Resumen Final FOD

Clase 1: Conceptos de DB y Archivos

Que es una base de datos?

- 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.

Propiedades de una BD

- Representa algunos aspectos del mundo real.
- Una BD es una colección coherente de datos, con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD, osea los datos deben tener cierta lógica.
- Una BD se diseña, construye y completa de datos para un propósito especifico.
- Una BD esta sustentada físicamente en archivos en dispositivos de almacenamiento persistente de datos.

Archivos:

- Colección de registros guardados en almacenamiento secundario.
- Colección de datos almacenados en dispositivos secundarios de memoria.

Acceso a los archivos:

- Secuencial Fisico/Serie: Acceso a los registros uno tras otro y en el orden físico en el que están guardados. Es de O(n) y la cantidad de lecturas en promedio es n/2.
- Secuencial indizado(logico)/Secuencial : Acceso a los registros de acuerdo al orden establecido por otra estructura/clave. Ej: guía telefónica o índice tematico de un libro. Este tipo de archivos su búsqueda es de o (log n)
- Acceso directo: Se accede a un registro determinado sin la necesidad de haber pasado por los registros anteriores. Se utiliza una función de hashing para determinar directamente la posición en la ue salta un registro. Orden de búsqueda (1) en el mejor de los casos y O(n) en el peor de los casos.

Tipos de archivos:

- Serie: Cada registro es accesible luego de procesar su antecesor, simples de acceder se lleva a cabo mediante el acceso secuencial físico (tienen orden físico).
- Secuencial: Los registros son accesibles en orden de alguna clave (acceso secuencial indizado) secuencial lógica.
- Directo: Se accede al registro deseado (acceso direto).

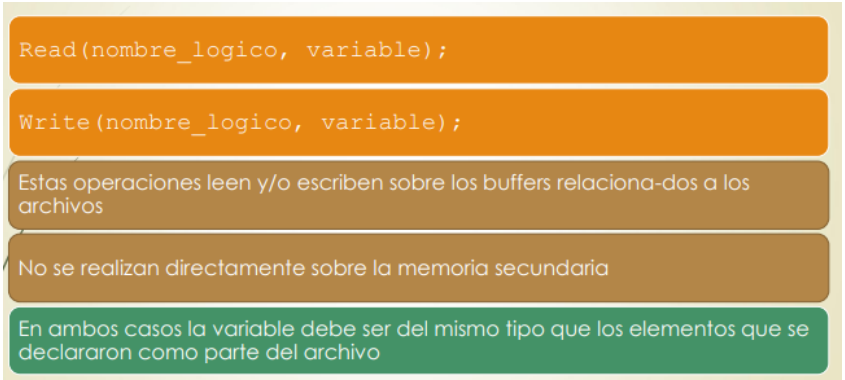
Chequearlo:

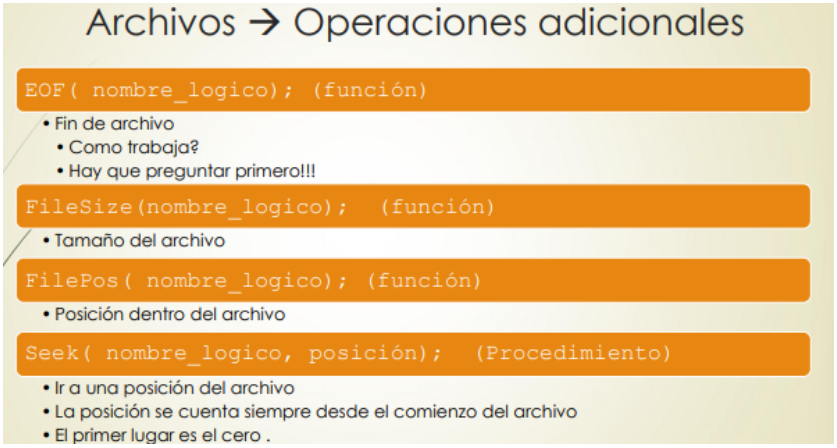
Tipo de Archivo	Orden de Búsqueda	Tiempo Promedio
Serie (Secuencial)	$O(n)$	$n/2$
Secuencial Indexado	$O(\log n)$	$\log_2(n)$
Acceso Directo	$O(1)$	1

Buffers:

Memoria intermedia entre un archivo y un programa donde los datos residen provisoriamente hasta ser almacenados en memoria secundaria. Los buffers ocupan lugar en RAM, el SO es el encargado de manipular los buffers. Se transfiere información desde el disco a la RAM y viceversa.

