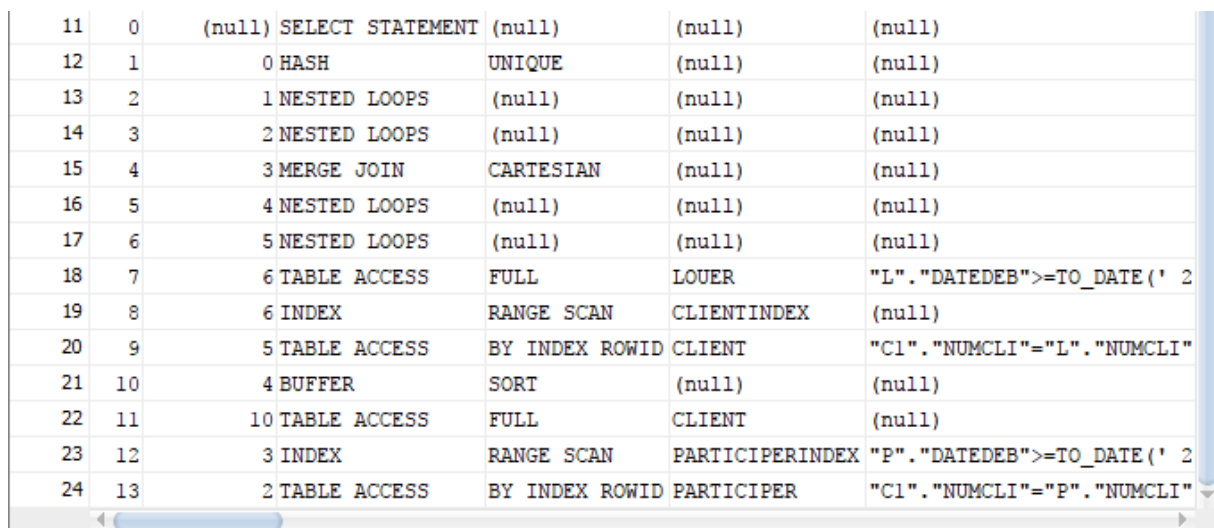
	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION
1	PARTICIPERINDEX	NORMAL	IUTB2039	PARTICIPER	TABLE	NONUNIQUE	DISABLED
2	CLIENTINDEX	NORMAL	IUTB2039	CLIENT	TABLE	NONUNIQUE	DISABLED

6. Affichez à nouveau le plan d'exécution de la requête en présence de ces nouveaux index. Que constatez-vous au niveau du coût de la requête (COST) ? Que constatez-vous au niveau du l'arbre algébrique ?



OPTIONS	CARDINALITY	COST
		17
UNIQUE		17
		16
		16
CARTESIAN		14
		11
	1071	11
FULL	1071	11

Coût théorique : 17, le coût a donc beaucoup baissé.



Step	Cost	Operation	Options	Cardinality	Cost
11	0	(null) SELECT STATEMENT	(null)	(null)	(null)
12	1	0 HASH	UNIQUE	(null)	(null)
13	2	1 NESTED LOOPS	(null)	(null)	(null)
14	3	2 NESTED LOOPS	(null)	(null)	(null)
15	4	3 MERGE JOIN	CARTESIAN	(null)	(null)
16	5	4 NESTED LOOPS	(null)	(null)	(null)
17	6	5 NESTED LOOPS	(null)	(null)	(null)
18	7	6 TABLE ACCESS	FULL LOUER	"L"."DATEDEB">=TO_DATE(' 2	
19	8	6 INDEX	RANGE SCAN CLIENTINDEX	(null)	
20	9	5 TABLE ACCESS	BY INDEX ROWID CLIENT	"C1"."NUMCLI"="L"."NUMCLI"	
21	10	4 BUFFER	SORT	(null)	(null)
22	11	10 TABLE ACCESS	FULL CLIENT	(null)	
23	12	3 INDEX	RANGE SCAN PARTICIPERINDEX	"P"."DATEDEB">=TO_DATE(' 2	
24	13	2 TABLE ACCESS	BY INDEX ROWID PARTICIPER	"C1"."NUMCLI"="P"."NUMCLI"	

On remarque qu'il y a plus d'étape, néanmoins la requête utilise les index et des *nested Loops* qui sont des jointures moins coûteuses.

