Procesamiento de Consultas

Práctico 3: 11.6, 11.7.1, 12.1, 12.2, 12.3 (sin prestar atención a la parte de actualización), 13.1, 13.2, 13.3, 13.4, 13.5 (hasta 13.5.5 no inclusive), 13.6

Almacenamiento y Estructura de Archivos

- ***Visión General de los medios físicos de almacenamiento:***

Tipos de Almacenamiento de Datos:
Un sector es la unidad de datos más chica que puede ser leída o escrita.

- Caché: Caché es la forma de almacenamiento más rápida y costosa. Su uso lo gestiona el hardware del sistema informático. No hay que preocuparse sobre la gestión del almacenamiento caché del sistema de bases de datos.

- Memoria Principal

- Memoria Flash: También conocida como memoria sólo de lectura programable y borrable eléctricamente (Electrically Erasable Programmable Read-Only Memory, EEPROM), la memoria flash se diferencia de la memoria principal en que los datos. pueden sobrevivir a los fallos del suministro eléctrico. La escritura es más complicada. Para sobreescribir la memoria que se ha
escrito previamente hay que borrar simultáneamente todo un banco de memoria; entonces queda preparado para volver a escribir en él.

- Discos.
- Almacenamiento óptico
- Almacenamiento en cinta : Acceso secuencial.

Jerarquía de almacenamiento: A medida que se desciende por la jerarquía el coste por bit disminuye, mientras que el tiempo de acceso aumenta

1) Cache
2) Memoria principal
3) Memoria Flas
4) Disco magnético
5) Disco óptico
6) Cintas Magnéticas

Los sistemas de almacenamiento desde la memoria principal hacia arriba son volátiles, mientras que los sistemas de almacenamiento por debajo de la memoria principal son no volátiles.

Organización de los Archivos

La base de datos es almacenada como colección de archivos. Los archivos se organizan lógicamente como secuencias de registros. Un registro es una secuencia de campos. Estos registros se corresponden con los bloques del disco. Aunque los bloques son de un tamaño fijo determinado por las propiedades físicas del disco y por el sistema operativo, los tamaños de los registros varían. En las bases de datos relacionales las tuplas de las diferentes relaciones suelen ser de tamaños distintos.
Un enfoque de la correspondencia entre la base de datos y los archivos es utilizar varios y guardar los registros de cada una de las diferentes longitudes fijas existentes en cada uno de esos archivos

- Registros de longitud fija: Este enfoque se basa en que cada registro ocupa la misma cantidad de bytes. Sin embargo, hay dos problemas con este enfoque.
    1) Es difícil borrar un registro en esta estructura, debe haber una forma de ‘marcar’ el registro que se borró o bien rellenarlo con algún otro registro.

2) Otro es que el tamaño de los bloques (del disco) no sea múltiplo del tamaño del arc

cabecera				
registro 0	C-102	Navacerrada	400	
registro 1				
registro 2	C-215	Becerril	700	
registro 3	C-101	Centro	500	
registro 4				
registro 5	C-201	Navacerrada	900	
registro 6				
registro 7	C-110	Centro	600	
registro 8	C-218	Navacerrada	700	

FIGURA 11.9. El archivo de la Figura 11.6 después del borrado de los registros 1, 4 y 6.

-
- Registros de longitud variable:

Headers es el encabezado del archivo

RESUMEN FILMINAS
Organización de archivos en registros

Hay dos alternativas para la ubicación de las tablas:

1. Cada tabla se aloja en archivo separado.
2. Registros de diferentes tablas se alojan en el mismo archivo (se lo llama **agrupamiento de tablas en archivos**).

Con respecto a la ubicación de los registros en un archivo:
1. **Heap:** un registro puede almacenarse en cualquier lugar del archivo donde hay espacio.
2. **Secuencial**: almacenar registros de una tabla en orden secuencial con respecto al valor de una clave de búsqueda de cada registro.

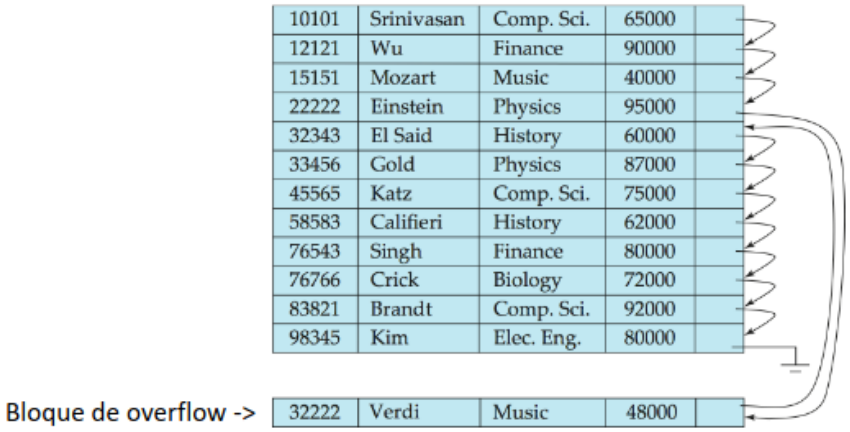
Organización de archivos secuenciales

La organización de archivos secuencial es adecuada para aplicaciones donde se requiere el procesamiento secuencial del archivo entero.
• Los registros en el archivo están ordenados por clave de búsqueda.

