Estructuras de Almacenamiento de Datos

Proyecto Especial

Segunda Parte

Trabajo 1

Medicamentos con índice sobre Grid File

UNCPBA

Facultad de Ciencias Exactas

Grupo 21

Integrantes

Alonso, Manuel <manuel.alonso.d@gmail.com>

Mendiola, Jorge Carlos <wrtfix@gmail.com>

Introducción al problema

Cuando se trabaja con grandes volúmenes de información, es necesario implementar índices adecuados y eficientes de modo de disminuir la cantidad de accesos a disco y de esta forma los tiempos de alta, baja y modificación.

Para aumentar la efectividad de dichos índices es posible construirlos a partir de varios atributos, obteniendo así, índices multiclave. De esta forma, es posible acceder a los datos a partir de varios criterios de búsqueda al mismo tiempo. A continuación estudiaremos y discutiremos la construcción e implementación de índices Gridfile, analizando ventajas y desventajas de dicha estructura.

Indices Gridfile

Un Gridfile consta de un vector multidimensional, donde cada vector unidimensional se corresponde con una escala de valores para cada uno de los atributos por los cuales se indexan los datos. De modo que para indexar a partir de N claves, se necesitará un vector de orden N.

Las celdas de dicho vector están vinculadas directamente con Baldes que almacenan valores enteros: posiciones en el archivo.

Cuando se desea insertar elementos en el arreglo, se accede a la celda correspondiente a través de las escalas y luego se inserta en el Balde de dicha celda. En caso de que dicho Balde este “lleno”, exiten diferentes alternativas:

- Mantener sin cambios la escala de la estructura, y adaptarse al crecimiento utilizando baldes de desborde.
- Ajustar las líneas de escala a medida que sea necesario para adaptarse al crecimiento de la estructura.
- Utilizar una estrategia mixta.

Objetivo

El objetivo del siguiente proyecto es diseñar e implementar una estructura de índice Gridfile capaz de indexar los datos contenidos en un archivo de acuerdo a los atributos Accion, Forma y Precio. A partir de dicha estructura se desea leer, insertar y eliminar datos de acuerdo con los distintos criterios de búsqueda.

Desarrollo

La correcta construcción de un Gridfile se basa en la optimización en la distribución de la información dentro de la estructura. Al igual que en cualquier tipo de índice, se busca insertar la información de forma homogénea de modo de no desperdiciar memoria principal ni disco al almacenar dicho índice.

Dicha optimización se obtiene analizando los datos a indexar, de modo de poder seleccionar correctamente las escalas lineales correspondientes a cada atributo y el tamaño de los baldes.

Datos a indexar

El archivo a indexar posee información sobre medicamentos, donde cada medicamento se define de la siguiente forma:

- atributo nro_registro es una secuencia de hasta 8 caracteres numéricos. (0 a 99999999)
- atributo descripcion puede contener un máximo de 256 caracteres (letras y números).
- atributo laboratorio puede contener un máximo de 16 caracteres (letras y números).
- atributo accion_medicamento es una secuencia de hasta 4 caracteres numéricos. (0 a 9999)
- atributo forma_medicamento es una secuencia de hasta 2 caracteres numéricos. (0 a 99)
- atributo tamaño_medicamento es una secuencia de hasta 2 caracteres numéricos. (0 a 99)
- atributo via_administracion es una secuencia de hasta 2 caracteres numéricos. (0 a 99)
- atributo precio es una secuencia de hasta 3 caracteres numéricos, una coma decimal y 2 caracteres numéricos más.

  La indexación del archivo se realizará a partir de las siguientes claves:
  - accion_medicamento.
  - forma_medicamento.
  - precio_medicamento.

Para construir las escalas de cada uno de estos atributos, es necesario primero analizar la distribución de los datos del archivo.

Análisis de Datos

El análisis de los datos se realizó sobre el total de los 1000 medicamentos almacenados en el archivos *datos_medicamentos.dat*. Allí se puede observar la dispersión de los datos de acuerdo a los distintos valores, llegamos a las siguientes conclusiones:

- Atributo acción: Los valores de este criterio se encuentran en su mayoría entre 0 y 100. Es por esto que dedicamos gran parte de la escala a estos valores. Se utilizó una escala de tamaño 16.

