Capítulo 4 - Almacenamiento de tablas en archivos e índices

Almacenamiento de tablas en archivos

Enfoque de registros de tamaño fijo

- Almacenar registro i desde el byte $n \times (i-1)$, donde n es el tamaño de cada registro
  - Para que no se curcen bloques, se puede modificar para que cada registro esté exactamente en un bloque
- *Borrado de registros* $\rightarrow$ Se elimina el i-ésimo
  - Correr registros $i+1$, ..., $n-1$
  - Mover registro $n-1$ a esa posición
  - Uso de lista enlazadas (marcan los registros vacíos con un header al primero)

Organización de registros en archivos

- Ubicación de tablas
  1. Cada tabla en archivo separado
  2. Registros de diferentes tablas en mismo archivo (*agrupamiento de tablas en archivos*)
- Ubicación de los registros en un archivo
  1. Heap: en cualquier lugar donde hay espacio
  2. Secuencial (más usado): en orden secuencial respecto al valor de una clave de búsqueda de cada registro
     - Atributo o conjunto de estos que no necesariemente son PK o superclave
     - Para usar este orden, encadenamos los registros por punteros, donde cada uno apunta al siguiente en el orden de la clave de búsqueda
     - *Borrado* $\rightarrow$ usar cadenas de punteros
     - *Inserción* $\rightarrow$ si la posición en la que tendría que ir está vacía, lo insertamos. Sino, lo ponemos en un bloque de overflow. Luego actualizamos los punteros
       - Cada cierto tiempo hay que reorganizar el archivo para restaurar el orden secuencial

Agrupamiento de tablas en archivos (secuencial)

- Cuando tenemos consultas que necesitan acceder a registros relacionados de dos tablas (por ejemplo).
- Podés almacenar registros de dos tablas relacionadas del siguiente modo: si tenemos *departamentos* e *instructores*, con relación Many-to-One, entonces:
  - Ponemos los departamentos
  - Luego de un departamento siguen *todos* los instructores de ese departamento con todos sus datos
- Implica registros de tamaño variable (porque son de diferentes tablas)

- Se pueden agregar cadenas de punteros para enlazar registros de una relación particular

Indexado - Archivos de índices

- Usados para acceder más rápido a los datos deseados de las tablas.
  - Son archivos más chicos y que podemos tener en memoria, para evitar la lentitud de pedir lectura en disco por cada registro "posta" que querramos ver
  - La idea es primero identificar en el archivo de índices (que es más pequeño) la clave que buscamos y, con ello, sabemos en qué posición del archivo *real* se encuentra nuestro registro
- Un archivo de índice consiste de registros llamados *entradas del índice* de la forma **<clave de búsqueda, puntero>**
  - Consideramos *índices ordenados*
- Pros $\rightarrow$ Archivos mucho más chicos que el archivo original de la tabla y más rápidez para una consulta
- Contras $\rightarrow$ Almacenamiento extra y actualización del índice ante alguna modificación (inserción o borrado)
- Clasificaciones
  - Puede tomar la forma de
    - **Lista de entradas** ordenadas por valor de clave de búsqueda
    - **Árbol ordenado** $\rightarrow$ además de otras propiedades para el ordenamiento, en cada nodo del árbol las entradas están ordenadas por valor de clave de búsqueda
  - Tipo
    - **Primario** $\rightarrow$ su clave de búsqueda especifica el orden secuencial del archivo f (donde está la tabla original)
      - Escaneo secuencial es eficiente
    - **Secundario** $\rightarrow$ su clave de búsqueda especifica un orden **diferente** del orden secuencial del archivo
      - Cada registro del índice apunta a un **bucket** que contiene punteros a *todos* los registros actuales con el valor particular de clave de búsqueda
      - Deben ser densos
      - Escaneo secuencial costoso
  - Densidad
    - **Denso** $\rightarrow$ una entrada del índice por cada valor de la clave de búsqueda en el arcihvo
    - **Disperso** $\rightarrow$ contiene registros de índice para solo algunos valores de clave de búsqueda (se usa cuando los registros de la tabla están secuencialmente ordenados por clave de búsqueda)
      - Pros $\rightarrow$ Requieren menos espacio y menos sobrecarga de mantenimiento para inserción/borrado
      - Contras $\rightarrow$ Son más lentos que índices densos para localizar registros
- Índice de multinivel
  - Si el índice primario no cabe en memoria, entonces el acceso pasa a ser costoso (porque pasa a ir a disco)
  - Para ello, se trata el índice primario mantenido en disco como un archivo secuencial y se construye un índice disperso para este

