Titulación: GRADO INGENIERÍA INFORMÁTICA Año Académico: 2023/24 -- Curso: 2º Asignatura: Ficheros y Bases de Datos Título: Memoria Práctica 3 – Diseño Físico en Oracle	uc3m | Universidad Carlos III de Madrid

Alumno/a:	**MARIO RAMOS SALSON**	**NIA:**	**100495849**
Alumno/a:	**VICTOR MARTINEZ DE LAS HERAS**	**NIA:**	**100495829**
Alumno/a:	**MIGUEL FIDALGO GARCIA**	**NIA:**	**100495770**

ÍNDICE

1. Introducción

La base de datos que se nos presenta está bien estructurada y organizada, sin embargo algunas acciones como la unión de dos tablas diferentes o la lectura completa y parcial de dichas tablas, puede llegar a ser algo lenta. Para ello hemos decido analizar las 5 consultas más frecuentes sobre dos de nuestras tablas, con el objetivo de optimizar al máximo su eficiencia. Entre estás 5 consultas, encontramos búsquedas completas, filtrados por cables primarias o secundarias y búsquedas por atributos que no poseen ningún tipo de restricción.

2. Análisis

Haciendo un análisis de las operaciones que más se realizan en nuestra base de datos, nos damos cuenta que todas ellas son consultas. Esto significa que los cambios de optimización que hagamos tienen que estar enfocados en mejorar las lecturas lo máximo posible, aunque pueda perjudicar las escrituras. En el diseño inicial, no se tiene en cuenta cuales son las operaciones más frecuentes, por lo que tiene un diseño equilibrado que no daña ni mejora las lecturas o escrituras en particular.

Las dos primeras instrucciones son lecturas con un filtro. Originalmente, tendrían que hacer un escaneo completo de la tabla y buscar las filas que cumplan la condición una a una. Haciendo un índice en las columnas en las que se aplica el filtro, la base de datos puede utilizarlo para buscar directamente las filas relevantes.

En la tercera consulta, como la multiplicidad de las filas que cumplen la condición dada es muy alta, es contraproducente hacer un índice. En primer lugar, se nos ocurrió hacer una vista materializada, ya que como se almacenan en memoria RAM y no en disco, hace que sus accesos sean mucho más rápidos y eficientes. Como esta opción no era válida debido a las restricciones de la práctica, optamos por usar paralelismo. Con el paralelismo se puede realizar la consulta más rápidamente aprovechando al máximo los recursos del sistema.

La cuarta consulta requería hacer un escaneo completo de la tabla, por lo que un índice no era una opción. A parte de implementar de nuevo el paralelismo, probamos a modificar el tamaño de bloque. Para ello hemos realizado unos cálculos para determinar qué tamaño de bloque es mejor.

Tamaño medio de registro = 917 bytes

Tamaño de 2K: 2048 bytes * 0.9 = 1843,2 $\longrightarrow$ 1843,2 mod 917 = 9,2

Tamaño de 8K: 8192 bytes * 0.9 = 7372,8 $\longrightarrow$ 7372,8 mod 917 = 36,8

Tamaño de 16K: 16384 bytes * 0.9 = 14745,6 $\longrightarrow$ 14745,6 mod 917 = 73,6

Según los cálculos, teóricamente el tamaño de cubo que deja menos espacio sería el segundo, pero esto es teniendo en cuenta que entran dos registros ajustados por cubo. En la práctica, esto puede ser difícil, por lo que decidimos probarlo. Al examinar empíricamente los tres tamaños, nos sale que el de 16K es la mejor opción.

En la quinta consulta hay que seleccionar datos de varias tablas, por lo que se nos ha ocurrido hacer un cluster. Esta estructura organiza físicamente los datos de manera que las filas que se incluyan en el cluster se almacenarán juntas en el disco. Introduciendo las columnas por las que se realizan la unión de las tablas en el cluster, mejora el rendimiento de la consulta.

