

Profesor:	DAVID APARICIO ESCRIBANO	Grupo	81
Alumno/a:	César López Mantecón	NIA:	100472092
Alumno/a:	Irene Subias Serrano	NIA:	100472108
Alumno/a:	Paula Subias Serrano	NIA:	100472119

1. Tabla de contenidos

2. Introducción.....	1
3. Situación actual.....	1
3.1. Operación de actualización.....	2
3.2. Consulta 1.....	3
3.3. Consulta 2.....	4
3.4. Costes globales.....	5
4. Propuestas de mejora.....	6
4.1. Iteración 1: creación del índice.....	6
4.2. Iteración 2: creación de un cluster.....	8
4.3. Iteración 3: Creación indice sobre Performances.....	12
4.4. Iteración 4: Modificación de pctfree de la tabla Tracks.....	14
5. Análisis del nuevo diseño.....	18
6. Conclusión.....	19

2. Introducción

Este documento recoge el desarrollo de la tercera práctica de la asignatura Ficheros y Bases de datos. Siguiendo el guión proporcionado, se incluye un análisis de la situación actual de la base de datos, una serie de propuestas de mejoras (con sus respectivos análisis y comparación entre ellos) y una conclusión final del proyecto.

3. Situación actual

La base de datos consta de 16 relaciones: *albums*, *attendances*, *clientes*, *concerts*, *involvement*, *languages*, *managers*, *musicians*, *nationalities*, *performances*, *performers*, *publishers*, *songs*, *studios*, *tours*, *tracks*.

Existen varias tablas relacionadas a través de claves ajena, todas ellas con política de actualización en cascada. Esto significa que cualquier modificación de algún atributo que sea clave ajena conlleva una actualización en la tabla hijo.

En cuanto a las operaciones que se van a realizar sobre la base de datos, en el enunciado se describe una carga de trabajo. La carga de trabajo consta de 3 operaciones: una actualización y dos selecciones.

3.1. Operación de actualización

La operación de actualización modifica la columna *lyrics* de una fila de *tracks* cuyo campo *searchk* coincide con uno dado. Su tasa de ejecución es del 98%.

Su plan de ejecución, junto con sus estadísticas, es el siguiente:

```
Plan hash value: 973582657

| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
|  0  | UPDATE STATEMENT  |         |       |        |            | 00:00:01 |
|  1  | UPDATE             | TRACKS |       |        |            | 00:00:01 |
| * 2  | TABLE ACCESS FULL | TRACKS |     1 | 123   |    719  (1)| 00:00:01 |

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("SEARCHK"='B0569SD08384Y9N//7')

Estadísticas
-----
      2 recursive calls
      10 db block gets
    2688 consistent gets
      606 physical reads
    2480 redo size
      494 bytes sent via SQL*Net to client
      929 bytes received via SQL*Net from client
        2 SQL*Net roundtrips to/from client
        0 sorts (memory)
        0 sorts (disk)
      1 rows processed
```

Lo más destacable del plan de ejecución es el *full scan* sobre la tabla *tracks*, siendo la operación con el mayor coste estimado. Esto se debe a que para la modificación primero se ha de localizar el registro a través de la clave identificativa *searchk*. Lo que supone un coste de N accesos

(siendo N el número de bloques que ocupan todos los registros de la tabla *tracks*) ya que no hay ordenación según la clave de búsqueda, y no se ha indicado que esta sea identificativa. El número de accesos se confirma gracias a la estimación de número de bloques traídos de disco presente en las estadísticas bajo el nombre “*consistent gets*”, que coincide con el número de bloques de la tabla *Tracks*, la única a la que accede Oracle para la modificación. Este tamaño se puede conseguir desde *user_tables* con el siguiente comando:

```
select table_name, num_rows, blocks, avg_row_len, avg_space, pct_free
from user_tables where table_name = 'TRACKS';
```

TABLE_NAME	NUM_ROWS	BLOCKS	AVG_ROW_LEN	AVG_SPACE	PCT_FREE
TRACKS	146275	2638	123	926	10

3.2. Consulta 1

La consulta 1 selecciona intérpretes que aparezcan como autores de una canción o sean miembros de una banda que haya interpretado una canción escrita por alguno de sus miembros. Su tasa de ejecución es del 1%.

Su plan de ejecución es el siguiente:

