RESUMEN DE FUNDAMENTOS DE ORGANIZACIÓN DE DATOS “FOD”

Base de Datos

Una base de datos es como una colección organizada de datos que representan algo del mundo real (tienen una coherencia) y físicamente se guardan en archivos.
Para que un Sistema de Software sea útil debe interactuar con una BD y recuperar su información en el menor tiempo posible.
Esta tiene 3 niveles de abstracción para simplificar la interacción con los usuarios:
• Nivel de Vista: corresponde al nivel más alto de abstracción, en este se describe parcialmente la BD, esta es la parte que se ve de la BD.
• Nivel Lógico: en este nivel se describe la BD completa, indicando que datos se almacenaran y las relaciones existentes entre esos datos.
• Nivel Físico: este es el nivel más bajo de la abstracción, en el cual se describe como se almacenan realmente los datos.

Archivos

Un archivo es una colección de registros en almacenamiento secundario, o sea en discos (generalmente en discos rígidos), no en memoria RAM, la RAM es más rápida, pero es volátil (se borra), el disco es más lento, pero guarda permanentemente y tiene más capacidad.

- Tipos de archivos:

• Archivo Físico: es el archivo residente en la memoria secundaria y es administrado por el sistema operativo.
• Archivo Lógico: es el archivo usado desde el algoritmo. Cuando el algoritmo necesita operar con un archivo, genera una conexión con el sistema operativo.
Tipos y Acceso: Los archivos pueden contener datos simples o registros heterogéneos. Se analizan principalmente los archivos de longitud fija de registros.
Las formas de acceder a los datos son:

  - Secuencial: Acceso elemento a elemento en orden físico.
  - Secuencial Indizado: Acceso basado en una organización previa, un orden lógico, este orden está definido por un índice, no es necesariamente el orden físico.
  - Acceso directo: Se va directo a un registro especifico sin pasar por los anteriores.

- Proceso de Bajas:
  - Baja física: Borra efectivamente la información y recupera el espacio en disco. Puede hacerse generando un archivo nuevo sin los datos eliminados (costoso en espacio y tiempo) o reubicando datos dentro del mismo archivo (costoso en tiempo si los elementos se desplazan mucho.
  - Baja Lógica: Marca la información como borrada pero sin liberar el espacio físico. Es más eficiente en tiempo (localizar y marcar con un solo acceso), pero desperdicia espacio.

- Recuperación y Reasignación de Espacio:

Para la baja lógica, se puede compactar el archivo periódicamente(similar a la baja física) o reasignar el espacio liberado para nuevas entradas, manteniendo una lista encadenada invertida de posiciones libres.

- Fragmentación

La fragmentación tiene que ver con la reutilización de espacio, existen do tipos de fragmentación:
Fragmentación Interna: Se llama fragmentación interna a aquella que se produce cuando un elemento de dato se le asigna mayor espacio del necesario.
Fragmentación Externa: Se llama fragmentación externa a el espacio disponible entre dos registros, pero que es demasiado pequeño para ser utilizado.

- Políticas de selección de espacio para nuevos registros:

El proceso de inserción debe localizar el lugar dentro del archivo más adecuado para el nuevo elemento.
• Primer Ajuste: consiste en seleccionar el primer espacio disponible donde quepa el registro a insertar.
• Mejor Ajuste: consiste en seleccionar el espacio más adecuado para el registro. Se considera el espacio más adecuado como aquel de menor tamaño donde quepa el registro.
• Peor Ajuste: consiste en seleccionar el espacio de mayor tamaño asignando para el registro solo los bytes necesarios.

- Índices:

Son estructuras de datos auxiliares que aceleran el acceso a la información en un archivo. Contienen las claves de los registros y una referencia a su ubicación física. Un índice es un archivo aparte, generalmente más chico y de registros de longitud fija, que esta ordenado y permite búsquedas rápidas.

  - Índice Primario: Basado en la clave principal del archivo. Su mantenimiento implica actualizar el propio índice. Su ventaja principal es mejorar la performance de busqueda
  - Índice Secundario: Basados en claves que pueden tener valores repetidos. Enlazan la clave secundaria a una o más claves primarias, que a su vez referencian la ubicación física. Esto minimiza las actualizaciones del índice secundario si los registros se reubican físicamente.

Arboles

Los árboles son estructuras de datos que se utilizan para organizar índices asociados a archivos de datos, permitiendo agilizar el acceso a la información y mantener el orden de manera eficiente, se busca minimizar los accesos a memoria secundaria, que son costos.

