Introducción a los índices

Tiempo estimado: 30min

Los índices son un aspecto muy importante de rendimiento que tanto desarrolladores como administradores de bases de datos deben tener muy en cuenta y presente. Nuestro tratamiento de los índices se realiza en dos lecciones. En ésta, se introduce los índices y en la siguiente se presenta cómo los soporta ArangoDB.

La lección comienza introduciendo los conceptos de índice y de elemento de indexación. A continuación, se hace hincapié en los escaneos de índices, distinguiéndolos de los escaneos de colección. Después se presenta varias clasificaciones de índice, atendiendo a varios puntos de vista. Finalmente, se habla del factor de ocupación, de la fragmentación de índice, de la compactación y de métodos de indexación y creación.

Al finalizar la lección, el estudiante sabrá:

- Qué es un índice y para qué sirven.
- Qué tipos de índices hay.
- Qué tipos de escaneos de datos puede realizar el motor de bases de datos.
- Qué son las consultas cubiertas por índice.
- Cuáles son las ventajas y desventajas de los índices.
- Qué es el factor de ocupación y para qué ese usa.
- Qué es un índice fragmentado y cómo se puede resolver.
- Cuál es el método de indexación más utilizado.
- Cuáles son los dos tipos de creación de índices: bloqueadora y no bloqueadora.

Introducción

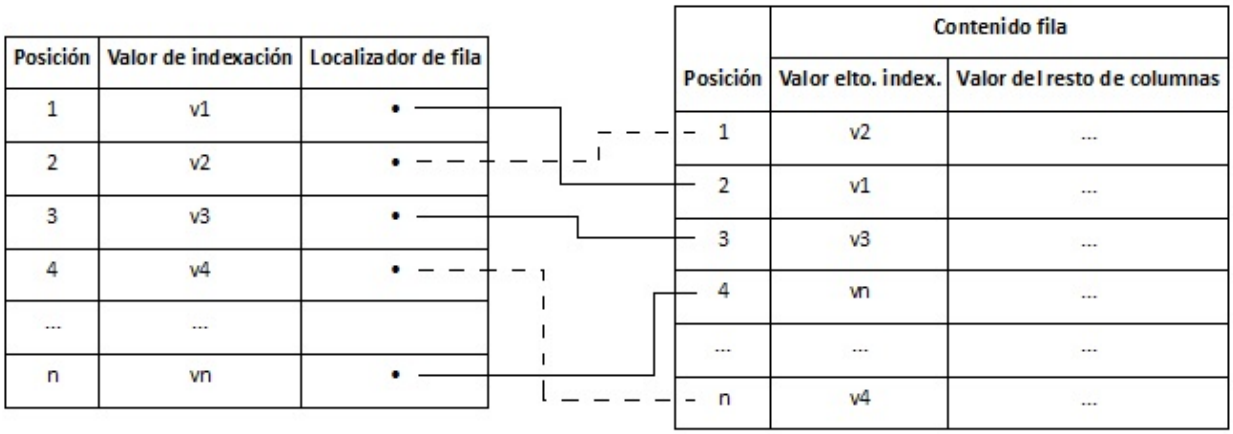
Un índice (*index*) es un objeto que mantiene ordenadas, en una estructura de datos aparte, los documentos de una determinada colección para facilitar y mejorar su acceso, de una manera más rápida que si no existiera el índice, con la consiguiente mejora del rendimiento de la base de datos. Los índices son soportados tanto por sistemas de gestión de bases de datos NoSQL como SQL.

Los índices se definen a nivel de colección, pudiendo asociar a una misma colección tantos índices como sea necesario.

Estos índices ayudarán al motor de bases de datos a reducir la cantidad de datos necesarios que debe acceder para resolver las consultas. Al reducirse la cantidad de documentos que debe acceder, reduce también la E/S a disco y, por ende, mejora el rendimiento de la base de datos. Los índices se utilizan para determinar qué documentos cumplen la condición de búsqueda, solicitando así sólo los bloques de disco que contienen esos documentos.