Plan de Ejecución								
Plan hash value: 3955265815								
Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time	Other
0	SELECT STATEMENT		695	36835		5236 (2)	00:00:01	
1	TEMP TABLE TRANSFORMATION							
2	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9DB084_334C22EB	695	61160		2401 (2)	00:00:01	
3	HASH UNIQUE		263K	22M		2389 (1)	00:00:01	
*	HASH JOIN		1224	35496		4 (8)	00:00:01	
5	INDEX FAST FULL SCAN	PK_ININVOLVEMENT	247K	13M		2384 (1)	00:00:01	
6	VIEW		247K	14M		2384 (1)	00:00:01	
7	SORT UNIQUE							
8	UNION-ALL							
9	INDEX FAST FULL SCAN	PK_SONGS	123K	3503K		165 (1)	00:00:01	
10	TABLE ACCESS FULL	SONGS	123K	3744K		171 (1)	00:00:01	
11	VIEW	VW_FOJ_0	695	36835		2836 (3)	00:00:01	
*	12	HASH JOIN FULL OUTER	695	55600		2836 (3)	00:00:01	
13	VIEW		542	21680		2037 (3)	00:00:01	
14	HASH GROUP BY		542	55284		2037 (3)	00:00:01	
*	15	HASH JOIN RIGHT OUTER	856K	83M		1999 (1)	00:00:01	
16	VIEW		695	41700		4 (8)	00:00:01	
17	TABLE ACCESS FULL	SYS_TEMP_0FD9DB084_334C22EB	695	48650		4 (8)	00:00:01	
18	TABLE ACCESS FULL	PERFORMANCES	856K	34M		1991 (1)	00:00:01	
19	VIEW		695	27800		799 (2)	00:00:01	
20	HASH GROUP BY		695	93825		799 (2)	00:00:01	
*	21	HASH JOIN RIGHT OUTER	146K	18M		792 (1)	00:00:01	
22	VIEW		695	41700		4 (8)	00:00:01	
23	TABLE ACCESS FULL	SYS_TEMP_0FD9DB084_334C22EB	695	48650		4 (8)	00:00:01	
*	24	HASH JOIN	146K	10M		788 (1)	00:00:01	
25	TABLE ACCESS FULL	ALBUMS	21561	631K		68 (8)	00:00:01	
26	TABLE ACCESS FULL	TRACKS	146K	6428K		718 (1)	00:00:01	

Acompañado de las siguientes estadísticas:

Estadísticas

```
37 recursive calls
  2 db block gets
11629 consistent gets
1234 physical reads
  376 redo size
24443 bytes sent via SQL*Net to client
1677 bytes received via SQL*Net from client
   48 SQL*Net roundtrips to/from client
     1 sorts (memory)
     0 sorts (disk)
  695 rows processed
```

Destacan 5 operadores *join*, 6 *full scan a tablas* y 3 accesos *full scan* a índices, siendo los costes más elevados los *joins* de la vista auxiliar creada en la fila de id 11. Las operaciones de más coste son en las que interviene un *join*.

Las tablas que intervienen en esta consulta son: *songs*, *involvement*, *albums*, *tracks* y *performances*.

Los datos de dichas tablas son los siguientes:

TABLE_NAME	NUM_ROWS	BLOCKS	AVG_ROW_LEN	AVG_SPACE	PCT_FREE
ALBUMS	21561	244	75	1276	10
INVOLVEMENT	1224	13	51	3053	10
PERFORMANCES	856432	7300	59	875	10
SONGS	123698	622	33	1143	10
TRACKS	146275	2638	123	926	10

3.3. Consulta 2

La consulta dos selecciona los diez intérpretes con mayor porcentaje de interpretaciones grabadas, junto con la antigüedad de la pieza separada en años, meses y días. Su tasa de ejecución es del 1%.

El plan de ejecución de la consulta es el siguiente:

Plan de Ejecucion								
			Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
	Id	Operation						
	* 1	SELECT STATEMENT		10	530		8658 (2)	00:00:01
	* 2	COUNT STOPKEY		542	28726		8658 (2)	00:00:01
	* 3	VIEW		542	56910		8658 (2)	00:00:01
	* 4	SORT ORDER BY STOPKEY		542	56910		8658 (2)	00:00:01
	* 5	HASH GROUP BY		856K	85M	9560K	8582 (1)	00:00:01
	* 6	HASH JOIN RIGHT OUTER		146K	7843K		3608 (1)	00:00:01
	* 7	VIEW		146K	11M	13M	3608 (1)	00:00:01
	* 8	HASH GROUP BY		146K	11M		788 (1)	00:00:01
	* 9	HASH JOIN		146K	11M		68 (0)	00:00:01
	10	TABLE ACCESS FULL	ALBUMS	21561	631K		719 (1)	00:00:01
	11	TABLE ACCESS FULL	TRACKS	146K	7570K		1991 (1)	00:00:01
		TABLE ACCESS FULL	PERFORMANCES	856K	40M			

Con las siguientes estadísticas:

Estadisticas	
4	recursive calls
0	db block gets
10209	consistent gets
0	physical reads
196	redo size
1142	bytes sent via SQL*Net to client
1011	bytes received via SQL*Net from client
2	SQL*Net roundtrips to/from client
1	sorts (memory)
0	sorts (disk)
10	rows processed

Lo más destacable son las operaciones con id entre 2 y 5, que presentan costes de más de 8000 unidades. Estas son una ordenación, una función de agregación *group by*, un *right outer join* y la creación de una vista auxiliar.

Las tablas que intervienen en esta consulta son: *albums*, *tracks* y *performances*. Cuyos datos han sido especificados en apartados anteriores.

