# Base de Datos

* Es una colección de datos relacionados.
* Colección de archivos diseñados para servir a múltiples aplicaciones
* Un dato representa hechos conocidos que pueden registrarse y que tienen un resultado implícito.

Para que estos datos persistan, se deben guardar en una memoria secundaria.

¿Qué es una Base de Datos?

- Una colección de datos interrelacionados.

Un dato es cualquier información dispuesta de manera adecuada para su tratamiento con computadora. La idea de conjunto expresa que son muchos. “Interrelación” indica que los datos no son aislados, que están relacionados entre sí y conforman un todo. Sin embargo, esa definición no expresa una característica fundamental de las BD, que es para qué esos datos se almacenan y relacionan:

*Una BD es una colección de datos interrelacionados con un propósito específico, vinculado a la resolución de un problema del mundo real.*

Propiedades implícitas de una BD Una Base de datos:

- Representa algunos aspectos del mundo real, a veces denominado Universo de Discurso.

- Es una colección coherente de datos con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD. O sea, los datos deben tener cierta lógica y no ser datos aleatorios.

- Se diseña, construye y completa de datos para un propósito específico. Está destinada a un grupo de usuarios concretos y tiene algunas aplicaciones preconcebidas en las cuales están interesados los usuarios.

- Está sustentada físicamente en archivos en dispositivos de almacenamiento persistente de datos (no volátil, como un disco rígido).

# Archivo

* Colección de registros guardados en almacenamiento secundario
* Colección de datos almacenados en dispositivos secundarios de memoria
* Colección de registros que abarcan entidades con un aspecto común y originadas para algún propósito particular

**Memoria RAM**: El tiempo de acceso es de micro o nano segundos, pero es muy costosa, volátil, y limitada.

**Memoria secundaria**: El tiempo de acceso es de milisegundos.

Los datos que necesitan ser preservados más allá de la ejecución de un algoritmo, deben ser almacenados en archivos. Para que esto suceda, un archivo no debe estar almacenado en memoria RAM, sino en algún dispositivo de memoria secundaria. Las estructuras de datos, por lo general, son definidas y utilizadas sobre la memoria RAM y tienen duración mientras el programa se encuentra activo. Por esta razón, cuando es requerido tener persistencia en la información que maneja un algoritmo, es necesario almacenarla en archivos en el disco. Es responsabilidad del SO encargarse de cuestiones referidas al lugar de almacenamiento del archivo en disco, cómo se ubica la información en este y cómo será recuperada. Entonces, un archivo es una estructura de datos homogénea. Se caracterizan por el crecimiento y las modificaciones que se efectúan sobre ellos. Crecimiento se refiere a agregar nuevos elementos, mientras que modificaciones hace referencia a alterar la información ya existente o eliminar

Los archivos son manejados con algoritmos y éstos, no referencian al lugar físico donde se encuentra el archivo, sino que tienen una visión lógica de éste, lo utiliza como si estuviera almacenado en memoria RAM. Por eso se debe establecer una conexión con el SO.

- Archivo físico: es el archivo propiamente dicho, aquel que reside en un dispositivo de almacenamiento secundario, y es manejado por el SO (ubicación y operaciones disponibles).

- Archivo lógico: es el archivo utilizado por el algoritmo. Cuando un algoritmo necesita operar con un archivo hace una conexión con el SO ya que éste es el responsable de la administración de archivos. Dicha acción se denomina “Independencia física”.

Organización de archivos

Los archivos pueden analizarse desde diversas perspectivas. Pueden clasificarse según la información que contienen, por la estructura de dicha información, o por la organización física del archivo en memoria secundaria.

- Secuencia de bytes.

No se puede determinar con facilidad el comienzo y fin de cada dato. Ej. archivos de texto. No hay estructura, una palabra se termina por un conjunto de caracteres que termina con blanco, pero eso es una “convención”

- Estructurados (Registros y campos):

Pueden ser de longitud fija o variable.

Campo: Un campo es la unidad más pequeña, lógicamente significativa de un archivo. Con longitud fija puede generarse desperdicio de espacio. Con longitud variable, puede existir un indicador de longitud al principio de cada campo, o delimitador al final de cada campo (carácter especial no usado como dato). El problema de esta última opción, es que aquel carácter especial es un acuerdo que debe ser conocido por todos aquellos que interactúen con la BD.

Registro: Un registro es una colección de campos agrupados, que define un elemento del archivo. Pueden ser de longitud predecible/fija (en cant. de bytes o cant. de campos), cuyos campo a su vez puede ser también fijos o variables.

También, pueden ser de longitud variable, y para ello se utilizan distintas “técnicas” para determinar el tamaño de cada dato para poder ser leído correctamente:

- indicador de longitud: al comienzo, indica la cant. de bytes que contiene, que deberán ser leídos.

- segundo archivo: mantiene la información de la dirección del byte de inicio de cada registro

- delimitador: carácter especial no usado como dato (mismo problema que en delimitador de campos).

Registros longitud fija 🡪 hay desperdicio de espacio, pero son más fáciles de trabajar.

Registros longitud variable 🡪 utiliza de mejor manera el espacio que hay en el disco, pero los algoritmos son más complicados.

Acceso a los archivos

El acceso a los datos de un archivo puede ser:

- Secuencial físico: acceso a los registros del archivo uno tras otro, en el orden físico en el cual están almacenados.

-Secuencial indizado (lógico): el acceso a los elementos de un archivo se hace teniendo en cuenta algún tipo de organización previa, esto implica que no se toma en cuenta el orden físico (Un índice de un libro puede ser un ejemplo de la vida real).

- Directo: Se accede a un registro determinado del archivo en un solo acceso sin necesidad de acceder a los que lo preceden, en este tipo de acceso no importa que exista un orden físico o lógico predeterminado.

Tipos de archivos

* Según la forma de acceso se clasifican en:

- Serie: cada registro es accesible luego de procesar aquel que lo antecede, es decir, aquellos cuya forma de acceso es secuencial física. No tienen orden 🡪 (archivos) O(n).

- Secuencial: los registros son accedidos en el orden de alguna clave. Forma de acceso secuencial indizada (lógica) 🡪 (arboles) O(log2(n)).

- Directo: los registros pueden ser accedidos directamente. Forma de acceso directo 🡪 (hash) O(1).

* Según cantidad (#) de cambios se clasifican en:

- Estáticos: Pocos cambios. Puede actualizarse en procesamiento por lotes. No necesita de estructuras adicionales para agilizar los cambios

- Volátiles: Sometido a operaciones frecuentes (agregar, borrar, actualizar). Su organización debe facilitar cambios rápidos. Necesita estructuras adicionales para mejorar los tiempos de acceso

Buffers

Los buffers son memoria intermedia (ubicados en RAM) entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados definitivamente en memoria secundaria, o donde residan una vez recuperados de la memoria secundaria. Los buffers se utilizan por cuestiones de performance 🡪 menos accesos a disco.

Unidad de transferencia 🡪 Es ineficiente transferir datos muy pequeños (de a 1 byte por ejemplo), debido al costo que implica el intercambio de información con el disco (es mecánico, tarda mucho). Por esta razón se transfieren al buffer sectores enteros (conjunto de registros), entre los cuales se encuentran los datos deseados. Entonces, hay que organizar la información de tal manera que aquellos datos del sector que se transfirieron “demás” también sean útiles.

Cantidad de accesos < cantidad de lecturas.

Algorítmia clásica sobre archivos

Se dividen en dos niveles:

- Físico: lo que respecta al almacenamiento secundario.

- Lógico: dentro del programa:

* Leer(read) -> Read(nom\_lógico,variable)
* Escribir(write) -> Write(nom\_lógico,variable)
* Definir(assign) -> Assign(nom\_lógico,nom\_físico)
* Crear un archivo (solo escritura) -> Rewrite (nom\_lógico)
* Abre archivo (lectura/escritura)-> Reset (nom\_lógico)
* Cerrar archivo -> Close(nom\_lógico). Esta instrucción indica que no se va a trabajar más con el archivo. Significa poner una marca de EOF (end of file) al final del mismo.

