| **Alumno/a:** | **David Merino Hernández** | **NIA:** | **100432028** |
| --- | --- | --- | --- |
| **Alumno/a:** | **Samuel Fernández Fernández** | **NIA:** | **100432070** |
| **Alumno/a:** | **Ángela Morales Valverde** | **NIA:** | **100432200** |

**ÍNDICE**

[**1 Introducción 1**](#_heading=h.gjdgxs)

[**2 Análisis 2**](#_heading=)

[**3 Diseño Físico 7**](#_heading=)

[Aumentar el tablespace 8](#_heading=h.qfl2tq4e8eeh)

[Creación de índices 12](#_heading=h.vdktj2jj4owi)

[Reducción del PCTFREE y uso de hints 13](#_heading=h.s4aw4y7wfdsj)

[**4 Evaluación 13**](#_heading=)

[Aumentar el tablespace 14](#_heading=h.zazy6c1jci14)

[Rendimiento global 14](#_heading=h.gc2ib04017nx)

[Rendimiento proceso 1 - modificación de fila 15](#_heading=h.q2in09fmhff2)

[Rendimiento proceso 2 - consulta 1 15](#_heading=h.jgvcqfat21d2)

[Rendimiento proceso 3 - consulta 2 16](#_heading=h.mmvh0fwobwd4)

[Creación de índices (teniendo tablespace 16k) 17](#_heading=h.1anuj8qq1rtp)

[Rendimiento global 17](#_heading=h.o72qkujr4t4d)

[Rendimiento proceso 1 - modificación de fila 18](#_heading=h.v3bfti93dr6e)

[Rendimiento proceso 2 - consulta 1 18](#_heading=h.1rcnujqsfq7p)

[Rendimiento proceso 3 - consulta 2 19](#_heading=h.h0e08133pufr)

[Reducción del PCTFREE y uso de hints (con tablespace de 16k y los índices) 20](#_heading=h.libvat9h4n2v)

[Rendimiento global 20](#_heading=h.rb3ttnc3dqsd)

[Rendimiento proceso 1 - modificación de fila 21](#_heading=h.q3qfqxhcpeke)

[Rendimiento proceso 2 - consulta 1 22](#_heading=h.ux5c09pbk3ul)

[Rendimiento proceso 3 - consulta 2 23](#_heading=h.c86h46eu12i5)

[**5 Conclusiones Finales 24**](#_heading=)

# Introducción

**Una introducción que sea el punto de partida del trabajo y sirva para analizar el problema que se va a resolver. Establece los objetivos que se persiguen, y describe los pasos que se van a seguir para alcanzarlos.**

En esta práctica partimos de la base de que vamos a inicializar de nuevo nuestra base de datos, borrando toda la información que hemos realizado hasta el momento e iniciando esta práctica con las tablas y la carga de datos que se nos proporcionaron como solución de la primera entrega.

Una vez hecho esto, hemos modificado la tabla TRACKS añadiendo las columnas que se nos piden.

El objetivo de esta práctica es conseguir medir el rendimiento del modelo inicial y el del final que consigamos obtener mediante las modificaciones que se nos piden, viendo así cuánto somos capaces de reducir los accesos a memoria y la cantidad de bloques, entre otros parámetros.

Necesitaremos ver cuales son los costes o el rendimiento de los procesos que nos marcan, teniendo en cuenta sus frecuencias relativas de uso.

# Análisis

**Explica el diseño físico actual (inicial) y describe la carga de trabajo prototípica (procesos frecuentes).**

El diseño físico actual está compuesto por las tablas y las inserciones que se propusieron como solución de la primera práctica. A eso se suman las dos consultas y el proceso de modificación de las filas que se incorporaron en la segunda práctica.

Para comprobar el rendimiento inicial que nos da ese diseño físico vamos a ejecutar el procedimiento de RUN\_TEST y así evaluar el tiempo que se consume y el número de bloques que usa en dicha ejecución. Como se indica en el enunciado, tenemos en cuenta que el resultado puede variar con el número de iteraciones que marquemos en dichos tests, por lo que hemos hecho dos ejemplos, uno con 5 iteraciones y otro con 10, para poder analizar así posibles cambios dentro del propio diseño inicial. Para ello ejecutamos el siguiente comando : **begin pkg\_costes.run\_test(nº iteraciones); end**;.

Los resultados que hemos obtenido los mostramos en la siguiente tabla:

**5 Iteraciones**

|  | TIME CONSUMPTION | CONSISTENT GETS |
| --- | --- | --- |
| 1ª vez | 3.552,2 ms | 22.109 blocks |
| 2ª vez | 3.523,4 ms | 22.109 blocks |
| 3ª vez | 3.441,4 ms | 22.109 blocks |

Como se puede apreciar, se consume un gran número de bloques que no varía independientemente de cuántas veces se ejecute, al contrario que el tiempo que se consume que es ligeramente menor cada vez que se ejecuta de nuevo. Esto se puede deber a que ciertos datos se pueden haber quedado almacenados en memoria las ejecuciones previas y así no lo tiene que cargar en la nueva ejecución.

**10 Iteraciones**

|  | TIME CONSUMPTION | CONSISTENT GETS |
| --- | --- | --- |
| 1ª vez | 3.489,1 ms | 22.109 blocks |
| 2ª vez | 3.314,5 ms | 22.109 blocks |
| 3ª vez | 3.291,7 ms | 22.109 blocks |

En este caso con 10 iteraciones vemos que el número de bloques que usamos son los mismos aunque sí que hay una notable diferencia en el tiempo que se consume, siendo en este caso más significativo el cambio entre la primera y tercera ejecución, bajando en torno a 200 ms.

De esta forma queda comprobado que el número de iteraciones que se realice sí que es significativo con referencia al tiempo que se consume, siendo inversamente proporcional, aunque no es tan relevante a la hora de medir o reducir los bloques consumidos.

Los procesos que tenemos y sus frecuencias relativas son los siguientes:

* Proceso P1: modificar una fila (selección por el atributo *searchk*) con frecuencia relativa de 0,98.
* Proceso P2: consulta 1 de la práctica 2 con frecuencia relativa de 0,01.
* Proceso P3: consulta 2 de la práctica 2 con frecuencia relativa de 0,01.

Dentro del proceso P1, que es la modificación de la fila, vemos que es el proceso que más se lleva a cabo de los tres que tenemos. Es un proceso el cual coje la columna que previamente hemos creado searchk, la copia en otra variable tabla que es changes y de ahí le asigna un valor aleatorio a la columna lyrics. Para ello usa un bucle que se hace dentro de un UPDATE.

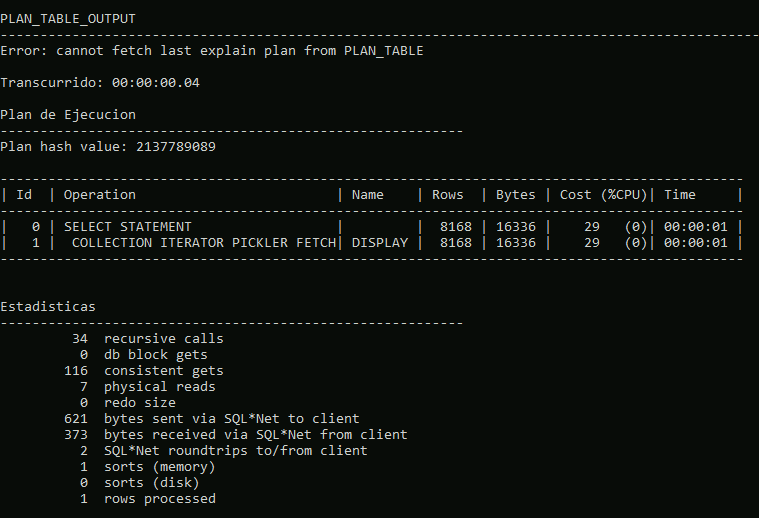
Dentro del proceso P2, que es la primera consulta de las que tenemos y calcula los intérpretes que cantan canciones escritas por ellos mismos (cuya autoría corresponde a cualquiera de los miembros de la banda o al solista, según sea el caso). La consulta proporcionará el nombre del intérprete, el porcentaje de pistas (tracks) grabadas que son canciones propias, y el porcentaje de interpretaciones en conciertos que son canciones propias.