3.4. Costes globales

Los costes de la carga de trabajo los obtenemos a través del procedimiento *run_test* facilitado en el código de apoyo para esta práctica:

Comando empleado:

```
exec pkg_costes.run_test( 5 );
```

```

exec pkg_costes.run_test(15);

exec pkg_costes.run_test(20);
  
```

Los resultados obtenidos son:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 4.720 ms	TIME CONSUMPTION: 4.748 ms	TIME CONSUMPTION: 4.712,9 ms
CONSISTENT GETS: 281.207,8 blocks	CONSISTENT GETS: 281.204,07 blocks	CONSISTENT GETS: 281.206,25 blocks

La media de estos datos es:

- TIME CONSUMPTION: 4.726,97 ms
- CONSISTENT GETS: 281.206,04 blocks

A la hora de optimizar la base de datos, emplearemos el dato de *consistent gets* (número de accesos a bloques de disco) ya que el tiempo de ejecución varía según la máquina asignada por el servicio de oracle.

Analizando la carga de trabajo, definida en el procedimiento *pr_workload*, se ejecutan 98 veces la actualización y una vez cada consulta. Teniendo en cuenta las estimaciones que nos da el plan de ejecución de cada operación, podemos ver que el coste en accesos de cada operación es el siguiente:

- Actualización: $98 \cdot 2688 = 263.424$ accesos.
- Consulta 1: 11.629 accesos.
- Consulta 2: 10.209 accesos.

Lo que resulta en un coste total de 285.262 accesos. Un número muy similar al que nos devuelve el proceso *run_test*. Dado que los datos que incluye el plan de ejecución son estimaciones, la diferencia entre estos datos no es significativa (tienen una diferencia del 1.44%). Para los siguientes cálculos emplearemos los 285.262 accesos como coste total.

Con esto en mente, vemos que la operación de actualización representa un 92,34% de los costes de la carga de trabajo; mientras que las consultas 1 y 2 representan un 4,08% y un 3,58% de la carga de trabajo.

En el siguiente apartado se plantearán propuestas para reducir estos valores.

4. Propuestas de mejora

Teniendo en cuenta los datos obtenidos en el análisis de la situación actual, es imperativo reducir el coste sobre la operación de actualización debido al significativo porcentaje que presenta sobre los costes operacionales. Consideraremos [N] propuestas con intención de reducir los costes de las operaciones. Estas propuestas se centran en reducir los costes de las operaciones más costosas de cada plan de ejecución como son los numerosos *table full scan* y *hash join*.

4.1. Iteración 1: creación del índice

Como primera propuesta se ha añadido un índice a la tabla *Tracks* para la clave *searchk*, ya que esta es la que se usa en la selección del proceso de modificación. Esto se ha realizado mediante el comando:

```
CREATE UNIQUE INDEX ind_searchk ON tracks(searchk) TABLESPACE TAB_8K;
```

Esta clave proviene de la concatenación de las dos claves que forman la clave primaria de la tabla, por lo que se puede asegurar que es una clave identificativa, y por lo tanto el índice se puede declarar como único. Así, no solo se mejora los accesos necesarios para encontrar una instancia, sino que también se asegura de que con la primera coincidencia es suficiente.

Después de la creación del índice se ha comprobado el plan de ejecución de la modificación:

```

Plan hash value: 3507082542

| Id  | Operation          | Name      | Rows | Bytes | Cost (%CPU)| Time     |
| 0   | UPDATE STATEMENT  |           | 1    | 123   | 3    (0) | 00:00:01 |
| 1   | UPDATE             | TRACKS   |       |        |          |           |
| * 2 | INDEX UNIQUE SCAN | IND_SEARCHK| 1    | 123   | 2    (0) | 00:00:01 |

Predicate Information (identified by operation id):
-----
 2 - access("SEARCHK"='B05695D08384Y9N//7')

Estadísticas
-----
 1 recursive calls
 6 db block gets
 3 consistent gets
 2 physical reads
 0 redo size
 494 bytes sent via SQL*Net to client
 933 bytes received via SQL*Net from client
 2 SQL*Net roundtrips to/from client
 0 sorts (memory)
 0 sorts (disk)
 1 rows processed

```

De manera inmediata se puede comprobar que el coste (bajo la columna *cost*) que anteriormente venía denotado como 719 ha bajado a 3. Aunque este número es solo una estimación del uso de recursos del proceso y no tiene una unidad real, se puede usar como comparación del coste general y ver la mejora. Además, en el apartado de estadísticas se comprueba que se han estimado 3 accesos a disco para extraer bloques, lo que significa una reducción del coste en un 99,89% para cada consulta. Este bajo número se debe a que al crearse

un índice de una clave única internamente se ha implementado un árbol que reduce los accesos para toda selección por la clave.

En las consultas no existen cambios significativos sobre el plan de ejecución, esto es porque *searchk* no es relevante para estas.

