Alumno/a:	**Javier Diaz Leyva**	**NIA:**	**100383310**
Alumno/a:	**Alejandro Diaz García**	**NIA:**	**100383181**

1 Introducción

Con este trabajo empezamos como punto de partida desde una base de datos relacional de ORACLE. En ella nos encontramos con una serie de tablas con datos cargados que nos proporcionan para el desarrollo de la práctica (scripts NEWcreation.sql y NEWload.sql). Adicionalmente se nos proporciona otro script en el que se incluye el código para realizar una carga de trabajo, además de un test para obtener métricas en cuanto a la ejecución de esta carga de trabajo. La carga de trabajo de la que partimos se compone de 2 sentencias de insert sobre una serie de tablas, la creación de dos vistas y por último 3 consultas. A partir de analizar con el test la eficiencia del sistema inicial se nos plantea realizar una serie de optimizaciones sobre los elementos de la base de datos para mejorar el rendimiento de la carga de trabajo dada, así como de las operaciones adicionales que pueden llegar a ser realizadas sobre las tablas de la base. Todo esto con el objetivo de mejorar el rendimiento individual de cada tarea del workload así como colateralmente el rendimiento global de las operaciones sobre estos elementos de la base de datos.

2 Análisis

Empezamos analizando cómo está compuesto el diseño físico de ORACLE por defecto. La organización que siguen los ficheros es serial no consecutiva por lo que está optimizado para las inserciones de datos al no seguir un criterio de orden como podría ser en las organizaciones base secuenciales o direccionadas. Al ser no consecutiva los cubos que se definen son de 8 KB por defecto, pero pudiendo ser modificado su tamaño. A falta de permisos en esta práctica para poder utilizar otros tamaños de cubo se nos proporcionan 3 tablespaces para insertar objetos, TAB_2k, con tamaño de cubo de 2 KB, TAB_8k con tamaño de cubo de 8 KB (por defecto) y TAB_16k con tamaño de cubo de 16 KB. Partiendo de esta estructura vamos a valorar la carga de trabajo dada, la cual carece de algún tipo de optimización más allá del planteado en el diseño relacional tanto de las tablas como en las consultas y vistas de la carga de trabajo. Tenemos una carga de trabajo compuesta por dos inserciones, tres querys y dos vistas. Para empezar, vamos a analizar el coste y tiempo que gastan cada una de estas tareas y vamos a ver cuales son los procesos más frecuentes dentro de cada una de ellas.

Insert 1:
Al analizar el plan de ejecución del primer insert de la carga de trabajo nos encontramos con que el coste es muy bajo, esto es debido a la organización que utiliza Oracle, que beneficia especialmente en las operaciones de inserción. Por el contrario, puede llegar a perjudicar en las búsquedas, pero no es el caso para

comentar ahora. Podemos apreciar que el coste y las filas a las que accede son bastante bajos dado que inserta en el primer hueco que haya disponible. Por lo que para lograr una optimización en todo caso podríamos mejorar el acceso a la tabla clubs dado que realiza un acceso a la tabla Full, lo que significa que la recorre entera, esto es debido al verificar con la condición where el atributo *end_date* que no se trata de una clave identificativa. Podríamos mejorar este aspecto introduciendo un índice para el atributo *end_date* con el fin de mejorar los accesos a la tabla *clubs*.

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 4121819873

-------------------------------------------------------------------------------------------
| Id  | Operation               | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT        |               |  4424 |   397K|    16   (0)| 00:00:01 |
|   1 |  LOAD TABLE CONVENTIONAL | PROPOSALS     |       |       |            |          |
|*  2 |   HASH JOIN             |               |  4424 |   397K|    16   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL    | CLUBS         |   362 | 14842 |     3   (0)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN | PK_MEMBERSHIP |  4827 |   240K|    13   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------

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

   2 - access("NAME"="MEMBERSHIP"."CLUB")
   3 - filter("END_DATE" IS NULL)
```

Coste total: 48
Tiempo total (segundos): 4s
Filas accedidas totales: 14037

Insert 2:
Al analizar el plan de ejecución del segundo insert de la carga de trabajo nos encontramos con más o menos lo mismo que en el anterior, pero observamos que el coste total se ha elevado en comparación. Principalmente es debido a la complejidad de la inserción, pero aun así observando de manera global no es un coste muy disparatado. No se realizan Full scans ni ningún elemento que pueda apreciarse como perjudicial para el rendimiento, por lo que de primeras no vemos que pueda ser optimizado más de lo que está.

