HAI901I : Optimisation de requêtes

I.Mougenot

FDS UM

2023

Optimisation et performance

Optimisation : capacité à adapter l'architecture d'un système aux différents besoins des applications

Performance : qualité d'un système à répondre aux exigences des usagers. Nécessite de prévoir et d'anticiper :

- demandes (requêtes) à venir
- délais à tenir
- nombre d'usagers et de transactions
- volumétrie des données
- ...

Ressources cibles

- processeurs
- mémoires caches
- disques (mémoires secondaires)
- processus (avant et arrière plans)
- réseau et bande passante (capacité de faire transiter les données entre le serveur et le client)
- ...

Eléments à considérer pour un SGBDR

- temps d'accès à un enregitrement physique
- écriture des calculs relationnels

1. méthodes d'accès aux données (ex. usage caches et index)
2. recours aux clusters (jointures physiques entre tables au travers d'un segment spécifique)
3. optimisation organisation physique des fichiers (taille allouée aux fichiers, taille du bloc, ...)
4. optimisation organisation logique de la base de données (choix tablespaces, taille des extensions, ...)
5. optimisation des requêtes

Paramètres associés à une requête

- chemins d'accès aux données
- opérations nécessaires à l'expression de la requête (opérateurs sous-jacents)
- enchaînement des opérations sous forme de chemin d'exécution (path)

Décortiquer une requête

1. à l'origine : déclarative (l'ordre des opérations n'est pas indiqué)
2. analyse, simplification, réécriture : arbre algébrique d'opérations
3. transformation et choix : plan d'exécution physique (une déclinaison procédurale de la requête)

Décortiquer une requête : illustration

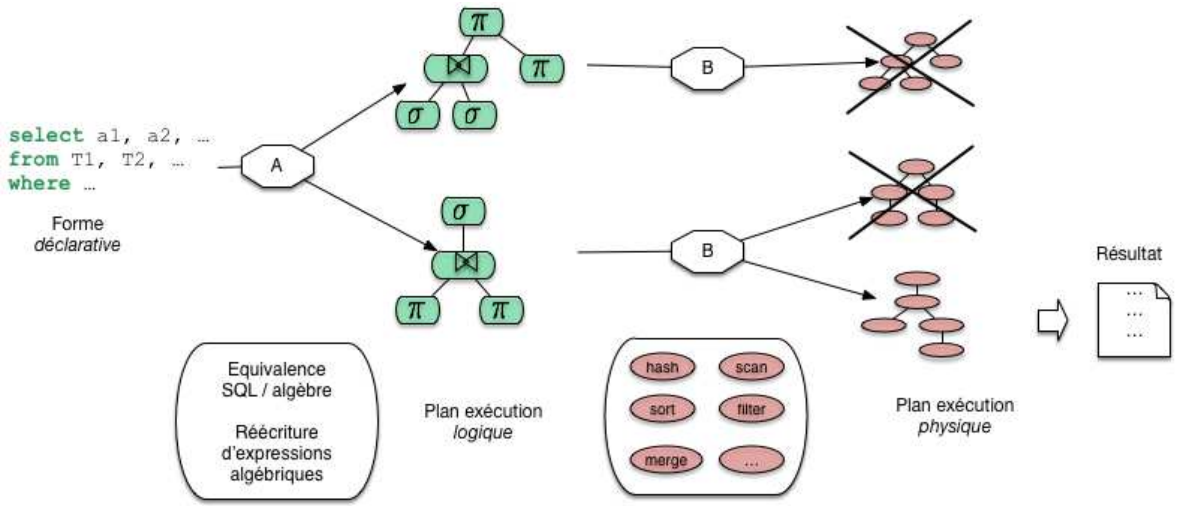


Figure: Grandes étapes : Fig. Extraite de la page `http://sys.bdpedia.fr/optim.html`

Plan d'exécution

= arbre contenant des nœuds et des liens orientés

1. nœuds = opérateurs physiques
2. liens : données en entrée et en sortie

Plans équivalents = partent des mêmes données pour produire les mêmes résultats

Les principaux opérateurs physiques

1. intersection et concaténation : respectivement intersection et union sur un ensemble de tuples
2. filter et projection : respectivement sélection et projection
3. nested loop (boucles imbriquées), sort/merge (tri/fusion), hash join : trois opérateurs pour le calcul de la jointure (pouvant utiliser ou non un index)

Retour vers l'algèbre relationnelle

1. exploiter les formes algébriques (notamment arborescentes) pour décider des écritures les plus optimisées
2. opérateurs algébriques spécifiques (sélection, projection, jointure, ...) et issus de la théorie des ensembles (union, intesection, différence, produit cartésien, différence, ...)
3. propriétés des opérateurs : commutativité (par exemple de la jointure ou du produit cartésien), associativité (par exemple de la jointure ou de l'intersection), distributivité (par exemple de la projection sur l'union)

Equivalences algébriques (exemples)

Soit trois relations (Emp, Dept et Fonction) :

