PRÁCTICA 4 – CONSULTAS

# APARTADO 1:

-- CONSULTA 0 --  
delete plan\_table;  
EXPLAIN PLAN INTO plan\_table  
FOR (select \* from cliente where DNI <> '00000005') union  
(select \* from moroso where NombreC = 'Client E');  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

-- CONSULTA 1 --  
delete plan\_table;

EXPLAIN PLAN  
INTO plan\_table  
FOR (select \* from cliente where DNI < '00000005') union  
(select \* from moroso where NombreC = 'Client E');  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

-- CONSULTA 2 --  
delete plan\_table;

EXPLAIN PLAN  
INTO plan\_table  
FOR (select \* from cliente where DNI = '00000005') union  
(select \* from moroso where NombreC = 'Client E');  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

-- CONSULTA 3 -- anidados ----  
delete plan\_table;  
EXPLAIN PLAN  
INTO plan\_table  
FOR select \* from cliente where DNI in  
(select DNI from moroso where NombreC = 'Client E');  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

-- CONSULTA 4 – ----  
delete plan\_table;  
EXPLAIN PLAN  
INTO plan\_table  
FOR (select \* from cliente where dni in  
(select dni from invierte));  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

---- CONSULTA 5 ---  
delete plan\_table;   
EXPLAIN PLAN  
INTO plan\_table  
FOR (select \* from cliente where dni in  
(select dni from invierte where cantidad < 30000));  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

---- CONSULTA 6 ---  
delete plan\_table;  
EXPLAIN PLAN INTO plan\_table  
FOR(  
select distinct NombreC  
from Cliente, Compras, Invierte  
where Cliente.DNI = Invierte.DNI and Invierte.NombreE = 'Empresa 55' and  
Compras.DNI = Cliente.DNI and Compras.Importe >1000);  
select operation,options,object\_name,cost,cardinality,parent\_id,id  
from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;

# APARTADO 2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Consulta | operation | options | object\_name | cost | cardinality | parent\_id | id |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| consulta 0 |  |  |  |  |  |  |  |
|  | SORT | UNIQUE |  | 8 | 7 | 0 | 1 |
|  | UNION-ALL |  |  |  |  | 1 | 2 |
|  | TABLE ACCESS | FULL | CLIENTE | 3 | 6 | 2 | 3 |
|  | TABLE ACCESS | FULL | MOROSO | 3 | 1 | 2 | 4 |
|  |  |  |  |  |  |  |  |
| consulta 1 |  |  |  |  |  |  |  |
|  | SORT | UNIQUE |  | 8 | 5 | 0 | 1 |
|  | UNION-ALL |  |  |  |  | 1 | 2 |
|  | TABLE ACCESS | FULL | CLIENTE | 3 | 4 | 2 | 3 |
|  | TABLE ACCESS | FULL | MOROSO | 3 | 1 | 2 | 4 |
|  |  |  |  |  |  |  |  |
| consulta 2 |  |  |  |  |  |  |  |
|  | SORT | UNIQUE |  | 6 | 2 | 0 | 1 |
|  | UNION-ALL |  |  |  |  | 1 | 2 |
|  | TABLE ACCESS | BY INDEX ROWID | CLIENTE | 1 | 1 | 2 | 3 |
|  | INDEX | UNIQUE SCAN | SYS\_C0034221 | 1 | 1 | 3 | 4 |
|  | TABLE ACCESS | FULL | MOROSO | 3 | 1 | 2 | 5 |
|  |  |  |  |  |  |  |  |
| consulta 3 |  |  |  |  |  |  |  |
|  | NESTED LOOPS |  |  |  |  | 0 | 1 |
|  | NESTED LOOPS |  |  | 4 | 1 | 1 | 2 |
|  | TABLE ACCESS | FULL | MOROSO | 3 | 1 | 2 | 3 |
|  | INDEX | UNIQUE SCAN | SYS\_C0034221 | 0 | 1 | 2 | 4 |
|  | TABLE ACCESS | BY INDEX ROWID | CLIENTE | 1 | 1 | 1 | 5 |
|  |  |  |  |  |  |  |  |
| consulta 4 |  |  |  |  |  |  |  |
|  | HASH JOIN | SEMI |  | 6 | 3 | 0 | 1 |
|  | TABLE ACCESS | FULL | CLIENTE | 3 | 7 | 1 | 2 |
|  | INDEX | FAST FULL SCAN | CLAVE\_INVIRTE\_PRIM | 2 | 8 | 1 | 3 |
|  |  |  |  |  |  |  |  |
| consulta 5 |  |  |  |  |  |  |  |
|  | NESTED LOOPS |  |  |  |  | 0 | 1 |
|  | NESTED LOOPS |  |  | 5 | 1 | 1 | 2 |
|  | SORT | UNIQUE |  | 3 | 1 | 2 | 3 |
|  | TABLE ACCESS | FULL | INVIERTE | 3 | 1 | 3 | 4 |
|  | INDEX | UNIQUE SCAN | SYS\_C0034221 | 0 | 1 | 2 | 5 |
|  | TABLE ACCESS | BY INDEX ROWID | CLIENTE | 1 | 1 | 1 | 6 |
|  |  |  |  |  |  |  |  |
| consulta 6 |  |  |  |  |  |  |  |
|  | HASH | UNIQUE |  | 10 | 2 | 0 | 1 |
|  | HASH JOIN |  |  | 9 | 2 | 1 | 2 |
|  | HASH JOIN |  |  | 7 | 4 | 2 | 3 |
|  | TABLE ACCESS | FULL | COMPRAS | 3 | 4 | 3 | 4 |
|  | TABLE ACCESS | FULL | CLIENTE | 3 | 7 | 3 | 5 |
|  | INDEX | FAST FULL SCAN | CLAVE\_INVIRTE\_PRIM | 2 | 4 | 2 | 6 |

b.- *Contestar a las siguientes preguntas basándote en las operaciones de los resultados anteriores:*

b.0 Comparando CONSULTA 0 y CONSULTA 1:

¿Porqué hay esa diferencia de operaciones y opciones?