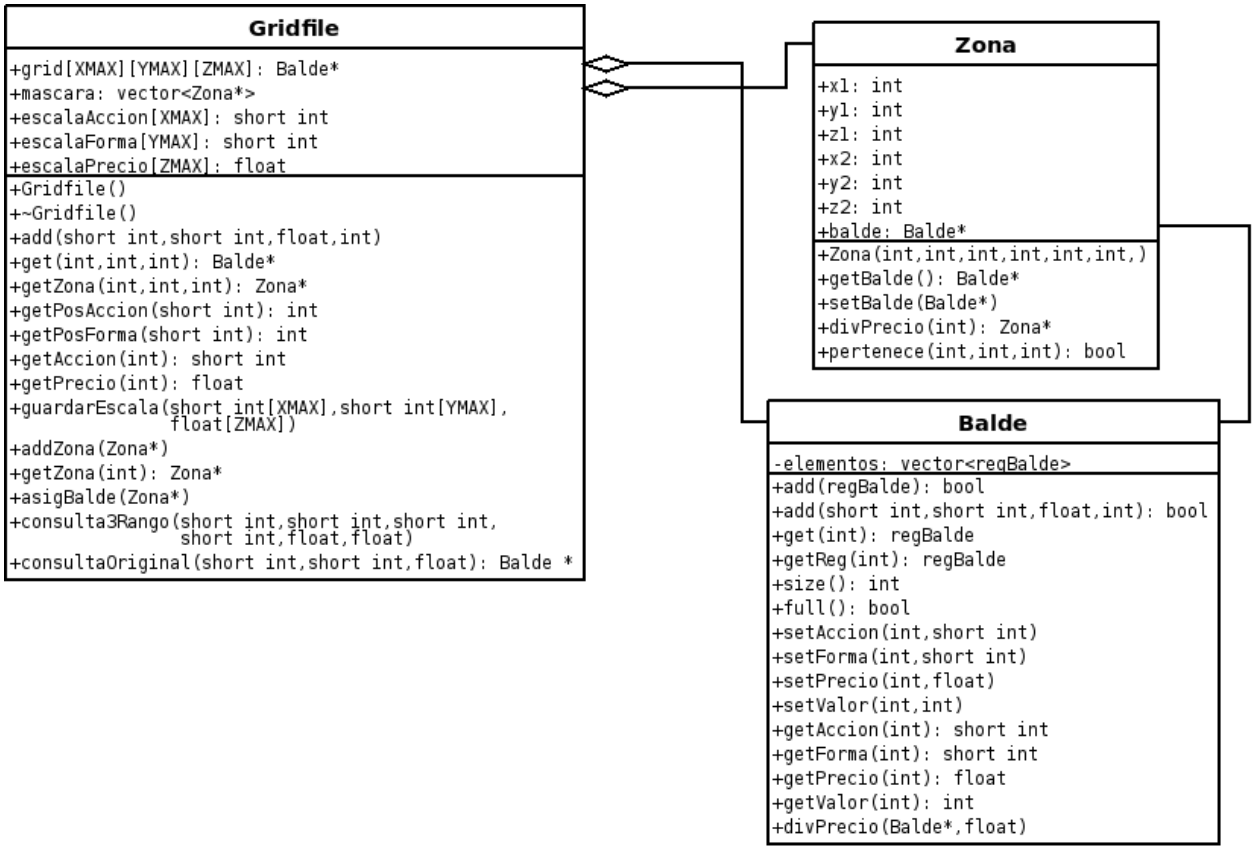
- Atributo forma: Sus valores se repiten constantemente y son varían muy poco. Esto nos llevó a utilizar una escala de tamaño 8.

- Atributo precio: Es el atributo con mayor variación y dispersión de sus datos. Se le asignó una escala de gran tamaño (64) ya que sus valores reclamaran constantemente particiones de Zona.

Observar que se usan siempre potencias de 2 de modo de poder maximizar las particiones.

Diseño del sistema

Diagrama de Clases



Especificación de Clases

A continuación se detallan y especifican los métodos y atributos más importantes de cada clase.

