Archivo Secuencial Indexado

Andrés Arcia

Un archivo secuencial indexado esta básicamente compuesto de una estructura secuencial, que se llamará archivo maestro y uno o más archivos índices. Los archivos índices, se encargarán de dar un acceso más rápido a los datos almacenados en el archivo secuencial. La rapidez del archivo depende, comparativamente con otros archivos, del numero de registros.

Cada registro de un archivo índice se corresponde con un bloque de otro archivo índice o con un bloque en el archivo principal. Esto implica que cuando se quiere extraer información precedida por un índice, tiene que cargarse toda la información a la que el índice apunta. El tamaño de esta información es el tamaño del bloque.

Estructuras y variables necesarias:

```
registro_maestro
        bool disponible;
        long proximo;
        Registro_Data data;

registro_indice
        Registro_Data::Clave clave;
        long tamaño_bloque;
```

Estructura del índice maestro (primer índice)
registro_indice * indice_maestro (será un arreglo de registros índice)

```
long tamaño_ppal;
long tamaño_desb;
string nombre_ppal;
string nombre_desb;
string * archivos_indice; // vector de nombres de archivos índice.
int niveles;  // numero de archivos índice.
```

Calculos internos:

```
long elem_indices; // numero de elementos en indices.
long tam_reg_ppal;
long tam_reg_indice;
long num_bloques_ppal;
long num_reg_blq;
long num_reg_blq_ind;
long num_reg_ppal;
```

```
long num_reg_desb;
long num_reg_disp_desb;
long tam_indice_maestro;
```

Existen algunos métodos que nos ayudarán a llevar a cabo esta tarea:

int crear_indice(string &)

int subir_indice_maestro()

Por otro lado están las rutinas de acceso público para operar sobre los archivos:

Constructor()

crear()

abrir()

vaciar()

buscar()

Por último están un conjunto de rutinas que sirven de soporte a las anteriores para poder extraer y depositar información en los archivos en forma de bloques.

El método crear se encarga de crear un archivo maestro para la organización ISAM a partir de otro archivo de entrada.

Podemos llamar entonces al archivo de entradas como: entrada y nuestro archivo maestro: maestro. De aquí salen un archivo maestro, un archivo de desborde y un archivo de índices.

```
crear()
1. abrir entrada
2. num_reg_ppal = 0
3. R.M. (!entrada.eof())
   3.1 LEER_SEC(entrada, R)
   3.2 reg_ppal.data = R, reg_ppal.disp=falso, reg_ppal.prox=-1
   3.3 r = ESCRIBIR_PPAL(maestro, num_reg_ppal, &reg_ppal, 1)
   3.4 Si (r = Verdadero)
       3.4.1 num_reg_ppal ++
4. tam_ppal = num_reg_ppal
5. cerrar entrada
6. num_blq_ppal = techo(num_reg_ppal/num_reg_blq)
7. abrir desborde
8. reg_desb.disp = verdadero
9. num_reg_desb = num_reg_ppal * 0.20
10. R.P. (n_disp = 0; n_disp < num_reg_desb; disp++)
     10.1 r=ESCRIBIR_PPAL(desborde, n_disp, reg_desb, 1)
     10.2 Si (r == Verdadero)
          10.2.1 pila_disponible.push(n_disp)
11.cerrar desborde
12.crear_indices(maestro)
```

crear_indices()
Son necesarios:
reg_indice
reg_maestro

1. niveles = techo (log (num_blq_ppal) / log (tam_buffer / tam_reg_indice)) ?
2. Construir un arreglo de nombres de índices (nom_indices)
3. R.P. (i=0, j=0; i<tam_ppal; i+=num_reg_blq)
   3.1 Si (LEER(maestro, i, reg_maestro, 1)
       3.1.1 reg_indice.clave = reg_maestro.data.clave
       3.1.2 reg_indice.indice_blq = i
       3.1.3 ESCRIBIR_REG_INDICE(nom_indice[0], j++, &reg_indice, 1)
4. num_reg_blq_ind = tam_buffer / tam_reg_indice
5. R.P. (k=1; k<niveles; k++)
    5.1R.P. (i=0, j=0; continuar=verdad; i+=num_reg_blq_ind)
        5.1.1Si (LEER_REG_INDICE(nom_indice[k-1], i, &reg_indice, 1))
            5.1.1.1 reg_indice.indice_blq = i;
            5.1.1.2 ESCRIBIR_REG_INDICE(nom_indice[k],j++, reg_indice,1)
          SI NO
            5.1.1.3 continuar = falso
6. Fin

abrir()

Se necesita registro ppal.