Después de esta iteración se ha vuelto a ejecutar el procedimiento *run_test* del paquete para poder comprobar los costes globales:

Los resultados obtenidos son:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 4.122 ms	TIME CONSUMPTION: 4.096,27 ms	TIME CONSUMPTION: 3.937,9 ms
CONSISTENT GETS: 22.442,4 blocks	CONSISTENT GETS: 22.349,2 blocks	CONSISTENT GETS: 22.431 blocks

La media de estos datos es:

- TIME CONSUMPTION: 4.052,06 ms
- CONSISTENT GETS: 22.407,53 blocks

Se ha conseguido una reducción del tiempo de ejecución del 14,28 % y de 92,03% de accesos entre todos los procesos. Tras esta iteración, el desglose de costes para cada operaciones son:

- Actualización: $98 \cdot 3 = 294$ accesos.
- Consulta 1: 11.629 accesos.
- Consulta 2: 10.209 accesos.

Con estos números, el nuevo coste global estimado es de 22.132 accesos, representando la actualización un 1.33%; mientras que las consultas 1 y 2 representan un 52,54% y un 46,13%.

Con este nuevo desglose, los esfuerzos futuros se centrarán en reducir el coste de las consultas puesto que, en total, presentan un 98,67% de los costes de la carga de trabajo.

4.2. Iteración 2: creación de un cluster

Para la segunda propuesta se ha creado un *cluster* sobre las tablas *albums* y *tracks* usando el campo *PAIR* con el fin de mejorar el desempeño de las consultas, ya que ambas realizan *join* usando estas dos tablas. Tras eliminar las tablas y el índice se ha creado el *cluster* usando el siguiente comando:

```
create cluster record (pair CHAR(15));
```

Se han vuelto a crear las tablas añadiendo el siguiente comando a la creación de *album* y *tracks*:

cluster record (pair)

Adicionalmente, se ha creado un índice para el *cluster* para poder realizar las operaciones de actualización:

```
create index ind_cluster on cluster record tablespace tab_8k;
```

Tras crear el *cluster* se han vuelto a comprobar los planes de ejecución de la actualización y las consultas.

El comportamiento individual del cluster, sobre la base de datos original es el siguiente:

- Operación de actualización

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	UPDATE STATEMENT			1	27	5854 (1) 00:00:01
1	UPDATE	TRACKS				
* 2	TABLE ACCESS FULL	TRACKS	1	27	5854 (1)	00:00:01


```
Predicate Information (identified by operation id):
-----
2 - filter("SEARCHK"='L6066I1S901905P//7')

Estadísticas
-----
      5 recursive calls
      4 db block gets
  21599 consistent gets
  21577 physical reads
   1548 redo size
  1012 bytes sent via SQL*Net to client
  1046 bytes received via SQL*Net from client
      3 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
      1 rows processed
```

Se puede observar que los costes, *Cost*, han aumentado de manera significativa pasando de 3 a 5.854. En el apartado de estadísticas se puede ver que se han estimado 21.599 accesos a disco lo que supone un incremento muy significativo respecto a los resultados de la iteración anterior.

Este aumento se debe a que un *cluster* agrupa varias tablas asegurándose de que todas las filas combinadas estén en el mismo bloque por lo que cada tabla tendrá sus filas ocupará en conjunto un mayor número de bloques. Esto hace que las operaciones de actualización sobre una tabla incluida en un *cluster* sean mucho más costosas.

- Consulta 1

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		7634K	385M		24101 (1)	00:00:01
1	TEMP TABLE TRANSFORMATION						
2	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9DBB36_334C22EB	264K	25M	28M	9654 (1)	00:00:01
*	HASH UNIQUE		264K	25M		3486 (1)	00:00:01
4	HASH JOIN		1224	52632		5 (0)	00:00:01
5	TABLE ACCESS FULL	INVOLVEMENT	248K	13M		3480 (1)	00:00:01
6	VIEW		248K	24M	14M	3480 (1)	00:00:01
7	SORT UNIQUE						
8	UNION-ALL						
9	INDEX FAST FULL SCAN	PK_SONGS	124K	5228K		170 (0)	00:00:01
10	TABLE ACCESS FULL	SONGS	124K	7163K		171 (1)	00:00:01
11	VIEW	VW_FOJ_0	7634K	385M	7744K	14448 (1)	00:00:01
*	HASH JOIN FULL OUTER		7634K	582M		14448 (1)	00:00:01
12	VIEW		152K	5952K		3350 (1)	00:00:01
13	HASH GROUP BY		152K	25M		3350 (1)	00:00:01
14	HASH JOIN OUTER		152K	25M	16M	3343 (1)	00:00:01
*	NESTED LOOPS		152K	15M		441 (1)	00:00:01
16	VIEW		17805	765K		364 (0)	00:00:01
*	HASH JOIN	index\$_join\$_006					
18	INDEX FAST FULL SCAN	PK_ALBUMS	17805	765K		149 (0)	00:00:01
19	INDEX FAST FULL SCAN	UK_ALBUMS	17805	765K		386 (0)	00:00:01
20	TABLE ACCESS CLUSTER	TRACKS	9	540		1 (0)	00:00:01
*	INDEX UNIQUE SCAN	IND_CLUSTER	1			0 (0)	00:00:01
22	VIEW		264K	18M		998 (1)	00:00:01
23	TABLE ACCESS FULL	SYS_TEMP_0FD9DBB36_334C22EB	264K	20M		998 (1)	00:00:01
24	VIEW		874K	33M		8529 (1)	00:00:01
25	HASH GROUP BY		874K	140M		8529 (1)	00:00:01
*	HASH JOIN RIGHT OUTER		874K	140M	21M	8490 (1)	00:00:01
27	VIEW		264K	18M		998 (1)	00:00:01
28	TABLE ACCESS FULL	SYS_TEMP_0FD9DBB36_334C22EB	264K	20M		998 (1)	00:00:01
29	TABLE ACCESS FULL	PERFORMANCES	874K	79M		1991 (1)	00:00:01
30	TABLE ACCESS FULL						