Agrega un nuevo registro al Balde
```
bool Balde::add(regBalde a)
```

Agrega un nuevo registro al Balde con los datos dados.
```
bool Balde::add(short int accion, short int forma, float precio, int filepos)
```

Devuelve un registro.
```
regBalde Balde::get(int pos)
```

Devuelve un registro y lo quita del balde.
```
regBalde Balde::getReg(int pos) {
```

Mueve los elementos de "this" en "destino" respecto a la variable "precio"
```
void Balde::divPrecio(Balde *destino,float precio)
```

Constructor de Zona, a partir de 2 coodenadas.
```
Zona::Zona(int x1,int y1,int z1, int x2,int y2,int z2)
```

Devuelve el Balde correspondiente a la Zona.
```
Balde* Zona::getBalde()
```

Divide la Zona respecto a la variable z
```
Zona* Zona::divPrecio(int z3)
```

True si la coordenada (x,y,z) pertenece a la Zona
```
bool Zona::pertenece(int x,int y, int z)
```

Retorna el puntero a Balde en (x,y,z) del grid.
```
Balde* Gridfile::get(int x,int y,int z)
```

A partir de 3 valores, retorna el Balde donde debería ser insertado.
```
Balde* Gridfile::getOriginal(short int a, short int f, float p)
```

Settea las escalas del Grid.
```
void Gridfile::guardarEscalas(short int a[XMAX],short int f[YMAX], float
p[ZMAX])
```

Agrega resultados al grid. Realiza el proceso de partionado.
```
void Gridfile::add(short int accion,short int forma,float precio,int valor)
```

Retorna el valor de la escala de accion correspondiente a una accion dada
```
int Gridfile::getPosAccion(short int accion)
```

Retorna el valor de la escala de forma correspondiente a una forma dada
```
int Gridfile::getPosForma(short int forma)
```

Retorna el valor de la escala de precio correspondiente a un precio dado
```
int Gridfile::getPosPrecio(float precio)
```

A partir de una pos de escala de precios, retorna su valor de precio.
```
float Gridfile::getPrecio(int pos)
```

Agrega una Zona al Gridfile. Realiza la actualizacion de las celdas abarcadas diche Zona.
```
void Gridfile::addZona(Zona* z)
```

Devuelve una Zona determinada.
```
Zona* Gridfile::getZona(int i)
```

A partir de una coordenada, retorna la Zona a la que pertenece.
```
Zona* Gridfile::getZona(int x, int y, int z)
```

Asigna ese balde a todas las celdas que pertenecen a la Zona.
```
void Gridfile::asigBalde(Zona *zona)
```

Funcion utilizada para consultar. Dados los valores y/o intervalos, retorna las posiciones de archivo que satisfacen la consulta.
```
vector<int> Gridfile::consulta3rangos(short int a1, short int a2, short int f1,
short int f2, float p1,float p2)
```

Implementación

El sistema fue desarrollado en lenguaje de programación C++, con un diseño orientado a objetos (ver Diseño de Clases).

Estructuras de Datos

A continuación se especificarán y justificarán las distintas estructuras de datos implementadas para el correcto funcionamiento del Gridfile.

Medicamentos:

Luego de ser levantado del archivo de texto, cada medicamento es almacenado en un **struct**

(registro) de C. Esto permite almacenarlo fácilmente en el archivo binario con el cual se construirá y actualizará el Gridfile. Dicho struct está definido de la siguiente manera:

```
struct regMedicamento
{
	//una secuencia de hasta 8 caracteres numéricos. (0 a 99999999)
	long int nro_registro;

	//puede contener un máximo de 256 caracteres (letras y números).
	char descripcion[100];

	// puede contener un máximo de 16 caracteres (letras y números).
	char laboratorio[16];

	//una secuencia de hasta 4 caracteres numéricos. (0 a 9999)
	short int accion_medicamento;

	//una secuencia de hasta 2 caracteres numéricos. (0 a 99)
	short int forma_medicamento;

	//una secuencia de hasta 2 caracteres numéricos. (0 a 99)
	short int tamanio_medicamento;

	//una secuencia de hasta 2 caracteres numéricos. (0 a 99)
	short int via_administracion;

	//una secuencia de hasta 3 caracteres numéricos, una coma decimal y 2
	caracteres numéricos más.
	float precio;
};
```

