Profesor:	**Daniel Esteban Villamil Sierra**	**Grupo**	**85**
Alumno/a:	**Álvaro González Fúnez**	**NIA:**	**100451281**
Alumno/a:	**Jhonatan Barcos Gambaro**	**NIA:**	**100548615**

1 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 tercera práctica, tenemos como objetivo el análisis y la optimización del diseño físico de la bbdd inicial proporcionada en la segunda práctica. Así mismo se nos ha proporcionado el paquete PKG_COSTES, el cual simula el comportamiento típico de acceso a los datos en un sistema bibliotecario. Nuestro objetivo es el de identificar los posibles problemas de rendimiento en el workload proporcionado, analizarlos y proponer una nueva implementación en términos de diseño para optimizar el rendimiento del mismo. Para lograrlo, hemos de seguir los siguientes pasos:

1. Restablecer la BBDD inicial eliminando las estructuras anteriores y ejecutando de nuevo los scripts iniciales proporcionados con una pequeña modificación en NEW_load.

2. Análisis de la situación inicial a través del procedimiento run_test(10) del paquete PKG_COSTES.

3. Análisis individual de cada consulta del worklad utilizando herramientas como AUTOTRACE.

4. Propuesta y justificación de un nuevo diseño físico utilizando clúster/s e índices.

5. Implementación de las propuestas de diseño en SQL.

6. Evaluación comparativa de los resultados en términos de rendimiento respecto a la situación inicial y avanzada.

7. Conclusiones finales sobre la propuesta.

2 Análisis

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

Como preparación previa a nuestra análisis hemos de explicitar que hemos dropeado todas las tablas, vistas y triggers definidos en la práctica 2 con el fin de rehacer de nuevo la base de datos con los scripts proporcionados previamente *NEW_creation* y *NEW_load* (modificado

añadiendo antes del último commit una condición de actualización de valores de copias para aportar aleatoriedad a nuestra bbdd con el objetivo de simular condiciones reales). Finalmente, ejecutamos el script proporcionado para la creación del paquete *PKG_COSTES*.

Para comprender el diseño físico actual ejecutaremos el siguiente code para visualizar las estadísticas almacenadas en la vista *USER_TABLES*, observando el número de filas, tamaño medio de cada fila y número de bloques ocupados por cada tabla. Se adjuntan capturas de pantalla a continuación.

```
Unset
select table_name, avg_row_len, num_rows, blocks from user_tables;
```

TABLE_NAME	AVG_ROW_LEN	NUM_ROWS	BLOCKS
ASSIGN_BUS	23	150	5
ASSIGN_DRV	32	150	5
BIBUSES	25	14	5
BOOKS	117	181435	3142
COPIES	24	241572	1000
DRIVERS	128	13	5
EDITIONS	219	240632	7552
LOANS	68	23709	244
MORE_AUTHORS	156	23333	622
MUNICIPALITIES	35	1365	13
POSTS	878	5447	748

TABLE_NAME	AVG_ROW_LEN	NUM_ROWS	BLOCKS
ROUTES	6	150	5
SERVICES	66	1365	20
STOPS	69	1365	20
USERS	249	2771	103

15 rows selected.

Mide el rendimiento global de la carga de trabajo y coméntalo (añade captura de pantalla).
Con el objetivo de realizar un análisis inicial sobre la carga de trabajo proporcionada por el paquete *PKG_COSTES* hemos utilizado el procedimiento *run_test(10)* para ejecutar 10 veces una secuencia de consultas representativas del uso típico de nuestra bbdd cuyas frecuencias relativas aportadas por el enunciado son:

```
Unset
select * from editions where pub_place='…';
select * from editions where publisher='…';
select * from copies where condition='…';
select * from editions;
```

Las consultas muestran estas frecuencias, respectivamente: {0.3, 0.3, 0.3, 0.1}.

