Fundamentos de Organización de Datos

Autor: Massimo Parzanese

Profesor/es: Rodolfo Bertone y Pablo Thomas

**Año 2024**

**CLASE 1**

**¿Qué es una base de datos?**

Definición:

Es una colección de datos relacionados.

Es una colección de archivos diseñados para servir a múltiples aplicaciones.

Un dato representa hechos conocidos que pueden registrarse y que tienen un resultado implícito.

Propiedades:

* Representa algunos aspectos del mundo real, a veces denominado Universo de discurso
* Una BD 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.
* Una BD se diseña, construye y completa de datos para un propósito específico.

**Archivos**

Definiciones:

* Colecciones de registros guardados en almacenamiento secundario
* Colección de registros que abarcan entidades con un aspecto común y originados para algún propósito particular.

Organización:

**Secuencia de bytes**:

* Archivos de texto.
* No se puede determinar fácilmente comienzo y fin de cada dato.

**Registros y campos:**

* Campo: Unidad más pequeña, lógicamente significativa de un archivo.
* Registro: Conjunto de campos agrupados que define un elemento de un archivo.

Acceso:

* Secuencial Físico: Acceso a los registros uno tras otro en el orden en el que están guardados
* Secuencial Indizado(lógico): Acceso a los registros de acuerdo al orden establecido por otra estructura. Ejemplo: índice temático de un libro.
* Directo: Se accede a un registro determinado sin necesidad de haber accedido a los predecesores.

Tipos de archivos:

De acuerdo a su forma de acceso hay varios…

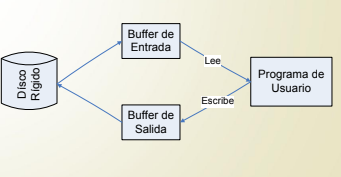
**Serie:** Cada registro es accesible solo luego de procesar su antecesor, simples de acceder (acceso secuencial físico).

**Secuencial:** Los registros son accesibles en orden de alguna clave (Acceso secuencial indizado/secuencial lógico).

**Directo:** Se accede al registro deseado (directo).

**Buffers**

* Memoria intermedia entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados.
* Ocupan lugar en RAM.
* El sistema operativo es el encargado de manipular los buffers.



**CLASE 2**

**Algorítmica clásica**

**Agrupan elementos en un archivo existente:**

* Se abre el archivo
* Seek en la última posición
* Agrego con write

**Precondiciones:**

* Se procesa un solo archivo
* Ya tiene información

Actualización Maestro Detalle:

* Se denomina maestro al archivo que resume un determinado conjunto de datos.
* Se denomina detalle al que agrupa la información que se utilizará para modificar el contenido del maestro.
* En general: Un Maestro y N detalles.

Precondiciones: Ambos archivos (Maestro y detalles) están ordenados por algún criterio (puede no estarlo).

Corte de control:

* Es un problema clásico en el manejo de una BD.
* Si bien los DBMS lo manejan diferente, veremos la algorítmica clásica de los mismos.

Merge:

Consiste en resumir la información de archivos similares bajo una única estructura.

Puede hacerse tanto con archivos ordenados por uno, o varios criterios. Como con archivos totalmente desordenados.

**CLASE 3**

**Almacenamiento Secundario:**

* Necesita más tiempo para tener acceso a los datos que en RAM.
* Su acceso es tan “Lento” que es imprescindible enviar y recuperar datos con inteligencia.
* Al buscar 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.
* La información está organizada en archivos.

**Archivo Físico:**

* Que existe en almacenamiento secundario.
* Es el archivo como lo conoce el S.O. y que aparece 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.

Viaje de un byte

* Escribir un dato en un archivo( Write(archivo, variable) ).
* Quienes están involucrados: Administrador de archivos, Buffer de E/S, Procesador de E/S, Controlador de Disco.

Administrados de Archivos: Conjuntos de programas del S.O. que tratan aspectos relacionados con dispositivos de E/S.