¿Qué reglas aplica en ambas consultas para que no coincidan las operaciones que realizan?

(Repasa la teoría : Reglas de Optimización en Selección)

La diferencia entre las consultas se debe a las condiciones de selección. En la consulta 1 se utiliza el operador "<" en la condición DNI < '00000005' para seleccionar todos los clientes cuyo DNI sea menor que '00000005', mientras que en la consulta 2 se utiliza el operador "=" en la condición DNI = '00000005' para seleccionar solo el cliente cuyo DNI sea exactamente igual a '00000005'.

En términos de reglas de optimización de selección, para consulta 1, el operador de “no igualdad” ("<"), significa que el optimizador no puede utilizar índices en la columna DNI para realizar la selección. En la consulta 2, el operador de "igualdad" ("="), permite utilizar índices en la columna DNI para realizar la selección. Por lo tanto, la consulta 2 tiene más eficiencia mediante el uso de índices en comparación con la consulta 1. La restricción de filas en la tabla Cliente explica estas diferencias: el Index unique scan permite acceder a una sola fila por índice único en Cliente.

b.1 Comparando CONSULTA 1 y CONSULTA 2:

¿Porqué hay esa pequeña diferencia de opciones?

¿Qué reglas aplica en ambas consultas para que no coincidan las opciones que realizan?

En la consulta 1, se accede a todas las filas de la tabla Cliente para averiguar la condición mientras que en la consulta 2, el acceso se hace a solo una fila mediante un índice único en Cliente (DNI es clave primaria). Así, el acceso a las filas de Cliente es “Full” para la consulta 1 y “by index rowID” para la consulta 2.

b.2 Porqué en CONSULTA 1 no accede por índice a MOROSO ?

(Repasa la teoría : Reglas de Optimización, Combinación dos tablas)

En la consulta 1, no se puede acceder al índice MOROSO porque el atributo usado en la comparación de igualdad, NombreC, no es clave primaria. Debido a que el nombre no es una clave primaria, es posible que existan elementos duplicados. El optimizador no puede acceder a la tabla MOROSO por índice y debe realizar una búsqueda de tabla completa.

b.3 Porqué en CONSULTA 2 sí accede por el índice a CLIENTE, cual es el nombre del índice ?

El atributo en el que se hace la comparación de igualdad (DNI) es clave primaria de CLIENTE, así se puede acceder por índice. El nombre del índice es SYS\_C0034221.

b.4 Porqué en CONSULTA 0 no accede por el índice a CLIENTE ?

No se puede acceder por el índice a CLIENTE debido a la condición "DNI <> '00000005'”: el operador de desigualdad (“<>”) hace que los valores de DNI deben ser comparados con un rango de valores, y no con un valor específico. La consulta tiene más de una solución. Para hacer esto, se debe realizar una búsqueda de tabla completa para seleccionar las filas correspondientes a la regla de desigualdad.

b.5 Pensando solo en los tipos de accesos:¿Cuál es más eficiente y porqué, CONSULTA 0 o la 1?

Las dos consultas tienen la misma eficiencia porque deben acceder a todas las filas de la tabla CLIENTE y de la tabla MOROSO.

c.- Solo para la CONSULTA 6:

- Dibuja en el fichero word el árbol del Plan de Ejecución, como el (e) de la diapositiva 22 después del paso 5. Usa como base lo que te ha devuelto el Explain: incluye el núm. de operación, nombre en cada operación con su opción y las tabla o índice sobre los que se aplica.

Añade las condiciones de selección y las proyecciones de los atributos en el lugar donde deberían estar para obtener el árbol optimizado del paso 5 en diapositiva 22 de Teoría.

π NombreC

|

|><| Cliente.DNI = Invierte.DNI

| |

π Cliente.DNI π Invierte.DNI

Cliente.NombreC |

| |

|><| Compras.DNI = Cliente.DNI |><| Invierte.NombreE = ‘Empresa 55’

| | |

π Cliente.DNI π Cliente.DNI.NombreC Invierte

| |

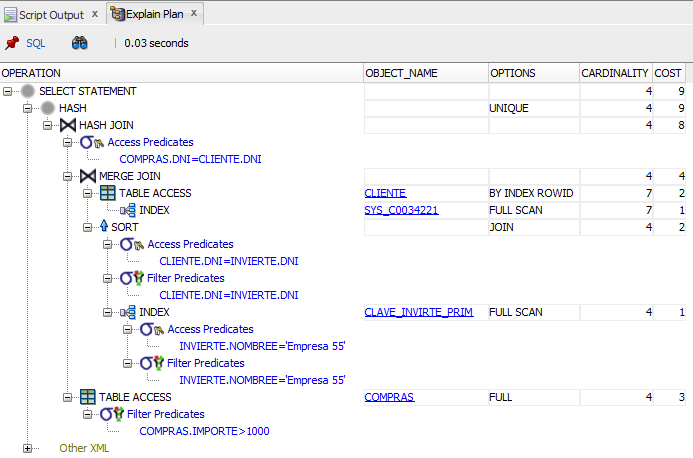
|><| Compras.Importe > 1000 Cliente

|

Compras

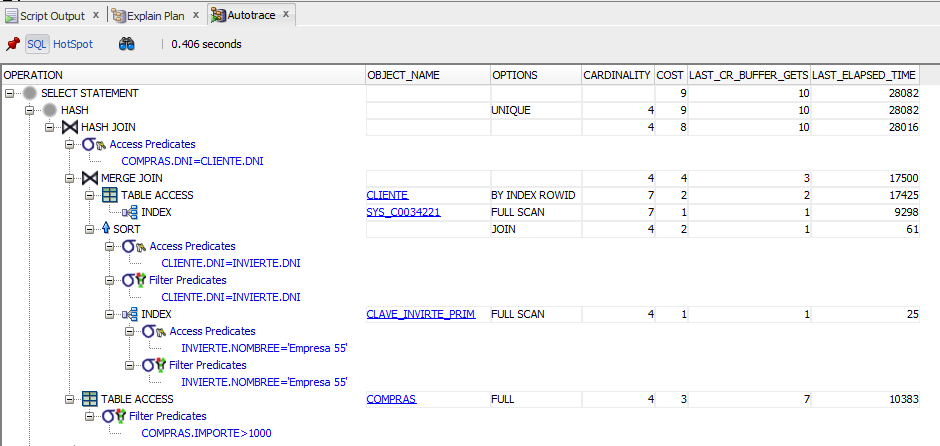
d.- Compara tu árbol con el árbol que muestra SQLDeveloper al marcar la consulta y pulsa F10.

