irenchuchu
www.wuolah.com/student/irenchuchu

10852

Tema 1.pdf

Resumen

3° Administración de Bases de Datos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Tema 1: Nivel Interno.

1.1 Introducción.

SGBD: Sistema de software que almacena una cantidad de datos en diversos dispositivos que estructuran los datos en ficheros de distintos sistemas de archivos.

Campo: almacena un valor.

Registro: almacena un conjunto de campos.

Bloque: unidad mínima que se puede escribir en un disco. Almacena un conjunto de registros.

Fichero: almacena un conjunto de bloques.

Clúster: Conjunto de bloques agrupados en una misma vertical de un disco.

Pista: todo el circulo, todos los bloques que están a la misma distancia del centro.

Cilindro: conjunto de clústeres que están a la misma distancia del centro del disco (pista de arriba y pista de debajo de un disco).

Factor de bloqueo (Bfr): es el número de registros que caben en un bloque.

FAT: File Allocation Table.

Sistemas de Microsoft: Estructuras centralizadas.

Sistemas de Linux: Estructuras descentralizadas, inodos. Se puede recuperar la información ya que todos los inodos se conectan entre sí.

1.2 Medidas para evaluar un sistema de archivos.

Parámetro	Mide...
R	la memoria necesaria para almacenar un registro
T	el tiempo para encontrar un registro arbitrario
T_F	el tiempo para encontrar un registro por clave
T_W	el tiempo para escribir un registro cuando ya se tiene su posición
T_N	el tiempo para encontrar el siguiente registro a uno dado
T_I	el tiempo necesario para insertar un registro
T_U	el tiempo necesario para actualizar un registro
T_X	el tiempo necesario para leer el archivo
T_Y	el tiempo necesario para reorganizar el archivo


CUNEF
Lidera tu futuro.
Define tu éxito.
POSTGRADO EN
DATA SCIENCE
PARA FINANZAS
SÚMATE AL ÉXITO
Excelencia, futuro, éxito.
www.cunef.edu

Estos parámetros nos permiten saber el tiempo de operaciones.

1.3 Registros y bloques.

Un SGBD almacena la información en tablas. Una tabla es una estructura bidimensional que tiene anchura y altura, y está determinada por las columnas y la información que se almacenan en filas. Cada fila es un registro.

En el modelo E/R, horizontalmente se le denomina esquemas y verticalmente se le denomina instancias.

Un campo es un atributo. Un registro está compuesto por campos, y al conjunto de registros se le llama bloque. Una estructura es una secuencia de registros.

Un bloque es una unidad de información transferida por un dispositivo de almacenamiento masivo y almacenada en el área de trabajo de la memoria denominada buffer.

En el SGBD un bloque no pertenece a dos tablas u objetos relacionales, aunque no podemos llevarlo al extremo ya que existen los clústers (objeto relacional).

Siendo más concretos, un clúster es un agrupamiento de tablas con elementos en común que se ponen físicamente juntos, significa que los registros de las n tablas se almacenan en el mismo bloque porque no hay forma de separarlos ya que la estructura está súper entrelazada.

Un clúster es malísimo para acceder a una sola tabla y no a las demás, para ahorrar espacio degrega la información y si solo quieres una de las tablas y no las demás, tiene que volver a eliminar toda la información que no haga falta, luego solo están pensadas para consultarlas conjuntamente mediante la operación de reunión natural.

Tipos de registros:

- **Registro de longitud fija**: Tamaño fijo.
  - Se calcula lo que ocupa el registro sumando el tamaño de cada uno de los campos.

$$R = V_i$$

- **Registro de longitud variable**: Depende de la longitud del dato.
  - No se puede calcular lo que ocupa un registro, solo podemos estimarlo que suele ser el tamaño medio.
  - Básicamente, alguno de sus campos tiene un dato de tamaño variable tipo cadena de caracteres, large object, block … En nuestro caso hablaremos de registros de longitud variable con factor de bloqueo. Hay que tener en cuenta que para este tipo de registros es necesario almacenar con el registro la estructura del mismo, y para almacenar la estructura, por cada uno de los campos, se almacena un identificador del campo que nosotros pondremos como cadena de caracteres, un separador o delimitador, valor del campo y otro separador para separar un campo de otro, y, además, otro separador. Por cada campo estamos desperdiciando dos bytes.
  - Además, nos encontramos con que los registros, al ser de longitud variable, no podemos saltar una cantidad fija para llegar al siguiente, por ello el uso de delimitadores.

$$\mathbf{R = a'(A + V + s)}$$

a': número medio de atributos

A: longitud media de los nombres de los atributos

V: longitud media de los valores de los atributos

s: número de separadores por atributo

- **Registro de estructura homogénea**: Todos los registros tienen los mismos campos. No tiene nada que ver si es de longitud fija o variable, sino de la estructura.
- **Registro de estructura heterogénea**: Distintos campos o número de campos (clúster).

