Capítulo 10

ARCHIVOS

1. INTRODUCCIÓN

En los capítulos anteriores se han usado datos almacenados en diferentes estructuras para ser procesados en los distintos programas de aplicación. Estas estructuras requieren el almacenamiento de los datos internamente en memoria, lo cual implica dos limitaciones muy importantes. Por un lado, en la mayoría de situaciones de la vida real se generan grandes cantidades de datos, sin embargo, la cantidad de datos que hasta ahora ha podido tratarse con arrays es bastante reducida, habida cuenta de la limitación de la memoria central del ordenador y la tediosa tarea de introducir los datos en tiempo de ejecución. En segundo lugar, el tiempo de vida de estos datos está condicionado al tiempo de ejecución del programa, de manera que desaparecen cuando este termina, a no ser que se hayan inicializado los arrays dentro de los programas. Por estas razones, las aplicaciones que pueden realizarse con estas estructuras de datos suelen ser triviales.

Para dar respuesta a estos inconvenientes se implementan nuevas estructuras de datos, que permiten tratar grandes cantidades y almacenarlos de forma permanente, para posibles futuros usos, en dispositivos de almacenamiento no volátiles, discos duros, disquetes, cintas, CD's, etc. Estas estructuras de datos se conocen como **archivos** o **ficheros**.

Los archivos son sumamente importantes, pues permiten almacenar diferente tipo de información, por ejemplo, programas, como los realizados hasta ahora en lenguaje Java, gráficos, textos, datos, etc. En este capítulo se estudiarán en particular los archivos que sirven para almacenar los datos que deben ser procesados por un programa.

2. GENERALIDADES

2.1. Definición y características

Un *archivo* o *fichero* es una estructura de almacenamiento externo de datos, es decir, los datos residen en memoria secundaria y no en memoria principal. Esta forma de almacenamiento tiene como consecuencia sus dos características principales: persistencia de los datos e imposibilidad de tratarlos directamente. Para trabajar con los datos de un archivo, es necesario cargarlos previamente en la memoria principal.

El archivo está constituido por un conjunto de elementos todos del mismo tipo, organizados en unidades de acceso, llamadas **registros**. Los registros tienen los formatos preestablecidos y su número es finito e indeterminado, lo que significa que no se conoce de antemano cuantos son, pero su número es limitado. Si se desea saber la cantidad de registros de un fichero, habrá que contarlos.

Los archivos tienen las siguientes características:

- **Residen en soporte de almacenamiento externo**. Esto posibilita que la información pueda ser transportada de un lugar a otro.

- **La información se almacena de forma permanente**. Los datos almacenados permanecen en el tiempo, mientras no sean borrados explícitamente por un programa o una instrucción del sistema operativo.

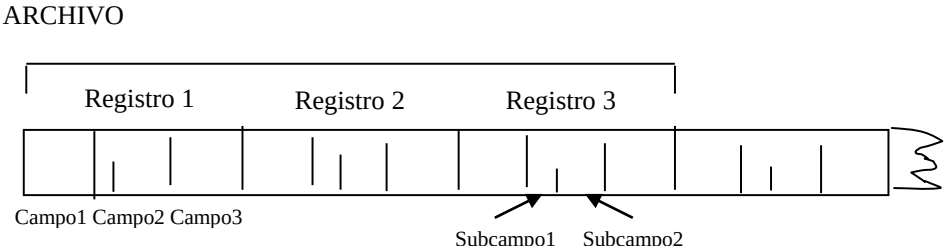
- **Establecen la independencia de la información que guardan respecto a los programas que los usan**. Diferentes programas pueden usar el mismo archivo.

- **Tienen una alta capacidad de almacenamiento de datos**. El límite ni siquiera lo impondrá el soporte físico en el que resida el archivo, puesto que pueden existir archivos que ocupen varias unidades de soporte externo. Un archivo almacenado en estas condiciones se llama **archivo multivolumen**.

2.2. Concepto de Registro

Cuando se almacenan datos en un fichero aparece el concepto de registro. Un registro es una agrupación de datos. Los registros pueden estar formados por elementos simples o complejos, pero, en cualquier caso, la estructura de todos los registros de un archivo es única y predeterminada.