```
Plan hash value: 2031669321

-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT         |               | 77664 |    10M|    56   (2)| 00:00:01 |
|   1 |  LOAD TABLE CONVENTIONAL | COMMENTS      |       |       |            |          |
|*  2 |   HASH JOIN              |               | 77664 |    10M|    56   (2)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN  | PK_MEMBERSHIP |  4827 |   240K|    13   (0)| 00:00:01 |
|   4 |    VIEW                  |               |  6355 |   601K|    42   (0)| 00:00:01 |
|   5 |     COUNT                |               |       |       |            |          |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
|*  6 |      INDEX FAST FULL SCAN| PK_PROPOSALS  |  6355 |   521K|    42   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   2 - access("from$_subquery$_002"."CLUB"="MEMBERSHIP"."CLUB")
   6 - filter("TITLE" LIKE 'workload_%')
```

Coste total: 209
Tiempo total (segundos): 5s
Filas accedidas totales: 172865

Query 1:
En cuanto a la primera consulta podemos observar que tiene menor coste que la segunda inserción y que donde más trabajo está realizando es en las operaciones de selección. Podemos observar que realiza una operación de intersección y dos accesos Full a la tabla *genres_movies*. Si nos vamos al código de la query podemos ver que cuando se accede a la tabla *genres_movies* es para la condición where para filtrar por el atributo *genre* en ambos lados de la operación de intersección. Por lo que se podría optimizar el acceso a la tabla para evitar tener que hacer un acceso completo, por lo demás nada que destacar en cuanto a posible mejora de rendimiento. Se podría crear un índice para *genres_movies* dentro de la tabla, pero el atributo *genre* es clave ajena de la tabla *genres* por lo que puede ser que no optimice demasiado.

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 1537988869

-------------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |  1841 |   499K|    53   (6)| 00:00:01 |
|   1 |  INTERSECTION       |               |       |       |            |          |
|   2 |   SORT UNIQUE       |               |  1841 |   208K|    26   (4)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| GENRES_MOVIES |  1841 |   208K|    25   (0)| 00:00:01 |
|   4 |   SORT UNIQUE       |               |  2572 |   291K|    26   (4)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL| GENRES_MOVIES |  2572 |   291K|    25   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------

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

   3 - filter(UPPER("GENRE")='COMEDY')
   5 - filter(UPPER("GENRE")='DRAMA')
```

Coste total: 155
Tiempo total (segundos): 5s
Filas accedidas totales: 10667

Query 2:
En cuanto a esta consulta podemos apreciar que el coste es más elevado que la anterior. Podemos apreciar que es dado a que realiza dos full access a dos tablas distintas por lo que se ve perjudicado su rendimiento de forma notable. El proceso que más coste trae es el de selección, después el de ordenación (para este podrían insertarse los datos de forma ordenada en las tablas para evitar luego este coste en la consulta) seguido de un join. Pero confiamos en que mejorando los accesos a las tablas *contracts* y *profiles* puede mejorar considerablemente su desempeño y esto puede producirse gracias a implementar índices. También podría aplicarse usar un cluster para mejorar la operación de join al tener clusterizadas las tablas de *contracts* y *profiles* por *citizenID*.

```
Plan hash value: 3106799928

-------------------------------------------------------------------------------------------
| Id  | Operation               | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |               | 12039 |   707K|   148   (4)| 00:00:01 |
|   1 |  MINUS                  |               |       |       |            |          |
|   2 |   MINUS                 |               |       |       |            |          |
|   3 |    SORT UNIQUE          |               | 12039 |   329K|    34   (6)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL   | USERS         | 12039 |   329K|    32   (0)| 00:00:01 |
|   5 |    SORT UNIQUE          |               |  4827 | 91713 |    14   (8)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
|   6 |     INDEX FAST FULL SCAN| PK_MEMBERSHIP |  4827 | 91713 |    13   (0)| 00:00:01 |
|   7 |   SORT UNIQUE           |               |  8959 |   288K|   100   (2)| 00:00:01 |
|*  8 |    HASH JOIN            |               |  8959 |   288K|    99   (2)| 00:00:01 |
|   9 |     TABLE ACCESS FULL   | CONTRACTS     |  8958 | 62706 |    68   (0)| 00:00:01 |
|  10 |     TABLE ACCESS FULL   | PROFILES      | 13388 |   339K|    30   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - filter(SYSDATE@!-"REG_DATE">182)

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
   8 - access("CONTRACTS"."CITIZENID"="PROFILES"."CITIZENID")
```

Coste total: 538
Tiempo total (segundos): 9s
Filas accedidas totales: 86035

Query 3:

Con esta consulta podemos apreciar que el coste de cada una de las operaciones se dispara considerablemente comparado con las anteriores. Podemos calcular que el coste total es de 40060 por lo que el coste es desorbitado. A primera vista empezando por la última operación podemos ver que se realiza un acceso completo (full) a la tabla *comments* por lo que sería buena idea mejorarlo a un fast full scan utilizando un índice sobre la tabla *comments* por *title* y *director*. Siguiendo con el resto de las operaciones vemos que la selección consume bastante como es de esperar, junto a la operación de ordenar y de agrupar. De primeras no se nos ocurre nada mas para optimizar, en todo caso aumentar la densidad de registros por cubo en caso de ser posible, podría hacerse cambiando de tablespace alguna de las tablas a uno con un tamaño de cubo mayor.