Maneja 2 capas:

* Capas superiores: Aspectos lógicos de datos(tabla). Establece si las características del archivo son compatibles con la operación deseada.
* Capas inferiores: Aspectos físicos. Determina donde se guarda el archivo. Si el sector está ubicado en RAM se utiliza, como contrario, debe traerse previamente.

Buffer de E/S: agiliza la E/S de datos. Es una estructura de información intermedia donde se transfiere la información desde el disco y hacia memoria RAM para que el programa la use y viceversa. Donde el programa coloca la info antes de ser transferida a disco. Implica trabajar con grandes datos en RAM y reduce el acceso de almacenamiento secundario. En cada transferencia se transfieren N registros/datos.

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

Una operación de write escribe sobre el Buffer de salida del archivo. Cuando el buffer se llena, el procesador de E/S toma el control y se encarga de realizar la transferencia.

Controlador de disco:

* Encargado de controlar la operación de disco.

**Capas de 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 Administrados de Archivos
* El Adm. busca el archivo en su tabla de arch 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 que está en el Buffer y graba el dato donde va dentro del sector en el Buffer.
* El Adm. De arch da instrucciones al procesador de E/S (donde está el byte en RAM y en que parte del disco debe 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 de disco.

**Archivos estructurados**

Poseen registros y campos

Campos:

* Longitud predecible: longitud fija.
* Longitud Variable: posee un indicador de longitud, que dice cuántos bytes tendrá ese campo. A su vez, tiene un delimitador a final de cada campo.

Registros:

* Longitud predecible (en cant. de bytes o cant. de campos)

• Campos fijos o variable.

* Longitud variable

• Indicador de longitud (al comienzo, indica la cant. de

bytes que contiene)

• Segundo archivo (mantiene la info de la dirección del

byte de inicio de cada registro)

• Delimitador (carácter especial no usado como dato)

**Claves**

Clave:

* Se concibe al registro como la cantidad de info que se lee o escribe
* Nuestro objetivo es resolver de manera eficiente la búsqueda de información.
* Es conveniente identificar un registro con una llave o clave que se base en el contenido del mismo
* Permite la identificación de un registro
* Debe permitir generar orden en el archivo con ese criterio

Pueden ser:

* Unívoca/ primaria: identificar un elemento particular dentro de un archivo.
* Secundaria: Reconocer un conjunto de elementos con igual valor

Forma canónica: Forma estándar para una clave, puede derivarse a partir de reglas bien definidas. Dicha representación está ajustada a una regla. Hay que definir un formato general que se va a respetar a la hora de búsqueda.

Estudio de performance:

* Punto de partida para futuras evaluaciones
* Caso promedio

Un archivo puede estudiarse acorde a varias posibilidades:

* formas de acceso
* cantidad de cambios

Forma de acceso:

* Serie: Cada registro es accesible solo luego de procesar su antecesor, son de simple acceso.
* Secuencial: Los registros son accesibles en orden de alguna clave
* Directo: Se accede al registro deseado.

Cantidad de cambios:

* estáticos 🡪 pocos cambios: puede actualizarse en procedimiento por lotes
* volátiles 🡪 sometido a operaciones frecuentes:
  + Agregar/borrar/actualizar.
  + Su organización debe facilitar cambios rápidos.
  + Necesita estructuras adicionales para mejorar tiempos de acceso.

Operaciones:

* Altas
* Bajas
* Modificaciones
* Consultas
* Como influye los reg. de long variable y fija

Eliminación:

* Baja lógica
* Baja física

Cualquier estrategia de eliminación de registros debe proveer alguna forma para reconocerlos una vez eliminados (ej: colocar una marca especial en un registro eliminado).

Con dicho criterio, se puede anular la eliminación fácilmente.

Los programas que usan archivos deben incluir cierta lógica para ignorar los registros eliminados.

Compactación:

* Recuperar espacio
* La forma más simple es copiar todo en un nuevo archivo a excepción de los eliminados 🡪 baja física.
* Frecuencia
  + Tiempo
  + Ante la necesidad de espacio

Aprovechamiento de espacio:

* Registros de longitud fija 🡪 es necesario garantizar:
  + Marca especial: registros borrados 🡪 baja lógica
* Registros de longitud variable 🡪 los nuevos elementos deben “caber” en el lugar.