Una **clave de búsqueda** es cualquier atributo o conjunto de atributos. No necesita ser la clave primaria o una superclave.
Para permitir retorno rápido de registros en orden de clave de búsqueda encadenamos los registros por punteros.
Un puntero en cada registro apunta al siguiente registro en el orden de la clave de búsqueda.
Para minimizar el número de acceso a bloques en procesamiento de archivos secuenciales, almacenamos los registros físicamente en orden de clave de búsqueda o tan cercano al orden de clave de búsqueda como sea posible.

Borrado: usar cadenas de punteros.
Inserción : localizar la posición donde el registro va a ser insertado.
• Si hay espacio libre insertar allí.
• Si no hay espacio libre, insertar el registro en bloque de overflow

• En ambos casos el puntero de la cadena debe ser actualizado.
• Hace falta reorganizar el archivo cada cierto tiempo para restaurar el orden secuencial.



Agrupamiento de tablas en archivos
Agrupamiento de dos tablas. Resulta en registros de tamaño variable.

Conceptos
Un archivo de BD es particionado en bloques de tamaño fijo.
Los bloques son tanto unidades de almacenamiento como de transferencia de datos.
Un búfer en memoria principal es usado para almacenar copias de bloques de disco.
El gestor de búfer aloja espacio de búfer en memoria principal.

Meta de SGBD: Minimizar cantidad de transferencias de bloques entre disco y memoria.

Indices:
Archivos de índices son usados para acceder más rápido a los datos deseados de las tablas. Los datos deseados se refieren a ciertos atributos de una tabla.
Un archivo de índice tiene consiste de registros llamados **entradas del índice** de la forma
<clave de búsqueda, puntero>
Estos archivos tienen como desventaja almacenamiento extra en disco y actualización del índice, pero estos archivos son mucho más chicos que el archivo original de la tabla.
Los índices de los sistemas de bases de datos juegan el mismo papel que los índices de los libros o los catálogos de fichas de las bibliotecas. Por ejemplo, para recuperar un registro cuenta dado su número de cuenta, el sistema de bases de datos buscaría en un índice para encontrar el bloque de disco en que se encuentra el registro correspondiente, y entonces extraería ese bloque de disco para obtener el registro cuenta.
Hay dos tipos básicos de índices:

- Indices ordenados: están basados en una disposición ordenada de los valores
- Indices asociativos: (hash indices) Estos índices están basados en una distribución uniforme de los valores a través de una serie de cajones (buckets).
  El valor asignado a cada cajón está determinado por una función, llamada función de asociación (hash function)

Un índice **ordenado** puede tomar la forma de

- Una lista de entradas ordenada por valor de clave de búsqueda
- Un árbol ordenado:en cada nodo del árbol las entradas están ordenadas por valor de clave de búsqueda. .

Se clasifican en primarios y secundarios

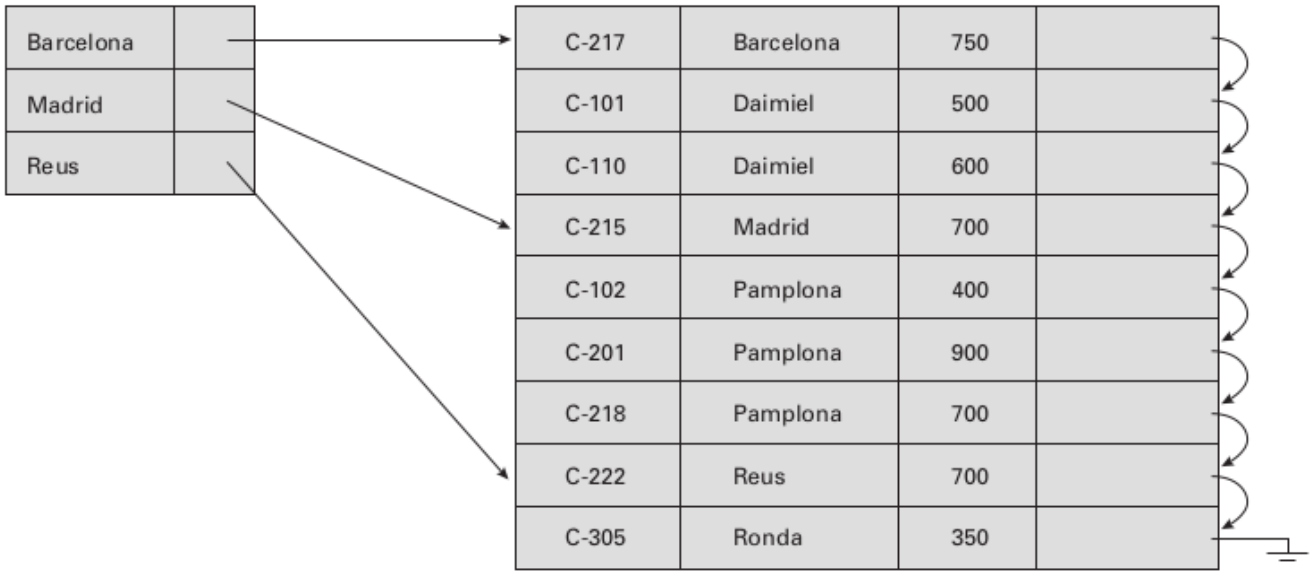
- Primarios: Sea f una archivo secuencialmente ordenado; el índice primario es el índice cuya clave de búsqueda especifica el orden secuencial del archivo f. La clave de búsqueda de un índice primario es normalmente la clave primaria, aunque no es así necesariamente.
  Primarios: Cuando se indexa usando un campo sin repeticiones (por ejemplo la clave primaria) de la tabla, y esa es la clave de ordenación de la tabla en disco..
  Si el archivo que contiene los registros está ordenado secuencialmente, el índice cuya clave de búsqueda especifica el orden secuencial del archivo es el índice primario.
- Secundarios: Sea f un archivo secuencialmente ordenado; un índice secundario es un índice cuya clave de búsqueda especifica un orden diferente del orden secuencial del archivo.