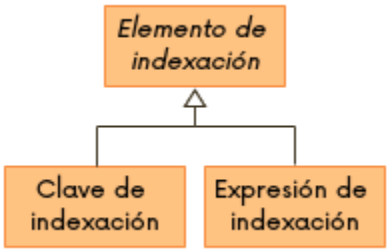
Elementos de indexación

Un elemento de indexación (*index item*) es el medio o dato a través del cual se indexa. Un índice se puede ilustrar como un conjunto de entradas, donde cada documento dispone de su propia entrada de indexación (*index entry*) en el índice y está formada por el valor para el elemento de indexación y un puntero a la ubicación de ese documento, conocido formalmente como localizador de documento (*document locator*). Gráficamente, se puede visualizar como muestra la siguiente ilustración:



Como los índices se encuentran ordenados, el motor de base de datos puede encontrar aquellas entradas que tienen un determinado valor de indexación muy rápidamente. Una vez encontradas las entradas, le bastará con seguir el localizador para acceder al documento que contiene ese valor de indexación. Con los índices, el motor de bases de datos puede encontrar mucho más rápida y fácilmente los documentos que contienen un determinado valor para el elemento de indexación.

Básicamente, los elementos de indexación se pueden clasificar en claves y expresiones. Todo índice debe utilizar una clave o una expresión como elemento de indexación, pero no ambos.



El valor de una clave de indexación (*index key*) se forma a partir del valor de uno o más campos de documento. Mientras que el valor de una expresión de indexación (*index expression*) es el resultado de la evaluación de una determinada expresión con el documento.

No todos los sistemas de gestión de bases de datos soportan ambos tipos de elementos de indexación. Por ejemplo, en ArangoDB, Cassandra y SQLite sólo se puede utilizar claves de indexación. Mientras que, por ejemplo, las expresiones de indexación sí son soportadas por PostgreSQL y SQL Server.

Claves de indexación

Una clave de indexación (*index key*) utiliza como valor de indexación los valores de uno o más campos. Así, el valor de indexación de un documento estará formado por la concatenación del valor particular de cada uno de los campos que forman la clave. Un índice que utiliza una clave como elemento de indexación se conoce formalmente como índice de clave (*key index*).

Una buena elección de los campos que forman una clave de indexación es vital para mejorar el rendimiento de la instancia. No hay que elegirlas aleatoriamente, es necesario conocer cómo se utiliza la colección para poder hacer la mejor elección.

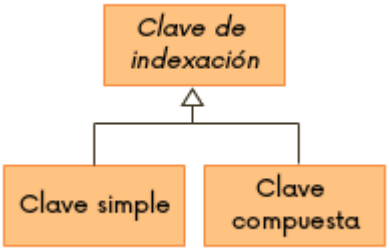
El mantenimiento de los índices, es decir, la actualización de sus entradas, es transparente a los usuarios. Los índices los mantiene actualizados automáticamente el motor de bases de datos. Cada vez que se ejecuta una operación de modificación, si se ve afectado alguno de los campos de la clave de indexación, el motor realiza el cambio tanto en la colección como en el índice. Así pues:

- Cuando se inserta un nuevo documento, además de añadirse a la colección, se añadirá automáticamente la entrada asociada a ese nuevo documento al índice.
- Si se actualiza un documento y alguno de los campos que forman la clave de indexación se ve afectado por esta operación, el motor actualiza automáticamente la entrada del índice asociado al documento para reflejar su nuevo valor y mantener la información sincronizada.

Incluso, si es necesario, el motor ubicará la entrada índice en una nueva posición, todo ello con objeto de seguir manteniendo ordenado el índice.

- Finalmente, si la operación es una supresión de documento, el motor eliminará también del índice su entrada asociada.

Atendiendo al número de campos que forman las claves de indexación, se distingue entre claves simples o compuestas.



Una clave de indexación simple (*simple index key*) es aquella que está formada por un único campo. Una clave de indexación compuesta (*compound index key*) o clave de indexación combinada (*combined index key*) es aquella que está formada por dos o más campos.

No todos los sistemas de gestión de bases de datos soportan ambos tipos de claves de indexación. Por ejemplo, en Cassandra sólo se puede utilizar claves de indexación simples.

Orden de los campos de indexación

Tan importante como definir los índices adecuados, es determinar el orden de los campos que forman las claves de indexación compuestas. El orden de los campos claves (*key field order*) es la colocación o disposición de los campos de la clave de indexación compuesta. Este orden puede hacer que un mismo índice pueda utilizarlo el motor para distintas consultas. Esto se ve más claramente mediante un ejemplo.