Recuperación de espacio con reg. de longitud fija:

* Utilizar lista o pila(encabezado)

Recuperación de espacio con registros de longitud variable:

* El problema de los registros de longitud variable está en que no se puede colocar en cualquier lugar, para poder ponerlo debe caber, necesariamente.
* Lista: No se puede usar NRR como enlace. Se utiliza un campo binario que explícitamente indica el enlace.
  + Cada registro indica en su inicio la cantidad de bytes.
  + Reutilización: buscar el registro borrado de tamaño adecuado.
  + Como se necesita buscar, no se puede organizar la lista de disponibles con una pila.

Fragmentación en aprovechamiento de espacio:

* **Interna:**
  + Ocurre cuando se desperdicia espacio en un registro, se le asigna el lugar, pero no lo ocupa totalmente.
  + Ocurre en general con registros de longitud fija
  + Registros de longitud variable evitan el problema
  + Solución 🡪 el “residuo” una vez ocupado el espacio libre, pasa a ser un nuevo registro libre. Si este es muy chico 🡪 fragmentación interna
* **Externa:**
  + Ocurre cuando el espacio que no se usa es demasiado pequeño como para ocuparse.
  + Soluciones:
    - Unir espacios libres pequeños adyacentes para generar un espacio disponible mayor
    - Minimizar la fragmentación, eligiendo el espacio más adecuado en cada caso.
    - Estrategias de colocación en registros de longitud variable:
      * Primer ajuste
      * Mejor ajuste
      * Peor ajuste

Primer ajuste: Se selecciona la primera entrada de la lista de disponibles, que pueda almacenar al registro, y se le asigna al mismo.

Mejor ajuste: Elige la entrada que más se aproxima al tamaño del registro y se le asigna completa.

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.

**CLASE 4**

Archivos

Búsqueda:

* Secuencial
* Directa
* Binaria

Búsqueda de información(costo)

* # comparaciones
* Se pueden mejorar con algoritmos más eficientes.
* # accesos

Buscar un registro:

* + rápido si conocemos el NRR (directo).
* Secuencia debe buscarse desde el principio.
* Trataremos de incorporar el uso de llaves o claves.

Búsqueda binaria 🡪 precondiciones:

* Archivo ordenado por algún criterio
* Registros de longitud fija

Búsqueda 🡪 partir el archivo a la mitad y comparar clave

* Puedo acceder al medio por tener long fija.
* Si N es el # de registros, la performance será log2(N).
* Se mejora la performance de la búsqueda secuencial.

Como clasificar(ordenar) un archivo:

* En RAM.
* Claves en RAM.

**Llevar el archivo a RAM**

¿Eficiencia? 2N

**Llevar las claves a RAM**

¿Eficiencia? 3N

**¿Si no cabe ni el archivo ni las claves en RAM?**

* Partir el archivo
* Ordenar cada parte
* Juntar las partes ordenadas por un Merge

Algunas conclusiones:

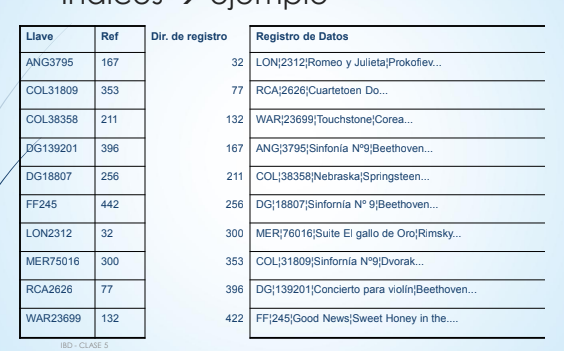
* Búsqueda binaria mejora la búsqueda secuencial. Aunque, dicha búsqueda presenta algunos problemas:
  + # accesos baja, pero no llega a uno.
    - Acceder por el NRR requiere una lectura.
  + Clasificación en RAM solo para archivos más pequeños.
  + Costo de mantener orden.
* Mejorar el método de ordenación:
  + No reordenando todo el archivo.
  + Reorganizando con métodos más eficientes(árboles).