Las claves de búsqueda son atributos que sirven para encontrar registros en un archivo.

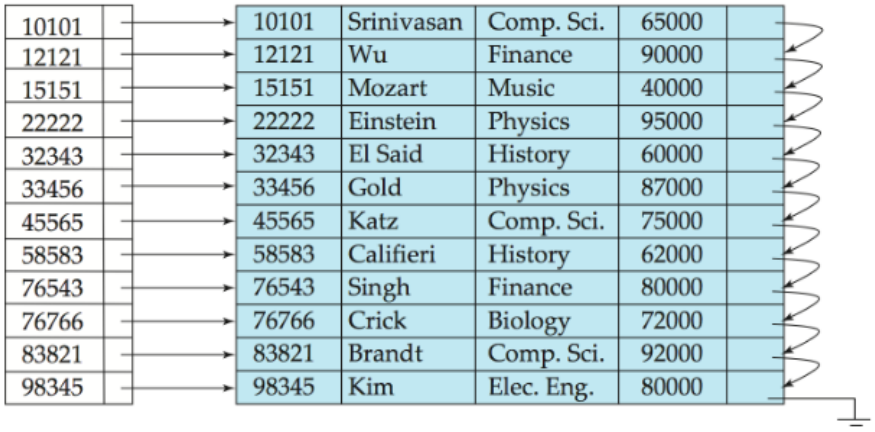
Primarios:

Estos archivos con índice primario según una clave de búsqueda se llaman archivos secuenciales indexados. Representan uno de los esquemas de índices más antiguos usados por los sistemas de bases de Datos.

En esta figura se muestra un archivo secuencial de los registros *cuenta* del ejemplo. En esta figura, los registros están almacenados según el orden de la clave de búsqueda, siendo esta clave nombre-sucursal.

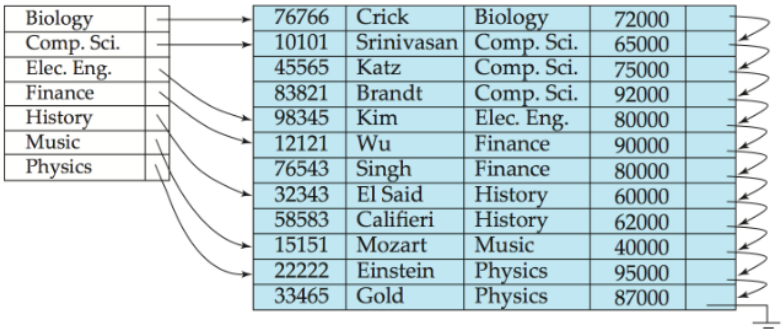


Indice Denso: En un índice denso hay una entrada del índice por cada valor de la clave de búsqueda en el archivo.

