

# CONCEPTOS DE BASES DE DATOS

## CLASE 3



- **Organización de un archivo**
  - **Características de estructura**
    - **Secuencia de bytes**
      - Unidad de lectura/escritura → **byte**
      - Difícil determinar el comienzo y el final de cada dato.
    - **Campos**
      - Unidad de lectura/escritura → **campo**
      - Permite separar cada dato
    - **Registros**
      - Unidad de lectura/escritura → **registro**
      - Es un conjunto de campos agrupados que **definen un elemento del archivo**

- **Organización de un archivo**
  - **Archivos con registros de longitud fija**
    - Ej: `persona = record`  
    `nyap: string[50];`  
    `dir: string[50];`  
    `dni: string[12];`  
    `edad: integer;`  
    `obs: string[200];`  
    `end`
    - Tamaño de registro = **314 bytes**
    - Tamaño promedio de registro: **124 bytes**
      - Desperdicio de espacio
      - Mayor tiempo de procesamiento

- **Campos**

- **Longitud predecible**

- Long. Fija
  - Relleno de campo: desperdicio de espacio
  - Si no entra un dato, se agranda el tamaño fijo → más desperdicio

- **Indicador de longitud**

- Valor al principio de cada campo

- **Delimitador**

- Se usa una marca al final de cada campo, que es un carácter especial no utilizado como dato

- Ej: lectura y presentación de un archivo con delimitadores para los campos

```
PROGRAM leerArchivoCampos
CONST
    delimitador = '|'
BEGIN
    ABRIR archivo (como lectura/escritura)
    nroCampos = 0
    longCampo = leerCampo(archivo, contenidoCampo)
    MIENTRAS (longCampo > 0)
        INC nroCampos
        ESCRIBIR_PANTALLA nroCampos
        ESCRIBIR_PANTALLA contenidoCampo
        longCampo = leerCampo(archivo, contenidoCampo)
    FIN MIENTRAS
    CERRAR archivo
END
```

- **Ej: lectura y presentación de un archivo con delimitadores para los campos**

```
FUNCTION leerCampo(var archivo, var contenidoCampo)
BEGIN
    long = 0
    car = ""
    MIENTRAS(NOT EOF(archivo) & car <> delimitador )
        LEER_UN_CARACTER(archivo, car)
        SI (car <> delimitador )
            contenidoCampo[long] = car
            INC long
    FIN MIENTRAS
    DEVOLVER(long)
END
```



- **Registros**

- **Longitud predecible**

- Long. Fija → relleno de c/ campo: desperdicio de espacio
- Cantidad de bytes → nro fijo de bytes
- Cantidad de campos → nro fijo de campos

- **Indicador de longitud**

- Se indica al inicio de cada registro → registros de longitud variable

- **Delimitador**

- Se usa una marca al final de cada registro (carácter especial no utilizado como dato y diferente de otros delimitadores)

- **Segundo archivo**

- Mantiene la dirección del byte de inicio de cada registro

- Ej: archivo con indicador de longitud (registros) y delimitadores (campos)

**PROGRAM** leerArchivoRegistros

**BEGIN**

ABRIR archivo

**tomarReg**(archivo, buffer, longRegistro) *{traslada un registro a un buffer}*

posBuscada = 0

**MIENTRAS** (longRegistro > 0)

**tomarCampo**(buffer, longRegistro, posBuscada, campo) *{lee un campo}*

**MIENTRAS** (posBuscada > 0 )

**ESCRIBIR\_PANTALLA** campo

**tomarCampo**(buffer, longRegistro, posBuscada, campo)

**FIN MIENTRAS**

**tomarReg**(archivo, buffer, longRegistro)

**FIN MIENTRAS**

CERRAR archivo

**END**



# Archivos

## Organización

- Ej: archivo con indicador de longitud (registros) y delimitadores (campos)

PROCEDURE tomarReg (*var* archivo, *var* buffer, *var* long\_reg)

BEGIN

SI EOF(archivo) ENTONCES

*long\_reg* = 0

SINO

LEER *long\_reg* {nro entero que indica la cantidad de bytes del registro}

LEER *long\_reg* bytes del archivo en *buffer*

END

# Archivos

## Organización

- Ej: archivo con indicador de longitud (registros) y delimitadores (campos)

PROCEDURE tomarCampo(buffer, long\_reg, var pos\_bus, var campo)

BEGIN

SI (*pos\_bus* == *long\_reg*) ENTONCES

*pos\_bus* = 0

SINO

LEER byte en la *pos\_bus* de *buffer*

MIENTRAS(*pos\_bus* < *long\_reg* & byte <> delimitador)

COLOCAR byte en *campo*

INC(*pos\_bus*)

LEER byte en la *pos\_bus* de *buffer*

FIN MIENTRAS

END

- **Pascal**

- En Pascal es posible aplicar este tipo de organizaciones definiendo una variable como **archivo sin tipo** (file)
- Esta elección de archivo permite realizar la transferencia **carácter a carácter** (byte a byte)
- De esta forma es posible el uso de archivos con **registros de longitud variable**

- **Pascal**

- Registros de longitud variable:
  - Espacio de almacenamiento → optimizado
  - Lectura/escritura registros → implementación ad-hoc
- Una organización posible:
  - Secuencia de caracteres (archivo sin tipo)
  - Delimitadores de fin de campo y fin de registro
  - Marca EOF
  - Ejemplos.