Arboles Binarios:

Son estructuras de datos dinámicas no lineales donde cada nodo puede tener como máximo dos hijos. Un árbol binario ordenado mantiene los elementos menores a la izquierda y los mayores a la derecha, tienen busqueda rápida y logarítmica, pero tienen un problema, se pueden desbalancear y quedar como una lista y pierde la gracia.

Arboles AVL:

Son arboles binarios balanceados en altura donde la diferencia máxima entre las alturas de dos subárboles con la misma raíz no supera un delta determinado (normalmente 1), mantenerlos es complejo, para discos los que realmente se usan son los Arboles B.

Arboles B

Los arboles B, se usan porque son multicamino, anchos y de poca altura, cada nodo tiene muchas claves y muchos hijos, esto minimiza la cantidad de accesos a disco, que es lo lento y tienen reglas estrictas, de cuantos elementos e hijos puede tener cada nodo para mantenerse balanceado, hay variantes de los Arboles B, como lo B* y B++. Los arboles B tienen una eficiencia de busqueda similar a la de los arboles binarios, pero con una altura mucho menor, lo que se traduce en muy pocos accesos a disco.

- Propiedades de los Arboles B:
  - Cada nodo contiene máximo M-1 elementos y M hijos.
  - La raíz tiene al menos dos descendientes o ninguno.
  - Nodos con M descendientes directos contienen M-1 elementos.
  - Los nodos terminales (hojas) tienen un mínimo de [M/2] – 1 elementos
  - Todos los nodos terminales se encuentran al mismo nivel.

- Creación/Inserción:

El proceso comienza con una estructura vacía. Los elementos se insertan en nodos terminales. Si un nodo se satura (overflow), se divide, y la clave central se promociona al nodo padre. Esto asegura que el árbol crezca de abajo hacia arriba y se mantenga balanceado.

- Eliminación:

Para eliminar un elemento, se debe localizar en un nodo terminal. Si no lo es, se intercambia con el menor de sus hijos mayores (o mayor de sus menores). Si el borrado causa underflow de un nodo, se intenta redistribuir elementos con nodos hermanos adyacentes; si no es posible, se realiza una concatenación de nodos. Las operaciones de borrado son eficientes, con pocas lecturas y escrituras.

Arboles B*

Es una variante de los arboles B que busca minimizar la generación de nuevos nodos al completar los nodos (llenarlos a 2/3 de su capacidad) antes de dividirlos.

- Propiedades de los Arboles B*:
  - Cada nodo del árbol pude contener, como máximo, M descendientes y M-1 elementos.
  - La raíz no posee descendientes o tiene al menos dos.
  - Un nodo con x descendientes contiene x-1 elementos.
  - Los nodos terminales tienen, como mínimo, [(2M-1) / 3] - 1 elementos y como máximo, M-1 elementos.
  - Los nodos que no son terminales ni raíz tienen, como mínimo, [(2M-1) / 3] descendientes.
  - Todos los nodos terminales se encuentran al mismo nivel.

- Inserción sobre los arboles B*:

El proceso de inserción en un árbol B* puede ser regulado de acuerdo con tres políticas básicas: política de un lado, política de un lado u otro lado, política de un lado y otro lado. Así cada política determina, en caso de overflow, el nodo adyacente hermano a tener en cuenta.
Esto resulta en una menor altura del árbol y por ende mejor performance.

Arboles B+

Los arboles B+ son arboles multicamino donde toda la información de las claves esta contenida en los nodos terminales(hojas). Los nodos no terminales actúan solo como separadores y poseen una copia de los elementos de los nodos terminales. Además, los nodos terminales están enlazados entre sí, lo que permite un recorrido secuencial eficiente. Esto permite búsquedas aleatorias rápidas (gracias a la estructura del árbol) y acceso secuencial eficiente (gracias al enlace de los nodos hoja).

- Propiedades de los Arboles B+:
  - Cada nodo del árbol pude contener, como máximo, M descendientes y M-1 elementos.
  - La raíz no posee descendientes o tiene al menos dos.
  - Un nodo con x descendientes contiene x-1 elementos.
  - Los nodos terminales tienen, como mínimo, ([M/2] -1) elementos y como máximo, M-1 elementos.
  - Los nodos que no son terminales ni raíz tienen, como mínimo, [M/2] descendientes.
  - Todos los nodos terminales se encuentran al mismo nivel.
  - Los nodos terminales representan un conjunto de datos y son enlazados entre ellos.