Archivos – Operaciones -Basicas:





Clase 2: Maestro - Detalle

Se denomina maestro al archivo que resume un determinado conjunto de datos.

Se denomina detalle al archivo que agrupa información que se utilizara para modificar el contenido del archivo maestro. En el archivo detalle solo aparecen registros que existen en el archivo maestro.

Ambos archivos (maestro y detalle/s, deben estar ordenados por el mismo criterio.

Clase 3: archivos introducción.

Archivo físico: Archivo que existe en almacenamiento secundario. Es el archivo tal como lo conoce el S.O y que aparece en su directorio de archivos. (Almacenado en almacenamiento secundario)

Archivo lógico: Es el archivo visto por el programa, no se sabe cual archivo físico real se utiliza o donde está ubicado. (Almacenado en RAM)

La memoria RAM es rápida y de acceso simple, pero su uso tiene algunas desventajas respecto al almacenamiento secundario, como la RAM es de tipo volátil, no nos asegura que al trabajar con datos sensibles, que estén preservados de cualquier problema, ya que si se corta la luz,o se apaga la computadora, los datos se pierden de la memoria RAM, podemos solucionar este problema, almacenando los datos en memoria secundaria.

La información debe estar almacenada en un tipo de almacenamiento que su recuperación esté garantizada, y que su no perdida esté garantizada, por eso usamos almacenamiento secundario, nos proporciona esas ventajas, pero con un costo de acceso mayor, el acceso a almacenamiento secundario es lento.

Campo de un archivo: Unidad lógicamente mas pequeña de un archivo. Permite organizar y separar información. Un campo puede tener o no su dimensión definida, es decir puede tener su dimensión física o variable.

Campos de longitud fija: Se define una cantidad fija de bytes para el campo. Si el dato no ocupa todos los bytes asignado al campo, se produce fragmentación interna, desperdiciando espacio de memoria. La ventaja es que conocemos donde comienza y donde termina físicamente cada campo dentro del archivo.

Campo de longitud variable: El campo ocupa la cantidad de bytes que necesita el dato. No se conoce de antemano la dimensión ocupada. El campo se ajusta al tamaño del dato. Ventaja: No se produce fragmentación interna. Desventaja: No se sabe con precisión donde comienza y donde termina cada campo.

Fragmentacion:

Para detallar el proceso de selección de espacio, es necesario avanzar en conceptos de fragmentación. Existen dos tipos de fragmentación: interna y externa.

Se denomina **fragmentación interna** a aquella que se produce cuando a un elemento de dato se le asigna mayor espacio del necesario.

Los registros de longitud fija tienden a generar fragmentación interna, se asigna tanto espacio como lo necesario de acuerdo con la definición del tipo de dato. Pero este espacio no siempre se condice con lo que realmente utiliza el registro. Por ejemplo, si un campo del registro es Nombre definido como un string[50], y el nombre en cuestión es “juan Perez”, solo se ocupan realmente 10 de los 50 lugares asignados.

Se denomina **fragmentación externa** al espacio disponible entre dos registros, pero que es demasiado pequeño para poder ser reutlizado.

Las dos fragmentaciones tienen que ver con la utilización del espacio. En el primer caso, fragmentación interna, se reserva lugar que no se utiliza; en tanto, la fragmentación externa se genera por dejar espacios tan pequeños que no pueden ser utilizados

Clave:

Es conveniente identificar un registro con una llave o clave.

Una clave es un concepto que permite la identificación de un registro en particular. Deben permitir generar orden en el archivo por ese criterio.

Un archivo puede tener N claves.

Clave univoca/primaria: Identifican un elemento particular dentro de un archivo. Este tipo de clave, no admite repeticiones. Ej: No puede existir dos alumnos por el mismo DNI, el DNI actua como una clave univoca/primaria.

Clave secundaria: Reconocen un conjunto de elementos con igual valor. Ej: Podemos reconocer todos los alumnos con un determinado nombre.

Performance de claves

En archivos serie(Acceso secuencual):

- Mejor caso: Leer 1 reg. Peor caso leer n registros.
- Promedio n/2 comparaciones.
- Es de O(n), porque depende de la cantidad de registros,

En archivo directo(Acceso directo):

- Requiere una sola lectura para traer el dato => O(1)
- Debe conocerse el lugar donde comienza el registro requerido. (debemos conocer su NRR)

NRR: Numero relativo de registro

- Indica la posición relativa con respecto al principio del archivo
- Solo aplicable con registros de longitud fija.