**CLASE 5**

Índices

Definición: Es una herramienta para encontrar registros en un archivo. Consiste de un campo llave(búsqueda) y un campo referencia que indica donde encontrar el registro dentro del archivo de datos.

**Ejemplo:**



Índices 🡪 como implantarlos?

* Índice en memoria (búsqueda binaria + rápida,

comparada con archivos clasificados)

* 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. Se almacena en un

arreglo)

* Reescritura del archivo de índice (cambios 🡪 reescribir)

Agregar nuevos registros

* Implica agregar al archivo de datos y al archivo de índices
* Archivo de datos: copiar al final (se debe saber el NRR (fija) o distancia en bytes (variable) para el índice)
* Índice ordenarse con cada nuevo elemento en forma canónica (en mem.), setear 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 (¿que pasa con el índice?)
  + 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 (¿qué sucede?)
  + Se modifica el archivo de datos
  + Se debe actualizar y reorganizar el archivo de índices
  + Cómo simplificar  Modificar = Eliminar + Agregar (ya vistos)

Índices 🡪 Resumen

Ventajas:

* Se almacena en memoria principal
* Permite búsqueda binaria
* El mantenimiento es menos costoso

Desventajas:

* Si no caben en RAM.
* Reescritura del archivo índice
* Persistencia de datos.

Índices 🡪 Persistencia de datos

Organización por dispersión(Hashing)

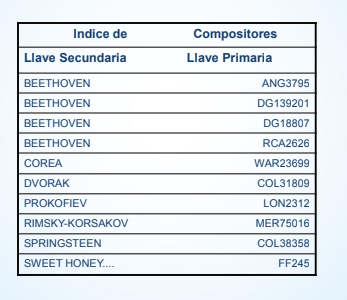
Uso de árboles

Índices grandes para entrar en memoria 🡪 Soluciones

Índices Secundarios

* No sería natural solicitar un dato por clave
* En su lugar se utiliza normalmente un campo más fácil de recordar ( ej: buscar una canción por su título o por su compositor)
* Este campo es un campo que pertenece a una llave secundaria porque puede repetirse
* Las claves secundarias se pueden repetir
* El índice secundario relaciona la llave secundaria con la llave primaria

Ejemplo:



**CLASE 6**

Árboles

* Binarios
* AVL
* Multicamino
* Balanceados

Problemas con los índices:

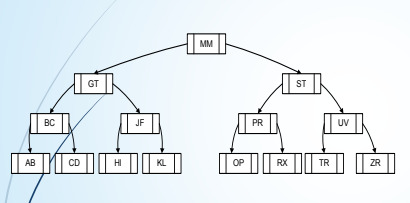
* Las búsquedas binarias aún son costosas
* Mantener los índices ordenados es costoso

Los árboles son estructuras de datos que permiten localizar en forma más rápida información en un archivo, tienen intrínsecamente búsqueda binaria

Árboles Binarios:

* Estructuras de datos donde cada nodo tiene a lo sumo dos sucesores, a izquierda y a derecha.
* Inconveniente de los binarios: se desbalancean fácilmente.

Ejemplo de árbol binario:



Árboles AVL:

* Árbol binario balanceado en altura, en el que las inserciones y eliminar se efectúan con un mínimo de accesos
  + Árbol balanceado en altura: Para cada nodo existe un límite entre la diferencia que se permite entre las alturas de los subárboles del nodo

Ejemplos de AVL:



Características y conclusiones de Árboles Binarios y AVL:

* Estructura debe ser respetada
* Mantener el árbol, rotaciones restringidas a un área local del árbol
  + Binario 🡪 Búsqueda: Log2(N+1)
  + AVL 🡪 Búsqueda: 1.44 log2(N+2)
  + Ambas performance en el peor caso posible

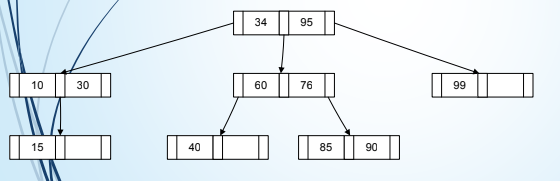
Árboles Binarios Paginados:

* Problemas de almacenamiento secundario, buffering, páginas de memoria, varios registros individuales, minimiza el número de accesos
* Problema: construcción descendente, ¿cómo se elige la raíz?, ¿cómo va construyendo balanceado?