Indica qué diferencias hay. (sin contar las selecciones y las proyecciones que has añadido para optimizarlo)



Este árbol muestra una estructura jerárquica que agrupa los datos en función de las relaciones entre las tablas y los valores de las columnas. Por otro lado, nuestro árbol del plan de ejecución muestra cómo el motor de la base de datos procesa la consulta. Además muestra una estructura jerárquica que representa las diferentes etapas del procesamiento de la consulta, como la selección de tablas, la aplicación de filtros, la ordenación y la agrupación. Se utiliza para optimizar el rendimiento de las consultas, ya que permite analizar cómo se está ejecutando la consulta e identificar posibles cuellos de botella o áreas de mejora en la consulta o en el diseño de la base de datos.

e.- Genera otro árbol con F6 y compáralo con el obtenido usando F10. ¿Son ambas funciones iguales? ¿Qué significan las diferencias?



Las funciones son diferentes : en la tabla al lado del árbol hay dos columnas más donde hacen referencia a los accesos que ha ido realizando y pone un total de accesos a esa tabla y su último acceso.

f.- Exporta a un fichero html el árbol del plan (botón dcho sobre raíz de salida de F10).

[Fichero del árbol del plan](https://drive.google.com/file/d/1A_kc8TWI1Ahy6tKL93yvzJGXP5ixUYPJ/view?usp=share_link)

g.- Dibuja en word el árbol de CONSULTA 3 igual que para la consulta 6, y obtén el árbol con F10:

π DNI Nombre Direccion

|

|><| Cliente.DNI = Moroso.DNI

| |

π Cliente.DNI π Moroso.DNI

Cliente.NombreC |

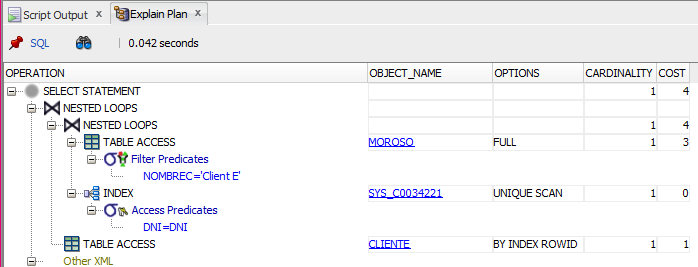
Cliente.Direccion |

| |

Cliente |><| Moroso.NombreC = ‘Cliente E’

|

Moroso



g.1.-¿Salen NESTED LOOPS?¿qué encuentras de particular en los NESTED LOOPS?

En la consulta 3 hay dos nested loops. Estos bucles anidados son una buena alternativa para la unión, permiten recuperar los datos de la tabla después de tener un acceso al índice.

g.2.- ¿Qué hace cada uno?

El primer bucle anidado comprueba las filas de la tabla Moroso para buscar DNI del cliente cuyo nombre es “Cliente E”. Y el segundo bucle comprueba las filas de la tabla Cliente que contienen este DNI.

g.3.- ¿Porqué hay dos?

Se requieren dos bucles anidados para comparar si los elementos están presentes en ambas tablas Cliente y Moroso.

j.- Dibuja en word el árbol de CONSULTA 5 igual que para en g.- (para pistas usa F10)

π DNI Nombre Direccion

|

|><| Cliente.DNI = Invierte.DNI

| |

π Cliente.DNI π Invierte.DNI

Cliente.NombreC |

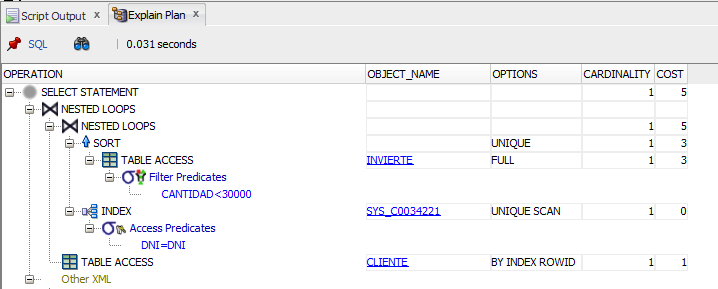
Cliente.Direccion |

| |

Cliente |><| Invierte.cantidad < 30000

|

Invierte



j.1.- ¿El sort : porqué atributo y de qué tabla clasifica?

El Sort se utiliza para ordenar la cantidad de la tabla Invierte.

j.2.- ¿Qué elementos participan en el Nested Loop de la operación Id\_Fi 2 ?

Son las operaciones Index y Sort.

j.3.- ¿porqué es una optimización hacerlo?

Con el Sort, las cantidades están ordenadas, así cuando el índice llega a la cantidad 30000, no va a recorrer más. Permite disminuir el número de filas recorridas.