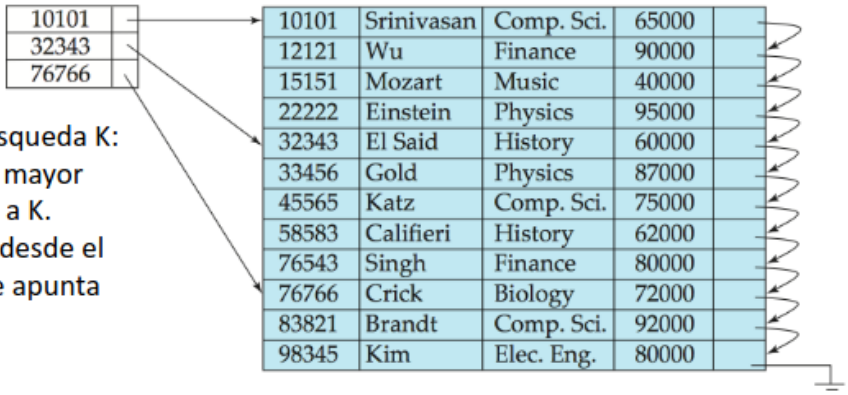


Índice denso en *nombre de departamento* con archivo de *instructor* ordenado por *nombre de departamento* (no clave primaria).

Observar que los punteros en el índice apuntan al primer registro con ese *nombre de departamento* y siguiendo punteros en archivo de tabla se puede recorrer todos los registros de ese *nombre de departamento*.



Indice Disperso: contiene registros de índice para solo algunos valores de clave de búsqueda. Se usan cuando los registros de la tabla están secuencialmente ordenados por clave de búsqueda.



Para ubicar un registro de clave de búsqueda K:

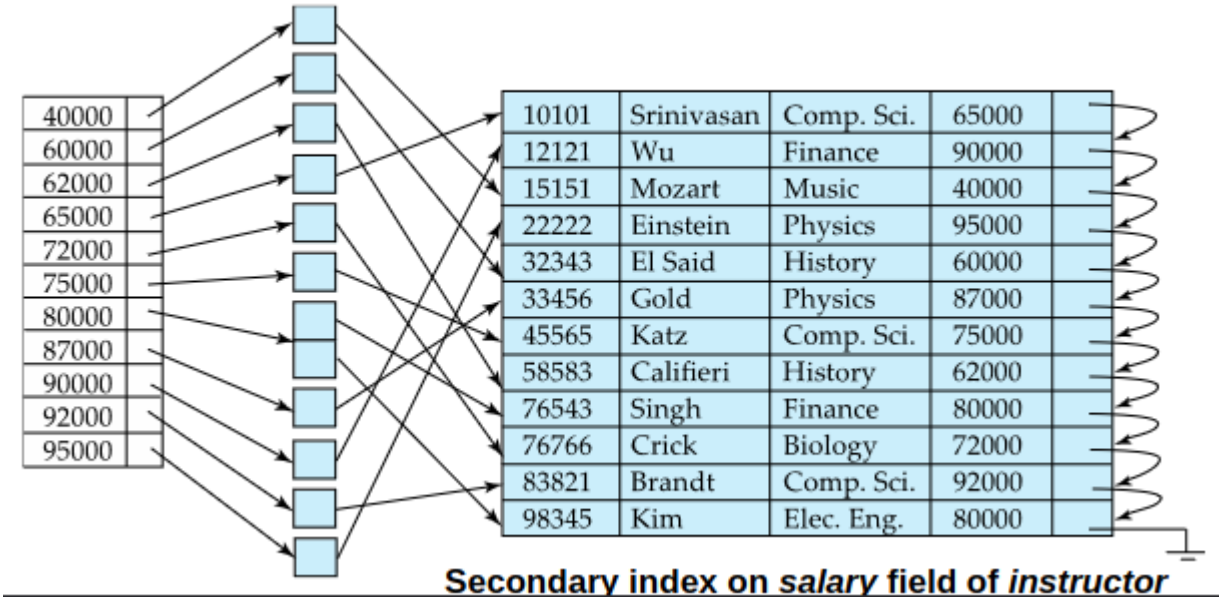
- Encontrar el registro del índice con mayor valor de clave de búsqueda menor a K.
- Buscar el archivo secuencialmente desde el registro al cual el registro del índice apunta

Estos registros requieren menos espacio y menos sobrecarga que los densos.

Índices Secundarios:

Cada registro del índice apunta a un bucket que contiene punteros a todos los registros actuales con el valor particular de clave de búsqueda.

Los índices secundarios deben ser densos.



Podemos destacar que escaneo secuencial con índice primario es eficiente, y costoso para índice secundario. Actualizar índices impone una sobrecarga en la modificación de la BD. Cuando un archivo se modifica, cada índice del archivo debe ser actualizado.

Índice de multinivel

Si el índice primario no cabe en memoria, entonces el acceso pasa a ser costoso.
Se soluciona tratando el índice primario mantenido en disco como archivo secuencial y construir un índice disperso en el mismo.

- Índice externo: índice disperso del índice primario
- Indice interno: archivo del índice primario

Indices de los dos niveles deber ser actualizados al insertar y borrar en la tabla.