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 2430975129

-------------------------------------------------------------------------------------------
| Id  | Operation               | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |     1 |    40 |  5727   (1)| 00:00:01 |
|*  1 |  COUNT STOPKEY          |          |       |       |            |          |
|   2 |   VIEW                  |          | 92602 |  3617K|  5727   (1)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY|          | 92602 |  3617K|  5727   (1)| 00:00:01 |
|   4 |     HASH GROUP BY       |          | 92602 |  3617K|  5727   (1)| 00:00:01 |
|   5 |      VIEW               |          | 92602 |  3617K|  5719   (1)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
|   6 |       HASH GROUP BY     |          | 92602 |  7144K|  5719   (1)| 00:00:01 |
|   7 |        TABLE ACCESS FULL| COMMENTS | 92602 |  7144K|  5714   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM=1)
   3 - filter(ROWNUM=1)
```

Coste total: 40060
Tiempo total (segundos): 7s
Filas accedidas totales: 555613

Vista 1 “cp_aragna”:
Como podemos apreciar las vistas tienen un coste bastante elevado dada la complejidad de la consulta que tienen y el hecho de crear la estructura para la vista. Podemos apreciar que se producen dos accesos completos a las tablas de proposals y comments los cuales podrían ser optimizados con el uso de índices. También podemos ver que se realiza un LEFT JOIN que podría ser optimizado mediante la implementación de un cluster. El resto de las operaciones como de orden no podría optimizarse porque se ordena en base a una operación que realiza la vista.

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
Plan hash value: 1837425452

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           | 14689 |   459K|       |  6956   (1)| 00:00:01 |
|   1 |  VIEW                 | CP_ARAGNA | 14689 |   459K|       |  6956   (1)| 00:00:01 |
|   2 |   SORT ORDER BY       |           | 14689 |  3858K|       |  6956   (1)| 00:00:01 |
|*  3 |    FILTER             |           |       |       |       |            |          |
|   4 |     HASH GROUP BY     |           | 14689 |  3858K|       |  6956   (1)| 00:00:01 |
|*  5 |      HASH JOIN OUTER  |           | 14689 |  3858K|  2040K|  6953   (1)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
|   6 |       TABLE ACCESS FULL| PROPOSALS | 14689 |  1864K|       |   478   (1)| 00:00:01 |
|   7 |       TABLE ACCESS FULL| COMMENTS  | 92602 |    12M|       |  5714   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   3 - filter(SUM(NVL2("COMMENTS"."MSG_DATE",0,1))>0)
   5 - access("MEMBER"="COMMENTS"."NICK"(+) AND "CLUB"="COMMENTS"."CLUB"(+) AND
              "TITLE"="COMMENTS"."TITLE"(+) AND "DIRECTOR"="COMMENTS"."DIRECTOR"(+))
```

Coste total: 40969
Tiempo total (segundos): 7s
Filas accedidas totales: 180736

Vista 2 “leader”:

En cuanto a la vista leader es la tarea del workload que más consume de forma global, un total de 59003 por lo que es conveniente optimizarla para reducir el coste. Podemos observar que tenemos dos accesos completos a las tablas de comments y proposals, por lo que sería interesante indizar esos accesos para ganar velocidad y reducirlos. Como en la anterior se puede hacer uso de un cluster para las tablas que están involucradas en la operación de LEFT JOIN. En conclusión, si se lograse mejorar el coste de esta vista la carga de trabajo se vería positivamente beneficiada en lo que a rendimiento se refiere.

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------
Plan hash value: 2949163629

-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |    10 |   640 |       |  6896   (1)| 00:00:01 |
|   1 |  VIEW                    | LEADER    |    10 |   640 |       |  6896   (1)| 00:00:01 |
|*  2 |   COUNT STOPKEY          |           |       |       |       |            |          |
|   3 |    VIEW                  |           |   108 |  6912 |       |  6896   (1)| 00:00:01 |
|*  4 |     SORT ORDER BY STOPKEY|           |   108 | 27432 |       |  6896   (1)| 00:00:01 |
|   5 |      HASH GROUP BY       |           |   108 | 27432 |       |  6896   (1)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------
|*  6 |       HASH JOIN OUTER     |           | 14689 |  3643K|  2040K|  6893   (1)| 00:00:01 |
|   7 |        TABLE ACCESS FULL  | PROPOSALS | 14689 |  1864K|       |   478   (1)| 00:00:01 |
|   8 |        VIEW               |           | 92602 |    10M|       |  5719   (1)| 00:00:01 |
|   9 |         HASH GROUP BY     |           | 92602 |     9M|       |  5719   (1)| 00:00:01 |
|  10 |          TABLE ACCESS FULL| COMMENTS  | 92602 |     9M|       |  5714   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   2 - filter(ROWNUM<=10)