# APARTADO 3:

delete plan\_table;

EXPLAIN PLAN INTO plan\_table

FOR [escribe aquí la consulta];  
select operation,options,object\_name,cost,cardinality, bytes, parent\_id,id from plan\_table connect by prior id=parent\_id  
start with id = 1 order by id;  
--------------------------------------------------------  
-- CONSULTA C1 –  
select PELISAHORA.ID  
from PELISAHORA, PELISHIST  
where PELISAHORA.ID = PELISHIST.ID;  
-- CONSULTA C2 --  
select PELISAHORA.DESCRIPCION  
from PELISAHORA, PELISHIST  
where PELISAHORA.DESCRIPCION = PELISHIST.DESCRIPCION;  
-- CONSULTA C3 --  
select PELISAHORA.TITULO  
from PELISAHORA, PELISHIST  
where PELISAHORA.TITULO = PELISHIST.TITULO;  
-- CONSULTA C4 --  
select PELISAHORA.TITULO  
from PELISAHORA  
where PELISAHORA.TITULO in (select PELISHIST.TITULO from PELISHIST);  
-- CONSULTA C5 --  
select PELISAHORA.TITULO  
from PELISAHORA  
where PELISAHORA.TITULO in  
(select PELISHIST.TITULO from PELISHIST  
where PELISAHORA.TITULO = PELISHIST.TITULO);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Consulta | Coste (total) | Num Filas | Num Operaciones | Bytes | ¿Es fiable? |
| C1 | 1 | 285 | 3 | 1708 | 294 líneas reales, card 285 -> FIABLE |
| C2 | 88 | 807 | 3 | 2098074 |  |
| C3 | 3 | 664 | 3 | 16926 |  |
| C4 | 4 | 1185 | 4 | 27867 |  |
| C5 | 4 | 1185 | 4 | 27867 |  |

a.1) Sobre C1: ¿Porqué no accede a la tabla PELISHIST?

Con el atributo "ID" indexado, no es necesario acceder a toda la tabla PELISHIST para obtener una fila específica. Por lo tanto, se puede acceder directamente a la fila requerida sin tener que buscar en toda la tabla.

a.2) Sobre C1: ¿Cuál es el criterio principal aplicado para escoger un plan de ejecución u otro?

El criterio principal para escoger un plan de ejecución en una base de datos es la optimización de la consulta en términos de tiempo de respuesta y coste (número de operaciones, número de filas ejecutadas, utilización de CPU…).

a.3) Compara C1 y C2: ¿Qué problema hay con los índices para que el coste sea tal alto en C2?

La diferencia entre C1 y C2 es que C1 tiene un atributo indexado (“ID”), lo que significa que no necesita recorrer toda la tabla para obtener los resultados necesarios, mientras que en C2 el atributo no está indexado (“Descripción”), lo que obliga a la base de datos a revisar toda la tabla cada vez que se hace una consulta. Así, sin la indexación, C2 tiene un coste más alto.

a.4) Compara C2 y C3: ¿Porqué, en C2, accede a las tablas completas?

La C2 recorre todas las tablas mientras que la C3 no. En efecto, hemos visto que C2 debe recorrer las tablas completas porque el atributo “Descripción” no está indexado. Mientras que C3 se hace con un atributo indexado “Título” así, no necesita recorrer todas las tablas.

a.5) Compara C2 y C3: ¿Porqué, en C2, el coste es tal alto?

Debido a que el atributo “Descripción” no está indexado en C2, la base de datos necesita realizar operaciones de HASH JOIN con otras tablas para obtener la información necesaria. Como resultado, la consulta tiene un alto costo debido a que se accede a las tablas completas varias veces.

a.6) Compara C2 y C3: ¿En C2, porqué el Hash Join solo usa 142 filas si ha leído las 521 filas de pelishist?

La consulta C2 realiza una selección de la tabla PELISAHORA, la cual tiene un total de 143 filas y un selección de la tabla PELISHIST que tiene 522 filas. Para hacer la consulta con una tabla pequeña a otra bastante más grande, se utiliza la tabla más pequeña para crear una tabla hash en memoria con la clave join, como parte de la operación de HASH JOIN que se lleva a cabo.

a.7) Compara C3 y C4: ¿Porqué ambas NO hacen las mismas operaciones de bajo nivel?

La consulta C4 realiza una fusión de las tablas a través de la cláusula WHERE, lo que lleva a la operación SORT. Mientras que en la consulta C3, se fusionan las tablas a través de la cláusula FROM, por lo que no es necesario realizar la operación SORT. Esta diferencia en la forma en que se fusionan las tablas hace que no se realizan las mismas operaciones de bajo nivel en ambas consultas.

a.8) Compara C4 y C5: ¿Porqué ambas SI hacen las mismas operaciones de bajo nivel?

Las consultas C4 y C5 realizan la misma función pero tienen diferentes estructuras. La diferencia está en que en la cláusula WHERE de C4 se incluye una subconsulta que utiliza un atributo indexado.Y esto hace posible que la consulta no tenga un costo adicional.

b)- *Explica cuál de las consultas es más eficiente si tienes en cuenta:*

b.1) Sólo el coste y las filas usadas.

En cuanto al coste y las filas usadas, vemos que la consulta C1 es mucho más eficiente que las otras. Recorre menos filas gracias al atributo indexado “ID”.

