Resumen Clase 5

Archivos Secuenciales: Otro tipo de archivos donde cambia la organización y mejoramos la eficiencia dejando de lado la eficiencia lineal y empezando a trabajar con eficiencia logarítmicas.

En la búsqueda de información siempre debemos minimizar el numero de accesos.

Busqueda secuencial : Poco eficiente. n/2, eficiencia lineal

Busqueda Binaria : Eficiente, pero es costoso porque se debe tener el archivo ordenado ,Log2n

Introduccion a índice:

Vamos a tener en una estructura auxiliar, como es el índice de un archivo, información que me diga el tema y el lugar donde encontrar. En nuestro caso vamos a tener la clave de ese índice y el lugar donde esa clave está.

Ejemplo: En el índice tengo el legajo y el NRR del registro, el índice va a estar ordenado por numero de legajo, por lo cual buscar por numero de legajo va a ser eficiente, una vez que ya encuentro el numero de legajo, en ese numero de legajo tengo asociado el NRR y luego para acceder hago un seek hacia esa posición.

Ahora, si tenemos que buscar libros en una biblioteca, podemos encontrarlo por 3 criterios:

* Por Autor
* Por Titulo
* Por Categoria

Si tuviéramos que plantear este problema pero utilizando un archivo, como haríamos?

UN archivo puede estar ordenado solo por UN criterio. En este caso, puede estar ordenado por titulo, por autor o por categoría. Sino debería tener n copias de ese archivo y ordenarlo cada uno por sus n criterios (ineficiente)

Una solución a este problema, es usar un catalogo de tarjetas, que seria un conjunto de 3 indices, cada uno con una clave distinto, pero todos tienen el mismo numero de catalogo como campo de referencia.

Lo anterior podría resolverse teniendo tantos índices como criterios de ordenación (en este caso serian 3 indices) y un archivo.

Un archivo de datos puede tener asociados tantos índices (criterios de busqueda) como el problema lo requiera.

La búsqueda por índices en un archivo es una técnica utilizada para mejorar la eficiencia en la recuperación de datos, especialmente en archivos grandes. Consiste en crear estructuras de datos adicionales que almacenan información sobre la ubicación de los registros en el archivo, basándose en uno o mas criterios de búsqueda.

Indices Definicion:

Herramienta: Encuentra registros en un archivo de una forma mas eficiente. Consiste de un campo de llave (busqueda) y un campo de referencia que indica donde encontrar el registro deltro del archivo de datos.

Es una tabla:recibe información acerca de ciertos valores como entrada (LLAVE) y provee como salida donde está lo que se está buscando dentro del archivo.

Estructura de datos: Tiene clave, dirección, sobre la tabla se hace búsqueda dicotómicas y una vez que se encontró el elemento se accede a donde está el registro.

En un índice para acceder a un archivo yo tengo la clave, y el NRR (numero relativo de registro) donde ese elemento está, si el registro es de longitud fija.

Si el registro es de longitud variable, en vez del NRR, vamos a tener la distancia en bytes, un ejemplo es que debemos acceder al byte 143 por ejemplo, que ahí se encuentra el primer byte del dato que estoy buscando.

Los índices se pueden implementar tanto en registro de longitud fija como en registro de longitud variable.

La estructura mas simple para el índice es un árbol, porque el árbol tiene búsqueda binaria, por lo cual están ordenados y tienen eficiencia logarítmica para encontrar información.

Una característica de usar un árbol es que permite imponer un orden sin que el archivo se reacomode.

Nosotros en nuestro archivo, los registros los vamos a agregar al final, lo que vamos a tener que hacer es construir un índice que tenga esa estructura de árbol binario, y que permita ver los datos desordenados como si estuviesen ordenados.

Entonces, de esta manera, en el archivo tendré datos desordenados, pero tendré índices, que me permitan trabajar sobre el archivo de manera ordenada.

Por mas de que tengamos un archivo ordeanado, si los registros son de longitud variable, no podemos aplicar una búsqueda binaria directamente sobre el archivo, porque no tenemos la certeza de donde se encuentra cada dato. Una solución a priori es usar a parte del archivo de longitud variable, una estructura que me permita almacenar los índices de manera ordenada.