Dentro del proceso P3, que es la segunda consulta de las que tenemos y calcula los diez intérpretes con mayor porcentaje de interpretaciones grabadas (canciones interpretadas en conciertos que ese mismo grupo ha grabado anteriormente), proporcionar la edad media de las canciones interpretadas (siendo la edad el tiempo transcurrido entre la grabación y la interpretación). La salida debe incluir el nombre del intérprete, el porcentaje de interpretaciones grabadas, y la edad media de las canciones (en años, meses, y días).

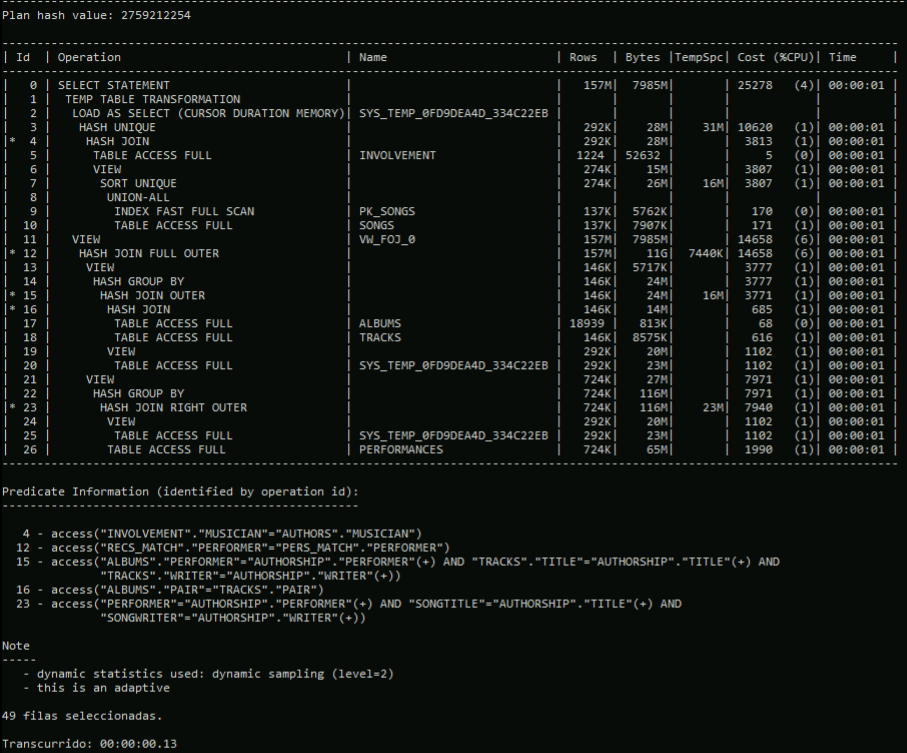
**Analiza cada instrucción de la carga de trabajo (todas las consultas, inserciones y actualizaciones). Señala las debilidades y fortalezas de ese diseño inicial atendiendo a las necesidades (carga de trabajo).**

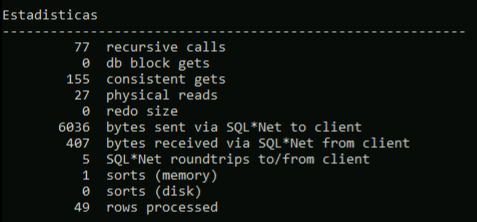
Primera instrucción de la carga de trabajo : Modificación de una fila

La base de datos hará un análisis de si la sentencia es correcta o no. Después, en caso de haber algún índice para buscar la condición WHERE, que lo habrá posteriormente, lo buscaría. Al no haber dicho índice aún, la base de datos hace una búsqueda secuencial en tracks para buscar las filas y actualizar su valor. Por último se generarán las estadísticas relevantes del proceso que son las que vamos a evaluar.

Al hacer esto, los datos que hemos obtenido son los que se muestran en la imagen que viene a continuación. Pese ha haber podido sacar unas estadísticas similares a las esperadas, hemos tenido problemas al sacar el plan de ejecución ya que la función que necesitamos usar, como se nos indica en el fichero que nos proporcionaron, ‘bulk collect’ nos sale que no está implementada dentro de la base de datos, por lo que no se llega a llenar o usar la variable changes y el UPDATE no se termina llevando a cabo de la manera correcta. Por ello, no hemos podido proporcionar la estructura de su plan de ejecución pero sí una estimación que hemos logrado extraer el rendimiento de este proceso dentro de la base de datos. 

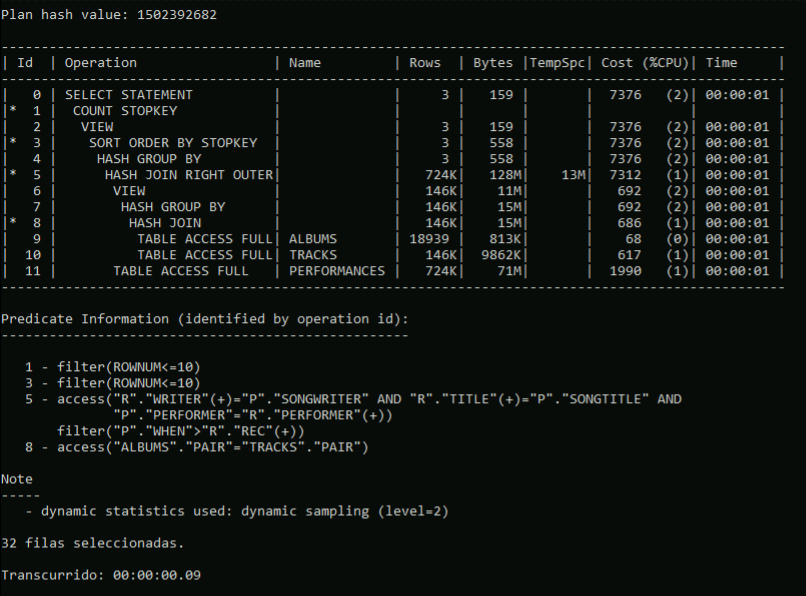
Segunda instrucción de la carga de trabajo : Primera consulta

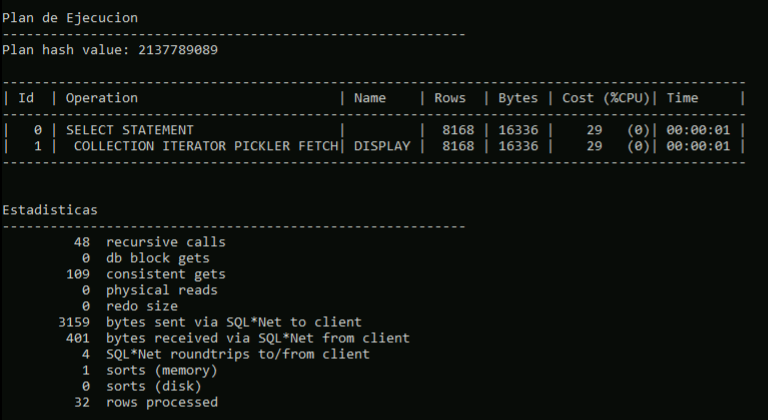
En el caso de la primera consulta, vemos como en el plan de ejecución se muestra como tiene que ir recorriendo las distintas tablas que se necesitan para esta y tiene que hacer un full access de ellas, lo que no es muy eficiente y ralentiza mucho este proceso. Por otra parte se puede ver que usa funciones de hash, lo que podría estar ayudando a la búsqueda de los datos, aunque aún así es un proceso lento. También se aprecia cómo se van haciendo procesos en paralelo. 