Las operaciones read y write, leen o escriben sobre los buffers relacionados a los archivos. No se realizan directamente sobre el disco rígido.

Nombre lógico representa una variable de tipo archivo sobre la que se realizó la asignación y la variable (de read y write) debe ser del mismo tipo que los elementos que se declaran como parte del archivo.

Siempre que se hace la operación read o write, el puntero avanza

Operaciones adicionales

* EOF(nom\_lógico);(Función) -> Retorna verdadero o falso, cuando el puntero este señalando el último archivo.
* Fin de archivo
* ¿Cómo trabaja? -> Hay que preguntar primero, si estoy en EOF y leo EXPLOTA
* FileSize(nom\_lógico);(Función) -> Retorna valor numérico, cantidad de registros que tiene el archivo
* Tamaño del archivo
* FileSize(nom\_lógico);(Función) -> Retorna la posición actual del puntero del archivo.
* Seek(nom\_lógico, pos);(Procedimiento) -> Genera un acceso directo de un archivo (entre 0 y n-1)
* Ir a una posición del archivo
* La posición se cuenta siempre desde el comienzo del archivo
* El primer lugar es el 0

Algoritmia: Básica o Clásica

* Básica: Agregar nuevos elementos.
* Clásica: Actualización, merge y corte de control.

Concepto de archivo maestro y archivo detalle:

🡪 Maestro: archivo que resume información sobre un dominio de problema específico. 🡪Detalle: archivo que contiene movimientos realizados sobre la información almacenada en el maestro.

Generalmente hay un maestro y N detalles.

La algorítmica clásica sobre archivos se resumen en tres tipos: actualización, merge y corte de control.

* La actualización se dá entre archivo maestro con archivo detalle, tiene su primer variante en el caso donde exista un solo archivo detalle o el caso donde existan N archivos detalle donde N > 1. Las variantes dependen de las precondiciones. Se puede actualizar tanto un archivos ordenados o no.
* El merge consiste en la generación de un archivo maestro (que no existe) el cuál tiene el resumen de la información de uno o varios archivos existentes.
* El corte de control es el proceso mediante el cuál la información de un archivo se presenta de forma organizada con un formato especial.

Viaje del byte

Hace referencia a el proceso que implica el intercambio de datos con algún dispositivo

de almacenamiento secundario.

* Cuando se busca un dato, se espera encontrarlo en el primer intento (o en pocos)
* Si se buscan varios datos, se espera obtenerlos todos de una sola vez

Existen 2 archivos paralelamente:

-Archivo físico: es aquel que existe en el almacenamiento secundario. Es el archivo tal y como lo conoce el SO, y se encuentra en su directorio de archivos.

- Archivo lógico: Es el archivo visto por el programa. Permite a un programa describir las operaciones a efectuarse en un archivo. No se sabe cuál archivo físico real se utiliza o donde está ubicado.

El viaje del byte no es sencillo, abarca ciertos ciclos para escribir.

¿Quiénes están involucrados cuando se escribe un dato en un archivo?

- Administrador de archivos: conjunto de programas del S.O. (capas de procedimientos) que tratan aspectos relacionados con archivos y dispositivos de E/S.

* Capas superiores se encargan de aspectos lógicos de datos (tabla), establecer si las características del archivo son compatibles con la operación deseada.
* Capas Inferiores se tratan aspectos físicos (FAT), determinar donde se guarda el dato (cilíndro, superficie, sector). Si el sector está ubicado en RAM se utiliza, caso contrario, debe traerse previamente

- Buffer de E/S: agilizan la E/S de datos. Manejar buffers implica trabajar con grandes grupos de datos en RAM , para reducir el acceso a almacenamiento secundario.

- Procesador de E/S: dispositivo utilizado para la transmisión desde o hacia almacenamiento externo. Independiente de la CPU.

- Controlador de disco (driver): encargado de controlar la operación de disco: colocarse en la pista, colocarse en el sector, transferencia a disco.

Capas del protocolo de transmisión de un byte

* El Programa pide al **S.O**. escribir el contenido de una variable en un archivo
* El **S.O.** transfiere el trabajo al **Administrador de archivos**
* El **Adm**. busca el archivo en su tabla de archivos y verifica las características
* El **Adm**. obtiene de la FAT la ubicación física del sector del archivo donde se guardará el byte.
* El **Adm** se asegura que el sector del archivo está en un **buffer** y graba el dato donde va dentro del sector en el **buffer**
* El **Adm.** de archivos da instruccciones al **procesador de E/S** (donde está el byte en RAM y en que parte del disco deberá almacenarse)
* El **procesador de E/S** encuentra el momento para transmitir el dato a disco, la CPU se libera
* El **procesador de E/S** envía el dato al **controlador de disco** (con la dirección de escritura)
* El **controlador** prepara la escritura y transfiere el dato bit por bit en la superficie del disco.

Claves

Un registro es concebido como una cantidad de información que se lee o escribe. Por esta razón, es necesario extraer sólo un registro específico en vez del archivo completo. Para ello, es conveniente identificar cada registro con una llave o clave que se base en el contenido del mismo. Una clave, entonces, permite la identificación del registro, y debe permitir generar orden en el archivo por ese criterio.

Una clave puede ser:

* Únivoca / Primaria: Identifica un elemento particular dentro de un archivo. El valor que tome dicha clave pertenece a un y sólo un elemento.
* Secundaria: reconocen un conjunto de elementos con igual valor, es decir, esa clave puede repetirse.

Archivo 🡪 debe tener claves univocas

🡪 puede tener claves secundarias

Forma canónica es una forma estándar para una clave (para como se guarda y como se busca, una conversión de datos ingresados), puede derivarse a partir de reglas bien definidas.

- Representación única para la llave, ajustada a la regla. Ej: llave sólo con letras mayúsculas y sin espacios al final.

- Al introducir un registro nuevo. 1ro se forma una llave canónica para ese registro, 2do se la busca en el archivo. Si ya existe, y es univoca no se puede ingresar.

Performance de claves

El estudio de performance tiene en cuenta 3 puntos:

• Punto de partida para futuras evaluaciones

• Costo: acceso a disco, numero de comparaciones

• Caso promedio

- En el caso secuencial: el mejor caso es leer 1 registro, y el peor caso, leer n registros. El promedio es de n/2 comparaciones, la mitad de la cantidad de registros. Es de O(n), porque depende de la cantidad de registros.

Con la lectura de bloques de registros se mejora el acceso a disco, pero no varían las comparaciones.

- En el caso de acceso directo: Permite acceder a un registro preciso. Requiere una sola lectura para traer el dato, es de O(1). Debe necesariamente conocerse el lugar donde comienza el registro requerido.

El acceso directo es preferible sólo cuando se necesitan pocos registros específicos, pero este método NO siempre es el más apropiado para la extracción de información, aunque a simple vista lo pareciera.

Número relativo de registro (NRR): Indica la posición relativa con respecto al principio del archivo. Solo aplicable con registros de longitud fija. Ej. NRR 546 y longitud de cada registro 128 bytes, para leer ese determinado registro basta con calcular la distancia en bytes= 546 \* 128= 69.888.

Las operaciones esenciales sobre archivos son:

* Alta: ingresar nuevos datos al archivo.
* Modificación: alterar el contenido de algún dato del archivo.
* Consulta: presentar el contenido total o parcial del archivo.
* Baja: quitar información del archivo.

Bajas y Altas

Bajas 🡪 Lógica o Física. Tanto las bajas lógicas como físicas pueden ser realizadas en archivos de longitud fija como variable.

* Lógica: consiste en borrar la información del archivo, pero sin recuperar el espacio físico. No eliminar el registro de manera permanente, físicamente, sino considerarlo como eliminado.
  + Estrategias: colocación marca de borrado.