Supongamos que se desea indexar dos campos mediante una clave de indexación compuesta, por ejemplo, el nombre y los apellidos de una colección de empleados. Si la clave de indexación la forman los campos nombre y apellidos, en este orden, no será la misma que si es en el orden apellidos y nombre. Eso debe quedar claro. Si se indexa primero por nombre y después por apellidos, lo que significa es que el índice mantendrá, digamos, todas los documentos con el mismo nombre agrupados en un mismo punto, por lo que encontrar todas las personas cuyo nombre es Chuck consistirá en navegar por el índice buscando la entrada para los Chuck. Una vez encontrado el grupo de empleados con el nombre Chuck, el grupo se encontrará ordenado por apellidos.

A su vez, el orden de los campos del índice puede ayudar a no mantener tantos índices. Si ordenamos por nombre y después por apellidos, si lanzamos una consulta que solicite todos los empleados cuyo nombre es Chuck, podrá utilizar el índice compuesto por nombre y apellidos, porque el índice mantiene ordenadas sus entradas primero por nombre y, en caso de tener el mismo nombre, ordena a continuación por apellidos. En cambio, ese mismo índice no se podrá utilizar para encontrar los empleados cuyo apellido sea Levi, porque el índice primero ordena por nombre y, en caso de haber varios documentos con el mismo valor de nombre, entonces ordena esos por apellido; en ningún momento hay una ordenación en primera instancia por apellido. Por lo tanto, un índice compuesto por nombre y apellidos, se puede utilizar tanto en condiciones de búsqueda que incluyan sólo el nombre o bien el nombre y los apellidos, no así las consultas que sólo incluyan los apellidos.

Es importante tener esto en cuenta. En la mayoría de los casos, si se dispone de un índice simple para un determinado campos, por ejemplo, *Ix(c)*, se puede eliminar si se dispone de un índice compuesto donde su primer campo de indexación sea esa mismo campo *c*, por ejemplo, *Ix(c, d)*. Es más, si se dispone de un índice compuesto, por ejemplo, *Ix(c, d)*, se podrá eliminar si se dispone de otro índice compuesto cuyos primeros campos de indexación sean *c* y *d*, como por ejemplo *Ix(c, d, e, f)*.

Expresiones de indexación

Una expresión de indexación (*index expression*) es una expresión cuyo valor, tras su evaluación, actúa como elemento de indexación. Las claves utilizan los valores de los campos como elemento de

indexación, es decir, para ordenar las entradas en el índice. En cambio, un índice que utiliza una expresión como elemento de indexación usará el valor devuelto por el documento por esa expresión para mantener las entradas del índice ordenadas. Un índice que utiliza una expresión de indexación se conoce formalmente como índice de expresión (*expression-based index*). Son menos frecuentes que los índices de clave, pero en algunas ocasiones pueden llegar a ser muy útiles.

No todos los motores de bases de datos soportan este tipo de indexación. Por ejemplo, PostgreSQL y SQL Server sí lo soportan, no así ArangoDB, Cassandra y SQLite.

Cardinalidad de los índices

Se define formalmente el término cardinalidad de los índices (*index cardinality*) como el número de valores distintos que puede haber para el elemento de indexación, ya sea una clave o una expresión de indexación.

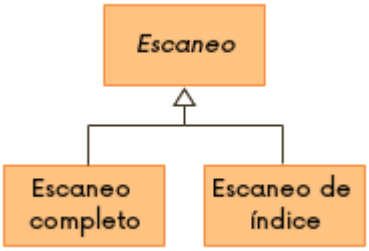
Tipo de ordenación

Recordemos que los índices se encuentran ordenados, es decir, sus registros se encuentran ordenados atendiendo a los valores de los elementos de indexación. El tipo de ordenación (*sort type*) indica el método de ordenación de las entradas, ascendente o descendente, y la ubicación de las entradas con nulos para la clave de indexación.

La mayoría de los motores permiten indicar este tipo de ordenación.

Escaneos de índice

Una de las principales utilidades de los índices es la realización de búsquedas indexadas, en vez de los escaneos de colección completos. Un escaneo de colección completo (*full collection scan*), también conocido como escaneo secuencial (*sequential scan*), es una exploración completa de una colección, o sea, de todos y cada uno de sus documentos, con el objeto de encontrar aquellos que cumplen una determinada condición de búsqueda.