El acceso directo es preferible cuando se necesiten pocos registros específicos. Pero no siempre es el mas apropiado para la extracción de información. Es decir, es apropiado para recuperar un alumno por legajo, un alumno por dni, es adecuado cuando nos interesa un registro puntual.

Ahora, si yo necesito por alguna razón acceder a todos los registros, por ejemplo para sacar el precio total de todos los productos vendidos, la mejor forma es tener un acceso secuencial a los registros, ya que no nos importa un registro en especifico, sino TODOS los registros.

Archivos Estaticos y Volatiles.

Archivos Estaticos: Pocos cambios,no necesita de estructuras adicionales para agilizar cambios.Por ejemplo un archivo con las provincias de Argentina. En general tienden a ser pequeños.

Archivos Volatiles: Sometido a operaciones frecuentes, por ejemplo agregar,borrar,actualizar información. Su organización debe facilitar cambios rapidos. Se necesitan estructuras adicionales para mejorar los tiempos de acceso. Un ejemplo es un archivo de alumnos a la facultad el cual está sometido a operaciones frecuentes debido a la alta y baja de los alumnos.

Eliminacion de información en archivos:

En general, las organizaciones no eliminan información, el proceso de eliminar información no ocurre frecuentemente. Un dato viejo no es un dato sin un sentido, podría servir a futuro. Generalmente no es natural hoy en día borrar información.

Baja física: Se elimina el dato y el dato no se encuentra más almacenado físicamente. El archivo se achica al borrar el registro.

Borrar un registro de forma física es más lento pero el dato no se encuentra más almacenado físicamente, para borrar de forma física, debo tener en cuenta que dependiendo de donde se encuentre el registro, se debe hacer corrimientos para poder aprovechar el espacio. Si queremos agregar un dato, lo agregamos siempre al final.

Baja lógica: Eliminar el dato de forma lógica, es decir, marcar el dato como borrado, el dato no se encuentra mas visible al usuario,pero sigue estando almacenado físicamente. Un ejemplo de esto es cuando se utiliza un campo de activo por ejemplo para un producto o cualquier otra entidad. El espacio de memoria va a seguir estando ocupado porque el dato se va a seguir encontrando, pero al estar como marcado, debe ser invisible al usuario.

Poner una marca de borrado es rápido pero el dato va a seguir ocupando almacenamiento.Al momento de agregar un dato, lo debemos agregar en aquel/aquellos registros que se encuentren marcados como baja lógica, para aprovechar el espacio de memoria.

Compactación: El objetivo de la compactación es recuperar el espacio.

Aprovechamiento de espacio: Cuando trabajamos con bajas lógicas, debemos aprovechar el espacio de aquellos registros que están marcados como borrados, pero no siempre se puede, ya que si tenemos un espacio disponible de 50bytes y el nuevo dato tiene 80bytes, no va a ser posible reutilizar el espacio.

Diferencia entre acceso y comparación:

Si tengo un archivo con 1Millon de registros, voy a hacer 1Millon de lecturas para recuperar los datos, y voy a hacer 1Millon de comparaciones para encontrar el dato (hablando del peor de los casos).

Ahora, el hecho de hacer 1 millon de lecturas, no corresponde a que se hagan un millón de accesos a disco, ya que al momento de hacer una lectura sobre un archivo, nos traemos al buffer más de un registro del archivo.

Cuantos accesos reales voy a hacer a disco? Depende de nuestra cantidad de registros y de la cantidad de registros que pueda almacenar el buffer.

Ejemplo: Tenemos 1Millon de registros, y en cada buffer caben mil registros, cuantos buffers debo llenar para asegurarme que el registro que yo quiero leer está en ese buffer?

Teniendo el cuenta el peor de los casos hacemos: Cant registros almacenados/cant registros del buffer => 1.000.000 / 1.000 = 1.000 accesos a disco para recuperar la información.

Fragmentación

Fragmentación interna: Ocurre cuando se desperdicia espacio en un registro, se le asigna el lugar al dato pero no lo ocupa totalmente.

Los registros de longitud variable evitan el problema de fragmentación interna.