Se consume en elevado número de filas y de bytes, por lo que se ve claro que en posteriores optimizaciones de esta consulta como puede ser el hecho de introducir índices para las búsquedas debería reducir este consumo. 

En cuanto a las estadísticas que hemos conseguido como resultado se puede apreciar que hay 155 consistent gets y 27 lecturas a memoria.

Tercera instrucción de la carga de trabajo : Segunda consulta

Al igual que pasaba en la consulta anterior, vemos que se realiza de la misma manera, teniendo que hacer full access y consumiendo así una gran cantidad de bytes, incluso superior a los que vimos que se usaban en la consulta previa. 

En este caso hemos obtenido 109 consistent gets y 0 lecturas a memoria, esto se puede deber a que como se ha ejecutado tras la ejecución de la primera aún haya almacenados datos en caché por lo que no haya necesita cargarlos, pero en una ejecución aislada posiblemente sí que hubiese necesitado hacer al menos alguna lectura a disco. El rendimiento de esta consulta es ligeramente mejor al de la primera consulta aunque sigue teniendo mucho margen de optimización. 

Vemos en los tres casos que hay bastante margen para mejoras, por lo que posteriormente vamos a tratar de agilizar esas búsquedas para quitar el efecto negativo de los full access y tratar de aumentar el tamaño de bloque para tratar así de reducir el número de los que usamos.

**Propón mejoras al diseño físico en base a ese análisis (para las instrucciones ejecutadas individualmente) y comenta los beneficios esperados y los inconvenientes que acarree (en su caso).**

Una de las principales mejoras que planteamos a partir de este análisis que acabamos de realizar es el de conseguir bajar el número de consistent gets que hacen cada uno de los tres procesos, ya que en todos los casos superan los 100. Haciendo esto también esperamos reducir el tiempo que se consume de ejecución.

Para poder realizar estas mejoras proponemos incluir en el diseño de las tablas una ampliación del tablespace que por defecto está en 8k, poniéndolo a 16k. También proponemos incluir dentro de algunas tablas unos índices que faciliten la búsqueda de la información tanto en la modificación de la fila como en las consultas. Así abordaremos el full access y trataremos de reducir los bytes que usamos.

Otra posible optimización podría ser el hecho de reducir el PCTFREE aunque esto puede no solo no beneficiar al diseño sino empeorarlo, por lo que se deberá valorar si su uso es aplicable desde este diseño inicial que partimos.

Además, todo esto puede venir acompañado de clusters sobre los datos y de hints que hagan que el número de accesos a memoria baje al igual que el número de bloques que se usen.

Analizaremos estas implementaciones de forma gradual ya que puede darse el caso de que algunas no beneficien al diseño ya que generan mucho coste dentro de la base de datos, como podría pasar en el caso de acabar incluyendo un cluster.

# Diseño Físico

**Siguiendo el análisis realizado, propón y describe un diseño físico completo (al menos uno, puedes proponer varias alternativas). Ten en cuenta que un cambio que mejora un proceso puede estar perjudicando a otros. Justifica todas las decisiones de diseño tomadas. Implementa los diseños físicos en SQL para Oracle (incluye sólo el código nuevo).**

## Aumentar el tablespace

Queremos aumentar el tablespace con el que trabajamos de 8k a 16k ya que consideramos que de esta forma podemos reducir significativamente los consistent gets que hacemos ya que los bloques al ser más grandes deberemos consumir menos de ellos.

Para efectuar este cambio hemos modificado el script de creación de las tablas que se nos proporcionó marcando en cada una de ellas que trabaje con un tablespace de 16k. Una vez creadas de nuevo estas tablas, tendremos que volver a insertar los datos en ellas con el script que se nos facilitó y evaluar así si ha habido optimizaciones o no.

