Conceptos de Bases de Datos

Agustin Murray

March 25, 2024

Contents

1 Conceptos generales de bases de datos

1.1 Conceptos Basicos

- Se considera **Base de Datos**, a una coleccion o conjunto de datos interrelacionados con un proposito especifico vinculado a la resolucion de un problema del mundo real.
- Cualquier informacion dispuesta de manera adecuada para su tratamiento por una computadora.
- Una coleccion de archivos diseñados para servir a multiples aplicaciones.

1.2 Origenes de las BD

En los origenes de las **BD**, las unidades de almacenamiento de gran volumen eran muy lentas, entonces se buscaba reducir el acceso a los mismos. Para esto, era necesario repetir datos en distintos puestos de trabajo y al final del dia actualizar los mismos en todas las unidades para evitar problemas de perdida/modificacion de informacion.
Con el tiempo, la tecnologia fue avanzando y los sistemas de informacion evolucionaron. Se logra integrar aplicaciones, interrelacionar archivos y eliminar la redundancia de datos.

1.3 Gestores de Bases de Datos

Un **Sistema de Gestion de Bases de Datos (SGBD)** consiste en un conjunto de programas necesarios para acceder ya administrar una BD.
Actualmente, cualquier sistema de software necesita interactuar con informacion almacenada en una BD y para ello requiere del soporte de un SGBD.
Un SGBD posee dos tipos diferentes de lenguajes: uno para especificar el esquema de una BD, y el otro para la manipulacion de los datos.
La definicion del esquema de una BD implica:

- Diseño de la estructura que tendra efectivamente la BD.
- Describir los datos, la semantica asociada y las restricciones de consistencia.

Para ello se utiliza un lenguaje especial, llamado **Lenguaje de Definicion de datos (LDD)**.El resultado de compilar lo escrito con el LDD es un archivo llamado Diccionario de Datos.Un Diccionario de Datos es un archivo con metadatos, es decir, datos acerca de los datos.

Los objetivos mas relevantes de un SGBD son:

- **controlar la concurrencia:** varios usuarios pueden acceder a la misma informacion en un mismo periodo de tiempo. Si el acceso es para consulta, no hay inconvenientes, pero si mas de un usuario quiere actualizar el mismo dato a la vez, se puede llegar a un estado de inconsistencia que, con la supervicion del SGBD, se puede evitar.
- **Tener control centralizado:** tanto de los datos como de los programa que acceden a los datos.

- **Facilitar el acceso a los datos:** dado que provee un lenguaje de consulta para recuperacion rapida de informacion.
- **Proveer seguridad para imponer restricciones de acceso:** se debe difinir explicitamente quienes son los usuarios autorizados a acceder a la BD.
- **Mantener la integridad de los datos:** esto implica que los datos incluidos en la BD respeten las condiciones establecidas al definir la estructura de la BD y que, ante una falla del sistema, se posea la capacidad de restauracion a la situacion previa.

2 Archivos, estructuras y operaciones basicas

2.1 Definicion

Un **archivo** es una coleccion de registros semejantes, guardados en dispositivos de almacenamiento secundario de una computadora.
Los archivos se caracterizan por el crecimiento y las modificaciones que se efectuan sobre estos. El crecimiento indica la incorporacion de nuevos elementos, y las modificaciones involucran alterar datos contenidos en el archivo, o quitarlos.

2.2 Aspectos fisicos

Almacenamiento primario (RAM):

- Capacidad de almacenamiento limitada.
- Volatil.
- Alto costo.
- Acceso rapido (orden de los nanosegundos)

Almacenamiento secundario (cintas y discos):

- Alta capacidad de almacenamiento.
- No volatil.
- Menor costo que el primario.
- Acceso lento a comparacion del primario (orden de los milisegundos). Se busca optimizar este aspecto:
  1. Busqueda de un unico dato: Obtencion en un intento o en pocos.
  2. Busqueda de varios datos: obtencion de todos de una sola vez.
- **Cintas:** acceso secuencial, economico, estable en diferentes condiciones ambientales, facil de transportar.
- **Discos:** acceso directo, se almacenan datos en sectores.

