Unidad IV: Operación y mantenibilidad

4.1 Bitácoras de trabajo del DBMS

En caso de que sea multiusuario existen muchas ventajas adicionales, donde la BD es con toda probabilidad mucho más grande y compleja. Ofrece control centralizado de su información.

* Es compacto: no hacen falta archivos de papales que pudieran ocupar mucho espacio.

* Es rápido: la máquina puede obtener y modificar con mucha mayor velocidad que un ser humano.

* Es menos laborioso: se elimina gran parte del tedio de mantener archivos a mano.

* Es actual: se dispone en cualquier momento de información precisa y al día.

Tiene aún más importancia, en un ambiente multiusuario, donde la BD es con toda probabilidad mucho más grande y compleja que un uno de un solo usuario. El sistema de BD ofrece a la empresa un control centralizado de su información.

4.1.1. Funciones específica de las bitácoras

La estructura más ampliamente usada para grabar las modificaciones de la base de datos es la Bitácora. Cada registro de la bitácora escribe una única escritura de base de datos y tiene lo siguiente:

- Nombre de la Transaccion
- Valor antiguo
- Valor Nuevo

Es fundamental que siempre se cree un registro en la bitácora cuando se realice una escritura antes de que se modifique la base de datos.
También tenemos la posibilidad de deshacer una modificación que ya se ha escrito en la base de datos, esto se realizará usando el campo del valor antiguo de los registros de la bitácora.
Los registros de la bitácora deben residir en memoria estable como resultado el volumen de datos en la bitácora puede ser exageradamente grande.
Las operaciones COMMIT y ROLLBACK establecen lo que se le conoce como punto de sincronización lo cual representa el límite entre dos transacciones consecutivas, o el final de una unidad lógica de trabajo, y por tanto al punto en el cual la base de datos esta (o debería estar) en un estado de consistencia. Las únicas operaciones que establecen un punto de sincronización son COMMIT, ROLLBACK y el inicio de un programa. Cuando se establece un punto de sincronización:
Se comprometen o anulan todas las modificaciones realizadas por el programa desde el punto de sincronización anterior.
Se pierde todo posible posicionamiento en la base de datos. Se liberan todos los registros bloqueados. Es importante advertir que COMMIT y ROLLBACK terminan las transacción, no el programa.

4.1.2 Recuperación (rollback)

En tecnologías de base de datos, un rollback es una operación que devuelve a la base de datos a algún estado previo. Los Rollbacks son importantes para la integridad de la base de datos, a causa de que significan que la base de datos puede ser restaurada a una copia limpia incluso después de que se han realizado operaciones erróneas. Son cruciales para la recuperación de crashes de un servidor de base de datos; realizando rollback(devuelto) cualquier transacción que estuviera activa en el tiempo del crash, la base de datos es restaurada a un estado consistente.

En SQL, ROLLBACK es un comando que causa que todos los cambios de datos desde la última sentencia BEGIN WORK, o START TRANSACTION sean descartados por el sistema de gestión de base de datos relacional (RDBMS), para que el estado de los datos sea "rolled back"(devuelto) a la forma en que estaba antes de que aquellos cambios tuvieran lugar.

Una sentencia ROLLBACK también publicará cualquier savepoint existente que puediera estar en uso.

En muchos dialectos de SQL, ROLLBACKs son específicos de la conexión. Esto significa que si se hicieron dos conexiones a la misma base de datos, un ROLLBACK hecho sobre una conexión no afectará a cualesquiera otras conexiones. Esto es vital para el buen funcionamiento de la Concurrencia.

La funcionalidad de rollback está normalmente implementada con un Log de transacciones, pero puede también estar implementada mediante control de concurrencia multiversión.

En el proceso de “Rollback”, SQL Server comienza a hacer un rollback de todas las transacciones que no fueron confirmadas además de las que fueron rechazadas, dejando de esta manera la base de datos en un estado consistente.

Este proceso de recuperación en algunos casos puede tardar mucho tiempo debido a la gran cantidad de información que tienen que replicar desde el log de transacciones. Es por eso que la frecuencia con la que se hacen los checkpoints dentro de la base de datos es crucial para el tiempo que tardara el servidor en ejecutar el proceso de recuperación.