Técnicas para aprovechar el espacio cuando ocurre fragmentación interna

Primer ajuste: Selecciona y asigna la primer entrada disponible donde pueda almacenarse el registro. Rápida, eficiente y puede generar fragmentación interna

Mejor ajuste: Elige la entrada que mas se aproxime al tamaño del registro y se le asigna completa. Se genera la menor fragmentación interna posible.

Peor ajuste: Selecciona la entrada mas grande para el registro y se le asigna solo al espacio necesario, el resto queda libre para otro registro.

Búsqueda en archivos:

Como medir el costo de búsqueda de información?

Cantidad de comparaciones (operaciones en memoria)

Cantidad de acceso (Operaciones en disco)

Archivo serie: Ineficientes si están desordenados, ya que tienen orden de búsqueda lineal. Si el archivo esta ordenado, se puede hacer una búsqueda binaria en este tipo de archivos, es una búsqueda mas eficiente. Ahora, si el archivo secuencial no está ordenado, la búsqueda es de O(n). El costo de hacer una búsqueda binaria es mantener el archivo ordenado.

Archivos indizados: Son mas eficientes que los archivos serie, se utilizan índices/claves

Buscar un registro es mas rápido si conocemos el NRR, se hace un seek a esa posición y se recupera la información. Los archivos director permiten trabajar de esa manera.

https://www.youtube.com/watch?v=40igxVUjiWM&list=PLaIJ3UPfCo7sYKD5SAJ7_a-QI19Fnmer4&index=2 En este video se explica el tema de ordenar un archivo.

Formas de ordenar un archivo:

- Ordenarlo en disco: Muy ineficiente, se necesitan !n (n factorial) lecturas, teniendo en cuenta que el acceso a disco es costoso, este método es muy ineficiente.
- Llevar el archivo en RAM: eficiencia 2n (leer cada registro, ordenarlo en ram,escribir cada registro.)
- Llevar las claves a RAM: Eficiencia 3n. Usado cuando no entraba todo el archivo en memoria RAM, y solo se lleva a RAM la clave, el cual es el criterio por el cual se ordena el vector.

  Para llevar las claves a RAM se hacen N lecturas, pero para hacer una escritura, se debe hacer una escritura indirecta, ya que en el primer registro solo tenemos la clave.y no tendríamos el registro completo. Entonces claves en RAM lleva:
  N lecturas, N escrituras pero para escribir se debe leer el registro completo, entonces cada escritura, requiere una lectura del archivo viejo.
  Este método de llevar las claves a RAM, es 50% mas ineficiente que llevar el archivo a RAM, pero debemos tener en cuenta que no todos los archivos completos entran en RAM, por ende, si solo llevamos las claves es una buena alternativa, ademas la eficiencia del algoritmo de llevar las claves a ram (3n), no se puede comparar con lo costoso que es trabajar con el archivo en disco (!n)

  Que pasa si un archivo es demasiado grande para caber en RAM?

  Se parte el archivo, se ordena cada parte del archivo y después se juntan las partes ordenadas (merge).

Clase 5: Busqueda de datos – Indices

La búsqueda de información siempre debe minimizar la cantidad de accesos.

La búsqueda secuencial es muy poco eficiente:O(n/2)

Busqueda binaria es eficiente pero es costoso mantener el archivo ordenado: O (log n)

La idea es crear una estructura que permite ver a la información como si estuviese ordenada, aunque no lo esté físicamente. Esta estructura auxiliar, la vamos a llamar índice.

Indices para claves candidatas: Las claves candidatas son claves que no admiten repeticiones de valores para sus atributos, similares a una clave primaria, pero que por cuestiones operativas no fueron seleccionadas como clave primaria. Entonces, una clave primaria, fue una clave candidata, pero no toda clave candidata, es necesariamente una clave primaria, si no que se debe elegir.

Indices secundarios: Surgen por que no es muy común realizar búsquedas por claves primarias, por ejemplo si queremos buscar una canción, es mas fácil buscarla por nombre , pero el nombre podría repetirse para distintas canciones, por eso no es posible pensar el nombre de una canción como una clave primaria. La clave que soporta varios valores repetidos se denomina clave secundaria. **Defiinicion:** Un índice secundario es una estructura adicional que permite relacionar una clave secundaria con una o mas claves primarias.

Definicion de índice:

Es una herramienta porque encuentra registros en un archivo de una forma mas eficiente, consiste de un campo de llave/clave y un campo de referencia.