Recordemos que para que un documento sea procesado por el motor de bases de datos es necesario que se encuentre cargado en memoria, en un búfer interno de la instancia. Debe quedar claro, pues, que un escaneo completo conlleva leer todos los bloques de disco y cargarlos en la memoria con el consiguiente consumo de recursos y la reducción del rendimiento de la base de datos. Es necesario cargar todas los documentos en memoria porque es la única manera que tiene el motor de visitar cada uno de ellos para comprobar cuáles cumplen la condición de búsqueda. En colecciones pequeñas, la reducción del rendimiento, que conlleva el escaneo completo, puede ser baja y aceptable, pero en colecciones con cientos de miles o de millones de documentos, puede llegar a ser intolerable.

Para evitar los escaneos completos de colección, se utiliza principalmente los índices. Cuando una consulta se puede resolver mediante el uso de un índice, se habla de búsqueda indexada (*indexed search*), es decir, es una búsqueda que se realiza contra un índice, en vez de contra una colección. Las búsquedas indexadas realizan escaneos de índice (*index scan*) en vez de escaneos completos; ahora, en vez de recorrer todas los documentos de la colección, se recorre el índice correspondiente. Por un lado, no hay que cargar en memoria los documentos de la colección, en su lugar, se carga las entradas del índice que, en la mayoría de los casos, son siempre mucho más pequeñas, generando así menor E/S a disco. Una vez se conoce cuáles son los documentos que cumplen la condición de búsqueda, entonces se solicita a disco sólo los bloques que contienen *esos* documentos. Por otro lado, como los índices se encuentran ordenados, al motor le resulta más fácil y rápido encontrar aquello documentos que cumplen la condición de búsqueda, porque tiene que ir a esa parte del índice donde se encuentran los que cumplen la condición. Si no se encuentra en esa sección, está claro que no está y, entonces, puede

abandonar la búsqueda sin necesidad de recorrer todo el índice y sin necesidad de solicitar a disco todos los documentos de la colección. En cambio, en una colección desordenada, sería necesario recorrerla al completo porque un documento que cumpla la condición de búsqueda se puede encontrar en cualquier parte de la colección.

Debe quedar claro que el motor realizará un escaneo completo de colección siempre que no disponga de un índice para la realización de una búsqueda indexada o también, todo hay que decirlo, si el índice que puede utilizar puede llevar más E/S que la simple lectura de la colección al completo.

Selectividad de los índices

La selectividad de los índices (*index selectivity*) es una característica, que debe presentar necesariamente todo índice, que implica el grado de utilidad que tiene el índice de cara a las consultas que se ejecutan contra la base de datos. Se habla de índice selectivo (*selective index*) como aquel que presenta una alta selectividad y, por tanto, puede usarse en muchas consultas. El orden de los campos que forman la clave de indexación puede ayudar a convertir un índice en selectivo.

Los índices sólo son útiles, mereciendo la pena la sobrecarga inherente que su mantenimiento obliga al motor de bases de datos, si son selectivos, es decir, si pueden ser usados por muchas consultas o por las consultas más importantes que consumen muchos recursos. Si un índice no es selectivo, mejor suprimirlo.

Consultas cubiertas por índice

Una consulta cubierta por índice (*covered index query*) es aquella cuyos datos son obtenidos usando las propias entradas de un índice, sin necesidad de recurrir a los documentos de la colección. Por una parte, el índice se utiliza para recuperar más rápidamente los documentos que cumplen la condición de búsqueda. Y por otro lado, parte o todos los campos que forman la clave de indexación son los proyectados por la consulta, por lo que el motor puede resolver toda la consulta directamente desde el índice sin necesidad de recurrir a sus respectivos documentos.

Por ejemplo, si se dispone de un índice *Ix(a, b, c)*, cualquier consulta que proyecte los campos de indexación *a*, *b* y *c*, puede resolverse directamente desde el índice, sin necesidad de recurrir a los documentos. En cambio, si la consulta proyecta también el campo *d*, está claro que obligará al motor a buscar los documentos asociados para obtener el valor de ese campo *d* no almacenado en el índice.

Estas consultas son más rápidas que aquellas que utilizan los índices para determinar los documentos que cumplen la condición de búsqueda y, después, necesitan recurrir a los documentos correspondientes para obtener los datos faltantes.

Índices con campos incluidos

Algunos motores de bases de datos permiten añadir a los índices campos que no forman parte de la clave de indexación como en el caso de SQL Server. Este tipo de índices se conoce formalmente como índice con campos incluidos (*index with included fields*). Otros motores como ArangoDB, Cassandra, MongoDB, PostgreSQL o SQLite no permiten los campos incluidos.