$\bowtie$ commutative : Dept $\bowtie$ Emp = Emp $\bowtie$ Dept

$\bowtie$ associative : (Dept $\bowtie$ Emp) $\bowtie$ Fonction = Dept $\bowtie$ (Emp $\bowtie$ Fonction)

Soit deux relations (Etudiant et Enseignant) :

$\cup$ commutative : Etudiant $\cup$ Enseignant = Enseignant $\cup$ Etudiant

Distributivité de Π sur $\cup$: Π nom, prenom (Etudiant $\cup$ Enseignant) = Π nom, prenom (Etudiant) $\cup$ Π nom, prenom (Enseignant)

Exemples d'arbres algébriques équivalents

Trois plans d'illustration

Donner le numéro et le nom des employés qui travaillent dans un département localisé à Montpellier

SELECT numE, nomE from Dept D join Emp E on D.n_dept = E.n_dept where localisation = 'Montpellier' ;

Exemples d'arbres algébriques équivalents

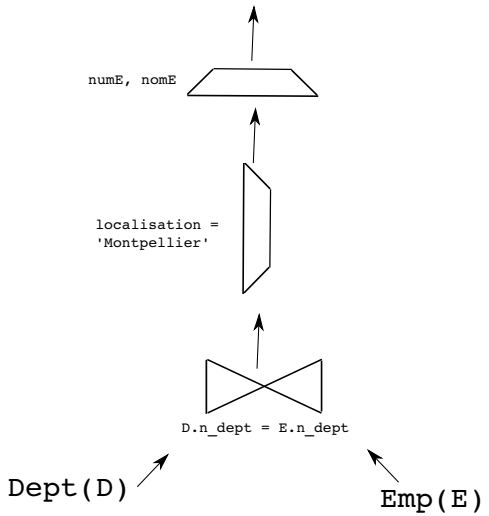


Figure: plan 1

Filtrer dès que possible

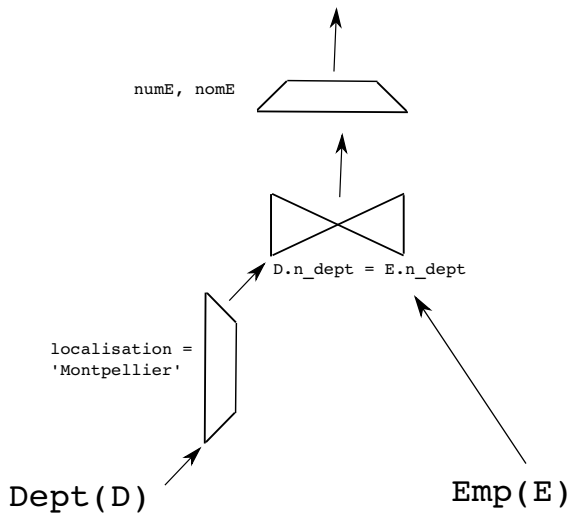


Figure: plan 2 meilleur que plan 1

Filtrer dès que possible

$$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie S$$

** *if* c *contains only attributes in* R

$$= R \bowtie \sigma_c(S)$$

** *if* c *contains only attributes in* S

if c *contains attributes from BOTH* R *and* S
we cannot push σ_c *further inside.*

Figure: Réécriture de l'ordre

Commencer par la relation de plus faible cardinalité

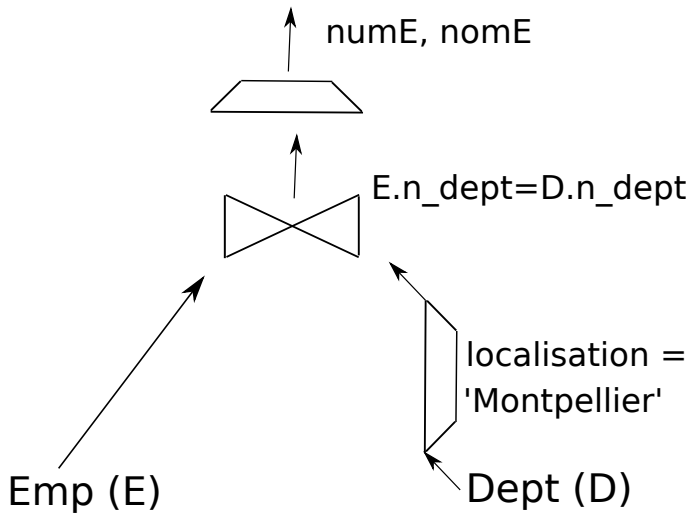


Figure: plan 3 moins bon que plan 2 car card(Emp) $>$ Card(Dept)

Autres formes déclaratives équivalentes

Donner les numéro et nom des employés travaillant dans un département localisé à Montpellier = semi-jointure

```
SELECT numE, nomE from Emp where n_dept =
(select n_dept from Dept where localisation =
'Montpellier') ;
```

```
SELECT numE, nomE from Emp E where exists
(select * from Dept D where D.n_dept = E.n_dept
and localisation = 'Montpellier') ;
```