A continuación se adjunta una captura de pantalla del rendimiento al correr el paquete PKG_COSTES:

```
SQL> EXEC pkg_costes.run_test(10);
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 30/04/2025 15:31:45
TIME CONSUMPTION (run): 1089.2 milliseconds.
CONSISTENT GETS (workload):60502 acc
CONSISTENT GETS (weighted average):6050.2 acc

PL/SQL procedure successfully completed.

Elapsed: 00:00:12.51
```

Analiza cada instrucción de la carga de trabajo (todas las consultas, inserciones y actualizaciones).

Para llevar a cabo dicho análisis individual de la carga de trabajo, para cada consulta/inserción capturaremos el AUTOTRACE y veremos si realiza o no un FULL TABLE SCAN, así como ver cuantos consistents gets obtenemos y cuánto tiempo tarda. Se adjuntan a continuación las capturas de pantalla de dicho análisis individual:

```
SQL> SELECT * FROM editions WHERE pub_place = 'Madrid';

82450 rows selected.

Elapsed: 00:00:05.16

Statistics
----------------------------------------------------------
         27  recursive calls
          0  db block gets
      12946  consistent gets
       7572  physical reads
          0  redo size
   17921947  bytes sent via SQL*Net to client
      60831  bytes received via SQL*Net from client
       5498  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      82450  rows processed
```

```
SQL> SELECT * FROM editions WHERE pub_place = 'Segovia';

71 rows selected.

Elapsed: 00:00:00.23

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       7567  consistent gets
       7552  physical reads
          0  redo size
      16885  bytes sent via SQL*Net to client
        424  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
         71  rows processed
```

```
SQL> SELECT * FROM editions WHERE pub_place = 'Barataria';

no rows selected

Elapsed: 00:00:00.06

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       7562  consistent gets
          0  physical reads
          0  redo size
       1467  bytes sent via SQL*Net to client
        371  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

```
SQL> SELECT * FROM editions WHERE publisher = 'B';

12 rows selected.

Elapsed: 00:00:00.05

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       7563  consistent gets
          0  physical reads
          0  redo size
       3265  bytes sent via SQL*Net to client
        374  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
         12  rows processed
```

```
SQL> SELECT * FROM editions WHERE publisher = 'SM';

1704 rows selected.

Elapsed: 00:00:00.11

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       7673  consistent gets
          0  physical reads
          0  redo size
     322955  bytes sent via SQL*Net to client
       1618  bytes received via SQL*Net from client
        115  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
       1704  rows processed
```

```
SQL> SELECT * FROM editions WHERE publisher = 'C';

no rows selected

Elapsed: 00:00:00.05

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       7562  consistent gets
          0  physical reads
          0  redo size
       1467  bytes sent via SQL*Net to client
        363  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

```
SQL> SELECT * FROM copies WHERE condition = 'D';

48168 rows selected.

Elapsed: 00:00:01.79

Statistics
----------------------------------------------------------
          8  recursive calls
          0  db block gets
       4196  consistent gets
        498  physical reads
          0  redo size
    1881510  bytes sent via SQL*Net to client
      35693  bytes received via SQL*Net from client
       3213  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      48168  rows processed
```

```
SQL> SELECT * FROM copies WHERE condition = 'G';

47724 rows selected.

Elapsed: 00:00:01.52

Statistics
----------------------------------------------------------
         15  recursive calls
          0  db block gets
       4186  consistent gets
        997  physical reads
          0  redo size
    1864087  bytes sent via SQL*Net to client
      35363  bytes received via SQL*Net from client
       3183  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      47724  rows processed
```

```
SQL> SELECT * FROM editions;

240632 rows selected.

Elapsed: 00:00:11.73

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
      23094  consistent gets
       7552  physical reads
          0  redo size
   57147402  bytes sent via SQL*Net to client
     176814  bytes received via SQL*Net from client
      16044  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
     240632  rows processed
```

```
SQL> SELECT * FROM copies WHERE condition = 'N';

48479 rows selected.

Elapsed: 00:00:01.33

Statistics
----------------------------------------------------------
          8  recursive calls
          0  db block gets
       4216  consistent gets
        989  physical reads
          0  redo size
    1893007  bytes sent via SQL*Net to client
      35913  bytes received via SQL*Net from client
       3233  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      48479  rows processed
```