Como ordenar ficheros/registros/archivos … en un bloque depende de la aplicación.

El bloqueo es la forma de almacenar registros dentro de un bloque.

El factor de bloqueo es el número de registros que caben en un bloque que depende del tamaño del mismo bloque y del de los registros.

Métodos:

- **BLOQUEO FIJO:** Los bloques almacenan registros. Si hablamos de bloques que almacenan registros completos significa que, si no cabe un registro porque no hay espacio suficiente para ese registro, se dejará vacío.

    - Cálculo del factor de bloqueo entero para registros de longitud fija:

$$Bfr = \left\lfloor \frac{B-C}{R} \right\rfloor$$

    - Cálculo del factor de bloqueo entero para registros de longitud variable:

$$Bfr = \left\lfloor \frac{B-C}{R+M} \right\rfloor$$

M: Marcas/separar registros

- **BLOQUEO PARTIDO O ENCADENADO:** A veces ese espacio es demasiado grande así que elegimos que se parta ese registro y, por tanto, hay una parte de un registro en un bloque y otra parte en otro (encadenado porque indica cual es el siguiente registro con información, que puede estar consecutivo o no estarlo. Interesa coger otro bloque a la vez por parte de la cabeza lectora – escritora).
  El problema del bloqueo partido son las búsquedas secuenciales, ya que hay que ir leyendo dos bloques distintos, y la actualización de ficheros, porque los registros que estén separados, hay que unirlos para actualizarlos y volver a separarlos.
  La ventaja es que no se desperdicia espacio en bloques y es la única solución cuando el tamaño del registro es mayor al del bloque.

    - Cálculo del factor de bloqueo partido para registros de longitud variable:

$$Bfr = \left\lfloor \frac{B-P-C}{R+M} \right\rfloor$$

El espacio desperdiciado es aquel en el que no cabe un registro. El bloqueo fijo es más eficiente en registros pequeños y el bloqueo partido es más eficiente en registros grandes.

$$W = \frac{P + (Bfr \cdot M)}{Bfr} = \frac{P}{Bfr} + M$$

1.4 Organización de archivos y métodos de acceso:

En el caso de las Bases de Datos, se almacena la información en un sistema de ficheros y lo de debajo de la Base de Datos es el Sistema Operativo encargado de manejar los sistemas de ficheros en un gestor de disco (estructura lógica de una partición de un disco duro).

Un archivo es una unidad de información donde se acumulan todos los bytes que se refieren a la misma aplicación. Hay cuatro formas de estructurarlo y si no coincide con ninguna de ellas es porque se habrán combinado entre ellas.

- **Archivo secuencial físico (ASF):**

Los registros se almacenan por orden de llegada y no se puede alterar su posición.

Calculamos el tamaño de un registro de la siguiente forma:

$$R = a' \cdot (A + V + 2)$$

a': número medio de campos por registro
A: tamaño medio de los nombres de los atributos
V: tamaño medio de los valores
2: dos separadores necesarios

- **Recuperación de un registro:**

Tf es el tiempo necesario para encontrar un registro por un valor especifico en el ASF.

$$T_F = \sum_{i=1}^{n} \frac{i}{n} \cdot T = \frac{n+1}{2} \cdot T \approx \frac{n}{2} \cdot T$$

- **Siguiente registro (por clave):**

$$T_N = T_F$$

- **Inserción de registro:**

Los ficheros pueden abrirse en dos modos:

- Modo APEND: modo añadir, en el que se abre el fichero y pones el indicador del fichero al final y a partir de ahí se meten cosas nuevas.
- Modo SOBREESCRITURA: Abre el fichero y se pone al principio y ya se sabrá lo que hacer.
- Modo LECTURA: Se abre, pero no se toca. Se permite hacer copias.

$$T_I = T_W$$

Los registros se van almacenando por orden de llegada, sin poder alterar su posición.

El tiempo que se tarda en insertar es el tiempo que se tarda en escribirlo, y se escribe al final.