Optimiseur Oracle

Principaux rôles

- trouver les expressions équivalentes au moyen de ré-écritures
- évaluer les expressions et en choisir la meilleure
- le choix se fait à partir de différents critères
- l'optimisation peut privilégier le débit (traiter tous les tuples à la fois) ou le temps de réponse (pour un sous-ensemble de tuples donné)

Optimiseur d'Oracle

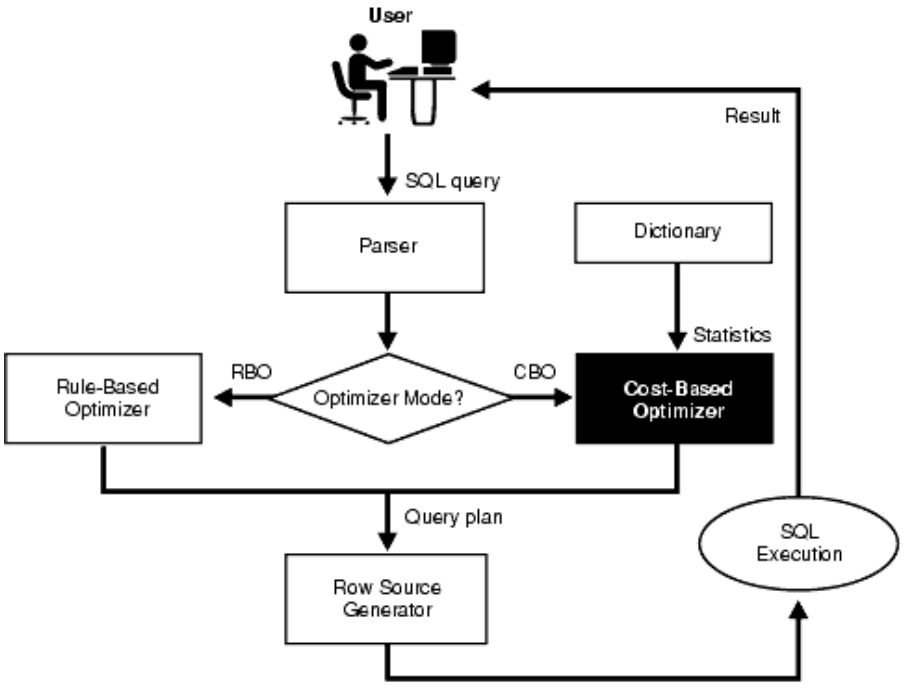


Figure: Grandes activités associées (figure documentation Oracle)

Objectifs ciblés : débit ou temps de reponse

1. all_rows : valeur par défaut qui privilégie le débit
2. first_rows_1 : privilégie le temps de réponse pour premier tuple
3. first_rows_10 : . . . pour les dix premiers tuples
4. first_rows_100 : . . . pour les cent premiers tuples
5. first_rows_1000 : . . . pour les mille premiers tuples

Un exemple de manipulation :

```
alter session set optimizer_mode=FIRST_ROWS_10;
```

Orienter l'optimiseur :

1. vers le coût (représenté par les unités de travail et/ou les ressources exploitées (coût CPU et I/O)) => "le plan le plus économe est le le meilleur"
2. vers la performance (représentée par le temps écoulé) => "le plan le plus rapide est le le meilleur"

Critères aidant à l'évaluation

En dehors des statistiques collectées

- schéma logique de la base de données : description tables, attributs, contraintes
- schéma physique de la base de données : index, taille des blocs, chemin d'accès aux données
- algorithmes disponibles supportant les opérations algébriques
- particularités de l'architecture du système : parallélisation par exemple

Optimiseur d'Oracle

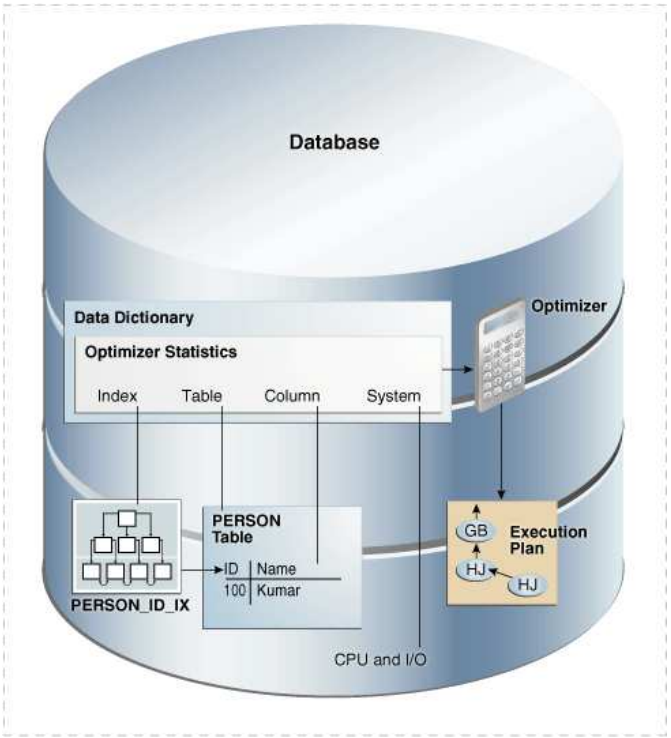


Figure: Schéma général (figure documentation Oracle)

