Fundamentos de Organización de Datos – Cursada 2025

Bibliografía

- Introducción a las Bases de Datos. Conceptos básicos (Bertone, Thomas).
- Estructuras de Archivos (Folk-Zoellick).
- Files & Databases: An introduction (Smith-Barnes).

1) Archivos

Persistencia de datos

Una **base de datos** es una colección de datos relacionados, específicamente de archivos diseñados para servir a múltiples aplicaciones. Estos datos representan hechos conocidos que pueden registrarse y que tienen un resultado implícito.

Propiedades implícitas de una BD

Una base de datos…

1. ...representa algunos aspectos del mundo real, a veces denominado Universo de Discurso.
2. ...es una colección coherente de datos con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD. O sea los datos deben tener cierta lógica.
3. ...se diseña, construye y completa de datos para un propósito específico. Está destinada a un grupo de usuarios concretos y tiene algunas aplicaciones preconcebidas en las cuales están interesados los usuarios.
4. ...está sustentada físicamente en archivos en dispositivos de almacenamiento persistente de datos.

Definiciones de archivo

Un archivo es una colección de...

1. ...registros guardados en almacenamiento secundario.
2. ...datos almacenados en dispositivos secundarios de memoria.
3. ...registros que abarcan entidades con un aspecto común y originadas para algún propósito particular.

Hardware

1. Almacenamiento primario: Memoria RAM

@author Federico Dobal

2. Almacenamiento secundario (DR): platos, superficies, pistas, sectores, cilindros.

Organización de un archivo

1. En una **secuencia de bytes:** archivos de texto dónde se leen o recuperan caracteres sin formato previo. Una palabra se determina por un conjunto de caracteres finalizados en blanco (convención).
2. En **registros y campos**: un campo es la unidad más pequeña lógicamente significativa de un archivo; un registro es un conjunto de campos agrupados que definen un elemento del archivo.

Acceso de archivos

1. Secuencial físico: acceso a los registros uno tras otro y en el orden físico en el que están guardados (sin orden específico, simplemente en como llegaron).
2. Secuencial indizado (lógico): acceso a los registros de acuerdo al orden establecido por otra estructura o criterio.
3. Directo: se accede a un registro determinado sin necesidad de haber accedido a los predecesores.

Tipos de archivos

Se determinan de acuerdo a su forma de acceso:

1. Serie: cada registro es accesible solo luego de procesar su antecesor, simples de acceder (acceso secuencial físico).
2. Secuencial: los registros son accesibles en orden de alguna clave (acceso secuencial indizado/secuencial lógico).
3. Directo: se accede al registro deseado (acceso directo).

Operaciones con archivos

Los **archivos físicos** aparecen en el disco y están a cargo del SO. En cambio, los **archivos lógicos** se definen dentro del programa. El archivo se puede definir de dos formas:

- Como variable:

  ```
  Var archivo: file of Tipo_de_dato;
  ```
- Como tipo:

  ```
  Type archivo: file of Tipo_de_dato;
  Var arch: archivo;
  ```

Relación con el SO: se debe asignar la correspondencia entre el nombre físico y el lógico:

```
Assign(n_logico,n_fisico);
```

`Rewrite`: de solo escritura (creación):

```
Rewrite(n_logico);
```

Reset: lectura/escritura (apertura del archivo):

```
Reset(n_logico);
```

Close: cierre de archivo. Luego del ultimo dato del archivo se coloca la marca EOF (End Of file), es decir, al final del archivo:

```
Close(n_logico);
```

Read: leer un archivo sobre una variable del mismo tipo del archivo:

```
Read(n_logico, variable);
```

Write: escribo en el buffer:

```
Write(n_logico, variable);
```

Estas operaciones (read y write) leen y/o escriben sobre los buffers relacionados a los archivos No se realizan directamente sobre la memoria secundaria.

Ejemplo 1: generar archivo

```
program Generar_Archivo;
type archivo = file of integer; {definición del tipo de dato para el
archivo }
var
    arc_logico: archivo; {variable que define el nombre lógico del archivo}
    nro: integer; {nro será utilizada para obtener la información de
    teclado}
    arc_fisico: string[12]; {utilizada para obtener el nombre físico del
archivo
    desde teclado}

begin
    write( 'Ingrese el nombre del archivo:' );
    read( arc_fisico ); { se obtiene el nombre del archivo}
    assign( arc_logico, arc_fisico );
    rewrite( arc_logico ); { se crea el archivo }
    read( nro ); { se obtiene de teclado el primer valor }
    while nro <> 0 do begin
        write( arc_logico, nro ); { se escribe en el archivo cada número }
        read( nro );
    end;
    close( arc_logico ); { se cierra el archivo abierto oportunamente con la
instrucción
    rewrite }
end.
```

EOF: fin del archivo:

```
EOF(n_logico); // true o false
```

FileSize: tamaño del archivo:

```
FileSize(n_logico); // tamaño del archivo
```

FilePos: posición dentro del archivo:

```
FilePos(n_logico); // posición (integer)
```