A continuación se adjunta una tabla a modo de resumen del análisis inicial de cada consulta:

#	Nº	Consulta	Plan	#	Consistent Gets	#	Physical Reads	Elapsed Time	#	Rows Returned	Observaciones
	1	editions WHERE pub_place = 'Madrid'	FULL TABLE SCAN		12946		7572	5.16 s		82450	Valor muy frecuente → alto volumen de filas → alto coste
	2	editions WHERE pub_place = 'Segovia'	FULL TABLE SCAN		7567		7552	0.23 s		71	Valor poco frecuente → innecesario escaneo total
	3	editions WHERE pub_place = 'Barataria'	FULL TABLE SCAN		7562		0	0.06 s		0	Valor inexistente → escaneo completo inútil
	4	editions WHERE publisher = 'B'	FULL TABLE SCAN		7563		0	0.05 s		12	Valor raro, innecesario escaneo completo
	5	editions WHERE publisher = 'SM'	FULL TABLE SCAN		7673		0	0.11 s		1704	Alto nº de filas, sin índice
	6	editions WHERE publisher = 'C'	FULL TABLE SCAN		7562		0	0.05 s		0	Valor inexistente, escaneo total
	7	copies WHERE condition = 'G'	FULL TABLE SCAN		4186		997	1.52 s		47724	Alta distribución → lectura innecesaria de muchos bloques
	8	copies WHERE condition = 'N'	FULL TABLE SCAN		4216		989	1.33 s		48479	Idem
	9	copies WHERE condition = 'D'	FULL TABLE SCAN		4196		498	1.79 s		48168	Idem
	10	editions (sin filtro)	FULL TABLE SCAN		23094		7552	11.73 s		240632	Escaneo completo esperado, solo representa el 10% del workload

Señala las debilidades y fortalezas de ese diseño inicial atendiendo a las necesidades (carga de trabajo).

Por una parte, las principales debilidades detectadas en este diseño inicial atendiendo a la carga de trabajo son la ausencia de índices sobre las columnas críticas utilizadas como *editions(pub_place), editions(publisher)* y *copies(condition)*. Así mismo, hemos detectado que todas las consultas relevantes se resuelven mediante FULL TABLE SCANS, dicho hecho conlleva valores altos de consistents gets y a veces también de physical reads, un tiempo de respuesta alto para valores frecuentes como "Madrid" o "G" y escaneos innecesarios cuando ni siquiera hay resultados como "Barataria" o "C". Por último, respecto a la tabla Editions, la cual cuenta con más de 240k registros y 7k bloques, haciendo que cualquier consulta relacionada con dicha tabla tenga un coste muy alto.

Por otra parte, las principales fortalezas detectadas en este diseño inicial atendiendo a la carga de trabajo son la simplicidad y limpieza del diseño, el cual no cuenta con estructuras complejas que dificultan la optimización, una carga de datos realista y aleatoria gracias a la actualización de NEW_load.

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).

1. Índices recomendados: El utilizar índices para las columnas clave de las consultas del workload aporta un acceso directo a dichas filas pudiendo así reducir el coste de ejecución al no tener que recorrer toda la tabla. Sin embargo, ésto generaría un mínimo impacto en inserciones pues no son relevantes para este caso.

2. Clúster sobre pub_place: La columna *pub_place* tiene más de 5k valores distintos, donde unos pocos de ellos como por ejemplo "Madrid" se repiten con mucha frecuencia. Al utilizar un cluster, reducimos el número de bloques accedidos. Sin embargo, requiere modificar los scripts de creación *NEW_creation*, cargar los datos de nuevo y en caso de que se sobrecarguen los buckets, pueden quedar bloques vacíos.