Estadísticas

```

113 recursive calls
 34 db block gets
54927 consistent gets
13043 physical reads
205512 redo size
24444 bytes sent via SQL*Net to client
1671 bytes received via SQL*Net from client
 48 SQL*Net roundtrips to/from client
  4 sorts (memory)
  0 sorts (disk)
 695 rows processed

```

En la sección de costes del plan se puede observar que que a pesar de que el coste del join de tracks y albums (id 18) ha disminuido, el coste del resto de operaciones han aumentado de manera significativa. Esto hace que el coste total de la consulta termine siendo mayor al original. Los accesos a disco también aumentan por la misma razón que en el caso de la actualización.

- Consulta 2

Id	Operation	Name	Rows	Bytes	TempSpc	Cost	(%CPU)	Time
0	SELECT STATEMENT					8000	(2)	00:00:01
* 1	COUNT STOPKEY					8000	(2)	00:00:01
2	VIEW		3	159		8000	(2)	00:00:01
* 3	SORT ORDER BY STOPKEY		3	558		8000	(2)	00:00:01
4	HASH GROUP BY		3	558		8000	(2)	00:00:01
* 5	HASH JOIN RIGHT OUTER		874K	155M	13M	7923	(1)	00:00:01
6	VIEW		152K	11M		448	(3)	00:00:01
7	HASH GROUP BY		152K	16M		448	(3)	00:00:01
8	NESTED LOOPS		152K	16M		441	(1)	00:00:01
9	VIEW	index\$_join\$_001	17805	765K		364	(0)	00:00:01
* 10	HASH JOIN							
11	INDEX FAST FULL SCAN	PK_ALBUMS	17805	765K		149	(0)	00:00:01
12	INDEX FAST FULL SCAN	UK_ALBUMS	17805	765K		306	(0)	00:00:01
13	TABLE ACCESS CLUSTER	TRACKS	9	621		1	(0)	00:00:01
* 14	INDEX UNIQUE SCAN	IND_CLUSTER	1			0	(0)	00:00:01
15	TABLE ACCESS FULL	PERFORMANCES	874K	86M		1991	(1)	00:00:01

Estadísticas

```

95 recursive calls
  6 db block gets
51392 consistent gets
29297 physical reads
1000 redo size
1146 bytes sent via SQL*Net to client
1011 bytes received via SQL*Net from client
   2 SQL*Net roundtrips to/from client
   3 sorts (memory)
   0 sorts (disk)
  10 rows processed
  
```

La consulta 2 ve reducido su coste general. Sin embargo, no de manera significativa. Principalmente, se reduce el coste de la operación de id 5 (*hash join*). No obstante, la reducción no es lo suficientemente grande como para tenerse en cuenta. Esto sumado a la baja tasa de ejecución de esta consulta, hace que su impacto sobre el coste global no sea suficiente como para contrarrestar el aumento de costes de las otras dos operaciones.

Tras esta iteración se ha vuelto a ejecutar el procedimiento de *run_test*, aunque modificado para evitar la ejecución de *pr_reset* ya que entraba en conflicto con el cluster (no se permite la operación de truncado sobre una tabla incluida en cluster).

Estos son los resultados obtenidos:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 9.313,6 ms	TIME CONSUMPTION: 12.398,27 ms	TIME CONSUMPTION: 9.898,3 ms
CONSISTENT GETS: 2.218.965 blocks	CONSISTENT GETS: 2.176.073,4 blocks	CONSISTENT GETS: 2.176.073 blocks

La media de estos datos es:

- TIME CONSUMPTION: 10.536,72 ms
- CONSISTENT GETS: 2.190.370,47 blocks

La creación del cluster no presenta un muy significativo aumento en los costes operacionales con respecto a la iteración anterior. El coste global aumenta en un 9.796,84% lo que nos hace concluir que **se descarta la propuesta**.

El comportamiento del cluster, combinado con el índice, presenta los siguientes resultados.