        - *Índice externo* $\rightarrow$ índice disperso del índice primario
        - *Índice interno* $\rightarrow$ archivo del índice primario
    - La única contra es en modificación (inserción/borrado) porque deben actualizarse los dos índices
- Tipos de clave de búsqueda a utilizar
    - Simple (un solo atributo)
    - Compuesta (varios atributos)
        - Se considera el orden lexicográfico
        - Es decir, $(a_1, b_1) < (a_2, b_2) \Leftrightarrow a_1 < a_2 \vee (a_1 = a_2 \wedge b_1 < b_2)$
        - Esto hace eficiente la búsqueda de rangos para b, pero no para a por el orden que se establece

Índices árbol B^+

- Árbol balanceado
- Ampliamente utilizado
- El índice se reorganiza con cambios pequeños locales frente a inserciones y borrados (del orden de log)

Condiciones

Dado un n fijado, se cumple que:

- Todos los caminos raíz $\leftrightarrow$ hoja son de la misma longitud
- Cada nodo que **no** es raíz u hoja (i.e., **intermedio**) tiene entre $\lceil \frac{n}{2} \rceil$ y n hijos
- Un nodo **hoja** tiene entre $\lceil \frac{n-1}{2} \rceil$ y $n-1$ valores
- Caso de la *raíz*
    - Si no es hoja, entonces tiene al menos 2 hijos
    - Si es hoja, puede tener entre 0 y $n-1$ valores

Estructura típica

Estructura de nodo de árbol B^+

- $P_1|K_1|P_2|K_2|\ldots|P_{n-1}|K_{n-1}|P_n$
- K_i son valores de clave de búsqueda
    - Ordenadas $K_1 < K_2 < K_3 < \ldots < K_{n-1}$
- P_i son
    - Para *nodos no hoja* $\rightarrow$ punteros a hijos
    - Para *nodos hoja* $\rightarrow$ punteros a registros o *buckets* de registros
- 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.

Nodo hoja

- Para $i = 1, \ldots, n-1$, P_i apunta al registro del archivo con el valor de clave de búsqueda K_i

- P_n apunta al próximo nodo hoja en el orden de clave de búsqueda
- Si L_i, L_j son nodos hoja y $i < j$, entonces los valores de la clave de búsqueda de $L_i \leq$ que los de L_j

Nodo no hoja

- Forman 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 P_1 apunta son menores que K_1
  - Para $2 \leq i \leq n-1$, todas las claves de búsqueda al cual P_i 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 P_n apunta tiene valores mayores o iguales que K_{n-1}

Observaciones

- Si hay K valores de clave de búsqueda en la tabla, la altura del árbol B^+ no es mayor a $\lceil \log_{\lceil \frac{n}{2} \rceil}(K) \rceil$
  - Entonces las consultas pueden ser realizadas eficientemente
- Borrados e inserciones a la tabla también son realizados en tiempo logarítmico
- Un nodo tiene generalmente el **mismo tamaño** que un bloque de disco (típicamente $4KiB$)
- n es típicamente cercano a 100 (Es decir, $40B$ por entrada de índice)
- Truquito cuando la clave es un string $\rightarrow$ **Compresión de prefijos**
  - Consideramos los prefijos de las claves tal que las diferencian en ese subárbol (no hace falta guardar y comparar todo el string completo)
- También se usa este esquema de **árbol** B^+ para la **organización de archivo**, en donde la única diferencia es que los *nodos hoja* en vez de guardar puntero, almacenan los registros