Titulación: GRADO INGENIERIA INFORMATICA
Año Académico: 2013/2014
Curso: 2º
Asignatura: Ficheros y Bases de Datos
Título: Memoria Práctica 3 – Diseño Físico en Oracle

UNIVERSIDAD CARLOS III DE MADRID

Profesor:	**Francisco Javier Calle**		
Grupo Peq:	**User0341**	**Grupo**	**82**
Alumno/a:	**Carlos Contreras Sanz**	**NIA:**	**100303562**
Alumno/a:	**Álvaro Gómez Ramos**	**NIA:**	**100307009**

1 Introducción

En este trabajo consiste en medir el rendimiento de la carga de trabajo estándar que nos han dicho en el enunciado de la práctica. Mediante el uso de un script que se nos ha dado podremos medir el tiempo y número de veces que se accede a un bloque del disco, ya que los accesos a memoria intermedia no los incluye. Tras realizar unas cuantas mediciones nos piden crear una serie de índices, clúster,… que consigan mejorar el rendimiento.

A continuación mostraremos pruebas como capturas realizadas a las mediciones de rendimiento y explicaremos el porque nos hemos decidido a utilizar los índices que hemos usado para mejorar el rendimiento.

2 Análisis

El diseño Físico inicial que posee nuestro diseño es el siguiente:

- Tamaño de cubo 8KB, el tamaño de cubo predefinido que utiliza Oracle.
- Los ficheros de datos se organizan de manera serial no-consecutiva
- Utiliza cubos con un espacio libre distribuido PCTFREE = 10 y PCTUSED = 60
- No utilizamos ningún índice creado por nosotros, solo los generados automáticamente por Oracle.

La carga de trabajo estándar consta de:

- Consultas, 5 consultas en total.
- Inserciones:
    - Inserción de Deportista.
    - Inserción de Federado.
    - Inserción de Fichas (2 Fichas).
    - Inserción de Controles (6 Controles).
    - Inserción de Evidencia.
- Vistas, 2 vistas en total.
- Borrado de Deportista.

Para ver los puntos débiles de nuestro diseño hemos decidido probar cada una de las tareas incluidas en la carga de trabajo estándar, y así poder ver los accesos a bloque de cada una y poder comparar y ver las debilidades más fácilmente.

2.1. Coste de inserciones del diseño físico inicial.

Las inserciones se pueden resumir en 1,27 accesos a bloque de media, la única que puede resaltar un poco es la inserción en la tabla deportistas, la mas costosa con 4 accesos.

2.2. Coste de Consultas del diseño físico inicial.

Las consultas las veremos individualmente ya que tienen un mayor coste. Comentaremos cada una de las consultas para ver sise puede mejorar con un índice.

Consulta 1:

Los resultados obtenidos en la consulta 1 son:

```
|   0 | SELECT STATEMENT             |            |     1 |    74 |     3   (0)|
 00:00:01 |

|   1 |  SORT AGGREGATE              |            |     1 |    74 |            |
          |

|*  2 |   TABLE ACCESS BY INDEX ROWID| GANADOR    |     1 |    74 |     3   (0)|
 00:00:01 |

|*  3 |    INDEX RANGE SCAN          | PK_GANADOR |     1 |       |     2   (0)|
 00:00:01 |

-------------------------------------------------------------------------------
-----------


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

   2 - filter("FECHA_GANA"=2013)
   3 - access("DEPORTE"='futbol' AND "NOM_COMPETICION"='Copa del Rey')

Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
----------------------------------------------------------
        237  recursive calls
          0  db block gets
        145  consistent gets
```

Vemos que hace 145 accesos a bloque, para conseguir este número de accesos vemos como accede a la tabla ganador mediante el índice que crea SQL automáticamente por clave primaria, el INDEX PK_GANADOR.

En esta Consulta se podría crear un índice sobre la tabla Ganador en el atributo Deporte para intentar ahorrar algunos accesos.

