TP - Traitement de requête - Titouan GAUTIER

Partie 1 : Prise en main

La découverte du PEL :

Question 1 :

```
EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode u1 JOIN unicode u2 ON u2.codepoint = u1.uppercas
WHERE u2.category_ = 'Lu';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

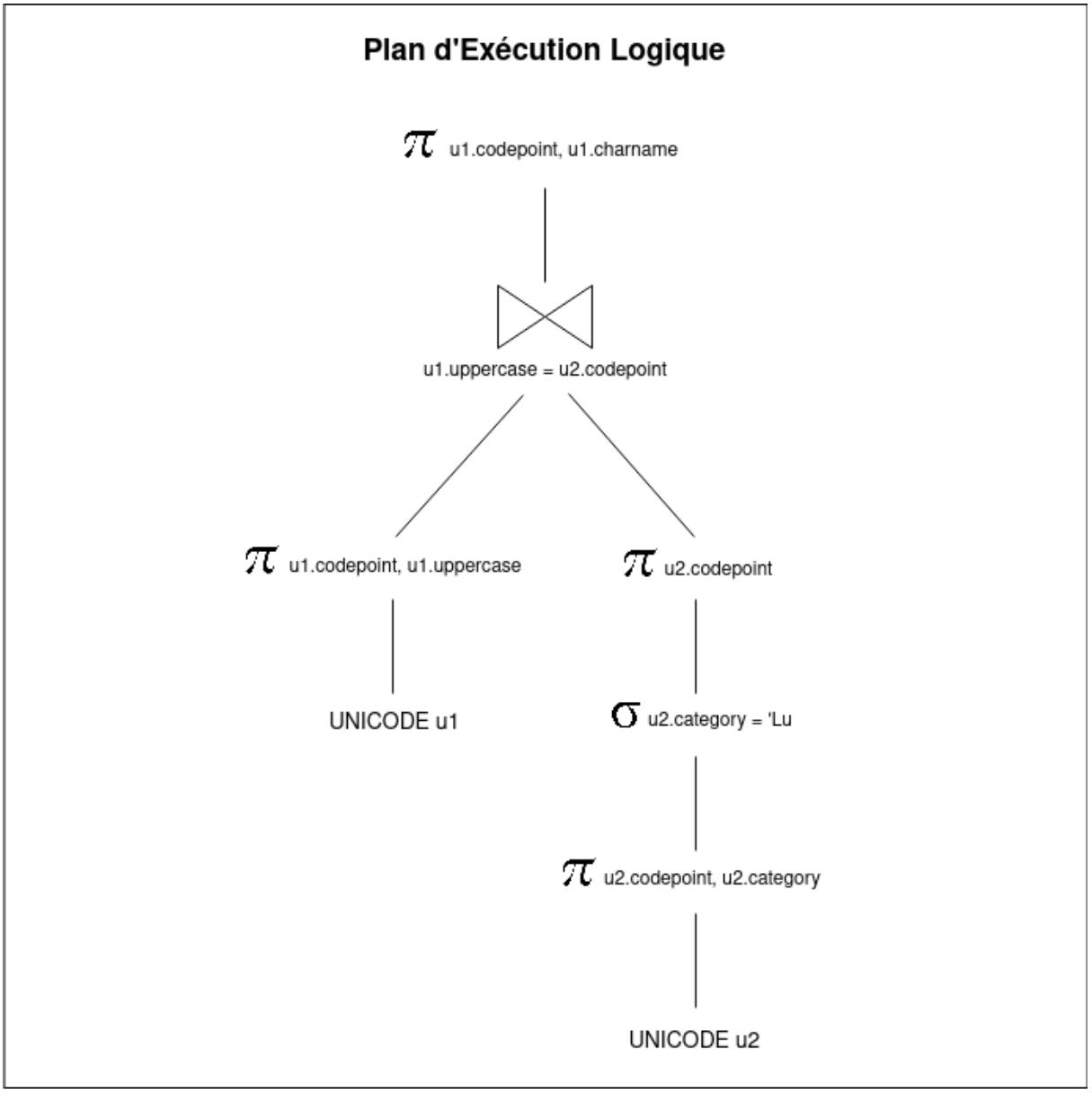
```
Plan hash value: 2187253078

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1781 | 26715 |   122   (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| UNICODE |  1400 | 91000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

```
[2024-02-22 17:45:48] 331 rows retrieved starting from 1,001 in 156 ms (execution: 36 ms, fetching: 120 ms)
E217657J> EXPLAIN PLAN for
          SELECT u1.codepoint, u1.charname
          FROM unicode u1 JOIN unicode u2 ON u2.codepoint = u1.uppercase
          WHERE u2.category_ = 'Lu'
```

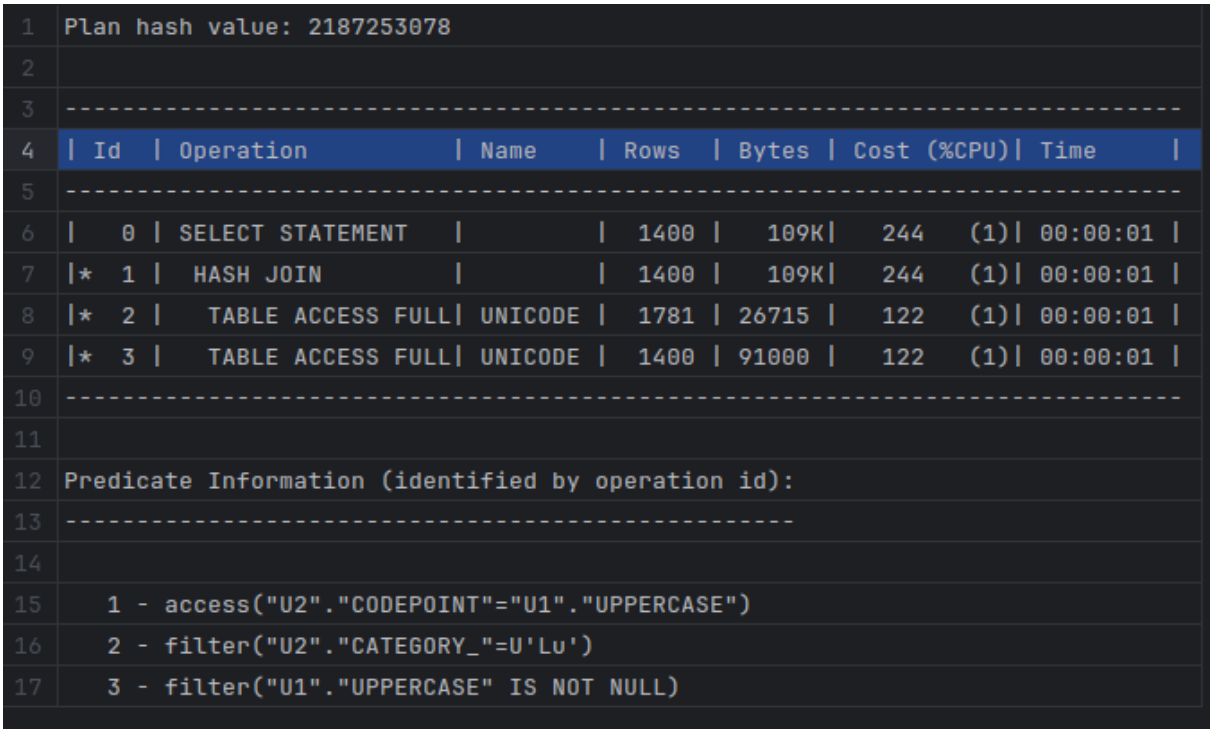
Le plan d'exécution me donne le résultat de 1400 n-uplet et l'IDE me montre un résultat de 1331. Le temps d'exécution donnée par l'IDE est de 36 ms.