Actualización de índice por <u>borrado</u> de registro:

- Indice de un solo nivel
  - Indice denso: Si el registro borrado fue el único registro en el archivo con su clave de búsqueda particular: la clave es borrada del índice también.
    Sino, si la entrada del índice almacena punteros a todos los registros con el mismo valor de clave de búsqueda: se borra el puntero al registro borrado de la entrada del Índice.
    Sino, si el registro borrado es el primer registro con el valor de la clave de búsqueda, se actualiza la entrada del índice para que apunte al siguiente registro

  - Índice disperso: Si el índice no contiene una entrada con el valor de clave de búsqueda del registro borrado: nada necesita hacerse en el índice.
    Sino, si el registro borrado era el único registro con su clave de búsqueda B:
    • Si el próximo valor de clave de búsqueda tiene una entrada en el índice: la entrada para B es borrada del índice.

• Sino: se reemplaza el registro de índice correspondiente con un registro de índice para el próximo valor de clave de búsqueda (en el orden de clave de búsqueda).

Sino: si la entrada del índice para la clave de búsqueda apunta al registro siendo borrado: se actualiza la entrada del índice para apuntar al próximo registro con el mismo valor de clave de búsqueda

Actualización de índice por inserción de registro

- Indice de un solo nivel: Primero se hace búsqueda usando el valor de clave de búsqueda que aparece en el registro a ser insertado.
  - Indice denso: Si el valor de la clave de búsqueda no aparece en el índice: insertar una entrada de índice con valor de clave de búsqueda en el índice en la posición apropiada.
    Sino, si la entrada del índice almacena punteros a todos los registros con el mismo valor de clave de búsqueda: el sistema agrega un puntero al nuevo registro en la entrada del índice.
    Sino: la entrada del índice almacena un puntero a solo el primer registro con el valor de la clave de búsqueda. Se coloca el registro siendo insertado luego de los otros registros con los mismos valores de clave de búsqueda.

  - Indice disperso: Asumimos que el índice almacena una entrada para cada bloque.
    Si el sistema crea un nuevo bloque: inserta el primer valor de la clave de búsqueda (en orden de clave de búsqueda) apareciendo en el nuevo bloque en el índice.
    Sino:
      - o Si el nuevo registro tiene el menor valor de clave de búsqueda en su bloque: el sistema actualiza la entrada del índice apuntando al bloque.
      - o Sino: el sistema no hace cambios al índice.

Un archivo está ordenado según un índice primario (según la secuencialidad del índice).

Actualización de índice de varios niveles

Algoritmos de inserción y borrado para índices de varios niveles son una extensión simple del esquema ya descripto.

• Luego de inserción o borrado de registro el sistema actualiza el índice de más bajo nivel como se describió.

• Con respecto al segundo nivel, el índice de más bajo nivel es meramente un archivo conteniendo registros.

• Si hay algún cambio en el índice de más bajo nivel: el sistema actualiza el índice de segundo nivel como se describió.

Indices con clave de búsqueda compuesta

Una clave de búsqueda conteniendo más de un atributo se llama **clave de búsqueda compuesta**.

La estructura del índice es la misma como la de cualquier otro índice; la única diferencia es que la clave de búsqueda del índice contiene una lista de atributos.

La clave de búsqueda puede representarse como una tupla de valores de la forma: (a1,…,an), donde los atributos individuales son: A1,…, An.

El orden de los valores de la clave de búsqueda es el orden lexicográfico

Acceso a múltiples claves de búsqueda

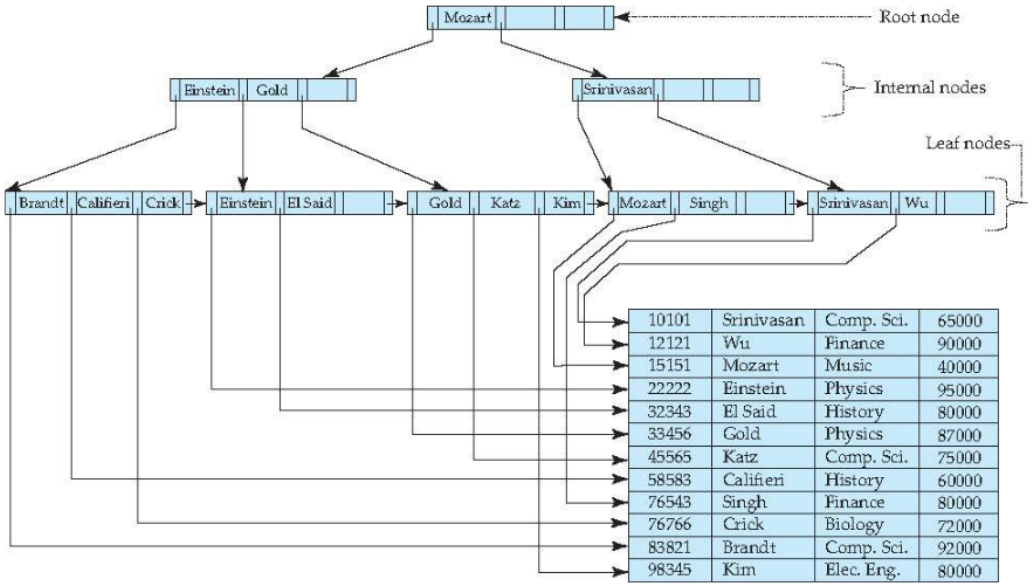
- Se usan múltiples índices para ciertos tipos de consultas.