Ventajas:

* + - Se anula la eliminación fácilmente.
    - Performance, sólo hay que realizar lecturas hasta encontrar el registro que se desea eliminar y, una vez encontrado, se realiza sólo 1 escritura para colocar la marca de no disponible.

Desventaja:

* + - No se recupera espacio en disco, y el archivo tiende a crecer continuamente, a menos que se utilicen técnicas para reutilizar/recuperar ese espacio
* Física: borrar aquellos registros de forma permanente.
  + Ventaja:
    - El lugar de aquellos registros considerados borrados desaparece, el archivo se vuelve más pequeño y se cuenta con más espacio para utilizar. Siempre se administra un archivo de datos que ocupa el menor espacio posible

Desventaja:

* Performance de los algoritmos que implementan este tipo de solución. Este tipo de técnicas son muy costosas. La frecuencia con la que se realiza depende del dominio de aplicación y la necesidad de espacio.

Hay dos técnicas para realizar bajas físicas:

- La forma más simple es copiar todo en un archivo nuevo excepto los registros con marca de borrado. Una vez finalizado el proceso, se debe eliminar el archivo inicial del dispositivo de memoria secundaria, y renombrar al generado con el nombre original. Antes de eliminar el archivo original, se tienen en el disco 2 archivos, por lo que se debe contar con memoria suficiente para poder realizar esta técnica. Esto se llama **compactación**.

- Utilizar el mismo archivo de datos realizando los corrimientos necesarios. No se requiere mucha memoria, sin embargo, en el peor de los casos se deben hacer n-1 escrituras, en el caso que el elemento a eliminar sea el 1.

Altas

Aprovechamiento de espacio en bajas lógicas:

Existe una diferencia entre recuperar y reasignar:

* Recuperar espacio: realizar periódicamente el proceso de baja física para compactar el archivo, quitando todos esos archivos que tienen marcas de borrado.
* Reutilizar/reasignar espacio: utilizar los lugares marcados como borrados para hacer nuevas altas.

Registros longitud fija 🡪 simplemente los pongo lugares marcados.

Registros longitud variable 🡪 los nuevos elementos deben “caber” en el lugar.

Aprovechamiento en registros de longitud fija

¿Cómo saber dónde existe espacio libre donde colocar el nuevo registro?

- Búsqueda secuencial: usa las marcas de borrado. Para agregar un nuevo registro, recorre el archivo buscando una marca, es decir, el primer registro eliminado. Si no existe, se llega al final del archivo y se agrega allí. La desventaja es que es muy lento para operaciones frecuentes.

Es necesario forma de saber de inmediato si hay lugares vacíos en el archivo, sin necesidad de recorrerlo completamente:

- Lista o pilas (header): Se trata de una lista encadenada de registros disponibles. Al insertar un registro nuevo en un archivo de registros de longitud fija, cualquier registro disponible es bueno, debido a que todos los registros son de longitud fija y todos los espacios libres son iguales. Por esta razón, la lista NO necesita tener un orden particular. Se trata de una lista invertida, cuando un registro se elimina, pasa a ser el header de la lista de disponibles. Se una NRR como enlace.

Aprovechamiento en registros de longitud variable

En registros de longitud variable, el problema reside en que cada registro posee su propio tamaño y, por lo tanto, los lugares disponibles también. Por esta razón, los nuevos elementos no pueden colocarse en cualquier lugar disponible, deben “caber” necesariamente. Para indicar lugares disponibles, se utiliza también una lista invertida donde a partir del registro cabecera se disponen las direcciones libres dentro del archivo. Además, ahora es necesario indicar también la cantidad de bytes disponibles en cada caso. Como se necesita buscar, no se puede organizar la lista de disponibles como una pila.

Cada registro indica en su inicio la cant. de bytes.

Cuando se quiere recuperar/reutilizar espacio borrado, aparece un problema conocido como FRAGMENTACIÓN, que puede ser de dos tipos: interna o externa.

Fragmentación interna: Aquella que se produce cuando a un elemento se le asigna más espacio del que requiere realmente. Cuando se desperdicia lugar en un registro, es decir, se le asigna lugar y no lo ocupa totalmente. Una solución es que ese residuo, una vez ocupado el espacio libre, pase a ser otro espacio libre. El problema es cuando es demasiado pequeño.

Fragmentación externa: Espacio libre entre dos registros, pero que es demasiado pequeño para almacenar otro. El espacio que sobra cuando se ocupa un lugar libre pasa a ser un nuevo espacio libre. El problema es cuando es tan pequeño que no se puede utilizar. Posibles soluciones serían unir espacios pequeños adyacentes o elegir los espacios que mejor se adecuen a cada caso.

F. interna → se asigna espacio que no se utiliza

F. externa → quedan espacios que son tan pequeños que no se pueden utilizar. No se puede asignar.

Minimizar externa: Unir espacios libres pequeños adyacentes para generar un espacio disponible mayor (unir los huecos en el espacio de almacenamiento) o eligiendo el espacio más adecuado en cada caso.

Estrategias de colocación 🡪 son tres, y cada una tiene ventajas y desventajas.

- Primer ajuste: se selecciona la primer entrada de la lista de disponibles que sea capaz de almacenar al nuevo registro, y se le asigna al mismo. La ventaja es que minimiza la búsqueda, es más rápido y menos costoso. Sin embargo, no se preocupa por la exactitud del ajuste y tiende a generar fragmentación interna.

- Mejor ajuste: elige la entrada que más se aproxime al tamaño del nuevo registro y se le asigna completa. La ventaja es que reduce la fragmentación y aprovecha mejor el espacio. Sin embargo, exige búsqueda (procesar todos los disponibles). Puede generar fragmentación interna.

- Peor ajuste: selecciona la entrada más grande para el registro, y se le asigna solo el espacio necesario, el resto queda libre para otro registro. También exige búsqueda. Puede generar fragmentación externa.

En cuanto a las tres técnicas, la de primer ajuste es la más rápida, debido a que recorre la lista de libres hasta encontrar uno que quepa, lo asigna completo, y lo elimina de la lista de libres. Por otro lado, mejor y peor ajuste, son más ineficientes debido a que deben recorrer toda la lista buscando el mejor lugar según el caso. Una vez encontrado el lugar, mejor ajuste resuelve con mayor rapidez debido a que asigna todo el lugar completo y lo elimina de la lista de libres. Peor ajuste además de encontrarlo, debe asignar sólo lo que requiere y el espacio restante ponerlo en la lista de libres. Sin embargo, es el que más coincide con la filosofía de longitud variable, que cada registro ocupe sólo lo que necesita.

Modificación de registros

Cuando los registros de longitud fija se modifican, no existe ningún problema, porque la modificación consiste en sobrescribir el registro con el nuevo dato. El problema surge cuando quiere modificarse un registro de longitud variable, debido a que puede significar que el nuevo registro necesite el mismo espacio, ocupe menos espacio, o requiera más espacio del que utilizaba antes. Si requieren el mismo espacio, no se genera ningún problema. Si el nuevo registro ahora ocupa menos espacio, tampoco se generaría algún problema además de generar fragmentación interna, debido a que el espacio sería capaz de almacenar el registro. El verdadero problema surge cuando el nuevo registro requiere un espacio mayor. Por esta razón, **el proceso de modificación en registros de longitud variable suele dividirse en dos etapas. Primero se da de baja el elemento viejo, y luego de modificarse, el registro nuevo es insertado según la política de ajuste que se maneje.**

Búsqueda de información

Busqueda 🡪 costo

* Cantidad de accesos a disco que se realizan en función de encontrar la información deseada 🡪 op. en disco. Como se realiza en mem. secundaria tiene costo alto.
* Cantidad (#) de comparaciones, es decir verificaciones de si el dato obtenido es el que se estaba buscando 🡪 op. en memoria. Puede mejorarse los algoritmos, costo menor.