Critères aidant à l'évaluation

Du côté des statistiques collectées

- par table : cardinalité de chaque relation, taille moyenne des tuples, nombre d'enregistrements
- par bloc : (blocking factor), nombre de blocs nécessaires pour le stockage de la table
- par opérateur : coût de l'exécution de l'algorithme de chaque opérateur

Critères aidant à l'évaluation

Du côté des statistiques collectées

- par attribut : type de données et précision, distribution des valeurs (nombre de valeurs distinctes et probabilité de la distribution), sélectivité (prend ses valeurs entre 0 et 1 et est calculée comme étant 1/nombre de valeurs distinctes pour un attribut donné).
  Plus la valeur de la sélectivité tend vers 0 et plus c'est sélectif

Pour la sélectivité

- La sélectivité est accessible au travers de l'attribut density dans la vue user_tab_columns. L'optimiseur exploite density lorsque des statistiques sur la base d'histogrammes plus précis pour évaluer la distribution des valeurs ne sont pas disponibles. Un exemple pour l'attribut NOMCOMMAJ de la table COMMUNE :

```
select density from user_tab_columns where table_name
= 'COMMUNE' and column_name = 'NOMCOMMAJ';
```

Autres exemples de critères aidant à l'évaluation

Du côté des statistiques collectées

- par index : type d'index, dense/creux, blocs en cache (o/n), index plaçant sur la clé primaire (o/n)
- général : nombre de blocs en mémoire vive
- penser à rafraîchir les statistiques :

```
exec dbms_utility.analyze_schema(user,'COMPUTE')
```

Optimiseur : vérifie aussi si requête déjà jouée

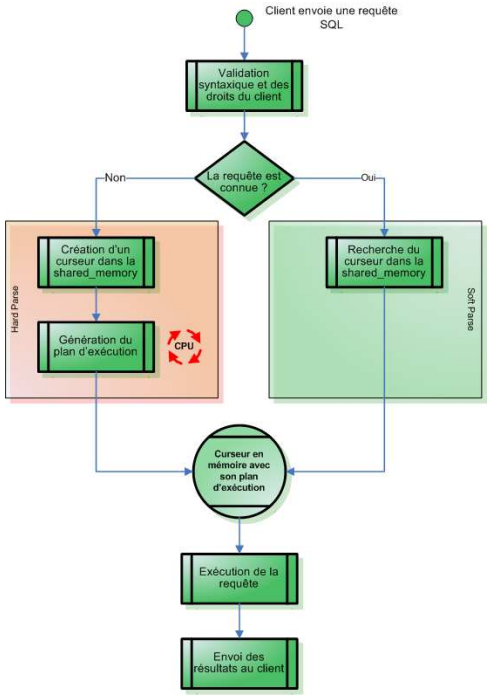


Figure: Rappel Hard Parse / Soft Parse

Outil Explain

Estimer la performance d'un plan d'exécution

- expliciter un plan avec un exemple :

```
explain plan for select * from emp where num=33000 ;
```

- exploiter le paquetage dbms_xplan `select`

```
plan_table_output from table(dbms_xplan.display()) ;
```

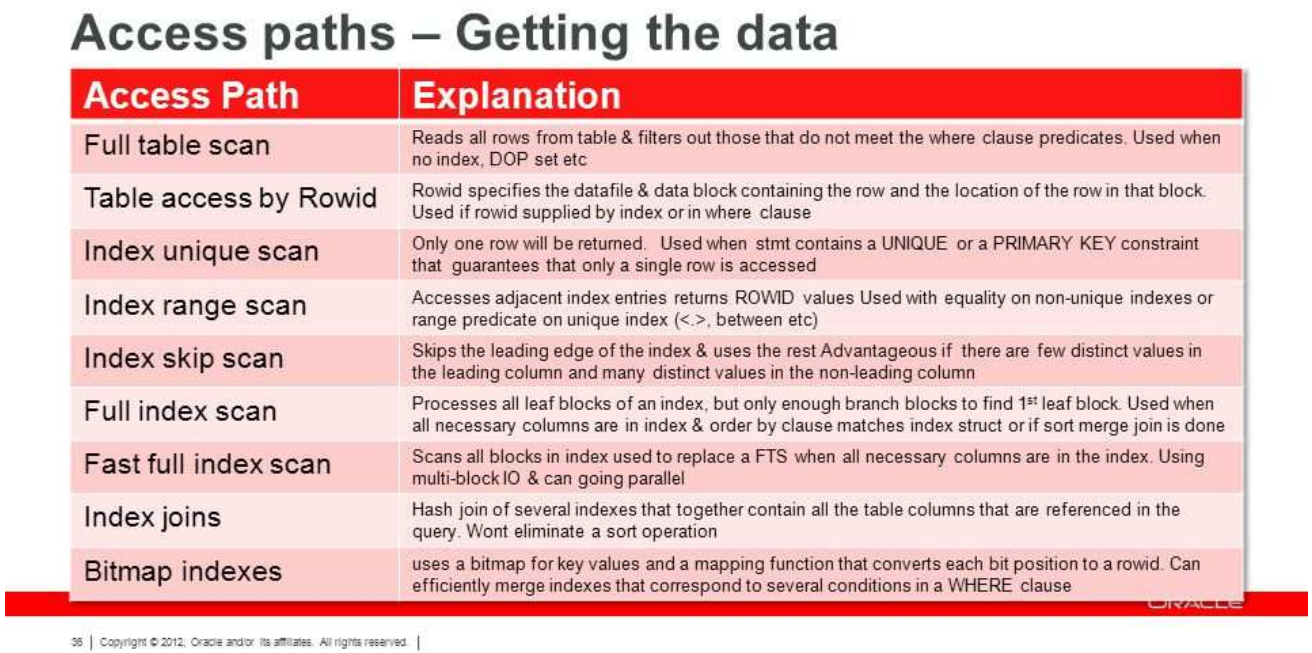
- exploiter plan_table `col optimizer for a30`

```
select plan_id, cpu_cost, io_cost, optimizer,
operation, time from plan_table;
```