Baldes

Para la implementación de los baldes se utilizaron vectores de la librería STL. Esto permite acceder en forma aleatoria cada uno los datos almacenados. Respecto a los elementos del balde, surgen 2 alternativas posibles:

- Almacenar sólo números enteros, índices del archivo.
- Almacenar registros de balde que contengan:
  - número entero, índice en el archivo;
  - valor de accion_medicamento;
  - valor de forma_medicamento;
  - valor de precio_medicamento;

Se optó por la segunda alternativa debido a que no requiere recurrir al archivo cuando se desea dividir el balde o filtrar sus elementos. Esto conlleva a un incremento en el uso de memoria principal pero disminuye notablemente la cantidad de accesos a disco.

Escalas

Cada escala es construida a partir de el análisis de datos y es almacenada en un vector ya que no posee ningún comportamiento especial, sólo sirve de referencia para acceder al Gridfile. Ejemplo:

posición	0	1	2	3	4
valor	0,0	25,0	50,0	100,0	500,0

Vector de Escala atributo **A**

A la hora de acceder a un elemento **e** cuyo **A = a**:

- Si 0,0 ⩽ **a** < 25,0 : **e** se corresponde con la posición 0 respecto a **A**;
- Si 25,0 ⩽ **a** < 50,0 : **e** se corresponde con la posición 1 respecto a **A**;

- Si 50,0 ⩽ **a** < 10,0 : **e** se corresponde con la posición 2 respecto a **A**;
- Si 100,0 ⩽ **a** < 500,0 : **e** se corresponde con la posición 3 respecto a **A**;
- Si 500,0 ⩽ **a** : **e** se corresponde con la posición 4 respecto a **A**;

Nota: Se asume que ∄ **e: a <** 0,0

Gridfile

El Gridfile es, básicamente, una matriz tridimensional de punteros a baldes donde cada dimensión se corresponde con una escala de igual tamaño. Su tamaño es estático, es decir que no varía durante la ejecución del programa.

La cantidad de celdas no se corresponde directamente con la cantidad de Baldes, ya que muchas celdas pueden apuntar a un mismo Balde en memoria.

Zonas

Una zona está formada básicamente por 2 coordenadas (x_1,y_1,z_1) y (x_2,y_2,z_2) que la delimita. No son estructuras complejas, sin embargo, de ellas depende la división y expansión del Gridfile.

Las estructuras mencionadas, interactúan constantemente de modo tal de poder brindar al índice la funcionalidad requerida. Como puede observarse en el diagrama de clases, Gridfile es quien conoce y controla todas las estructuras.

Funcionamiento

Consultas: Son realizadas a través de varios criterios y de distintas formas. Se pueden formar distintas combinaciones de criterios y además se puede variar entre consultar valores exactos y/o rangos de valores. En caso de no especificar algún criterio se toman todos sus valores.

Se obtienen los valores correspondientes a cada escala. Con estos, los Baldes del Gridfile y luego se filtran de dichos Baldes, los elementos que no se satisfacen la consulta.

Inserciones: Resulta la operación mas compleja. Para insertar un elemento en el Gridfile, se obtienen los valores correspondientes a cada una de las escalas y se forma una coordenada. Con esta se accede a un balde. En este paso pueden ocurrir que:

- El balde no está lleno: Se inserta el elemento (un registro de balde)
- El balde está lleno:
    - Se obtiene la Zona a la cual pertenece la coordenada;
    - Se redimensiona dicha Zona (se achica respecto a un atributo **A**);
    - Se crea una nueva Zona que ocupara el complemento de la Zona original.
    - Se crea un nuevo Balde, se insertan en él los elementos del Balde de la Zona origianl de acuerdo a **A**.
    - Se asigna el nuevo Balde a la nueva Zona.
    - Se redireccionan al nuevo Balde todas las celdas abarcadas por la nueva Zona.
    - Se inserta el elemento.