- **Actualizar un registro:**

Si el tamaño de registro no cambia:

$$T_A = T_F + T_W$$

Si el tamaño del registro cambia:

$$T_A = T_F + T_W + T_I$$

La diferencia entre ambos es que si el registro no cambia de tamaño, el tiempo es el que tardas en encontrarlo y en sobreescribirlo (marcarlo como borrado y quedarte con el nuevo). Pero si cambia de tamaño, es lo mismo que antes más el tiempo que tardas en insertarlo al final (Ti).

- **Lectura de fichero:**

Independientemente del contenido de registro: Se lee secuencialmente, no depende de ningún orden. Entonces, n elementos a leer por T (tiempo constante).

$$T_X = n \cdot T$$

Lectura ordenada según el valor de un atributo: Implica lo que tarda en encontrar cada uno de los registros (Tf) y lo multiplica por el número de registros, porque al final lo que quieres es recorrer el fichero entero y obtenerlo de forma ordenada así que lees n elementos.

$$T_X = n \cdot T_F$$

- **Reorganización de fichero:**

Si hemos ido actualizando los registros y hemos ido dejando huecos en medio, que además el sistema tiene que saltarse y si son registros de longitud variable tiene que saltarse lo que haya en medio hasta llegar al separador entre un registro y otro, eso es espacio sacrificado.

Cuando tienes un sistema de ficheros y cada uno tiene más de un 10% desperdiciado hay que hacer borrados para conseguir más espacio.

$$T_Y = (n + O) \cdot T + (n + O - d) \cdot T_W$$

O: registros añadidos desde la creación del fichero

d: número de registros a borrar

T: tiempo que tardas en leer un registro

Tw: tiempo que tardas en escribir un registro

El "+" indica que se crea otra estructura de datos para ir escribiendo en uno el contenido del otro, mirando si un registro vale o no vale en el caso de estar borrado.

Conclusión: (n+O)*T indican todos los registros incluidos los que están borrados. Y (n+O-d)*Tw indica que se quitan los registros borrados.

El ASF se suele usar para las copias de seguridad integral.

- **Archivo secuencial lógico (ASL):**

A todos los efectos, es un archivo secuencial en parte. Se compone de una parte ordenada que es la que está en los bloques en memoria. Y una parte desordenada o zona de desbordamiento, en la cual se aplica un algoritmo de ordenación (a medida que inserto los datos en el bloque). El algoritmo que se usa es el que desplaza todos los elementos en una posición e inserta el nuevo, hasta que se hayan llenado esos bloques de memoria y una vez llenos se hace de forma secuencial. Hay un algoritmo MERGESORT para mezclar ambas partes.

No se aplica en registros de longitud variable.

En esta organización de archivos, los registros se ordenan según una secuencia especifica que viene determinada por el contenido de un campo, la clave física (formada por un atributo o varios).

- **Espacio para un registro:**

$$R=\sum_i V_i$$

- **Recuperación de un registro**:

El valor de búsqueda no es clave: El tiempo que tarda es el tiempo en buscar un elemento en un archivo secuencial físico.

$$T_F=\frac{n}{2}\cdot T$$

Hay registros en la zona de desbordamiento: Hay que buscar en la zona ordenada (n) y en la desordenada (O).

$$T_F=\frac{n}{2}\cdot T+\frac{O}{2}\cdot T$$

Valor de búsqueda es clave física: búsqueda binaria.

$$T_F \cong \log_2(n)\cdot T$$

- **Siguiente registro:**

El siguiente valor de clave está en el propio fichero: Es el tiempo que se tarda en leer el siguiente registro.

$$T_N = T$$

El siguiente valor de clave está en el fichero de desbordamiento: Tiempo en leer el siguiente registro y descubrir que no es el que buscas, así que tienes que buscar en la zona de desbordamiento que es la que recorres secuencialmente.

$$T_N = T + O \cdot T$$

- **Inserción de un registro:**

A la hora de insertar debemos mantener el orden.

El procedimiento a seguir es:

- Localizar el punto de inserción.
- Escribir el registro.
- Leer y escribir los restantes.

$$T_I = T_F + \frac{n}{2} \cdot T + \frac{n}{2} \cdot T_W [+ T_W]$$

Si hay varios registros para insertar y hay fichero de desbordamiento, podemos insertar todos los nuevos registros en él.

$$T_I = T_W$$

Y después hay que insertarlos en el fichero maestro:

$$T_I = \frac{T_Y}{O}$$

Si se intentan insertar varios registros, si todos caben en el ordenado no vas a abrir huevos sino meterlos entre los ordenados. No trabajas con toda la parte ordenada sino con una parte entre limite y limite (hueco justo para insertar los registros a insertar). Si no caben en la zona ordenada no haces nada.