Influencer le plan d'exécution au travers de directives (hints)

```
explain plan for select /*+ full(f) */ * from fonction f
where nom_f like 'd%';

explain plan for select /*+ ordered */ *
from emp e, dept d where e.n_dept=d.n_dept;

explain plan for select /*+ ordered use_nl(e d) */ *
from emp e, dept d where e.n_dept=d.n_dept;

explain plan for select /*+ ordered use_merge(e d) */ *
from emp e, dept d where e.n_dept=d.n_dept;
```

Chemins d'accès aux données



Access paths – Getting the data

Access Path	Explanation
Full table scan	Reads all rows from table & filters out those that do not meet the where clause predicates. Used when no index, DOP set etc
Table access by Rowid	Rowid specifies the datafile & data block containing the row and the location of the row in that block. Used if rowid supplied by index or in where clause
Index unique scan	Only one row will be returned. Used when stmt contains a UNIQUE or a PRIMARY KEY constraint that guarantees that only a single row is accessed
Index range scan	Accesses adjacent index entries returns ROWID values Used with equality on non-unique indexes or range predicate on unique index (<,>, between etc)
Index skip scan	Skips the leading edge of the index & uses the rest Advantageous if there are few distinct values in the leading column and many distinct values in the non-leading column
Full index scan	Processes all leaf blocks of an index, but only enough branch blocks to find 1st leaf block. Used when all necessary columns are in index & order by clause matches index struct or if sort merge join is done
Fast full index scan	Scans all blocks in index used to replace a FTS when all necessary columns are in the index. Using multi-block IO & can going parallel
Index joins	Hash join of several indexes that together contain all the table columns that are referenced in the query. Wont eliminate a sort operation
Bitmap indexes	uses a bitmap for key values and a mapping function that converts each bit position to a rowid. Can efficiently merge indexes that correspond to several conditions in a WHERE clause

Figure: 9 façons d'envisager l'accès aux données (extrait de SQLMaria)

Jointure et opérateurs physiques

Join methods

Join Methods	Explanation
Nested Loops joins	For every row in the outer table, Oracle accesses all the rows in the inner table Useful when joining small subsets of data and there is an efficient way to access the second table (index look up)
Hash Joins	The smaller of two tables is scan and resulting rows are used to build a hash table on the join key in memory. The larger table is then scan, join column of the resulting rows are hashed and the values used to probing the hash table to find the matching rows. Useful for larger tables & if equality predicate
Sort Merge joins	Consists of two steps: 1. Sort join operation: Both the inputs are sorted on the join key. 2. Merge join operation: The sorted lists are merged together. Useful when the join condition between two tables is an inequality condition

Figure: Mérites comparés des opérateurs (extrait de SQLMaria)

Premier exemple

```
SQL> explain plan for select nom, num from Emp where num=17232;

Explained.

SQL> select * from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 3956160932

-----------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    20 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    20 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

   1 - filter("NUM"=17232)

Note
-----
   - dynamic sampling used for this statement
```