Consulta 2:

Los resultados obtenidos en la consulta 2 son:

```
|   0 | SELECT STATEMENT          |               |     1 |    16 |   112   (3)|
 00:00:02 |

|   1 |  SORT AGGREGATE           |               |     1 |    16 |            |
          |

|   2 |   NESTED LOOPS            |               |  2031 | 32496 |   112   (3)|
 00:00:02 |

|   3 |    VIEW                   |               |  2031 | 16248 |   111   (2)|
 00:00:02 |

|   4 |     MINUS                 |               |       |       |            |
          |

|   5 |      SORT UNIQUE          |               |  2031 | 34527 |            |
          |

|*  6 |       INDEX FAST FULL SCAN| PK_FEDERADO   |  2031 | 34527 |   102   (0)|
 00:00:02 |

|   7 |      SORT UNIQUE          |               |  1197 | 20349 |            |
          |

|*  8 |       TABLE ACCESS FULL   | EVIDENCIA     |  1197 | 20349 |     7   (0)|
 00:00:01 |

|*  9 |    INDEX UNIQUE SCAN      | SYS_C00935841 |     1 |     8 |     0   (0)|
 00:00:01 |

---------------------------------------------------------------------------
-----------


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

   6 - filter("DEPORTE"='Ciclismo')
   8 - filter("SUSTANCIA" IS NOT NULL)
   9 - access("SUJETO_CONTROL"="CID")

Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
-----------------------------------------------------------

          0  recursive calls
          0  db block gets
        499  consistent gets
```

Vemos que esta consulta hace 499 accesos, también podemos ver que hace un full scan sobre la tabla Evidencia, y utiliza un índice sobre la tabla Federado por clave primaria que tiene la mayor parte del coste de esta consulta. Por lo que tras mirar la consulta vemos que busca federados por su deporte, lo que nos da la idea de crear un índice de Federado por el atributo Deporte.

CREATE INDEX in_federado ON Federado (Deporte);

Tras probarlo vemos como este indice en vez de mejorar nos empeora sumando unos 300 accesos mas de los que el diseño inicial nos hacia, por lo que no lo incluimos. También probamos el índice:

CREATE INDEX in_evidencia ON Evidencia (Sujeto_Control);

Para ver si puede mejorar esta y otras consultas, como en la consulta 4 y consulta 5, ya que siempre que accedemos a Evidencia es para devolver el Sujeto_Control, pero tras probarlo no mejora, empeorando levemente el resultado global.

Consulta 3:

Los resultados obtenidos en la consulta 3 son:

```
|   6 |       SORT UNIQUE                  |                 | 51940 |  2231K|
 2880K|   901   (2)| 00:00:11 |

|   7 |        UNION-ALL                    |                 |       |       |
      |            |          |

|*  8 |         HASH JOIN                   |                 | 51939 |  2231K|
      |   312   (1)| 00:00:04 |

|   9 |          TABLE ACCESS FULL          | CATEGORIA       |     7 |   140 |
      |     3   (0)| 00:00:01 |

|  10 |          TABLE ACCESS FULL          | FICHA           | 51939 |  1217K|
      |   308   (1)| 00:00:04 |

|  11 |         NESTED LOOPS                |                 |       |       |
      |            |          |

|  12 |          NESTED LOOPS               |                 |     1 |    44 |
      |     3   (0)| 00:00:01 |

|  13 |           TABLE ACCESS FULL         | HISTORICO_FICHA |     1 |    24 |
      |     2   (0)| 00:00:01 |

|* 14 |           INDEX UNIQUE SCAN         | SYS_C00935836   |     1 |       |
      |     0   (0)| 00:00:01 |

|  15 |          TABLE ACCESS BY INDEX ROWID| CATEGORIA       |     1 |    20 |
      |     1   (0)| 00:00:01 |

--------------------------------------------------------------------------------
-------------------------------


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

   3 - filter(MAX("NUMERO")>COUNT("NUMERO"))
   8 - access("CATEGORIA"."NOMBRE"="CATEGORIA")
  14 - access("CATEGORIA"."NOMBRE"="HISTORICO_FICHA"."CATEGORIA")

Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
----------------------------------------------------------
        675  recursive calls
          0  db block gets
       1431  consistent gets
```

Vemos que esta consulta hace 1431 accesos, también podemos ver que hace un full scan sobre las tablas Categoría, Ficha e Historico_ficha. Es el acceso a Ficha el que tiene la mayor parte del coste de esta consulta. Por lo que tras mirar la consulta vemos que busca en Ficha por Deportista, Categoría y Deporte, lo que nos da la idea de crear un índice de Ficha por esos atributos.

CREATE IDNEX in_ficha ON Ficha(Deportista, Categoría, Deporte);

Tras probarlo vemos como este índice si que nos mejora los accesos, bajándolos a un tercio aproximadamente para esta consulta, sin perjudicar al global. Esto se debe a que hemos puesto el hint sobre esta consulta, a continuación probaremos a usar este mismo índice para el resto de consultas usan la tabla Ficha.

