Tema 2. Organizaciones de ficheros y estructuras de acceso

Ficheros y Bases de Datos

10 de junio de 2002

1. Introducción

Este documento contiene preguntas del tema 2 recogidas por estudiantes de la asignatura (M.Carmen Vilar Blasco, Yolanda Biosca Vázquez, Sergi Navarro Galán, Claudio Sarrio Llopis, Oscar Masip Carceller, Antonio Ruben Vega Castilla, Joan Manuel Ferrer i Peris, Belen Bru Ubeda y Gemma Bermudez Jovani) y que han sido contestadas por el profesorado.

2. Preguntas y respuestas

2.1. ¿Qué es un índice no denso?

Primero debemos saber qué es un índice denso: es un índice que tiene una entrada por cada registro del fichero de datos. Si el fichero de datos tiene, por ejemplo, 10.000 registros, en el índice hay 10.000 registros, es decir, una entrada por cada uno de ellos. Un índice no denso es un índice que no tiene una entrada por cada registro del fichero de datos, sino que tiene menos entradas. Dependiendo del tipo de fichero y del tipo de índice, puede tener una entrada por cada valor distinto del campo de indexación o bien una entrada por cada bloque del fichero de datos (en un bloque caben, normalmente, varios registros de datos).

2.2. ¿Qué es un campo clave?

Es un campo cuyos valores no se repiten en el fichero, es decir, no hay dos registros en el fichero que tengan el mismo valor en ese campo.

2.3. ¿Cuántas entradas tiene el índice?

Cada tipo de índice tiene un número de entradas. Un índice primario es un índice no denso que tiene una entrada por cada bloque del fichero de datos. Un índice de agrupamiento es un índice no denso que tiene una entrada por cada valor distinto del campo de indexación.

En cuanto a los índices secundarios, el número de entradas depende de cómo sea el campo de indexación. Si es un campo clave, el índice es denso. Si es un campo no clave, hay varias alternativas: puede ser un índice denso, puede ser un índice no denso con entradas de longitud variable o puede ser un índice no denso con entradas de longitud fija y un nivel extra de indirección.

2.4. ¿Por qué en los índices primarios es mejor poner en las entradas el valor del campo de indexación del último registro de cada bloque?

La bibliografía dice que el algoritmo que realiza la búsqueda es ligeramente más eficiente si se hace de ese modo, aunque no muestra el algoritmo. Lo buscaremos.

2.5. Sobre un fichero desordenado ¿qué tipo de índices puedo definir?

Se pueden definir índices secundarios y cualquier tipo de índice multinivel.

2.6. Sobre un mismo fichero ¿puedo tener un índice primario y uno de agrupamiento a la vez?

Imposible. Estos índices son los que se construyen sobre el campo de ordenación de un fichero ordenado. Cuando el campo de ordenación es un campo clave, al índice se le llama "primario"; cuando el campo de ordenación es un campo no clave, al índice se le llama "de agrupamiento".

2.7. ¿Qué significa que los índices secundarios proporcionan un ordenamiento lógico del fichero?

Los índices secundarios se definen sobre campos a través de los que el fichero de datos no está ordenado. Si en un fichero queremos encontrar, por ejemplo, los registros cuyo valor del campo de indexación se encuentra entre x e y, y no disponemos de ningún índice, debemos hacer una búsqueda lineal recorriendo todos los registros del fichero para encontrar aquellos que cumplen la condición. Esto se debe a que el fichero no está ordenado a través de ese campo.

Si sobre ese mismo fichero y sobre ese campo de búsqueda se define un índice secundario, haciendo una búsqueda binaria en el índice se pueden encontrar los punteros a los registros que cumplen la condición de búsqueda e ir a por ellos directamente desde el índice. El acceso es más rápido porque el índice funciona "como si" el fichero estuviera ordenado a través de ese campo, aunque no lo está. Por eso decimos que proporciona un ordenamiento lógico, porque lo podemos manejar como si estuviera ordenado aunque físicamente no lo está.

2.8. ¿Puede haber índices primarios y secundarios en un mismo fichero? ¿E índices de agrupamiento y secundarios?