Bajas: Simplemente se obtiene el Balde correspondiente al elemento que se desea eliminar y luego se borra el elemento del Balde.

Discusión

Archivos

Cuando trabajamos con archivos, decidimos hacerlo a través de archivos binarios de registros de medicamentos. Para esto, construimos un binario con todos los registros a indexar y luego realizamos la carga, alta, bajas y modificaciones correspondientes. Esto nos permite trabajar con registros enteros y no con lineas de texto, acercándonos de esta forma a cómo funciona realmente un Gridfile.

En este caso sólo se trata de datos ficticios, pero en la realidad, muchas veces se busca encriptar los datos de un archivo de modo que no sean legibles tan fácilmente por los usuarios ajenos al sistema. Esto genera una menor tolerancia a corrupciones en los datos.

No se consideró necesario ordenar dicho archivo ya que se accede a sus registros a partir del Gridfile.

Respecto al almacenamiento del índice, lo ideal sería guardarlo en un archivo binario para luego cargarlo de modo tal de no tener que reconstruirlo en cada ejecución del sistema.

Eficiencia

El principal objetivo de la construcción de índices es efectivizar y reducir la cantidad de accesos a disco a la hora de hacer consultas respecto uno o mas criterios.

En el caso del Gridfile, el índice nos permite realizar búsquedas por uno más campos y por intervalos o valores exactos de dichos campos. Podemos afirmar que un Gridfile de 3 dimensiones funciona como muchos índices:

- índice para atributo **accion**;
- índice para atributo **forma**;
- índice para atributo **precio**;
- índice para atributos **accion** y **forma**;
- índice para atributos **accion** y **precio**;
- índice para atributos **forma** y **precio**;
- índice para atributos **accion**, **forma** y **precio**.

Naturalmente, si necesitamos almacenar cada uno de los índices mencionados, por separado, implicaría mucho mas espacio en disco.

A continuación se comparan y analizan la cantidad de accesos a disco de consultas de datos usando el índice Gridfile y sin usarlo, en ambos casos sobre un archivo desordenado de N registros.

Optimización

Para evitar el desperdicio de memoria principal, es fundamental elegir correctamente las escalas lineales de cada atributo. Este proceso se realizó fue realizado a mano y luego se ingresado en cada una de las escalas.

Del mismo modo es posible controlar las divisiones (cuando el balde a ingresar datos está lleno). La redimensión de las Zonas resulta óptima si en cada uno de los balde se almacena la mitad de la información.

De todos modos, resulta indispensable analizar los datos a ser almacenados de manera de no encontrarse con sorpresas en la ejecución. Resulta impensable desarrollar índices sin tener una noción de la información con la que se trabajará. Esto influye directamente en la confección de las escalas e indirectamente en el crecimiento de la estructura principal.

Otra forma de optimizar el crecimiento del Gridfile es realizando las particiones de las Zonas de acuerdo a distintos atributos. En nuestro caso, hemos realizado dichas particiones de acuerdo al atributo “precio” (dimensión “Z” de la matriz) , pero el particionado es limitado en esta forma de trabajo.

Conclusiones

Los objetivos se cumplieron satisfactoriamente. Se desarrolló con éxito un índice Gridfile capaz de soportar eficientemente la información contenida en el archivos. Luego se agregaron los datos del archivo *altas_medicamentos.dat* y finalmente se dieron de baja los medicamentos contenidos en *bajas_medicamentos.dat*.

La falta de tiempo y de capacidades técnicas no nos han permitido indagar en algunos aspectos de los Gridfile como lo son las particiones de Zonas o Escalas respecto a múltiples criterios y la fusión de baldes semi vacíos que podría realizarse en la baja de elementos.

Hemos observado y corroborado la disminución de accesos a disco y la importancia de implementar índices para manejar grandes volúmenes de información, como lo son las enormes bases de datos que nos rodean a diario en nuestro.

A diferencia de otros proyectos de cátedra, la construcción de este tipo de índices no consta sólo de desarrollo en lenguaje C++, si no que requiere también un análisis exhaustivo de los datos que serán indexados. Como se mencionó anteriormente, es fundamental realizar dicho análisis a fin de garantizar una estructura homogénea, estable y uniforme.