Esta estructura esta ordenada a partir de la siguiente forma canonica: cia+numero de identificación. Esta estructura, aparte de tener la llave, va a almacenar el byte donde se encuentra el dato en el archivo. (Estructura de la pagina 10). La estructura del índice va a ser ordenada con registros de longitud fija, porque la llave tiene 12bytes y la referencia 4 bytes

Entonces cada campo del índice va a ocupar SI O SI 16 bytes. Por lo tanto, al tener los índices en una estructura con registros de longitud fija, podemos aplicarle una búsqueda binaria

Si estamos buscando un dato, en este caso el índice esta ordenado por cia+id , entonces seria eficiente buscarlo por ese criterio. Una vez que encontramos la llave en la estructura, vamos a tener asociado el byte donde se encuentra en el archivo, de esta manera, hacemos un seek hacia esa posición de memoria, para acceder al dato.

Modificaciones/Actualizacion con registros de longitud variable:

Si no se modifica la clave:

* Si el registro no cambia de longitud, se almacena en la misma posición física, y el índice no se toca.
* Si el registro cambia de longitud (se agranda), se debe reubicar en el archivo de datos. Reubicar el dato en el archivo, también implica guardar la nueva posición inicial en el índice.

Si el registro es de longitud fija, nos evitamos estos problemas anteriores.

Si se modifica la clave:

* Se debe modificar el archivo de datos.
* Se debe actualizar y REORGANIZAR el archivo de índices.
* Como simplificar? Modificar clave = Eliminar + Agregar.

Ventajas de utilizar Indices:

-En todo momento, tener un índice, al trabajar con registros de longitud fija, y ser una estructura ordenada, nos va a permitir realizar una búsqueda binaria. Por lo cual encontrar la información se logra con una performance logarítmica, Log2 (en base 2)

-Si se puede almacenar ese índice en memoria principal, esa búsqueda logarítmica, se logra hacer sobre una estructura en ram,por lo cual es muy eficiente.

-El mantenimiento es menos costoso, mantener el índice actualizado es muy eficiente

Desventajas:

-Un problema es si el índice no entra en memoria RAM. El tema es que un índice va a entrar si o si en memoria ram, pero si tenemos N índices, no sabemos si podemos almacenarlos todos en ram

-El problema gravísimo es la persistencia de datos, un ejemplo es que se corte la luz mientras esté trabajando con los índices, y no los haya actualizado, todos los índices se van a perder. Por lo cual habría que reconstruir los índices, y es bastante costoso.

Persistencia de datos:

* Una alternativa para almacenar los índices, es almacenarlos en memoria secundaria, porque si se corta la luz, no se van a perder los índices, tampoco tendremos limitaciones de memoria, como pasaría en RAM. Ganamos persistencia, pero vamos a tener una diferencia en performance, van a seguir siendo búsquedas logarítmicas, pero no es lo mismo hacer una lectura en disco, que en RAM.

Tener índices en arboles binarios es eficiente, pero los arboles son estructuras dinamicas, las cuales se almacenan en memoria RAM, pero los arboles se pueden implementar sobre disco sin mucha complejidad. La clase que viene vamos a ver eso.

De esta manera, vamos a lograr persistencia, y despreocuparnos del tamaño del índice porque lo tenemos en disco, la eficiencia sigue siendo log2 (en base dos) pero al no usar mas la ram, dejamos de trabajar con nanosegundos, y en memoria secundaria empezamos a trabajar en milisegundos, hay una diferencia de eficiencia.

Otra opción es que siempre sabemos donde vamos a guardar cada registro,y usamos el método de dispersión o hashing, y lo buscamos en ese lugar, pero en este caso no estaríamos trabajando con índices. Esta solución permite encontrar registros con un acceso a disco siempre, pero no utiliza índices. Esto lo vemos mas adelante.

Claves univocas: NO repiten información, el elemento es único y lo distinguimos por su clave univoca.