La búsqueda puede ser

* Secuencial, en la cual el registro deseado debe buscarse desde el comienzo del archivo.
* Directa, utilizando el NRR,
* Binaria/dicotomica, incorporando el uso de llaves/claves

En la **búsqueda secuencial** se trata de acceder a un registro luego de todos sus predecesores, hasta llegar al dato deseado. En el peor caso se accede a todos los registros del archivo.

Es posible encontrar un registro en 1 sólo acceso a través del NRR, debido a que es posible calcular la distancia en bytes desde el principio del archivo hasta cualquier registro. Sin embargo, es muy poco probable que se conozca el NRR del registro que contiene el dato a buscar.

Búsqueda binaria: El archivo debe estar fisicamente ordenado por clave y debe tener registros de longitud fija. La búsqueda binaria consiste en partir el archivo a la mitad y comparar esa clave con aquella que se está buscando, para determinar en qué mitad está, y así sucesivamente hasta dar con el registro deseado. Es de O(log2 N) considerando N como la cantidad de registros del archivo. Mejora la performance de la búsqueda secuencia. La ventaja es que reduce el tiempo para encontrar información. La desventaja es que se debe garantizar que el archivo esté ordenado para poder llevarse a cabo, y es costoso.

Clasificación (ordenación)

¿Cómo ordenar un archivo? Puede ser:

- Si el archivo cabe en memoria RAM, se lleva allí para poder ordenarse. Esto implicaría leer los datos de memoria secundaria, pero sería secuencialmente, y como no requiere muchos desplazamientos en el disco (cabezal, etc) no requiere un costo excesivo. Una vez en memoria RAM se ordena con una performance alta, y vuelve a escribirse el archivo ordenado en memoria secundaria. Esta opción constituye la mayor performance para ordenar archivos físicamente, pero sólo sirve para archivos pequeños.

- Si el archivo no entra en memoria RAM, otra opción es transferir a ella sólo las claves por las que se desea ordenar el archivo, junto con los NRR, desde la memoria secundaria. De esta manera, al transferir sólo esos datos, es posible almacenar una mayor cantidad de registros en RAM y, además, ¿para qué llevar a ram todo el registro completo si lo que importa para ordenar el archivo es la clave? Sin embargo, luego debe leerse el archivo completo según el orden establecido para poder escribirse nuevamente en disco. Es un costo muy alto. Además, ¿Qué sucede cuando las claves no entran en RAM?

Una alternativa posible a este problema es partir el archivo, ordenar cada parte y juntar las partes ordenadas (merge).

Entonces…

La búsqueda binaria mejora la secuencial, pero requiere estar ordenada, con el costo que eso implica. Además, el ordenamiento en RAM es sólo para archivos pequeños. Algunas formas de solucionar estos problemas, y mejorar el método de búsqueda son:

- No reordenar todo el archivo, sino sólo sus claves.

- Reorganizar con métodos más eficientes, como árboles o hashing.

# Índices

Entonces, el propósito de ordenar un archivo radica en tratar de disminuir los accesos a memoria secundaria durante la búsqueda.

Hay distintos métodos de búsqueda:

- Secuencial (poco eficiente)

- Binaria (costosa)

- Utilizando estructuras auxiliares. 🡪 ajeno al contenido que se utiliza para saber a dónde ir.

¿Qué son los índices?

- Herramienta para encontrar registros en un archivo. Consiste en un campo de llave (clave. búsqueda) y un campo de referencia que indica donde encontrar el registro dentro del archivo de datos.

- Tabla que opera con un procedimiento que acepta información acerca de ciertos valores de atributos como entrada (llave), y provee como salida, información que permite la rápida localización del registro con esos atributos.

- Estructura de datos con formato (clave, dirección) usada para decrementar el tiempo de acceso a un archivo.

En definitiva:

**Un índice es una estructura de datos adicional, que permite agilizar el acceso a la información almacenada en un archivo.** Es **otro archivo, con registros de longitud fija**, independientemente de qué estructura tenga el archivo de datos. La característica principal de un índice es que permite imponer orden a un archivo, sin que este sea reacomodado. 🡪 La estructura de un índice puede variar, la más simple es un árbol

Este índice permite imponer orden en un archivo sin que realmente este se **reacomode.**

Un índice puede mantenerse en memoria, y es más fácil de manejar que el archivo de datos, la búsqueda binaria es más rápida.

¿Cómo implementar índices?

Operaciones básicas en un archivo indizado:

- Crear los archivos: el índice y el archivo de datos se crean vacíos, solo con registro cabecera.

- Cargar el índice en memoria: se supone que cabe, ya que es lo suficientemente pequeño.

- Reescritura del archivo de índice: cambios 🡪 reescribir.

- Agregar nuevos registros: Implica agregar nueva información al archivo de datos y al archivo de índices. Respecto al archivo de datos, el nuevo registro se copia al final del archivo, y es necesario saber el NRR (si es de longitud fija) o la distancia en bytes (si es variable) para poder utilizarlo en el índice. Para el índice, se ordena con cada nuevo elemento en forma canónica, en memoria, y se setea el flag anterior.

- Eliminar un registro

* Arch. datos 🡺 Cualquier técnica de las vistas para reutilizar el espacio
* Arch. índices 🡺 se quita la entrada (ó se podría marcar como borrado).

- Actualización de registros:

- Sin modificar la clave:

* Si el registro no cambia de longitud, se almacena en la misma posición física, el índice “no se toca”.
* Si el reg. cambia de longitud (se agranda) y se reubica en el arch. de datos 🡺 se debe guardar la nueva posición inicial en el índice
* Si se trata de long. Fija, no hay que hacer más actividad

- Modificando la clave: Cuando se modifica el archivo de datos, también se debe actualizar y reorganizar el archivo de índices. Cómo simplificar → Modificar = Eliminar + Agregar (ya vistos).

Ventajas

- Ventajas: Al ser de menor tamaño que el archivo de datos asociado, y ser de longitud fija, mejora la performance de búsqueda. Podrían caber en RAM. Siempre permite búsqueda binaria, porque al ser más pequeño el mantenimiento del orden es menos costoso y siempre son de longitud fija.

Cuando el archivo de índices no entra en RAM hay dos soluciones: Arboles / Hashing.

Índices secundarios

No sería natural solicitar un dato por clave, si no por algo más fácil de recordar. Este campo es un campo que pertenece a una llave secundaria porque puede repetirse.

**El índice secundario relaciona una clave secundaria, con una o más claves primarias.**

Es decir, se cuenta con muchas claves primarias, que son unívocas, para identificar registros en particular. Se tiene, además, un archivo índice, que almacena esas claves primarias y dónde ir en el archivo original para encontrar el elemento que identifican. Por otro lado, se cuenta con una clave secundaria, que puede repetirse, pero es más fácil y natural recordarla. También, se cuenta con un índice secundario, que relaciona una clave secundaria con una o más claves primarias.

Acceso 🡺 1º por llave secundaria (se obtiene la clave primaria) y luego llave primaria (en índice primario)

¿Por qué se accede de esta manera, en vez de que el índice secundario directamente tenga la dirección física?

Porque un archivo puede tener varios índices secundarios, y si el registro llegara a cambiar de ubicación en el archivo, habría que actualizar todos los índices secundarios. Por esta razón, sólo el índice primario tiene definida la dirección física del registro, y en caso de ser necesario, sólo debería cambiar el índice primario.

**El principal problema** **de los índices secundarios** es la repetición de información. Esto hace que en la organización de índices se almacene la misma clave secundaria en distintos registros, tantos como ocurrencias haya. Esto implica mayor espacio, generando una menor posibilidad de que el índice quepa en memoria.

Una primera solución a este problema implicaría almacenar en un registro todas las

ocurrencias de una clave secundaria. Cada registro está formado por la clave secundaria + vector de claves primarias correspondientes.