7. Ajoutez les index multiples sur les attributs clés étrangères de la requête

- Sur Participer (numCliP)
- Sur Louer (numCli)
- Sur Participer (numCli,numLoc,dateDeb)

```
CREATE INDEX ParticiperIndexNumcliP
```

```
ON Participer (numCliP);
```

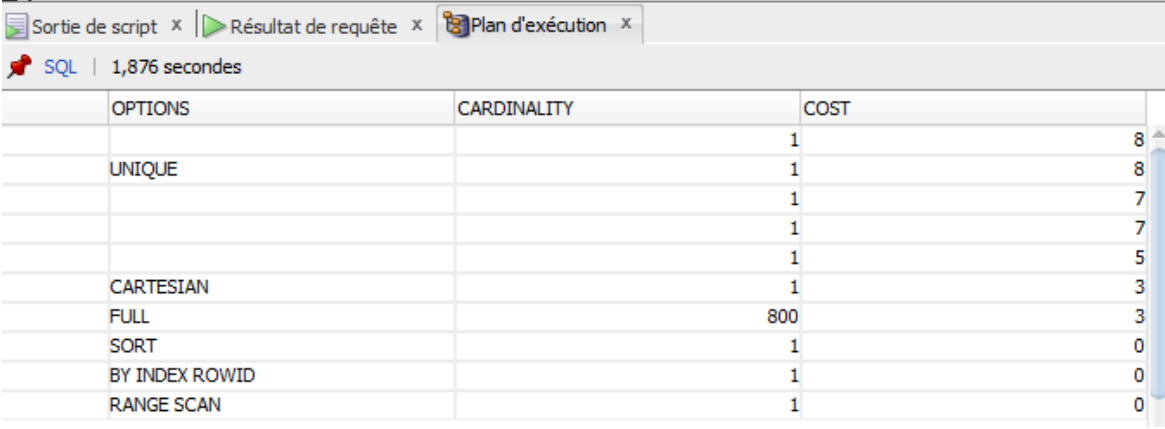
```
CREATE INDEX LouerIndexNumcli
```

```
ON Louer (numCli);
```

```
CREATE INDEX ParticiperIndexLouer
```

```
ON Participer (numCli,numLoc,dateDeb);
```

8. Affichez à nouveau le plan d'exécution de la requête en présence de ces nouveaux index. Que constatez-vous au niveau du coût de la requête (COST) ? Que constatez-vous au niveau du l'arbre algébrique ?



The screenshot shows a window titled 'Plan d'exécution' with a sub-header 'SQL | 1,876 secondes'. It displays a table with three columns: 'OPTIONS', 'CARDINALITY', and 'COST'. The table lists various execution options and their associated costs.

OPTIONS	CARDINALITY	COST
	1	8
UNIQUE	1	8
	1	7
	1	7
	1	5
CARTESIAN	1	3
FULL	800	3
SORT	1	0
BY INDEX ROWID	1	0
RANGE SCAN	1	0