- **Actualizar un registro:**

Si se cambia el valor de la clave: lo encuentro, lo borro y lo inserto.

$$T_U = T_F + T_W + T_I$$

Si no se cambia el valor de la clave: lo encuentro y lo cambio.

$$T_U = T_F + T_W$$

Si no te permiten cambiar la clave es porque están ordenados y si cambias el valor de la clave tienes que reordenarlos. En reordenarlo tardas: lo que tardes en eliminar un registro (log en base 2 de n) y en insertarlo (log en base 2 de n) así que tardas log en base 2 al cuadrado y por tanto no se permite.

- **Lectura del fichero:**

El fichero ya ha de estar ordenado (independientemente de que tenga una parte ordenada y otra no, a ti te lo tienen que dar ordenado), y luego hacer mergesort para mezclar.

Solo se usa archivo maestro:

$$T_X = n \cdot T$$

Se usa desbordamiento (hay que ordenarlo) y leer en mergesort:

$$T_X = T_C(O) + (n+O) \cdot T$$

Tc: tiempo que se tarda en ordenar la zona de desbordamiento (registros que no han cabido en la zona ordenada)

*(n+0)*T: mergesort*

- **Reorganización de fichero:**

Después de una reorganización, la zona desordenada no existe, pero está completa. Desde esa perspectiva es la misma fórmula que se usa para recorrer ordenadamente el fichero, primero se ordena la parte desordenada y después se hace un mergesort entre la parte ordenada y la parte desordenada (que ya estaría ordenada). Para cada uno de los registros de cada una de las partes, se escribe en el nuevo fichero quitando registros vacíos (que pueden estar en la parte ordenada o en la desordenada, si la parte ordenada está completa, hay que irse a la zona de desbordamiento dando igual que haya huecos o no).

$$T_Y = T_C(O) + (n+O)\cdot T + (n+O-d)\cdot T_W$$

- **Archivo secuencial indexado (ASI):**

Es para todo tipo de registro.

Un archivo secuencial indexado tiene dos estructuras, una almacenada que se llama Fichero Maestro (ordenado) en el que se almacenan los registros en toda su extensión conforme van llegando, y se sabe cuál es su posición y a la vez que escribes el registro te quedas con su valor de clave y su posición, y esta pareja se incrusta donde le corresponda en el índice, que sería la otra estructura que compone un ASI.

El fichero de índice es un ASL con zona de desbordamiento. Al cerrar el fichero lo primero que hace el sistema es ordenar la zona desordenada del índice, hacer mergesort y entonces es cuando se cierra el fichero. Mientras esté en memoria el índice podrá estar desordenado salvo que quieras hacer una búsqueda, que entonces no tiene más remedio que ordenar.

ASI con INDICE DENSO:

En un índice denso estamos hablando de un índice que tiene una entrada clave-posición para cada registro del fichero maestro. Se pueden tener varias estructuras para ordenar la misma, es decir, índices secundarios. A mayor número de índices mayor tiempo tardarás en recorrer el fichero.

Por cada registro, sea de longitud fija o variable, no solo almacenas la información del registro, sino que además tienes que consumir una parte de otro fichero, es decir, en un índice denso cada registro lleva una representación en dos partes, en el archivo maestro que es el propio registro (R) y luego, en el índice tiene una pareja (clave-posición). A la hora de calcular el tamaño de un registro en el índice denso hay que tener en cuenta esa obligación.

La fórmula para el cálculo de tamaño de registro en un archivo de índice denso, es la suma del tamaño de los valores, más el tamaño del valor de clave (Vk), más la referencia a la posición (P) de disco que ocupa el registro en cuestión, porque el índice solo almacena eso.

$$R=\sum_i V_i+(V_k+P)$$

Al ser dos ficheros independientes, el maestro y el índice, se trabaja con dos tamaños de registro y dos factores de bloqueo distintos (uno para el maestro y otro para el índice); la sumatoria suele ser un número bastante grande y la otra suma un valor bastante pequeño (habitualmente).