Al agregar un nuevo registro de una clave existente simplemente se debe insertar de forma ordenada la clave primaria en el vector de ocurrencias correspondiente. Al agregar un nuevo registro con una clave nueva, se genera un arreglo con la clave y un

elemento en el vector de punteros.

El problema que presenta esa solución es la elección del tamaño del vector, ya que conviene que sea de longitud fija. Al decidir previamente el tamaño del vector, puede haber casos en los que sea insuficiente, o puede haber casos en los que sobre espacio, provocando fragmentación interna 🡪 Mejora: clave secundaria + lista de claves primarias asociadas.

Listas invertidas

Cuando en un archivo una clave secundaria lleva a un conjunto de una o más claves primarias, se puede utilizar una lista de referencias de claves primarias. No se desperdicia espacio, porque no debe realizarse una reserva de éste previamente, y pueden existir cualquier número de claves primarias para cada clave secundaria. Si se agrega un elemento a la lista, no es necesaria una reorganización completa, debido a la naturaleza de las listas invertidas. Primero se accede al archivo de claves secundarias y se obtiene la dirección de acceso a la lista de claves primarias. Dicha lista, se encuentra almacenada en otro archivo.

- Operaciones: Agregar un nuevo consiste en agregar concurrencias en la lista invertida. Lo mismo sucede con la operación borrar. Las modificaciones dependen del caso.

Ventajas y desventajas de índice secundario implementado con listas de llaves primarias

Ventajas: El único reacomodamiento en el archivo índice se produce cuando se agrega una nueva clave secundaria. Si se agrega o borran elementos de una clave secundaria ya existente, sólo se debe modificar el archivo que contiene la lista. Como se generan 2 archivos, uno de ellos podría residir en memoria secundaria, liberando la RAM.

Desventajas: el archivo de listas es conveniente que esté en memoria principal, debido a que podría haber muchos desplazamientos en disco. Costoso si hay muchos índices secundarios.

# Arboles

Un árbol es una estructura de datos que permite localizar en forma más rápida,

información de un archivo. Tiene intrínsecamente búsqueda binaria (en el interior, propio, internamente).

Árboles binarios

Se denomina árbol binario a una estructura de datos dinámica no lineal, donde cada nodo tiene a lo sumo dos sucesores: a izquierda y a derecha. Tiene sentido cuando está ordenado.

La búsqueda siempre parte desde la raíz y continúa hacia la izquierda o derecha según el elemento que se desee encontrar. De esta manera, en cada comparación se descarta la mitad del archivo restante, donde es seguro que el elemento no se encuentra. Tiene orden Log2(N) accesos a disco.

Los árboles como estructura se implementan en memoria RAM. Sin embargo, para que pueda ser utilizado como una estructura índice de búsqueda se requiere persistencia de la información y, por lo tanto, deben implementarse sobre almacenamiento secundario. Un nodo, además de contener el elemento propiamente dicho, en los hijos izq y der no tiene punteros como siempre, tiene un número que hace referencia al NRR

Performance

Si se debe agregar un nuevo elemento, se busca al padre, y se actualiza al padre haciendo referencia al nuevo hijo: esto serían Log2(n) lecturas a disco para encontrar el padre del nuevo elemento y 2 escrituras (nuevo elemento y actualización del padre).

Mejor que en un archivo de índices ordenado, ya que este se debe reacomodar completo.

Con respecto a las bajas, el análisis es el mismo. Para borrar un elemento, éste debe ser terminal y, de otra manera, se reemplaza por el menor de sus hijos mayores. Nuevamente serían Log2(N) lecturas y 2 escrituras.

La desventaja recae en que un árbol binario se desbalancea fácilmente.

*Un árbol está balanceado cuando la altura de la trayectoria más corta hacia una hoja no difiere de la altura de la trayectoria más grande, es decir, todas las hojas se encuentran a la misma distancia del nodo raíz.*

En el peor de los casos, él desbalanceo provoca que el árbol se transforma en una estructura de tipo lista, la performance de búsqueda decae transformándose en orden lineal.

Árboles AVL

Un árbol AVL, es un árbol binario balanceado en altura (BA(1)). Estos árboles tienen la característica que, para cualquier nodo, la diferencia máxima entre las alturas de sus dos subarboles, derecho e izquierdo, no pude superar determinado limite donde ese límite es el nivel de balanceo en altura del árbol. Por ejemplo, si el árbol es BA(1) significa que la diferencia no puede superar 1, es el máximo desbalanceo posible. Los árboles AVL son balanceados en altura.

El problema con los árboles AVL es que puede suceder que, al insertar algún elemento, se infrinja la regla establecida. En ese caso, habría que rebalancear el árbol con algoritmos de rotación, y los costos de acceso a disco aumentan. Algoritmia complicada y costosa.

Árboles binarios paginados

El árbol binario se divide en páginas (se página), y cada una de ellas contiene un conjunto de nodos, que están ubicados en direcciones físicas cercanas. De esta manera, cuando se accede a disco no se trasfieren una determinada cantidad de bytes, sino que se transmite una página completa.

Como en cada página caben una cierta cantidad de nodos, en pocos accesos podría encontrarse el elemento buscado. Para transferir una página completa, si bien se accede a direcciones físicas diferentes, éstas son cercanas y los desplazamientos son mínimos y no implican un costo.

La performance de un árbol binario paginado depende de la cantidad de nodos que entren cada página. Si la página contiene 255 nodos, la performance de búsqueda sería de log256(N), es decir Logk+1(N), donde N es la cantidad total de elementos del árbol y k la cantidad de nodos que entran en una página.

Se necesita que esté todo en el mismo lugar del disco rígido, si no cada página debería quedar incompleta hasta que llegue un elemento que se relacione a ella, en vez de entrar en el primero disponible.

Se debería tener un algoritmo muy costoso de implementar, y también en cuanto a la performance. Los árboles binario paginados traen un costo mucho mayor que el beneficio.

Árboles multicamino

Generalización de árboles binarios, c/nodo tiene k punteros y k-1 claves (o registros), disminuye la profundidad del árbol. Sigue habiendo problemas de balanceo.

Árboles balanceados

Son árboles multicamino con una construcción especial en forma ascendente que permite mantenerlo balanceado a bajo costo.

Propiedades de un árbol B de orden M:

* Ningún nodo tiene más de M hijos.
* Cada nodo (menos la raíz y los terminales) tienen como mínimo [M/2] hijos.
* La raíz tiene como mínimo 2 hijos (o sino ninguno)
* Todos los nodos terminales a igual nivel.
* Nodos no terminales con k hijos contienen k-1 registros. Los nodos terminales tienen:

➔ Mínimo: [M/2] - 1 registros

➔ Máximo: M-1 registros.

Creación

Los árboles B se construyen de forma ascendente: cuando no se dispone de suficiente espacio, es la raíz quien debe alejarse de los nodos terminales y no al revés.

Cuando llega un nuevo registro a un nodo lleno, y produce overflow, ese nodo se divide en 2 y un registro pasa arriba como nuevo padre. Crece el nivel del árbol. **División y promoción.**

*\* Cuando un nodo es de orden impar, ej 5, y al llegar otro produce overflow y queda un # de elementos pares, ej 6,se divide a la mitad y promociona el más chico del segundo nodo (los más grandes) En cambio, si es de orden par, ej 4, cuando llega uno y se divide el nodo, promociona justo el del medio y quedan ambos nodos con igual cantidad de elementos.*

Búsqueda en árboles B

Comienza la búsqueda por la raíz y se va bifurcando según el dato que se busque, descartando la mitad de los elementos, hasta encontrar el elemento deseado o un nodo sin hijos, que no contenga el elemento.

Busca la llave en el nodo raíz

Localiza la llave

Toma el puntero anterior a la llave mayor

Puntero nulo

Fin de proceso

No está el elemento

Performance de búsqueda: La eficiencia de búsqueda consiste en contar los accesos a disco que se requieren para encontrar un elemento o determinar que no se encuentra. La cantidad de accesos va a ser un número acotado entre el rango [1, H], siendo H la altura del árbol.