2.3 Niveles de vision

- **Archivo fisico:** es el archivo residente en la memoria secundaria y es administrado (ubicacion, tipo de operaciones disponibles) por el sistema operativo.
- **Archivo logico:** es el archivo utilizado desde el algoritmo. Cuando el algoritmo necesita operar con un archivo, genera una conexion con el sistema operativo, el cual sera el responsable de la administracion. Esta accion se denomina independencia fisica.

2.4 Organizacion interna de los datos

- **Secuencia de bytes:** Se determina como unidad mas pequeña de L/E al byte. No se puede determinar facilmente el comienzo y el final de cada dato por lo que generalmente son archivos de texto.
- **Campos:** Se determina como unidad mas pequeña de L/E al campo. El campo es un item de datos elemental y se caracteriza por su tipo de dato y su tamaño.
- **Registros:** Se determina como unidad mas pequeña de L/E al registro. El registro es un conjunto de campos agrupados que definen un elemento del archivo. Los campos internos a un registro deben estar logicamente relacionados, como para ser tratados como una unidad.

2.5 Acceso a informacion contenida en los archivos

Basicamente, se pueden definir tres formas de acceder a los datos de un archivo:

- **Secuencial:** el acceso a cada elemento de datos se realiza luego de haber accedido a su inmediato anterior. El recorrido es, entonces, desde el primero hasta el ultimo de los elementos, siguiendo el orden fisico de estos.
- **Secuencial indizado:** el acceso a los elementos de un archivo se realiza teniendo presente algun tipo de organizacion previa, sin tener en cuenta el orden fisico.
- **Directo:** es posible recuperar un elemento de dato de un archivo con un solo acceso, conociendo sus caracteristicas,k mas alla de que exista un orden fisico o logico predeterminado.

2.6 Operaciones basicas sobre archivos

Para poder operar con archivos, son necesarias una serie de operaciones elementales disponibles entodos los lenguajes de programacion que utilicesn archivos de datos. Estas operaciones incluyen:

- La definicion del archivo logico.
- La definicion de la forma de trbajo del arcchivo (creacino inicial, utilizacion).
- La administracion de datos (L/E info).

2.6.1 Definicion de archivos

Como cualquier otro tipo de datos, los archivos necesitan ser definidos. Se reserva la palabra clave **file** para indicar la definicion del archivo.

```
Var archivo_logico: file of tipo_de_dato;
```

Otra opcion para definir archivos se presenta a continuacion:

```
Type archivo= file of tipo_de_dato;
Var archivo_logico: archivo;
```

2.6.2 Correspondencia archivo logico - archivo fisico

Se debe indicar que el archivo logico utilizado por el algoritmo se corresponde con el archivo fisico adminstrado por el sistema operativo. La sentencia encargada de hacer esta correspondencia es:

```
Assign (nombre_logico, nombre_fisico);
```

2.6.3 Apertura y creacion de archivos

Hasta el momento, se ha detallado como definir un archivo y se ha esctablecido la relacion con el nombre fisico. Para operar con un archivos desde un algoritmo, se debe realizar la apertura.

- La operacin **rewrite** indica que el archivo va a ser creado y, por lo tanto, la unica operacion valida sobre el mismo es escribir informacion.
- La operacino **reset** indica que el archivo ya existe y, por lo tanto, las operaciones validas sobre el mismo son L/E de informacion.

```
rewrite(nombre_logico);
reset(nombre_logico);
```

2.6.4 Cierre de archivos

Para mantener valida la marca de end of file (EOF) se cierra el archivo de la siguiente forma:

```
close(nombre_logico);
```

2.6.5 Lectura y escritura de archivos

Para leer o escribir informacion en un archivo, las instrucciones son:

```
read(nombre_logico, var_dato);
write(nombre_logico, var_dato);
```

Buffers de memoria: Las lecturas y escrituras desde o hacia un archivo se realizan sobre buffers. Se denomina buffer a una memoria intermedia (ubicada en RAM) entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados definitivamente en la memoria secundaria, o dodnde los datos residen una vez recuperados de dicha memoria secundaria. La razon de esto es la mejora de performance al trabajar con mayor frecuencia en memoria principal.
La operacion read lee desde un buffer y, en caso de no contar con informacion, el sistema operativo realiza automaticamente una operacion input, trayendo mas informacion al buffer. La diferencia radica en que cada operacion input transfiere desde el disco una serie de registros. De esta forma, cada determinada cantidad de instrucciones read, se realiza una operacion input.
De forma similar procede la operacion write; en este caso, se escribe en el buffer, y si no se cuenta con espacio suficiente, se descarga el buffer a disco por medio de una operacion output, dejandolo nuevamente vacio.