- **Recuperación de registro:**

Como el índice está ordenado, o bien desde el principio o bien porque se ha tenido que ordenar si o si, el tiempo que tarda en leer el índice es: el tiempo que tarda en encontrar un elemento en el índice denso, luego en el peor de los casos se tarda log en base 2 de n multiplicado por el tiempo T más el tiempo de tener que leerlo en el fichero maestro.

$$T_F=\log_2(n)\cdot T+T$$

- **Recuperación del siguiente registro:**

Leer la siguiente posición a la que has buscado. Como el índice está ordenado se lee la siguiente entrada del índice que es la que está al lado, se lee la posición en esa entrada y se lee del fichero maestro el registro correspondiente a la entrada.

$$T_N=T+T=2\cdot T$$

- **Inserción de registro:**

Mismas operaciones que para la inserción en un fichero secuencial, pero hay que actualizar también el índice, que se hace con inserción directa al encontrar la posición después de leer y abrir hueco.

$$TI=T_{I1}+T_{I2}=2\cdot\left(T_F+\frac{n}{2}\cdot T+\frac{n}{2}\cdot T_W[+T_W]\right)$$

Si hay zona de desbordamiento en el fichero maestro:

$$T_{I2} = T_F + \frac{n}{2} \cdot T + \frac{n}{2} \cdot T_W [+T_W]$$

$$TI = T_W + T_{I2}$$

- **Actualización del registro:**

Un registro se puede actualizar, en cualquier caso, si no se cambia el valor de clave: tiempo que se tarda en encontrarlo más el tiempo que se tarda en sobreescribirlo.

$$T_U = T_F + T_W$$

Si se cambia el valor de la clave es mejor eliminarlo y volver a escribirlo (insertarlo), y si está en la zona de desbordamiento no hay que hacer nada, lo único que afecta es al índice, lo cual se encuentra la posición y se cambia. Si está en la zona ordenada, se altera el funcionamiento, por tanto, lo marcas y lo escribes de nuevo en la zona de desbordamiento.

$$T_U = 2 \cdot (T_F + T_W) + T_{I1} + T_{I2}$$

- **Lectura de fichero:**

Por índice principal: lo mismo que en el archivo secuencial lógico.

$$T_X = n \cdot T$$

Por índice secundario: Es el tiempo en leer todos los elementos, después reordenarlos y luego leerlos del registro.

$$T_X = n \cdot T + n \cdot T = 2 \cdot n \cdot T$$

Por índice secundario y desbordamiento:

$$T_X = (n+O) \cdot T + n \cdot T + O \cdot T = 2 \cdot (n+O) \cdot T$$

Funciona únicamente con registros de longitud fija, ya que el algoritmo que se usa es el de búsqueda binaria.

- **Reorganización del fichero:**

Tenemos el fichero maestro con la parte ordenada y la parte desordenada, ordenas la desordenada y haces mergesort creando así un nuevo dichero con un nuevo índice sobre la fusión de ambas partes. Se aprovecha para escribir el fichero maestro y el índice simultáneamente.

$$T_Y = T_C(O) + (n+O)\cdot T + (n+O-d)\cdot T_w + (n+O-d)\cdot T_w$$

El **problema** que tienen los índices densos es que son demasiado grandes para memoria. Cuando el tamaño crece es un problema mantener un índice ordenado, porque llega un momento en el que va todo secuencialmente y no cabe más. Eso quiere decir que cada cierto tiempo el sistema cuando comprueba como funciona el índice denso decide que ya no es una buena opción. En cuyo caso lo que hace es, en vez de almacenar una entrada por cada registro, almacena la posición del bloque en la que está el registro a buscar. De forma que un bloque se entiende como un conjunto de registros y por ello, habría que almacenar solo el valor de la clave de uno de los representantes de ese bloque. Un representante que sirva para encontrar el registro, que suele ser el menor de ellos.

ASI con INDICE NO DENSO:

El índice no denso solo puede definirse sobre la clave física, es decir, solo puede haber un índice no denso por fichero.

Cuando se busca en el índice lo que buscas es la página en la que está el registro, no la posición.

La ventaja es que cuando cargas el bloque, el bloque ya está en memoria con lo cual es rápido, solo recorres el bloque.

El índice no denso está formado por parejas de valores de clave y posición de bloques, no de registros. Y además no todos los valores de clave están representados, con lo cual el tamaño es mucho menor.

El número de entradas de un índice se calcula como:

$$\frac{n}{Bfr}$$

Número de registros del maestro / El factor de bloqueo se usa el del maestro.

El proceso de búsqueda en un índice no denso: si buscas una clave específica, el no encontrarla en el índice no quiere decir que no esté. Se encuentra la página y secuencialmente se recorre ya que está ordenado.

En el caso de un índice denso, para saber si un registro está o no está no es necesario mirar el fichero maestro. Pero en el caso de un índice no denso, hay que mirarlo sí o sí, excepto si fuese el representante del bloque que ya se sabría con mirar el índice.