| DROP TABLE MANAGERS CASCADE CONSTRAINTS;  DROP TABLE PUBLISHERS CASCADE CONSTRAINTS;  DROP TABLE STUDIOS CASCADE CONSTRAINTS;  DROP TABLE PERFORMERS CASCADE CONSTRAINTS;  DROP TABLE MUSICIANS CASCADE CONSTRAINTS;  DROP TABLE INVOLVEMENT CASCADE CONSTRAINTS;  DROP TABLE ALBUMS CASCADE CONSTRAINTS;  DROP TABLE SONGS CASCADE CONSTRAINTS;  DROP TABLE TRACKS CASCADE CONSTRAINTS;  DROP TABLE TOURS CASCADE CONSTRAINTS;  DROP TABLE CONCERTS CASCADE CONSTRAINTS;  DROP TABLE PERFORMANCES CASCADE CONSTRAINTS;  DROP TABLE CLIENTS CASCADE CONSTRAINTS;  DROP TABLE ATTENDANCES CASCADE CONSTRAINTS;  DROP TABLE LANGUAGES CASCADE CONSTRAINTS;  DROP TABLE NATIONALITIES CASCADE CONSTRAINTS;  -- -----------------------------------------  -- validation tables  -- -----------------------------------------  CREATE TABLE LANGUAGES(  name VARCHAR2(20),  CONSTRAINT PK\_LANGUAGES PRIMARY KEY(name)  )tablespace tab\_16k;  CREATE TABLE NATIONALITIES (  name VARCHAR2(20),  CONSTRAINT PK\_NATIONALITIES PRIMARY KEY(name)  )tablespace tab\_16k;  -- -----------------------------------------  -- auxiliary tables  -- -----------------------------------------  CREATE TABLE MANAGERS(  name VARCHAR2(35) not null,  f\_name VARCHAR2(20) not null,  surname VARCHAR2(20),  mobile NUMBER(9),  CONSTRAINT PK\_MANAGERS PRIMARY KEY(mobile)  )tablespace tab\_16k;  CREATE TABLE PUBLISHERS(  name VARCHAR2(25),  phone NUMBER(9) NOT NULL,  CONSTRAINT PK\_PUBLISHERS PRIMARY KEY(name)  )tablespace tab\_16k;  CREATE TABLE STUDIOS(  name VARCHAR2(50),  address VARCHAR2(80) NOT NULL,  CONSTRAINT PK\_STUDIOS PRIMARY KEY(name)  )tablespace tab\_16k;  -- -----------------------------------------  -- musicians part  -- -----------------------------------------  CREATE TABLE PERFORMERS(  name VARCHAR2(50),  nationality VARCHAR2(20) NOT NULL,  language VARCHAR2(20) NOT NULL,  CONSTRAINT PK\_PERFORMERS PRIMARY KEY(name),  CONSTRAINT FK\_nationality FOREIGN KEY(nationality) REFERENCES nationalities,  CONSTRAINT FK\_language FOREIGN KEY(language) REFERENCES languages  )tablespace tab\_16k;  CREATE TABLE MUSICIANS (  name VARCHAR2(50) NOT NULL,  passport CHAR(14),  birthdate DATE NOT NULL,  nationality VARCHAR2(20) NOT NULL,  CONSTRAINT PK\_MUSICIANS PRIMARY KEY(passport),  CONSTRAINT FK\_MUSICIANS FOREIGN KEY(nationality) REFERENCES nationalities  )tablespace tab\_16k;  CREATE TABLE INVOLVEMENT (  band VARCHAR2(50),  musician CHAR(14),  role VARCHAR2(15),  start\_d DATE NOT NULL,  end\_d DATE,  CONSTRAINT PK\_INVOLVEMENT PRIMARY KEY(band,musician,role),  CONSTRAINT FK\_INVOLVEMENT1 FOREIGN KEY(band) REFERENCES PERFORMERS ON DELETE CASCADE,  CONSTRAINT FK\_INVOLVEMENT2 FOREIGN KEY(musician) REFERENCES MUSICIANS ON DELETE CASCADE,  CONSTRAINT CK\_INVOLVEMENT CHECK (end\_d is null OR end\_d>=start\_d)  )tablespace tab\_16k;  -- -----------------------------------------  -- works part  -- -----------------------------------------  CREATE TABLE ALBUMS(  PAIR CHAR(15),  performer VARCHAR2(50) NOT NULL,  format CHAR(1) NOT NULL, -- (T)streaming (C)CD (M)Audio File (V)Vynil (S)Single  title VARCHAR2(50) NOT NULL,  rel\_date DATE NOT NULL,  publisher VARCHAR2(25) NOT NULL,  manager NUMBER(9) NOT NULL,  CONSTRAINT PK\_ALBUMS PRIMARY KEY(PAIR),  CONSTRAINT UK\_ALBUMS UNIQUE (performer,format,title,rel\_date),  CONSTRAINT FK\_ALBUMS1 FOREIGN KEY(performer) REFERENCES PERFORMERS,  CONSTRAINT FK\_ALBUMS2 FOREIGN KEY(manager) REFERENCES MANAGERS,  CONSTRAINT FK\_ALBUMS3 FOREIGN KEY(publisher) REFERENCES PUBLISHERS,  CONSTRAINT CK\_format CHECK (format in ('T','C','M','V','S'))  )tablespace tab\_16k;  CREATE TABLE SONGS (  title VARCHAR2(50),  writer CHAR(14),  cowriter CHAR(14),  CONSTRAINT PK\_SONGS PRIMARY KEY(title, writer),  CONSTRAINT FK\_SONGS1 FOREIGN KEY(writer) REFERENCES MUSICIANS,  CONSTRAINT FK\_SONGS2 FOREIGN KEY(cowriter) REFERENCES MUSICIANS ON DELETE SET NULL,  CONSTRAINT CK\_SONGS CHECK (writer!=cowriter)  )tablespace tab\_16k;  CREATE TABLE TRACKS (  PAIR CHAR(15),  sequ NUMBER(3) NOT NULL,  title VARCHAR2(50) NOT NULL,  writer CHAR(14) NOT NULL,  duration NUMBER(4) NOT NULL, -- in seconds  rec\_date DATE NOT NULL,  studio VARCHAR2(50),  engineer VARCHAR2(50) NOT NULL,  CONSTRAINT PK\_TRACKS PRIMARY KEY(PAIR, sequ),  CONSTRAINT FK\_TRACKS1 FOREIGN KEY (PAIR) REFERENCES ALBUMS ON DELETE CASCADE,  CONSTRAINT FK\_TRACKS2 FOREIGN KEY (title, writer) REFERENCES SONGS,  CONSTRAINT FK\_TRACKS3 FOREIGN KEY (studio) REFERENCES STUDIOS ON DELETE SET NULL,  CONSTRAINT CK\_duracion CHECK (duration<=5400)  )tablespace tab\_16k;  -- -----------------------------------------  -- concerts part  -- -----------------------------------------  CREATE TABLE TOURS (  performer VARCHAR2(50),  name VARCHAR2(100),  manager NUMBER(9) NOT NULL,  CONSTRAINT PK\_TOURS PRIMARY KEY (performer,name),  CONSTRAINT UK\_TOURS UNIQUE (performer,name,manager),  CONSTRAINT FK\_TOURS FOREIGN KEY(performer) REFERENCES PERFORMERS  )tablespace tab\_16k;  CREATE TABLE CONCERTS (  performer VARCHAR2(50),  when DATE,  tour VARCHAR2(100),  municipality VARCHAR2(100) NOT NULL,  address VARCHAR2(100),  country VARCHAR2(100),  attendance NUMBER(7) DEFAULT (0) NOT NULL,  duration NUMBER(4),  manager NUMBER(9) NOT NULL,  CONSTRAINT PK\_CONCERTS PRIMARY KEY (performer,when),  CONSTRAINT FK\_CONCERTS1 FOREIGN KEY(performer) REFERENCES PERFORMERS,  CONSTRAINT FK\_CONCERTS2 FOREIGN KEY(manager) REFERENCES MANAGERS,  CONSTRAINT FK\_CONCERTS3 FOREIGN KEY(performer, tour, manager) REFERENCES TOURS(performer,name,manager)  )tablespace tab\_16k;  CREATE TABLE PERFORMANCES (  performer VARCHAR2(50),  when DATE,  sequ NUMBER(3),  songtitle VARCHAR2(100) NOT NULL,  songwriter CHAR(14) NOT NULL,  duration NUMBER(4),  CONSTRAINT PK\_PERFORMANCES PRIMARY KEY (performer,when,sequ),  CONSTRAINT FK\_PERFORMANCES1 FOREIGN KEY (performer,when) REFERENCES CONCERTS ON DELETE CASCADE,  CONSTRAINT FK\_PERFORMANCES2 FOREIGN KEY (songtitle,songwriter) REFERENCES SONGS  )tablespace tab\_16k;  -- -----------------------------------------  -- clients part  -- -----------------------------------------  CREATE TABLE CLIENTS (  e\_mail VARCHAR2(100),  name VARCHAR2(80),  surn1 VARCHAR2(80),  surn2 VARCHAR2(80),  birthdate DATE,  phone NUMBER(9),  address VARCHAR2(100),  DNI VARCHAR2(8),  CONSTRAINT PK\_CLIENTS PRIMARY KEY (e\_mail),  CONSTRAINT UK\_CLIENTS UNIQUE (DNI)  )tablespace tab\_16k;  CREATE TABLE ATTENDANCES (  client VARCHAR2(100),  performer VARCHAR2(100),  when DATE,  RFID VARCHAR2(120) NOT NULL,  purchase DATE,  CONSTRAINT PK\_ATTENDANCES PRIMARY KEY (client,performer,when),  CONSTRAINT UK\_ATTENDANCES UNIQUE (performer,when,RFID),  CONSTRAINT FK\_ATTENDANCES1 FOREIGN KEY (performer,when) REFERENCES CONCERTS ON DELETE CASCADE,  CONSTRAINT FK\_ATTENDANCES2 FOREIGN KEY (client) REFERENCES CLIENTS  )tablespace tab\_16k; |
| --- |

## 

## Creación de índices

Otra optimización que se puede llevar a cabo es la incorporación de los índices en aquellos campos por los que busquemos los datos.