Se define *registro* como cada uno de los componentes de un archivo, que posee una cierta estructura predefinida, común para todos ellos y almacena información relativa a un objeto o tema concreto. Por ejemplo, se desea guardar los datos de los alumnos de un curso en un fichero (número de matrícula, nombre, apellidos, curso, asignaturas y notas), por tanto, se necesita que esté formado por registros cuya estructura permita almacenar los mismos datos referentes a cada uno de los alumnos.



2.3. Contenido de los registros

Cuando los registros están formados por elementos complejos se dice que está dividido en **campos**. Registros y campos obedecen a tipos de datos preestablecidos, que, habitualmente en aplicaciones de gestión, coinciden con los tipos de datos estructurados. Los campos a su vez pueden también estar divididos en **subcampos.**

Para distinguir un registro de otro dentro de un mismo fichero, se usa información que sea única para cada uno de los registros y que por tanto, lo identifique de forma exclusiva. Esta información la proporciona uno o varios de los campos del registro y se le conoce como el **campo clave**. Se define formalmente como aquél, o aquellos, que permiten identificar de forma unívoca a cada registro del archivo. Para poder ser clave, el contenido del campo no sólo ha de ser único para cada registro, sino que, además, no debe permitir valores nulos.

La clave puede ser creada con independencia del resto de la información del registro, o puede utilizarse uno o varios campos existentes en la estructura de dicho registro, siendo siempre el objetivo el de facilitar a través de ella las operaciones de búsqueda y clasificación.

Ejemplo:

El archivo *alumnos.dat* tiene una cantidad determinada de registros donde se guarda cierta información referente a los alumnos de un curso. Cada registro puede estar formado por los siguientes campos: *apellido1*, *apellido2*, *nombre*, *DNI*, *notas*. El campo *notas* contiene los subcampos: *programación*, *análisis*, *sistemas operativos y bases de datos*.

El campo *DNI*, por ejemplo, puede utilizarse como campo clave, puesto que es personal e irrepetible, y, por tanto, puede ser utilizado como identificador del registro al que pertenece.

Cuando se programa con metodología orientada a objetos, lo habitual será al se almacenen objetos en un fichero, por lo que los registros serán objetos y los campos atributos de estos objetos, que por supuesto podrán ser objetos de otra clase que también tendrán sus atributos.

2.4. Longitud de los registros

Los registros pueden tener todos sus campos de longitud fija, en cuyo caso todos los registros también serán de longitud fija. Pero, también puede ocurrir que la longitud de los campos sea variable y, por tanto, los registros también tendrán longitud variable, o sea, cada registro puede tener una longitud distinta. Por ejemplo, los ficheros que almacenan datos en modo texto, *ficheros de texto* en Java o C.

No todos los compiladores permiten ficheros con registros de longitud variable.

2.5. Nombre externo y nombre interno de un fichero

Se entiende que **nombre externo** de un archivo es aquél con el que el sistema operativo reconoce a dicho archivo, mientras que **nombre interno** es una variable, declarada en el programa, con la que el programa que lo usa referencia al fichero. En algunos compiladores, ambos nombres coinciden, siendo este el nombre externo.

3. TIPOS DE FICHEROS

Se pueden realizar diferentes clasificaciones de ficheros atendiendo a criterios también distintos. A continuación, se señalan las más comunes desde el punto de vista del programador.

3.1. Según su función

Según el objetivo al que van destinado o uso que se hace de ellos, los ficheros se clasifican en:

PERMANENTES

Son aquellos ficheros cuya información no varía o varía muy poco a lo largo del tiempo. Existen varios tipos:

- **De situación**. Los datos que contienen revelan el estado actual de dicha información. Estos ficheros se acceden con mucha frecuencia. Suelen llamarse también **ficheros maestros**. Por ejemplo, fichero de inventario con las existencias de un almacén.

- **De constantes**: Guardan información para consultar. Los registros tienen una frecuencia de modificación nula o muy baja. Por ejemplo, un fichero con los códigos postales de cada localidad española.