Los índices no densos son bastante útiles en el sentido de cuando nos encontramos con registros muy grandes, hay veces que no tenemos posibilidad.

Porque digamos que cada registro como ocupa varios bloques en el fichero maestro, el índice no denso acaba convirtiéndose en un índice denso porque los registros son tan grandes que no entran dos en el mismo bloque.

Diferencias con el índice denso:

- Procedimiento de búsqueda en no denso:
    - Se selecciona en el índice la clave inmediatamente inferior a la buscada.
    - Se carga el bloque del maestro.
    - Se busca secuencialmente.
- Si no está, hay que acceder al maestro. En un denso no es necesario.

Algunas de las características del índice no denso (ya dichas anteriormente):

- En cada página del maestro se suele dejar un espacio para nuevos registros.
- En maestros con fichero de desbordamiento, cada registro tiene un enlace al siguiente en el fichero de desbordamiento. En una página, hay un enlace a registro de desbordamiento (si la página está completa). En el fichero de desbordamiento, un registro puede tener enlace al siguiente.

ASI MULTINIVEL:

Se crea un índice no denso sobre un índice denso.

Cuando tienes índices que te permiten acelerar construyendo otros índices sobre el propio índice.

Empeoran las inserciones ya que esta vez tienes que insertar en dos índices.

El tamaño de la página del índice:

$$y = \left\lfloor \frac{B-C}{V+P} \right\rfloor$$

Tamaño útil del bloque (B-C) se divide entre la entrada del índice (V+P).

Cuando se habla de un ASI multinivel se está hablando de que todos los niveles son no densos. Yo fijo el tamaño máximo de un índice, al sobrepasarse se construye otro sobre él. Con esto mejoramos la eficiencia.

Nos encontramos con una estructura de árbol, pirámide que a partir de cada índice me lleva a otro índice. El nivel 1 es el que está lo más cerca posible del fichero maestro.

- **Espacio de registro:**

$$r = P + \sum_i V_i$$

El índice de primer nivel tiene una entrada por bloque del maestro, luego el número de entradas es: el número de registros del maestro entre el factor de bloqueo de éste:

$$i_1 = \left\lceil \frac{n}{Bfr} \right\rceil$$

Los niveles superiores tienen su propio número de bloques y su propio número de entradas con lo cual, se calcula el número de entradas del resto de niveles de esta forma:

$$i_k = \left\lceil \frac{i_{k-1}}{y} \right\rceil$$

Y el número de bloques:

$$b_k = \left\lceil \frac{i_k}{y} \right\rceil = i_{k+1}$$

Espacio necesario para índices:

$$I = (b_1 + b_2 + \ldots + b_m) \cdot B$$

Espacio medio necesario por registro:

$$R = r + \frac{O}{n} \cdot r + \frac{I}{n}$$

- **Recuperación de registro:**

$$T_F = T_M + (m-1) \cdot T_{F_i} + T_F^{'} + T_{F\,Cadena}$$

- **Siguiente registro:**

No se suele cambiar de página, por tanto, básicamente suele consistir en leer el registro que hay al lado.

$$T_N \approx T$$

- **Insertar registro:**

Se tiene que insertar en el fichero maestro y queda espacio en el bloque:

- Siempre se procura tener espacio para mantener próximos aquellos registros que tengan los mismos valores de clave, de forma que, si hay espacio para insertar el registro, encontrar página y mirar si hay espacio y meter nuevo valor.

$$T_I \approx T_F + \frac{1}{2} \cdot Bfr \cdot (T + T_W) + T_W$$

Se tiene que insertar en el fichero maestro y no queda espacio en el bloque:

- Si no nos cabe, lo que hacemos es crear otro bloque en otra posición, y redistribuir los valores del bloque que no nos permite insertar de nuevo el valor. De esta forma estamos consiguiendo otra entrada para el índice. Mantengo espacio en cada bloque, si me caben 5 registros en un bloque solo puedo meter 4.

$$T_I \approx T_F + \frac{1}{2} \cdot Bfr \cdot (T + T_W) + 2 \cdot T_W$$

Se tiene que insertar en el fichero de desbordamiento:

$$T_I \approx T_F + 2 \cdot T_W$$

- **Actualización de registro:**

Si no cambia el valor de la clave: Se busca en todos los niveles del índice, hasta llegar al maestro.

$$T_U = T_F + T_W$$

Si cambia el valor de la clave: Tiempo de encontrarlo/escribirlo/insertarlo. Ósea, se borra y se inserta, es más eficiente que moverlo.