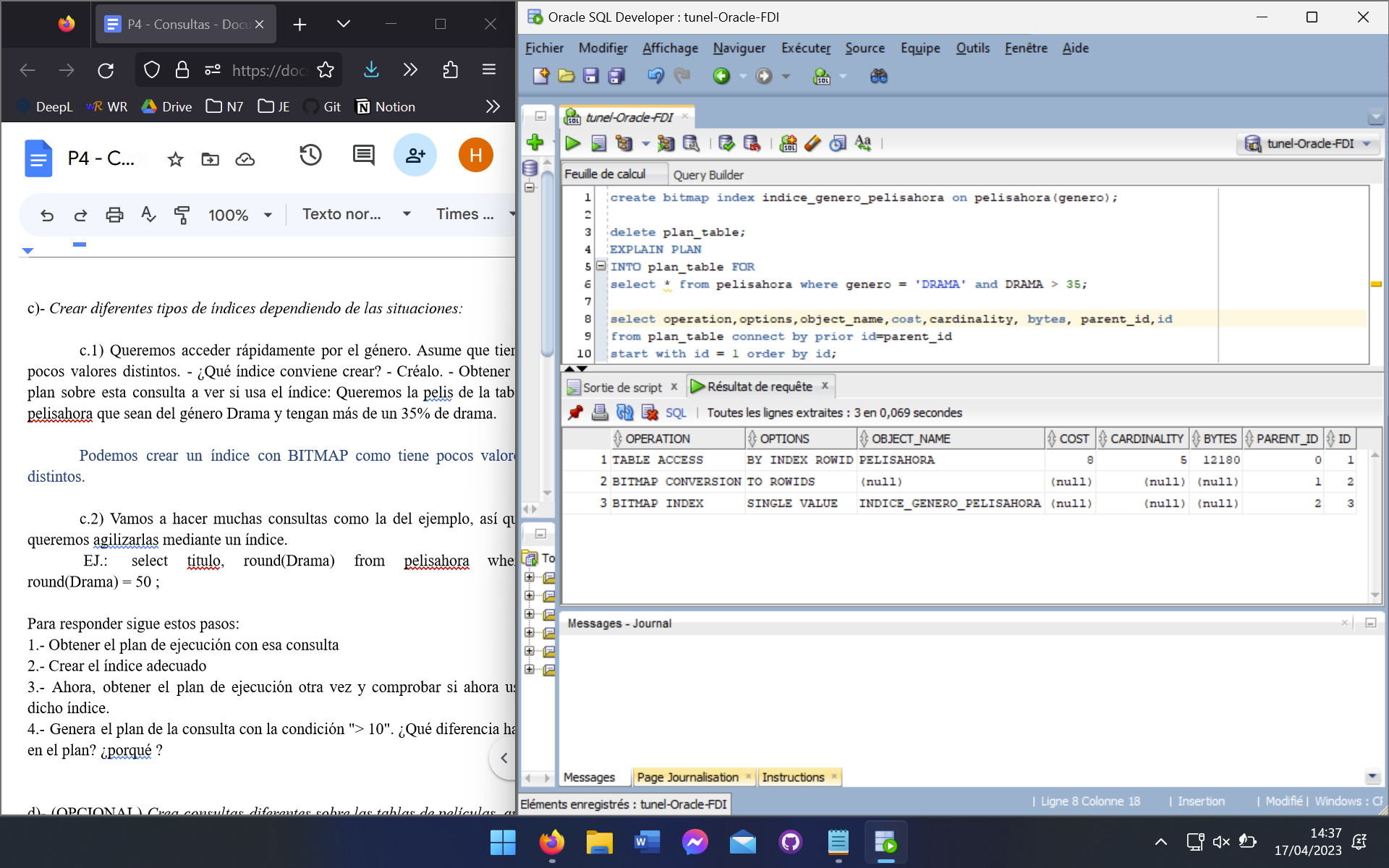
b.2) Sólo el coste y los bytes usados.

En cuanto al coste y a los bytes usados, la consulta C1 es también la más eficiente. Recorre menos filas y el coste es menor así que necesita menos bytes.

c)- *Crear diferentes tipos de índices dependiendo de las situaciones:*

c.1) Queremos acceder rápidamente por el género. Asume que tiene pocos valores distintos. - ¿Qué índice conviene crear? - Créalo. - Obtener el plan sobre esta consulta a ver si usa el índice: Queremos la pelis de la tabla pelisahora que sean del género Drama y tengan más de un 35% de drama.

Podemos crear un índice con BITMAP como tiene pocos valores distintos.