Es una tabla que opera con un procedimiento que acepta información acerca de ciertos valores de atributos como entrada (llave) y provee como salida información que se encuentra en el archivo. Los índices nos permiten realizar búsqueda binaria.

El índice se puede implantar tanto sobre registros de longitud fija como registros de longitud variable. La estructura mas simple para implementar un índice es un árbol, especialmente árbol binario, porque nos permite realizar búsquedas binaria. De esta manera, con el índice podemos imponer orden en un archivo sin que realmente el archivo este ordenado.

Como implantar índices?

Cuando empezamos a trabajar con índices, vamos a tener al menos dos archivos: Un archivo de datos, y tantos archivos de índices como sean necesarios. Para que sea efectivo, debemos cargar el índice en memoria ram, ya que las operaciones sobre memoria ram se miden en nanosegundos, mientras que las operaciones de los discos en memoria ram se mide en milisegundos. El índice se debe modificar luego de una alta o baja.

Agregar nuevos registros: Implica agregar al archivo de datos y al archivo de índices, se agrega al final en el archivo de datos, y se reordena el índice en memoria ram.

Eliminar un registro: Borrarlo del archivo de datos y sacarlo del índice (en ambos archivos se pueden utilizar las técnicas para borrar información que ya vimos anteriormente.)

Actualización de registros:

- Si la modificación altera la clave: Si se modifica la clave, seguro hay que modificar el archivo de datos, pero además se debe modificar el índice.
- Si la modificación no altera la clave: En archivos con registros de longitud fija, si se modifica un registro, no hay cambio en el índice, por lo tanto, el registro se almacena en la misma posición física. Si el registro cambia de longitud se reubica en el archivo de datos y además se debe guardar la nueva posición inicial en el índice.

En todo momento, el índice va a permitir búsqueda binaria, es de longitud fija, va a estar ordenado. Al permitir búsqueda binaria, encontrar la información se logra con una performance log2. Si puedo almacenar el índice en RAM, aun es más eficiente.

Que pasa si los índices son demasiado grandes para entrar en memoria RAM?

Solución: El índice en todo momento debe estar sobre disco, si en algún momento se pierde la energía no importa porque el índice se encuentra en el disco y no vamos a perder información, vamos a tener la ventaja de no perder información, pero al hacer las operaciones en disco, no es el mismo performance que hacerlas sobre RAM.

Si el índice va a estar en disco, una solución es utilizar arboles, para realizar búsquedas eficientes. (Los arboles se pueden implementar sobre disco también.)

Al utilizar el disco para la persistencia de datos, la eficiencia de utilizar índices sigue siendo log2, pero con el costo de acceso al disco (en milisegundos).

Indices primarios: Cada clave es univoca, no puede repetirse.

Indices secundarios: Cada clave no es univoca, por ende pueden repetirse.

Una clave primaria es una clave univoca que fue elegida como primaria. Cada archivo tiene una y solo una clave primaria. Tengo dos claves univocas (que no se repiten): Legajo y DNI, cual elijo? Cualquiera de las dos, dependiendo de mi problema.

Clase 6 parte 1: Arboles

Los árboles no tienen que ser necesariamente binarios. Que un árbol sea binario nos proporciona una búsqueda eficiente, pero no necesariamente todos los arboles deben ser binarios, tenemos el caso de los arboles multicamino.

Arbol multicamino: De cada nodo padre pueden salir mas de dos hijos.

Tenemos distintos tipos de arboles : B, B+, b* (toos los anteriores son balanceados), Binarios, Multicaminos, AVL

Cuando hablamos de árbol B, es árbol Balanceado, no hablamos de árbol Binario.

Que es un árbol binario:

Estructura de datos donde cada nodo tiene dos sucesores, a izquierda y a derecha.

Un árbol binario puede implantarse sobre disco? SI.

Arbol balanceado: Un árbol está balanceado cuando la altura de la trayectoria mas corta hacia una hoja no difiere de la altura de la trayectoria mas grande. El inconveniente de los arboles binarios es que se desbalancean fácilmente.

Arboles AVL (Arbol binario balanceado en altura): es un árbol en el que las inserciones y eliminaciones se efectúan con un minimo de accesos.En un árbol AVL , si yo miro **el trayecto mas largo hacia un nodo terminal y lo resto al trayecto mas corto a un nodo terminal**, **la diferencia no puede ser mayor a uno.**

Para que sirve un árbol AVL? Si construyo el árbol AVL con un algoritmo especifico, logramos que el árbol se mantenga nivelado/balanceado, entonces la eficiencia se convierte en algo parecido a O (log n) y esta estructura pasa a ser una buena opción por que el árbol AVL no se desbalancea mucho.