Sí, claro. Sobre un fichero ordenado por campo clave podemos definir un índice primario sobre el campo de ordenación e índices secundarios sobre los otros campos. Sobre un fichero ordenado por campo no clave podemos definir un índice de agrupamiento sobre el campo de ordenación e índices secundarios sobre los otros campos.

2.9. En un índice secundario ¿se pueden realizar búsquedas binarias?

Sí, claro. Un índice secundario es un índice de un solo nivel. Estos índices son ficheros ordenados y sobre un fichero ordenado podemos realizar búsquedas binarias.

2.10. ¿Qué es un índice de agrupamiento?

Es un índice que está definido sobre el campo de ordenación de un fichero ordenado siendo éste un campo no clave.

2.11. En los índices de un solo nivel ¿por qué se pueden realizar búsquedas binarias?

Porque un índice de un solo nivel es un fichero ordenado y sobre un fichero ordenado la búsqueda binaria es, en general, más rápida que la búsqueda lineal.

2.12. ¿Qué ocurre cuando los índices son demasiado grandes y cambian con frecuencia?

Un índice de un sólo nivel es un fichero ordenado, el índice cambia si lo hace el fichero de datos al que va asociado y más concretamente si estos cambios afectan al campo de ordenación del índice. En estos casos hay que reorganizar el índice, lo cual supone un elevado coste y es un problema que se agrava si el índice es grande.

Se puede solucionar utilizando zonas de desborde. De esta forma el índice sólo se reorganiza cada cierto tiempo. Otra posibilidad consiste en dejar espacio libre a priori en los bloques del fichero de índice de forma que, mientras haya espacio en los bloques, se puedan hacer las inserciones sin tener que reorganizar todo el fichero, obviamente esto supone el tener que utilizar más espacio del necesario para almacenar el fichero.

2.13. Enumera ventajas/desventajas de realizar la búsqueda binaria sobre ficheros ordenados.

No es cuestión de ventajas o desventajas, es una cuestión de costes de algoritmos. Si tenemos un fichero ordenado con n registros y hacemos una búsqueda por un campo clave, en el caso medio recorreremos la mitad del fichero hasta encontrarlo, por lo que el coste es $n/2$. Si el campo no es clave, habrá que seguir leyendo registros mientras cumplan la condición de búsqueda. Esto no es muy costoso si tenemos en cuenta que estos registros están uno a continuación del otro porque hablamos de un fichero ordenado.

Si sobre ese fichero ordenado hacemos una búsqueda binaria y el campo de ordenación es un campo clave, en el caso medio nos cuesta $\log_2 n$. Si el campo de ordenación es no clave, seguiremos leyendo registros consecutivos como en el caso anterior.

Si comparamos los costes de ambos algoritmos, vemos claramente que el de búsqueda binaria es más rápido: $\log_2 n << n/2$.

2.14. ¿En qué circunstancia es aconsejable utilizar el agrupamiento (clustering)?

Cuando se tienen varios ficheros a los que se accede juntos con frecuencia a través de las relaciones lógicas que guardan. Es más fácil de entender si pensamos en un ejemplo: cuando se accede al fichero de líneas de facturas se accede también al fichero de facturas para concatenar cada una con sus líneas. Si los registros de ambos ficheros se almacenan agrupados en un solo fichero, el acceso es más rápido porque los registros que están relacionados se almacenan físicamente juntos: cada cabecera de factura va seguida en el fichero por sus líneas de factura, de modo que si caben en un bloque, con un solo acceso nos traemos la factura completa.

2.15. ¿Qué restricciones tienen los árboles B para tener mejores prestaciones que los árboles de búsqueda?

Los árboles B siempre están equilibrados, es decir, tienen todas sus hojas al mismo nivel (eso evita que haya hojas muy profundas y otras poco profundas) y los nodos están, al menos, al 50 % de su capacidad (si se aprovecha bien el espacio en cada nodo, el número de niveles del árbol es menor que si los nodos están casi vacíos).

2.16. ¿Qué sucede si se añade en un árbol B un registro repetido? Así, en el ejemplo visto en clase ¿qué ocurriría al añadir otro 10?