- **Ej: creación de un archivo de empleados**

```
program ejemplo_4_5;
Var
  empleados: file; {archivo sin tipo}
  nombre,apellido,direccion,documento: string;

Begin
  Assign(empleados,'empleados.txt');
  Rewrite(empleados,1);
  writeln('Ingrese el Apellido');
  readln(apellido);
  writeln('Ingrese el Nombre');
  readln(nombre);
  writeln('Ingrese direccion');
  readln(direccion);
  writeln('Ingrese el documento');
  readln(documento);
```

# Archivos

## Organización

```
while apellido<>'zzz' do
  Begin
    BlockWrite(empleados,apellido,length(apellido)+1);
    BlockWrite(empleados,'#',1);
    BlockWrite(empleados,nombre,length(nombre)+1);
    BlockWrite(empleados,'#',1);
    BlockWrite(empleados,direccion,length(direccion)+1);
    BlockWrite(empleados,'#',1);
    BlockWrite(empleados,documento,length(documento)+1);
    BlockWrite(empleados,'@',1);
    writeln('Ingrese el Apellido');
    readln(apellido);
    writeln('Ingrese el Nombre');
    readln(nombre);
    writeln('Ingrese direccion');
    readln(direccion);
    writeln('Ingrese el documento');
    readln(documento);
  end;
  close(empleados);
end.
```



- **Ej: visualización del archivo de empleados**

```
program ejemplo_4_6;  
Var  
    empleados: file; {archivo sin tipo}  
    campo, buffer :string;  
Begin  
    Assign(empleados,'empleados.txt');  
    reset(empleados,1);  
    while not eof(empleados) do  
        Begin  
            BlockRead(empleados,buffer,1);
```

# Archivos

## Organización

```
while (buffer<>'@') and not eof(empleados)do
begin
  while ((buffer<>'@') and
    (buffer<>'#') and
    not eof(empleados))do
  begin
    campo := campo + buffer ;
    BlockRead(empleados,buffer,1);
  end;
  writeln(campo);
end;

if not eof(empleados) then
  BlockRead(empleados,buffer,1);
end;
close(empleados);
end.
```

- **Clave o llave**

- El **objetivo** de una clave es **facilitar el acceso** a un registro en particular dentro del archivo
  - Se concibe al registro como la **unidad de L/E**
  - Útil para casos en los que no se debe acceder al archivo completo
  - Se usa la clave para **identificar** al registro
  - La clave debe estar **basada en el contenido** del registro
- **Identificación**
  - **Primaria** → identifica de forma **unívoca** a un registro
  - **Secundaria** → puede identificar a uno o más registros

- **Clave o llave**

- **Forma canónica:** forma estándar para una clave
  - Puede derivarse a partir de reglas bien definidas
  - Representación única para la clave, ajustada a la regla
  - Ej: usar letras mayúsculas y sin espacios
  - **Al agregar un registro** → se debe comprobar que no hay otro registro con la misma clave primaria:
    - Si no existe, entonces se inserta el registro
    - Si ya existe, se debe comprobar/modificar los datos del nuevo registro

# Archivos

## Acceso a los datos

- Ej: búsqueda secuencial de un empleado usando su clave

**PROGRAM BuscarEmpleado**

**BEGIN**

**ABRIR** archivo

**LEER** apellidoBuscado

**LEER** nombreBuscado

*{A partir de los datos leídos desde teclado → construyo la clave en Forma Canónica}*

claveBuscada = **CONSTRUIR\_FC** (apellidoBuscado, nombreBuscado)

encontro = false

**tomarRegistro**(archivo, buffer, longRegistro)

# Archivos

## Acceso a los datos

- **Ej: búsqueda secuencial de un empleado usando su clave**

**MIENTRAS** (NOT encontro AND longRegistro > 0 )

posBuscada = 0

**tomarCampo**(buffer, longRegistro, posBuscada, apellido)

**tomarCampo**(buffer, longRegistro, posBuscada, nombre)

*{A partir de los datos leídos desde archivo → construyo la clave en Forma Canónica}*

claveRegistro = **CONSTRUIR\_FC** (apellido, nombre)

**SI** (claveRegistro == claveBuscada) **ENTONCES**

encontro = true

**SINO** **tomarRegistro**(archivo, buffer, longRegistro)

**FIN MIENTRAS**

**SI** (encontro) **ENTONCES** **MOSTRAR\_REGISTRO**();

**CERRAR** archivo

**END**



- **Acceso directo**

- Modo de acceso que permite **saltar directamente** hasta el lugar de un registro en particular
  - El acceso directo es **preferible** sólo cuando se necesitan pocos registros específicos
  - Es necesario **conocer el lugar de comienzo** del registro requerido
  - Posible con reg. long fija → **registro encabezado**
    - Mantiene información general del archivo como la cantidad de registros, el tamaño del registro, tipos de delimitadores, etc.