En conclusión, el análisis se ha realizado con el objetivo de optimizar lo máximo posible las lecturas, ya que son predominantes en esta base de datos, sin fijarnos demasiado en las consecuencias que puedan acarrear a las escrituras.

3. Diseño físico y evaluación

3.1 PRIMERA CONSULTA:

```
C/C++
select * from posts where barcode='OII044550419282';
```

Plan 1

El plan 1 consiste en la ejecución inicial de la consulta

```
C/C++
-- PLAN 1
EXPLAIN PLAN SET statement_id = 'consulta1_plan1' FOR select *
from posts where barcode='OII044550419282';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |    30 | 34020 |   136   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| POSTS |    30 | 34020 |   136   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("BARCODE"='OII044550419282')

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:04:47
TIME CONSUMPTION (run): 34,4 milliseconds.
CONSISTENT GETS (workload):7085 acc
CONSISTENT GETS (weighted average):708,5 acc
```

Plan 2

El plan 2 consiste en la adición de índice para lograr una mayor eficiencia, además del uso de hints para indicarle a Oracle que debe utilizar el índice que hemos creado para el plan de la consulta. De esta manera cuando se vaya a ejecutar la consulta utilizará el índice creado.

```
C/C++
-- PLAN 2
CREATE INDEX index_barcode ON POSTS(BARCODE);


EXPLAIN PLAN SET statement_id = 'consulta1_plan2' FOR select
/*+index(index_barcode)*/ *
from posts where barcode='OII044550419282';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 1405381025

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                |     9 | 10206 |     8   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| POSTS          |     9 | 10206 |     8   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                          | INDEX BARCODE |     9 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   2 - access("BARCODE"='OII044550419282')

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:06:27
TIME CONSUMPTION (run): 36,1 milliseconds.
CONSISTENT GETS (workload):6596 acc
CONSISTENT GETS (weighted average):659,6 acc
```

Observamos que se reducen tanto los accesos como el coste de las operaciones realizadas gracias al índice.

3.2 SEGUNDA CONSULTA:

```
C/C++
select * from posts where product='Compromiso';
```

Plan 1

El plan 1 consiste en la ejecución inicial de la consulta

```
C/C++
-- PLAN 1
```

```
EXPLAIN PLAN SET statement_id = 'consulta2_plan1' FOR select *
from posts where product='Compromiso';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |    47 | 53298 |   136   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| POSTS |    47 | 53298 |   136   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("PRODUCT"='Compromiso')

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:07:53
TIME CONSUMPTION (run): 37,5 milliseconds.
CONSISTENT GETS (workload):6596 acc
CONSISTENT GETS (weighted average):659,6 acc
```

Plan 2

El plan 2 consiste en la adición de índice para lograr una mayor eficiencia. En este caso no utilizamos hints para demostrar que el programa selecciona automáticamente la forma más eficiente de ejecutarlas y por ello escoge la ejecución mediante el índice index_product.

```
C/C++
CREATE INDEX index_product ON POSTS(PRODUCT);


EXPLAIN PLAN SET statement_id = 'consulta2_plan2' FOR select *
from posts where product='Compromiso';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3395067892

-----------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |               |    57 | 64638 |    65   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| POSTS         |    57 | 64638 |    65   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                       | INDEX_PRODUCT |    57 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   2 - access("PRODUCT"='Compromiso')

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:08:39
TIME CONSUMPTION (run): 35,9 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Plan 3

El plan 3 consiste en la utilización de hints para indicarle al optimizador que debe usarlo para la consulta. Al añadir este hint, el optimizador de consultas dará prioridad al uso del índice especificado en lugar de considerar otras opciones de ejecución.