Question 2 :

> PEL de la requete Q0



Question 3 :



```
Plan hash value: 2187253078

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1781 | 26715 |   122   (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| UNICODE |  1400 | 91000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

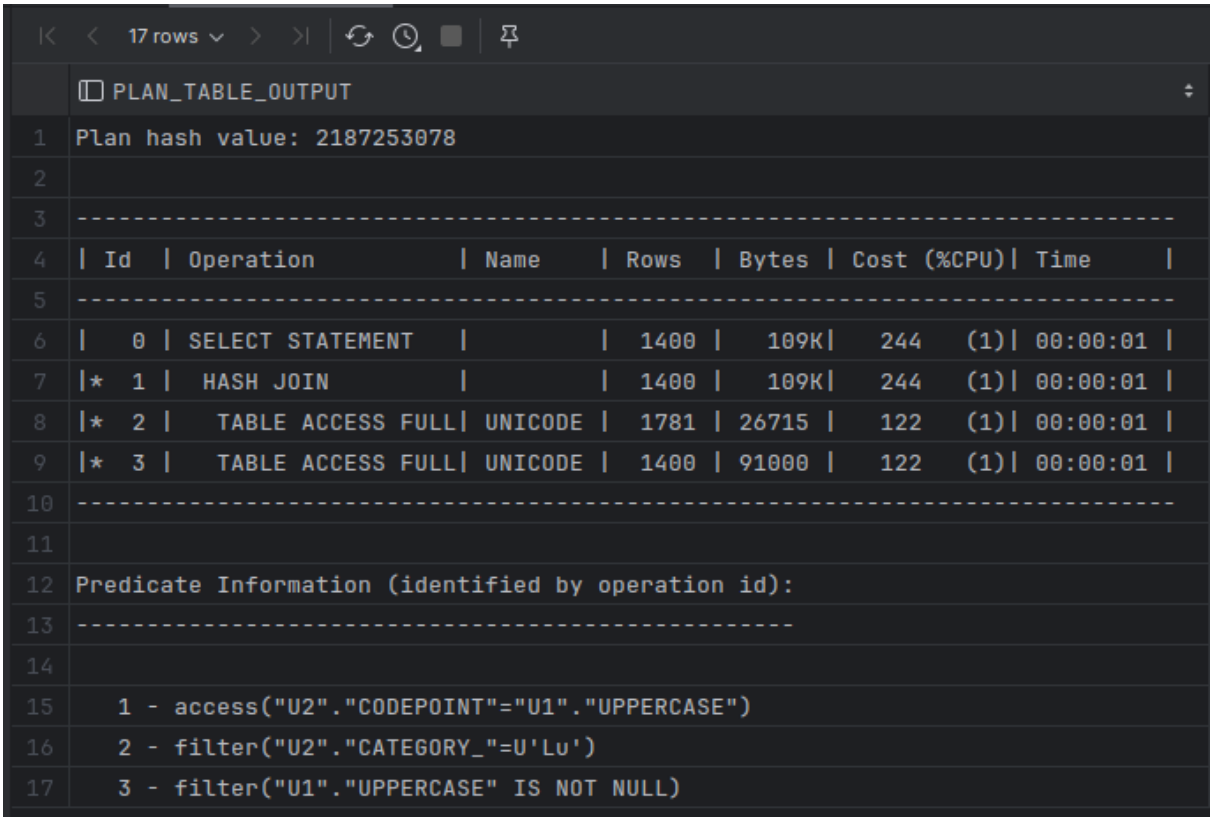
La requête commence par accéder à la table UNICODE via un FULL ACCESS et filtre la colonne UPPERCASE en y appliquant une condition que la valeur ne soit non nulle. Ensuite, il accède une deuxième fois à la table UNICODE toujours en FULL ACCESS et filtre la colonne catégorie en ne gardant que les valeurs égale à ‘Lu’. Ensuite une jointure par table de hachage est réalisé puis on fait une projection sur codepoint et charname.

Question 4 :

> Clause Q0 avec un *exist :*

```
EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode u1
WHERE exists(
    SELECT u2.codepoint
    FROM unicode u2
    WHERE u2.codepoint = u1.uppercase
```

```
    and u2.category_ = 'Lu'
);
```



```
17 rows
PLAN_TABLE_OUTPUT
Plan hash value: 2187253078

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1781 | 26715 |   122   (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| UNICODE |  1400 | 91000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

Ici, on ne voit pas de différence avec le PEL de la requête Q0, car l'optimiseur optimise la requête à notre place.

Clause Q0 avec un *in* :

```
EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode u1 JOIN unicode u2 ON u2.codepoint = u1.uppercas
WHERE u2.category_ in 'Lu';
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 2187253078

-------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  1400 |   109K|   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1781 | 26715 |   122   (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| UNICODE |  1400 | 91000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

Ici, on ne voit pas de différence avec le PEL de la requête Q0, car l'optimiseur optimise la requête à notre place.

Clause Q0 avec une tautologie triviale :

```
SELECT u1.codepoint, u1.charname
FROM unicode u1 JOIN unicode u2 ON u2.codepoint = u1.uppercas
WHERE u2.category_ = 'Lu'
and (u1.COMBINING > 0 or u1.COMBINING <= 0);
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 2187253078

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1400 |   113K|   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  1400 |   113K|   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1781 | 26715 |   122   (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| UNICODE |  1400 | 95200 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

Ici, il y a une petite différence au niveau de la mémoire utilisée qui est plus haute, mais le reste ne change.