Cuando el campo sobre el que se define el índice es no clave, lo que se suele utilizar es un nivel extra de indirección. Los punteros de las entradas del índice no apuntan a un registro de datos sino que apuntan a un bloque en donde se encuentran todos los punteros a los registros de datos que tienen el mismo valor en el campo de indexación.

2.17. En un árbol B+ de orden p, en el mejor de los casos ¿a qué puntero se llega? ¿y en el peor?

En los árboles B+ los punteros a los registros de datos se encuentran sólo en las hojas, por lo que en todos los casos hay que llegar hasta ellas para encontrar el puntero al registro de datos, esté o no esté. El número de accesos a nodos del árbol es siempre igual al número de niveles que tiene el árbol.

2.18. ¿Por qué en los árboles B se suman las entradas para saber el número total de entradas y en los árboles B+ no?

Pues porque en el árbol B hay punteros a registros de datos en todos los nodos, desde la raíz hasta las hojas. Para saber cuántas entradas hay en total, tenemos que sumar las entradas que hay en cada nivel. Sin embargo, en los árboles B+ los punteros a registros de datos sólo están en las hojas, por lo que sólo hay que mirar el número de entradas del último nivel, el nivel más bajo.

2.19. ¿Qué tipo de accesos se pueden hacer en un árbol B? ¿ Y en un árbol B+?

Los árboles B y B+ los utilizamos como índices, lo que nos permite acelerar el acceso a los datos frente a una búsqueda lineal. Si tenemos un índice con estructura de árbol B, el acceso se hace siempre desde la raíz y se van bajando niveles hasta encontrar el puntero a los datos (o hasta llegar a no encontrarlo, claro).

Si el índice tiene estructura de árbol B+, podemos acceder a través de la raíz y bajar hasta las hojas hasta encontrar el puntero a los datos. O bien podemos acceder directamente a la primera de las hojas y recorrerlas todas ya que están encadenadas según el orden del campo de indexación (este tipo de acceso se realiza cuando la condición de búsqueda es una desigualdad: $A < x, A \leq x, A > x, A \geq x$). O bien podemos acceder desde la raíz a una de las hojas y después recorrer las hojas que están encadenadas (este tipo de acceso se realiza cuando la condición de búsqueda es un rango, por ejemplo $x \leq A \leq y$)

2.20. ¿De qué modo se buscaría un registro en un fichero utilizando una función de dispersión como índice?

Se aplica la función de dispersión $f(x)$ al campo de dispersión A. El resultado $f(A)$ indica una entrada en un tabla de dispersión o directorio. En la entrada correspondiente del directorio encontramos un puntero al índice; este índice es un fichero disperso. En el índice ya encontramos una entrada (si existe) que apunta al registro de datos que cumple la condición de búsqueda, que siempre es la igualdad, no hay que olvidarlo.

En este caso, lo que es disperso es el índice, no el fichero de datos. Cuando se inserta un registro de datos (por ejemplo, si es un fichero desordenado, se añadirá al final), la entrada que apunta a ese registro se sitúa en el índice en la posición que indique la función de dispersión.

2.21. Diferencia entre "estructura de un fichero", "organización de un fichero" y "estructura de acceso".

En la asignatura son términos equivalentes los de "estructura de un fichero" y "organización de un fichero". Se trata de saber cómo están organizados los registros dentro del fichero, si son de longitud fija o variable, etc. Una "estructura de acceso" es un fichero adicional que utilizamos para facilitar el acceso a los datos de otro fichero.

Si tenemos un libro de cuentos, su estructura u organización es lo que describimos a continuación: "los cuentos están ordenados por temas: cuentos fantásticos, historias de animales, ogros y brujas, etc.; para cada cuento, hay una introducción donde se relatan sus orígenes y una indicación sobre la edad mínima para la que está recomendado, su duración media en minutos, el lugar donde se desarrolla y sus principales personajes". Las estructuras de acceso que nos permiten encontrar un cuento más rápidamente que si vamos recorriendo el libro página a página, serían las siguientes:

- un índice que sigue el orden de los cuentos tal y como aparecen en el libro, es decir, por temáticas;
- un índice de cuentos por personajes (cada entrada es un personaje y apunta a los cuentos en donde éste sale);
- un índice de cuentos por su duración (cada entrada es un número de minutos y apunta a los cuentos que tienen esa duración media);
- un índice de cuentos por edades.