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------
   4 - filter(ROWNUM<=10)
   6 - access("PROPOSALS"."CLUB"="A"."CLUB"(+) AND "PROPOSALS"."TITLE"="A"."TITLE"(+)
              AND "PROPOSALS"."DIRECTOR"="A"."DIRECTOR"(+))
```

Coste total: 59003
Tiempo total (segundos): 10s
Filas accedidas totales: 307528

Tabla ordenando de menor a mayor por coste las tareas de la carga de trabajo:

	Coste total	Tiempo total (s)	Filas accedidas
Insert 1	48	4s	14037
Query 1	155	5s	10667
Insert 2	209	5s	172865
Query 2	538	9s	86035
Query 3	40060	7s	555613
Vista 1	40969	7s	180736
Vista 2	59003	10s	307528

Como podemos analizar en la tabla las dos vistas junto a la query 3 son las tareas que más coste tienen y por lo que van a ser nuestro foco para optimizar, pero sin dejar de lado la query 2 y el resto de las tareas en caso de que se pudiera mejorar algo. Como se puede ver nos vamos a fijar principalmente en el coste para evaluar las

optimizaciones dados los problemas que estamos teniendo para poder valorar los tiempos en la base de datos.

En cuanto el coste total inicial al ejecutar **PKG_COSTES.RUN_TEST** el cual ejecuta ciertas iteraciones la carga de trabajo dada dentro del procedimiento **PR_WORKLOAD**. Para esta ejecución inicial del runtest activamos el *serveroutput* para obtener las métricas de los bloques consumidos y el tiempo de penalización. Por otro lado activamos también *timing* para ver el tiempo que demora la ejecución del test y el *autotrace* para seguir las estadísticas de la ejecución.

```
Salida de Script                                                    1/1

Seguimiento Automático Activado
Muestra el plan de ejecución, así como las estadísticas de la sen
RESULTS AT 20/05/20
TIME CONSUMPTION: 83828,6 milliseconds.
CONSISTENT GETS: 91871,1 blocks

Procedimiento PL/SQL terminado correctamente.
Transcurrido: 00:15:47.734
```

Obtenemos que la ejecución inicial consume unos 91871,1 bloques que será la métrica que tendremos en cuenta a nivel global de la ejecución del test para percibir una mejora en cuanto a coste se refiere. Por otra parte, el tiempo de penalización no es una métrica que sea de mucha relevancia por lo que no se tendrá demasiado en cuenta. Respecto al tiempo transcurrido ofrecido por el timing es de 15 minutos 47 segundos aproximadamente, pero esto puede variar en función de la cantidad de gente que esté utilizando el servidor de la base de datos de la universidad dentro de aula virtual.

3 Diseño Físico

Como principal objetivo de esta práctica está el mejorar el rendimiento de las operaciones que se realizan en el workload por lo que hemos realizado una serie de implementaciones para poder alcanzar el objetivo. Entre estas implementaciones está la creación de índices para mejorar el acceso de recuperación de datos, entre otras. Los índices han sido la fuente donde hemos logrado obtener mejor resultado para realizar optimizaciones frente a otras como sería la implementación de clusters, mover objetos de tablespaces, así como modificar las características de los tablespaces.
Hemos decidido crear sobre cada una de las tareas del workload realizar una serie de optimizaciones para mejorar su desempeño a nivel global, procurando que estas optimizaciones empeoren el la menor manera al rendimiento de otras operaciones.

Insert 1

No hemos aplicado ninguna optimización en la inserción 1.

Insert 2
No hemos aplicado ninguna optimización en la inserción 2.

Query 1
En cuanto a esta query no hemos realizado ninguna optimización. Pero como hemos mencionado en el apartado del análisis podría haberse creado un índice sobre la tabla genres_movies con clave de indización genre, pero no mejora en nada. Esto es debido a que es clave ajena y en la tabla de origen es clave primaria y solo existe ese atributo (tabla genres).

Query 2
Revisando la tabla de costes que hicimos anteriormente, podemos comprobar que la query 2 se puede optimizar usando un índice que tome como clave la propiedad *citizenID* debido a que es un atributo único y no nulo de la tabla de perfiles. Este atributo es usado por la query y hace un table access full de él, por lo que claramente se puede hacer un índice suyo para optimizar la query.

```
CREATE INDEX ind_citi1 ON contracts(citizenID);
```

También hemos considerado el crear un cluster sobre las tablas contracts y profiles dado que se realiza una operación de JOIN y tener físicamente estas dos tablas unidas mediante la clave citizenID consideramos que podría dar como fruto una mejora en los accesos a la hora de realizar cada operación dentro del JOIN.

```
DROP CLUSTER clusq2;
CREATE CLUSTER clusq2 (citizenID VARCHAR2(10));
CREATE INDEX index_clusq2 ON CLUSTER clusq2;
```

Después en cada una de las tablas las añadimos dentro del cluster:
En la tabla *profiles*:

```
CREATE TABLE profiles (
nick          VARCHAR2(35),
name          VARCHAR2(35),
surname       VARCHAR2(35),
secsurname    VARCHAR2(35),
citizenID     VARCHAR2(10) NOT NULL,
mobile        NUMBER(12),
birthdate     DATE,
CONSTRAINT PK_PROFILES PRIMARY KEY (nick),
CONSTRAINT UK_PROFILES UNIQUE (citizenID),
CONSTRAINT FK_PROFILES_USERS FOREIGN KEY (nick)
REFERENCES users ON DELETE CASCADE
)CLUSTER clusq2(citizenID);
```

En la tabla *contracts*:

```
CREATE TABLE contracts (
idcontract    VARCHAR2(20),
citizenID     VARCHAR2(10) NOT NULL,
contract_type VARCHAR2(50) NOT NULL,
startdate     DATE NOT NULL,
enddate       DATE,
address       VARCHAR2(100) NOT NULL,
town          VARCHAR2(100) NOT NULL,
ZIPcode       VARCHAR2(8) NOT NULL,
country       VARCHAR2(100) NOT NULL,
CONSTRAINT PK_CONTRACTS PRIMARY KEY (idcontract),
CONSTRAINT FK_CONTRACTS_PRODUCTS FOREIGN KEY (contract_type) REFERENCES products,
CONSTRAINT FK_CONTRACTS_PROFILES FOREIGN KEY (citizenID) REFERENCES profiles(citizenID),
CONSTRAINT CK_CONTRACTS CHECK (enddate IS NULL OR enddate>startdate)
)CLUSTER clusq2(citizenID);
```

Query 3

Ahora tenemos en cuenta la query más pesada en cuanto a costes que tenemos. En esta query tenemos un *table access full* de la tabla *comments* por lo que hemos decidido hacer un índice en el que los atributos *title* y *director* sean la clave de indización, debido a que forman una clave única, para que de esta manera mejore el rendimiento.

```
CREATE INDEX ind_coments ON comments(title, director);
```

Vista 1

A continuación nos ponemos a optimizar las vistas que son los elementos que más recursos consumen, para ello nos fijamos en primer lugar en la vista 1, en la que podemos observar que hace un table access full de *proposals* y de *comments*, por lo que podemos hacer un índice de las variables a las que accede de *comments*, que son *nick*, *club*, *title* y *director*, haciendo esto podemos comprobar que si hay una mejora clara de rendimiento.

```
CREATE INDEX ind_view1 ON comments(nick, club, title, director);
```

Vista 2

En esta ocasión podemos comprobar que hay varias similitudes con la vista anterior, ya que también hace un table access full también de *proposals* y de *comments*, esto quiere decir que, como en la ocasión anterior, podemos hacer un índice de las variables de *comments* a las que accede la vista que en esta ocasión son *club*, *title* y *director*. Una vez hecho esto podemos volver a comprobar que existe una clara mejoría respecto al estado inicial.

```
CREATE INDEX ind_view2 ON comments(club, title, director);
```

Para ambas vistas hemos intentado crear clusters para mejorar las operaciones de JOIN pero tras varias pruebas no hemos considerado que sean buenas optimizaciones dado que perjudicaban a otras operaciones y empeoraba el rendimiento global, algunos de los clusters que hemos probado sobre las tablas *proposals* y *comments* han sido con *nick*/*member*, *club* entre otras. Por lo que decidimos no incluirlo en la implementación final. También realizamos pruebas creando las tablas en otro de los tablespaces proporcionados pero el rendimiento no mejoró como nos lo esperábamos. Por lo que concluimos con nuestro diseño físico propuesto con las implementaciones especificadas en este apartado.

4 Evaluación

Insert 1

Tal y como hemos comentado no se ha realizado ninguna optimización en la primera inserción por lo que no mejora en rendimiento.

```
Plan hash value: 4121819873

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT         |               |  4940 |   443K|    16   (0)| 00:00:01 |
|   1 |  LOAD TABLE CONVENTIONAL | PROPOSALS     |       |       |            |          |
|*  2 |   HASH JOIN              |               |  4940 |   443K|    16   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL     | CLUBS         |   362 | 14842 |     3   (0)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN  | PK_MEMBERSHIP |  5431 |   270K|    13   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------

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

   2 - access("NAME"="MEMBERSHIP"."CLUB")
   3 - filter("END_DATE" IS NULL)
```

Coste inicial: 48
Tiempo total (segundos): 4s
Filas accedidas totales: 14037

Coste tras la optimización: 48
Tiempo total (segundos): 4s
Filas accedidas totales: 15673

Insert 2

De la misma manera que en la inserción 1 no hemos considerado que sean tareas de un coste significativo para optimizar por lo que no hemos realizado ninguna optimización en la inserción 2. Hemos decidido ir a los puntos débiles del diseño físico inicial.