- **Históricos:** Almacenan información referente a otro tiempo, por lo que no se modifican nunca. Generalmente, esta información sirve para elaborar estadísticas. Por ejemplo, fichero con datos significativos de los países de la

Unión Europea que firmaron el Tratado de Roma; fichero con datos sobre la población española por provincias en el último cuarto del siglo pasado.

DE MOVIMIENTOS

Estos ficheros contienen datos producidos en ciertos procesos que servirán para actualizar un fichero permanente. Su tiempo de vida es corto y, una vez finalizado el proceso de actualización, desaparecen. Por ejemplo, fichero con las compras y ventas del día de una empresa.

DE TRABAJO

Son creados por el sistema operativo, por los compiladores o por cualquier otra utilidad, para almacenar transitoriamente datos intermedios. A veces, estos ficheros desaparecen cuando acaba la aplicación que los genera y en otras ocasiones no. Por ejemplo, los archivos temporales de Windows o de Microsoft Office.

3.2. Según su organización

Organizar un archivo significa definir una disposición física concreta de los registros de ese fichero, es decir, aplicar ciertas reglas para determinar la colocación de los registros sobre el dispositivo de almacenamiento. El tipo de organización se establece durante la fase de creación. Las formas de organizar un fichero son: secuencial, directa o aleatoria y secuencial indexada. Solo se estudiarán con detalle los secuenciales en sus dos formas de acceso.

4. MODOS DE ACCESOS

A los registros de un fichero se puede acceder de dos maneras.

4.1. Acceso secuencial

Se accede secuencialmente a un fichero cuando para obtener la información contenida en un registro deben haberse accedido todos los registros que le preceden. Este modo puede ser impuesto por la propia organización del fichero que, a su vez, puede estar determinada por el dispositivo de almacenamiento. Por ejemplo, un archivo cuyos datos están almacenados en cinta magnética, no tiene más remedio que organizarse de manera secuencial y accederse de igual forma.

4.2. Acceso directo

Este modo de acceso permite abordar un registro directamente, sin necesidad de acceder antes a todos los registros que le preceden. Sólo se permite este modo de acceso en soportes direccionables, disquetes, discos duros, etc. El acceso puede

realizarse bien por el número de registro o por la clave de acceso (sólo si se organización es directa).

5. OPERACIONES SOBRE FICHEROS

Todas las operaciones que se realizan sobre los archivos tienen como objetivo actuar sobre sus registros, sin embargo, algunas de ellas atañen al fichero completo. Generalmente, las operaciones que se realizan sobre ficheros son las que se detallan a continuación.

5.1. Creación

Para poder realizar cualquier operación sobre un fichero, es imprescindible que haya sido creado con anterioridad. La creación consiste, generalmente, además de en darle nombre, en escribir los registros que inicialmente forman el fichero. Los datos pueden venir del teclado, desde otro fichero o como resultado de algún proceso intermedio.

Siempre que los datos vengan de la entrada estándar se aconseja realizar un programa para creación del fichero. Este programa será absolutamente independiente del resto de programas de aplicación generados para usarlos.

Normalmente, se requiere que, al crear el fichero, se establezca la forma en que se almacena la información, es decir, si será en modo texto o binario, así como el tipo de organización y acceso de los datos, esto último no ocurrirá en C.

5.2. Recorrido

El recorrido es una operación sobre todos los elementos de un fichero. Supone la lectura secuencial de todos los registros, con el objetivo de realizar sobre ellos algún proceso en particular, por ejemplo, sacar un listado por impresora.

5.3. Clasificación u ordenación

Esta operación permite ordenar un archivo en función de ciertas especificaciones dadas y en consonancia con alguna característica de algún o algunos de los campos de la estructura de registro.

5.4. Actualización

Es un proceso sobre el fichero para operaciones de altas, bajas, o modificaciones de algún o algunos de los registros de dicho fichero. Esta operación atañe a todo el fichero, y afecta a todos o parte de los registros. Se realiza,

habitualmente, mediante la sincronización de ficheros que permite tener actualizado el fichero original.

5.5. Partición

El proceso de partición permite descomponer un fichero en varios atendiendo a alguna característica de algún o algunos de sus campos, de manera que los registros quedan repartidos entre los distintos ficheros finales.