1. Determinar el número de registros del ppal (num_reg_ppal)
2. num_blq_ppal = num_reg_ppal / num_reg_blq
3. num_reg_desb = num_reg_ppal * 0.20
4. reg_desb.disp = verdadero
5. R.P. (n_disp = 0; n_disp < num_reg_desb; n_disp++)
   5.1 Si (ESCRIBIR(desborde, n_disp, reg_desb, 1))
       5.1.1 pila_disponibles.push(n_disp)
6. crear_indices
7. fin

buscar()

1. subir_indice_maestro()
2. encontro = falso, i=0
3. R.M. (i < tam_ind_maestro – 1 && encontro = falso)
   3.1Si (dato < indice_maestro[i+1].clave)
      3.1.1 encontro = verdadero
      si no
      3.1.2 i++
4. iblq = indice_maestro[i].indice_blq
   (buscar en mem. sec en arch. indices)
5. hacer un vector de reg. índices: vec_reg_ind.
6. R.P. (i=niveles – 2; i >=0; i--)
   6.1 reg_leidos = LEER_REG_IND(nom_indice[i], iblq, vec_reg_ind, num_reg_blq_ind)
   6.2 j=0
   6.3 encontro = falso
   6.4 R.M. ((j < reg_leidos -1) && encontro == falso)
      6.4.1 Si (dato < vec_reg_ind[j+1].clave)
         6.4.1.1 encontro = verdadero
         6.4.1.2 iblq = vec_reg_ind[j].ind_bloque
         si no
         6.4.1.3 j++
   6.5Si (encontro = falso)
      6.5.1 iblq = vec_reg_ind[j].ind_bloque

(busqueda dentro del bloque del archivo principal)

7. Crear un vector para registros del archivo principal (vec_reg_ppal)
8. reg_leidos = LEER(maestro, iblq, vec_reg_ppal, num_reg_blq_ppal)
9. i = 0
10. encontro = falso
11. R.M. (i < reg_leidos && encontro = falso)
   11.1 Si (dato.clave = vec_reg[i].data.clave)
      11.1.1 dato = vec_reg[i].data
      11.1.2 encontro = verdadero
   si no
   11.2 Si (vec_reg[i].data < dato ^ (i + 1) == reg_leidos)
      11.2.1
      11.2.2 prox = vec_reg[i].prox
      11.2.3 R.M. (prox != -1)
         11.2.3.1 Si (LEER(desborde, prox, reg_desb, 1))
            Si (dato.clave == reg_desb.data.clave)
               dato = reg_desb.data
               encontro = verdadero
         11.2.3.2 prox = reg_desb.prox
      sino
      11.2.4 i++

12. Si (encontrado = verdadero)
    12.1 retornar dato.
    si no
    12.2 retornar NO ENCONTRADO

agregar(R)

//Busqueda en el Indice Maestro
subir_indice_maestro()

1. encontro = falso, i=0
2. R.M. (i < tam_ind_maestro – 1 && encontro = falso)
    2.1 Si (dato < indice_maestro[i+1].clave)
      3.1.1 encontro = verdadero
     si no
      3.1.2 i++
3. iblq = indice_maestro[i].indice_blq
   (buscar en mem. sec en arch. indices)
4. hacer un vector de reg. índices: vec_reg_ind.
5. R.P. (i=niveles – 2; i >=0; i--)
   5.1 reg_leidos = LEER_REG_IND(nom_indice[i], iblq, vec_reg_ind, num_reg_blq_ind)
   5.2 j=0
   5.3 encontro = falso
   5.4 R.M. ((j < reg_leidos -1) && encontro == falso)
       5.4.1 Si (dato < vec_reg_ind[j+1].clave)
             5.4.1.1 encontro = verdadero
             5.4.1.2 iblq = vec_reg_ind[j].ind_bloque
             si no
             5.4.1.3 j++
  5.5 Si (encontro = falso)
       5.5.1 iblq = vec_reg_ind[j].ind_bloque

En este punto obtenemos el indice en el Archivo Maestro.