Figure: Requête de projection/sélection sur table sans index

Premier exemple

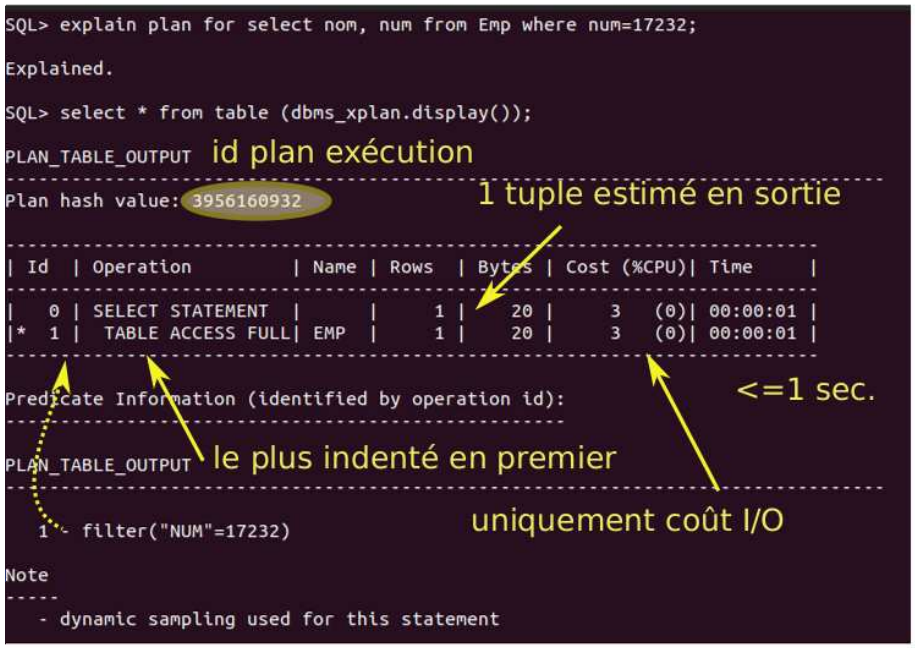


Figure: Plan avec explications

Second exemple

```
SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------
----------------------------------------
Plan hash value: 4024650034

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |     1 |    20 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    20 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | EMP_PK |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------

   2 - access("NUM"=17232)
```

Figure: Même requête mais avec un index sur num

Troisième exemple

```
SQL> explain plan for select num, e.nom, d.nom, d.n_dept from emp e, dept d where e.n_dept=d.n_dept;

Explained.

SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
------------
Plan hash value: 615168685

-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    17 |   935 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN         |      |    17 |   935 |     7  (15)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPT |     4 |    88 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMP  |    17 |   561 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------


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

   1 - access("E"."N_DEPT"="D"."N_DEPT")

Note
-----
   - dynamic sampling used for this statement
```

Figure: Exemple de jointure sans aucun index sur n_dept

Troisième exemple

```
SQL> explain plan for select num, e.nom, d.nom, d.n_dept from emp e, dept d where e.n_dept=d.n_dept;

Explained.

SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
------------
Plan hash value: 615168685

-----------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    17 |   935 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN         |      |    17 |   935 |     7  (15)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPT |     4 |    88 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMP  |    17 |   561 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------

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

   1 - access("E"."N_DEPT"="D"."N_DEPT")

Note
-----
   - dynamic sampling used for this statement
```