• Arboles b+ con prefijos simples:
El agregado de los prefijos simples intenta aprovechar el mejor uso del espacio físico. Los separadores en los nodos no terminales se representan por la mínima expresión posible de la clave que permita decidir si la busqueda se realiza hacia la izquierda o hacia la derecha.

Hashing

El Hashing es un método para mejorar la eficiencia de busqueda en archivos de datos, buscando un solo acceso a disco en promedio para recuperar la información.
El Hashing lo que hace es convertir la clave primaria de un registro de datos en una dirección física de almacenamiento dentro del archivo.
La función de hash ideal distribuye los registros de la manera más uniforme y aleatoria.
• Ventajas:
-No requiere almacenamiento adicional para una estructura de índice separada.
-Facilita la inserción y eliminación rápida de registros.
-Localiza registros con un solo acceso a disco en la mayoría de los casos.
• Desventajas:
-No se aplica a registros de longitud variable.
-No es posible obtener un orden lógico de los datos.
-No es posible tratar con claves duplicadas.

• Tipos de Dispersión:

▪ Hashing con espacio de direccionamiento estático:
El espacio disponible para dispersar los registros es fijado previamente.
Parámetros esenciales:

◦ Función de Hash: Transforma la clave en una dirección dentro de un rango predefinido. La simplicidad puede llevar a colisiones.

◦ Tamaño del nodo de almacenamiento (cubeta): Cada dirección de hash apunta a un nodo o sector con capacidad para almacenar más de un registro, lo que reduce las colisiones para sinónimos.

◦ Densidad de Empaquetamiento (DE): Relación entre la cantidad de registros (K) y el espacio disponible (M direcciones * RPN registros por nodo). A mayor DE, mayor posibilidad de colisiones; a menor DE, menor probabilidad, pero mayor desperdicio de espacio.

◦ Métodos de tratamiento de desbordes (overflow): Cuando un nodo está lleno, se necesita reubicar el registro y asegurar su posterior recuperación.

▪ Saturación Progresiva: Almacena el registro en la siguiente dirección disponible más cercana. Es simple, pero puede crear saturación sectorizada y requerir múltiples accesos para buscar.

▪ Saturación Progresiva Encadenada: Similar a la anterior, pero enlaza la dirección base inicial con la nueva dirección del registro, formando una cadena de búsqueda. Generalmente mejora la performance al reducir el número de accesos.

▪ Doble Dispersión: Utiliza una segunda función de hash para calcular un desplazamiento que se suma a la dirección base, dispersando los registros de desborde por todo el archivo. Esto puede aumentar el desplazamiento de la cabeza lectora del disco.

▪ Área de Desbordes por Separado: Distingue los nodos direccionales por la función de hash de una zona reservada exclusivamente para registros en overflow. Los nodos originales se encadenan a las direcciones de esta área.
▪ Hashing Asistido por Tabla: Usa tres funciones de hash (F1H, F2H, F3H) y una tabla en memoria principal. Permite asegurar un solo acceso a disco para recuperar un registro, incluso en caso de saturación, reubicando el registro con la F3H más alta. El costo es el espacio extra de la tabla y la complejidad adicional en la inserción con saturación.

▪ Hashing con espacio de direccionamiento dinámico:
El espacio disponible para los nodos aumenta o disminuye en función de las necesidades del archivo, sin fijar una cantidad de nodos *a priori*.

◦ Ventaja: Evita el alto costo de redispersar todo el archivo cuando el espacio se agota, una operación muy costosa en el hashing estático.

◦Hashing Extensible: Una implementación popular. Comienza con un solo nodo y va aumentando la cantidad de direcciones a medida que los nodos se completan. La función de hash retorna una secuencia de bits, y la tabla en memoria principal utiliza una cantidad variable de esos bits para direccionar los nodos. Asegura encontrar cada registro en un solo acceso a disco. El costo principal se da en el procesamiento adicional cuando una inserción genera un overflow y el sistema debe duplicar su tabla de direcciones.

En resumen, la elección del método de gestión de archivos (secuencial, indizado, o disperso) dependerá de las necesidades específicas de consulta. Mientras que los árboles balanceados (especialmente B+) ofrecen una solución muy eficiente tanto para búsquedas aleatorias como secuenciales, el hashing (particularmente el hashing extensible) se destaca por garantizar un único acceso a disco para la recuperación de información en un alto porcentaje de los casos, a costa de no mantener un orden físico de los datos ni manejar claves duplicadas.