Uno de los campos en los que es necesario agilizar esa búsqueda sería en el caso del searchk de la tabla TRACKS, ya que en el primer proceso este cambio haría que fuese una búsqueda mucho más dinámica y rápida. En este caso, podría dar varios problemas el incluirlo pudiendo ser un proceso aún más lento que el que disponíamos en un inicio ya que ante cualquier cambio o actualización se debería cambiar el índice entero, siendo así muy costoso. Al ser el caso precisamente de un proceso de actualización puede pasar que no nos compense añadirlo ya que se encarece demasiado. Posteriormente evaluaremos si su uso es óptimo en este diseño físico o no.

De igual manera hemos tratado de insertar más índices en aquellos campos que usamos o bien para un WHERE o para algún JOIN ya que así facilitamos su búsqueda y, en teoría, minimizamos los consistent gets.

| CREATE INDEX idx\_searchk on TRACKS(searchk);  CREATE INDEX idx\_songs\_writer ON songs(writer);  CREATE INDEX idx\_songs\_cowriter ON songs(cowriter);  CREATE INDEX idx\_involvement\_musician ON involvement(musician);  CREATE INDEX idx\_tracks\_pair ON tracks(pair);  CREATE INDEX idx\_performances\_performer ON performances(performer);  CREATE INDEX idx\_performances\_songtitle ON performances(songtitle);  CREATE INDEX idx\_performances\_songwriter ON performances(songwriter); |
| --- |

## 

## Reducción del PCTFREE y uso de hints

Nuestra base de datos por defecto tiene un valor de PCTFREE del 10%. Pensamos en que otra alternativa para hacer más óptimo este diseño podría ser el disminuir este valor, para tratar así de aprovechar al máximo los bloques de los que disponemos.

| ALTER TABLE LANGUAGES PCTFREE 5;  ALTER TABLE NATIONALITIES PCTFREE 5;  ALTER TABLE MANAGERS PCTFREE 5;  ALTER TABLE PUBLISHERS PCTFREE 5;  ALTER TABLE STUDIOS PCTFREE 5;  ALTER TABLE PERFORMERS PCTFREE 5;  ALTER TABLE MUSICIANS PCTFREE 5;  ALTER TABLE INVOLVEMENT PCTFREE 5;  ALTER TABLE ALBUMS PCTFREE 5;  ALTER TABLE SONGS PCTFREE 5;  ALTER TABLE TRACKS PCTFREE 5;  ALTER TABLE TOURS PCTFREE 5;  ALTER TABLE CONCERTS PCTFREE 5;  ALTER TABLE PERFORMANCES PCTFREE 5;  ALTER TABLE CLIENTS PCTFREE 5;  ALTER TABLE ATTENDANCES PCTFREE 5; |
| --- |

# Evaluación

**Mide el rendimiento de la base en la ejecución de la carga de trabajo estándar, tanto sobre el diseño físico inicial como sobre cada una de las alternativas implementadas. Compara y analiza los resultados obtenidos (comenta las divergencias con los resultados esperados, en su caso).**

## Aumentar el tablespace

El propósito de aumentar el tamaño de bloque es tener que recorrer menos a la hora de realizar alguna búsqueda y así reducir el valor de consistent gets que recibimos en las estadísticas. Tenemos en cuenta que podía haber una posible subida de las lecturas en memoria, pero como veremos sí que hay cambios buenos significativos.

### Rendimiento global

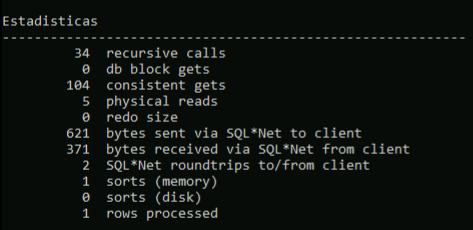
| **5 ITERACIONES** | | | | |
| --- | --- | --- | --- | --- |
|  | TIME CONSUMPTION inicial | TIME CONSUMPTION  final | CONSISTENT GETS  inicial | CONSISTENT GETS  final |
| 1ª vez | 3.552,2 ms | 6.418,6 ms | 22.109 blocks | 12.241,6 blocks |
| 2ª vez | 3.523,4 ms | 5.918,2 ms | 22.109 blocks | 12.035,4 blocks |
| 3ª vez | 3.441,4 ms | 5.397,6 ms | 22.109 blocks | 11.849,2 blocks |
| **10 ITERACIONES** | | | | |
| 1ª vez | 3.489,1 ms | 5.468,7 ms | 22.109 blocks | 12.069,1 blocks |
| 2ª vez | 3.314,5 ms | 5.341,2 ms | 22.109 blocks | 12.024,6 blocks |
| 3ª vez | 3.291,7 ms | 5.279,3 ms | 22.109 blocks | 11.982,9 blocks |

Con esta tabla podemos ver que pese a que hay una subida del tiempo de ejecución que puede ser causado porque tiene que buscar en partes más grandes, vemos que los consistent gets, es decir, el número de búsquedas en dichos bloques, sí que baja significativamente, llegando a reducirse casi a la mitad con respecto al valor que teníamos del diseño inicial. Ahora con bloques más grandes se puede almacenar más información en cada uno de ellos reduciendo así su número y su posterior búsqueda de información.

En este caso vemos que cuanto mayor sea el número de ejecuciones que realizamos, menor es el tiempo de ejecución pero de igual manera menor es el número de consistent gets que se consiguen optimizar.

### Rendimiento proceso 1 - modificación de fila

|  | INICIAL | TABLESPACE 16K |
| --- | --- | --- |
| CONSISTENT GETS | 116 | 104 |
| PHYSICAL READS | 7 | 5 |

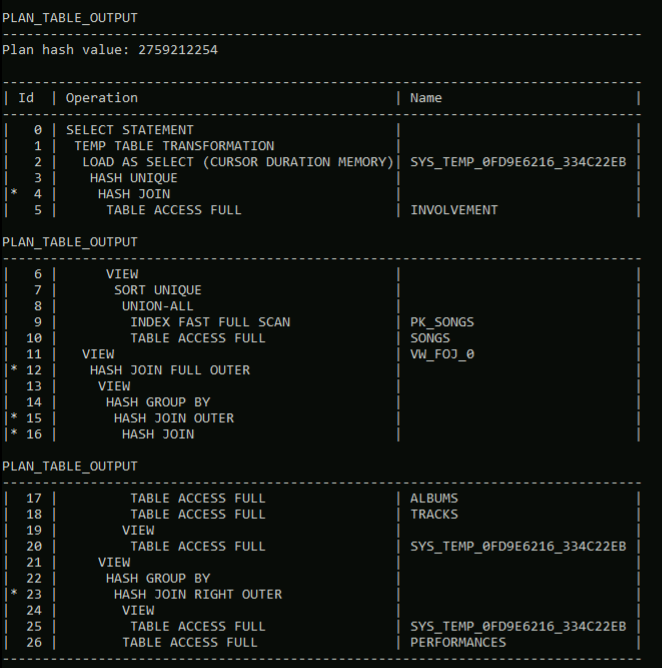


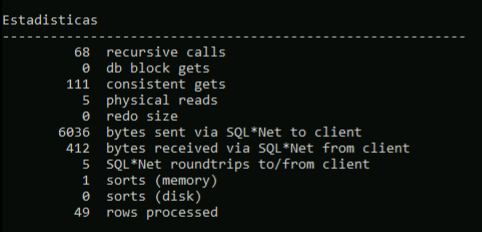
Como se puede apreciar en la tabla, se ve que ha habido una reducción de ambos valores que estamos estudiando, no es un cambio muy significativo pero sí que se aprecia que tiene cierto efecto. También el hecho de que no haya tanto efecto puede deberse a que al ser una actualización de datos, se tienen que buscar de igual manera y recorrer todos los bloques en los que esté la información, por lo que sigue siendo un proceso lento.