mesure coût I/O + CPU

% coût CPU

débute par l'activité la plus en haut

Figure: Plan avec explications

Sous forme algébrique

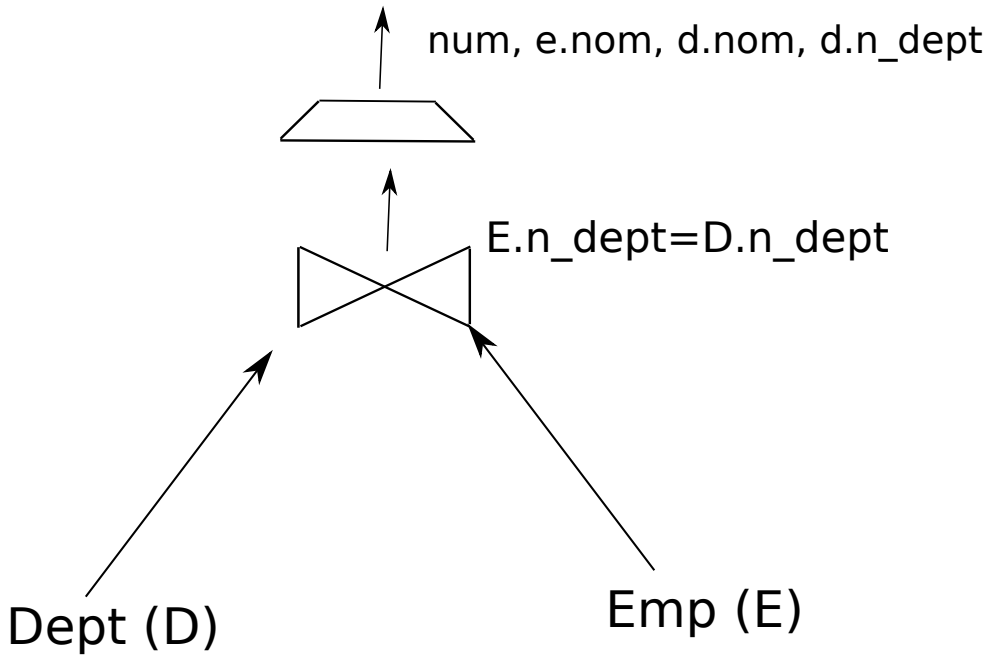


Figure: Arbre algébrique

Sous forme algébrique avec annotations

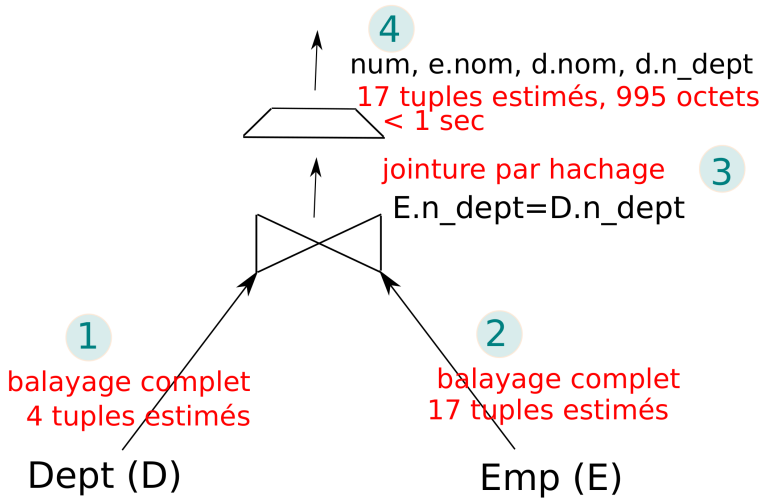


Figure: Arbre algébrique et plan d'exécution

Quatrième exemple

```
SQL> explain plan for select num, e.nom, d.nom, d.n_dept from emp e, dept d where e.n_dept=d.n_dept;

Explained.

SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
------------
Plan hash value: 3487251775

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |    17 |   935 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |         |    17 |   935 |     4   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL          | EMP     |    17 |   561 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    22 |     1   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | DEPT_PK |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
------------

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

   4 - access("E"."N_DEPT"="D"."N_DEPT")

Note
-----
   - dynamic sampling used for this statement
```

Figure: Même requête mais avec un index sur n_dept de DEPT

Cinquième exemple

```
SQL> explain plan for select num, e.nom, d.nom, d.n_dept from emp e, dept d where e.n_dept=d.n_dept;

Explained.

SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
------------
Plan hash value: 1281522714

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |    17 |   935 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP       |     4 |   132 |     1   (0)| 00:00:01 |
|   2 |   NESTED LOOPS               |           |    17 |   935 |     4   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL         | DEPT      |     4 |    88 |     3   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN          | IDX_NDEPT |     6 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
------------

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

   4 - access("E"."N_DEPT"="D"."N_DEPT")

Note
-----
   - dynamic sampling used for this statement
```

Figure: Même requête mais ajout d'index non unique sur n_dept de EMP

Sixième exemple

```
SQL> explain plan for select /*+ use_merge(e d) */ num, e.nom, d.nom, d.n_dept from emp e, dept d where e.n_dept=d.n_dept;

Explained.

SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 699560814

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |    17 |   935 |     6  (17)| 00:00:01 |
|   1 |  MERGE JOIN                  |           |    17 |   935 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP       |    17 |   561 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN           | IDX_NDEPT |    17 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT JOIN                  |           |     4 |    88 |     4  (25)| 00:00:01 |
|   5 |    TABLE ACCESS FULL         | DEPT      |     4 |    88 |     3   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------

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

   4 - access("E"."N_DEPT"="D"."N_DEPT")
       filter("E"."N_DEPT"="D"."N_DEPT")
```

Figure: Opérateur de jointure : tri/fusion

Septième exemple

```
SQL> explain plan for select /*+ no_index(dept dept_pk) */ n_dept from dept;

Explained.

SQL> select plan_table_output from table (dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Plan hash value: 3383998547

-----------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     4 |    52 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| DEPT |     4 |    52 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------
```

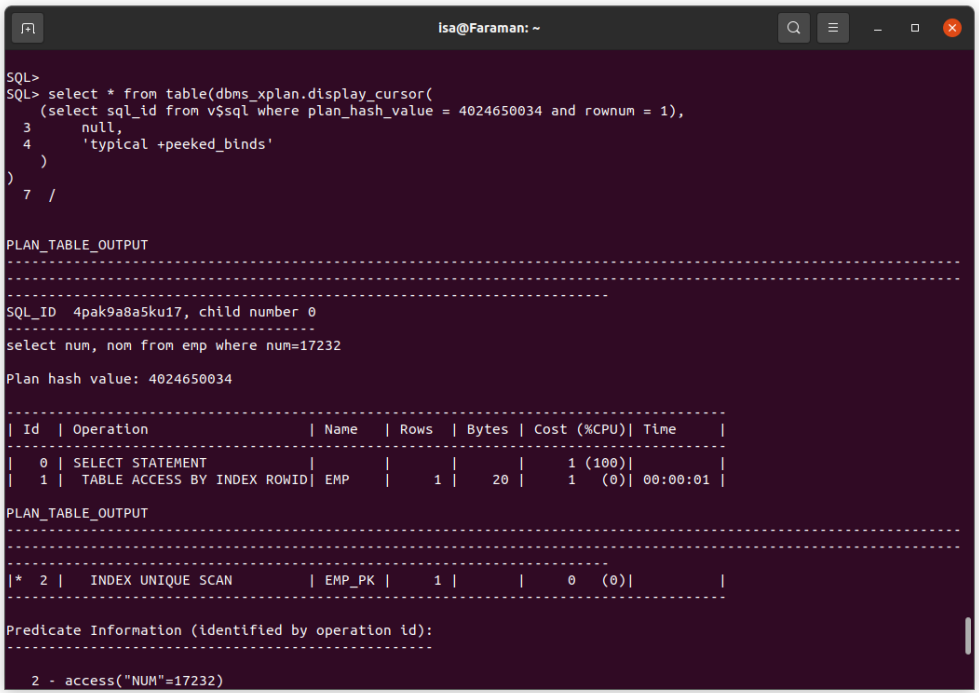
Figure: Requête sans exploitation d'un index cible