- Mejor caso: 1 lectura.

- Peor caso: h lecturas, donde h es la altura del árbol.

Medir eficiencia en base a N (cantidad de registros), entonces, ¿Cuál es el valor de h?

Axioma: árbol balanceado de orden M, si el número de elementos del árbol es N → hay N+1 punteros nulos en nodos terminales

Cota para h

|  |  |
| --- | --- |
| Nivel | N° min. de descendientes |
| 1 | 2 |
| 2 | 2\*[M/2] |
| 3 | 2\*[M/2]\*[M/2] |
| ... | ... |
| h | 2\* |

Relación entre h y n° de nodos

N+1 2\*

h [1 + ]

Si M = 512 y N = 1000000 → h 3,37 (4 lecturas encuentran un registro)

Inserción

Hay que recordar una propiedad de los árboles B, que es que los elementos siempre se insertan en los nodos terminales. Por lo tanto, en una inserción es necesario realizar H lecturas para encontrar el nodo al que corresponde el nuevo elemento. Casos posibles:

- El registro tiene lugar en el nodo terminal, no se produce overflow:

* Solo se hacen reacomodamientos internos en el nodo.
* H lecturas
* 1 escritura

- El registro no tiene lugar en el nodo terminal, se produce overflow:

* El nodo se divide y los elementos se reparten entre los nodos nuevos. Hay una promoción de una clave al nivel superior, y esta promoción puede propagarse y generar una nueva raíz.
* H lecturas
* 2h + 1 escrituras (2 por nivel más la raíz)

Eliminación

* Siempre eliminar de nodos terminales.
* Si se va a eliminar un elemento que no está en nodo terminal → llevarlo primero a nodo terminal.
* Posibilidades ante eliminación
* **Mejor caso:** borra un elemento del nodo y no produce underflow, solo reacomodas (n° elementos [M/2] - 1)
* **Peor caso:** se produce underflow, n° elementos < [M/2] - 1
* 2 soluciones:

1. **Redistribuir:** Cuando un nodo tiene underflow, puede trasladarse llaves de un nodo adyacente hermano (en caso que este tenga suficientes elementos).
2. **Concatenar:** Si un nodo adyacente hermano está al mínimo (no le sobra ningún elemento), no se puede redistribuir, se concatena con un nodo adyacente disminuyendo el n° de nodos (y en algunos casos la altura del árbol).

**Nodo adyacente hermano**: 2 nodos son adyacentes hermanos si tienen el mismo padre y son apuntados por punteros adyacentes en el padre. El nodo adyacente depende de la política o que sea un caso donde se tiene 1 solo hermano.

Performance de la eliminación

* **Mejor caso**: (borra de un nodo terminal)
* H lecturas
* 1 escritura
* **Peor caso**: (concatenación lleva a decrementar el nivel del árbol en 1)
* 2h - 1 lecturas
* H + 1 escrituras

Árboles Balanceados B\*

Los árboles B\* utilizan la también en las altas. Cuando tanto un nodo como su adyacente se encuentran completos, recién en ese momentos se produce la división. Esta forma de crear un árbol B\* produce nodos más llenos (al menos ⅔) y no a la mitad como los árboles B. Por eso, todos los nodos quedan mejor distribuidos. Se puede posponer la creación de páginas nuevas (nodos). Árbol más eficientes en términos de utilización de espacio

Propiedades (orden M)

* Cada página tiene máximo M descendientes.
* Cada página, menos la raíz y las hojas, tienen al menos [(2M-1)/3] descendientes.
* La raíz tiene al menos 2 descendientes (o ninguno).
* Todas las hojas aparecen en igual nivel.
* Una página que no sea hoja si tiene k descendientes contiene k-1 llaves.
* Una página hoja contiene por lo menos [(2M-1)/3]-1 llaves, y no más de M-1.

Existe una excepción a las propiedades de un árbol B\* y es cuando sólo se tiene el nodo raíz. En este caso, al tener overflow, es imposible redistribuir porque no se cuenta con nodos hermanos.

Búsqueda y Baja en B\* igual que en B.

Inserción

* Tres casos posibles:
* De un lado: El hermano adyacente tenido en cuenta para la redistribución será uno solo, izquierda o derecha (a menos que el nodo en overflow sea el extremo). De no ser posible porque ese hermano adyacente también se encuentra en overflow, se produce una división.
* De un lado u otro: si el nodo de la derecha está lleno y no se puede redistribuir, se busca el de la izquierda. Y viceversa. De no ser posible, se toma un hermano (ej izq si es política izq o derecha) y se divide, generando 3 nodos (2/3 llenos).
* De un lado y otro: se busca redistribuir por un lado y si no es posible, por el otro. De no ser posible tampoco, se divide entre los tres nodos, generando 4 con ¾ partes completas cada uno.



Estrategias de reemplazo: LRU (last recently used) 🡪 los mas usados se dejan en un buffer

Archivos secuenciales indizados

Un archivo con acceso secuencial indizado es aquel que permite visualizar la información de dos maneras:

- Indizada: puede verse como un conjunto de registros ordenados por clave.

- Secuencial: puede accederse al archivo secuencialmente, con registros físicamente contiguos y ordenados nuevamente por una clave.

Si se deseara acceder a la información de manera secuencial en B o B\* utilizando el orden de la llave, la única alternativa sería utilizando el índice. Recuperar todos los registros del archivo, significaría acceder a nivel hoja, volver al padre, y acceder al nodo terminal siguiente. Es mucho menos eficiente que tener la estructura ordenada físicamente.

Tener el archivo ordenado físicamente resuelve este problema, pero resulta

inaceptable al momento de realizar una búsqueda, una inserción o eliminación.

Se requiere una estructura en la que sean compatibles ambos casos.

Árboles B+

Consiste en un conjunto de grupos de registros ordenados por clave en forma secuencial, junto con un conjunto de índices, que proporciona acceso rápido a esos registros. La característica principal de los árboles B+, las hojas (nodos terminales) están enlazadas y forman una lista. Todos los elementos se encuentran en las hojas. En cambio, los nodos no terminales funcionan como índices, hay punteros a los datos. Un árbol B+ utiliza la estructura de un B o de un B\*, por lo tanto, las inserciones y bajas se manejan de esa manera.

Esto permite acceso aleatorio rápido (como los B y B\*) y también acceso secuencial rápido.

Conjunto índice: proporciona acceso indexado a los registros. En la raíz y nodos internos se encuentran las claves necesarias para establecer caminos de búsqueda.

Conjunto secuencia: Tienen todos los registros del archivo. Las hojas se vinculan entre sí (enlazan, como una lista, con punteros) para hacer el acceso secuencial más rápido.

#### 

Propiedades

* Cada página tiene máximo M descendientes
* Cada página, menos la raíz y las hojas, tienen entre [M/2] y M hijos
* La raíz tiene al menos 2 descendientes (o ninguno)
* Todas las hojas aparecen en igual nivel
* Una página que no sea hoja si tiene k descendientes, contiene k-1 llaves
* Los nodos terminales representan un conjunto de datos y son linkeados juntos
* Los nodos no terminales no tienen datos, sino punteros a los datos.

Separadores

* Derivados de las llaves de los registros que limitan un bloque en el conjunto de secuencia.
* Separadores más cortos, ocupan espacio mínimo

Árbol B+ de prefijos simples

Se trata de un árbol B+ en el cual el conjunto índice está constituido por separadores más cortos. El uso de prefijos simples busca aprovechar mejor el uso de espacio físico. Un árbol B+ de prefijos simples es un árbol B+ donde los separadores están representados por la mínima expresión posible de la clave, que sea capaz de terminar si la búsqueda se realiza a la izquierda o la derecha. (Se utiliza lo mínimo que pueda servir como separador)

Búsqueda en B+

Es similar a los árboles B. Se sigue con la búsqueda hasta el último nivel del árbol (hojas), porque allí están todos los elementos.