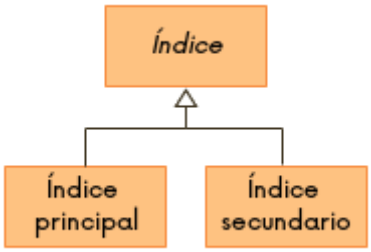
Desventajas de los índices

El uso de índices obliga a tener en cuenta los siguientes aspectos negativos:

- Son estructuras de almacenamiento aparte que ocupan espacio en disco. A medida que se añaden más índices, más recursos de almacenamiento serán necesarios.
- Hay que tenerlos actualizados, obligando al motor a realizar trabajo extra para mantenerlos al día cada vez que se realiza una operación de modificación de datos que afecta al valor del elemento de indexación.

Índices principales y secundarios

Los índices se pueden clasifican en principales y secundarios.

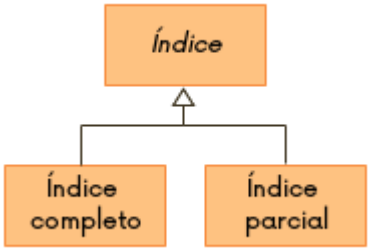


Un índice principal (*primary index*) es aquel cuya clave de indexación es la propia clave principal de los documentos. Por su parte, un índice secundario (*secondary index*) es aquel que no es el principal.

El índice principal lo crea automáticamente la instancia de bases de datos. Mientras que los índices secundarios los tenemos que crear nosotros de manera explícita. El objetivo del índice principal es asegurar el acceso rápido a los documentos mediante su clave principal.

Índices completos y parciales

Otra posible clasificación de los índices se puede realizar atendiendo a qué conjunto de documentos de la colección se indexará. Se distingue entre índice completo e índice parcial.



Un índice completo (*full index*) o índice no disperso (*non-sparse index*) es aquel que indexa todos los documentos de la colección asociada. Es el índice predeterminado en la mayoría de los sistemas de gestión de bases de datos. Este tipo de índices se implementa en muchos motores de bases de datos SQL y NoSQL como, por ejemplo, ArangoDB, Cassandra, MongoDB, PostgreSQL, SQL Server y SQLite.

En cambio, un índice parcial (*partial index*), también conocido como índice disperso (*sparse index*) o índice filtrado (*filtered index*) según el sistema de gestión de bases de datos, es aquel que sólo indexa determinados documentos de la colección asociada, centrándose sólo en aquellos que cumplen una determinada condición de restricción (*constraint condition*) o condición de filtro (*filter condition*). Este tipo de índices son más pequeños que los completos. Lo implementan también muchos sistemas de gestión de bases de datos como ArangoDB, MongoDB, PostgreSQL, SQL Server y SQLite.

Índices únicos

Un índice único (*unique index*) se utiliza para asegurar que una colección no contiene valores duplicados para el elemento de indexación. Cuando se intenta insertar un nuevo documento que presenta un valor de indexación ya existente, el índice lo detectará e impedirá su inserción. O si lo que se está ejecutando es una operación de actualización que modifica el valor de indexación dejándolo a un valor ya existente en el índice, el motor impedirá la actualización.

Factor de ocupación

Algunos sistemas de gestión de bases de datos permiten especificar un factor de ocupación para cada índice, esto es, la cantidad de espacio que debe intentar llenar el motor como máximo de cada página del índice. El factor de ocupación no se puede indicar en todos los motores de bases de datos, entre los que sí lo soportan encontramos PostgreSQL y SQL Server.

Al igual que las colecciones, los índices reparten su contenido en páginas o bloques, según el término utilizado por el sistema de gestión de bases de datos. Una vez el índice ha llenado una página (o bloque), solicita una nueva para seguir añadiendo entradas. El factor de ocupación (*fill factor*) es la

cantidad de espacio que deseamos llene el motor antes de solicitar una nueva página o bloque para registrar nuevas entradas de índice. Viéndolo desde otra perspectiva, indica el espacio libre que deseamos mantener en cada página del índice. Así, por ejemplo, si el factor de ocupación es del 90%, cuando la instancia encuentre el bloque actual al 90%, dejará de escribir en ese bloque y solicitará uno nuevo en el que continuar.