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 7.796,4 ms	TIME CONSUMPTION: 2.640,3 ms	TIME CONSUMPTION: 7.354,35 ms
CONSISTENT GETS: 103.161,2 blocks	CONSISTENT GETS: 103.203,27 blocks	CONSISTENT GETS: 103.194,4 blocks

La media de estos datos es:

- TIME CONSUMPTION: 5.930,35 ms
- CONSISTENT GETS: 103.186,29 blocks

Aunque se reducen los costes significativamente, incluso llegando a mejorar la situación inicial, se obtienen peores resultados que en la iteración 1. Por esto, se concluye que la implementación de este cluster no es una buena propuesta de mejora y, por tanto, **queda descartada**.

4.3. Iteración 3: Creación índice sobre Performances

Con el fin de reducir el coste de la primera consulta, se ha implementado un índice sobre las claves *performer*, *songtitle* y *songwriter* de la tabla *Performer*. Con esto, se espera mejorar el coste de la operación *join* de esta tabla y la subconsulta *autorship*.

```
create index ind_performance on Performances(performer, songtitle, songwriter);
```

Al ser el índice sobre las columnas de la primera consulta, esta es en la única en la que tiene un cambio notable:

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		201M	9G		25017 (5)	00:00:01
1	TEMP TABLE TRANSFORMATION						
2	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9DBD40_334C22EB	258K	25M	28M	9440 (1)	00:00:01
3	HASH UNIQUE		258K	25M		3414 (1)	00:00:01
*	4	HASH JOIN	1224	52632		5 (0)	00:00:01
5	TABLE ACCESS FULL	INVOLVEMENT	242K	13M		3408 (1)	00:00:01
6	VIEW		242K	23M	14M	3408 (1)	00:00:01
7	SORT UNIQUE						
8	UNION-ALL						
9	INDEX FAST FULL SCAN	PK_SONGS	121K	5101K		178 (0)	00:00:01
10	TABLE ACCESS FULL	SONGS	121K	6999K		171 (1)	00:00:01
11	VIEW	VW_FOJ_0	201M	9G		15577 (7)	00:00:01
*	12	HASH JOIN FULL OUTER	201M	14G	7432K	15577 (7)	00:00:01
13	VIEW		146K	5714K		3492 (1)	00:00:01
14	HASH GROUP BY		146K	22M		3492 (1)	00:00:01
*	15	HASH JOIN OUTER	146K	22M	13M	3486 (1)	00:00:01
*	16	HASH JOIN	146K	11M		788 (1)	00:00:01
17	TABLE ACCESS FULL	ALBUMS	25136	10880K		68 (0)	00:00:01
18	TABLE ACCESS FULL	TRACKS	146K	5999K		718 (1)	00:00:01
19	VIEW		258K	18M		976 (1)	00:00:01
20	TABLE ACCESS FULL	SYS_TEMP_0FD9DBD40_334C22EB	258K	20M		976 (1)	00:00:01
21	VIEW		921K	35M		8479 (1)	00:00:01
22	HASH GROUP BY		921K	147M		8479 (1)	00:00:01
*	23	HASH JOIN RIGHT OUTER	921K	147M	20M	8438 (1)	00:00:01
24	VIEW		258K	18M		976 (1)	00:00:01
25	TABLE ACCESS FULL	SYS_TEMP_0FD9DBD40_334C22EB	258K	20M		976 (1)	00:00:01
26	INDEX FAST FULL SCAN	IND_PERFORMANCE	921K	83M		1746 (1)	00:00:01

Estadísticas

```

-----  

       64 recursive calls  

        3 db block gets  

  10622 consistent gets  

  2681 physical reads  

   560 redo size  

24444 bytes sent via SQL*Net to client  

 1921 bytes received via SQL*Net from client  

    48 SQL*Net roundtrips to/from client  

     1 sorts (memory)  

     0 sorts (disk)  

   695 rows processed
  
```

El acceso completo a la tabla *Performances* cambia a un acceso a índice, lo cual consigue reducir los accesos en aproximadamente 1000 bloques.. Este cambio supone una reducción de un 8,66% en el coste de la consulta con respecto a la interacción 1.

Después de este cambio costes totales son los siguientes:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 6.334,4 ms	TIME CONSUMPTION: 6.449,5 ms	TIME CONSUMPTION: 5.900,15 ms
CONSISTENT GETS: 21.334 blocks	CONSISTENT GETS: 21.334,86 blocks	CONSISTENT GETS: 21.335,75 blocks

La media de estos datos es:

- TIME CONSUMPTION: 6228 ms
- CONSISTENT GETS: 21.334,87 bloques

Con el siguiente desglose:

- Actualización: 294
- Consulta 1: 10.622
- Consulta 2: 10.209

Lo que significa que el coste global estimado es de 21.125 accesos. La actualización representa un 1,39% de este coste, la consulta 1 representa un 50,28% y la consulta 2 representa un 48,33%. Vemos que las consultas siguen representando un alto porcentaje de la carga de trabajo (98,61%), aunque se han conseguido reducir los costes globales en un 4.55%.