Ejemplo:
select ID
from instructor
where dept_name= 'Finance' and salary = 80000

Posibles estrategias:
Indices en varios atributos, se usa uno para dept_name y otro para salary. Luego se toma la intersección de ambos conjuntos de punteros.

Índices en varios atributos
Índices con claves de búsqueda compuesta
Índices en claves combinadas

Indices de arbol B+.
Problema: al usar índices que son listas de entradas con archivos secuenciales el desempeño se degrada a medida que el archivo crece porque muchos bloques overflow(cuando no entran en una tabla) son creados.
• Solución 1: reorganizar periódicamente el archivo de la tabla.
• Solución 2: usar índices ordenados con forma de árbol balanceado por ejemplo árbol B+. De esta forma, no hay que reorganizar el archivo de la tabla para mantener el desempeño, el índice se reorganiza con cambios pequeños locales mediante inserciones y borrados.

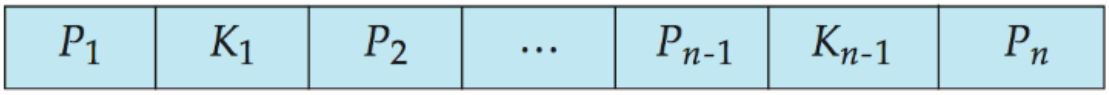


Propiedades de arbol b+

o Todos los caminos desde la raíz a una hoja son de la misma longitud.
o Cada nodo que no es la raíz o una hoja tiene entre n/2 y n hijos.
o Un nodo hoja tiene entre (n–1)/2 y n -1 valores.
o Si la raíz no es una hoja, tiene al menos 2 hijos.
o Si la raíz es una hoja, puede tener entre 0 y n -1 valores.

Estructura de un nodo de árbol B^+

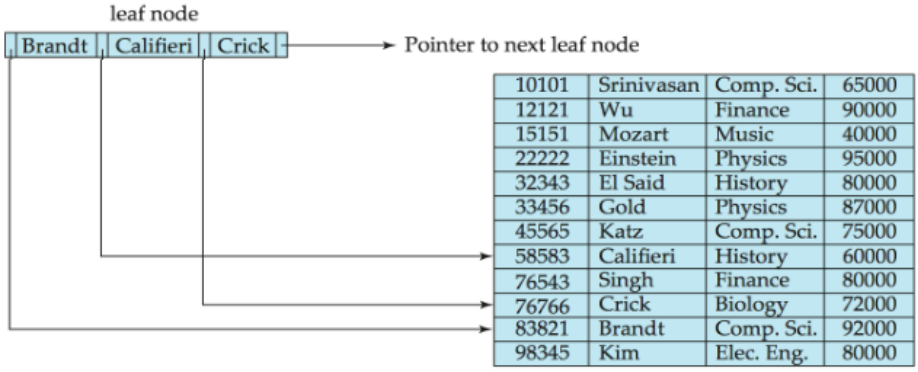
- **Nodo típico**:



- Los Ki son valores de clave de búsqueda.
- Los Pi son:
  • Para nodos no hoja: punteros a hijos
  • Para nodos hoja: punteros a registros o buckets de registros.
- Claves de búsqueda ordenadas $K_1 < K_2 < \ldots < K_{n-1}$
- Los rangos de valores en cada hoja no se solapan, excepto cuando hay valores de clave de búsqueda duplicados, en cuyo caso un valor puede estar presente en más de una hoja.
- Asumir inicialmente que no hay claves duplicadas; manejar duplicaciones después

Propiedades de un Nodo Hoja

- Para i = 1, 2, …, n-1, el puntero Pi apunta al registro del archivo con el valor de clave de búsqueda Ki.
- Si Li, Lj son nodos hoja y i < j, los valores de la clave de búsqueda de Li son menores o iguales a los valores de la clave de búsqueda de Lj.(claves de búsqueda ordenada)
- P_n apunta al próximo nodo hoja en el orden de clave de búsqueda.



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Propiedades de nodos no hojas

Los nodos no hoja forman un índice disperso multinivel en los nodos hoja. Para un nodo no hoja con n punteros:

- Todas las claves de búsqueda en el subárbol al cual P1 apunta (apunta a un subarbol), son menores que K1.
- Para $2 \leq i \leq n - 1$: todas las claves de búsqueda en el subárbol al cual Pi apunta tienen valores mayores o iguales a K_{i-1} y menores que K_i .
- Todas las claves de búsqueda en el subárbol al cual Pn apunta tiene valores mayores o iguales que K_{n-1}.

Es decir, las claves de búsqueda están ordenadas, respetando los subárboles.