Exemple effectif

```
SQL> select num, nom from emp where num=17232;

       NUM NOM
---------- ----------
     17232 BALIN

SQL> select sql_id, plan_hash_value from v$sql where sql_text like 'select num, nom from emp where num=17232';

SQL_ID        PLAN_HASH_VALUE
------------- ---------------
4pak9a8a5ku17      4024650034
```

Figure: Retrouver l'id du curseur/requête et la valeur de hachage du plan à partir de v$sql

Plan d'exécution avec le paquetage dbms_xplan et la fonction display_cursor



Autre façon de faire

```
SELECT /*+ GATHER_PLAN_STATISTICS */ codeinsee, nomcommaj
from commune where codeinsee like '34%';

SELECT * FROM
TABLE(DBMS_XPLAN.display_cursor(format=>'ALLSTATS LAST'));

SELECT * FROM
TABLE(
DBMS_XPLAN.display_cursor
(sql_id=>'4tw9xx',format=>'ALLSTATS LAST +cost +outline'));
```

Plan d'exécution avec estimation et évaluation

```
SELECT * FROM TABLE(DBMS_XPLAN.display_cursor(format=>'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------
SQL_ID  4tw9rnm4yphzu, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ codeinsee, nomcommaj from commune
where codeinsee like '34%'

Plan hash value: 248204825

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |      1 |        |    343 |00:00:00.01 |     150 |      3 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| COMMUNE    |      1 |    279 |    343 |00:00:00.01 |     150 |      3 |
|*  2 |   INDEX RANGE SCAN                  | COMMUNE_PK |      1 |    279 |    343 |00:00:00.01 |      26 |      3 |
-------------------------------------------------------------------------------------------------------------

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

   2 - access("CODEINSEE" LIKE '34%')
       filter("CODEINSEE" LIKE '34%')
```

Figure: Obtenir le plan d'exécution exploité et estimations associées

Le plus simple avec Autotrace

```
set autotrace on
select num, nom from Emp where num=17232;

       NUM NOM
---------- ----------
     17232 BALIN


Plan d'execution
----------------------------------------------------------
Plan hash value: 3956160932

--------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    12 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    12 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------

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

   1 - filter("NUM"=17232)
```

Figure: Exécution, plan d'execution

Le plus simple avec Autotrace

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

Figure: Exécution, statistiques

Mobiliser des outils externes

SQL Developer, Extension VSCodium PL/SQL Developer, SQL Monitor, ...

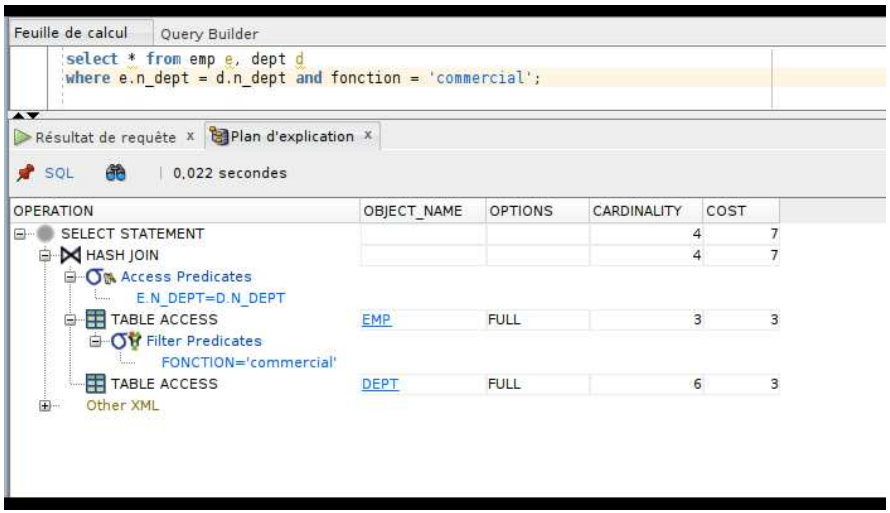


Figure: Exemple de SQL Developer