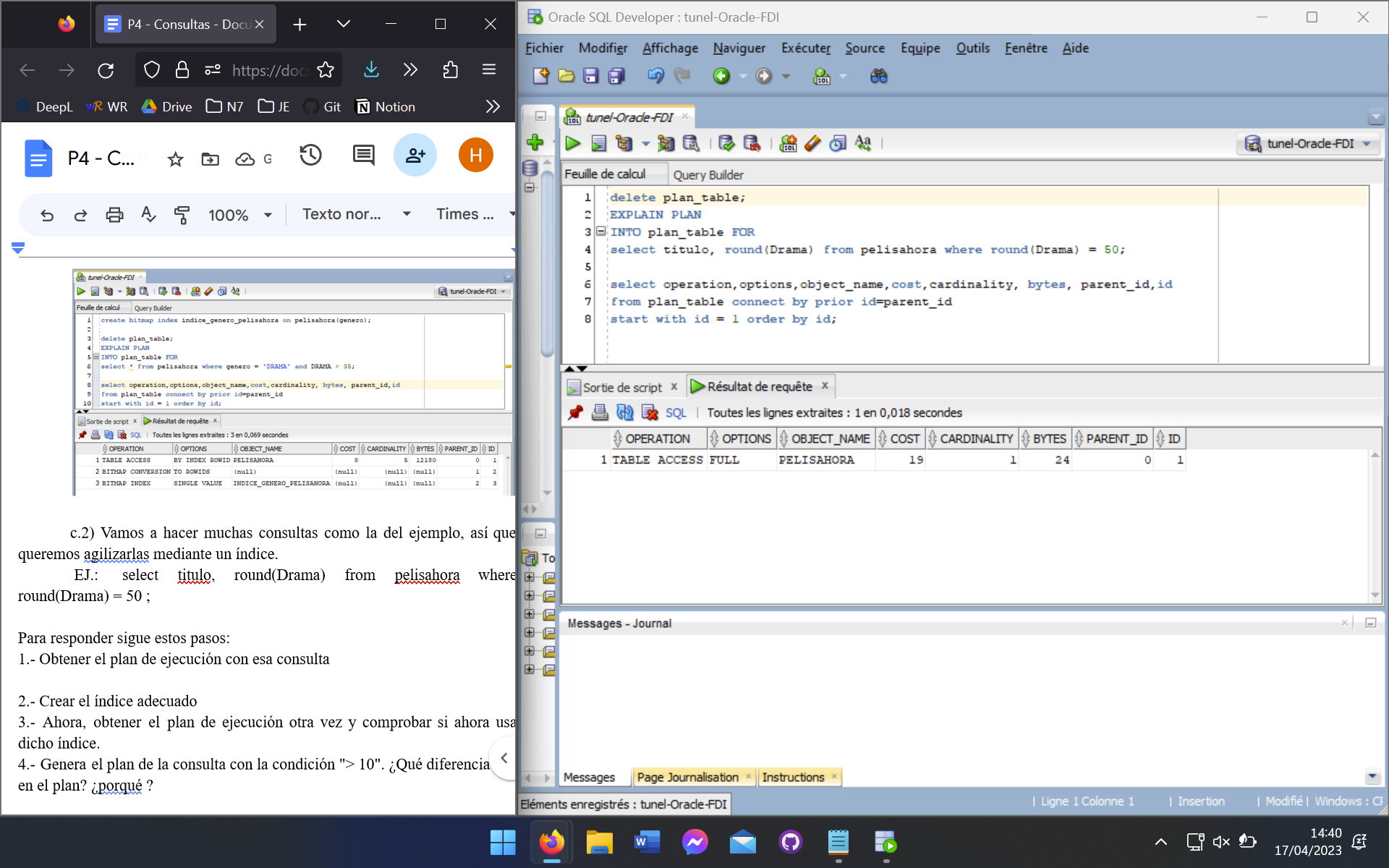


c.2) Vamos a hacer muchas consultas como la del ejemplo, así que queremos agilizarlas mediante un índice.

EJ.: select titulo, round(Drama) from pelisahora where round(Drama) = 50 ;

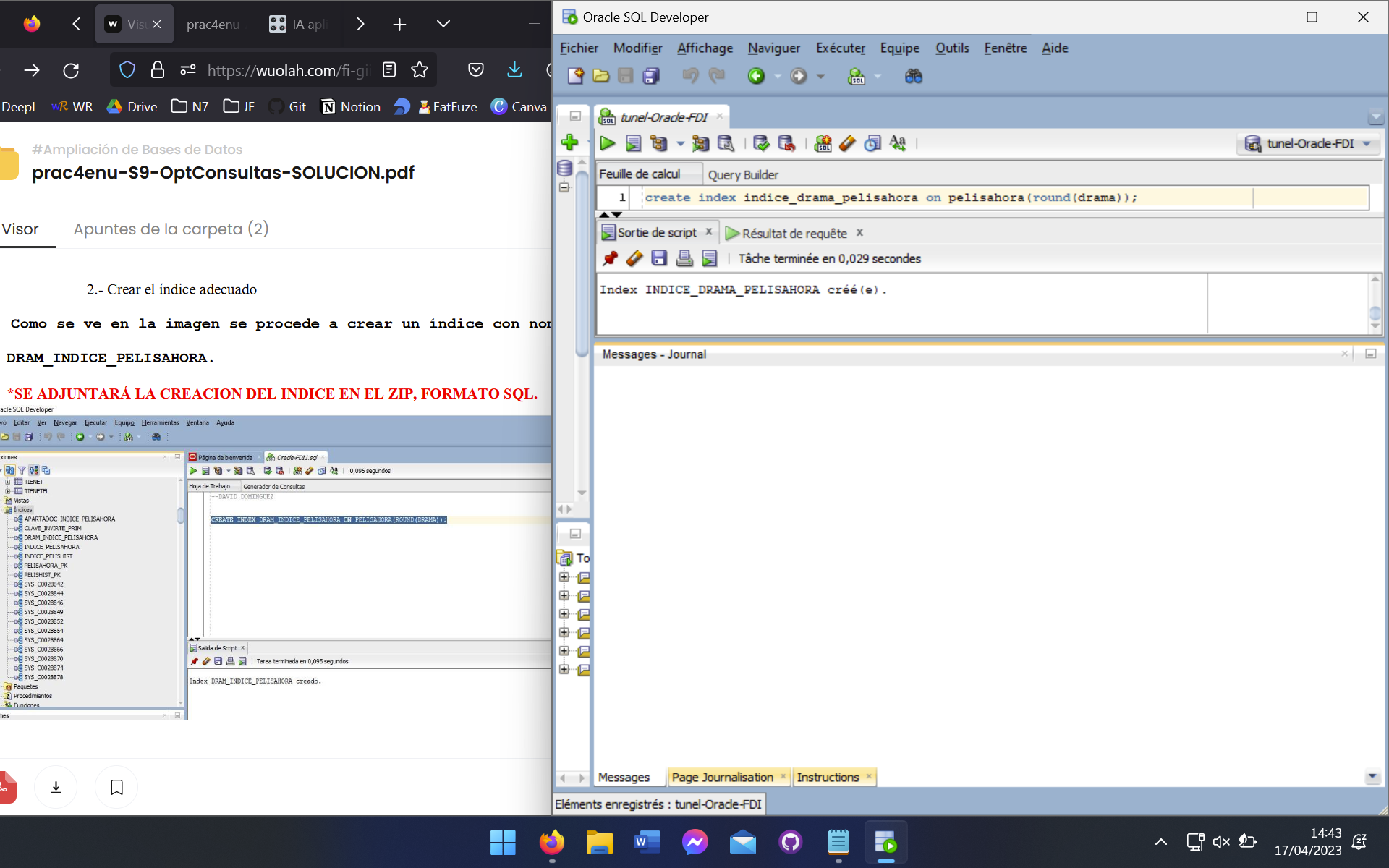
Para responder sigue estos pasos:

1.- Obtener el plan de ejecución con esa consulta



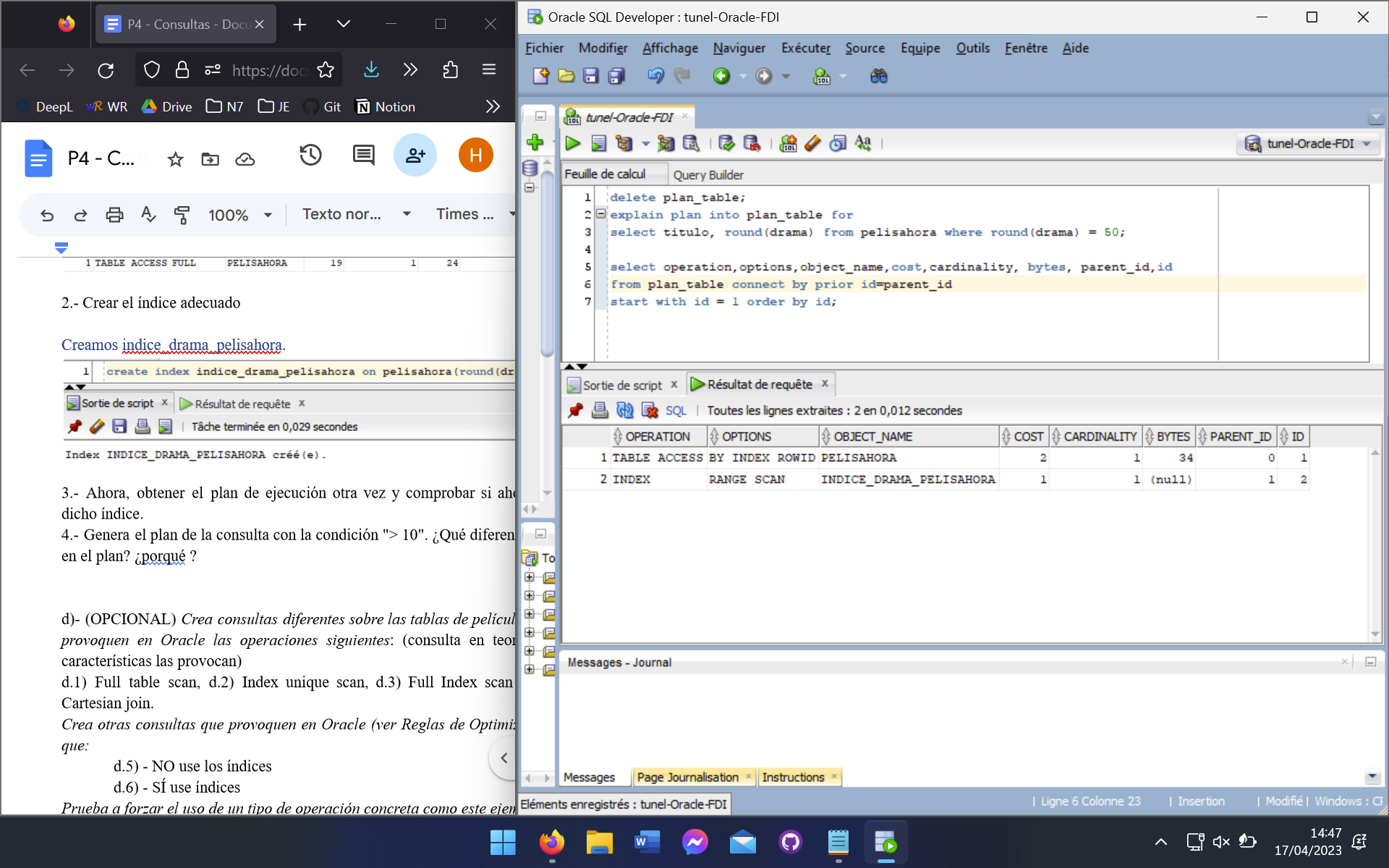
2.- Crear el índice adecuado

Creamos indice\_drama\_pelisahora.



3.- Ahora, obtener el plan de ejecución otra vez y comprobar si ahora usa dicho índice.