Este espacio libre que se deja en los bloques ayuda al índice a adaptarse a sus modificaciones debido a las operaciones de inserción y actualización. Por ejemplo, supongamos que el índice puede contener, por decir algo, hasta cuatro entradas y que nuestro factor de ocupación es de 75%. Veamos cómo trabajará el índice con un escenario que añade las seis primeras entradas:

1. Cuando se inserte la primera entrada, el índice la escribirá en la primera página. Esto dejará la ocupación de la página en el 25%.
2. Cuando se inserte la segunda entrada, el índice la escribirá también en la primera página, dejando la ocupación al 50%.
3. Con la tercera entrada, se usará también la primera página, dejando ahora la ocupación al 75%.
4. Cuando se almacene la cuarta entrada, el factor de ocupación se habrá alcanzado y automáticamente el índice solicitará una nueva página. Ahora el índice contiene dos páginas: una primera al 75% con tres entradas y otra al 25% con una única entrada.
5. Ahora, cuando aparece la quinta entrada puede ocurrir básicamente dos cosas, recordemos que el índice mantiene su contenido ordenado: primera, que la entrada tenga que ubicarse en la primera página, porque se encuentra antes que la cuarta entrada que se encuentra ubicada en la segunda página, o bien, que la nueva entrada sea posterior a la última añadida.

   Está claro que la segunda opción no le genera mucho dolor de cabeza al índice. Simplemente, va a la última página y añade en ella la nueva entrada. Ahora bien, la primera tiene más miga. Si la nueva entrada tiene que ubicarse entre la tercera, que se encuentra en la primera página, y la cuarta, ubicada en la segunda página, la entrada se puede ubicar sin ningún problema en la primera página, en ese espacio *libre* que se solicitó al índice que mantuviera. No hay que hacer nada extraordinario y la añadidura de la nueva entrada no genera mucho trabajo al índice.
6. En este punto, el índice tiene su primera página al completo, al ciento por ciento, y la segunda al 25%. Si ahora entra una nueva entrada que tiene que ubicarse, debido al valor de su clave de indexación, entre la cuarta y la quinta, el índice no tiene manera de hacerlo sin consumir recursos. Tiene que hacerle hueco entre la última entrada de la primera página y la primera entrada de la segunda página. Ahora, para ver realmente el problema, supongamos que el índice no tiene sólo dos páginas sino que tiene cien. Pues el índice tiene que hacer hueco, le guste o no. Esto consumirá tiempo y recursos. Si el índice es un árbol balanceado, entonces tendrá todavía más trabajo. Tendrá que desplazar entradas y adaptar el árbol a la entrada *conflictiva*. Pues ahora imagine que podría suceder con un índice de miles o millones de entradas.

Como vemos, si el factor de ocupación deja un espacio libre, algunas operaciones pueden usar ese espacio libre facilitando así la adaptación del índice a los cambios. Si los cambios son frecuentes y no hay espacio libre, el índice sobrecargará la instancia y el rendimiento se reducirá, adaptándose peor a los cambios que generalmente son inevitables. También hay que decir que dejar mucho espacio libre, o sea, usar factores de ocupación bajos, hará crecer innecesariamente el tamaño del índice en disco y recorrerlo conllevará más E/S de disco. Para recorrer un índice, al igual que una colección, es necesario que sus bloques se encuentren en memoria.

Por lo general, los índices asociados a colecciones estáticas se definen con un factor de ocupación del 100%, porque no tiene sentido mantener huecos libres ya que los índices no cambiarán a lo largo de su ciclo de vida. En cambio, los índices definidos contra colecciones dinámicas se suelen definir con factores de ocupación del 90% al 100% y, en algunos casos, menores incluso. Algunos sistemas de gestión de bases de datos como PostgreSQL definen un factor de ocupación predeterminado del 90%, mientras que otros como SQL Server lo tienen del 100%, requiriendo así que se especifique explícitamente su valor. El valor adecuado depende de si el índice puede desbaratarse mucho a medida que van entrando nuevas entradas o éstas se van reubicando por lo cambios de los valores de sus elementos de indexación. En caso de duda, se recomienda usar un valor de 90% ó 95%.