3. Uso de Hints en las consultas del Workload: El uso de hints forzará el uso de los índices creados, aumentando su efectividad. Sin embargo, según las indicaciones en la segunda clase de laboratorio, no se recomienda la modificación del workload proporcionado por lo que no utilizaremos este método.

3 Diseño Físico

Siguiendo el análisis realizado, propón y describe un diseño físico completo (al menos uno, puedes proponer varias alternativas).
Hemos decidido utilizar un diseño para la práctica basado en el uso de un clúster e índices. La razón es clara, aportar el mayor equilibrio entre rendimiento y coste a través de las necesidades evaluadas y analizadas en la situación inicial de nuestra bbdd.

Ten en cuenta que un cambio que mejora un proceso puede estar perjudicando a otros. Justifica todas las decisiones de diseño tomadas.
El hecho de utilizar el clúster *places* sobre la columna *pub_place* de la tabla *editions* mejorará notablemente las consultas que tengan relación con dicha variable, sin embargo, somos conscientes de que ésto puedo perjudicar otras operaciones que no filtren por dicha columna. Por ejemplo, futuras consultas que accedan por *isbn*, puesto que el diseño de nuestro clúster puede producir un mayor número de saltos de bloque o una mayor dispersión.

A pesar de estos posibles inconvenientes, hemos decidido sacrificar la eficiencia de estos otros accesos en pro de optimizar las consultas definidas en el paquete PKG_COSTES, siendo éste el objetivo principal de nuestro diseño físico.

Implementa los diseños físicos en SQL para Oracle (incluye sólo el código nuevo).
Implementación del clúster sobre pub_place:
Para la implementación de dicho clúster deberemos de modificar el script de creación *NEW_creation*, justamente en la creación de la tabla *Editions*. Se adjunta el código a continuación:

```
Unset
CREATE CLUSTER places(pub_place varchar2(50))
SINGLE TABLE HASHKEYS 251;

CREATE TABLE Editions(
  ISBN               VARCHAR2(20),
  TITLE              VARCHAR2(200) NOT NULL,
  AUTHOR             VARCHAR2(100) NOT NULL,
  LANGUAGE           VARCHAR2(50) default('Spanish') NOT NULL,
  ALT_LANGUAGES      VARCHAR2(50),
  EDITION            VARCHAR2(50),
  PUBLISHER          VARCHAR2(100),
  EXTENSION          VARCHAR2(50),
  SERIES             VARCHAR2(50),
  COPYRIGHT          VARCHAR2(20),
  PUB_PLACE          VARCHAR2(50),
  DIMENSIONS         VARCHAR2(50),
  PHY_FEATURES       VARCHAR2(200),
  MATERIALS          VARCHAR2(200),
  NOTES              VARCHAR2(500),
  NATIONAL_LIB_ID    VARCHAR2(20) NOT NULL,
  URL                VARCHAR2(200),
  CONSTRAINT pk_editions PRIMARY KEY(isbn),
```

```
  CONSTRAINT uk_editions UNIQUE (national_lib_id),
  CONSTRAINT fk_editions_books FOREIGN KEY(title,author)
    REFERENCES books(title,author)
) CLUSTER cl_pub_place(pub_place);
```

Implementación de los índices:
Una vez realizada la rehabilitation de nuestra bbdd con las actualizaciones definidas anteriormente, crearemos los siguientes índices:

```
Unset
CREATE INDEX idx_pub_place ON editions(pub_place);
CREATE INDEX idx_publisher ON editions(publisher);
CREATE INDEX idx_condition ON copies(condition);
```

4 Evaluación

Mide el rendimiento de la base en la ejecución de la carga de trabajo estándar, tanto sobre el diseño físico inicial (ya realizado en el punto 2) como sobre cada una de las alternativas implementadas.