$$T_U = T_F + T_W + T_I$$

- **Lectura exhaustiva de fichero:**

Lectura exhaustiva significa que no tiene porqué ser ordenada. Se leen los registros tal y como aparecen.

$$T_X \approx (n + O) \cdot T$$

- **Reorganización de fichero:**

Hay que conseguir reordenarlo todo. Lo que se tarda en leer de forma ordenada todo el fichero más lo que se tarda en escribir los registros ordenados teniendo en cuenta que hay algunos que pueden estar marcados para borrar (d) más lo que se tarda en construir cada uno de los k índices que tenemos. Si teníamos un fichero de índices multinivel, tenemos que volverlo a construir reordenado.

$$T_Y = T_X + (n + O - d) \cdot T_W + k \cdot I$$

Cuantos factores de bloqueo: el del maestro más el de cada nivel.

Criterios para selección de índices:

- Solo son útiles si hay consultas que los utilicen.
- Los atributos candidatos son los involucrados en la cláusula WHERE (cuando se ven afectados por la igualdad o el uso de rangos) o aquellos atributos de reunión.
- Si el número de tuplas con valores repetidos para un campo supera el 5%, no debería crearse un índice sobre ese campo.
- Los campos de un índice multiatributo deben ordenarse según las consultas más frecuentes.
- No se deben indexar campos actualizados frecuentemente.

- **Ordenación por Acceso Directo (AAD):** Tabla hash (a partir de un valor de clave obtengo una posición relativa con respecto al comienzo del fichero que estás manejando).

Para cada valor de clave se consigue una posición de disco, lo que llamamos un HASH. Con el valor de clave sabes su posición en el disco y no necesitas ninguna estructura adicional.

De qué orden es la búsqueda en un archivo hash teórico: constante. Lo que tardas en aplicar la función, que básicamente es una operación sobre el dato. Función hash más tonta que hay para hash es el módulo: coger tamaño y dividir entre 100.

Posiciones de una tabla hash: cubetas/cubos(vector, lista enlazada…).

Para resolver el hash: una solución sería el doble hash, hash cíclico (vas buscando posiciones y cuando encuentras uno vacío o borrado lo que se hace es escribir en esa posición), rehashing (volver a colocar los elementos de un hash original en otro hash, reestructurar los elementos).

Supón un campo clave de tamaño V (si es numérico hay 10^v valores posibles; si es alfabético, hay 26^v valores posibles).

El fichero tendrá m celdas para n datos, incluyendo espacios libres, por tanto m>=n.

Si una clave tiene b símbolos distintos, entonces hay b^v posibles claves, que cumplirá b^v>>m. El algoritmo transforma cada clave en un valor entre 0 y m-1.

Puesto que b^v>>m:

- Se dan colisiones, es decir, hay dos claves con la misma posición.
- Hay huecos: ninguna clave da una posición.

Resolución de colisiones:

- Direccionamiento cerrado: el espacio de posiciones en un único archivo.
  - Búsqueda lineal: una colisión se almacena en una posición libre del mismo bloque (secuencial en búsqueda de registro)
  - Realeatorización: una colisión se almacena en otra posición mediante otra transformación de clave, y sucesivamente.
- Direccionamiento abierto: el espacio de posiciones en más de un fichero (maestro y desbordamiento):
  - Listas enlazadas: colisiones en fichero de desbordamiento (estructura ASI).
  - Bloques de desbordamiento: colisiones en bloques del fichero de desbordamiento.
- Hashing dinámico lo que hace es que en función del comportamiento en las colisiones puede aplicar modificaciones en la función hash. Solo ocurre cuando la función es de tipo modulo, significa que cuando hay muchas colisiones es capaz de cambiar el espacio de almacenamiento y reestructurar el contenido cambiando la función hash, es decir, cambia tamaño del bloque.

  - **Espacio de registro:**

Implica que, si cada registro tiene una referencia, eso representa que hay desbordamiento. Es decir, todo hash viene con resolución de conflictos.

$$r = P + \sum_i V_i$$

$$S_F = m \cdot r + c \cdot r = (m+c) \cdot r$$

$$R = \frac{S_F}{n} = \frac{(m+c)}{n} \cdot r$$

  - **Recuperación de registro:**

$$T_F = C + T + p \cdot T_{F\_Cadena}$$

C: tiempo que se tarda en aplicar la función hash.

T: tiempo en leer un registro.