Observaciones:

- Como las conexiones entre nodos son hechas mediante punteros, los bloques lógicamente cercanos no necesitan estar físicamente cercanos.
- Si hay K valores de clave de búsqueda en la tabla, la altura del árbol B+ no es mayor a $\log_{n/2}(K)$.
- Borrados e inserciones a la tabla pueden ser manejados eficientemente, porque el índice puede ser reestructurado en tiempo logarítmico
- Si hay K valores de clave de búsqueda en el archivo, la altura del árbol no es mayor a $\log_{n/2}(K)$.
- Un nodo tiene generalmente el mismo tamaño que un bloque de disco, típicamente 4 KiB.

Consultas en árboles B⁺

- Encontrar registro con valor de clave de búsqueda V

Function find(value V)

1. *C = root*
2. While *C* is not nodo hoja {
   1. Sea *i* el menor valor tal que $V \leq K_i$.
   2. If no existe: set *C = último puntero non-null en C*
   3. Else { if ($V = K_i$): $C = P_{i+1}$ else: $C = P_i$}
   }
3. *Sea i* el menor valor tal que $K_i = V$
4. If existe tal valor *i:* seguir el puntero P_i al registro deseado.
5. Else no existe registro con valor de clave de búsqueda *k*.

Manejando duplicados

Con claves de búsqueda duplicadas (tanto en nodos hoja como internos), no podemos garantizar que K1 < K2 < K3 < . . . < Kn–1, pero podemos garantizar que K1≤ K2 ≤ K3 . . . Kn–1

Las claves de búsqueda en el subárbol al cual Pi apunta son ≤ Ki, pero no necesariamente < Ki.

Para ver por qué, supongamos que el mismo valor de la clave de búsqueda V está presente en dos nodos hoja Li y Li+1.

• Entonces en el nodo padre Ki debe ser igual a V.

- Modificación del procedimiento de búsqueda:
  - Recorrer P_i incluso si $V = K_i$

- Tan pronto como alcanzamos un nodo hoja C, chequear si C tiene solo valores de clave de búsqueda menores que V.
- Si es así, sea C = hermano derecho de C antes de chequear si C contiene V.

Indexando strings

- Comprensión de prefijos

Organización de archivo con árbol B+

- El problema de la degradación de archivo de índice secuencial se resuelve usando índice de árbol B+
- El problema de la degradación de un archivo de datos se resuelve usando una organización de archivo de árbol B+

Los nodos hoja en una organización de archivo de árbol B+ almacena registros en lugar de punteros. Son todavía requeridos que estén la mitad llenos.
Como los registros ocupan más espacio que los punteros, el máximo número de registros que pueden ser almacenados en un nodo hoja es menor que el número de punteros en un nodo no hoja.
La inserción y borrado son manejados de la misma manera que inserción y borrado de entradas en índices de árbol B

Procesamiento de Consultas

El árbol binario de ejecución de una consulta es el árbol binario de una expresión de consulta.
o Los nodos hoja son tablas de la base de datos
o Los nodos internos son operadores del álgebra de tablas

- Un operador lógico del álgebra de tablas considera(usa) el algoritmo ineficiente (usado para calcularlo) definido en el capítulo del álgebra de tablas.(operadores con foldr, recursión.)
- Operadores físicos son algoritmos específicos para operadores del álgebra de tablas.

En general tienden a ser más eficientes que los algoritmos lógicos porque hacen uso de informaciones adicionales, como índices, tamaños de búfer en memoria, técnicas avanzadas de algorítmica, etc.
• Un operador del álgebra de tablas va a tener unos cuantos operadores físicos.
• La evaluación de un árbol binario de ejecución va a estar en términos de operadores físicos.

Plan de evaluación: Dada una consulta C del álgebra de tablas un plan de evaluación consiste de un árbol binario de ejecución para una expresión equivalente a C y operadores físicos para ese árbol de ejecución.
La máquina de ejecución de consultas toma el plan de evaluación de consulta, ejecuta ese plan y retorna las respuestas de la consulta.

Los diferentes planes de evaluación para una consulta dada pueden tener diferentes costos.
• Es responsabilidad del SGBD construir un plan de evaluación que minimiza el costo de evaluación de consultas.

• Una vez que un plan de evaluación es elegido, la consulta es evaluada con este plan.
• Optimización de consultas: entre todos aquellos planes de evaluación equivalentes encontrar aquel con menor costo.
• El costo es estimado usando información estadística de la base de datos.
El optimizador de consultas debe conocer el costo de cada operacion.

El objetivo es La meta aquí es dada una expresión del álgebra de tablas, elegir bien los operadores físicos y dado el árbol binario de ejecución de esa expresión, poder estimar el costo de procesamiento.