### Rendimiento proceso 2 - consulta 1

|  | INICIAL | TABLESPACE 16K |
| --- | --- | --- |
| CONSISTENT GETS | 155 | 111 |
| PHYSICAL READS | 27 | 5 |

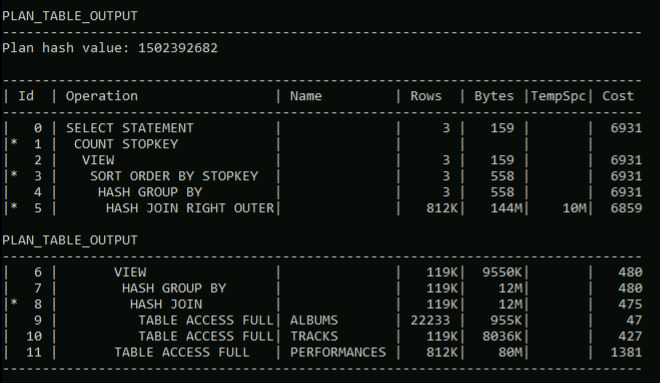
Vemos que en este caso el número de lecturas en bloques o consistent gets baja bastante pero donde se puede observar mayor diferencia es en las lecturas en memoria, ya que baja más de cinco veces el valor que se tenía en el diseño inicial.

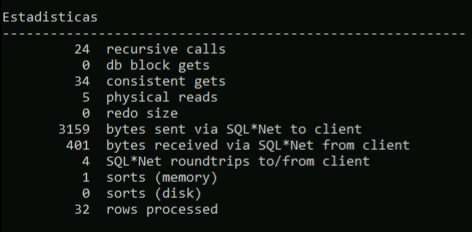
Vemos que el plan de ejecución no ha cambiado ya que el cambio que hemos introducido no le afecta directamente.



### Rendimiento proceso 3 - consulta 2

|  | INICIAL | TABLESPACE 16K |
| --- | --- | --- |
| CONSISTENT GETS | 109 | 34 |
| PHYSICAL READS | 0 | 5 |



En el caso de esta segunda consulta vemos que los consistent gets bajan bastante. Estos datos no son del todo fiables ya que el valor inicial del que partimos supusimos que sí que tendría que haber alguna lectura a disco y en este caso, igual no es que haya subido si no que se haya mantenido al igual que el rendimiento inicial que teníamos del diseño. Aún así, vemos que sí que optimizaría esta consulta. También como esta depende de la consulta 1 y esa sí que se ha visto bastante optimizada, está en consecuencia también sufre una mejora en su tiempo de ejecución y en las lecturas de bloques. 

Vemos que el diseño de ejecución no ha cambiado ya que el cambio que hemos introducido no le afecta.

## Creación de índices (teniendo tablespace 16k)

Además de introducir el cambio que anteriormente hemos evaluado, hemos decidido introducir índices auxiliares en aquellas variables que más usamos para las búsquedas y para hacer las combinaciones. En los siguientes apartados vamos a ver si esta decisión es acertada o en qué casos sería mejor no introducirlos.

### Rendimiento global

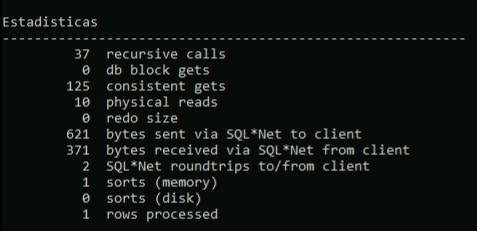
| **5 ITERACIONES** | | | | |
| --- | --- | --- | --- | --- |
|  | TIME CONSUMPTION inicial | TIME CONSUMPTION  final | CONSISTENT GETS  inicial | CONSISTENT GETS  final |
| 1ª vez | 3.552,2 ms | 6.492 ms | 22.109 blocks | 11.338,6 blocks |
| 2ª vez | 3.523,4 ms | 5.733,2 ms | 22.109 blocks | 11.171,2 blocks |
| 3ª vez | 3.441,4 ms | 4.990,4 ms | 22.109 blocks | 11.129,2 blocks |
| **10 ITERACIONES** | | | | |
| 1ª vez | 3.489,1 ms | 5794,5 ms | 22.109 blocks | 11.117,9 blocks |
| 2ª vez | 3.314,5 ms | 5.500,3 ms | 22.109 blocks | 11.111,4 blocks |
| 3ª vez | 3.291,7 ms | 5.233,1 ms | 22.109 blocks | 11.065,8 blocks |

En cuanto a este rendimiento inicial vemos que el número de lecturas de bloques se reduce a la mitad aunque hay una subida del tiempo de ejecución, pese a esto, sí que resulta una optimización eficiente. Estos tiempos son estimados ya que sufren de algunos problemas con el sqlplus por lo que podrían ser un poco superiores a los tiempos que nos saldrían en condiciones idóneas.

Vemos que con más iteraciones el tiempo disminuye pero más poco a poco y que con menos es un cambio más abrupto, esas disminuciones de tiempos es debido a los datos que se van quedando almacenados en caché.

### Rendimiento proceso 1 - modificación de fila

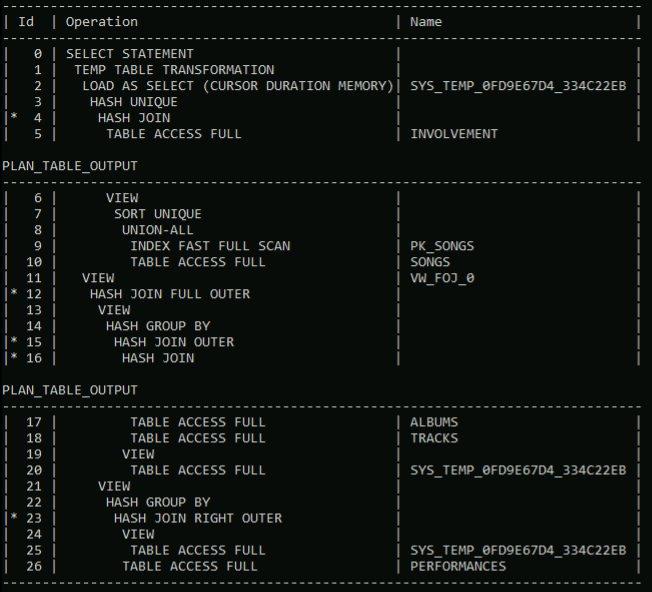
|  | INICIAL | TABLESPACE 16K (\*1) | 1 + ÍNDICES |
| --- | --- | --- | --- |
| CONSISTENT GETS | 116 | 104 | 125 |
| PHYSICAL READS | 7 | 5 | 10 |

Vemos que en este caso, al tratarse de un proceso de actualización, el uso de índices como puede ser el caso del que incluimos en la variable ‘searchk’ no es el mejor proceso a llevar a cabo si se quiere optimizar. Cuando hay un cambio en los datos supone que el índice entero tiene que actualizarse también, siendo así un proceso muy costoso. De esta forma, no sería una buena forma de reducir los consistent gets en este proceso. 