Árboles multicamino:

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

Ejemplo:



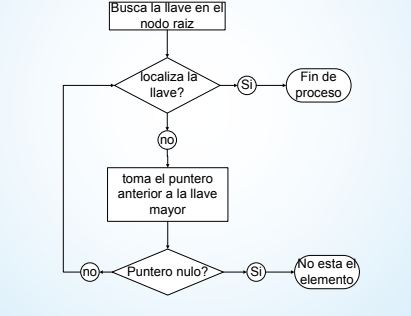
Árboles balanceados:

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

Propiedades de un árbol B de orden M:

* Ningún nodo tiene más de M hijos.
* C/ nodo (menos raíz y hojas) tienen como mínimo [M/2] hijos.
* La raíz tiene como mínimo 2 hijos, o ninguno.
* Todos los nodos terminales están a igual nivel.
* Nodos no terminales con K hijos tienen K-1 registros.

Búsqueda de información de Árbol B de forma gráfica:



Performance de búsqueda:

* Mejor caso: 1 lectura.
* Peor caso: h lecturas (h es la altura del árbol).
* ¿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 los hijos

h 🡪 [ 1 + log[M/2] ((N+1)/2) ]

Siendo [M/2] la base del logaritmo

Inserción(creación):

* Los registros e insertan en un nodo terminal
  + Casos posibles:
    - El registro tiene lugar en el nodo terminal (no se produce overflow).
    - El registro no tiene lugar en el nodo terminal(overflow): El nodo se divide y los elementos se reparten entre los nodos, hay una promoción al nivel superior, esta puede propagarse hasta la raíz

Performance de búsqueda:

* Mejor caso (sin overflow)
  + H lecturas
  + 1 escritura
* Peor caso (overflow hasta la raíz, aumenta en uno el nivel del árbol)
  + H lecturas
  + 2H+1 escrituras(dos por nivel mas raíz)

Eliminación:

* Siempre eliminar nodos terminales.
* Si se va a eliminar un elemento que no está en nodo terminal, hay que llevarlo a un nodo terminal.
* Posibilidades ante eliminación:
  + Mejor caso: borra un elemento del nodo y no se produce underflow, solo reacomodas.
  + Peor caso: Se produce underflow, # elementos < [M/2] - 1
* Dos soluciones:
  + Redistribuir
  + Concatenar

**Algunas definiciones:**

* Nodo adyacente hermano: Dos nodos son adyacentes hermanos si tienen el mismo padre y son apuntados por punteros adyacentes al padre.
* Overflow: Se quiere insertar un elemento en un nodo, y no es posible ya que en nodo no posee más espacio para el registro.
* Underflow: Se quiere eliminar un elemento de un nodo, y al realizar la eliminación, el nodo deja de cumplir una de las propiedades del árbol. Que sería la cantidad de elementos que debe poseer un nodo.

Redistribuir:

Cuando un nodo tiene underflow, puede trasladarse llaves de un nodo adyacente hermano (en caso de que tenga suficientes elementos).

Concatenación:

Si un nodo adyacente hermano está al mínimo (no le sobra ningún elemento) no se puede redistribuir, se concatena un nodo adyacente disminuyendo la # de nodos (en algunos casos la altura del árbol).

Performance de Eliminación:

* Mejor caso:
  + H lecturas
  + 1 escritura
* Peor caso:
  + 2H – 1 lecturas
  + H+1 escrituras

Algunas conclusiones:

* La redistribución podría posponer la creación de páginas nuevas.
* Se pueden generar árboles B más eficientes en términos de utilización de espacio.