// Preparación del Registro para ser almacenado
7. Crear arreglo de registros del tamaño del bloque (**v_reg**).
8. reg_leidos = leer(maestro, iblq, v_reg, num_reg_blq)

// Verificar si hay una localidad disponible en el area principal
// También se determina el sitio donde va almacenado

i=0; encontro = falso; num_disp = 0;   pos = reg_leidos – 1; libre = -1;
band = falso

R.M. (i < reg_leidos)
9.1Si (v_reg[i].disp = verdadero ^ libre = -1)
9.1.1 libre = i
9.2Si (dato.clave < v_reg[i].data.clave ^ band = falso)
9.2.1 pos = i – 1
9.2.2 band = verdadero
9.3 i++
10.Si libre >= 0
10.1 Hay espacio en el area principal, en la posición **libre**
10.2 v_reg[libre] = // Rellenar de información
10.3 Ordenar el vector en memoria Ram (v_reg)
10.4 Utilizar cualquier método: burbuja, inserción, etc.
10.5 ESCRIBIR_PPAL(maestro, iblq, v_reg, reg_leidos)
10.6 Si pos == -1 (// se modifico el registro ancla que sirve de indice)
10.6.1 crear_indices()
10.6.2 retornar
si no
10.6.3 Si (pos >0) (// No hay espacio en el principal)
10.6.4 prox = pila_disponibles.pop()
10.6.5 Si (prox == -1)
10.6.5.1. Retornar ERROR
10.6.6 Preparar registro nuevo (aux.prox = v_reg[pos].prox)
10.6.7 ESCRIBIR_PPAL(desborde, iblq, &aux, 1)
10.6.8 num_reg_disp_desb --
10.6.9 vreg_[pos].prox = prox (inserción al principio de la lista)
10.6.10 ESCRIBIR (maestro, iblq, v_reg, reg_leidos)
10.6.11 retornar EXITO
si no
10.6.12 // El registro es el proximo registro ancla.
10.6.13 prox = pila_desborde.pop()
10.6.14 Si prox = -1
10.6.14.1.Retornar ERROR
10.6.15 ESCRIBIR_PPAL(desborde, prox, &v_reg[0], 1)
10.6.16 num_reg_disp_desb--
10.6.17 v_reg[0].data = data
10.6.18 v_reg[0].prox = prox
10.6.19 ESCRIBIR(maestro, iblq, v_reg, reg_leidos)
10.6.20 crear_indices

Eliminar

Una vez conseguido el bloque donde se encuentra el registro (**iblq**), se procede a eliminarlo físicamente.

1. reg_leidos = LEER_PPAL(maestro, iblq, v_reg, num_reg_blq)
2. i=0; encontro = falso
3. R.M. (i<reg_leidos ^ encontro = falso)
   3.1. Si (dato == v_reg[i].data)
      3.1.1. v_reg[i].data = // valor especial indicando vacio
      3.1.2. v_reg[i].disp = 1
      3.1.3. ESCRIBIR_PPAL (maestro, iblq, v_reg, num_reg_blq)
      3.1.4. encontro = verdadero
   SI NO
      3.1.5. Si ((dato < v_reg[i].data ^ i > 0) v (i+1 == reg_leidos))
         3.1.5.1. // Buscamos en el desborde
         3.1.5.2. Si ((i+1) == reg_leidos)
            3.1.5.2.1. i++
         3.1.5.3 prox = v_reg[i].prox; primero_lista = falso
         3.1.5.4 anterior = i
         3.1.5.5 R.M. (prox != -1)
            3.1.5.5.1. Si (LEER_PPAL(desborde, prox, &aux, 1))
               3.1.5.5.1.1. Si (dato == aux.dato ^ primero_lista = falso)
                  1. aux.data = // valor especial indicando vacio
                  2. aux.disp = 1
                  3. LEER_PPAL (desborde, anterior, &aux2, 1)
                  4. aux2.prox = aux.prox
                  5. ESCRIBIR_PPAL(desborde, anterior, &aux2, 1)
                  6. ESCRIBIR_PPAL(desborde, prox, &aux, 1)
                  SI NO
                  7. Si (dato == aux.dato ^ primero_lista = true)
                     // Igual al anterior, pero reemplazar el registros
                     // del principal.
            3.1.5.5.2anterior = prox
            3.1.5.5.3prox = aux.prox
         SI NO
            3.1.5.6 i++