|  |  |  |  |
| --- | --- | --- | --- |
| **Alumno/a:** | **CARLOS RUBIO OLIVARES** | **NIA:** | **100405834** |
| **Alumno/a:** | **JORGE RODRIGUEZ FRAILE** | **NIA:** | **100405951** |

# Introducción

Este proyecto lo vamos a afrontar con ‘fuerza bruta’, es decir, vamos a realizar todos los cambios que creamos convenientes, y basándonos en los resultados y en la apariencia interna del diseño, veremos si de verdad merece la pena mantener estos cambios para mejorar la estructura o si, sin embargo, la empeoran (global o individualmente).

# Análisis

Explica el diseño físico actual (inicial) y describe la carga de trabajo prototípica (procesos frecuentes). Analiza cada instrucción de la carga de trabajo (todas las consultas, inserciones y actualizaciones). Señala las debilidades y fortalezas de ese diseño inicial atendiendo a las necesidades (carga de trabajo). 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).

Partimos de un diseño XXXX (creo que era secuencial no consecutivo) con tamaño de cubo de 8K, con un índice por cada tabla según la clave primaria y otro por clave alternativa si la hay. Este análisis, nos permite saber que es eficiente en acceso a la totalidad y en ahorro de espacio, por lo que nos centraremos en el mejorar el acceso aleatorio. Hacemos un test incial:

RESULTS AT 01/05/20

TIME CONSUMPTION: 66777,1 milliseconds.

CONSISTENT GETS: 67586,6 blocks

Teniendo en cuenta el resultado del test, hay bastantes cosas que mejorar, sobre todo porque muchas querys y views utilizan un full scan a las tablas pertinentes, por esto, creemos que una de nuestras mejores herramientas para solucionar esto pueden ser los índices, aunque debemos ir con cuidado, ya que estos también ocupan espacio y si no se implementan de la manera adecuada pueden generar malos resultados. Por otro lado, también creemos que los cluster nos pueden ayudar a mejorar estos aspectos, sobre todo en torno a las vistas, aunque el lado negativo es que los clusters pueden generar un impacto grave sobre la estructura general a pesar de mejorar algún que otro aspecto individual. También tendremos en cuenta el uso del pctfree y pctused, sobre todo en tablas como comments o proposals, donde apenas se modifica, pero las inserciones pueden ser masivas.

En definitiva, creemos que nuestra mejor arma contra esta organización serán los índices, aunque debemos tener cuidado y no sobrecargar la organización con ellos; en cuanto a los clusters, no estamos muy seguros de si funcionarán de la manera que esperamos, pero intentaremos probar alguna que otra posibilidad.

# Diseño Físico

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

Para nuestra primera organización hemos decidido indizar ciertas tablas para facilitar algunos tipos de búsquedas. La primera tabla en la que hemos pensado, obviamente, ha sido movies, ya que es sobre la que más operaciones y consultas se ejecutan. Nuestra primera opción ha sido hacer un índice primario con los atributos películas y director, pero al intentar crear este nos hemos dado cuenta de que esta tabla ya estaba indizada de esta manera, por lo que hemos pensado en crear un índice secundario sobre director, para facilitar las búsquedas sobre estos. Para comprobar los resultados hemos hecho que se impriman todos los directores, en la organización base tenemos un tiempo total de 11 segundos aprox. y 813 accesos a bloque, con esta nueva indización obtenemos 3 segundos y 716 accesos a bloque, también los hemos probado con diferentes atributos, y el resultado es favorable para la nueva modificación, por lo que podemos asumir que mejora el diseño de las consultas en la tabla movies. Una indización con title no sería muy efectivo, ya que hay muchas películas únicas que no comparten nombre.

Por otro lado, también hemos creado otro índice para la tabla comments, que favorece la búsqueda de comentarios por nick, pero en cuanto a la vista de captain aragna y leader no los benefician en nada, ya que necesitan obtener datos de todos los comentarios de los usuarios, por lo que hacer un full scan con la clave primaria es óptimo. También hemos aplicado el mismo proceso con una indización a proposals, y hemos acabado obteniendo el mismo resultado, solo que esta si que beneficia a la vista Captain Aragna, por el hecho de que es una manera más rápida de obtener todas las proposals de un usuario para poder luego ver si el mismo las ha comentado o no.

Ahora que hemos creado algunos índices, hemos decidido probar a insertar algún que otro cluster, sobre todo para mejorar el rendimiento de nuestras vistas, teniendo en cuenta su estructura interna pensamos que lo mas efectivo sería crear un cluster de nick y club, creando su índice en el tablespace por defecto de 8K, el resultado obtenido es muy negativo para captain aragna, triplicando sus accesos a bloques, pero beneficioso para Leader, que los reduce a la mitad. Si creamos este cluster solo con nick, o solo con club, el output es bastante similar. Con estos resultados llegamos a la conclusión de que no nos es muy beneficioso crear clusters para mejorar views, ya que, aunque mejore bastante leader, el daño es muy grande para la view de captain aragna.