Consulta 4:

Los resultados obtenidos en la consulta 4 son:

```
|*  7 |       HASH JOIN            |                 |  3974 |   395K|       |
 317   (1)| 00:00:04 |

|   8 |        VIEW                | VW_DTP_82597CAA |  1197 | 21546 |       |
   8  (13)| 00:00:01 |

|   9 |         HASH UNIQUE        |                 |  1197 | 32319 |       |
   8  (13)| 00:00:01 |

|* 10 |          TABLE ACCESS FULL | EVIDENCIA       |  1197 | 32319 |       |
   7   (0)| 00:00:01 |

|  11 |        INDEX FAST FULL SCAN| PK_FICHA        | 51939 |  4260K|       |
 308   (1)| 00:00:04 |

|* 12 |       HASH JOIN            |                 |     1 |   111 |       |
  10  (10)| 00:00:01 |

|  13 |        TABLE ACCESS FULL   | HISTORICO_FICHA |     1 |    84 |       |
   2   (0)| 00:00:01 |

|* 14 |        TABLE ACCESS FULL   | EVIDENCIA       |  1197 | 32319 |       |
   7   (0)| 00:00:01 |

--------------------------------------------------------------------------------
----------------------


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

   2 - access("GANADOR"."NOM_CLUB"="NOMBRE_CLUB" AND "GANADOR"."PAIS"="PAIS_COMP
ARA")

       filter("from$_subquery$_003"."CONTROL_1">="GANADOR"."FECHA_GANA"-1 AND
              "from$_subquery$_003"."CONTROL_1"<="GANADOR"."FECHA_GANA")
   7 - access("FICHA"."DEPORTISTA"="ITEM_1")
  10 - filter("EVIDENCIA"."SUSTANCIA" IS NOT NULL)
  12 - access("HISTORICO_FICHA"."DEPORTISTA"="EVIDENCIA"."SUJETO_CONTROL")
  14 - filter("EVIDENCIA"."SUSTANCIA" IS NOT NULL)

Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
----------------------------------------------------------
        370  recursive calls
          0  db block gets
       2002  consistent gets
```

Vemos que esta consulta realiza 2002 accesos. Hace full scan a la tabla Evidencia (dos veces) y al índice que se crea con la clave primaria de la tabla Ficha. Ya hemos descartado el índice de Sujeto_control para Evidencia, por lo que intentamos mejorar el número de accesos a Ficha, observando la consulta, con un índice para la condición del join de la consulta.

CREATE INDEX in_ficha_dep ON ficha(deportista);

Tras probarlo, vemos que si que mejora los accesos, en al menos un 20%.

Consulta 5:

Los resultados obtenidos en la consulta 5 son:

```
|  10 |          UNION-ALL             |                 |       |       |
     |          |

|* 11 |           HASH JOIN            |                 |  3974 |   357K|   317
   (1)| 00:00:04 |

|  12 |            VIEW                |                 |  1197 |  9576 |     8
  (13)| 00:00:01 |

|  13 |             HASH UNIQUE        |                 |  1197 | 20349 |     8
  (13)| 00:00:01 |

|* 14 |              TABLE ACCESS FULL | EVIDENCIA       |  1197 | 20349 |     7
   (0)| 00:00:01 |

|  15 |            INDEX FAST FULL SCAN| PK_FICHA        | 51939 |  4260K|   308
   (1)| 00:00:04 |

|* 16 |           HASH JOIN            |                 |     1 |   101 |    10
  (10)| 00:00:01 |

|  17 |            TABLE ACCESS FULL   | HISTORICO_FICHA |     1 |    84 |     2
   (0)| 00:00:01 |

|* 18 |            TABLE ACCESS FULL   | EVIDENCIA       |  1197 | 20349 |     7
   (0)| 00:00:01 |

--------------------------------------------------------------------------------
------------------


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

   4 - access("NOMBRE"="NOMBRE_CLUB" AND "CLUB"."PAIS"="C"."PAIS")
   7 - filter(COUNT(*)>3)
  11 - access("FICHA"."DEPORTISTA"="SUJETO_CONTROL")
  14 - filter("SUSTANCIA" IS NOT NULL)
  16 - access("HISTORICO_FICHA"."DEPORTISTA"="SUJETO_CONTROL")
  18 - filter("EVIDENCIA"."SUSTANCIA" IS NOT NULL)

Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
----------------------------------------------------------
        179  recursive calls
          0  db block gets
       1397  consistent gets
```

Vemos que esta consulta realiza 1397 accesos. Hace full scan de las tablas Evidencia e Historico_ficha, además de al indice creado por la clave primaria de Ficha. Habiendo descartado mejoras para Evidencia en otras consultas (se accede de la misma forma), y teniendo Historico_ficha, lo que vamos a probar es si podemos mejorar el acceso a Ficha con el índice de Deportista, de la consulta 4, pero no lo mejora, lo empeora bastante.