- **Acceso directo**

- **NRR** (nro relativo de registro): indica la posición de un registro con respecto al inicio del archivo
  - En archivos con reg. de **longitud fija**
    - Utilizando el NRR se calcula la distancia en bytes para acceder al registro buscado
    - Ej: **NRR = 546**, **tamaño reg. = 128 bytes**.  
Inicio del reg. =  $546 * 128 = 69.888$
  - En archivos con reg. de **longitud variable**
    - No se conoce el tamaño de cada registro → no es posible aplicar el NRR → **lectura secuencial**

# Archivos

## Mantenimiento

- Los procesos de algorítmica clásica que se examinaron utilizan archivos con registros de **longitud fija**
  - Agregar registros
  - Modificar registros
- Aún resta analizar:
  - Esos procesos en archivos con registros de **longitud variable**
  - El proceso de **eliminación de registros**

- Tipos de archivos (según sus cambios)
  - **Estáticos** → pocos cambios
    - Puede actualizarse en procesamiento por lotes
    - No necesita de estructuras adicionales para agilizar los cambios
  - **Volátiles** → operaciones frecuentes
    - Agregar / Borrar / Actualizar
    - Su organización debe facilitar cambios rápidos
    - Necesita estructuras adicionales para mejorar los tiempos de acceso

- Inserción de un registro
  - Registro de **longitud fija** → sin problemas
  - Registro de **longitud variable** → sin problemas
- Modificación de un registro
  - Registro de **longitud fija**
    - Dado que el tamaño del registro es siempre el mismo → sin problemas
  - Registro de **longitud variable**
    - Si ahora el registro es menor → sobra lugar
    - Si ahora el registro es mayor → no cabe

- Modificación de un registro
  - Registro de **longitud variable**, donde el registro actualizado tiene un tamaño mayor:
    - Agregar los datos adicionales al final del archivo (con un vínculo al registro original) → **complica el procesamiento del registro**
    - Eliminar e insertar el registro actualizado al final del archivo → **más simple** pero queda un espacio vacío (**desperdiciado**) en el lugar origen
  - Nos centraremos en los métodos de **eliminación de registros** y de **recuperación de espacio**



# Archivos

## Eliminación

- Eliminación de un registro
  - **Baja física**
    - El registro eliminado deja de estar físicamente en el archivo
  - **Baja lógica**
    - El registro eliminado sigue estando en el archivo, pero marcado como eliminado

- **Baja Física** → Compactación
  - Forma más simple: copiar todo a excepción de los registros eliminados
  - Frecuencia (depende del dominio de aplicación)
    - Cantidad de registros eliminados
    - Periodo de tiempo determinado
    - Ante la necesidad de espacio
  - Archivos muy volátiles → **no es conveniente**
    - Análisis de **recuperación dinámica del espacio** de almacenamiento

- Baja Lógica

- Se coloca una **marca especial** en los registros eliminados para hacer posible el posterior reconocimiento de los mismos
- Permite **anular** el proceso de eliminación fácilmente → **costo de espacio**
- Los programas que usan archivos deben cambiar su metodología de trabajo:
  - **LECTURA**: se debe ignorar registros eliminados
  - **INSERCIÓN**: se debe reutilizar el espacio de registros eliminados

- Recuperación dinámica de espacio
  - La recuperación dinámica de espacio (reasignación de espacio) consiste en **reutilizar** el espacio **al momento de insertar un nuevo registro** en el archivo
  - Usando las marcas de borrado lógico se puede identificar un **espacio vacío** en el que se pueda **almacenar el nuevo registro**

# Archivos

## Recuperación de espacio

- En archivos con registros de **longitud fija**
  - **Búsqueda secuencial**
    - Al insertar un nuevo registro se busca el 1º registro eliminado (marcado)
    - Si no existe, se llega al final del archivo y se agrega allí
    - Puede ser **muy lento** en algunos casos de aplicación
  - Es necesario:
    - Una forma de **saber de inmediato** si hay lugares vacíos en el archivo
    - Una forma de **saltar directamente** a unos de esos lugares, en caso de existir