Clause Q0 avec une tautologie non triviale :

```
EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode u1 JOIN unicode u2 ON u2.codepoint = u1.uppercas
WHERE u2.category_ = 'Lu'
and (u1.digit > 0 or u1.digit <= 0 or u1.COMBINING > 0 or u1.
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 2187253078

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1400 |   116K|   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  1400 |   116K|   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1781 | 26715 |   122   (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| UNICODE |  1400 | 98000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

Ici, le mémoire utilisé est encore un peu plus élevé qu'avant mais le reste ne change pas.

Les statistiques :

Question 1 :

```
BEGIN
    dbms_stats.delete_table_stats('E217657J', 'unicode');
    dbms_stats.gather_table_stats('E217657J', 'unicode');
END;

select * from user_tab_statistics us where us.table_name = 'U
select * from user_tab_col_statistics uc where uc.table_name
```

USER_TAB_STATISTICS :

- NUM_ROWS: Le nombre total de lignes dans la table.
- BLOCKS: Le nombre de blocs alloués pour stocker les données de la table.

- EMPTY_BLOCKS: Le nombre de blocs vides dans la table.
- AVG_SPACE: L'espace moyen utilisé par une ligne dans la table.
- CHAIN_CNT: Le nombre de chaînes de migration, qui indique le nombre de blocs nécessaires pour stocker une ligne à la suite d'une autre.
- AVG_ROW_LEN: La longueur moyenne d'une ligne en octets.

USER_TAB_COL_STATISTICS :

- NUM_DISTINCT: Le nombre de valeurs distinctes dans la colonne.
- LOW_VALUE et HIGH_VALUE: Les valeurs minimale et maximale de la colonne.
- DENSITY: La densité, qui est le nombre moyen de valeurs distinctes par bloc.
- NUM_NULLS: Le nombre de valeurs nulles dans la colonne.
- NUM_BUCKETS: Le nombre de compartiments utilisés pour l'histogramme.
- SAMPLE_SIZE: La taille de l'échantillon utilisée pour collecter les statistiques.

Question 2 :

```
BEGIN
    dbms_stats.gather_table_stats('E217657J', 'unicode');
END;

SELECT *
FROM USER_TAB_COL_STATISTICS
WHERE TABLE_NAME = 'UNICODE' AND HISTOGRAM <> 'NONE';
```

La colonne CATEGORY_ possède un histogramme. Un histogramme est souvent construit sur une colonne lorsque la distribution des valeurs dans cette colonne n'est pas uniforme, et cela permet à l'optimiseur de prendre des décisions plus intelligentes lors de l'évaluation des plans d'exécution des requêtes.

Question 3 :

```
BEGIN
    dbms_stats.delete_table_stats('E217657J', 'unicode');
END;

EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode u1 JOIN unicode u2 ON u2.codepoint = u1.uppercas
WHERE u2.category_ = 'Lu';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 2187253078

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |   690 | 91080 |   244   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |   690 | 91080 |   244   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |   460 |  6440 |   122   (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| UNICODE | 30786 |  3547K|   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

On remarque que le nombre de lignes est bien plus grand que le nombre de lignes du premier PEL. On remarque une note qui dit que les statistiques dynamiques ont été utilisés, c'est-à-dire qu'il recueille des statistiques en

temps réel de l’exécution de la fonction. Malgré cela, la requête est moins performante dû au manque de statistiques.

Les indexes :

Question 1 :

```
select * from user_indexes where table_name = 'UNICODE';
select * from user_segments where segment_name = 'UNICODE';
```

USER_INDEXES :

- INDEX_NAME : Le nom de l'index.
- TABLE_NAME : Le nom de la table sur laquelle l'index est créé.
- TABLE_OWNER : Le propriétaire de la table.
- TABLE_TYPE : Le type de la table (par exemple, "TABLE" pour une table standard).
- UNIQUENESS : Indique si l'index est unique ("UNIQUE") ou non ("NONUNIQUE").
- COMPRESSION : Indique si l'index utilise la compression ("ENABLED") ou non ("DISABLED").
- STATUS : Indique si l'index est actif ("VALID") ou invalide ("INVALID").
- PARTITIONED : Indique si l'index est partitionné ("YES") ou non ("NO").
- NUM_ROWS : Nombre approximatif de lignes dans l'index.
- BLOCKS : Nombre de blocs alloués à l'index.
- INDEX_TYPE : Type d'index (par exemple, "NORMAL" pour un index standard).

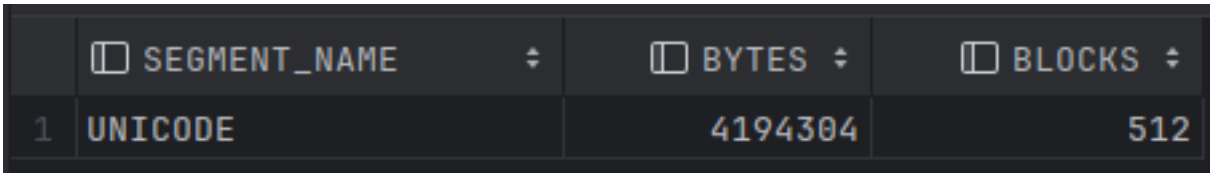
USER_SEGMENTS :

- SEGMENT_NAME : Le nom du segment.

- SEGMENT_TYPE : Le type de segment (par exemple, "TABLE" pour une table, "INDEX" pour un index).
- TABLE_NAME : Le nom de la table associée au segment (si applicable).
- TABLESPACE_NAME : Le nom de l'espace de table (tablespace) où le segment est stocké.
- BYTES : La taille en octets du segment.
- BLOCKS : Le nombre de blocs alloués pour le segment.
- SEGMENT_CREATED : La date à laquelle le segment a été créé.

Pour trouver la taille de l'index, je fais :

```
SELECT segment_name, bytes, blocks
FROM user_segments
where segment_name = 'UNICODE';
```



	SEGMENT_NAME	BYTES	BLOCKS
1	UNICODE	4194304	512

Question 2 :

> Je crée la contrainte unique

```
ALTER TABLE unicode
ADD CONSTRAINT unicite_oldname UNIQUE (oldname);
```

> Je regarde si l'index a été créé

```
select index_name, table_name, uniqueness
from user_indexes
```

```
where table_name = 'UNICODE';
```

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	UNICITE_OLDNAME	UNICODE	UNIQUE
2	PK_UNICODE	UNICODE	UNIQUE

On peut voir qu'un nouvel index a été créé et qu'il est unique (colonne UNIQUENESS).