Adicionalmente cabe mencionar que en algunas pocas ocasiones el terminar el servicio de SQL Server de manera inesperada puede causar corrupciones de datos, y esto sí es grave debido a que en algunos casos puede ser recuperable la información, pero siempre con un riesgo de perder algo de data, y en otros no es posible arreglar la base de datos, entonces lo único que queda en estas situaciones es la restauración de backups y es ahí donde si se tiene una buena estrategia de backups se puede llegar a recuperar absolutamente toda la información hasta el momento del desastre.

4.1.3 Permanencia (commit)

En cualquier momento, el programa podría decidir que es necesario hacer fallar la transacción, con lo que el sistema deberá revertir todos los cambios hechos por las operaciones ya hechas. En el lenguaje SQL se denomina COMMIT a aplicar_cambios y ROLLBACK a cancelar_cambios.

Las transacciones suelen verse implementadas en sistemas de bases de datos y, más recientemente, se han visto incorporadas a como gestiona un sistema operativo la interacción con un sistema de archivos (como varias características de las bases de datos, debido a que son muy similares arquitectónicamente).

Una sentencia COMMIT en SQL finaliza una transacción de base de datos dentro de un sistema gestor de base de datos relacional (RDBMS) y pone visibles todos los cambios a otros usuarios. El formato general es emitir una sentencia BEGIN WORK, una o más sentencias SQL, y entonces la sentencia COMMIT. Alternativamente, una sentencia ROLLBACK se puede emitir, la cual deshace todo el trabajo realizado desde que se emitió BEGIN WORK. Una sentencia COMMIT publicará cualquiera de los savepoints(puntos de recuperación) existentes que puedan estar en uso.

En términos de transacciones, lo opuesto de commit para descartar los cambios "en tentativa" de una transacción, es un rollback.

4.2 Definición de los modos de operación de un DBMS. (alta, baja, recovery)

El sistema de gestión de bases de datos es esencial para el adecuado funcionamiento y manipulación de los datos contenidos en la base. Se puede definir como: "El Conjunto de programas, procedimientos, lenguajes, etc. que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar

y manipular los datos almacenados en la base, manteniendo su integridad, confidencialidad y seguridad".

Las funciones esenciales de un SGDB son la descripción, manipulación y utilización de los datos.

Descripción: Incluye la descripción de: Los elementos de datos, su estructura, sus interrelaciones, sus validaciones. Tanto a nivel externo como lógico global e interno esta descripción es realizada mediante un LDD o Lenguaje de Descripción de Datos.

Manipulación: Permite: Buscar, Añadir, Suprimir y Modificar los datos contenidos en la Base de Datos.

La manipulación misma supone: Definir un criterio de selección, Definir la estructura lógica a recuperar, Acceder a la estructura física. Esta manipulación es realizada mediante un LMD o Lenguaje de Manipulación de Datos.

Utilización: La utilización permite acceder a la base de datos, no a nivel de datos sino a la base como tal, para lo cual: Reúne las interfaces de los usuarios y suministra procedimientos para el administrador.