Evaluación del árbol binario de ejecución
El resultado de evaluar un operador de un nodo interno del árbol binario de ejecución que no es la raíz del árbol se llama resultado intermedio.
Para esto hay dos enfoques:
- Materialización:los resultados intermedios se guardan en disco en tablas temporales a las cuales tiene acceso el sistema gestor de BD (SGBD).
No hay índices sobre este tipo de tablas.(- memoria, + disco)
- Encauzamiento: a medida que se van generando los resultados intermedios se van pasando al siguiente operador.
Los resultados intermedios no se guardan en disco.(+ memoria, -disco)

Medidas de costo de consultas
Se contará el número de transferencia de bloques de disco.
(Vamos a asumir que todas las transferencias de bloques tienen el mismo costo)
- No se distingue entre transferencia de lectura/escrituras.
Contamos también la cantidad de accesos a bloques: seek time.
Los costos de los algoritmos físicos dependen del tamaño del búfer en memoria principal.

Materialización
Con materialización:
o Cambiar nodos lógicos por físicos en el árbol binario de ejecución.
o Si hay más de un operador físico posible, elegir el menos costoso.
o Evaluamos un operador físico por vez comenzando en el nivel más bajo.
o Usamos resultados intermedios en tablas temporales para evaluar los operadores físicos del siguiente nivel

#bloques en disco

Para estimar el tamaño de los resultados intermedios en cantidad de bloques a escribir a disco para los operadores de selección y reunión usamos la función de probabilidad llamada factor de selectividad para calcular la # registros del resultado intermedio.
A partir de la #registros, calcular la cantidad de bloques del resultado intermedio.

Si el operador de selección/reunión usa predicado P, y el input del operador es i, denotamos al factor de selectividad mediante: fs(P,i).

- Para selección:
  #registros resultado intermedio: |r| * fs(P,r) r es la tabla.
- Para reunión:
  #registros resultado intermedio: |r| * |s| * fs(P,r,s)

(recordar conceptos de independencia y uniformidad de probabilidad)

#nro de bloques
Cálculo para resultado con N registros de tamaño R cada uno y B es el tamaño del bloque
NumBloques = N*R/B

Para procesar y estimar el costo de una consulta
1) Decidir el plan de ejecución
   - Armar el árbol binario de ejecución
   - Calcular el factor de selectividad para selecciones y reuniones
   - Decidir operadores físicos

2) Estimar el costo de ejecutar el plan de evaluación
   1. Calcular el tamaño en bloques de las tablas de la BD
   2. Calcular el tamaño de los resultados intermedios en bloques
   3. Calcular el costo de los operadores físicos
   4. Sumar los costos totales

Materialización es la más usada por disponer de mayor cantidad de espacio en disco.

Costo total(acceso a disco) : sumatoria(costo(operaciones))+sumatoria(costo(materialización))
Costo total * velocidad de transferencia = tiempo.

ver ejemplo

Encauzamiento
Materialización produce archivos temporales para los resultados intermedios. Esto implica muchas transferencias de bloques del disco.

Encauzamiento trata de resolver este problema. Se pueden combinar varias operaciones del AT en una tubería de operaciones en la cual los resultados de unaoperación son pasados para la siguiente operación de la tubería.

Beneficios:

- Elimina el costo de leer y escribir tablas temporales, reduciendo así el costo de evaluación de consultas.
- Puede comenzar generando resultados rápidamente, si el operador raíz de un plan de evaluación de consulta es combinado en una tubería con sus inputs.
- Bajos requisitos de memoria porque los resultados son almacenados poco tiempo.

Implementación:

- Cada operación en la tubería puede implementarse como un **iterador** que provee las siguientes funciones en su interfaz: abrir(), siguiente(), y cerrar().
- Un iterador produce la salida de una tupla por vez. Varios iteradores pueden estar activos en un tiempo dado, pasando resultados hacia arriba en el árbol de ejecución.
  El plan de consulta puede ejecutarse invocando los iteradores en un cierto orden.
- Cuando describimos iteradores y sus métodos asumimos que hay una clase para cada operador físico implementado como un iterador y la clase define abrir(), siguiente() y cerrar() en las instancias de la clase.

- Abrir():
  - Inicializa el iterador alojando búferes para su entrada y salida e inicializando todas las estructuras de datos necesarias para el operador.
  - Además llama abrir() para todos los argumentos de la operación.

- Siguiente():
  - Cada llamada a siguiente() retorna la próxima tupla de salida de la operación;
  - Ajusta las estructuras de datos para permitir que tuplas subsiguientes sean obtenidas.
  - Siguiente() ejecuta el código específico de la operación siendo realizada en los input.
  - Además llama siguiente() una o más veces en sus argumentos.
  - En siguiente() el estado del iterador es actualizado para mantener la pista de la cantidad de input procesados.
  - Cuando no se pueden retornar más tuplas se retorna un valor especial: NotFound.

- Cerrar():
  - Termina la iteración luego que todas las tuplas que pueden ser generadas han sido generadas, o el número requerido de tuplas ha sido retornado.
  - Se llama cerrar() en todos los argumentos del operador

Ejemplos: escaneo ordenado, iterador usando búsqueda lineal, iterador para proyección, iterador implementando reunión por combinación (merge sort join)