Estudiadas las vistas, vamos a intentar mejorar las querys de la prueba:

En cuanto a la primera query, tratamos de crear un índice secundario en genre\_movies utilizando genre, para que se puedan obtener de una manera más fácil los géneros de una película, pero, para nuestra sorpresa el número de bloques asciende de 184 a 238, seguramente por el hecho de que se usa un intersect y esto afecta a la búsqueda, obtenemos resultados similares al cambiar dicho índice de tablespace. En cuanto a esta query no creemos que se pueda mejorar de otra manera, por lo que la dejaremos como está.

Para la segunda query hemos decidido volver a probar a introducir un cluster, esta vez para nick, esperando que, aunque afectará de manera global, esta query fuera más efectiva, pero de nuevo, el cluster resulta muy ineficiente, aumentando en 10000 el número de bloques leídos, por lo que abandonamos esa idea, e intentamos añadir algún índice para ver si podemos incrementar su efectividad. Probamos con un índice secundario de nick en la tabla membership, y conseguimos una mejora de 120 bloques aproximadamente, por lo tanto, en lo que respecta a esta query, hemos conseguido mejorarla con un nuevo índice.

Por último, en la tercera query hemos creado un índice de título y director en comments, ya que, esencialmente, es lo único que busca en las tablas. El resultado es muy bueno, ya que, de 10873 bloques leídos, hemos pasado a 724, por lo que mantenemos este cambio.

Al terminar estos cambios, los permanentes acaban siendo la creación de los siguientes índices:

-CREATE INDEX MEMBER\_INDEX\_PROPOSALS ON PROPOSALS (MEMBER) TABLESPACE TAB\_8K;

-CREATE INDEX NICK\_INDEX\_COMMENTS ON COMMENTS (NICK) TABLESPACE TAB\_8K;

-CREATE INDEX NICK\_MEMBERSHIP\_INDES ON MEMBERSHIP (NICK) TABLESPACE TAB\_8K;

-CREATE INDEX DIRECTORTITLE\_INDEX\_COMMENTS ON COMMENTS (TITLE, DIRECTOR) TABLESPACE TAB\_8K;

-CREATE INDEX DIRECTOR\_INDEX\_MOVIES ON MOVIES (DIRECTOR) TABLESPACE TAB\_8K;

Con estos 5 índices creados, hemos conseguido una reducción de 700 bloques aproximadamente en las dos vistas de la prueba. Una vez hecho esto, hemos comprobado si es más efectivo cambiar el tablespace de algunos de estos índices, y decidimos mover NICK\_INDEX\_COMMENTS a un bloque de 2k, ya que conseguimos un resultado muy parecido al de 8k, pero con significativamente menos espacio, en cuanto a los demás índices, muchos mejoran en 16k, pero la diferencia es tan mínima que decidimos dejarlos en el tablespace predeterminado, puesto que con 16K ocuparían el doble y podrían no llenarse por completo.

Lo siguiente que pasamos a comprobar son cambios en el pctused y pctfree, intentamos modificar algunas tablas como comments o proposals, donde las modificaciones son mínimas, pero el cambio de pctfree o pctused no afecta en lo más mínimo el resultado de la prueba, por lo que lo dejamos como estaba.

Una vez comprobado el apartado de las consultas en la prueba, vamos a estudiar cómo mejorar las inserciones:

El primer insert masivo que se hace (en membership) ya usa nuestro índice NICK\_MEMBERSHIP\_INDEX, tras un total de 5 inserciones, obtenemos una media de 886 bloques accedidos, mientras que sin este índice obtenemos una media de 890 bloques, por lo que dejamos nuestro índice, que mejora el rendimiento de la inserción.

El siguiente caso es un insert en la tabla proposals; en esta inserción se hace una consulta a los clubs activos, por lo que creamos un índice con el atributo end\_date, el resultado es bastante positivo, ya que sin el índice obtenemos una media de inserción de 2108 bloques, mientras que con el obtenemos 1158 bloques.

El último insert es en la tabla comments, por lo que intentamos crear un índice para club y nick, pero el resultado es bastante peor al original, con una diferencia de 4000 bloques aproximadamente, por lo que no aplicamos ninguna modificación en este sentido.