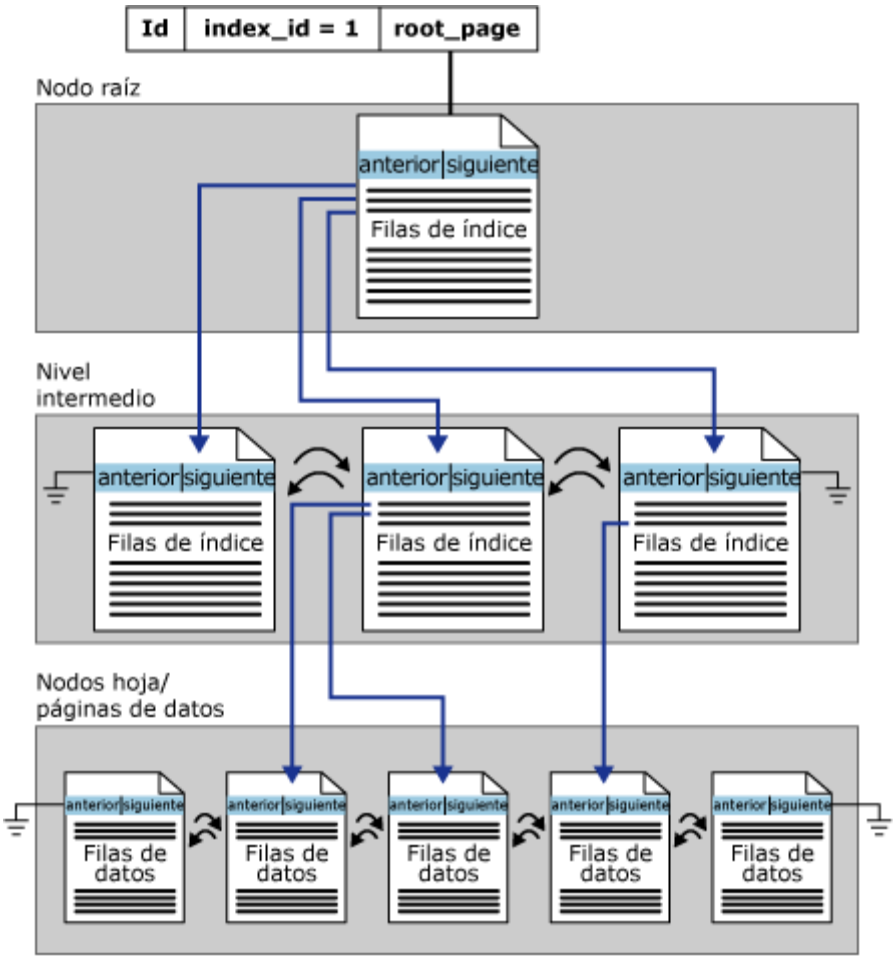
En términos ideales, un DBMS debe contar con estas funciones, sin embargo, no todos las poseen, así existen algunos manejadores que no cumplen la función de respaldo o de seguridad, dejándola al usuario o administrador; sin embargo un DBMS que sea completo y que deba manejar una base de datos multiusuario grande, es conveniente que cuente con todas estas operaciones.

4.4. Manejo de índices

Los índices son "estructuras" alternativa a la organización de los datos en una tabla. El propósito de los índices es acelerar el acceso a los datos mediante operaciones físicas más rápidas y efectivas. Para entender mejor la importancia de un índice pongamos un ejemplo; imagínate que tienes delante las páginas amarillas, y deseas buscar el teléfono de Manuel Salazar que vive en Alicante. Lo que harás será buscar en ese pesado libro la población Alicante, y guiándote por la cabecera de las páginas buscarás los apellidos que empiezan por S de Salazar.

De esa forma localizarás más rápido el apellido Salazar. Pues bien, enhorabuena, has estado usando un índice.

4.4.1 Tipos de índices



Los índices agrupados, definen el orden en que almacenan las filas de la tabla (nodos hoja/página de datos de la imagen anterior). La clave del índice agrupado es el elemento clave para esta ordenación; el índice agrupado se implementa como una estructura de árbol b que ayuda a que la recuperación de las filas a partir de los valores de las claves del índice agrupado sea más rápida. Las páginas de cada nivel del índice, incluidas las páginas de datos del nivel hoja, se vinculan en una lista con vínculos dobles. Además, el desplazamiento de un nivel a otro se produce recorriendo los valores de claves.