Tf_cadena: tiempo que se tarda en buscar una cadena de desbordamiento

p: probabilidad de colisión

- **Inserción de un registro:**

Tiempo que tardas en calcular su posición y el tiempo que se tarda en escribir dos veces.

$$T_I = C + 2 \cdot T_W$$

- **Actualización de un registro:**

Tiempo que se tarda en encontrarlo más en escribirlo.

$$T_U = T_F + T_W$$

- **Lectura exhaustiva de fichero:**

Leerse la tabla y luego la zona de desbordamiento haciendo valor absoluto.

$$T_X \simeq (m + c) \cdot T$$

- **Reorganización de fichero:**

Tiempo que tarda en hacer una lectura de cada uno de ellos, pero tengo que leer todo el fichero y meterlo en una nueva estructura, así que al final tienes la sumatoria de la inserción de cada uno de los elementos.

$$T_Y = T_X + T_{Carga}$$

$$T_{Carga} = \sum_{i=1}^{n} T_I(i)$$

Es el rehash, que consiste en aplicar una segunda función hash. Conclusión se hace una lectura exhaustiva escribiendo en un nuevo fichero.

Evaluación del sistema:

- Muchos de estos parámetros pueden estimarse, pero mejor estimar que ir a ciegas.

Estimación de uso del sistema:

- Carga por demandas de almacenamiento:
    - Se ve afectada por: número de registros, número total de atributos, número medio de atributos por registro, longitud de campos, longitud de identificadores de atributos.
    - Carga por recuperación de información: se calcula teniendo en cuenta el número de solicitudes a un archivo en un conjunto de transacciones en un tiempo dado.
    - Carga por actualización: se calcula teniendo en cuenta el número de actualizaciones a la base de datos en base a la frecuencia de creación de registro, de actualización de registro, de eliminación y de ampliación.

Análisis de beneficios:

- Basado en la probabilidad de las operaciones.
- Estimación de beneficios:
    - Generalmente basados en factores económicos.
    - Se tienen en cuenta los gastos de personal.
    - Tiempos de respuesta bajos centran al personal. Se calculan en base a:
        - Tiempo de solicitud.
        - Tiempo de computación.
        - Tiempo de presentación de resultados.
- En términos de uso del sistema:
    - En general:
        - Tiempo medio para plantear una consulta.
        - Tiempo medio para describir una salida de datos.
        - Tiempo medio de procesamiento de operación.
        - Retraso medio en presentación de resultados.
    - Hay que tener en cuenta el tipo de usuarios, porque incluye un costo adicional en el manejo.
- Mantenimiento de los datos:
    - El mayor coste está asociado a la actualización.
    - Eso conlleva un coste de detección y corrección de errores:
        - Antes de actualización: difícil y costoso.
        - Durante la actualización: puede llevar a acciones no deseadas e incrementos de costes en el sistema.
        - Después de actualización: hay que rastrear el alcance y es mejor no cambiarlo.

Necesidades de almacenamiento:

- Ante cambios frecuentes, será necesario prever más espacio reservado.
- La densidad mide la proporción de espacio ocupado (y depende del tipo de archivo):

$$D = \sum_{F_i} n_i \cdot \frac{R+W}{\mu_{D_i}}$$

- Componentes:
  - Discos,
  - Controladoras,
  - Canales…
  - En general, mejor una sola unidad mayor que varias más pequeñas.
- Distribución:
  - Varios procesadores y un dispositivo de almacenamiento.

Costes de explotación:

- Para parámetros conocidos de carga y efectividad de archivos, podemos escoger.
- Primer enfoque: modelar cada diseño físico y seleccionar.
  - Es costoso.
- Segundo enfoque: reducir posibilidades. Primero, el hardware adecuado. Después el diseño físico para el uso estimado dados los recursos existentes.
  - Estimar la demanda.
  - Estimar recursos.
  - Comparar para encontrar una combinación satisfactoria.
  - Si no acoplan, ambas estimaciones se ajustan de forma más precisa.
  - Si la discordancia es importante, cambiar el equipo o el diseño físico y empezar de nuevo.

Costes de explotación:

- Procesos compartidos: existe competencia.
  - A nivel de procesador: afecta al tiempo de respuesta.
  - A nivel de disco: provoca recolocación constante del cabezal, lo que afecta al registro siguiente y a la lectura exhaustiva.
  - La programación concurrente permite bloquear un dispositivo hasta que una tarea concluya o ésta lo libere.

Análisis coste/beneficio.

- Beneficios:
  - Cantidad de datos disponibles.
  - Intensidad de uso de la BD.
- Costes:
  - Almacenamiento (inicial + aplicaciones).
  - Procesamiento (escalabilidad de discos, controladoras o canales).