2.6.6 Operaciones adicionales con archivos

- Control fin de datos:

```
eof(nombre_logico);
```

  La funcion retornara verdadero si el puntero del archivo referencia e EOF, y falso en caso contrario.

- Control de tamaño del archivo:

```
filesize(nombre_logico);
```

- Control de posicion de trabajo dentro del archivo:

```
Filepos(nombre_logico);
```

- Ubicacion fisica en alguna posicion del archivo:

```
seek(nombre_logico, posicion);
```

- Truncamiento de un archivo a partir de la posicion acutal:

```
truncate(nombre_logico);
```

3 Algoritmica clasica sobre archivos

3.1 Objetivo

Desarrollar algoritmos considerados clasicos en la operatoria de archivos secuenciales. Estos algoritmos se resumen en tres tipos: de actualizacion, merge y corte de control.

El primer caso, con todas sus variantes, permite introducir problemas donde se actualiza el contenido de un archivo resumen o "maestro", a partir de un conjunto de archivos con datos vinculados a este archivo maestro.

En el segundo caso, se dispone de la informacion distribuida en varios archivos que se reune para generar un nuevo archivo, producto de la union de los anteriores.

Por ultimo, el corte de control, muy presente en la operatoria de BD, determina situaciones done, a partir de informacion contenida en archivos, es necesario generar reportes que resuman el contenido, con un formato especial.

Se denomina archivo maestro al archivo que resume informacion sobre un dominio de problema especifico. Ejemplo: el archivo de productos de una empresa que contiene el stock actual de cada producto. Por otra parte, se denomina archivo detalle al archivo que contiene novedades o movimientos realizados sobre la informacion almacenada en el maestro. Ejemplo: el archivo con todas las ventas de los productos de la empresa realizadas en un dia particular.

3.2 Actualizacion de un archivo maestro con un archivo detalle (I)

Presenta la variante mas simple del proceso de actualizacion. Las precondiciones del problema son las siguientes:

- Existe un archivo maestro.
- Existe un unico archivo detalle que modifica al maestro.
- Cada registro del detalle modifica a un registro del maestro. Esto significa que solamente apareceran datos en el detalle que se correspondan con datos del maestro. Se descarta la posibilidad de generar altas en ese archivo.
- No todos los registros del maestro son necesariamente modificados.
- Cada elemento del maestro que se modifica es alterado por uno y solo un elemento del archivo detalle.
- Ambos archivos estan ordenados por igual criterio. Esta precondicion, considerada esencial, se debe a que hasta el momento se trabaja ocn archivos de datos de acuerdo con su orden fisico.

```
begin
  assign(mae1, 'maestro');
  assign(det1, 'detalle');
  reset(mae1);
  reset(det1);
  while not eof(det1) do begin
    read(mae1, regm);
```

```
    read(det1, regd);
    while(regm.cod<>regd.cod) do read(mae1,regm);
    regm.stock := regm.stockk - regd.cant_vendida;
    seek(mae1, filepos(mae1)-1);
    write(mae1, regm);
  end;
  close(det1);
  close(mae1);
end.
```

3.3 Actualizacion de un archivo maestro con un archivo detalle (II)

Este es otro caso, levemente diferente del anterior; solo se modifica una precondicion del problema y hace, de esta forma, variar el algoritmo resolutorio.

- Cada elemento del archivo maestro puede no ser modificado, o ser modificado por uno o mas elementos del detalle (hay registros repetidos).

```
program ejemplo_3_3;
  const valoralto='9999';
  type str4 = string[4]

  procedure leer(var archivo: detalle;
  var dato: venta_prod);
  begin
    if not eof(archivo) then read (archivo, dato)
    else dato.cod:=valoralto;
  end;

  begin
    assign(mae1, 'maestro');
    assign(det1, 'detalle');
    reset(mae1);

\include{1.tex}
    reset(det1);
    read(mae1, regm);
    read(det1, regd);

    while(regd.cod<>valoralto) do
    begin
      aux:=regd.cod;
      total:=0;

      while(aux = regd.cod) do
      begin
        total := total + regd.cant_vendida;
        leer(det1,regd);
      end;
```

```
      while (regm.cod<>aux) do read(mae1,regm);

      regm.cant := regm.cant - total;

      seek(mae1, filepos(mae1)-1);

      write(mae1, regm);

      if not eof(mae1) then read(mae1,regm);
    end;
    close(det1);
    close(mae1);
  end.
```