Inserción (altas)

Siempre se agregan elementos en el último nivel del árbol. Si hay overflow:

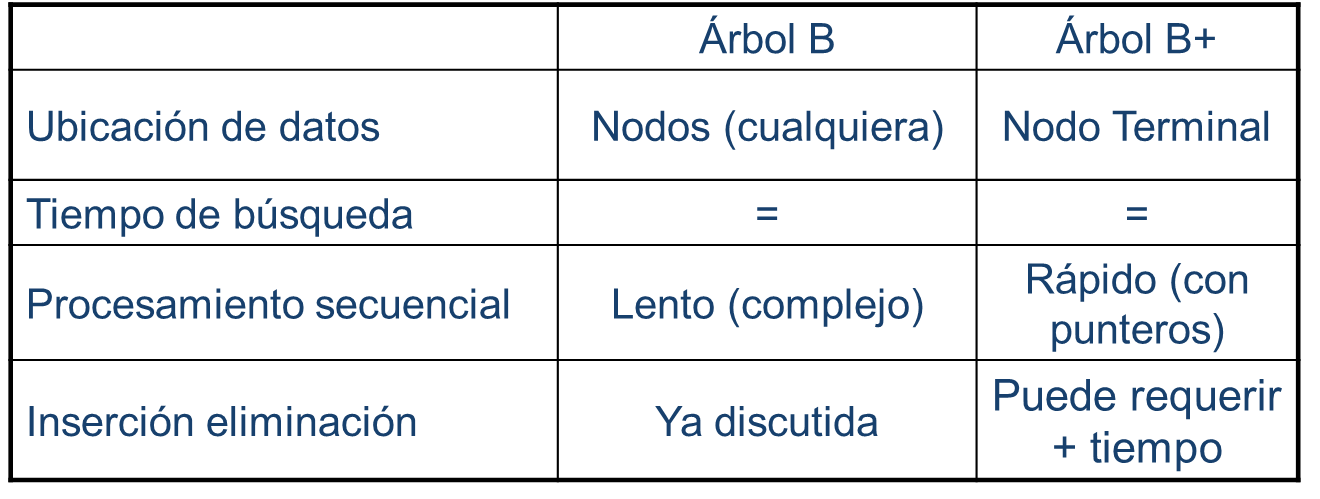
- El nodo en overflow se divide en 2, distribuyendo las claves equitativamente.

- Una copia de la clave del medio (o la más chica de los grandes), se promociona al nodo padre. Esto sucede solamente cuando el overflow sucede a nivel hoja.

Bajas en B+

Las claves a eliminar siempre están en las hojas. Los underflow son tratados de igual manera que en los árboles B. !! Importante: Las claves de la raíz o nodos internos NO se modifican. Son índices.

Comparación Arbol B y Arbol B+



Resumen árboles

B → Mínimo en nodos no terminales (M/2) y en hojas (M/2)-1

B\* → Mínimo [(2M – 1) / 3] -1

B+ → Mínimo (M/2) -1

El árbol B, es el único que difieren los mínimos entre las hojas y los nodos no terminales. En la raíz, para todos los B es mínimo 2 hijos o ninguno.

AVL → binario

Familia de árboles B → multicamino.

Los árboles de familia B son muy flexibles y útiles para la administración de índices asociados a un archivo de datos. Sin embargo, no es la única solución, y la mejor estructura para un archivo va a depender del archivo en sí y de su propósito. A veces utilizar un árbol no es necesario, por ejemplo, si se tienen pocos registros.

# Hashing

* Secuencia: N/2 accesos promedio
* Ordenado:
* Árboles: 3 o 4 accesos

¿Qué es hashing o dispersión?

- Es una técnica para generar una dirección base única para una clave dada. La dispersión se usa cuando se requiere acceso rápido a una llave.

- Es una técnica que convierte la clave del registro en un número aleatorio, el que sirve después para determinar donde se almacena el registro.

- Es una técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en dirección de almacenamiento.

Es un método que mejora la eficiencia obtenida con árboles B, asegurando un promedio de 1 acceso para recuperar la información.

Archivos secuenciales indexados

* Archivos de datos
* Archivo con índice primario
* Archivos con índices unívocos o secundarios

Archivos directos

* Un acceso
* No puede haber estructuras adicionales
* Se organiza el archivo de datos
* Solo puede organizarse por un único criterio
* Ese criterio es la clave primaria

Atributos del hash

- No requiere almacenamiento adicional: no requiere índice, es el archivo de datos el que resulta disperso, dicha dispersión se genera a partir de la llave primaria.

- Facilita inserción y eliminación rápida de registros: se realiza de manera más eficiente en términos de acceso a disco (1 acceso a disco).

- Encuentra registros con muy pocos accesos al disco en promedio: el promedio es de un acceso a disco, si bien no es posible asegurar que cualquier registro sea hallado en un solo acceso, la mayoría de las búsquedas serán satisfechas con un acceso a disco.

Limitaciones: situaciones donde el método no es aplicable.

- No podemos usar registros de longitud variable: debido a que cada dirección física obtenida debe tener capacidad para almacenar un registro de tamaño conocido. Cuando se obtiene la dirección física, ésta tiene un espacio de almacenamiento asociado, conocido, limitado y NO puede ocurrir que el registro no quepa en él. Además, la función de hash genera, en teoría, un número único para cada clave, y ese espacio obtenido no le serviría al registro.

- No es posible un orden lógico de datos: los registros son esparcidos en el archivo, el orden está dado por la función de hash.

- No permite claves duplicadas: la función de hash no se puede aplicar a claves secundarias, porque es necesario que genere un número único.

Para determinar la dirección

Cuando se quiere almacenar un nuevo registro, la clave primaria de éste se convierte en un número casi aleatorio, a dicho número se le aplica la función de Hash que devuelve otro valor. Ese número se mapea \*\* y corresponde a una dirección de memoria donde se guarda el registro. Si la dirección está ocupada se genera colisión/overflow(tratamiento especial)

\*\* La función de Hash no contempla retornar un valor en el rango correcto de direcciones disponibles, por lo que se debe mapear dicho resultado para que este en el rango.

Claves sinónimas: Dos claves son sinónimas cuando, al aplicarse al función de hash, obtienen el mismo resultado, que corresponde a la misma dirección donde almacenarse.

El método de hashing presenta 2 alternativas para su implementación:

- Hashing con espacio de direccionamiento estático: aquella política en la cual el espacio disponible para dispersar los registros en un archivo está fijado previamente. La función de hash aplicada a una clave da como resultado una dirección física posible. Tiene espacio determinado y limitado.

- Hashing con espacio de direccionamiento dinámico: aquella política en la cual el espacio disponible para dispersar los registros en un archivo aumenta o se disminuye según las necesidades de espacio del archivo en cada momento.

Hashing estatico

Cuando se utiliza con espacio de direccionamiento estático la eficiencia depende de 4 propiedades:

1) Función de hash: Técnica para generar la dirección

2) Tamaño de los nodos: Dependiendo del disco

3) Densidad de empaquetamiento: DE = n° de reg/total archivo

4) Método de tratamiento de desbordes

1) Función de Hash:

* Caja negra que a partir de una clave se obtiene la dirección donde debe estar el registro.
* Diferencias con índices
  + Dispersión no hay relación aparente entre llave y dirección. Las claves son independientes, no influyen una sobre la otra. En cambio, en los índices por ejemplo cuando eran de longitud fija de 128 se podría utilizar el NRR, por ejemplo 4, para determinar en qué dirección está el registro.
  + Dos llaves distintas pueden transformarse en iguales direcciones (colisiones)

Colisión: Situación en la que un registro es asignado a una dirección que está utilizada por otro registro.