En definitiva, estos son todos los cambios que hemos realizado a nuestro diseño físico, comprobando y dejando solo aquellos que mejoren los procesos individualmente, creemos que hemos obtenido un diseño bastante óptimo y objetivamente mejor al original, pero la manera definitiva de comprobarlo es de manera general, ejecutando el paquete de prueba.

# Evaluación

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

Este es el valor inicial de nuestra organización básica, sin ningún cambio:

RESULTS AT 01/05/20

TIME CONSUMPTION: 66777,1 milliseconds.

CONSISTENT GETS: 67586,6 blocks

Una vez insertados los 5 primeros índices, obtenemos muchísimos menos bloques, pero cabe recalcar que al volver a ejecutar esta prueba obtenemos unos 200 bloques menos en comparación a la organización inicial, por lo que este resultado no habría que tenerlo en cuenta, aún así, en los siguientes runs se puede apreciar de una mejor manera la reducción de los bloques leídos.

RESULTS AT 02/05/20

TIME CONSUMPTION: 68606,8 milliseconds.

CONSISTENT GETS: 48501,6 blocks

Run test de todas las mejoras, incluido el índice end\_clubs\_index, con estas mejoras incluidas, el resultado es bastante bueno, por lo que podemos empezar a afirmar que las mejoras que hemos incluido si que mejoran la organización de una manera tanto global como individual, aún con esto, no estábamos muy seguros de si nuestro índice end\_clubs\_index realmente mejoraba nuestro diseño.

RESULTS AT 03/05/20

TIME CONSUMPTION: 69357,1 milliseconds.

CONSISTENT GETS: 59469,7 blocks

Run test de todas las mejoras, excepto el índice end\_clubs\_index, con estos resultados podemos ver perfectamente que end\_clubs\_index sí que mejora nuestro diseño, hemos hecho varias runs con esta organización para asegurarnos lo máximo posible, como se puede observar.

RESULTS AT 03/05/20

TIME CONSUMPTION: 68859,3 milliseconds.

CONSISTENT GETS: 63092,8 blocks

RESULTS AT 03/05/20

TIME CONSUMPTION: 68663,1 milliseconds.

CONSISTENT GETS: 78957 blocks

RESULTS AT 03/05/20

TIME CONSUMPTION: 68662,8 milliseconds.

CONSISTENT GETS: 133977,7 blocks

RESULTS AT 03/05/20

TIME CONSUMPTION: 68223,4 milliseconds.

CONSISTENT GETS: 59664,8 blocks

Run test de todas las mejoras, excepto el índice end\_clubs\_index, además de poner todos los índices a 8K, excepto NICK\_INDEX\_COMMENTS, que le hemos puesto un tamaño de 2K, con estas pruebas queremos asegurarnos si los tamaños de bloque son correctos, o podríamos cambiar alguno.

RESULTS AT 03/05/20

TIME CONSUMPTION: 69681,1 milliseconds.

CONSISTENT GETS: 61033,1 blocks

Runtest de todas las mejoras, excepto el índice y todos los índices 8K, aquí vamos a comprobar de nuevo otra combinación de tablespaces con los índices.

RESULTS AT 03/05/20

TIME CONSUMPTION: 68972,5 milliseconds.

CONSISTENT GETS: 117680,4 blocks

RESULTS AT 03/05/20

TIME CONSUMPTION: 69146,5 milliseconds.

CONSISTENT GETS: 61816,5 blocks

Con estos resultados llegamos a la conclusión de que todas las mejoras que hemos ido añadiendo individualmente también tienen un efecto positivo en un concepto global, aunque hemos dudado bastante a la hora de definir los índices en el tablespace adecuado, creemos que dejando NICK\_INDEX\_COMMENTS a 2K y el resto a 8K es lo más correcto, mientras que los clusters son un refuerzo negativo de manera global. Como se ha podido ver, hemos tenido que probar varias veces debido a la inconsistencia del cuerpo del test, pero aún así hemos podido ver resultados que nos han ayudado a tomar estas decisiones. Por otro lado, hemos conseguido que nuestra organización fluctue entre 48000 y 60000 bloques aproximadamente, lo que consideramos una buena mejora.

# Conclusiones Finales

En cuanto a esta práctica, hemos aprendido bastante a como optimizar una base de datos, y en ver que afecta negativamente a las mismas a la hora de realizar cambios u operaciones sobre ellas. La mayoría del tiempo que hemos volcado en este proyecto ha sido en probar cosas que, en realidad, no sabíamos como iban a afectar a nuestra base de datos, pero teniendo en cuenta los resultados y la apariencia interna de la base hemos llegado a diversas conclusiones.

En general, esta asignatura ha sido algo pesada en la parte de ficheros, pero al combinarla con la primera parte de diseño se obtiene una mejor vista de la utilidad de la asignatura, además de su ¿utilidad?.