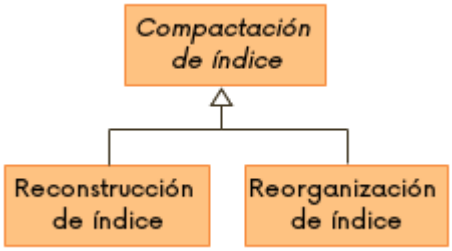
Fragmentación de índice

Con el tiempo, a medida que el índice acepta nuevas entradas y actualiza y elimina otras, el índice se va degradando poco a poco, pudiendo llegar a producirse lo que se conoce formalmente como fragmentación de índice (*index fragmentation*), que no es más que la aparición de páginas del índice fragmentadas, es decir, con huecos libres que, en algunos casos, son tan pequeños que no se pueden asignar a otras entradas. En resumen, la fragmentación es un aspecto negativo de los índices que indica que los datos se encuentran dispersos por el disco, lo que degrada su rendimiento. Todo índice que presenta fragmentación de datos se conoce formalmente como índice fragmentado (*fragmented index*). Por lo general, los índices asociados a colecciones con abundantes operaciones DML suelen acabar fragmentándose más tarde o temprano.

Compactación de índices

La fragmentación se puede arreglar mediante la compactación de índice (*index compactation*), operación que agrupa las entradas dispersas en el disco y las reordena para mejorar su acceso, recuperándose el rendimiento perdido por la fragmentación. La idea es doble: por un lado, reordenar las entradas lo más cerca las unas de las otras; y, por otro lado, reducir el número de páginas que tiene asignadas el índice, lo que hará que cargar el índice requiera menos operaciones de E/S a disco que, recordemos, es uno de los principales cuellos de botella de los sistemas de gestión de bases de datos.

Básicamente, existen dos tipos de compactación, la reconstrucción y la reorganización.



Por lo general, cuando el promedio de fragmentación del índice es inferior al 30%, se suele utilizar la reorganización; en cambio, cuando es mayor al 30%, se utiliza la reconstrucción. La reconstrucción también se suele utilizar cuando el índice se ha corrompido y el motor de bases de datos, entonces, no puede utilizarlo porque no puede garantizar que los resultados sean correctos. Independientemente del tipo de compactación, esta operación la debe llevar a cabo el DBA, tras detectar, por lo general en un informe de rendimiento de la base de datos, que hay fragmentación en el índice.

También es posible planificar operaciones de mantenimiento de reorganización y reconstrucción de índices. Por ejemplo, a algunos DBAs les gusta realizar una operación de reorganización todos los días de la semana, salvo otro día que hacen una reconstrucción; por ejemplo, de domingo a viernes, ambos inclusive, se realiza una reorganización, dejándose la reconstrucción para el sábado. A otros les gusta más realizar una reorganización semanal, sustituyendo la última semana del mes la reorganización por una reconstrucción. Independientemente de cuándo se realice cada operación, los DBAs no deben olvidar planificar estas operaciones para su ejecución en horario de menor carga, por ejemplo, a medianoche o de madrugada.

La reconstrucción de índice (*index rebuild*) o reindexación (*reindex*) destruye el índice y vuelve a crearlo de cero. La reconstrucción del índice se suele llevar a cabo tanto para compactación como para regenerar índices corruptos. Algunos motores de bases de datos como, por ejemplo, SQLite no permiten la reconstrucción de índices. En estos casos, su comportamiento se puede simular a la vieja

usanza: eliminándolos y volviéndolos a crear.

En cambio, la reorganización de índice (*index reorganization*) es una operación de mantenimiento mediante la cual se reorganiza las páginas del índice y su contenido. No se elimina el índice ni se vuelve a crear como en la reconstrucción, se mantiene el índice pero se compacta su contenido eliminando huecos y, si es posible, incluso páginas que acaben quedando libres.

Métodos de indexación

Un método de indexación (*index method*) es el algoritmo utilizado por el índice para indexar sus entradas. El método de indexación más utilizado es el de árboles balanceados (*b-tree*), el cual es soportado por la mayoría de los motores de bases de datos, entre ellos, Cassandra, MongoDB, PostgreSQL, SQL Server y SQLite. El árbol balanceado es una estructura de datos que ordena las entradas del índice mediante un grafo balanceado.