Árboles B\*:

Es un Árbol B especial en que cada nodo está lleno por lo menos en 2/3 partes

Propiedades:

* Cada página tiene un máximo de M descendientes.
* La raíz tiene al menos dos descendientes (o ninguno).
* Cada pág., menos la raíz y hojas, tienen al menos [2M-1/3] descendientes
* Todas las hojas aparecen a 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

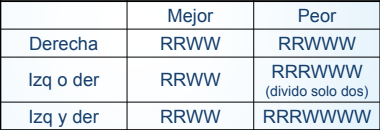
Operaciones de búsqueda:

Igual que un árbol B común

Operaciones de Inserción:

* Tres casos posibles
  + Derecha: redistribuir con nodo adyacente hermano de la derecha (o izquierda si es el último.
  + Izquierda: redistribuir con nodo adyacente hermano de la Izquierda (o der. si es el último)
  + Izquierda o derecha: si el nodo de la derecha está lleno se busca redistribuir con la izquierda, y viceversa.
  + Izquierda y derecha: Busca rellenar los tres nodos, estos tendrán ¾ parte llena

Costo de la redistribución:



Árboles B+:

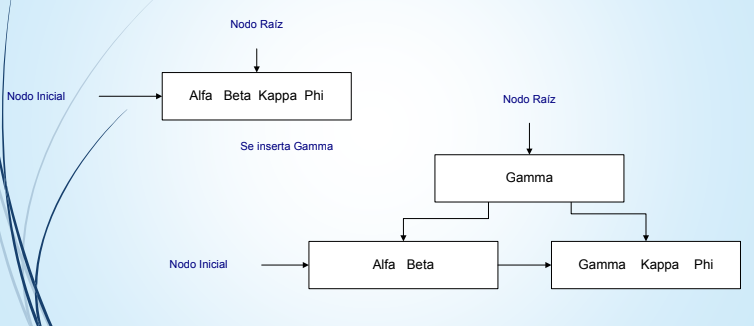
Consiste en un conjunto de grupos de registro ordenados por clave en forma secuencial junto con un conjunto de índices, que proporciona acceso rápido a los registros.

Los nodos no terminales no tienen datos, si no, punteros a los datos.

Propiedades:

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

Ejemplo:



Separadores:

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

Árboles B+ de prefijos simples:

* Árbol B+ en el cual el conjunto índice está constituido por separadores más cortos
* Los nodos no terminales pueden ser de diferente orden a los terminales
* Se pueden utilizar únicamente con claves alfanuméricas.

**CLASE 7**

Hashing(Dispersión)

Necesitaos un mecanismo de acceso a registros con una lectura solamente.

Claves primarias:

* No se repiten
* Resto de las claves actúan a través de ellas

Definición 1 Hash:

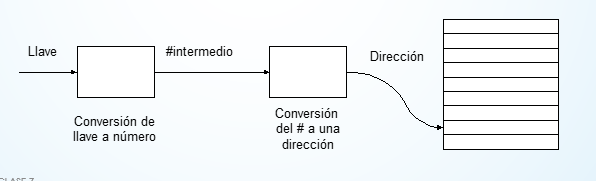
El hashing es una técnica para generar dirección base única para una llave dada. La dispersión se usa cuando requiere acceso rápido a una llave.

Definición 2 Hash:

Técnica que convierte la llave del registro en un número aleatorio, el que sirve después para determinar donde se almacena el registro.

Atributos del Hash:

* Facilita inserción y eliminación rápida de registros
* Encuentra registros con muy pocos accesos al disco en promedio



Costo del hash:

* No podemos usar registros de longitud variable
* No puede haber orden físico en los datos
* No permite llaves duplicadas

Para determinar la dirección:

* La clave se convierte en un numero casi aleatorio
* # se convierte en una dirección de memoria
* El registro se guarda en esa dirección
* Si la dirección está ocupada 🡪 colisión/overflow

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 ay relación aparente entre llave y dirección
  + Dos llaves distintas pueden transformarse en la misma dirección

Colisión:

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

Overflow:

Situación en la que un registro es asignado a una dirección que está utilizada por otro registro y no queda espacio para este nuevo.