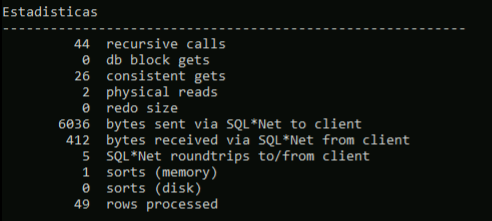
### Rendimiento proceso 2 - consulta 1

|  | INICIAL | TABLESPACE 16K | 1 + ÍNDICES |
| --- | --- | --- | --- |
| CONSISTENT GETS | 155 | 111 | 26 |
| PHYSICAL READS | 27 | 5 | 2 |

Vemos que, la combinación de aumentar el tablespace al doble de tamaño que tenían sumado a los índices que hemos generado para los valores que más usamos, vemos que el número de consistent gets baja más de una cuarta parte de lo que teníamos en el caso anterior.

Al ser una consulta que consulta tantos datos de distintas tablas, el hecho de agilizar esa búsqueda de datos hace que se tarde mucho menos en hacer las distintas combinaciones que vamos necesitando. 

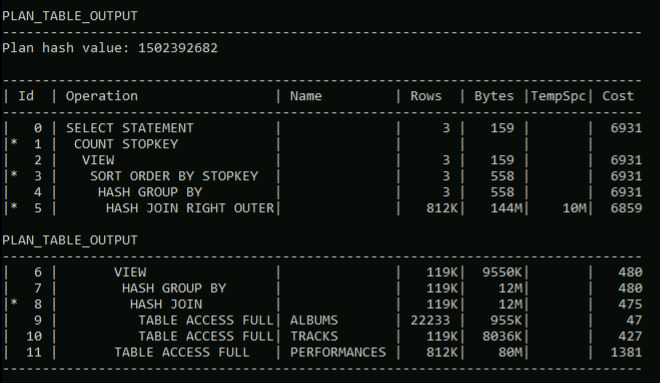
Además de esto, se podría hacer que los índices en vez de ser completos, se podría hacer un hash sobre ellos, reduciendo así su tamaño y agilizando aún más dicha búsqueda.

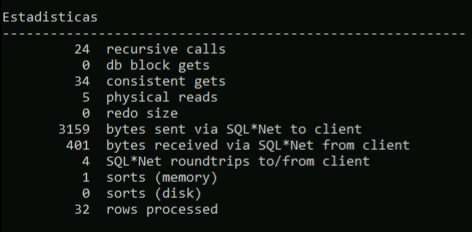


### Rendimiento proceso 3 - consulta 2

|  | INICIAL | TABLESPACE 16K | 1 + ÍNDICES |
| --- | --- | --- | --- |
| CONSISTENT GETS | 109 | 34 | 25 |
| PHYSICAL READS | 0 | 5 | 5 |

En el caso de esta segunda consulta, vemos que no hay muchos cambios, esto puede deberse a que como esta segunda selecciona información que previamente hemos buscado en la primera consulta, no hay demasiada carga de información. Si que baja mínimamente los consistent gets con los índices, pero este proceso no sufre una gran optimización con este método.

Aunque se pudiera pensar que entonces no merecería la pena introducir los índices a estos dos procesos, sí que como procedimiento resulta apto para mejorar el diseño físico inicial del que partíamos. 



## Reducción del PCTFREE y uso de hints (con tablespace de 16k y los índices)

Habiendo visto que los anteriores cambios que hemos realizado, a rasgos generales, sí que sirven para mejorar el diseño físico del que partíamos, decidimos seguir aún más tratando de modificar el PCTFREE. Teniendo ya incorporados los otros dos cambios, vamos a evaluar el caso en el que el valor del PCTFREE sea del 5% en vez del 10% como viene por defecto. Así ocuparemos más espacio de los bloques, y así trataremos de reducir el número de bloques que usamos.

### Rendimiento global

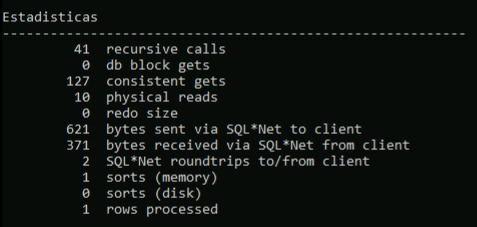
| **5 ITERACIONES** | | | | |
| --- | --- | --- | --- | --- |
|  | TIME CONSUMPTION inicial | TIME CONSUMPTION  final | CONSISTENT GETS  inicial | CONSISTENT GETS  final |
| 1ª vez | 3.552,2 ms | 5.484,6 ms | 22.109 blocks | 11.018,2 blocks |
| 2ª vez | 3.523,4 ms | 5.152,6 ms | 22.109 blocks | 10.978 blocks |
| 3ª vez | 3.441,4 ms | 5.047,2 ms | 22.109 blocks | 10.978 blocks |
| **10 ITERACIONES** | | | | |
| 1ª vez | 3.489,1 ms | 5.441,5 ms | 22.109 blocks | 10.975,8 blocks |
| 2ª vez | 3.314,5 ms | 5.235,9 ms | 22.109 blocks | 10.904,4 blocks |
| 3ª vez | 3.291,7 ms | 4.854,3 ms | 22.109 blocks | 10.864 blocks |

Se aprecia que, pese a que de esta forma sigue subiendo un poco el tiempo de ejecución total, sí que conseguimos una gran mejora en cuanto a los consistent gets, siendo menos de la mitad de los que partíamos en un inicio.

Esto se puede deber a que estemos aprovechando mejor los bloques de los que disponemos y eso haga que la búsqueda dentro de dichos bloques sea mucho más rápida y eficiente. Por tanto, el conjunto de los tres cambios aplicados a la vez nos lleva a una gran mejora del diseño físico del que partíamos en un inicio.

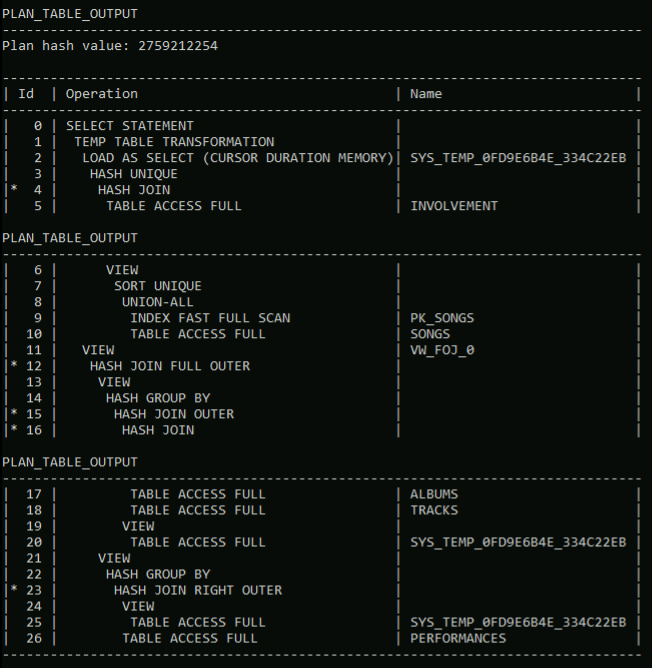
### Rendimiento proceso 1 - modificación de fila

|  | INICIAL | TAB 16K (\*1) | 1 + ÍNDICES (\*2) | 1+2+PCTFREE 5% |
| --- | --- | --- | --- | --- |
| CONSISTENT GETS | 116 | 104 | 125 | 127 |
| PHYSICAL READS | 7 | 5 | 10 | 10 |

Al igual que con el último cambio que introdujimos, al ser un proceso UPDATE estos cambios no le son muy beneficiosos, empeorando ligeramente con respecto al anterior y con respecto al inicial. Aunque no es un empeoramiento muy significativo si que se puede apreciar que por la naturaleza del proceso, tanto los índices como la reducción del PCTFREE son muy costosos de mantener para este tipo de diseño.