```
Plan hash value: 2031669321

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT         |               | 65035 |  9399K|    56   (2)| 00:00:01 |
|   1 |  LOAD TABLE CONVENTIONAL | COMMENTS      |       |       |            |          |
|*  2 |   HASH JOIN              |               | 65035 |  9399K|    56   (2)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN  | PK_MEMBERSHIP |  5431 |   270K|    13   (0)| 00:00:01 |
|   4 |    VIEW                  |               |  4766 |   451K|    42   (0)| 00:00:01 |
|   5 |     COUNT                |               |       |       |            |          |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
|*  6 |      INDEX FAST FULL SCAN| PK_PROPOSALS  |  4766 |   390K|    42   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   2 - access("from$_subquery$_002"."CLUB"="MEMBERSHIP"."CLUB")
   6 - filter("TITLE" LIKE 'workload_%')
```

Coste inicial: 209 **Coste tras la optimización: 209**
Tiempo total (segundos): 5s **Tiempo total (segundos): 5s**
Filas accedidas totales: 172865 **Filas accedidas totales: 140267**

Query 1

Como hemos planteado hemos creado un índice pero no afectaba en absoluto a la mejora del rendimiento, dado su bajo coste inicial no nos es de mucha importancia el optimizar esta tarea.

```
Plan hash value: 1537988869

-------------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |  1940 |   515K|    53   (6)| 00:00:01 |
|   1 |  INTERSECTION       |               |       |       |            |          |
|   2 |   SORT UNIQUE       |               |  1940 |   219K|    26   (4)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| GENRES_MOVIES |  1940 |   219K|    25   (0)| 00:00:01 |
|   4 |   SORT UNIQUE       |               |  2611 |   295K|    26   (4)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL| GENRES_MOVIES |  2611 |   295K|    25   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------

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

   3 - filter(UPPER("GENRE")='COMEDY')
   5 - filter(UPPER("GENRE")='DRAMA')
```

Coste inicial: 155 **Coste tras la optimización: 155**

Tiempo total (segundos): 5s **Tiempo total (segundos): 5s**
Filas accedidas totales: 10667 **Filas accedidas totales: 11042**

Query 2

Con esta consulta hemos creado un índice que lograba mejorar considerablemente el rendimiento de la consulta, pero por diseño hemos decidido incluir un cluster con el objetivo de mejorar las operaciones de los JOIN, pero este intento se ha visto afectado negativamente incrementando el coste total y ocultando la mejora realizada por el índice. Dada la situación con el servidor no hemos podido realizar otra serie de mejoras y pruebas sobre esta consulta. A pesar de no mejorar no afecta negativamente al resto de tareas ni afectan colateralmente a su rendimiento, por lo que en ese aspecto estamos medianamente satisfechos, aunque nos hubiera gustado probar otra serie de mejoras como crear el cluster en el tablespace 16K y reducir el PCTFREE y aumentar el PCTUSED.

```
Plan hash value: 52120829

-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  | 11084 |   579K|   145   (4)| 00:00:01 |
|   1 |  MINUS                   |                  |       |       |            |          |
|   2 |   MINUS                  |                  |       |       |            |          |
|   3 |    SORT UNIQUE           |                  | 11084 |   303K|    34   (6)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL    | USERS            | 11084 |   303K|    32   (0)| 00:00:01 |
|   5 |    SORT UNIQUE           |                  |  5431 |   100K|    14   (8)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
|   6 |     INDEX FAST FULL SCAN | PK_MEMBERSHIP    |  5431 |   100K|    13   (0)| 00:00:01 |
|   7 |   SORT UNIQUE            |                  |  5450 |   175K|    97   (3)| 00:00:01 |
|   8 |    NESTED LOOPS SEMI     |                  |  5450 |   175K|    96   (2)| 00:00:01 |
|   9 |     VIEW                 | index$_join$_004 | 10317 |   261K|    95   (0)| 00:00:01 |
|* 10 |      HASH JOIN           |                  |       |       |            |          |
|  11 |       INDEX FAST FULL SCAN| PK_PROFILES      | 10317 |   261K|    55   (0)| 00:00:01 |
|  12 |       INDEX FAST FULL SCAN| UK_PROFILES      | 10317 |   261K|    64   (0)| 00:00:01 |
|* 13 |     INDEX RANGE SCAN     | IND_CITI1        |  2879 | 20153 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
---------------------------------------------------

   4 - filter(SYSDATE@!-"REG_DATE">182)
  10 - access(ROWID=ROWID)
  13 - access("CONTRACTS"."CITIZENID"="PROFILES"."CITIZENID")
```

Coste inicial: 538 **Coste tras la optimización: 645**
Tiempo total (segundos): 9s **Tiempo total (segundos): 11s**
Filas accedidas totales: 86035 **Filas accedidas totales: 88844**

Query 3

Llegando a la consulta más costosa de todas nos encontramos con una gran mejora de rendimiento gracias al índice implementado. Pasamos de tener un coste de 40060 a uno de 3234, debido a la disminución de accesos a la tabla a la hora de realizar consultas de selección. Podemos apreciar que ya no se realiza ningún acceso

completo a ninguna de las tablas, en cambio se realiza gracias a IND_COMENTS un acceso rápido por la clave de indización.