5.6. Fusión o mezcla

Consiste en obtener a partir de dos o más ficheros de idéntica estructura, un nuevo fichero que contenga los registros de los anteriores. La mezcla puede ser sobre archivos desordenados o sobre archivos ordenados, por lo que los algoritmos a aplicar en cada caso deberán tener en cuenta esta característica. La operación de fusión no afecta en absoluto a los ficheros originales que intervienen en ella.

5.7. Borrado

Es la operación inversa a la creación y supone la eliminación física del fichero sobre el soporte, dejando libre el espacio de almacenamiento en memoria secundaria que ocupaba. Como consecuencia, se pierde toda posibilidad de acceder a la información que almacenaba previamente.

5.8. Consideraciones para la utilización de archivos

Una vez creado un archivo para usarlo deben seguirse los siguientes pasos:

1. Realizar la apertura adecuadamente. El programa deberá controlar la existencia del fichero y prever el posible tratamiento en caso de error.

   La apertura de un fichero conlleva las siguientes características:

   - Se establece el vínculo entre el nombre interno y el externo del fichero.

   - Se reserva el fichero en exclusividad para uso del programa que ejecuta la apertura. Esta operación se denomina **bloqueo**.

   - Se crea un buffer de memoria para uso exclusivo del fichero. Este buffer hace de memoria intermedia entre el fichero, la memoria central del ordenador y el programa que lo va a usar. Al buffer se pasa, entre otros elementos, el contenido de un registro físico con cada acceso al fichero.

- El puntero del archivo se coloca en el primer registro e irá señalando, automáticamente, a los siguientes a medida que se procesa cada registro, conociéndose como **registro activo** el apuntado en un momento en particular. El proceso termina cuando el puntero alcanza el final de fichero.

- Hay que indicar si el fichero está abierto para leer, escribir o ambas cosas. Esto dependerá del lenguaje usado para la implementación, por ejemplo en Java sólo pueden abrirse para leer o para escribir, pero no para ambas cosas.

2. Proceso.

   Se realizarán las tareas especificadas sobre el fichero.

3. Cierre.

   Terminado el proceso de un archivo, éste deberá ser cerrado, lo que supone:

   - Se desvincula el nombre interno del externo.

   - Queda libre el fichero para posteriores usos.

   - Se coloca una marca especial de “fin de fichero”, que será detectada en posteriores lecturas y que será utilizada en el acceso para controlar, precisamente, que se han procesado todos los registros del fichero. Esto también dependerá del lenguaje de programación.

   - Vacía y elimina el buffer adjuntado en la apertura.

6. OPERACIONES SOBRE REGISTROS

Las operaciones sobre los registros de un fichero son las que llevan a cabo el procesamiento de la información y, por tanto, las que tienen mayor importancia para el sistema que se está desarrollando.

6.1. Consulta

Mediante esta operación, uno o varios registros son localizados mediante clave o como consecuencia de haber accedido a todos los registros que le preceden. El objetivo es visualizar el contenido de los campos de los registros accedidos. Generalmente, la información se presenta en pantalla o se saca por impresora.

6.2. Modificación

Consiste en realizar el cambio del contenido de todo o parte de los campos del registro, excepto del campo clave si lo hubiera, ya que si se modifica éste se estaría creando un registro nuevo. Para modificar un registro es necesario una búsqueda y lectura previa y un proceso de escritura para actualizar lo deseado.

6.3. Supresión

Un registro se puede suprimir después de una búsqueda y lectura previa de dos modos distintos:

- **Supresión por marca**. Consiste en la modificación del valor de uno de sus campos, siguiendo un protocolo determinado anteriormente, de manera que los programas que usen esos archivos, conocen dicho protocolo y saben que el contenido de los registros marcados no tiene información significativa. Por ejemplo, se puede suprimir marcando con asterisco (*) el campo nombre de un registro. Esto supone que ese registro no es significativo, es decir, no existe como dato válido para los programas que deban usar el fichero. De esta forma, al procesar dicho fichero no se procesarán aquellos registros cuyo campo nombre contenga un *.