Coût théorique : 8, il a encore baissé

Sortie de script x Plan d'exécution x Résultat de requête x						
SQL Toutes les lignes extraites : 14 en 0 secondes						
ID	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME	FILTER_PREDICATES	
1	0	(null) SELECT STATEMENT	(null)	(null)	(null)	
2	1	0 HASH	UNIQUE	(null)	(null)	
3	2	1 NESTED LOOPS	(null)	(null)	(null)	
4	3	2 NESTED LOOPS	(null)	(null)	(null)	
5	4	3 NESTED LOOPS	(null)	(null)	(null)	
6	5	4 MERGE JOIN	CARTESIAN	(null)	(null)	
7	6	5 TABLE ACCESS	FULL	CLIENT	(null)	
8	7	5 BUFFER	SORT	(null)	(null)	
9	8	7 TABLE ACCESS	BY INDEX ROWID	CLIENT	(null)	
10	9	8 INDEX	RANGE SCAN	CLIENTINDEX	(null)	
11	10	4 TABLE ACCESS	BY INDEX ROWID	PARTICIPER	"C2"."NUMCLI"="P"."NU	
12	11	10 INDEX	RANGE SCAN	PARTICIPERINDEXLOUER	"P"."DATEDEB">=TO_DAT	
13	12	3 INDEX	RANGE SCAN	LOUERINDEXNUMCLI	"C1"."NUMCLI"="L"."NU	
14	13	2 TABLE ACCESS	BY INDEX ROWID	LOUER	"L"."DATEDEB">=TO_DAT	

On remarque qu'il y a autant d'étape, néanmoins la requête utilise plus d'index et effectue moins de jointures.

9. Ajoutez les clés primaires (et donc des index uniques) sur les attributs de la requête

– Sur (numCli)

– Sur (numCli,numLoc,dateDeb)

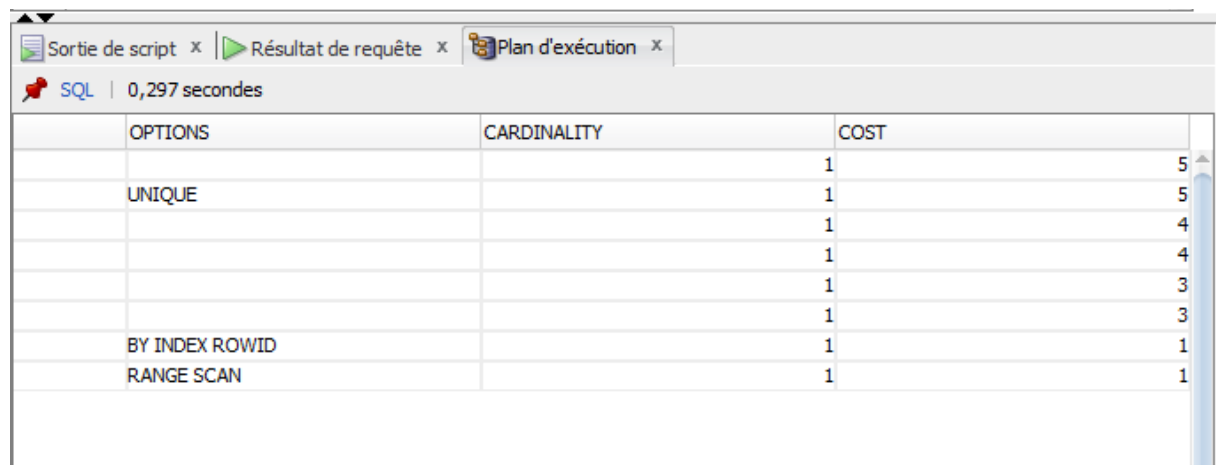
CREATE UNIQUE INDEX numCli

ON CLIENT (numCli);