Soluciones:

* Algoritmos de dispersión sin colisiones o que estas colisiones nunca produzcan overflow
* Almacenar los registros de alguna otra forma

Soluciones para colisiones:

* Esparcir registros: Buscar métodos que distribuyan los registros de forma más aleatoria posible.
* Una memoria adicional: distribuir pocos registros en muchas direcciones, baja la densidad de empaquetamiento:
  + Disminuye las colisiones y por ende el overflow
  + Desperdiciar espacio
* Colocar más de un registro por direcciones: direcciones con N claves, mejoras notables

Algoritmos simples de dispersión

Condiciones:

* Repartir los registros de forma uniforme
* Aleatoria (las claves son independientes, no influyen una sobre la otra)

Tamaño de nodos:

* Puede tener más de un registro
* A mayor tamaño:
  + Menor overflow
  + Mayor fragmentación
  + Búsqueda más lenta dentro del nodo

Densidad de empaquetamiento:

* Proporción de espacio del archivo asignado que realmente almacenan registros
* DE = números de registros del archivo / capacidad total del archivo
* Menor DE:
  + Menor overflow
  + Más desperdicio de espacio

¿Cómo podemos estimar el overflow?

* Es necesario analizar el comportamiento de un archivo directo
* Cuando encontrar un registro requiere de un solo acceso y cuando requiere más cantidad de accesos
* Estimar overflow:
  + Analizar probabilísticamente si la inserción de un registro genera o no colisión
  + Analizar si la colisión genera o no overflow
* Es necesario:
  + Conocer los elementos básicos de probabilidades
  + Vamos a utilizar la distribución de Poisson

Tratamiento de colisiones con Overflow

Hemos visto que él % se reduce, pero el problema se mantiene dado que no llegamos a 0

**Algunos métodos:**

* Saturación progresiva
* Saturación progresiva encadenada
* Doble Dispersión
* Área de desborde separado

Saturación progresiva:

* Cuando se completa el nodo, se busca el próximo hasta encontrar un nodo libre.
* Eliminación no debe obstaculizar la búsqueda.

Saturación progresiva encadenada:

* Similar a la saturación progresiva, pero los registros de saturación, se encadenan y “no ocupan” necesariamente posiciones contiguas.

Dispersión Doble:

* Saturación tiende a agrupar en zonas contiguas, búsquedas largas, cuando la densidad tiende a 0
* Solución: almacenar los registros de overflow en zonas no relacionadas
* Esquema con el cual se resuelven overflow aplicando una segunda función a la llave para producir N° C, él se suma a la dirección original tantas veces como sea necesario hasta encontrar una dirección con espacio

Encadenamiento de áreas separadas (o área de desborde separado):

* No utiliza nodos de direcciones para los overflow, estos serán nodos especiales
* Se mejora el tratamiento de inserciones o eliminaciones. Empeora el TAP
* Ubicación del desborde:
  + Cilindros de desborde
  + A intervalos regulares entre direcciones asignadas

Hash con espacio de direccionamiento estático:

* Necesita un numero de direcciones fijas virtualmente imposible
* Cuando el archivo se llena:
  + Saturación excesiva
  + Redispersar, nueva función, muchos cambios

Solución 🡪 Espacio de direccionamiento dinámico:

* Reorganizar tablas sin mover muchos registros
* Técnicas que asumen bloques físicos, pueden reutilizarse o liberarse

Hashing con espacio de direccionamiento dinámico

**Varias posibilidades:**

* Hash Virtual
* Hash dinámico
* Hash extensible

Hashing extensible

* Adapta el resultado de la función de hash de acuerdo al número de registros que tenga el archivo y de los nodos necesitados para su almacenamiento
* Función: Genera una secuencia de bits

**¿Cómo trabaja?**

* Se utilizan solo los bits necesarios de acuerdo a cada instancia del archivo
* Los bits tomados forman la dirección del nodo que se utilizará
* Se intenta insertar a un nodo lleno, deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nuevo, para ello se toma un bit mas
* La tabla tendrá tantas entradas(direcciones de nodos) como 2i(dos a la i), siendo i el N° de bits actuales para el sistema