```
C/C++
EXPLAIN PLAN SET statement_id = 'consulta2_plan3' FOR select
/*+index(index_product)*/ *
from posts where product='Compromiso';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3395067892

-----------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                |    57 | 64638 |    65   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| POSTS          |    57 | 64638 |    65   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                         | INDEX PRODUCT |    57 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   2 - access("PRODUCT"='Compromiso')

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:09:30
TIME CONSUMPTION (run): 34,4 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

En resumen, observamos que no hay ninguna diferencia ya que el optimizador la realiza de la forma más eficiente sin necesidad de especificarlo.

3.3 TERCERA CONSULTA:

```
C/C++
select * from posts where score>=4;
```

Plan 1

El plan 1 consiste en la ejecución inicial de la consulta

```
C/C++
EXPLAIN PLAN SET statement_id = 'consulta3_plan1' FOR select *
from posts where score>=4;
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  1000 |  1107K|   136   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| POSTS |  1000 |  1107K|   136   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("SCORE">=4)

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:12:15
TIME CONSUMPTION (run): 32,7 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Plan 2

El plan 2 consiste en la creación de un índice para la columna score, ya que hará más eficiente la consulta

```
C/C++
CREATE INDEX index_score ON POSTS(SCORE);


EXPLAIN PLAN SET statement_id = 'consulta3_plan2' FOR select *
from posts where score>=4;
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  1173 |  1299K|   136   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| POSTS |  1173 |  1299K|   136   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("SCORE">=4)

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:12:46
TIME CONSUMPTION (run): 35,9 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Observamos que no ha habido ningún cambio ya que sigue haciendo el full scan de la tabla.

Plan 3

El plan 3 consiste en la utilización de un hint que debe usarlo para que sí que utilice el índice creado para la columna score.

```
C/C++
EXPLAIN PLAN SET statement_id = 'consulta3_plan3' FOR select /*+
INDEX(Posts index_score) */ * from posts where score>=4;
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 225626779

-----------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |              |  1173 |  1299K|   720   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| POSTS        |  1173 |  1299K|   720   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | INDEX SCORE |  1173 |       |     6   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   2 - access("SCORE">=4)

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:13:39
TIME CONSUMPTION (run): 34,5 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Observamos que es menos eficiente con el índice. Por esta razón el optimizador seguía haciendo full scan en lugar de utilizarlo.

Plan 4

El plan 4 consiste en la utilización de una vista materializada para esta consulta y este es el resultado:

```
C/C++
-- PLAN 4 (VISTA MATERIALIZADA con actualización)
CREATE MATERIALIZED VIEW LOG ON POSTS
```

```
WITH PRIMARY KEY
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW posts_view
REFRESH FAST ON COMMIT
AS
SELECT * FROM posts WHERE score >= 4;

EXPLAIN PLAN SET statement_id = 'consulta3_plan4' FOR select *
from posts_view where score>=4;
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 2894666430

-------------------------------------------------------------------------------------
| Id  | Operation             | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |            |  1173 |  1045K|    49   (0)| 00:00:01 |
|*  1 |  MAT VIEW ACCESS FULL| POSTS VIEW |  1173 |  1045K|    49   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - filter("SCORE">=4)
```

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:14:24
TIME CONSUMPTION (run): 32,8 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

No cambian los accesos porque se hacen sobre la tabla original, pero el plan 4 sí que se hace sobre la vista materializada y es por ello que se reducen significativamente los costes. La mejoría se debe en gran medida a que las vistas materializadas se almacenan en memoria RAM

o memoria principal y no en disco. Esto hace que sus accesos sean mucho más rápidos y por consiguiente mucho más eficientes.

Plan 5

El plan 5 consiste en la utilización de hilos paralelos, lo que aumenta las operaciones que se están realizando de forma simultánea y por tanto implica la reducción en los costes para realizar la consulta.

Para ello utilizamos la cláusula /*+ parallel(n) */ para sugerir al optimizador de consultas que ejecute una determinada consulta en paralelo con un grado de paralelismo específico, en este caso 64 ya que es el número que mejor funcionaba en las pruebas realizadas.