4.4. Iteración 4: Modificación de pctfree de la tabla Tracks

Se propone incrementar el tamaño de pctfree para tener más espacio para modificaciones, con lo que se hace una prueba con un 20% de espacio en vez del 10% por defecto. Al contrario de lo esperado, aumenta el coste ya que el cambio no ayuda a las modificaciones como se había planteado. Además, empeora las consultas al necesitar más bloques para almacenar los registros de la tabla, lo que no afecta a las modificaciones gracias al índice. Los planes de ejecución no cambian en ninguno de los casos, al ser el cambio sobre el tamaño de los bloques y no sobre estructuras lógicas.

Para realizar este cambio se añade *pctfree* 20 en la creación de la tabla.

Con este cambio, las estadísticas de cada operación resultan en:

- Update:

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

- Consulta 1:

```
Estadisticas
```

```
-----  
    78  recursive calls  
     4  db block gets  
10947  consistent gets  
12200  physical reads  
   140  redo size  
24444  bytes sent via SQL*Net to client  
1670  bytes received via SQL*Net from client  
   48  SQL*Net roundtrips to/from client  
    0  sorts (memory)  
    1  sorts (disk)  
  695  rows processed
```

- Consulta 2:

```
Estadisticas
```

```
-----  
    67  recursive calls  
     0  db block gets  
10592  consistent gets  
10502  physical reads  
    0  redo size  
1137  bytes sent via SQL*Net to client  
1012  bytes received via SQL*Net from client  
    2  SQL*Net roundtrips to/from client  
    1  sorts (memory)  
    0  sorts (disk)  
  10  rows processed
```

Los costes globales, obtenidos mediante *run_test* son:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 7.162,2 ms	TIME CONSUMPTION: 6.823,27 ms	TIME CONSUMPTION: 6.798 ms
CONSISTENT GETS: 22.020 blocks	CONSISTENT GETS: 21.992,4 blocks	CONSISTENT GETS: 21.930,95 blocks

- TIME CONSUMPTION: 6.927,76 ms

- CONSISTENT GETS: 21981,12 blocks

Se aprecia, en general, que aumenta el número de accesos para las consultas. Lo que repercute en el coste global. Esto supone un peor rendimiento, por lo que la propuesta queda **descartada**.

Se plantea entonces realizar la acción contraria, y reducir el pctfree a un 0%, con lo que se consigue una pequeña mejora en las consultas, aunque sin cambio en las modificaciones. Esto se atribuye a un pequeño aumento del espacio disponible para registros, lo que se traduce en una mayor cantidad de registros por bloque. Se considera que en un proyecto general no sería buena idea realizar este cambio, sin embargo para la carga de trabajo definida, resulta beneficioso. Se añade *pctfree 0* a la creación de la tabla. Las estadísticas de cada operación tras este cambio serían:

Consulta 1:

```
Estadisticas
-----
 86 recursive calls
  0 db block gets
11311 consistent gets
13249 physical reads
  0 redo size
24444 bytes sent via SQL*Net to client
1670 bytes received via SQL*Net from client
  48 SQL*Net roundtrips to/from client
   1 sorts (memory)
   0 sorts (disk)
 695 rows processed
```

Consulta 2:

```
Estadisticas
-----
 67 recursive calls
  0 db block gets
10019 consistent gets
 9930 physical reads
  0 redo size
1138 bytes sent via SQL*Net to client
1012 bytes received via SQL*Net from client
   2 SQL*Net roundtrips to/from client
   1 sorts (memory)
   0 sorts (disk)
 10 rows processed
```

DATOS GLOBALES:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 5.559,2	TIME CONSUMPTION:	TIME CONSUMPTION:

ms	6.015,73 ms	5.660,85 ms
CONSISTENT GETS: 21.052 blocks	CONSISTENT GETS: 21.116,46 blocks	CONSISTENT GETS: 20.946 blocks

La media de estos datos es:

- TIME CONSUMPTION: 5.745,26 ms
- CONSISTENT GETS: 21.038 bloques

Tras el cambio, en los costes globales hay una mejora de unos 300 accesos , ya que afecta solo a la tabla Tracks que no tiene un gran peso en las consultas. Debido a esto, y ya que se ha conseguido una mejora, se plantea reducir a 0 el pctfree del resto de las tablas para intentar aumentar la mejora global. Como no cambian los costes de las modificaciones se ha llegado a la conclusión de que no se utiliza este espacio en la operación, por lo que el cambio realizado por la propuesta no tendría efectos negativos en los procesos planteados. Para esto a la creación de cada tabla se le añade *pctfree 0*.