El procedimiento de lectura, denominado leer, es el responsable de realizar el read correspondiente sobre el archivo, en caso de que este tuviera mas datos. En caso de alcanzar el fin de archivo, se asigna a la variable dato.cod, por la cual el archivo esta ordenado, un valor imposible de alcanzar en condiciones normales de trabajo. Este valor indicara que el puntero del archivo ha llegado a la marca de fin.

3.4 Actualizacion de un archivo maestro con N archivos detalle

Bajo las mismas consignas del ejemplo anteriosr, se plantea un procesa de actualizacion donde, ahora, la cantidad de archivos detalle se lleva a N (siendo $N > 1$) y el resto de las precondiciones son las mismas.

El ejemplo 3.4 presenta la resolucion de un algoritmo de actualizacion a partir de tres archivos detalle. Para ello, se agrega un nuevo procedimiento, denominado minimo, que actua como filtro. El objetivo de este proceso a partir de la iunformacion recibida es retornar el elemento mas pequeño de acuerdo con el criterio de ordenamiento del problema. El objetivo del procedimiento minimo es determinar el menor de los tres elementos recibidos de cada archivo y leer otro registro del archivo desde donde provenia ese elemento.

```
procedure leer(var archivo: detalle, var dato:venta_prod);
begin
  if not eof(archivo) then read(archivo, dato)
  else dato.cod := valoralto;
end;

procedure minimo(var r1,r2,r2,min:venta_prod);
begin
  if(r1.cod<=rd2.cod) and (r1.cod<=r3.cod) then
  begin
    mon:=r1;
    leer(det1,r1);
  end
  else if (r2.cod<=r3.cod) then
  begin
    min:=r2;
    leer(det,r2);
```

```
  end
  esle
  begin

\include{1.tex}
    min := r3;
    leer(det3,r3);
  end;
end;

begin
  {asignacion y apertura de archivos correspondientes}

  while(min.cod<>valoralto) do
  begin
    aux:=min.cod;
    total_vendido:=0;
    while(aux=min.cod) do
    begin
      total_vendido:=total_vendido+min.cantvendida;
      minimo(regd1,regd2,regd3,min);
    end;
    while(regm.cod<>min.cod) do read(mae1,regm);

    regm.cant:=regm.cant-total;
    seek(mae1, filepos(mae1)-1);
    write(mae1,regm);
    if not eof(mae1) then read(mae1,regm);
  end;
  close(det1);
  close(det2);
  close(det3);
  close(mae1);
end.
```

3.5 Proceso de generacion de un nuevo archivo a partir de otros existentes. Merge

3.5.1 Primer ejemplo

El primer ejemplo plantea un problema muy simple. Las precondiciones son las siguiente:

- Se tiene informacion en tres archivos detalle.
- Esta informacion se encuentra ordenada por el mismo criterio en cada caso.
- La informacion es disjunta; esto significa que un elemento puede aparecer una sola oportunidad en todo el problema.

```
begin
  leer(det1, regd1);
  leer(det2, regd2);
  leer(det3, regd3);

  minimo(regd1, regd2, regd3, min);
  while(min.codigo<>valoralto) do
  begin
    write(mae1, min);
    minimo(regd1, regd2, regd3, min);
  end;
  close(det1);
  close(det2);
  close(det3);
  close(mae1);
end.
```

3.5.2 Segundo ejemplo

Como segundo ejemplo se presenta un problema similar, pero ahora los elementos se pueden repetir dentro de los archivos detalle, modificando de esta forma la tercera precondicion del ejemplo anterior. El resto de las precondiciones permanecen inalteradas.

```
begin
  leer(det1, regd1);
  leer(det2, regd2);
  leer(det3, regd3);

  minimo(regd1, regd2, regd3, min);

  while(min.codigo<>valoralto) do
  begin
    codprod:=min.codigo;
    cantotal:=0;
    while(codprod=min.codigo)
    begin
      cantotal:=cantotal+min.cant;
      minimo(regd1,regd2,regd3,min);
    end;
    write(mae1,min);
  end;
  close(det1);
  close(det2);
  close(det3);
  close(mae1);
end.
```

3.6 Corte de control

Se denomina corte de control al proceso mediante el cual la informacion de un archivo es presentada en forma organizada de acuerdo con la estructura que tiene el archivo. Suponga que se almacena en un archivo la informacion de ventas de una cadena de electrodomesticos. Dichas ventas han sido efectuadas por los vendedores de cada sucursal de cada ciudad de cada provincia del pais. Luego, es necesario informar al gerente de ventas de la empresa el total de ventas producidas.