Overflow: Situación en la que se produce una colisión y, además, no queda espacio para el registro nuevo. Soluciones posibles:

* Algoritmos de dispersión sin colisiones o que estas colisiones nunca produzcan overflow (perfectos) (imposibles de conseguir)
* Almacenar los registros de alguna otra forma minimizando colisiones:
* Esparcir registros: buscar una función de dispersión que distribuya los registros de la forma más aleatoria posible.
* Usar más espacio en disco: distribuir pocos registros en muchas direcciones, baja la densidad de empaquetamiento. La ventaja es que disminuye colisiones y, por lo tanto, overflow, pero desperdicia espacio.
* Colocar más de un registro por dirección: direcciones con N claves, mejoras notables. Se relaciona con el tamaño de cubeta

Algoritmos simples de dispersión

● Condiciones

❖ Repartir registros en forma uniforme

❖ Aleatoria (las claves son independientes, no influyen una sobre la otra)

Tres pasos:

1. Representar la llave en forma numérica (en caso que no lo sea)
2. Aplicar la función
3. Relacionar el número resultante con el espacio disponible

2) Tamaño de las cubetas (Nodos)

El tamaño de cada nodo estará sujeto a la posibilidad de transferencia de información en cada operación de E/S entre la RAM y el disco.

- Cada dirección de memoria puede almacenar más de un registro.

- A mayor tamaño de la cubeta:

• Menor overflow

• Mayor fragmentación (interna)

• Búsqueda más lenta dentro de la cubeta (No es tan importante porque se trata de una búsqueda local)

3) Densidad de empaquetamiento

**La relación entre el espacio disponible para el archivo de datos, y la cantidad de registros que integran dicho archivo.**

* Proporción de espacio del archivo asignado que en realidad almacena registros
* DE = número de registros del archivo  
   capacidad total del archivo
* Densidad de empaquetamiento menor
  + Menos overflow
  + Más desperdicio de espacio, fragmentación

La DE no es constante. Al principio cuando se crea el archivo y, supongamos que tiene 1 registro, es muy pequeña. A medida que se van agregando registros, el archivo crece y la DE también.

Estimación del overflow

* N # de cubetas,
* C capacidad de nodo,
* R # reg. Del archivo
* DE = R  
   C x N

Probabilidad que una cubeta reciba I registros (distribución de Poisson). Sirve para estimar el overflow que habrá.

Análisis numéricos de Hashing

* En general si hay n direcciones, entonces el # esperado de direcciones con I registros asignados es N\*P(I).
* Las colisiones aumentan con al archivo más “lleno”.

Menos cubetas mas grandes 🡪 menos Overflow.

Si DE llega a 75 % debo hacer algo. 

4) Tratamiento de Colisiones con Overflow

Cuando ocurre overflow, deben realizarse 2 acciones: encontrar lugar para el registro en otra dirección, y asegurarse que posteriormente el registro pueda ser encontrado allí.

Algunos métodos para el tratamiendo de overflow son:

Saturación progresiva: Cuando se completa el nodo, se busca el próximo hasta encontrar uno libre, y se coloca el nuevo registro allí. Como los registros van almacenándose en el próximo nodo libre/vacio, cuando se realiza una búsqueda, esta termina al encontrar un nodo libre. Cuando se quiere eliminar un registro es necesario colocar una marca de borrado, que indique que ese nodo está libre, es decir puede ser nuevamente utilizado, pero estuvo ocupado. Sirve para no detener la búsqueda.

Saturación progresiva encadenada: similar a saturación progresiva, pero los registros de saturación se encadenan y “no ocupan” necesariamente posiciones contiguas. Se forma una lista, por lo que el algoritmo es más complejo. Cada cubeta contiene el registro y puntero a otra cubeta que debería estar allí.

! Importante: Si una clave A se ubica en un nodo 1, y una clave B debe ubicarse en la misma (sinónimas), pero se produce overflow y se ubica en el proximo nodo libre 2. Suponiendo que llega una clave C que debe ubicarse en el nodo 2 (base), la que debe reacomodarse es la clave B, porque la idea es que la mayoría de las claves estén en el lugar que les corresponde originalmente (base)

- Doble dispersión: el problema que tiene la saturación progresiva es que, a medida que se producen saturaciones, los registros tienden a esparcirse en nodos cercanos. Esto produce exceso de saturación sectorizada. Solucion 🡪 almaceno los registros de overflow en zonas no relacionadas.

Se aplica una segunda función de hash a la clave. Cuando se produce overflow, se suma a la dirección original el resultado de la segunda función de hash, tantas veces como sea necesario hasta encontrar una dirección con espacio.

- Área de desborde separado: No utiliza nodos direccionables por la función de Hash para los registros en overflow, estos van a nodos especiales. Se reservan X nodos para overflow, cuando en una casilla se produce overflow, voy al lugar del espacio reservado y lo guardo ahí. No hay que reacomodar/reubicar y en la base, todos se ubican en su lugar correspondiente.

* Ubicación del desborde
  + A intervalos regulares entre direcciones asignadas
  + Cilindros de desborde

**El hashing estático resulta ser un método que si bien satisface las búsquedas en un acceso en el 99,9% de los casos (siempre y cuando la DE sea menor al 75%). Sin embargo existen situaciones en las cuáles más de un acceso sean requeridas, y estas situaciones se dan cuando se generan situaciones de saturación.**

Hashing dinámico

Con hashing estático surgen algunos problemas:

* Requiere un número de direcciones fijas, virtualmente imposible.
* Cuando un archivo se completa:

🡪 Aumentar cant. nodos, por lo tanto, modificar función de hash para que pueda direccionar a esos nuevos nodos.

🡪 El archivo debe ser redispersado y eso tiene un costo muy alto. Mientras se realiza esta operación, no es posible que el usuario final acceda al archivo.

Solución 🡪 Hashing dinamico

**La eficiencia de búsqueda en Hash Dinámico SIEMPRE es uno. Ya que no encuentra cubetas en saturación que impliquen más accesos.**

Varias posibilidades:

- Hash virtual

- Hash dinámico

- Hash Extensible

Hash Extensible

El principio del método consiste en comenzar a trabajar con un único nodo para almacenar registros, e ir aumentando la cantidad de direcciones disponibles a medida que se van completando dichos nodos. Este método, como todos los que trabajan con espacio de direccionamiento dinámico, no utiliza el concepto de DE. Esto se debe a que el espacio que utiliza va aumentando o disminuyendo según las necesidades.

¿Cómo trabaja?

La función de hash retorna un string de bits (normalmente 32) cuya cantidad determina la cantidad máxima de direcciones a las que puede acceder el método.

Hash extensible trabaja bajo las siguientes pautas:

* Se utilizan solo los bits necesarios de acuerdo a cada instancia del archivo
* Los bits tomados forman la dirección del nodo que se utilizará
* Si se intenta insertar a una cubeta llena, deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nuevo, para ello se toma un bit más
* La tabla tendrá tantas entradas (direcciones de nodos) como , siendo i el número de bits actuales para el sistema

El método de Hash Dinámico requiere de una estructura auxiliar (requiere espacio adicional), una tabla que se almacena en memoria principal (RAM), que contiene la dirección física de cada nodo.

Se debe tener en cuenta que la función de hash no retorna una dirección, sino una secuencia de bits. Esta secuencia de bits permite obtener de la tabla en memoria principal, la dirección física del nodo para almacenar la clave.

Elección de organización

Los archivos se utilizan para acomodar datos para satisfacer requerimientos



Para poder realizar una correcta elección sobre la organización de un archivo, hay que tener en cuenta algunos factores.

¿Qué examinar?

- Características del archivo: número de registros, tamaño de registros.

Requerimientos de usuario: tipos de operaciones, número de accesos a archivos, características del hardware (tamaño de sectores, bloques, pistas, cilindros, etc.), parámetros, tiempo (necesario para desarrollar y mantener el soft, para procesar archivos), uso promedio (número de registros usados/ número de registros)