Con todas las tablas con un pctfree al 0%, se consigue reducir el número de bloques necesarios para almacenar los registros. Así, en ambas consultas se ha podido reducir los accesos necesarios para realizar las operaciones con estas tablas, y así reducir el coste global. Las estadísticas resultantes son:

Consulta 1:

```

Estadisticas
-----
  88 recursive calls
   1 db block gets
10312 consistent gets
12704 physical reads
  140 redo size
24444 bytes sent via SQL*Net to client
 1670 bytes received via SQL*Net from client
   48 SQL*Net roundtrips to/from client
    1 sorts (memory)
    0 sorts (disk)
   695 rows processed

```

Consulta 2:

Estadísticas

```

-----  

  67 recursive calls  

   0 db block gets  

 9263 consistent gets  

 9183 physical reads  

   0 redo size  

1141 bytes sent via SQL*Net to client  

1012 bytes received via SQL*Net from client  

   2 SQL*Net roundtrips to/from client  

   1 sorts (memory)  

   0 sorts (disk)  

 10 rows processed
  
```

El efecto del cambio se nota especialmente en la segunda consulta, que disminuye el número de accesos necesarios en gran medida. Esta mejora también se observa en los datos globales:

5 iteraciones	15 iteraciones	20 iteraciones
TIME CONSUMPTION: 6.686,4 ms	TIME CONSUMPTION: 6.843,6 ms	TIME CONSUMPTION: 6.833,1 ms
CONSISTENT GETS: 20.119,8 blocks	CONSISTENT GETS: 20.071,86 blocks	CONSISTENT GETS: 20.197,75 blocks

La media de estos datos es:

- TIME CONSUMPTION: 6.787,7 ms
- CONSISTENT GETS: 20.129,8 bloques

Se consigue reducir en 909 accesos los costes globales de todos los procesos, lo cual mejora un 4,32% respecto a la última propuesta. Se considera entonces esta propuesta como beneficiosa en el contexto de la práctica.

Después de este último cambio, y dado que las últimas iteraciones han tenido una mejora inferior al 5%, se considera finalizado el proceso de optimización de la base de datos.

5. Análisis del nuevo diseño

El diseño final cuenta con 2 índices, uno sobre *searchk* en la tabla *tracks* y otro sobre *performer*, *songtitle* y *songwriter* en la tabla *performer*. Estas modificaciones están descritas en las iteraciones 1 y 3, respectivamente. Además, se ha reducido el espacio *pctfree* a 0, lo que permite tener una base de datos más densa.

Gracias a estos cambios, se logra una gran mejora en el rendimiento de la base de datos. Logrando reducir el número de accesos para la carga de trabajo desde 281.206,04 a 20.129,8 después de cuatro iteraciones. Lo que supone una reducción de un 92,84% del coste, medido en número de accesos.

La distribución de la carga de trabajo sería la siguiente:

- Actualización: 294 accesos (1,46% de la carga de trabajo).
- Consulta 1: 10.312 accesos (51,22% de la carga de trabajo).
- Consulta 2: 9.263 accesos (47 % de la carga de trabajo).

Analizando cada plan de ejecución individualmente, podemos apreciar algunos cambios con respecto al original.

Para la actualización, se sustituye un *table full scan* por un *index unique scan*. Esto permite reducir en gran medida su coste gracias a una muy significativa reducción en el coste de localizar el registro que se debe actualizar. Esto, junto con su alta tasa de ejecución, permite una mejora muy notoria.

Para la primera consulta, se sustituyen dos *table full scan* por *index fast scan*, lo que supone una reducción en la localización de registros dentro de la consulta. Debido a que, las operaciones originales presentaban un alto coste dentro de la operación, su optimización supone una mejora notable en el rendimiento de la consulta.

En cuanto a la segunda consulta, no tiene cambios en su plan de ejecución más allá de la reducción de costes debido a la reducción de *pctfree*.

Tras este análisis, concluimos que el nuevo diseño logra el objetivo inicial, optimizar la base de datos. Siendo la principal mejora la reducción del coste de la primera operación tras la iteración 1.

6. Conclusión

Esta práctica estaba enfocada a la optimización de una base de datos. Para ello, hemos necesitado de nuestros conocimientos en organización de ficheros, bases de datos relacionales y sqlplus.

Por esto, el desarrollo de esta nos ha permitido afianzar nuestros conocimientos en todas estas áreas. Además, la labor de documentación acerca de sqlplus ha hecho que ahora seamos mucho más competentes con este lenguaje. También, hemos adquirido nuevos conocimientos acerca de la organización interna de Oracle.

Las principales dificultades encontradas tiene relación con la poca experiencia desarrollando trabajos de este tipo. Una vez superada esta barrera hemos logrado tener un desempeño del que nos sentimos orgullosos. Destacamos como especialmente laboriosas las tareas de lectura de documentación de Oracle (la consideramos vaga, de difícil navegación y escasa en ejemplos; al punto que nos hemos encontrado más de una vez consultando fuentes de información de terceros), de pruebas en sqlplus (aunque esto se debe al mal funcionamiento de la plataforma de aula virtual) y de comprensión del paquete.

En resumen, esta práctica nos ha permitido afianzar los conocimientos adquiridos en la asignatura de Ficheros y Bases de datos, cubriendo prácticamente la totalidad de los conceptos

explorados en la asignatura. Adicionalmente, nos ha incentivado a investigar sobre sqlplus y la organización interna de Oracle; obligándonos a familiarizarnos con el manejo de documentación oficial.