2.22. Diferencia entre clusters y bloques. ¿Cuántos bloques hay en un cluster?

Un *cluster* es un conjunto de bloques que están contiguos físicamente. Cuando se asigna espacio a un fichero, se asigna una cantidad de *clusters*. Si hay que añadir espacio a un fichero, se añade al menos un *cluster*, no es posible añadir una cantidad más pequeña. Este concepto es importante porque resalta la importancia de que los bloques que forman un mismo fichero es conveniente que estén próximos físicamente para que el acceso a los mismos sea más rápido. El gestor de ficheros del sistema operativo ve al fichero como un conjunto de *clusters*, dentro de cada *cluster* los bloques están uno a continuación del otro.

El número de bloques que hay en un *cluster* puede estar fijado por el sistema operativo, lo puede escoger el administrador del sistema o incluso se puede fijar para cada aplicación según sus propios requisitos.

2.23. Los árboles B+ al guardar menos punteros y tener mayor capacidad que los árboles B ¿qué les ocurre?

Pues que se ponen muy contentos, porque en los nodos les caben más punteros a nodos del árbol (se ahorran los punteros a datos, que sólo están en las hojas) y por lo tanto son más anchos (o regordetes) y, en consecuencia, tienen menos niveles. Además, permiten un modo de acceso adicional gracias a que las hojas están enlazadas.

2.24. Ventajas de los árboles B+ sobre los B

Ver pregunta 2.23.

2.25. ¿Qué problemas presentan los árboles?

La tala indiscriminada y sin control :-(

Bromas aparte, los árboles como estructuras de acceso adicional a ficheros de datos, suponen una mejora frente a los índices de un solo nivel, ya que mejoran el aprovechamiento del espacio utilizado, garantizando un acceso al bloque de datos que contiene el registro buscado de forma eficiente. El precio que hay que pagar es que los algoritmos de inserción, borrado y modificación son algo más complejos.

2.26. ¿Qué tipos de ficheros son los más rápidos y eficientes para realizar una búsqueda? ¿Por qué?

En realidad el fichero no es más o menos rápido o eficiente. Lo que puede ser eficiente es un método de acceso: es más eficiente hacer una búsqueda binaria en un fichero ordenado, que hacer una búsqueda lineal. Por otra parte, se puede mejorar también el tiempo de acceso (que éste sea más rápido) o bien cambiando la organización del fichero, o bien construyendo estructuras de acceso adicionales (índices) que permitan encontrar los datos más fácilmente ante determinados tipos de acceso.

Por lo tanto, ante una determinada organización de un fichero, habrá un método de acceso que sea el más eficiente para cada tipo de consulta. Por ejemplo, si se tiene un fichero de personas, ordenadas por el DNI, cuando se quiere consultar los datos de una persona con un determinado DNI lo más eficiente es hacer una búsqueda binaria. Si la búsqueda se realiza sabiendo los apellidos de la persona y no su DNI, la búsqueda se hará mediante búsqueda lineal. Si el fichero de personas es desordenado, no nos podemos beneficiar del algoritmo de búsqueda binaria ante ningún tipo de consulta. Si el fichero de personas es un fichero disperso y se ha utilizado como campo de dispersión el DNI, buscar una determinada persona por su DNI es más eficiente si se hace mediante la función de dispersión. Pero si la búsqueda de la persona se hace a partir de sus apellidos, de nuevo se debe hacer una búsqueda lineal.

Cada organización de ficheros favorece un tipo de consulta, o ninguno. Sin embargo, los usuarios van a querer acceder a los datos mediante distintos tipos de consulta. Aquellos accesos que vayan a ser lentos pueden hacerse más rápidamente si se añaden al fichero índices sobre los campos de la consulta (los campos que aparecen en el WHERE de una consulta SQL, por ejemplo). Estos índices se pueden implementar mediante distintas estructuras: ficheros ordenados, varios niveles de ficheros ordenados, árboles y ficheros dispersos. Cada una de estas estructuras tiene sus ventajas y sus inconvenientes.