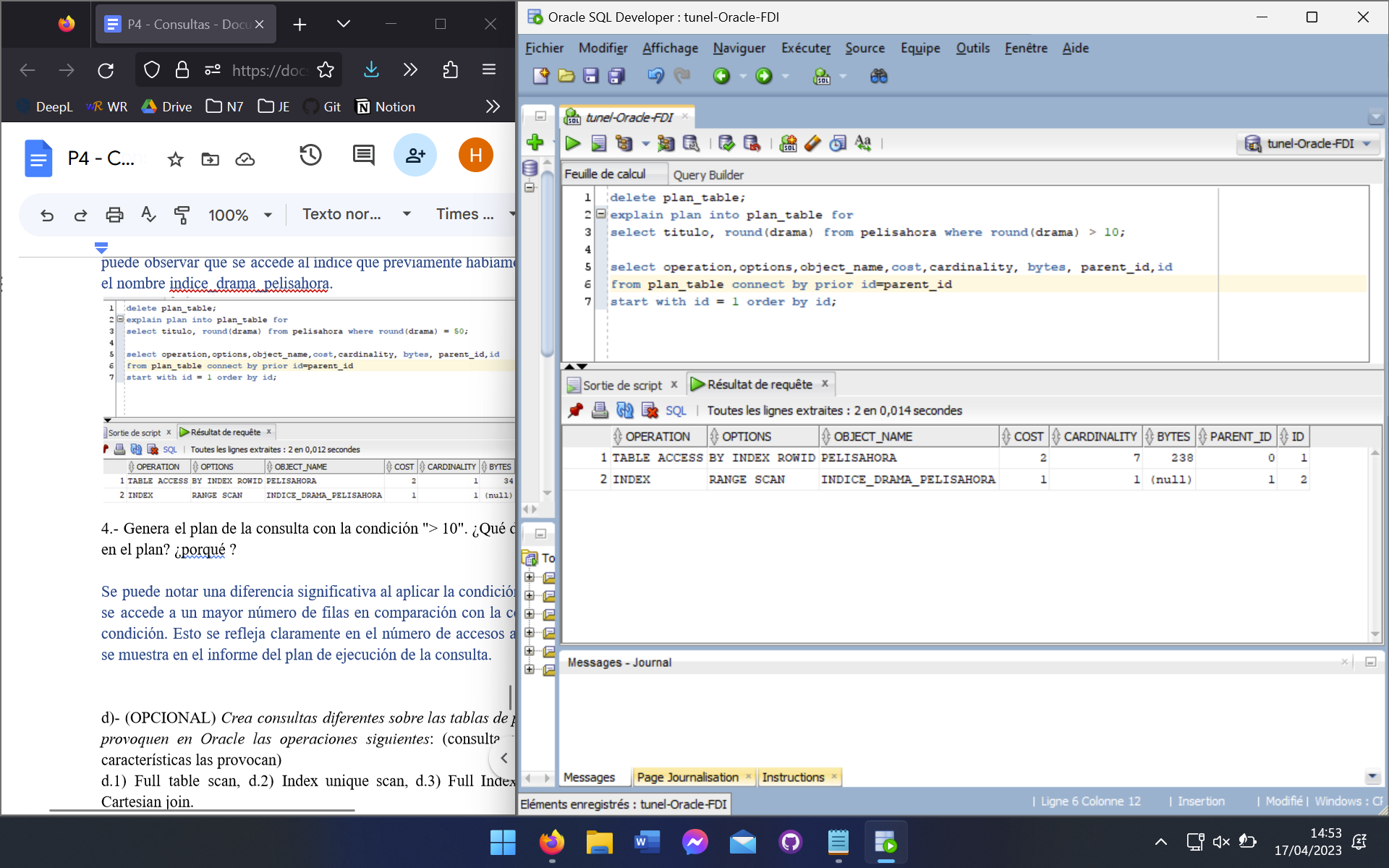
Después de haber introducido el nuevo índice, hemos ejecutado el plan y se puede observar que se accede al índice que previamente habíamos creado con el nombre indice\_drama\_pelisahora.



4.- Genera el plan de la consulta con la condición "> 10". ¿Qué diferencia hay en el plan? ¿porqué ?

Se nota una diferencia significativa al aplicar la condición >10, ya que se accede a un mayor número de filas en comparación con la consulta sin la condición. Esto se refleja claramente en el número de accesos a las filas que se muestra en el informe del plan de ejecución de la consulta.

De hecho, con una condición de no igualdad se recorre más filas que con una condición de igualdad, el optimizador no puede utilizar índices para realizar la selección.



d)- (OPCIONAL) *Crea consultas diferentes sobre las tablas de películas, que provoquen en Oracle las operaciones siguientes*: (consulta en teoría qué características las provocan)

d.1) Full table scan, d.2) Index unique scan, d.3) Full Index scan y d.4) Cartesian join.

*Crea otras consultas que provoquen en Oracle (ver Reglas de Optimización) que:*

d.5) - NO use los índices

d.6) - SÍ use índices

*Prueba a forzar el uso de un tipo de operación concreta como este ejemplo:*

d.7) Select /\*+ Full(Empleado) \*/ NumEmp, NomEmp from Empleado;

e)- (OPCIONAL) Prueba otro modo de conseguir los datos del plan usando el paquete XPLAN (basado en estimaciones, como el explain plan) :

SELECT PLAN\_TABLE\_OUTPUT FROM TABLE(DBMS\_XPLAN.DISPLAY());

NOTA: Para ampliar tu tablespace: (desde el usuario ADMINUSER)

alter DATABASE DATAFILE 'ESPACIOU-tuyo' autoextend ON

next 521k maxsize 25M;

# APARTADO 4: Consultas jerárquicas

(Entregar las instrucciones necesarias en sql y los resultado de las consulta)

Queremos hacer un diccionario que relaciona jerárquicamente los conceptos.

Ej.: el concepto “select” con todos los tipos de select. Para ello hacer lo siguiente:

a)- Crear una tabla diccionario DICCION, que tenga los siguientes atributos:

PalID , será como máximo de 20 caracteres. Identifica la palabra.

Descripción , de 50 caracteres

PadreID, de 20 char. Representa un concepto más genérico que PalID, en las filas insertadas en b)- se ve que ‘select compuesta’ es el PadreID de ‘select jerarquica’ y de ‘select correlativa’

b)- Incluir las siguientes filas: (estas comillas son del word, no válidas en oracle)

*(‘select jerarquica’,’estructura tabla en arbol’, ‘select compuesta’);*

*(‘fecha sistema’,’es la fecha que tiene el ordenador’,’fecha’);*

*(‘fecha’,’tipo de dato , en oracle : DATE’,’nada’);*

*(‘select compuesta’, ‘consultas con varias partes’, ‘select’);*

*(‘select simple’, ‘consultas con una sola instruccion’, ‘select’);*

*(‘select’, ‘hacer consulta’, ‘nada’);*

*(‘sql’,’lenguaje de consultas estructuradas’, ‘nada’);*

*(‘select correlativa’, ‘coordina resultado subconsulta’, ‘select compuesta’);*

c)- Hacer una consulta jerárquica conectada por PalID y PadreID que empiece con la palabra ‘select’.

(es como la que usamos para ver la operaciones de la PLAN\_TABLE)

d)- Insertar una fila con valores PalID = ‘ select anidada’, descripción = ‘consulta dentro de consulta’ y PadreID = ‘select compuesta’. Lo importante de esta inserción es que se quiere hacer solo en el caso que el padre exista, es decir solo hacerla en caso de que una consulta de PalID= ́select compuesta’ devuelve algo. Si no devuelve nada no se debe crear. (es una “inserción condicionada”).

e) Ejecuta de nuevo la consulta jerárquica y comprueba que sale correctamente.