Question 3 :

```
explain plan for
CREATE INDEX idx_unicode_covering
ON unicode (codepoint, uppercase, category_, charname);

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

	PLAN_TABLE_OUTPUT
1	Plan hash value: 4107150674
2	
3	----------
4	| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
5	----------
6	| 0 | CREATE INDEX STATEMENT | | 32292 | 2207K| 165 (0)| 00:00:01 |
7	| 1 | INDEX BUILD NON UNIQUE| IDX_UNICODE_COVERING | | | | |
8	| 2 | SORT CREATE INDEX | | 32292 | 2207K| | |
9	| 3 | INDEX FAST FULL SCAN| IDX_UNICODE_COVERING | 32292 | 2207K| 99 (0)| 00:00:01 |
10	----------
11	
12	Note
13	-----
14	- estimated index size: 3145K bytes

On peut voir que dans le explain plan la taille de l'index est estimé à 3145K bytes soit 3.06MB.

Question 4 :

> Je crée la table unicode 2 avec l'index plaçant

```
CREATE table unicode2 (
    codepoint NVARCHAR2(6) PRIMARY KEY,
    charname NVARCHAR2(100),
    uppercase NVARCHAR2(6),
    category_ NCHAR(2),
    FOREIGN KEY (codepoint) REFERENCES UNICODE(CODEPOINT)
)
ORGANIZATION INDEX
INCLUDING category_ overflow;
```

> J'ajoute les données de unicode dans unicode2

```
insert into unicode2 (codepoint, charname, uppercase, categor
SELECT u1.codepoint, u1.charname, u1.uppercase, u1.category_
```

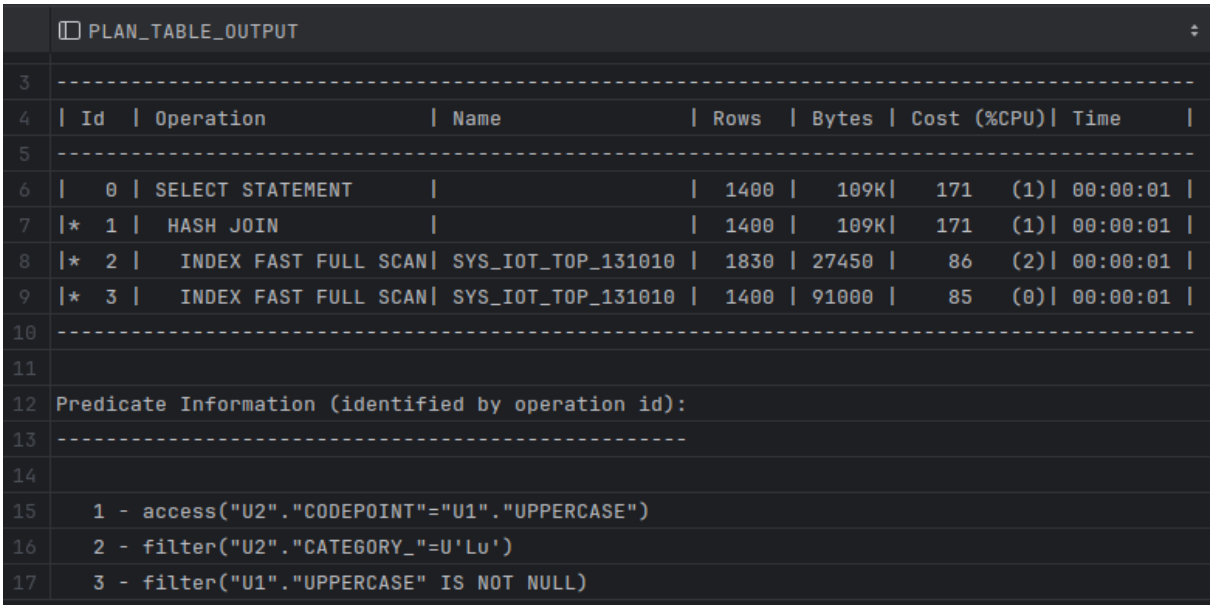
> Je vérifie que l'index a été créé

```
select index_name, table_name, table_owner
from user_indexes
where table_name='UNICODE2';
```

	INDEX_NAME	TABLE_NAME	TABLE_OWNER
1	SYS_IOT_TOP_131010	UNICODE2	E217657J

Question 5 :

```
EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode2 u1 JOIN unicode2 u2 ON u2.codepoint = u1.upperc
WHERE u2.category_ = 'Lu';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```



```
PLAN_TABLE_OUTPUT
3  ------------------------------------------------------------------------------------------
4  | Id  | Operation            | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT     |                    |  1400 |   109K|   171   (1)| 00:00:01 |
7  |*  1 |  HASH JOIN           |                    |  1400 |   109K|   171   (1)| 00:00:01 |
8  |*  2 |   INDEX FAST FULL SCAN| SYS_IOT_TOP_131010 |  1830 | 27450 |    86   (2)| 00:00:01 |
9  |*  3 |   INDEX FAST FULL SCAN| SYS_IOT_TOP_131010 |  1400 | 91000 |    85   (0)| 00:00:01 |
10 ------------------------------------------------------------------------------------------
11
12 Predicate Information (identified by operation id):
13 ---------------------------------------------------
14
15    1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
16    2 - filter("U2"."CATEGORY_"=U'Lu')
17    3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

Après avoir mis à jour les statistiques sur unicode2, on obtient ce PEL. On voit qu'il est similaire à celui obtenu lors du premier plan d'exécution de Q0. Le temps d'éxécution est approximativement le même que la première requête et elle affiche le même nombre de lignes.

Question 6 :

> PEL de la requête avec la table unicode et l'index couvrant

```
PLAN_TABLE_OUTPUT
Plan hash value: 4034532820

-----------------------------------------------------------------------------------------------
| Id  | Operation             | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                      |  1400 |   109K|   199   (1)| 00:00:01 |
|*  1 |  HASH JOIN            |                      |  1400 |   109K|   199   (1)| 00:00:01 |
|*  2 |   INDEX FAST FULL SCAN| IDX_UNICODE_COVERING |  1781 | 26715 |   100   (1)| 00:00:01 |
|*  3 |   INDEX FAST FULL SCAN| IDX_UNICODE_COVERING |  1400 | 91000 |    99   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