Un último apunte respecto a todo esto es que no siempre el acceso a los datos a través de un índice va a ser más rápido frente a una búsqueda lineal. La bibliografía dice que si en una consulta se accede a más de un 15-20 % de los registros de un fichero, será más rápido acceder directamente al fichero mediante una búsqueda lineal[1].

2.27. ¿Qué ocurre si un fichero completo no cabe en memoria principal?

Un fichero normalmente no va a estar completamente en memoria principal. El programa que trabaja con un fichero pedirá el sistema operativo bloques del mismo para trabajar con ellos.

En el caso en que el número de bloques pedidos no cupiera en memoria principal del ordenador, el sistema operativo, utilizando la memoria virtual, realizará el trasiego de páginas

[1] Esto es en el caso general en el que el fichero y el índice no están ordenados por el mismo campo.

entre memoria principal y disco para que los datos con los que se necesite trabajar estén en memoria principal. Obviamente, esto va a suponer un aumento considerable en los tiempos de acceso a los datos, pero por problemas ajenos al sistema de ficheros y a la forma en que los datos están almacenados en el fichero.

2.28. ¿Qué tipos de ficheros se suelen utilizar en los SGBD?

Los SGBD actuales suelen ser capaces de trabajar con muchos de los tipos de ficheros de datos que hemos visto en clase (ordenados, desordenados, dispersos, ...) así como distintas estructuras de acceso (índices de un solo nivel, árboles, ...).

El administrador de la base de datos puede indicarle al SGBD qué tipo de fichero ha de utilizar para almacenar las distintas tablas (en el modelo relacional) o los tipos de índices de un solo nivel o árboles que ha de definir sobre los ficheros de datos. Por este motivo es necesario conocerlos a fondo y saber cuales son sus ventajas e inconvenientes para, en función de las consultas que se vayan a realizar sobre la base de datos, definir los tipos de ficheros y estructuras de acceso que optimicen (o al menos mejoren) dichas consultas.

2.29. ¿Qué ocurre si los registros de un mismo fichero son de diferentes tipos?

Esto ocurre cuando se decide realizar un agrupamiento (o "clustering") de dos ficheros de datos de distinto tipo de registros (o de dos tablas en el modelo relacional) en un solo fichero. Con ello se obtienen una serie de ventajas (ver pregunta 2.14) como es la mejora del tiempo de acceso de determinadas consultas. También se tienen inconvenientes como es que empeora otros accesos (por ejemplo las búsquedas de ventas de un determinado artículo en el caso de haber realizado la agrupación comentada en el ejemplo de la pregunta 2.14). Otro inconveniente es que se trata de ficheros (de alguna manera) ordenados con todos los inconvenientes que ello conlleva.

2.30. Cita los objetivos a tener en cuenta a la hora del diseñar una buena estructura de ficheros

El objetivo que se debe tener siempre presente es reducir los accesos a disco ya que éstos son tremendamente costosos y suponen el cuello de botella a la hora de acceder a la información. Debemos tratar de que la búsqueda de los datos que se piden, se realice con el menor número de accesos (uno, si es posible) a disco.

2.31. ¿Cuál es la cantidad mínima que se puede leer o escribir en disco?

Desde el punto de vista del sistema operativo: el "cluster". Desde el punto de vista de los SGBD: el bloque. (Ver pregunta 2.22)

2.32. Enumera los 4 tipos de estructuras según el orden de almacenamiento de los registros en los ficheros

- Ficheros desordenados: los registros no siguen ningún orden dentro del fichero. Normalmente se escriben tal y como se van creando los datos.
- Ficheros ordenados: los registros se ordenan dentro del fichero según uno de sus campos.
- Ficheros dispersos: Se utiliza una función que, aplicada sobre un campo del registro, indica el bloque del fichero donde se debe almacenar dicho registro.
- Ficheros agrupados: Son ficheros en los que se guardan registros de distintos tipos. Esto sirve para mejorar el acceso a los datos, en 2.14 y 2.29 se puede ampliar este concepto.