2.3. Coste de Consultas del diseño físico inicial.

Vista 1:
Los resultados obtenidos en la vista 1 son:

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     1 |   314 |  1844   (1)| 00:00:23 |
|   1 |  SORT AGGREGATE     |          |     1 |   314 |            |          |
|*  2 |   HASH JOIN         |          |  3907 |  1198K|  1844   (1)| 00:00:23 |
|   3 |    TABLE ACCESS FULL| GANADOR  |  3907 |   599K|    23   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| INSCRITO |   345K|    51M|  1818   (1)| 00:00:22 |
-------------------------------------------------------------------------------

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

   2 - access("INSCRITO"."DIVISION"="GANADOR"."DIVISION" AND
              "INSCRITO"."EDICION"="GANADOR"."EDICION" AND
              "INSCRITO"."NOM_COMPETICION"="GANADOR"."NOM_COMPETICION" AND
              "INSCRITO"."CATEGORIA"="GANADOR"."CATEGORIA" AND
              "INSCRITO"."DEPORTE"="GANADOR"."DEPORTE" AND
              "INSCRITO"."PAIS"="GANADOR"."PAIS" AND
              "INSCRITO"."NOM_CLUB"="GANADOR"."NOM_CLUB")

Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
----------------------------------------------------------
        355  recursive calls
          0  db block gets
      13661  consistent gets
```

Vemos que esta vista realiza 13661 accesos. Como hace full scan a Ganador y a Inscrito al hacer la join, pensamos que hacer un indice para los campos en común de las tablas mejoraría, ya que es por los atributos por los que se hace la natural join.

CREATE INDEX in_inscrito_v2 ON inscrito(Nom_Club, Pais, Deporte, Categoria, Nom_Competicion, Edicion, Division);

Con esto mejoramos los accesos de esta vista a la mitad.

Vista 2:
Los resultados obtenidos en la vista 2 son:

```
| Id  | Operation                | Name     | Rows  | Bytes |TempSpc| Cost (%CPU
)| Time     |

-------------------------------------------------------------------------------
-------------

|   0 | SELECT STATEMENT         |          |     1 |       |       |  3580   (1
)| 00:00:43 |

|   1 |  SORT AGGREGATE          |          |     1 |       |       |
 |          |

|   2 |   VIEW                   |          | 51939 |       |       |  3580   (1
)| 00:00:43 |

|   3 |    MINUS                 |          |       |       |       |
 |          |

|   4 |     SORT UNIQUE          |          | 51939 |   405K|   824K|   500   (1
)| 00:00:07 |

|   5 |      INDEX FAST FULL SCAN| PK_FICHA | 51939 |   405K|       |   308   (1
)| 00:00:04 |

|   6 |     SORT UNIQUE          |          |   345K|  2698K|  5432K|  3080   (1
)| 00:00:37 |

|   7 |      TABLE ACCESS FULL   | INSCRITO |   345K|  2698K|       |  1815   (1
)| 00:00:22 |

-------------------------------------------------------------------------------
-------------


Note
-----
   - dynamic sampling used for this statement (level=2)


Estadísticas
----------------------------------------------------------
          7  recursive calls
          0  db block gets
      14558  consistent gets
```

Vemos que esta vista realiza 14558 accesos. Vemos que realiza un full scan a la tabla inscrito. Observando la selección de la vista, vemos que se puede reducir estos accesos con un índice para acceder a Incrito por Deportista.

CREATE INDEX in_inscrito ON inscrito(deportista);

Ademas para bajar los accesos al índice creado por la clave primaria de ficha, se hace uso del índice de ficha por deportista que usábamos para la consulta 4. Con todo ello conseguimos bajar los accesos de esta vista a casi una decima parte.

2.4. Coste de Borrado de Deportista del diseño físico inicial.
Los resultados que hemos obtenido al borrar un deportista son:

```
----

|   0 | DELETE STATEMENT   |               |     1 |     8 |     1   (0)| 00:00:
01 |

|   1 |  DELETE            | DEPORTISTA    |       |       |            |
   |

|*  2 |   INDEX UNIQUE SCAN| SYS_C00935841 |     1 |     8 |     1   (0)| 00:00:
01 |

---------------------------------------------------------------------------
----


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

   2 - access("CID"='AAAAAAAAAAAA')


Estadísticas
----------------------------------------------------------
        208  recursive calls
         74  db block gets
       1308  consistent gets
```

Este resultado se debe a que no solo borra el deportista de la tabla Deportista, sino que también lo borra del resto de tablas donde aparece este deportista, porque hemos puesto que se borre en cascada y así no tener problemas. En el apartado de evaluación veremos como el número de accesos se ve reducido por el uso de los índices.

3 Diseño Físico

El diseño Físico no lo cambiaremos apenas, ya que mantendremos el tamaño de cubo de 8KB, seguiremos organizando el fichero de datos de manera serial no-consecutiva, y tendremos cubos con un PCTFREE = 10 y PCTUSED = 60, tal y como utiliza Oracle de manera predefinida, los únicos cambios que realizaremos es la creación de nuevos índices que mejoren el rendimiento global de la carga de trabajo estándar.

Los índices que hemos creado son los siguientes:

3.1. Indice 1: in_ganador

Este índice se crea sobre la tabla Ganador, usando como clave de indización el atributo Deporte, creándose en Oracle así:

CREATE INDEX in_ganador ON Ganador (Deporte);

Este índice es creado para cuando se quiera utilizar la tabla Ganador para buscar a uno o varios ganadores de los que se conoce el Deporte al que pertenecen.

Básicamente lo creamos para la mejora de la Consulta 1 que busca el Ganador de la Copa del rey de futbol de 2013, viendo los campos de búsqueda nos damos cuenta de que el campo que menor cardinalidad tiene es el de Deporte (20 Deportes), por eso

decidimos crear el índice sobre este atributo, porque así mejorara la búsqueda al hacer un filtro por deporte.

3.2. Indice 2: in_ficha_dep

Este índice se crea sobre la tabla Ficha, usando como clave de indización Deportista, creándose en Oracle asi:

CREATE INDEX in_ficha_dep ON ficha(deportista);

El índice se crea para cuando se quiera acceder a la tabla ficha por su atributo Deportista. Esto lo creamos para la consulta 4, que realiza join por ese atributo, comparando quienes de los que tienen ficha, tienen ademas alguna evidencia.

Este índice se usa además en la vista 2, ya que en ella seleccionamos solo los deportistas de la tabla, para hacer la diferencia con los de inscrito.

3.3 Indice 3: in_ficha

Este índice lo creamos sobre la tabla Ficha, por los atributos Deportista, categoría y Deporte. Esto se escribe en Oracle de la siguiente forma:

CREATE INDEX in_ficha ON ficha(Deportista, Categoria, Deporte);

Lo creamos ya que se accede a Ficha, por esos atributos en la consulta 3, para luego hacer la join con categoría.

3.4 Indice 4: in_inscrito

En este caso, creamos el índice para la tabla Inscrito por el atributo Deportista, que escribimos en Oracle:

CREATE INDEX in_inscrito ON inscrito(deportista);

En este caso, este índice es para usarlo sobre la vista 2, ya que se accede a la tabla inscrito solo para buscar los deportistas, y después realizar una diferencia de tablas con los deportistas de Ficha.

3.5 Indice 5: in_inscrito_v2

Por ultimo, creamos este índice multiclave para la tabla Inscrito por los atributos que comparte con la tabla Ganador, estos atributos son Nom_Club, Pais, Deporte, Categoria, Nom_Competicion, Edicion, Division y esto se escribe en Oracle:

CREATE INDEX in_inscrito_v2 ON inscrito(Nom_Club, Pais, Deporte, Categoria, Nom_Competicion, Edicion, Division);

Este índice lo hemos creado para la vista 1, en la que se produce una natural join entre inscrito y ganador. Ya que la join se hace por los atributos que tienen en común hemos decidido hacer este índice para que el acceso sea menos costoso.

4 Evaluación

La medida del número de accesos del diseño físico inicial era la siguiente:

```
RESULTS AT 06/05/14
TIME CONSUMPTION: 3532 seconds.
CONSISTENT GETS: 38522 blocks

Procedimiento PL/SQL terminado correctamente.