# Archivos

## Recuperación de espacio

- En archivos con registros de **longitud fija**
  - **Lista encadenada o Pila**
    - Se mantienen enlazados los espacios libres
    - Al insertar un nuevo registro → cualquier espacio libre es bueno
    - La lista no necesita estar ordenada → pila
      - Mínima reorganización al agregar o sacar elementos



- En archivos con registros de **longitud fija**
  - **Lista encadenada o Pila**
    - Se utiliza el **NRR** como dirección de enlace entre nodos
    - El **NRR** que indica el primer registro disponible está en el **registro encabezado** del archivo
    - En el caso de que no haya registros disponibles para ser reutilizados, se guarda el número **-1** en el registro encabezado

# Archivos

## Recuperación de espacio

- Ejemplo de lista encadenada

Borrado NINGUNO	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
--------------------	----	----	----	----	----	----	----	----	----	----

↑  
Registro Cabecera

Archivo Original

Borrado Posición 2	R0	R1	Borrado	R3	R4	R5	R6	R7	R8	R9
-----------------------	----	----	---------	----	----	----	----	----	----	----

↑  
Registro Cabecera

Luego de borrar R2

Borrado Posición 5	R0	R1	Borrado	R3	R4	Próximo 2	R6	R7	R8	R9
-----------------------	----	----	---------	----	----	--------------	----	----	----	----

↑  
Registro Cabecera

Luego de borrar R5

# Archivos

## Recuperación de espacio

- Ejemplo de lista encadenada



↑  
Registro Cabecera

Luego de borrar R8



↑  
Registro Cabecera

Luego de insertar R10

# Archivos

## Recuperación de espacio

- En archivos con registros de **longitud variable**
  - También se utiliza **la marca de borrado** para identificar a los registros eliminados
  - El problema ahora es que el nuevo registro no se puede colocar en cualquier lugar → **debe caber**
  - Se necesita realizar una búsqueda en la lista de lugares disponibles → **no se organiza como una pila**

- En archivos con registros de **longitud variable**
  - **Lista**
    - No se puede usar **NRR** como enlace → se utiliza un **campo binario** que explícitamente indica el enlace
    - Cada registro indica en su inicio la **cantidad de bytes** que ocupa
  - Se busca un lugar disponible lo **suficientemente grande** para que entre el nuevo registro
    - Si el lugar encontrado no es exactamente del mismo tamaño → se origina **FRAGMENTACION**

# Archivos

## Recuperación de espacio

- Fragmentación **interna**
  - Ocurre cuando se desperdicia espacio **dentro de un registro** → el lugar está asignado al registro pero éste no lo ocupa totalmente
  - Puede darse en archivos con **reg. de longitud fija**
    - Ej: no todos los nombres de personas ocupan la misma cantidad de caracteres
  - Puede darse en archivos con **reg. de longitud variable**
    - Al escribir por 1ra vez el archivo → NO
    - Al borrar un reg. y reemplazarlo por otro más corto → **SI**
    - El espacio sobrante puede pasar a ser un **nuevo espacio libre**, pero esto puede originar **FRAGMENTACIÓN EXTERNA**



- Fragmentación **externa**
  - Ocurre cuando el espacio libre es demasiado pequeño como para ser ocupado por un nuevo registro
  - Soluciones
    - **Unir espacios libres pequeños adyacentes** para generar un espacio disponible mayor
    - **Regenerar el archivo** → cuando hay mucha fragmentación
    - **Minimizar la fragmentación**, eligiendo el espacio más “**adecuado**” en cada caso → **Estrategias de colocación**:
      - Primer ajuste
      - Mejor ajuste
      - Peor ajuste

- Estrategias de colocación
  - **Primer ajuste**
    - Se selecciona la primera entrada de la lista de disponibles que pueda almacenar al registro, y **se le asigna de forma completa** al mismo
    - Minimiza la búsqueda
    - No se preocupa por la exactitud del ajuste

# Archivos

## Recuperación de espacio

- Estrategias de colocación
  - **Mejor ajuste**
    - Elige la entrada que más se aproxime al tamaño del registro y **se le asigna de forma completa** al mismo.
    - Exige una búsqueda completa

- Estrategias de colocación
  - **Peor ajuste**
    - Elige la entrada más grande para el registro pero **sólo le asigna el espacio necesario** al mismo, quedando libre el resto.
    - El sobrante puede ser usado entonces por otro registro
    - Exige una búsqueda completa

- Estrategias de colocación
  - **Conclusiones**
    - Las estrategias de colocación sólo tienen sentido en archivos con registros de longitud variable
    - De acuerdo a lo visto, en resumen:
      - **Primer ajuste:** más rápido, generalmente genera fragmentación interna
      - **Mejor ajuste:** generalmente genera fragmentación interna
      - **Peor ajuste:** potencialmente genera fragmentación externa