Consideraciones para usar índices agrupados

· Columnas selectivas

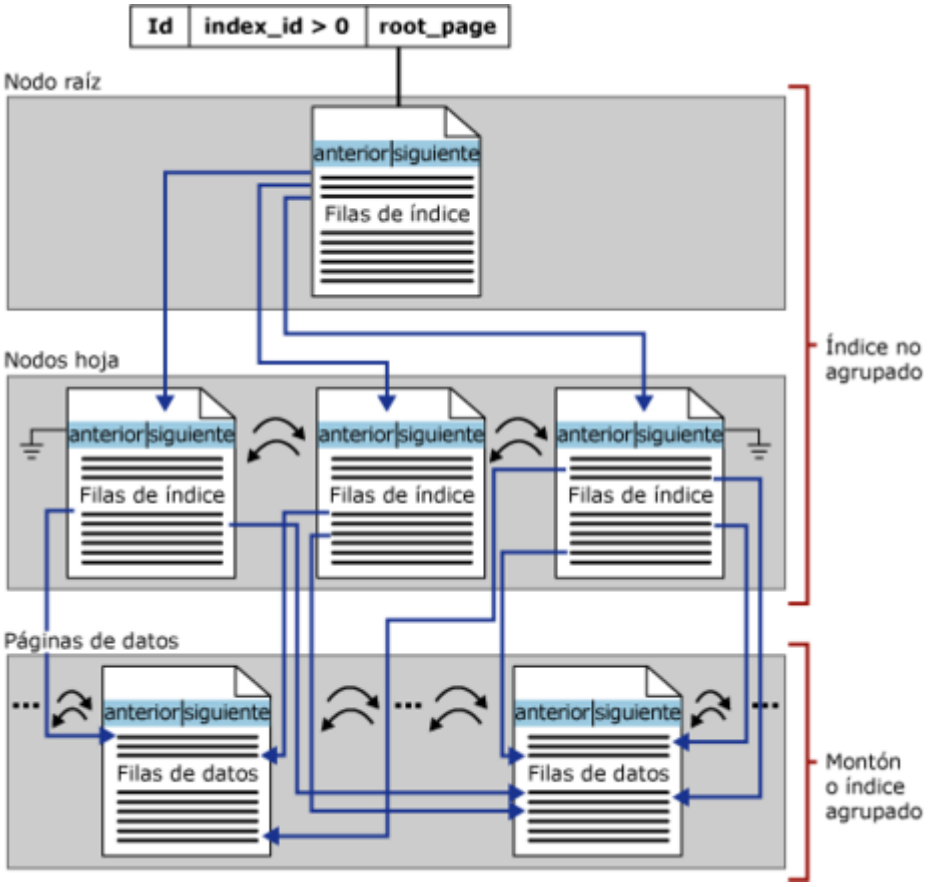
· columnas afectadas en consultas

· Columnas accedidas "secuencialmente"

· Columnas implicadas en JOIN, GROUP BY

· Acceso muy rápido a filas: lookups

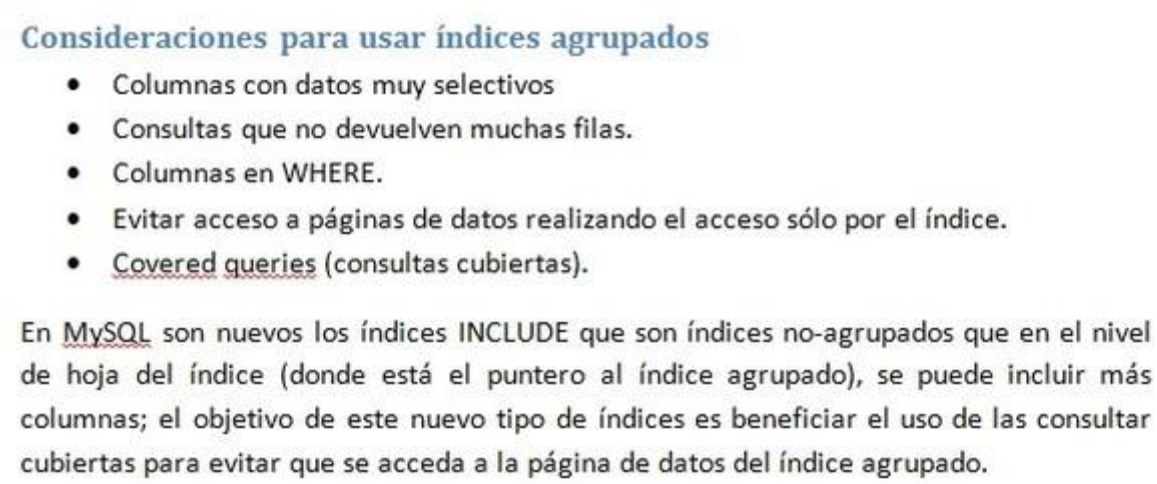
Indices No Agrupados



Los índices no agrupados tienen la misma estructura de árbol b que los índices agrupados, con algunos matices; como hemos visto antes, en los índices agrupados, en el último nivel del índice (nivel de hoja) están los datos; en los

índices no-agrupados, en el nivel de hoja del índice, hay un puntero a la localización física de la fila correspondiente en el índice agrupado. Además, la ordenación de las filas del índice está construida en base a la(s) columna(s) indexadas, lo cual no quiere decir (a diferencia de los índices agrupados), que la organización física de las páginas de datos corresponda con el índice.

Consideraciones para usar



4.4.2 Reorganización de índices

Un paquete puede usar la tarea Reorganizar índice para reorganizar los índices de una base de datos individual o de varias bases de datos. Si la tarea solo reorganiza los índices de una base de datos individual, puede elegir las vistas o las tablas cuyos índices reorganiza la tarea. La tarea Reorganizar índice también incluye la opción de compactar datos de objetos grandes. Los datos de objetos grandes son datos de tipo ***image, text, ntext, varchar(max), nvarchar(max), varbinary(max) o xml***.