Deben tenerse en cuenta las siguientes precondiciones:

- El archivo se encuentra ordenado por provincia, ciudad, sucursal y vendedor.
- Se debe informar el total vendido en cada sucursal, ciudad y provincia, asi como el total final.
- En diferentes provicinas pueden existir ciudades con el mismo nombre, o en diferentes ciudades pueden existir sucursales con igual denominacion.

```
begin
  {asignaciones y aperturas correspondientes.}
  leer(archivo, reg);
  total:=0;
  while(reg.Provincia<>valoralto) do
  begin
    writeln('Provincia:', reg.Provincia);
    prov:=reg.Provincia;
    totprov:=0;

    while(prov=reg.Provincia) do
    begin
      writeln('Ciudad:', reg.Ciudad);
      ciudad:=reg.Ciudad;
      totciudad:=0;

      while(prov=reg.Provincia) and (Ciudad=reg.Ciudad) do
      begin
        writeln('Sucursa:',reg.sucursal);
        sucursal:=regSucursal;
        totsuc:=0;
        while((prov=reg.Provincia) and
        (Ciudad=reg.Ciudad) and Sucursal=reg.Sucursal) do
        begin
          write('Vendedor:', reg.Vendedor);
          writeln(reg.MontoVenta);
          totsuc:=totsuc+reg.MontoVenta;
          leer(archivo, reg);
        end;
        writeln('Total sucursal', totsuc);
        totciudad:=totciudad+totsuc;
      end;
      writeln('Total ciudad', totciudad);
      totprov:=totprov+totciudad;
```

```
      end;
      writeln('Total provincia', totprov);
      total:=total+totprov;
    end;
    writeln('Total empresa', total);
    close(archivo);
end.
```

4 Eliminacion de datos. Archivos con registros de longitud variable

4.1 Proceso de bajas

Se denomina **proceso de baja** a aquel proceso que permite quitar informacion de un archivo.
El proceso de baja puese ser analizado desde dos perspectivas diferentes: aquella ligada con la algoritmica y performance necesarias para borrar la informacion, y aquella que tiene que ver con la necesidad real de quitar informacion de un archivo en el contexto informactico actual.
En la actualidad, las organizaciones que disponen de BD consideran la informacion su bien mas preciado. De esta forma, el concepto de borrar informacion queda condicionado. En general, el conocimiento adquirido no se quita, sino que se preserva en archivos o repositorios historicos.
El proceso de baja puede llevarse a cabo de dos modos diferentes:

- **Baja fisica:** consiste en borrar efectivamente la informacion del archivo, recuperando el espacio fisico.
- **Baja logica:** consiste en borrar la informacion del archivo, pero sin recuperar el espacio fisico respectivo.

4.1.1 Baja fisica generando un nuevo archivo de datos

```
program ejemplo_4_1;
...
begin
...
while(reg.Nombre<>'Carlos Garcia')
begin
  write(archivonuevo, reg);
  leer(archivo, reg);
end;
leer(archivo, reg);
while(reg.Nombre<>valoralto)do
begin
  write(archivonuevo, reg);
  leer(archivo,reg);
end;
...
end.
```

Un analisis de performance basico determino que este metodo necesita leer tantos datos como tenga el archivo original y escribir todos los datos, salvo el que se elimina. Esto significa n lecturas y $n-1$ escrituras; en ambos casos, las lecturas y escrituras se realizan en forma secuencial.