Una vez realizada la implementación desarrollada en el punto anterior, realizamos un análisis sobre la carga de trabajo proporcionada por el paquete *PKG_COSTES* hemos utilizado el procedimiento *run_test(10)*:

```
SQL> EXEC pkg_costes.run_test(10);
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
RESULTS AT 30/04/2025 17:33:17
TIME CONSUMPTION (run): 972.7 milliseconds.
CONSISTENT GETS (workload):18944 acc
CONSISTENT GETS (weighted average):1894.4 acc

PL/SQL procedure successfully completed.

Elapsed: 00:00:14.92
```

Compara y analiza los resultados obtenidos (comenta las divergencias con los resultados esperados, en su caso).

A continuación a modo de resumen y para facilitar el análisis y la comparativa entre la implementación inicial y la avanzada (añadiendo clúster e índices), se adjunta una tabla de comparación de resultados:

Métrica	Diseño Inicial	Diseño Optimizado	Mejora absoluta	Mejora relativa (%)
Tiempo medio (ms)	1089,2	972,7	116,5	10,7
Consistent gets (total)	60502	18944	41558	68,7
Consistent gets (por consulta)	6050,2	1894,4	4155,8	68,7

A través de un análisis exhaustivo de los resultados, observamos como el tiempo de ejecución total se ha reducido un 10.7%, lo cual es una pequeña mejora respecto a la situación inicial de nuestra bbdd. Por otra parte, los *consistent gets* han sido reducidos un 69%, lo que implica una mejora muy significativa en términos de eficiencia en el acceso a los datos.

Finalmente, podemos concluir con que nuestra propuesta de diseño implica una optimización del número de bloques accedidos propiciando a su vez una mejora de la eficiencia del buffer cache.

Destacar además que dicha mejoría es especialmente notable en consultas sobre las variables *pub_place*, *publisher* y *condition*, las cuales han sido cubiertas por el clúster y/o índices auxiliares.

5 Conclusiones Finales

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

Una vez realizada toda la evaluación de los resultados de nuestra propuesta de diseño podemos asegurar que este nuevo diseño físico ha mejorado efectivamente al reducir accesos de memoria secundaria, optimizando así el rendimiento general de nuestra bbdd.

A pesar de que la mejoría respecto al tiempo total no es elevada, la reducción de *consistent gets* demuestra que nuestra bbdd accede a un número mucho menor de bloques por consulta.

Por todo ello, consideramos que nuestra propuesta de combinar el clúster sobre *pub_place* con los tres índices propuestos ha sido una decisión acertada y justificada.

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.).

Desde nuestra perspectiva, la práctica tiene un tamaño adecuado, consideramos la simplicidad con la que comenzamos nos aporta bastante fluidez a la hora de comenzar. Al igual que las

indicaciones descritas exhaustivamente. Una sugerencia sobre dichas instrucciones es que estén recopilados preferiblemente en un único documento, ya que a pesar de estar muy bien detalladas, en ocasiones no nos quedaba muy claro cuales seguir al consultar tanto la memoria como el pdf de la práctica y las diapositivas.

Los plazos han sido buenos, se han separado correctamente ambas entregas. Nuestro principal inconveniente es que no hemos podido estar juntos para realizar la práctica en común, por lo que hemos dividido el trabajo e íbamos ayudándonos para presentar la mejor alternativa de diseño.

Nos ha gustado además la dinámica de esta práctica y como hemos avanzado en términos de fluidez en el sistema y en el diseño y entendimiento de todo el código sql. Creemos que es una buena práctica para finalizar la asignatura.

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.).

Creemos que la asignatura está muy bien planteada y se nota que la estructura del temario está cuidada. Las clases prácticas eran de mucha ayuda al enseñarnos detalladamente el funcionamiento de sql, lo que creemos más relevante de la asignatura para nuestro futuro laboral. Algunos temas que nos han parecido relevantes son los relacionados con el Modelo relacional dinámico y extensiones, así como Organizaciones base y auxiliares.