Transcurrido: 00:00:03.60
SQL>
```

Aquí vemos que los accesos son 38522, y a partir de los datos de la parte de análisis, en la que veíamos el coste de cada una de las operaciones que realizaba la carga de trabajo estandar, nos centramos en reducir los costes de las operaciones mas pesadas para nuestro diseño en particular.

La consulta mas pesada era la de la vista 2, tras pensar el índice que pudiera reducir el numero de accesos, nos dimos cuenta de que utilizando los índices 2 y 4 del apartado anterior, reducíamos enormemente el numero de accesos.

```
|   5 |      INDEX FAST FULL SCAN| IN_FICHA_DEP |
  (2)| 00:00:01 |

|   6 |     SORT UNIQUE          |               |
  (3)| 00:00:12 |

|   7 |      INDEX FAST FULL SCAN| IN_INSCRITO   |
  (1)| 00:00:04 |

-------------------------------------------------
-----------------



Estadísticas
-------------------------------------------------
          0  recursive calls
          0  db block gets
       1338  consistent gets
```

Dejándolo a tan solo 1338, bajándolo de 14558, lo que supone una reducción a una decima parte, siendo cambio mas brusco.

Donde también hemos notado un cambio grande, es con la consulta de la vista 1, que con el uso del índice 5 reducimos los accesos de 13661 a 6703.

```
|   4 |    INDEX FAST FULL SCAN| IN_INSCRITO_V2 |
0:00:21 |

---------------------------------------------------
---------


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

   2 - access("INSCRITO"."DIVISION"="GANADOR"."DIV
              "INSCRITO"."EDICION"="GANADOR"."EDIC
              "INSCRITO"."NOM_COMPETICION"="GANADO
              "INSCRITO"."CATEGORIA"="GANADOR"."CA
              "INSCRITO"."DEPORTE"="GANADOR"."DEPO
NADOR"."PAIS"

              AND "INSCRITO"."NOM_CLUB"="GANADOR".


Estadísticas
---------------------------------------------------
        224  recursive calls
          0  db block gets
       6703  consistent gets
```

Una reducción de la mitad de accesos.

Tambien al incluir índices vemos como reduce el numero de accesos a la hora del eliminar un deportista.

```
   2 - access("CID"='AAAAAAAAAAAA')


Estadísticas
-----------------------------------
       1164  recursive calls
          0  db block gets
        435  consistent gets
```

Bajando de unos 700 a 435.

El resto de índices aportan mejoras, aunque no son tan grandes como estas tres que pensamos que son las mas significativas a la hora de mejorar el numero de accesos.

Tras la introducción de los cinco índices, los resultados obtenidos con la carga de trabajo estándar son:

```
RESULTS AT 08/05/14
TIME CONSUMPTION: 2750 seconds.
CONSISTENT GETS: 12296 blocks

Procedimiento PL/SQL terminado correctamente.

Transcurrido: 00:00:02.76
SQL>
```

Con lo que tenemos una reduccion de 38522 a 12296, una reduccion del 70%.

Además de los índices que usamos , hemos llevado a cabo mas pruebas que a pesar de parecernos razonables, no han resultado todo lo efectivas que hubiésemos deseado, y en lugar de reducirnos los accesos, nos los aumentaban.

Algunos ejemplos de esto serian:

CREATE INDEX in_federado ON Federado (Deporte);

CREATE INDEX in_evidencia ON Evidencia (Sujeto_Control);

CREATE INDEX in_ficha ON ficha(categoria);

Pero resultaba que por como estaban implementadas las consultas no resultaban rentables,o bien en terminus generals, o bien a nivel de consulta/vista

Creimos que con un cluster podriamos reducir los accesos a la tabla 1:

CREATE CLUSTER deportista_clust (Nom_club VARCHAR(100),Pais VARCHAR(45), Deporte VARCHAR(15), Categoria VARCHAR(11), Nom_competicion VARCHAR(100), Edicion VARCHAR(14), Division VARCHAR(5));

CREATE INDEX IND_DEPORTISTA ON CLUSTER deportista_clust;

Pero resulto que la empeoraba enormemente, creemos que por el funcionamiento del cluster.

5 Conclusiones Finales

Esta practica nos ha resultado un poco frustrante, ya que al no poder diferenciar entre resultados con datos en memoria intermedia y no, no sabíamos si eran reales. Además nos ha llevado mas del tiempo que se suponía (4 tardes completas, de 5-6 horas).

Evidentemente se aprende, y hemos aprendido pero, al igual que con las demás practicas de la asignatura, con un nivel de esfuerzo y tiempo bastante altos.