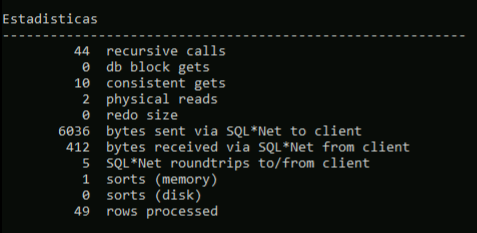
### Rendimiento proceso 2 - consulta 1

|  | INICIAL | TAB 16K (\*1) | 1 + ÍNDICES (\*2) | 1+2+PCTFREE 5% |
| --- | --- | --- | --- | --- |
| CONSISTENT GETS | 155 | 111 | 26 | 10 |
| PHYSICAL READS | 27 | 5 | 2 | 2 |



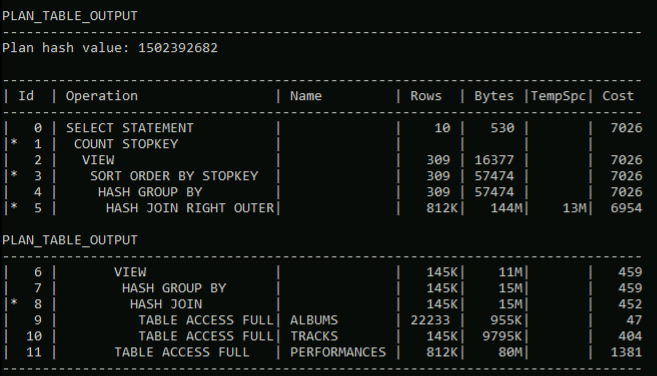
Al contrario que pasa con el proceso 1, esta primera consulta si que se ve drásticamente optimizada con respecto al valor inicial del que partíamos gracias a estas tres mejoras que hemos ido introduciendo.

Así, hemos conseguido que todas las combinaciones de tablas que teníamos se hagan de una manera mucho más estructurada y por tanto más rápida, teniendo así que recorrer bastantes menos bloques para obtener la información que se necesita.



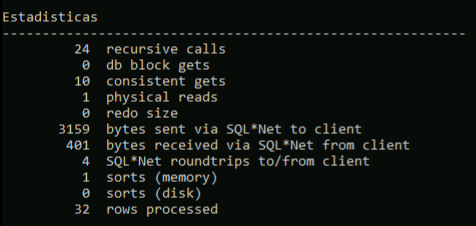
### Rendimiento proceso 3 - consulta 2

|  | INICIAL | TAB 16K (\*1) | 1 + ÍNDICES (\*2) | 1+2+PCTFREE 5% |
| --- | --- | --- | --- | --- |
| CONSISTENT GETS | 109 | 34 | 25 | 10 |
| PHYSICAL READS | 0 | 5 | 5 | 1 |



Al igual que pasaba con la mejora previa, el gran cambio se lo lleva la consulta primera aunque en esta también se aprecie que este cambio ha reducido a más de la mitad los consistent gets y los accesos a memoria.

Al igual que pasaba con el proceso 2, de esta forma estamos aprovechando al máximo todo el espacio de los bloques de los que disponemos, que de por sí ya son más grandes, y reduciendo el espacio libre hacemos que sea más eficiente.



El único inconveniente que hemos tenido en cuenta que pudiese ser perjudicial para la base de datos el reducir este valor es que en caso de posibles actualizaciones futuras, al tener menos PCTFREE, podría provocar que hubiese más desbordamiento. Si ese desbordamiento fuese significativamente grande podría requerir un cambio de la base de datos ya sea aumentando el tamaño de bloque de nuevo o subiendo de nuevo este valor de espacio libre.

Para nuestro diseño y con las condiciones de las que partimos sí que creemos que puede ser una solución para mejorar el rendimiento de nuestra base de datos.

# Conclusiones Finales

**Exponed vuestras conclusiones sobre esta práctica. Reflexionad sobre los resultados obtenidos (si son buenos o no, y por qué), la herramienta utilizada, posibilidades futuras, etc.**

En esta práctica de Diseño Físico en Oracle hemos tenido la oportunidad de comprobar la optimización de bases de datos. Nuestro objetivo era mejorar el rendimiento del sistema y aprender cómo monitorizar y ajustar una base de datos.

Comenzamos estableciendo el entorno de evaluación, eliminando objetos innecesarios y ejecutando los scripts de creación y población de la base de datos. También agregamos nuevas columnas a la tabla "tracks" mediante el código suministrado. Luego, analizamos las transacciones más costosas y creamos una carga de trabajo que reflejaba la frecuencia de cada proceso. A continuación, estudiamos en detalle el diseño físico, las operaciones a ejecutar y los planes de ejecución. Propusimos cambios para reducir los accesos a memoria secundaria y mejorar el tiempo de ejecución.

En conclusión, esta práctica nos permitió aplicar nuestros conocimientos y aprender sobre la optimización de bases de datos. Aprendimos cómo mejorar el rendimiento de un sistema y la importancia del diseño físico.

**Después, comentad vuestro desempeño en esta práctica (esfuerzo requerido, conocimiento que reporta, progreso, etc.). Comentad también vuestro desempeño en todas las prácticas de manera conjunta. También podéis proponer mejoras en el planteamiento de la práctica para el futuro (enfoque, dimensión del problema, conocimiento requerido, materiales de soporte, elementos que os hubiera gustado haber podido practicar pero que la práctica no contempla, etc.).**

Esta práctica requería un esfuerzo considerable para comprender y aplicar los conceptos de diseño físico en una base de datos real. A medida que avanzábamos, adquirimos un conocimiento más profundo sobre cómo el diseño físico puede afectar el rendimiento de la base de datos además, a lo largo de todas las prácticas, experimentamos un progreso significativo. Desde la creación inicial de la base de datos hasta el análisis y la optimización, pudimos aplicar de manera práctica los conocimientos adquiridos en clase

Sin embargo, creemos que la práctica podría haberse beneficiado de instrucciones más claras y una estructura más guiada. En ocasiones, nos sentimos un poco perdidos y nos llevó tiempo comprender completamente los pasos a seguir y los objetivos específicos. Además, nos habría gustado tener más ejemplos y ejercicios prácticos para practicar los conceptos antes de abordar la tarea principal ya que nos atascamos al principio con el análisis de datos de carga.

**Finalmente, comentad vuestro desempeño en la asignatura en general, y vuestra opinión en la estructura y enfoque de la asignatura de cara a su evolución: temas que no encontráis relevantes, otros temas que no están pero os hubiera gustado que se trataran o que sí están pero os hubiera gustado ver en mayor profundidad, etc.).**

En general, nos sentimos satisfechos con el aprendizaje obtenido y el progreso que hemos logrado a lo largo del curso, además, la estructura de la asignatura fue en su mayoría adecuada, ya que nos permitió adquirir una comprensión sólida de los conceptos fundamentales de los sistemas de bases de datos. Apreciamos la combinación de teoría y práctica, que nos permitió aplicar los conocimientos adquiridos en ejercicios y proyectos prácticos.