Clave secundaria: Permite que un elemento, aparezca mas de una vez en el archivo

En el ejemplo de la diapositiva 10 de la clase 5, es mas natural buscar los archivos por el compositor, que buscarlo por clave univoca, otro ejemplo es la búsqueda de alumnos en un archivo, es mas natural que sepamos el nombre de la persona y no el legajo, ese nombre de persona es una clave secundaria, se repite, o puede repetirse.

La construcción de un índice con repeticiones no es igual a la construcción de un indicie sin repeticiones

El uso de las claves secundarias es mas ineficiente que las claves univocas.

Un índice secundario relaciona la llave secundaria con la llave primaria.

Indice univoco=> NO tiene repeticiones

Indice secundario => Puede tener repeticiones.

Una clave primaria es una clave univoca que fue elegida como primaria.

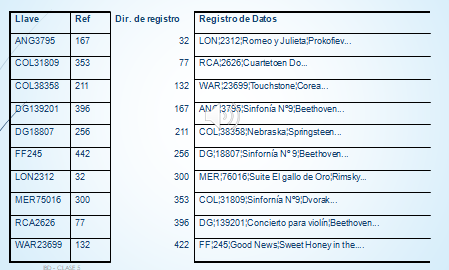
Ejemplo, en un archivo de alumnos tenemos dos claves univocas: Dni y Legajo, debemos elegir una de ellas. Ambas identifican a un alumno de manera única.

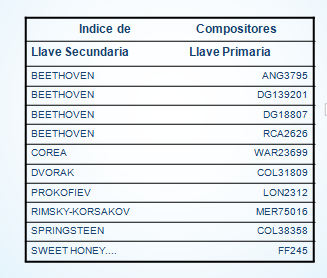
La clave secundaria NO va a tener dirección física del archivo de datos. Una clave secundaria no tiene asociada la posición donde el elemento esta en el archivo de datos.

Una clave secundaria tiene asociada la clave primaria, podemos acceder al dato a través de la clave primaria.

Esto es porque si se modifica el archivo de datos, solo altera un índice, que es el índice primario, no altera los demás índices, de esta manera, es mas eficiente.

Ejemplo de índice Primario:



Ejemplo de índice Secundario con repeticiones:   


En el ejemplo anterior, tenemos 4 BEETHOVEN, pero estaría mal, tenemos que llegar a una forma de compactar esos 4 BEETHOVEN en un solo registro, con 4 referencias en este caso,

Podemos compactarlo en un vector que tenga clave + vector de punteros con ocurrencias, pero entre las desventajas están que, el vector es una estructura dinámica, por lo cual es de tamaño fijo, puede haber casos que sea insuficiente , o casos donde sobre espacio y provoque fragmentación interna.

Ejemplo de índice secundario sin repeticiones:

Vamos a tener dos estructuras:

(Estructura izquierda) Archivo de índice secundario

Elmacenemos de forma única a cada compositor, asociado a un NRR propio, y asociado a otro NRR de otra estructura.

(Estructura derecha) Archivo de listas de llaves primarias

Tenemos asociado un NRR, la clave primaria, y a la derecha de todo tenemos asociado el NRR del siguiente registro del mismo compositor, es decir, de esta manera podemos tener acceso a los registros repetidos, a través de una segunda estructura con llaves primarias.

Ejemplo: Vemos que BEETHOVEN tenia 4 registros, en la estructura del índice secundario (estructura izquierda) aparece solo una vez, y ese 3 que esta a la derecha, ,es una referencia al NRR del archivo de listas de llaves primarias.

Entonces accedemos al NRR 3, el cual tiene, la llave primaria, y a su derecha un 8, ese 8 es otro registro del mismo compositor, y asi de esta manera podemos ir accediendo a registros el mismo compositor, sin tener elementos repetidos, solo tenemos elementos referenciados. Si su referencia es -1, quiere decir que no tiene otra referencia asociado al mismo índice secundario.



Un ejemplo de índices secundarios, es usar listas invertidas:   