Arboles AVL y Binarios (Características y conclusiones):

Estructura que debe ser respetada, mantener árbol, rotaciones restringidas a un area local del árbol.

- Binario => Busqueda: Log2 (N+1)
- AVL => Busqueda 1.44 log2(N+2)
  Ambas performance por el peor caso posible.

Arboles Binarios Paginados

Cuando se transfiere desde o hacia el disco rigido, dicha transferencia no se limita a un registro, sino que son enviados un conjunto de registros, es decir, se envían los registros que quepan en el buffer de memoria. Las operaciones de lectura y escritura de un archivo utilizando buffers presentan una mejora de performance. Esto es útil cuando se genera un archivo que contiene un árbol binario, dicho árbol se divide en paginas, es decir se pagina y cada pagina tiene un conjunto de nodos, los cuales están ubicados en direcciones físicas cercanas. De esta manera, en una transferencia de datos, no se accede a disco para transferir unos pocos bytes, sino que se transfiere la pagina completa. **Esta organización reduce el numero de accesos a disco necesarios para poder recuperar información.** Al dividir un árbol binario en paginas, es posible realizar búsquedas mas rapidas de datos almacenados en memoria secundaria.

Arboles balanceados: Son arboles multicamino con una construcción especial en forma ascendente que permite mantenerlo balanceado a bajo costo. Los arboles balanceados, al menos se llenan a la mitad.

- **Propiedades de un árbol B :**
  - Ningun nodo tiene mas de M hijos
  - Al menos se llena la mitad (1/2)
  - Cada nodo menos raíz y terminales, tienen como minimo M/2 hijos
  - La raíz tiene como minimo 2 hijos o nnguno.
  - Todos los nodos terminales están a igual nivel.

Que es el orden del árbol? La máxima cantidad de descendientes (hijos) de un nodo. Es decir, el orden del árbol determina la cantidad de hijos. Ningun nodo puede tener mas de N hijos, si un árbol es de orden 5, puede tener como máximo 4 claves y como máximo 5 hijos.

Performance de búsqueda en un árbol:

  - Si el árbol es binario y esta balanceado, la performance es Log n
  - Si el árbol esta balanceado::
    - Mejor caso 1 lectuura
    - Peor caso: h lecturas (con h altura del arbol)
    - Cual es el valor de h?
      - Si el árbol es de orden M, el numero de elementos del árbol es N, hay N+1 punteros en nodos terminales.

Arboles B*: Creamos arboles que se van a llenar al menos 2/3. De esta manera tenemos nodos mas llenos. La misma cantidad de elementos se va a desparramar en menos nodos, por ende, utilizar un árbol B* puede tener altura menor a un árbol B. Al tener menor altura, podemos recuperar información con menos accesos.

Propiedades:

- Cada pagina tiene máximo M descendientes
- Los nodos como minimo se llenan 2/3
- Todas las hojas aparecen en igual nivel

Conclusión entre arboles B y B*: Los arboles B se llenan al menos la mtiad (1/2), mientras que los arboles B* se llenan al menos 2/3. Por ende, en todo momento los arboles B* van a tener altura menor que el mismo árbol si fuese B. Un árbol con menor altura, significa una mayor eficiencia en operaciones de búsqueda. La raíz de un árbol B y B* va a ser igual.

Arbol Balanceados B+: Consiste en un conjunto de grupos de registros ordenados por clave en forma secuencial, con un conjunto de índices que proporciona el acceso rápido a los registros. Una característica es que los nodos no terminales no tienen datos, sino puntero a los datos.

Propiedades:

- Cada pagina, menos la raíz y las hojas tienen entre M/2 y M hijos
- En los nodos no terminales no vamos a tener elementos, sino vamos a tener una copia de los elementos.
- En los nodos terminales vamos a tener realmente el dato.

Ventajas de los arboles B+: Proporcionan acceso secuencial de manera rápida y por lo tanto eficiente. Mientras que los arboles B, no son eficientes en búsqueda secuencial.

Los arboles B+ son los arboles mas utilizados y los mas eficientes.

Preguntas sobre arboles B+:

Es verdad que los arboles B+ pueden llegar a tener mas altura que los arboles B?