> PEL de la requête avec la table unicode2 et l'index cluster

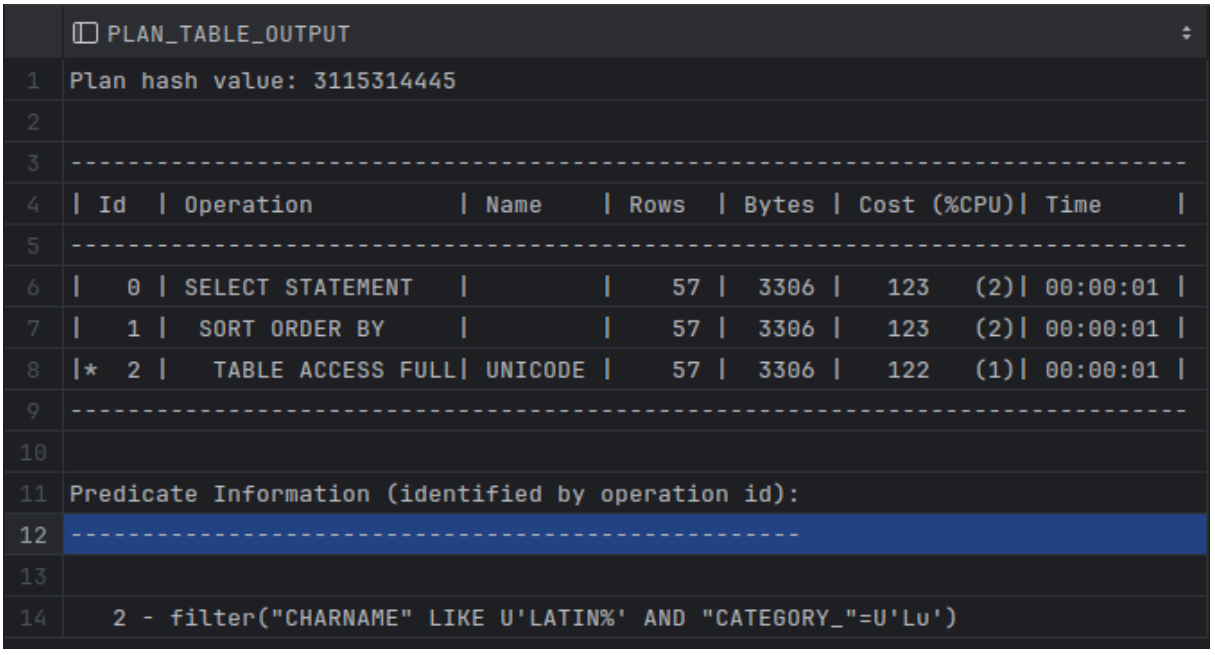
```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
| Id  | Operation             | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                      |  1400 |   109K|   171   (1)| 00:00:01 |
|*  1 |  HASH JOIN            |                      |  1400 |   109K|   171   (1)| 00:00:01 |
|*  2 |   INDEX FAST FULL SCAN| SYS_IOT_TOP_131010   |  1830 | 27450 |    86   (2)| 00:00:01 |
|*  3 |   INDEX FAST FULL SCAN| SYS_IOT_TOP_131010   |  1400 | 91000 |    85   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   1 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   2 - filter("U2"."CATEGORY_"=U'Lu')
   3 - filter("U1"."UPPERCASE" IS NOT NULL)
```

On remarque que les requêtes sont similaires. Cependant, on peut voir que le nombre de lignes traité est un peu plus faible pour la requête avec l'index couvrant, mais le cout en CPU est plus élevé que pour l'index cluster. Le temps d'exécution est approximativement le même.

Partie 2 : Analyse du PEL

Question 1 :

```
select charname
from unicode
where category_ = 'Lu'
and charname like 'LATIN%'
order by charname asc;
```



```
PLAN_TABLE_OUTPUT
Plan hash value: 3115314445

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |    57 |  3306 |   123   (2)| 00:00:01 |
|   1 |  SORT ORDER BY     |         |    57 |  3306 |   123   (2)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |    57 |  3306 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - filter("CHARNAME" LIKE U'LATIN%' AND "CATEGORY_"=U'Lu')
```

Ici l'optimiseur est obligé d'utiliser un TABLE FULL ACCESS pour accéder à la table, car il n'existe pas d'index sur les colonnes CHARNAME et CATEGORY_.

Question 2 :

```
select count(codepoint)
from unicode
where bidi = 'ON';
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 1474100434

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |     4 |   122   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |         |     1 |     4 |            |          |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1404 |  5616 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - filter("BIDI"=U'ON')
```

Ici, il y a un TABLE FULL ACCESS, car il n'y a pas d'index sur la colonne bidi. SORT AGGREGATE est utilisé parce que l'opérateur count est utilisé.

Question 3 :

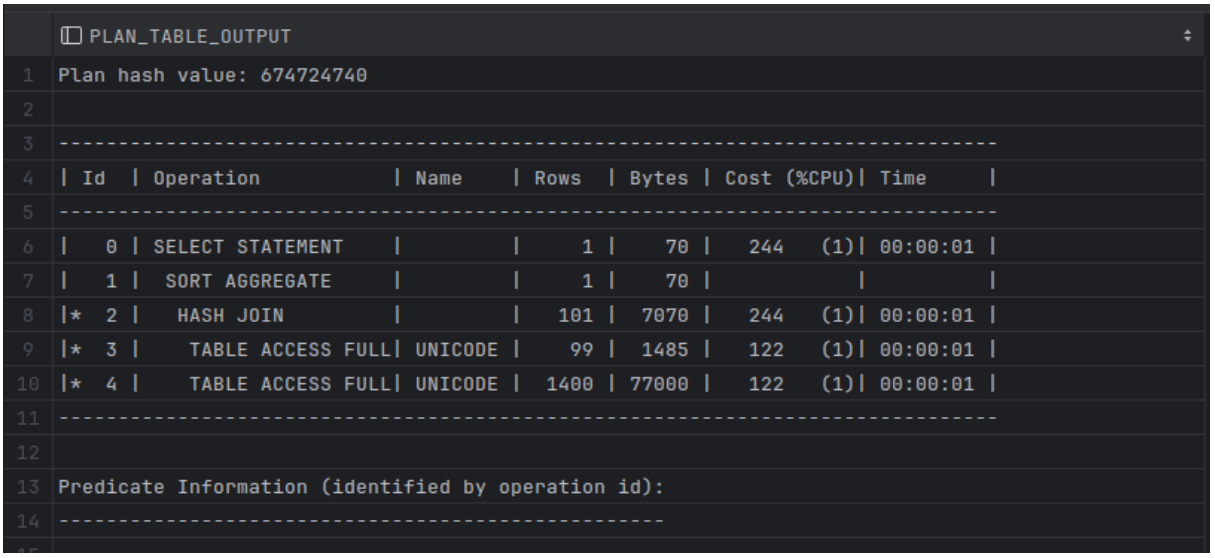
```
select u.codepoint,
             u.charname,
             u.category_,
             u.combining,
             u.bidi,
             u.decomposition,
             u.decimal_,
             u.digit,
             u.numeric_,
             u.mirrored,
             u.oldname,
             u.comment_,
             lc.charname as lowercase,
             uc.charname as uppercase,
             tc.charname as titlecase
from unicode u left outer join unicode lc on u.lowercase=lc.c
left outer join unicode uc on u.uppercase=uc.codepoint
left outer join unicode tc on u.titlecase=tc.codepoint
where u.codepoint='0405';
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 709436845

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     1 |   280 |     5   (0)| 00:00:01 |
|   1 |  NESTED LOOPS OUTER           |            |     1 |   280 |     5   (0)| 00:00:01 |
|   2 |   NESTED LOOPS OUTER          |            |     1 |   217 |     4   (0)| 00:00:01 |
|   3 |    NESTED LOOPS OUTER         |            |     1 |   154 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| UNICODE    |     1 |    91 |     2   (0)| 00:00:01 |
|*  5 |      INDEX UNIQUE SCAN        | PK_UNICODE |     1 |       |     1   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID| UNICODE    |     1 |    63 |     1   (0)| 00:00:01 |
|*  7 |      INDEX UNIQUE SCAN        | PK_UNICODE |     1 |       |     0   (0)| 00:00:01 |
|   8 |    TABLE ACCESS BY INDEX ROWID | UNICODE    |     1 |    63 |     1   (0)| 00:00:01 |
|*  9 |     INDEX UNIQUE SCAN         | PK_UNICODE |     1 |       |     0   (0)| 00:00:01 |
|  10 |   TABLE ACCESS BY INDEX ROWID  | UNICODE    |     1 |    63 |     1   (0)| 00:00:01 |
|* 11 |    INDEX UNIQUE SCAN          | PK_UNICODE |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Ici l'optimiseur utilise l'index sur la clé primaire pour faire la condition codepoint = '0405'. Ensuite la première jointure s'effectue entre UNICODE U et UNICODE TC, puis la jointure entre UNICODE U et UNICODE UC, puis la jointure entre UNICODE U et UNICODE LC. Les jointures sont des NESTED LOOPS, car on utilise une LEFT OUTER JOIN dans le code.

Question 4 :