2.33. Explica los ficheros hashing, y sus tipos a nivel externo

Los Ficheros dispersos (o "hashing") se caracterizan porque se puede averiguar el bloque que contiene un determinado dato aplicando una función de dispersión a uno de sus campos (denominado campo de dispersión).

La principal ventaja de este tipo de ficheros es que la dirección del bloque de cada registro se obtiene directamente aplicando la función de dispersión, por lo tanto, el acceso es rápido (un sólo acceso) si se busca por el campo de dispersión.

Aunque la técnica de dispersión también se puede utilizar a nivel interno (RAM) como una estructura de datos de un programa, su principal utilidad para nosotros es su uso a nivel externo, es decir, aplicada a ficheros.

Existes distintos tipos de dispersión cuando se trabaja con ficheros:

- Estática: el fichero tiene un tamaño fijo y predefinido a priori. Todos los registros con un mismo valor valor para la función de dispersión se almacenan en el mismo bloque. Si un bloque se llena se utilizan zonas (bloques) de desborde, en la pregunta 2.44 se explica su funcionamiento en su uso en memoria principal.
- Dinámica: Se utiliza un directorio en forma de árbol que, mediante los bits más significativos del resultado de la función de dispersión, permite acceder a los bloques del registro. El árbol, y por lo tanto el número de bloques que forman el fichero, puede crecer y disminuir dinámicamente.

- Extensible: El fichero, al igual que antes, puede cambiar de tamaño, pero en este caso se utiliza un vector como directorio. Los bits más significativos del resultado de la función de dispersión se utilizan para acceder a una posición del vector donde está almacenado el bloque del fichero que contiene el registro buscado.
- Lineal: En este caso, a grandes rasgos, cuando hay que agrandar el fichero se van añadiendo bloques consecutivos. Se utilizan varias funciones de dispersión para acceder a los bloques.

2.34. Explica ventajas e inconvenientes del Agrupamiento (clustering)

Ver preguntas 2.14 y 2.29.

2.35. ¿Es aconsejable definir un índice sobre todos y cada uno de los campos de los registros de un fichero?

No, aunque se pueden definir tantos índices como campos[2] tengan los registros de un fichero, el mantenimiento de los mismos (actualización, reorganizació, ...), cuando se realizan modificaciones sobre el fichero de datos, puede cargar en exceso dichas operaciones.

De nuevo, el administrador de la base de datos, haciendo las pruebas oportunas, debe decidir qué índices se deben crear para mejorar el acceso a los datos.

2.36. ¿Por qué los árboles-B se utilizan como índices?

Porque garantizan que el tiempo de acceso a los datos a través del índice es bastante uniforme sin penalizar demasiado la reorganización del árbol cuando se añaden o modifican datos en el fichero de datos.

2.37. Cita una ventaja de los árboles B+ frente a los árboles B

Hay varias, por ejemplo permiten el acceso directo a los datos y también en el orden que marca el índice, mientras que en el caso de los árboles B este último tipo de acceso es más costoso.

También permiten crear árboles más anchos lo cual redunda en una mejora en el tiempo de acceso, ya que para una misma cantidad de datos el árbol B+ tendrá menos niveles (que el B) y por tanto se accederá al fichero de datos con menos accesos.

[2] E incluso se pueden definir índices sobre varios campos de un fichero.

2.38. Diferencia principal entre los índices primarios e índices de agrupamiento

Ambos son índices que se crean sobre el campo de ordenación de un fichero ordenado.

Sin embargo, este fichero ordenado puede estarlo a través de un campo clave (por ejemplo, el fichero de clientes ordenado por su código: no hay dos clientes que tengan el mismo código) o puede estar ordenado por un campo no clave (por ejemplo, el fichero de facturas ordenado por la fecha: hay varias facturas en una misma fecha).

Cuando el fichero está ordenado por un campo clave, al índice que se define sobre este campo se le llama índice primario.

Cuando el fichero está ordenado por un campo no clave, al índice que se define sobre este campo se le llama índice de agrupamiento.

2.39. ¿Podríamos tener un índice primario de múltiples niveles y que fuera no denso?

Aquí se han mezclado varios conceptos. En primer lugar, un índice primario es, por definición, un índice de un solo nivel, por lo tanto no puede ser de múltiples niveles.