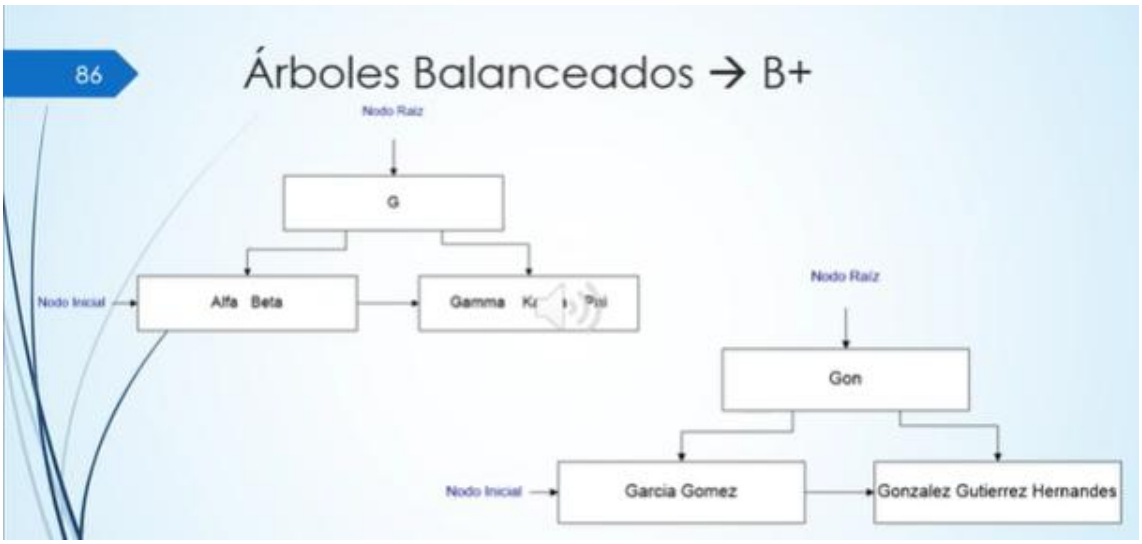
Si, porque en B+ se suben la copia de los separadores, por lo que **PUEDE** significar que la altura del árbol sea mas grande.

Consumimos mas espacio en el nodo por que se tiene que guardar el puntero ? Si.

Mantener un archivo ordenado FISICAMENTE es muy costos porque al desordenarlo se debía reacomodar la información, esto implica varios accesos a disco, lo cual el acceso a disco es bastante costoso.

Arbol B+ de prefijos simple: Su finalidad es dotar de mas eficiencia a aquellos arboles B+ que manejan claves alfanumericas. Se utilizan no el separador total, sino se utiliza un separador menor, de manera que podemos seguir realizando búsquedas de manera eficiente, pero almacenando la menor cantidad posible de información en el separador.

Conclusion: Los arboles B+ de prefijos simples, son arboles B+ en el cual el índice esta consituido por separadores mas cortos. El nodo separador puede tratarse como registro de longitud variable y puedo utilizar el espacio de forma eficiente.



Hashing: Se llevan a cabo mediante archivo directos, c Se recupera información con un solo acceso a disco.

Primer definición: Técnica para generar una dirección base única para una llave dada. La dispersión se usa cuando se requiere acceso rápido a una llave.

Segunda definición: Técnica que convierte la llave del registro en un numero aleatorio en el que sirve después para determinar donde se almacena el registro.

Tercer definición: Técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en dirección de almacenamiento.

Conclusión: La única forma de recuperar un registro en un solo acceso es saber dónde está el registro. Entonces se define una técnica para saber la dirección donde se va a guardar el registro, se lleva a cabo aplicando una función de hash para generar direcciones en teoría uniformes y aleatorias, aunque en lo práctico no siempre genere direcciones únicas.

Atributos de hash:

- No requiere almacenamiento adicional para índice.
- Facilita la inserción y eliminación rápida de registros
- Encuentra registros con muy pocos accesos al disco en promedio.
- NO PODEMOS USAR REGISTROS DE LONGITUD VARIABLE !
- NO PUEDE HABER ORDEN FISICO EN LOS DATOS.
- NO PERMITE LLAVES DUPLICADAS.
- NO PERMITE ACCESO SECUENCIAL A LOS DATOS.

Conclusion: si queremos realizar una búsqueda secuencial en archivos directos (utilizando hashing), no es una forma eficiente. Deberiamos usar archivos secuenciales indizados para poder lograr una búsqueda secuencial de forma eficiente.