CREATE UNIQUE INDEX Louer

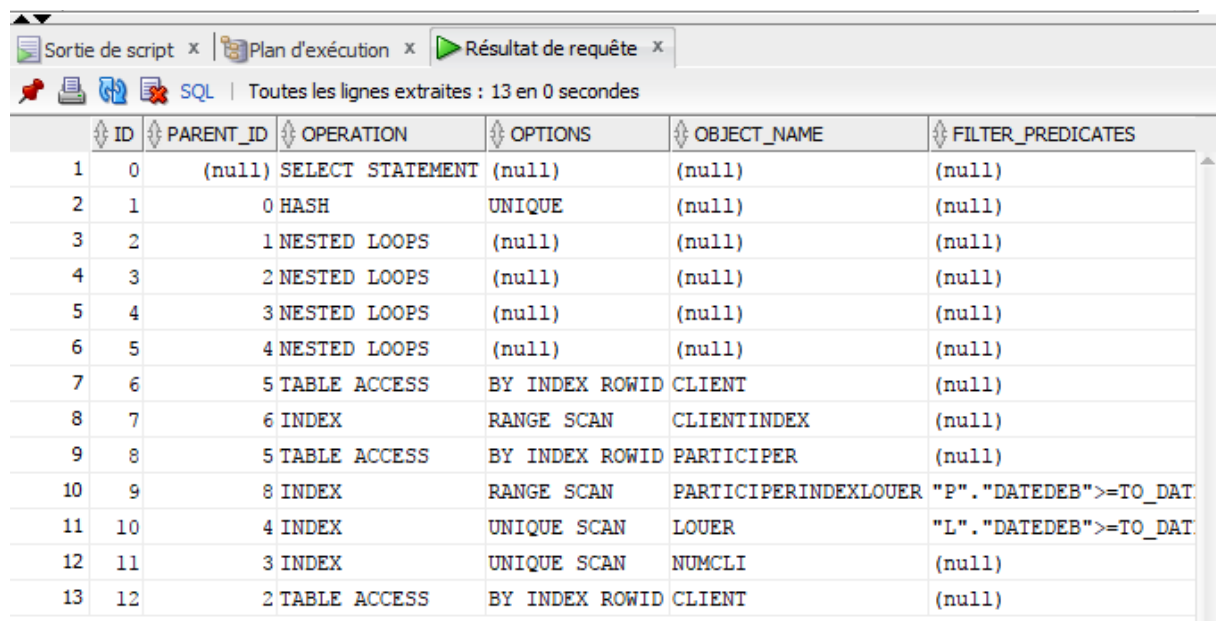
ON LOUER (numCli,numLoc,dateDeb);

10. Affichez à nouveau le plan d'exécution de la requête en présence de ces nouveaux index. Que constatez-vous au niveau du coût de la requête (COST) ? Que constatez-vous au niveau du l'arbre algébrique ?



OPTIONS	CARDINALITY	COST
	1	5
UNIQUE	1	5
	1	4
	1	4
	1	3
	1	3
BY INDEX ROWID	1	1
RANGE SCAN	1	1

Coût théorique : 5, il a encore baissé



ID	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME	FILTER_PREDICATES
1	0	SELECT STATEMENT	(null)	(null)	(null)
2	1	HASH	UNIQUE	(null)	(null)
3	2	1 NESTED LOOPS	(null)	(null)	(null)
4	3	2 NESTED LOOPS	(null)	(null)	(null)
5	4	3 NESTED LOOPS	(null)	(null)	(null)
6	5	4 NESTED LOOPS	(null)	(null)	(null)
7	6	5 TABLE ACCESS	BY INDEX ROWID	CLIENT	(null)
8	7	6 INDEX	RANGE SCAN	CLIENTINDEX	(null)
9	8	5 TABLE ACCESS	BY INDEX ROWID	PARTICIPER	(null)
10	9	8 INDEX	RANGE SCAN	PARTICIPERINDEXLOUER	"P"."DATEDEB">=TO_DAT
11	10	4 INDEX	UNIQUE SCAN	LOUER	"L"."DATEDEB">=TO_DAT
12	11	3 INDEX	UNIQUE SCAN	NUMCLI	(null)
13	12	2 TABLE ACCESS	BY INDEX ROWID	CLIENT	(null)

On remarque qu'il y a une étape en moins, la requête utilise uniquement des index, des *tables access* et *nested Loops* et Oracle n'effectue plus de jointures coûteuses.

3-Conclusion :

A l'issue de ce TP, nous savons **afficher l'arbre algébrique d'une requête, l'analyser et interpréter son coût et l'améliorer (le réduire) grâce aux index.**