```
select avg(length(u.charname))
from unicode u
join unicode uc on u.uppercase = uc.codepoint
where uc.oldname like '%GREEK%'
```



```
PLAN_TABLE_OUTPUT
Plan hash value: 674724740

-------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |     1 |    70 |   244   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE     |         |     1 |    70 |            |          |
|*  2 |   HASH JOIN         |         |   101 |  7070 |   244   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| UNICODE |    99 |  1485 |   122   (1)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| UNICODE |  1400 | 77000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

Ici l'optimiseur a choisi un HASH JOIN probablement, car la clause uc.oldname like '%GREEK%' est complexe.

Partie 3 : Les opérateurs

Question 1 :

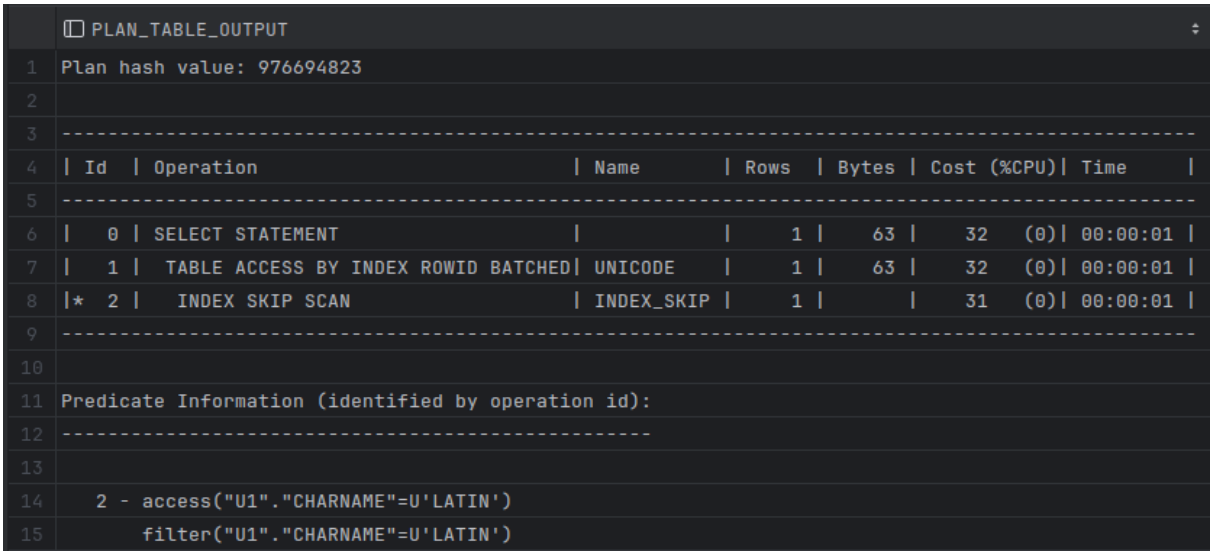
> Requete utilisant l'opérateur *INDEX RANGE SCAN*

```
create index index_digit on unicode(digit);

explain plan for
select u1.codepoint
from unicode u1
where u1.DIGIT > 2;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 1418107787

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                  |   574 |  6888 |    95   (2)| 00:00:01 |
|*  1 |  VIEW                 | index$_join$_001 |   574 |  6888 |    95   (2)| 00:00:01 |
|*  2 |   HASH JOIN           |                  |       |       |            |          |
|*  3 |    INDEX RANGE SCAN   | INDEX_DIGIT      |   574 |  6888 |     3   (0)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| PK_UNICODE      |   574 |  6888 |   114   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - filter("U1"."DIGIT">2)
   2 - access(ROWID=ROWID)
   3 - access("U1"."DIGIT">2)
```

Requete utilisant l'opérateur *INDEX SKIP SCAN*

```
create index index_skip on unicode(category_, charname);

explain plan for
select u1.codepoint
from unicode u1
where 'LATIN' = u1.CHARNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```



```
PLAN_TABLE_OUTPUT
Plan hash value: 976694823

-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |     1 |    63 |    32   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| UNICODE    |     1 |    63 |    32   (0)| 00:00:01 |
|*  2 |   INDEX SKIP SCAN                   | INDEX_SKIP |     1 |       |    31   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   2 - access("U1"."CHARNAME"=U'LATIN')
       filter("U1"."CHARNAME"=U'LATIN')
```

Question 2 :

> Requete utilisant une *NESTED LOOPS*

```
EXPLAIN PLAN for
SELECT u1.codepoint, u1.charname
FROM unicode u1
INNER JOIN unicode u2 ON u2.codepoint = u1.uppercase
WHERE u1.category_ = 'Lu'
  AND u2.CHARNAME LIKE 'LATIN%';
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 3928356621

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |    58 |  7714 |   199   (1)| 00:00:01 |
|   1 |  NESTED LOOPS                |             |    58 |  7714 |   199   (1)| 00:00:01 |
|   2 |   NESTED LOOPS               |             |    77 |  7714 |   199   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL         | UNICODE     |    77 |  5390 |   122   (1)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | PK_UNICODE  |     1 |       |     0   (0)| 00:00:01 |
|*  5 |   TABLE ACCESS BY INDEX ROWID| UNICODE     |     1 |    63 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   3 - filter("U1"."UPPERCASE" IS NOT NULL AND "U1"."CATEGORY_"=U'Lu')
   4 - access("U2"."CODEPOINT"="U1"."UPPERCASE")
   5 - filter("U2"."CHARNAME" LIKE U'LATIN%')