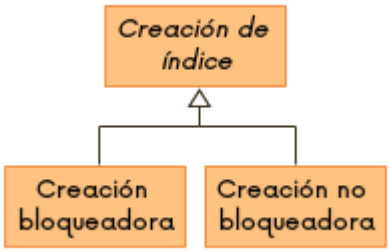
Otro tipo de índice muy utilizado es el índice de huella (*hash index*), el cual se implementa mediante pares clave-valor, no ordenadas. Donde la clave es la huella de la clave de indexación; y el valor, el localizador de documento. Este tipo de índices se puede utilizar únicamente para condiciones de búsqueda que utilizan *todos* los campos de indexación y, además, se comparan *todos* ellos mediante el operador de igualdad.

Un tercer método de indexación es el índice por saltos (*skiplist index*). Mediante este método, las entradas de índices se encuentran ordenadas mediante una lista enlazada. Son útiles para encontrar documentos a partir de los campos de indexación. Estas búsquedas se pueden indicar en las restricciones de búsqueda mediante operadores de igualdad y rangos. También se pueden utilizar en las ordenaciones.

ArangoDB y PostgreSQL implementan varios método o algoritmos de indexación, suministrando así mayor flexibilidad y rendimiento según el uso el elemento de indexación. En caso de duda, se recomienda utilizar los índices de árboles balanceados, como hemos indicado, se adaptan bastante bien a la mayoría de las casuísticas.

Creación de índices

La creación de un índice (*index creation*) es la operación mediante la cual se crea un índice sobre una determinada colección. Un aspecto muy importante a tener en cuenta es si la creación bloquea las colección asociada durante la indexación inicial o, por el contrario, no lo hace. Atendiendo a este bloqueo de la colección a indexar, se distingue entre creación bloqueadora y no bloqueadora.



La creación bloqueadora (*lockable index creation*) es aquella que bloquea la colección mientras extrae su contenido para indexarlo. Este tipo de creación puede afectar considerablemente el rendimiento del motor si la colección tiene muchos recursos pues algunas consultas que accedan a los objetos bloqueados tendrán que esperar hasta que se desbloqueen.

Para resolver el problema, se puede utilizar una creación no bloqueadora (*non-lockable index creation*), la cual crea el índice sin bloquear la colección asociada permitiendo así la ejecución, durante la creación, de instrucciones DML contra la colección bajo indexación. La creación no bloqueadora es más lenta, pero algunas veces es la única posible si se hace en un entorno de producción cuando no se puede impedir la ejecución de instrucciones DML contra ninguna colección, pues hacerlo podría llevar a pérdida de servicio a los usuarios.

Son varios los motores de bases de datos que permiten ambos tipos de creación de índices como, por ejemplo, MongoDB, PostgreSQL y SQL Server.

Mejores prácticas

Las mejores prácticas (o *best practices*) de bases de datos recomiendan principalmente:

- Crear índices para las claves ajenas. Recordemos que las reuniones (JOINs) se realizan principalmente entre claves ajenas y claves principales. Eso significa que el motor de bases de datos invierte muchos recursos en combinar los documentos de las colecciones padre e hija. Si podemos ayudar a que la combinación sea más rápida, el rendimiento mejorará considerablemente.

  Las claves principales se implementan inherentemente mediante índices, pero las claves ajenas no. Por lo que se recomienda encarecidamente crear índices sobre este tipo de claves.

- Crear índices sobre los campos más utilizados en las restricciones o filtros de las consultas. Comprender el comportamiento de las consultas que se ejecutarán más frecuentemente contra la base de datos ayuda muchísimo a elegir los mejores índices.

  También se recomienda prestar especial atención a aquellas consultas que acceden a colecciones con muchos documentos y que suelen consultar hasta el 50% de su contenido. Conociendo estas consultas, se puede determinar fácilmente qué campos son candidatos a ser indexados.

Por el contrario, no se recomienda crear índices innecesarios sobre:

- Colecciones con pocos datos. Por lo general, las colecciones con menos de 1000-10000 documentos no suele ser necesario indexarlas. Pero ojo, si con el tiempo se espera que superen este umbral, mejor indexarla desde el comienzo.

Finalmente, un aspecto muy valorado por los DBAs es que los diseñadores de bases de datos utilicen espacios de tabla, distinguiendo entre los que almacenarán datos de colecciones y de índices. Si una colección almacena sus documentos en un espacio de tabla y los índices en otro, ambos tipos de datos se podrán consultar simultáneamente si, además, cada espacio de tabla se encuentra en un disco aparte. Si ambos tipos de datos se ubican en el mismo disco, habrá más contención sobre el mismo disco.