4.1.2 Baja fisica utilizando el mismo archivo de datos

```
program ejemplo_4_2;
...
begin
...
while(reg.Nombre<>'Carlos Garcia')
begin
  leer(archivo,reg);
end;
leer(archivo, reg);
while(reg.nombre<>valoralto)do
begin
  seek(archivo, filepos(archivo)-2);
  write(archivo, reg);
  seek(archivo, filepos(archivo)+1);
  leer(archivo, reg);
end;
end.
```

El analisis de performance, para este ejemplo, determina que la cantidad de lecturas a realizar es n, en tanto que la cantidad de escrituras dependera del lugar donde se encuentre el elemento a borrar; en el peor de los casos, deberan realizarse nuevamente $n-1$ escrituras. No se necesita tanta memoria secundaria como en el ejemplo anterior.

4.1.3 Baja logica

Se realiza una baja logica sobre un archivo cuando el elemento que se desea quitar es marcado como borrado, pero sigue ocupando el espacio dentro dle archivo. La ventaja del borrado logico tiene que ver con la performance, baste con localizar el registro a eliminar y colocar sobre el una marca que indique que se encuentra no disponible. Entonces, la performance necesaria para llevar a cabo esta operacion es de tantas lecturas como sean requeridas hasta encontrar el elemento a borrar, mas una sola escritura que deja la marca de borrado logico sobre el registro.
La desventaja de este metodo esta relacionada con el espacio en disco. Al no recuperarse el espacio borrado, el tamaño del archivo tiene a crecer continuamente.

```
program ejemplo_4_3;
...
begin
...
while(reg.Nombre<>'Carlos Garcia') do
begin
  leer(archivo, reg);
end;
reg.nombre:='***' {marca de borrado}
Seek(archivo, filepos(archivo)-1);
write(archivo, reg);
close(archivo);
end.
```

4.2 Recuperacion de espacio

El proceso de baja logica marca la informacion de un archivo como borrada. Ahora bien, esa informacion sigue ocupando espacio en el disco rigido. La pregunta a responde seria: que hacer con dicha informacion? Hay dos respuestas posibles:

- **Recuperacion de espacio:** periodicamente utilizar el proceso de baja fisica para realizar un proceso de compactacion del archivo. El mismo consiste en quitar todos aquellos registros marcados como borrados, utilizando para ello cualquiera de los algoritmos vistos anteriormente para borrado fisico.
- **Reasignacion del espacio:** otra alternativa posible consiste en recuperar el espacio, utilizando los lugares indicados como borrados para el ingreso (altas) de nuevos elementos al archivo.

4.2.1 Reasignacion de espacio

Esta tecnica consiste en reutilizar el espacio indicado como borrado para que nuevos registros se inserten en dichos lugares. Asi, el proceso de alta discutido en capitulos anteriores se veia veria modificado; en lugar de avanzar sobre la ultima posicion del archivo(donde se encuentra la marca de **EOF**), se debe localizar alguna posicion marcada como borrada para insertar el nuevo elemento en dicho lugar.
Este proceso puede realizarse buscando los lugares libres desde el comienzo del archivo, pero se debe considerar que de esa manera seria muy lento. La alternativa consiste en recuperar el espacio de forma eficiente. Para ello, a medida que los datos se borran del archivo, se genera una lista encadenada invertida con las posiciones borradas.

4.3 Campos y registros con longitud variable

La informacion en una archivo siempre es homogenea. Esto es, todos los elementos almacenados en el son del mismo tipo. De esta forma, cada uno de los datos es del mismo tamaño, generando lo que se denomina archivos con registros de longitud fija.
Administrar archivos con registros de longitud fija tiene algunas importantes ventajas: el proceso de entrada y salida de informacion, desde y hacia los buffers, es responsabilidad del sistema operativo; los procesos de alta, baja y modificacion de datos se corresponden con todo lo visto hasta el momento.
No obstante, hay determinados problemas donde no es posible, no es deseable trabajar con registros de longitud fija.
Para evitar situaciones donde sobra memoria en cada elemento del archivo, es de interes contar con alguna organizacion que solo utilice el espacio necesario para almacenar la informacion. Este tipo de soluciones se representan con archivos donde los registros utilicen longitud variable. En estos casos, como el nombre lo indica, la cantidad de espacio utilizada por cada elemento del archivo no esta determinada a priori.
Cada elemento del dato debe descomponerse en cada uno de sus elementos constitutivos y asi, elemento a elemento, guardarse en el archivo. En el caso de necesitar transferir un string, debe hacerse caracter a caracter; en caso de tratarse de un dato numerico, cifra a cifra.

program ejemplo_4_5;

```
...
begin
...
while apellido<>'zzz' do
begin
  BlockWrite(empleados, apellido, length(apellido)+1);
  BlockWrite(empleados, '#', 1);
  ...
  BlockWrite(empleados, documento, length(documento)+1);
  BlockWrite(empleados, '@', 1);
end;
close(empleados);
end.
```

La primera conclusion que se puede obtener a partir del uso de registros de longitud variable es que la utilizacion de espacio en disco es optimizada, respecto del uso, con registros de longitud fija. Sin embargo, esto conlleva un algoritmo donde el programados debe resolver en forma mucho mas minuciosa las operaciones de agregar y quitar elementos.