```
Plan hash value: 1750156749

-----------------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |             |     1 |    40 |   471   (6)| 00:00:01 |
|*  1 |  COUNT STOPKEY           |             |       |       |            |          |
|   2 |   VIEW                   |             |   214K|  8379K|   471   (6)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY |             |   214K|  8379K|   471   (6)| 00:00:01 |
|   4 |     HASH GROUP BY        |             |   214K|  8379K|   471   (6)| 00:00:01 |
|   5 |      VIEW                |             |   214K|  8379K|   453   (3)| 00:00:01 |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
|   6 |       HASH GROUP BY        |             |   214K|    16M|   453   (3)| 00:00:01 |
|   7 |        INDEX FAST FULL SCAN| IND_COMENTS |   214K|    16M|   444   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM=1)
   3 - filter(ROWNUM=1)
```

Coste inicial: 40060 **Coste tras la optimización: 3234**
Tiempo total (segundos): 7s **Tiempo total (segundos): 7s**
Filas accedidas totales: 555613 **Filas accedidas totales: 1284001**

View 1 “CP_argana”

En la primera de las vistas tenemos inicialmente un coste de 40969 y tras aplicar el índice obtenemos una mejora a un coste de 2880. Es un decremento bastante considerable, haciendo de la segunda tarea más costosa más eficiente.

```
Plan hash value: 3806976041

-----------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |           | 16640 |   520K|   481   (1)| 00:00:01 |
|   1 |  VIEW                                    | CP_ARAGNA | 16640 |   520K|   481   (1)| 00:00:01 |
|   2 |   SORT ORDER BY                          |           | 16640 |  4371K|   481   (1)| 00:00:01 |
|*  3 |    FILTER                                |           |       |       |            |          |
|   4 |     HASH GROUP BY                        |           | 16640 |  4371K|   481   (1)| 00:00:01 |
|   5 |      NESTED LOOPS OUTER                  |           | 16640 |  4371K|   478   (1)| 00:00:01 |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
|   6 |       TABLE ACCESS FULL                  | PROPOSALS | 16640 |  2112K|   478   (1)| 00:00:01 |
|   7 |       TABLE ACCESS BY INDEX ROWID BATCHED| COMMENTS  |     1 |   139 |     0   (0)| 00:00:01 |
|*  8 |        INDEX RANGE SCAN                  | IND_VIEW1 |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   3 - filter(SUM(NVL2("COMMENTS"."MSG_DATE",0,1))>0)
   8 - access("MEMBER"="COMMENTS"."NICK"(+) AND "CLUB"="COMMENTS"."CLUB"(+) AND
              "TITLE"="COMMENTS"."TITLE"(+) AND "DIRECTOR"="COMMENTS"."DIRECTOR"(+))
```

Coste inicial: 40969 **Coste tras la optimización: 2880**
Tiempo total (segundos): 7s **Tiempo total (segundos): 8s**
Filas accedidas totales: 180736 **Filas accedidas totales: 99842**

View 2 "leader"

En la vista 2 es donde se ha producido el mayor decremento en cuanto a coste se refiere, este era uno de nuestros objetivos marcados al inicio de la práctica, el reducir el consumo al mínimo en las tareas que más recursos consumen. Pasamos de tener un coste total de 59003 a un coste de 18238, el cual sigue siendo elevado y confiamos en que podríamos haber realizado una mejor optimización sobre esta tarea, pero aun así la mejora afecta positivamente al cómputo global.

```
Plan hash value: 2030474191

-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |    10 |   640 |       |  2633   (1)| 00:00:01 |
|   1 |  VIEW                      | LEADER    |    10 |   640 |       |  2633   (1)| 00:00:01 |
|*  2 |   COUNT STOPKEY            |           |       |       |       |            |          |
|   3 |    VIEW                    |           |   218 | 13952 |       |  2633   (1)| 00:00:01 |
|*  4 |     SORT ORDER BY STOPKEY  |           |   218 | 55372 |       |  2633   (1)| 00:00:01 |
|   5 |      HASH GROUP BY         |           |   218 | 55372 |       |  2633   (1)| 00:00:01 |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
|*  6 |       HASH JOIN OUTER      |           | 16640 |  4127K|  2312K|  2630   (1)| 00:00:01 |
|   7 |        TABLE ACCESS FULL   | PROPOSALS | 16640 |  2112K|       |   478   (1)| 00:00:01 |
|   8 |        VIEW                |           |   214K|   25M|       |   658   (2)| 00:00:01 |
|   9 |         HASH GROUP BY      |           |   214K|   22M|       |   658   (2)| 00:00:01 |
|  10 |          INDEX FAST FULL SCAN| IND_VIEW2 |   214K|   22M|       |   649   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   2 - filter(ROWNUM<=10)
```

Coste inicial: 59003 **Coste tras la optimización: 18238**
Tiempo total (segundos): 10s **Tiempo total (segundos): 10s**
Filas accedidas totales: 307528 **Filas accedidas totales: 675954**

Aquí podemos ver la comparación de mejora en rendimiento de cada una de las tareas tras aplicar las optimizaciones descritas (ordenadas de menor coste a mayor en la antigua):

Antigua	Coste total	Tiempo total (s)	Filas accedidas
Insert 1	48	4s	14037
Query 1	155	5s	10667
Insert 2	209	5s	172865
Query 2	538	9s	86035

Nueva	Coste total	Tiempo total (s)	Filas accedidas	% coste mejorado
Insert 1	48	4s	15673	0%
Query 1	155	5s	11042	0%
Insert 2	209	5s	140267	0%
Query 2	645	11s	88844	-11%

Query 3	40060	7s	555613
Vista 1	40969	7s	180736
Vista 2	59003	10s	307528

Query 3	3234	7s	1284001	92%
Vista 1	2880	8s	99842	93%
Vista 2	18238	10s	675954	70%

PKG_COSTES.RUN_TEST INICIAL:

```
Salida de Script                                                    1/1

Seguimiento Automático Activado
Muestra el plan de ejecución, así como las estadísticas de la sen
RESULTS AT 20/05/20
TIME CONSUMPTION: 83828,6 milliseconds.
CONSISTENT GETS: 91871,1 blocks

Procedimiento PL/SQL terminado correctamente.
Transcurrido: 00:15:47.734
```

PKG_COSTES.RUN_TEST TRAS REALIZAR LAS OPTIMIZACIONES:

```
Seguimiento Automático Activado
Muestra el plan de ejecución, así como las estadísticas de la sentencia.
RESULTS AT 20/05/20
TIME CONSUMPTION: 89265,8 milliseconds.
CONSISTENT GETS: 74297 blocks

Procedimiento PL/SQL terminado correctamente.
Transcurrido: 00:17:57.869
```

Como se puede apreciar la mejora de rendimiento viene reflejada en los bloques consumidos pero en cuanto al tiempo transcurrido empeora considerablemente, creemos que esto es posible a la cantidad de gente que ha podido acceder en los momentos en los que hemos estado realizando las últimas pruebas de rendimiento. Globalmente hemos mejorado cada tarea en porcentajes considerables, quedando satisfechos con la mejora de rendimiento otorgada al diseño físico.

5 Conclusiones Finales

En definitiva, pensamos que hemos realizado un buen trabajo, ya que hemos conseguido optimizar una gran parte del coste de las querys y de las vistas, para ello, nos hemos apoyado en clusters e índices, analizando previamente que era mejor para cada circunstancia y haciendo muchas pruebas de clusters e índices que al final no han sido incluidos debido a que no mejoraba el coste final. Nos hubiera gustado poder tener más tiempo para dedicar a esta práctica en exclusiva y así aportar mejores soluciones en el diseño. También nos hubiera gustado profundizar más en este tema del "tuning" de bases de datos y mejora del rendimiento en cuestión por lo que consideramos que ha sido una de las prácticas más interesantes de la asignatura.

La realización de esta práctica se ha visto condicionada por varios factores, ya que, en primer lugar hemos tenido que lidiar con los problemas derivados del aula virtual, debido a que dicha plataforma dejaba de funcionar o iba lenta en las ejecuciones de los scripts por la gran carga de trabajo que tenía la plataforma causada por la cantidad de usuarios ejecutando scripts a la vez. La verdad es que nos hubiese venido mejor haber realizado las pruebas en el aula de informática de manera presencial como en la primera práctica. Por otro lado hemos tenido que lidiar con que es la época final del cuatrimestre, lo que quiere decir que las prácticas y los últimos parciales inundan nuestros calendarios y más teniendo en cuenta la situación que vivimos, porque la evaluación continua ha cogido una gran importancia en la mayoría de asignaturas, esto nos ha perjudicado en términos de horarios sobre todos, ya que teníamos que cuadrar nuestro día para poder ir avanzando en la práctica. Respecto al esfuerzo requerido por parte de la práctica, hemos tenido prácticas que necesitaban una mayor cantidad de esfuerzo, pero no de tiempo, debido a que en esta ocasión era más el estar esperando a que el aula virtual ejecutara, que a debido al esfuerzo que conllevaba la práctica. Nuestro desempeño en todas las prácticas ha sido bueno ya que hemos sabido planificarnos y sobre todo complementarnos muy bien para la realización de todas las prácticas a pesar de la curva de aprendizaje que supone la asignatura.

La asignatura en general nos parece que está bien planificada y tratada pero hay ciertas proposiciones que nos gustaría hacer como que se eliminaran los exámenes de las prácticas, en nuestro caso, no hemos tenido que realizar dichos exámenes pero nos parece que con la gran carga de trabajo que conllevan las prácticas, evaluarlas con un examen que te pueda bajar la nota y en ningún caso mejorar es un poco arriesgado, ya que, por ejemplo, sacando un 8 en la práctica y un 5 en el examen tendrías un 4 en la nota global de la práctica, cosa que habiendo aprobado las dos pruebas no hace justicia al trabajo realizado. Por otro lado, pasando a una propuesta más positiva nos gustaría que se incluyera más contenido acerca del big data, como unas nociones básica acerca de algoritmos con los que se trata o cosas por el estilo. Quitando estas dos cosas, es una asignatura de la que hemos sacado una gran cantidad de conocimientos y nos parece que se ha sabido llevar muy bien incluyendo con la cantidad de problemas que nos ha traído esta pandemia global.