```
C/C++
-- PLAN 5 (CON MULTIHILO)
EXPLAIN PLAN SET statement_id = 'consulta3_plan5' FOR select
/*+ parallel(64) */ * from posts where score>=4;
```

COSTES:

PLAN_TABLE_OUTPUT

```
Plan hash value: 2973548082

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows  | Bytes | Cost (%CPU)| Time      |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |  1173 |  1299K|     2   (0)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR        |           |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)  | :TQ10000  |  1173 |  1299K|     2   (0)| 00:00:01 |  Q1,00 | P->S | QC (RAND)  |
|   3 |    PX BLOCK ITERATOR   |           |  1173 |  1299K|     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|*  4 |     TABLE ACCESS FULL  | POSTS     |  1173 |  1299K|     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-----------------------------------------------------------------------------------------------------------------------

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

   4 - filter("SCORE">=4)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - Degree of Parallelism is 64 because of hint
```

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:17:34
TIME CONSUMPTION (run): 36 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

3.4 CUARTA CONSULTA:

```
C/C++
select * from posts;
```

Plan 1

El plan 1 consiste en la ejecución inicial de la consulta

```
C/C++
-- PLAN 1
EXPLAIN PLAN SET statement_id = 'consulta4_plan1' FOR select *
from posts;
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  3000 |  3322K|   136   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| POSTS |  3000 |  3322K|   136   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:19:44
TIME CONSUMPTION (run): 32,9 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Plan 2

El plan 2 consiste en la utilización de índice sobre las claves primarias para comprobar si aumenta la eficiencia.

```
C/C++
CREATE INDEX index_posts ON POSTS(POSTDATE, USERNAME);


EXPLAIN PLAN SET statement_id = 'consulta4_plan2' FOR select /*+
INDEX(Posts index_posts) */* from posts;
```

COSTES:

PLAN_TABLE_OUTPUT

```
Plan hash value: 275700665

-----------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |             |  3000 |  3322K|  3424   (1)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| POSTS       |  3000 |  3322K|  3424   (1)| 00:00:01 |
|   2 |   INDEX FULL SCAN                     | INDEX POSTS |  3000 |       |    19   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:20:20
TIME CONSUMPTION (run): 32,8 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Plan 3

El plan 3 consiste en el empleo de hilos paralelos ya que, como se debe hacer un full scan de toda la tabla y no hay forma de evitarlo, la utilización de varios hilos simultáneos ayudará a reducir el coste de esta consulta en gran medida.

```
C/C++
-- PLAN 3 (MULTIHILO)
EXPLAIN PLAN SET statement_id = 'consulta4_plan3' FOR select /*+
parallel(64) */* from posts;
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 2973548082

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |  3000 |  3322K|     2   (0)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR       |           |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  3000 |  3322K|     2   (0)| 00:00:01 |  Q1,00 | P->S | QC (RAND)  |
|   3 |    PX BLOCK ITERATOR  |           |  3000 |  3322K|     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   4 |     TABLE ACCESS FULL| POSTS     |  3000 |  3322K|     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-----------------------------------------------------------------------------------------------------------------------

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - Degree of Parallelism is 64 because of hint
```

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:21:12
TIME CONSUMPTION (run): 31,3 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

Plan 4

El plan 4 consiste en la utilización de distintos tablespaces para comprobar así cuál es el que menos espacio libre deja en el cubo y por tanto el más eficiente para la tabla posts de nuestra base de datos.

```C/C++
-- PLAN 4 (TABLESPACE 16K)
ALTER TABLE POSTS MOVE TABLESPACE TAB_16K;
ALTER INDEX index_barcode REBUILD TABLESPACE TAB_16K;
ALTER INDEX index_product REBUILD TABLESPACE TAB_16K;
ALTER INDEX index_score REBUILD TABLESPACE TAB_16K;
ALTER INDEX index_posts REBUILD TABLESPACE TAB_16K;