4.3.1 Alternativas para registros de longitud variable

Cuando se plantea la utilizacion de espacio con longitud variable sobre tecnicas de espacio fijo, existen algunas variantes que se pueden analizar. Estas tienen que ver ocn utilizar indicadores de longitud de campo y/o registro. De esta manera, antes de almacenar un registro, se indica su longitud; luego, los siguientes bytes corresponen a elementos de datos de dicho registro.

4.4 Elminacion con registros de longitud variable

El proceso de baja sobre un archivo con registros de longitud variable es, a priori, similar a lo discutido anteriormente. Un elemento puede ser eliminado de manera logica o fisica. En este ultimo caso, el modo de recuperar espacio es similar a lo planteado en el ejemplo 4.3.
El prcoeso de baja logico no tiene diferencias sustanciales con respecto a lo discutido anteriormente. Sin embargo, cuando se desea recuperar el espacio borrado logicamente con nuevos elementos, deben tenerse en cuenta nuevas consideraciones. Estas tienen que ver con el espacio disponible. Mientras que con registros de longitud fija los elementos a eliminar e insertar son del mismo tamaño, utilizando registros de longitud variable esta precondicion no esta presente.
El proceso de insercion debe localizar el lugar dentro del archivo mas adecuado al nuevo elemento. Existen tres formas genericas para la seleccion de este espacio:

- **Primer ajuste:** consiste en seleccionar el primer espacio disponible donde quepa el registro a insertar.
- **Mejor ajuste:** consiste en seleccionar el espacio mas adecuado para el registro. Se considera el espacio mas 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.

4.5 Fragmentacion

- Se denomina **fragmentacion interna** a aquella que se produce cuando a un elemento de dato se le asigna mayor espacio del necesario.
- Se denomina **fragmentacion externa** al espacio disponible entre dos registros, pero que es demasiado pequeño para poder ser reutilizado.

4.5.1 Fragmentacion y recuperacion de espacio

Como se menciono anteriormente, el procedimiento de recuperacion de espacio generado por bajas, utilizando registro de longitud variable, presenta tres alternativas. Cada una de estas alternativas selecciona el espacio considerado mas conveniente. Las tecnicas de primer y mejor ajuste suelen implementar una variante que genera fragmentacion interna. Asi, una vez seleccionado el lugar libre, el espacio asignado corresponde a la totalidad de lo disponible.
Por el contrario, la tecnica de peor ajuste solo asigna el espacio necesario. De esta forma, es posible que genere fragmentacion externa dentro del archivo. Es deseable, en esos casos, dispone de un algoritmo que se ejecute periodicamente para la recuperacion de estos espacios no asignados *(garbage collector)*.

4.6 Modificacion de datos con registro de longitud variable

Modificar un registro existente puede significar que el nuevo regstro requiero el mismo espacio en disco, que ocuper menos espacio o que requiera uno de mayor tamaño. como es natural, el problema no se genera cuando ambos registros requieren el mismo espacio. Se puede suponer que si el nuevo elemento ocupa menos espacio que el anterior, no se genera una situacion problematica dado que el espacio disponible es suficiente, aunque en ese caso se generaria fragmentacion interna.
El probema surge cuando el nuevo registro ocupa mayor espaico que el anterior. En este caso, no es posible utilizar el mismo espacio fisico y el registro necesita ser reubicado.
En general, para evitar todo este analisis y para facilitar el algoritmo de modificacion sobre archivos con registros de longitud varibale, se estila dividir el proceso de modificacion en dos etapas: en la primera se de da baja al elemento de dato viejo, mientras que en la segunda etapa el nuevo registro es insertado de acuerdo con la politica de recuperacion de espacio determinada.