Note
-----
   - this is an adaptive plan
```

Requete utilisant un *MERGE JOIN*

```
EXPLAIN PLAN for
select /*+ use_merge(u1,u2) */ u1.UPPERCASE, u2.LOWERCASE
from unicode u1 join unicode u2 on u1.UPPERCASE = u2.LOWERCAS

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 2973739042

-------------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |  1409 |  5636 |   246   (2)| 00:00:01 |
|   1 |  MERGE JOIN         |         |  1409 |  5636 |   246   (2)| 00:00:01 |
|   2 |   SORT JOIN         |         |  1383 |  2766 |   123   (2)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| UNICODE |  1383 |  2766 |   122   (1)| 00:00:01 |
|*  4 |   SORT JOIN         |         |  1400 |  2800 |   123   (2)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL| UNICODE |  1400 |  2800 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   3 - filter("U2"."LOWERCASE" IS NOT NULL)
   4 - access("U1"."UPPERCASE"="U2"."LOWERCASE")
       filter("U1"."UPPERCASE"="U2"."LOWERCASE")
   5 - filter("U1"."UPPERCASE" IS NOT NULL)
```

Question 3 :

> Requete d'une triple jointure utilisant plusieurs type de jointure

```
EXPLAIN PLAN FOR
SELECT /*+ use_merge(u2,u4) */ u1.UPPERCASE, u2.LOWERCASE
FROM unicode u1
JOIN unicode u2 on u2.LOWERCASE = u1.CODEPOINT and u2.UPPERCA
JOIN unicode u3 on u3.UPPERCASE = u1.CODEPOINT
JOIN unicode u4 on u4.LOWERCASE = u2.CODEPOINT;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 2275207555

-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |             |     3 |    90 |   428   (2)| 00:00:01 |
|   1 |  MERGE JOIN                    |             |     3 |    90 |   428   (2)| 00:00:01 |
|   2 |   SORT JOIN                    |             |     3 |    84 |   305   (1)| 00:00:01 |
|*  3 |    HASH JOIN                   |             |     3 |    84 |   304   (1)| 00:00:01 |
|   4 |     NESTED LOOPS               |             |    60 |  1560 |   182   (1)| 00:00:01 |
|   5 |      NESTED LOOPS              |             |    60 |  1560 |   182   (1)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL        | UNICODE     |    60 |   840 |   122   (1)| 00:00:01 |
|*  7 |       INDEX UNIQUE SCAN        | PK_UNICODE  |     1 |       |     0   (0)| 00:00:01 |
|   8 |      TABLE ACCESS BY INDEX ROWID| UNICODE    |     1 |    12 |     1   (0)| 00:00:01 |
|*  9 |     TABLE ACCESS FULL          | UNICODE     |  1400 |  2800 |   122   (1)| 00:00:01 |
|* 10 |   SORT JOIN                    |             |  1383 |  2766 |   123   (2)| 00:00:01 |
|* 11 |    TABLE ACCESS FULL           | UNICODE     |  1383 |  2766 |   122   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   3 - access("U3"."UPPERCASE"="U1"."CODEPOINT")
   6 - filter("U2"."LOWERCASE" IS NOT NULL AND "U2"."UPPERCASE" IS NOT NULL)
   7 - access("U2"."UPPERCASE"="U1"."CODEPOINT")
       filter("U2"."LOWERCASE"="U1"."CODEPOINT")
   9 - filter("U3"."UPPERCASE" IS NOT NULL)
  10 - access("U4"."LOWERCASE"="U2"."CODEPOINT")
       filter("U4"."LOWERCASE"="U2"."CODEPOINT")
  11 - filter("U4"."LOWERCASE" IS NOT NULL)
```

Question 4 :

> Requete utilisant un *SORT UNIQUE*

```
EXPLAIN PLAN for
SELECT distinct u1.DIGIT
from unicode u1
order by u1.DIGIT;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

```
PLAN_TABLE_OUTPUT
3  ------------------------------------------------------------------------------
4  | Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT   |         |    10 |    20 |   126   (4)| 00:00:01 |
7  |   1 |  SORT UNIQUE       |         |    10 |    20 |   124   (3)| 00:00:01 |
8  |   2 |   TABLE ACCESS FULL| UNICODE | 32292 | 64584 |   122   (1)| 00:00:01 |
9  ------------------------------------------------------------------------------
```

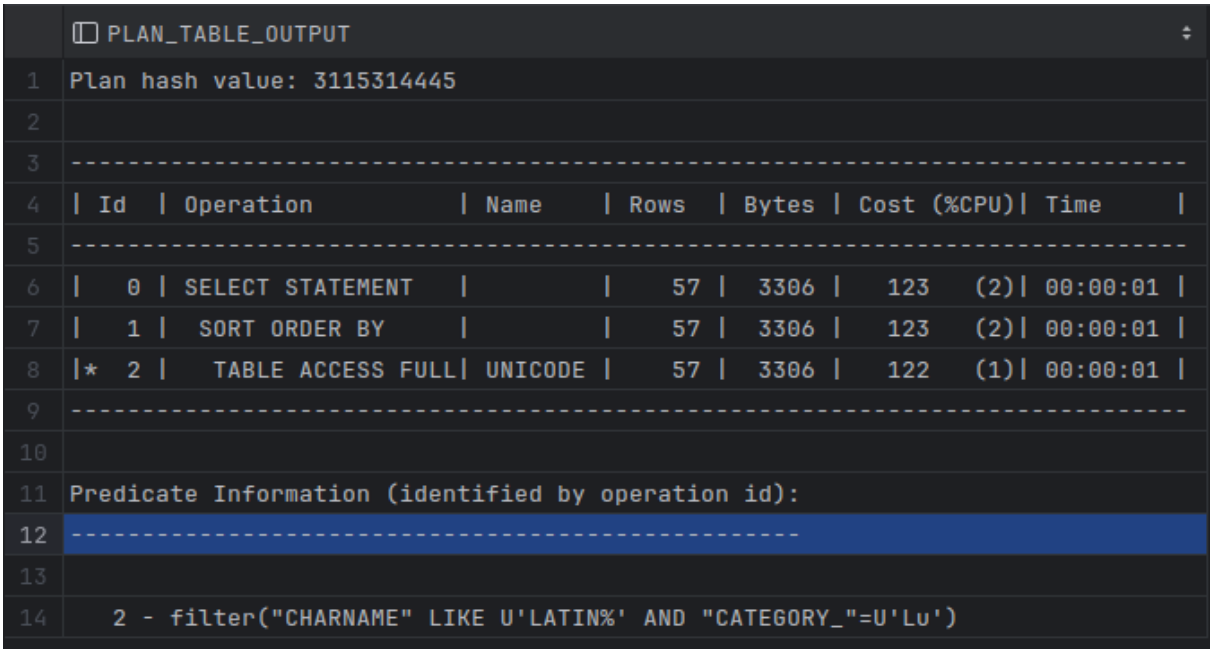
Requet utilisant un *HASH GROUP BY*

```
EXPLAIN PLAN FOR
select u1.CATEGORY_, count(*)
from unicode u1
group by U1.CATEGORY_;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE'));
```

```
PLAN_TABLE_OUTPUT
3  ------------------------------------------------------------------------------
4  | Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT   |         |    29 |   145 |   124   (3)| 00:00:01 |
7  |   1 |  HASH GROUP BY     |         |    29 |   145 |   124   (3)| 00:00:01 |
8  |   2 |   TABLE ACCESS FULL| UNICODE | 32292 |  157K|   122   (1)| 00:00:01 |
9  ------------------------------------------------------------------------------
```