EXPLAIN PLAN SET statement_id = 'consulta4_plan4' FOR select *
from posts;
```

TABLE SPACE 2K

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  3485 |  3859K|   483   (1)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| POSTS |  3485 |  3859K|   483   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:28:15
TIME CONSUMPTION (run): 43,9 milliseconds.
CONSISTENT GETS (workload):12029 acc
CONSISTENT GETS (weighted average):1202,9 acc
```

TABLE SPACE 8K

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  2984 |  3304K|   136   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| POSTS |  2984 |  3304K|   136   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:39:18
TIME CONSUMPTION (run): 34,5 milliseconds.
CONSISTENT GETS (workload):6149 acc
CONSISTENT GETS (weighted average):614,9 acc
```

TABLE 16K

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  3658 |  4050K|    89   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| POSTS |  3658 |  4050K|    89   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:52:41
TIME CONSUMPTION (run): 32,9 milliseconds.
CONSISTENT GETS (workload):5338 acc
CONSISTENT GETS (weighted average):533,8 acc
```

PCTFREE A 2 CON TABLESPACE 16K

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3606309814

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  3658 |  4050K|    89   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| POSTS |  3658 |  4050K|    89   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:44:38
TIME CONSUMPTION (run): 31,3 milliseconds.
CONSISTENT GETS (workload):5974 acc
CONSISTENT GETS (weighted average):597,4 acc
```

Por lo tanto, comprobamos que lo más eficiente es seguir de ahora en adelante con TABLESPACE 16k y PCTFREE 10, es decir, por defecto.

3.5 QUINTA CONSULTA:

```C/C++
select (quantity*price) as total, bill_town||'/'||bill_country
as place
from orders_clients join client_lines
using (orderdate,username,town,country)
```

```
where username='chamorro'; -- el valor de username es cualquiera
```

Plan 1

El plan 1 consiste en la ejecución inicial de la consulta

```
C/C++
EXPLAIN PLAN SET statement_id = 'consulta5_plan1' FOR
select (quantity*price) as total, bill_town||'/'||bill_country
as place
from orders_clients join client_lines
using (orderdate,username,town,country)
where username='chamorro';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 1654569925

-----------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |    12 |  2544 |   214   (1)| 00:00:01 |
|   1 |  NESTED LOOPS                 |                  |    12 |  2544 |   214   (1)| 00:00:01 |
|   2 |   NESTED LOOPS                |                  |    12 |  2544 |   214   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL          | CLIENT LINES     |    12 |  1080 |   205   (1)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | PK CLIENTORDERS  |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| ORDERS CLIENTS   |     1 |   122 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   3 - filter("CLIENT LINES"."USERNAME"='chamorro')
   4 - access("ORDERS CLIENTS"."ORDERDATE"="CLIENT LINES"."ORDERDATE" AND
              "ORDERS CLIENTS"."USERNAME"='chamorro' AND "ORDERS CLIENTS"."TOWN"="CLIENT LINES"."TOWN"
               AND "ORDERS CLIENTS"."COUNTRY"="CLIENT LINES"."COUNTRY")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
```

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:58:21
TIME CONSUMPTION (run): 31,2 milliseconds.
CONSISTENT GETS (workload):5353 acc
CONSISTENT GETS (weighted average):535,3 acc
```

Plan 2

El plan 2 consiste en la adición de un cluster para las tablas orders_clients y clients_lines ya que vamos a realizar un join y será más eficiente almacenar de forma conjunta los datos de estas dos tablas.