La tarea Reorganizar índice encapsula la instrucción ALTER INDEX de Transact-SQL. Si elige compactar datos de objetos grandes, la instrucción utiliza la cláusula

REORGANIZE WITH (LOB_COMPACTION = ON); en caso contrario, se establece LOB_COMPACTION en OFF

Dentro de las tareas habituales de Mantenimiento de las Bases de Datos se encuentran aquellas destinadas al control y respaldo de las mismas como ser: Control de Integridad, Chequeo de Consistencia, Copias de Seguridad o Compactación de las bases.

Pero también es necesario ejecutar trabajos de mantenimiento cuyos objetivos sean el de mantener la performance de las bases de datos y evitar su degradación.

Esos trabajos son la Reorganización de Índices y la Actualización de Estadísticas.

Estos trabajos son independientes del estado de la base de datos. Puede ocurrir que a la base le falten estudios de optimización pero, al menos, mantendremos la performance actual.

Si la base se encuentra optimizada, entonces más aún, son necesarios para evitar la degradación producto del uso continuo.

Cualquiera de estos trabajos deben realizarse fuera de línea por motivos de: alto consumo de recurso y bloqueo de las tablas en el momento de ejecución.

Las tablas que contienen índices al ser actualizadas o por inserción de nuevos datos, generan fragmentación de estos índices. Estas fragmentaciones conllevan a la pérdida de performance al acceder a ellas.

La instrucción DBCC DBREINDEX reorganiza el índice de una tabla o todos los índices definidos para una tabla. La reorganización de realiza dinámicamente sin necesidad de conocer la estructura de la misma o las restricciones que ella tenga. Por lo tanto no es necesario conocer si una tabla tiene clave primaria o si esta clave es única y además pertenece a algún índice, ya que la reorganización no necesita eliminar y recrear éstas restricciones para realizar su trabajo.

La sintaxis de esta instrucción es:

```
DBCC DBREINDEX
  (   'basededatos.dueño.nombre_de_tabla'
        [ , índice
           [ , fillfactor ]
```

]

) [WITH NO_INFOMSGS]

Fillfactor es el porcentaje de espacio de página destinado a ser ocupado. El valor definido reemplaza al que fue generado en el momento de la creación del índice. Si se quiere mantener el valor original, entonces se utiliza el valor 0.

WITH NO_INFOMSGS se suprimen los mensajes generados en la ejecución.

No es necesario conocer los nombres de todos los índices de todas las tablas, ya que si utilizamos la instrucción de la siguiente forma:

DBCC RBINDEX (Movimientos, ‘’, 0)

Se reorganizarán todos los índices que contengan la tabla Movimientos, conservándose el fillfactor original de cada índice en particular.

Una de las formas de utilizarlo es, escribir un script con una sentencia DBCC RBINDEX por cada tabla que necesitemos reorganizar y agendarlas en forma periódica mediante un trabajo de mantenimiento dentro de algún horario disponible.

Por lo tanto, la recomendación será: elegir las tablas más accedidas y/o actualizadas, y reorganizarlas una vez entre semana. Para reorganizar todas las tablas que contengan índices se utiliza el mismo concepto, pero dentro de un procedimiento que recorra todas la tablas de la base y las reorganice, sin necesidad que escribamos todas la tablas que contiene la base de datos. Estos procedimientos se pueden encontrar en el Forum bajo el nombre de Tips, y la idea es generar un trabajo de mantenimiento que se ejecute, por ejemplo, en el fin de semana

4.4.3 Reconstrucción de índices

Es importante periódicamente examinar y determinar qué índices son susceptibles de ser reconstruidos. Cuando un Índice está descompensado puede ser porque algunas partes de Éste han sido accedidas con mayor frecuencia que otras. Como resultado de este suceso podemos obtener problemas de contención

de disco o cuellos de botella en el sistema. Normalmente reconstruimos un Índice con el comando **ALTER INDEX**.

Es importante tener actualizadas las estadísticas de la base de datos. Para saber si las estadísticas se están lanzando correctamente podemos hacer una consulta sobre la tabla dba_indexes y ver el campo last_analyzed para observar cuando se ejecutaron sobre ese Índice las estadísticas.