Que es una función de hash?

Caja negra que a partir de una clave de entrada, se obtiene la dirección donde debe estar el registro. La diferencia que tiene con índices es que no hay relación entre llave y dirección. Ademas, dos llaves distintas pueden transformarse en iguales direcciones (se genera colisión.)

Para determinar la dirección: La clave se convierte en un número **casi** aleatorio.

Colision: Situacion en la que un registro es asignada a una dirección que esta utilizada por otro registro.

Overflow: situación en la que un registro es asignada a una dirección que esta utilizada por otro registro y no queda espacio para este nuevo.

Lo ideal seria tener una función de hash, que genere colisiones para aprovechar el espacio disponible, pero que no genere o que genere la menor cantidad de overflows posibles.

Densidad de empaquetamiento: Es la proporción entre la cantidad de registros almacenados y la opacidad total de almacenamiento de una estructura de datos. Es una característica propia de hashing estatico. En hashing dinamico/hashing extensible no hay DE, ya que se va ajustando dinámicamente el almacenamiento.

$$\text{Densidad de empaquetamiento} = \frac{\text{Número de registros ocupados}}{\text{Número total de espacios disponibles}}$$

Si la densidad de empaquetamiento es 0, significa que la estructura esta completamente vacia.

Si la densidad de empaquetamiento tiende a 1, significa que el almacenamiento esta casi lleno, hay poco espacio libre, lo que puede generar mayor cantidad de colisiones y overflow.

Tratamiento de colisiones con overflow.

Saturacion progresiva: Cuando ocurre una colision, se busca el siguiente espacio disponible siguiendo una secuencia predefinida. Es decir, se busca la próxima dirección disponible para almacenar el dato.

Saturacion progresiva encadenada: Similar a la saturación progresiva pero cada celda de la tabla puede actuar como nodo de una lista enlazada. Cuando ocurre una colision, el nuevo elemento se encadena a la lista de la posición hash.

Doble dispersión: Utiliza dos funciones de hash diferentes:

- La primera determina la posición inicial en la tabla
- La segunda define el paso de exploración en cada colisión.

Area de desborde separada: Se divide la tabla en dos partes:

- Area principal: Donde se insertan los registros sin colision
- Area de desborde: Donde se almacenan los registros que generaron colision.

Cuando ocurre una colision, el registro se mueve a la zona de desborde.

Apuntes de teóricos:

Diferencia entre archivos secuenciales indizados vs archivos directos:

- Archivos secuenciales: 2 Accesos a disco en el mejor de los caos. Se necesita una estructura adicional. El archivo de datos no se organiza, si no que se organiza sus índices.
- Archivos director: 1 Acceso a disco. No puede haber estructuras adicionales. Se organiza EL archivo de datos. Solo puede organizarse por un único criterio, ese criterio debe ser la clave primaria.

Caracteristicas de hash:

- NO PODEMOS USAR REGISTROS DE LONGITUD VARIABLE !
- NO PUEDE HABER ORDEN FISICO EN LOS DATOS.
- NO PERMITE LLAVES DUPLICADAS.
- NO PERMITE ACCESO SECUENCIAL A LOS DATOS.
- Si la dirección esta ocupada, ocurre colision, esa colision PUEDE producir overflow.

Conclusion: si queremos realizar una búsqueda secuencial en archivos directos (utilizando hashing), no es una forma eficiente. Deberiamos usar archivos secuenciales indizados para poder lograr una búsqueda secuencial de forma eficiente.

En hashing extensible / hashing dinamico, NO hay DE

Un índice se puede desbalancear? NO. El índice no se desbalancea, lo que se desbalancea es el árbol.

Clave primaria, clave candidata y clave univocas no se repiten.

Preguntas sobre arboles B+:

Es verdad que los arboles B+ pueden llegar a tener mas altura que los arboles B?

Si, porque en B+ se suben la copia de los separadores, por lo que **PUEDE** significar que la altura del árbol sea mas grande.

Consumimos mas espacio en el nodo por que se tiene que guardar el puntero ? Si.

Los arboles B+ son los mas utilizados y los mas eficientes porque proporcionan acceso secuencial de manera rápida.

Preguntas para clase de consulta:

Un índice, se puede implantar con un archivo de longitud variable? O tienen que ser si o si con archivo de longitud fija? Por ejemplo los arboles B+ de prefijos simples, utilizan registros de longitud variable....?