Partie 4 : Tuning

Requete 1

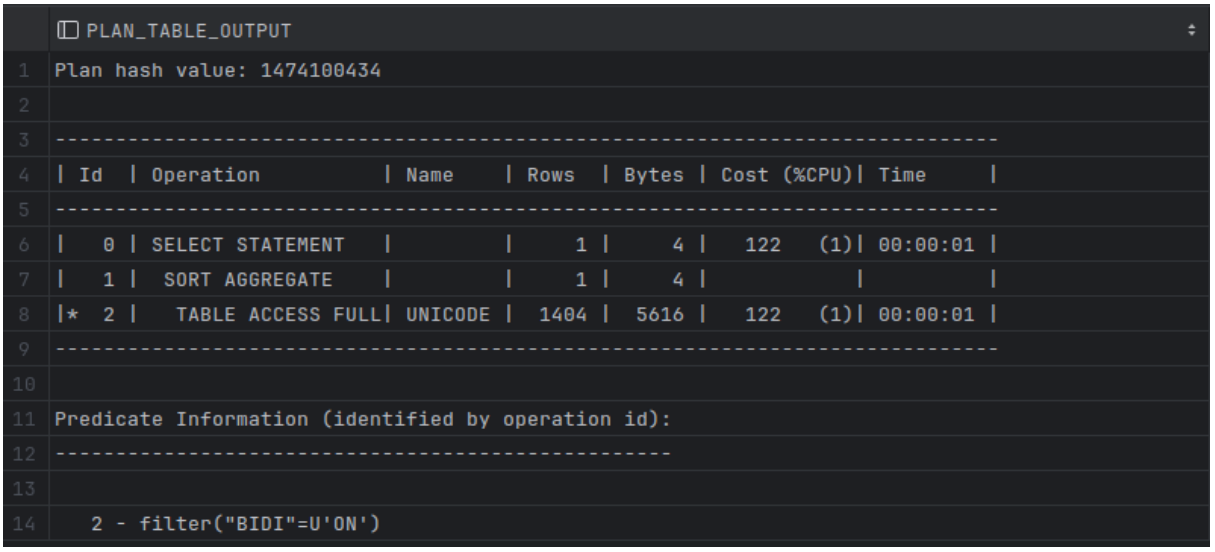


```
PLAN_TABLE_OUTPUT
Plan hash value: 3115314445

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |    57 |  3306 |   123   (2)| 00:00:01 |
|   1 |  SORT ORDER BY     |         |    57 |  3306 |   123   (2)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| UNICODE |    57 |  3306 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - filter("CHARNAME" LIKE U'LATIN%' AND "CATEGORY_"=U'Lu')
```

Sachant que la requête doit être performante, car elle est exécuté régulièrement, je pense qu'on peut l'améliorer : on peut créer un index sur category pour améliorer les performances et le temps d'exécution. On peut aussi créer un index sur charname. On peut le faire comme ceci :

```
CREATE INDEX index_category ON unicode(category_);
CREATE INDEX index_charname ON unicode(charname);
```

Requete 2



```
PLAN_TABLE_OUTPUT
Plan hash value: 1474100434

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |     4 |   122   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |         |     1 |     4 |            |          |
|*  2 |   TABLE ACCESS FULL| UNICODE |  1404 |  5616 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - filter("BIDI"=U'ON')
```

Ici la requête doit être performante. Par conséquent, je pense que l'on peut créer un index sur bidi pour améliorer la performance. On peut aussi utiliser count(*) à la place de count(codepoint) car codepoint est une clé primaire.

Requête 3

PLAN_TABLE_OUTPUT

```
Plan hash value: 709436845

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     1 |   280 |     5   (0)| 00:00:01 |
|   1 |  NESTED LOOPS OUTER           |            |     1 |   280 |     5   (0)| 00:00:01 |
|   2 |   NESTED LOOPS OUTER          |            |     1 |   217 |     4   (0)| 00:00:01 |
|   3 |    NESTED LOOPS OUTER         |            |     1 |   154 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| UNICODE    |     1 |    91 |     2   (0)| 00:00:01 |
|*  5 |      INDEX UNIQUE SCAN        | PK_UNICODE |     1 |       |     1   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID| UNICODE    |     1 |    63 |     1   (0)| 00:00:01 |
|*  7 |      INDEX UNIQUE SCAN        | PK_UNICODE |     1 |       |     0   (0)| 00:00:01 |
|   8 |    TABLE ACCESS BY INDEX ROWID | UNICODE    |     1 |    63 |     1   (0)| 00:00:01 |
|*  9 |     INDEX UNIQUE SCAN         | PK_UNICODE |     1 |       |     0   (0)| 00:00:01 |
|  10 |   TABLE ACCESS BY INDEX ROWID  | UNICODE    |     1 |    63 |     1   (0)| 00:00:01 |
|* 11 |    INDEX UNIQUE SCAN          | PK_UNICODE |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Ici, on peut créer des indexes sur les colonnes de jointure pour améliorer les performances.

```
CREATE INDEX idx_lower ON unicode(lowercase);
CREATE INDEX idx_upper ON unicode(uppercase);
CREATE INDEX idx_title ON unicode(titlecase)
```

Requête 4

```
PLAN_TABLE_OUTPUT
Plan hash value: 674724740

-------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |     1 |    70 |   244   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE     |         |     1 |    70 |            |          |
|*  2 |   HASH JOIN         |         |   101 |  7070 |   244   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| UNICODE |    99 |  1485 |   122   (1)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| UNICODE |  1400 | 77000 |   122   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

Ici, la requête suffit pour notre utilisation, car elle ne s'exécute pas souvent et la réponse peut attendre.

Conclusion

Le traitement des requêtes SQL et la compréhension du fonctionnement de l'optimiseur est très important pour pouvoir établir des requêtes performantes. En effet, nous avons vu au cours de ce TP que les performances peuvent grandement varier en fonction de l'optimisation de la requête.