Script creación:

```C/C++
CREATE CLUSTER cluster_consulta_5(orderdate Date, username VARCHAR2(30), town VARCHAR2(45), country VARCHAR2(45));

CREATE TABLE Orders_Clients (
  orderdate     DATE,
  username      VARCHAR2(30),
  town          VARCHAR2(45),
  country       VARCHAR2(45),
  dliv_datetime DATE,
  bill_town     VARCHAR2(45) NOT NULL,
  bill_country  VARCHAR2(45) NOT NULL,
  discount      NUMBER(2) default(0),
```

```
                CONSTRAINT        pk_clientorders        PRIMARY
KEY(orderdate,username,town,country),
  CONSTRAINT fk_order_address FOREIGN KEY(username,town,country)
REFERENCES Client_Addresses,
                 CONSTRAINT        fk_order_bill        FOREIGN
KEY(username,bill_town,bill_country)
                REFERENCES Client_Addresses
)CLUSTER cluster_consulta_5(orderdate, username, town, country);

CREATE TABLE Client_Lines (
  orderdate     DATE,
  username      VARCHAR2(30),
  town          VARCHAR2(45),
  country       VARCHAR2(45),
  barcode       CHAR(15),
  price         NUMBER(12,2) NOT NULL,
  quantity      VARCHAR2(2) NOT NULL,
  pay_type      VARCHAR2(15) NOT NULL,
  pay_datetime  DATE,
  cardnum       NUMBER(20),
                 CONSTRAINT        pk_clientlines        PRIMARY
KEY(orderdate,username,town,country,barcode),
           CONSTRAINT     fk_clientlines_anonyorders     FOREIGN
KEY(orderdate,username,town,country)
              REFERENCES Orders_Clients ON DELETE CASCADE,
    CONSTRAINT  fk_clientlines_references  FOREIGN  KEY(barcode)
REFERENCES References,
  CONSTRAINT fk_lines_creditcard FOREIGN KEY(cardnum) REFERENCES
Client_Cards,
```

```
  CONSTRAINT D_clientcards CHECK (UPPER(pay_type)!='CREDIT CARD'
OR cardnum IS NOT NULL)
)CLUSTER cluster_consulta_5(orderdate, username, town, country);

CREATE INDEX idx_cluster_consulta_5 ON CLUSTER
cluster_consulta_5;
```

Script plan de ejecución:

```
C/C++
EXPLAIN PLAN SET statement_id = 'consulta5_plan2' FOR
select (quantity*price) as total, bill_town||'/'||bill_country
as place
from orders_clients join client_lines
using (orderdate,username,town,country)
where username='chamorro';
```

COSTES:

```
PLAN_TABLE_OUTPUT
Plan hash value: 552299328

-----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                         |    97 | 20564 |    29   (0)| 00:00:01 |
|   1 |  NESTED LOOPS         |                         |    97 | 20564 |    29   (0)| 00:00:01 |
|   2 |   TABLE ACCESS CLUSTER| ORDERS CLIENTS          |    97 | 11834 |    28   (0)| 00:00:01 |
|*  3 |    INDEX SKIP SCAN    | IDX CLUSTER CONSULTA 5  |    97 |       |    26   (0)| 00:00:01 |
|   4 |   TABLE ACCESS CLUSTER| CLIENT LINES            |     1 |    90 |     1   (0)| 00:00:01 |
|*  5 |    INDEX UNIQUE SCAN  | IDX CLUSTER CONSULTA 5  |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   3 - access("ORDERS CLIENTS"."USERNAME"='chamorro')
       filter("ORDERS CLIENTS"."USERNAME"='chamorro')
   5 - access("ORDERS CLIENTS"."ORDERDATE"="CLIENT LINES"."ORDERDATE" AND
              "CLIENT LINES"."USERNAME"='chamorro' AND "ORDERS CLIENTS"."TOWN"="CLIENT LINES"."TOWN"
              AND "ORDERS CLIENTS"."COUNTRY"="CLIENT LINES"."COUNTRY")

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

ACCESOS:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 18:27:10
TIME CONSUMPTION (run): 50 milliseconds.
CONSISTENT GETS (workload):4423 acc
CONSISTENT GETS (weighted average):442,3 acc
```

Observamos que con el uso del cluster el tiempo de ejecución aumenta ligeramente, pero el número de accesos sí que disminuye de forma considerable. Es por ello que creemos que sigue siendo más efectivo mantener el uso del cluster para almacenar de forma conjunta las dos tablas implicadas en el join de la última consulta.