Seek: ir a una posición del archivo. Siempre se cuenta desde el comienzo del archivo (principio = 0):

```
Seek(n_logico, posicion); // procedimiento
```

Ejemplo 2: agregar datos a un archivo existente

```
procedure agregar (var emp: empleados);
var
    e:registro;
begin
    reset(emp);
    seek(emp, filesize(emp)); // se posiciona al final del archivo
    leer(e);
    while(e.nombre <> ' ') do begin
        write(emp,e);
        leer(e);
    end;
    close(emp);
end;
```

Archivos maestro y detalle

La relación entre estos archivos debe ser compatible para poder actualizar el maestro a partir del detalle. En general, hay **un archivo maestro por cada N archivos detalle**.

Precondiciones

1. Ambos “tipos” de archivos están ordenados por el mismo criterio.
2. En el archivo detalle solo aparece la información que existe en el archivo maestro.
3. Cada estructura de datos del archivo maestro solo puede aparecer a lo sumo una vez en su correspondiente archivo detalle.

Ejemplo 1: actualizar maestro a partir de detalle

```
program actualizar;
type
    emp = record
        nombre:string[30];
        direccion:string[30];
        cht:integer;
    end;
    e_diario = record
        nombre:string[30];
        cht:integer;
    end;
    detalle = file of e_diario;
    maestro = file of emp;
var
    regm:emp; regd:e_diario;
    mae1:maestro; det1:detalle;
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.nombre <> regd.nombre) do
            read(mae1,regm);
        regm.cht := regm.cht + regm.cht;
        seek(mae1, filepos(mae1)-1);
        write(mae1,regm);
    end;
end.
```

Ejemplo 2: actualizar maestro a partir de detalle con corte de control

Cada bloque del archivo maestro contiene a lo sumo una estructura de datos (un registro, por ejemplo) de sus correspondientes archivos detalles (**precondición 3**), por lo que no puede haber información repetida en el maestro. Para actualizar el maestro sin repetir información de los detalles se utilizarán cortes de control:

```
program actualizar;
const VALOR_ALTO=9999;
type
    str4 = string[4];
    prod = record
        cod:str4;
        descripcion:string[30];
        pu:real;
        cant:integer;
    end;
    v_prod = record
        cod:str4;
        ov:integer; {cantidad vendida}
    end;
    detalle = file of v_prod;
    maestro = file of prod:
var
    regm:prod; regd:v_prod;
    mael:maestro; det1:detalle;
    total:integer;
begin
    assign(mae1,'maestro');
    assign(det1,'detalle'); {proceso principal}
    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);
        while(not EOF(det1) and (regm.cod = regd.cod)) do begin
            regm.cant := regm.cant - regd.cv:
            read(det1,regd); // (!)
```

```
            end;
            if(not EOF(det1))
                seek(det1, filepos(det1)-1);
            seek(mae1, filepos(mae1)-1);
            write(mae1,regm):
        end;
    end.
```

(!) Nótese que si estoy ante el ultimo registro del archivo, al evaluar aquel **`while`** el **`read`** ya avanzó y me perdería este último registro. Para abstraerse de esta situación, podemos crear un procedimiento **`leer(archivoX,registroX)`**:

```
procedure leer(var archivo:detalle; var dato:v_prod);
begin
    if(not EOF(archivo)) then
        read(archivo,dato)
    else
        dato.cod := VALOR_ALTO;
end;
```

Y en el programa principal suprimimos **`not (EOF(det1):`**

```
...
while(regm.cod = regd.cod)) do begin
            regm.cant := regm.cant - regd.cv:
            leer(det1,regd); // (OK!)
end;
...
```

Múltiples archivos detalle

Las precondiciones y la declaración de tipos son los mismos. El problema es que al tener N archivos detalles (cada uno ordenado), se hace más complejo encontrar el primer elemento entre todos (el menor) para actualizar su respectivo en el archivo maestro. Por ejemplo, no puedo actualizar un elemento y luego uno menor que este mismo. Para ello, se deben recorrer los N archivos detalle y encontrar el elemento mínimo para arrancar a actualizar su archivo maestro a partir de él.

```
begin // Programa principal
    assign(mae1, 'maestro'); assign(det1, 'detalle1');
    assign(det2, 'detalle2'); assign(det3, 'detalle3');
    reset(mae1); reset(det1); reset(det2); reset(det3);
    leer(det1, regd1); leer(det2, regd2); leer(det3, regd3);
    minimo(regd1, regd2, regd3, min);
    while (min.cod <> valoralto) do begin
        read(mae1,regm);
        while (regm.cod <> min.cod) do
            read(mae1,regm);
        while (regm.cod = min.cod ) do begin
            regm.cant:=regm.cant - min.cantvendida;
            minimo(regd1, regd2, regd3, min);
        end;
        seek (mae1, filepos(mae1)-1);
        write(mae1,regm);
    end;
end.
```