Por otra parte, un índice primario (y un índice de agrupamiento) será siempre un índice no denso ya que no necesita una entrada por cada registro del fichero de datos: aprovechando que el fichero de datos está ordenado y que el índice va a seguir el mismo orden, con tener una entrada en el índice por cada bloque de datos, se tiene información suficiente en el índice para realizar accesos más rápidos a través de él, frente a realizar estos accesos mediante búsqueda binaria sobre el fichero de datos.

2.40. Problemas que plantea la inserción en índices de agrupamiento

Al ser ficheros ordenados físicamente, tenemos problemas al insertar.

Para evitarlos, la inserción en los índices de agrupamiento se puede hacer reservando un bloque completo por cada valor del campo de agrupamiento. Todos los registros con ese valor se colocan en el bloque. En el caso de que necesitáramos más espacio para almacenar los registros con un valor determinado, se asignan y enlazan bloques adicionales. Como casi siempre, esta mejora se produce a costa de ocupar mayor espacio en disco del necesario.

2.41. Explicar la técnica de "stripping" para resolver el problema del cuello de botella del disco

El "stripping" consiste en dividir un fichero en varios discos (cuantos más mejor). Al escribir o leer información del fichero, cada disco puede recibir o mandar la parte del fichero que tiene almacenada. Así se multiplica la velocidad de transferencia por el número de discos.

Esta técnica es cara porque el coste de almacenamiento de datos también se multiplica.

2.42. ¿Cómo modificar un dato en un fichero desordenado?

Los pasos a seguir son los siguientes:

- Hay que buscar el dato, que por estar el fichero desordenado, ha de ser una búsqueda lineal, muy costosa,
- se lleva a memoria RAM todo el bloque y se modifica el registro. Si el tamaño es variable y no cabe en el espacio que antes ocupaba, se borra y se inserta al final,
- se escribe el bloque en el disco ya modificado. Si el registro es de tamaño variable se escribe el registro con el dato borrado y se inserta el nuevo registro al final.

2.43. ¿Cómo insertar en un fichero ordenado?

Los pasos a seguir son los siguientes:

1. Se busca en qué posición ha de ir el nuevo registro y se lleva el bloque a memoria principal,
2. se hace hueco para almacenar el nuevo dato, esto seguramente supondrá leer y escribir todos los bloques por detrás del actual en el fichero para desplazar todos los registros hacia atrás,
3. se escribe el bloque en disco.

Si se ha dejado previamente espacio en los bloques del fichero, el segundo paso no tiene porque suponer el desplazamiento de todos los registros posteriores al que se está haciendo la inserción.

El paso segundo también se puede aliviar utilizando un fichero de desbordamiento: Si al insertar un dato no cabe, se escribe en el fichero de desbordamiento, que es un fichero desordenado, y cada cierto tiempo se fusionan el fichero ordenado y el de desborde. Por supuesto, esto introduce una pérdida de eficiencia en las búsquedas .

2.44. Describir el método de encadenamiento para la resolución de colisiones en la dispersión interna

Con el encadenamiento se enlazan los sinónimos, es decir, los registros que tienen un mismo valor para la función de dispersión.

Se añade un puntero a cada registro. Ese puntero señala la posición en el vector en la que hay almacenado un sinónimo.

Si un registro se ha de almacenar en la misma posición que otro (el 3 por ejemplo), se pone en una posición libre de la zona de desborde (K por ejemplo) y en el puntero del registro (el 3 en el ejemplo) se pone en qué posición se ha almacenado (K en el ejemplo).

Si otro registro se ha de almacenar en esa misma posición, se busca otra posición libre y se guarda dicha posición en el segundo puntero (el que estaba en K en el ejemplo).

Si queremos acceder a los datos, la función de dispersión nos indicará en qué posición del vector debería estar. Si no está en esa posición, pero el puntero de desborde contiene un valor, se accede a dicha posición. Este proceso se repite hasta encontrar el dato buscado o haber recorrido toda la lista encadenada.

2.45. ¿Qué es el Agrupamiento ("Clustering")?

Las preguntas 2.14 y 2.29 contestan a esta pregunta.