Además el motivo por el que se han reducido tanto los accesos (100 por iteración) se debe a que esta consulta es la que se ejecuta con mayor frecuencia relativa (0,5) por lo que cualquier cambio que se realice sobre ella tiene un gran impacto en la carga de trabajo (*workload*).

4. Conclusiones Finales

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 13:04:47
TIME CONSUMPTION (run): 34,4 milliseconds.
CONSISTENT GETS (workload):7085 acc
CONSISTENT GETS (weighted average):708,5 acc
```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 01/05/2024 18:27:10
TIME CONSUMPTION (run): 50 milliseconds.
CONSISTENT GETS (workload):4423 acc
CONSISTENT GETS (weighted average):442,3 acc
```

Como podemos observar en la foto de la izquierda y con la configuración inicial realizamos 708,5 accesos por cada iteración del procedimiento, mientras que al final de la práctica hemos conseguido reducirlo a 442,3, es decir, alrededor de un 38 por ciento de mejora gracias al empleo de índices, hints, hilos paralelos y clusters.

Consideramos que la práctica ha sido bastante útil y productiva para llegar a entender de una mejor manera como funciona la optimización de una base de datos. Esta empieza desde el uso de hints, cluster, multihilos, hasta modificar el tamaño de los cubos para que se adapten bien al tamaño ideal.

Con respecto a los resultados obtenidos, creemos que han sido bastante positivos, ya que hemos conseguido bajar el número de accesos en casi la mitad. Esto se traduce en unas consultas notablemente más productivas y por tanto mejores.

Además, hemos visto cómo en el desarrollo de la práctica la herramienta de Oracle nos ha facilitado mucho la visualización de tablas, índices o vistas que hay activos en la BBDD, así como el uso de filtros y ordenación propiamente integrados.

En cuanto al desempeño realizado en las prácticas, opinamos que las dos primeras contienen un volumen y dificultad mayor debido a la gran cantidad de tareas que suponen, realizar el grafo de la BBDD, estructurar las tablas y todos los insert y luego la creación de

consultas, vistas, procedimientos o triggers sobre ellas. Aunque hemos sentido cómo a lo largo del curso no ha sido tan complicado gracias a que hemos ido aprendiendo poco a poco con los laboratorios y primeras prácticas. Gracias a esto, hemos desarrollado una buena base sobre SQL que ha llevado a que esta última práctica no suponga un problema y que la hayamos podido completar en menos tiempo que las demás.

Para terminar, consideremos que la asignatura está bien estructurada en términos generales, teniendo tanto unas buenas clases de teoría como de ejercicios junto con unos laboratorios productivos que nos ayudan a entender mejor los conceptos explicados. No obstante sí consideramos que hemos sentido una ligera falta de retroalimentación y comentarios acerca de las prácticas a lo largo del curso. Esto en ciertas ocasiones nos causaba incertidumbre y preocupación por la posibilidad de estar arrastrando un error común durante las tres prácticas y no ser conscientes de ello. También queremos mencionar algunas recomendaciones que harían que las prácticas fueran más fáciles para los alumnos sin dejar de lado el trabajo que se exige. Creemos que a la hora de trabajar con las bases de datos que se nos otorgan, estas tiene excesivos datos y hace que muchas veces sature el servidor y nos veamos excesivamente perjudicados. Consideramos que si algunas tablas en vez de tener 50.000 datos tuviesen 10.000, no cambiaría nada en el desarrollo de la práctica como tal, pero sí haría que fuese más rápida. Esto se debe a que en un gran número de ocasiones tenemos que esperar largas colas para poder ejecutar una simple consulta debido a la gran cantidad de alumnos que se encuentran haciendo peticiones al servidor a la vez.

No obstante y para finalizar, consideramos que las tres prácticas han sido productivas y han conseguido que tengamos un conocimiento mucho más amplio acerca de las bases de datos que el que teníamos al inicio del curso.