- **Supresión real**. Consiste en hacer que el registro sea inaccesible o en eliminar físicamente dicho registro por ocupación de su espacio con un registro distinto. La supresión real del registro liberará el espacio que ocupa en el dispositivo donde se encuentra almacenado.

6.4. Inserción de un registro

Consiste en añadir un nuevo registro al fichero. Esta operación sólo es posible si el fichero ya existe.

En los archivos secuenciales sólo se añaden registros al final. En los directos e indexados se localiza el lugar que deberá ocupar dicho registro, si este lugar está libre, se graba y si está ocupado, se le localiza otro sitio o se busca otra ubicación al registro antiguo y, después, se graba el nuevo en el que ocupaba el anterior, todo ello con una política preestablecida.

7. FICHEROS DE ACCESO SECUENCIAL

Los ficheros con este tipo de acceso pueden crearse en todo tipo de soporte de almacenamiento. Los registros se almacenan ocupando posiciones consecutivas, en el mismo orden en que han sido introducidos. Por tanto, los registros se organizan como una lista lineal. Con este tipo de acceso un registro en particular sólo puede accederse si antes se han accedido los que le preceden, sin embargo, si el soporte es

direccionable puede localizarse directamente un registro por el número que ocupa dentro de la serie.

Esta organización puede implementarse en soportes no direccionables, por ejemplo, cintas y direccionables, discos duros, CD's, etc.

Entre sus ventajas se encuentran las siguientes:

-Para localizar un registro sólo se requiere conocer el campo de búsqueda.

-Si la tasa de actividad es alta, es decir, es elevado el número de consultas o modificaciones, resulta una organización bastante eficiente.

-El soporte donde se ubica, o el tramo de memoria designado para almacenarlo si el soporte es direccionable, se aprovecha en su totalidad, es decir, no quedan huecos libres.

Entre sus inconvenientes se encuentran las siguientes:

- La consulta puntual de un registro puede ser lenta, especialmente en archivos muy grandes, puesto que requiere obligatoriamente el paso por todos los registros precedentes.

- Cualquiera que sea la operación que se desee realizar sobre un fichero con esta organización requiere procesar todo el archivo.

- Para actualizarlo es necesario usar un ficheros auxiliares.

7.1. Creación

El fichero secuencial puede crearse directamente con un editor de textos, si se trata de un fichero de texto, por ejemplo, el Bloc de Notas (Windows), gedit (Linux) o el editor del entorno integrado de desarrollo del lenguaje que se utilice o bien diseñando un programa para su creación. En cualquier caso, es necesario abrirlo antes de escribir en él los registros iniciales y cerrarlo cuando se acabe la escritura.

Si la creación se realiza mediante programa, el algoritmo generalizado puede ser el que sigue, teniendo en cuenta que las singularidades del detalle se tratarán cuando se implemente en Java, según los requerimientos establecidos en cada caso en particular.

PSEUDOCÓDIGO GENERALIZADO PARA CREAR UN FICHERO SECUENCIAL

```
<CREAR FICHERO_N>
inicio
   abrir FICHERO_N para escribir
   leer(condicion)  /* Se lee la VCB para controlar la
```

```
                    entrada de datos */
   mientras(condicion sea cierta)
      leer(datos)
      escribir en registro de FICHERO_N(datos)
      leer (condicion);
   fin_mientras
   cerrar FICHERO_N
fin
```

La instrucción *escribir en registro* de *FICHERO_N(datos)* es similar a *escribir(datos)*. En esta última, no se detalla explícitamente donde se escribe, porque es una instrucción que se utiliza para escribir en la salida estándar (la pantalla), mientras que la primera se utiliza para escribir en un archivo en particular.

7.2. Recorrido

Cuando un fichero ha sido creado y, por tanto, existe, puede recorrerse secuencialmente desde el primero al último de sus registros para procesarlos: leer su contenido, pintar en pantalla, actualizarlo, etc. Tras el último registro existe una marca de fin de fichero, por lo que, se hará el recorrido mientras no se detecte esa marca, o sea, este es un bucle controlado por fin de fichero.

Recuerde que en el capítulo 3 se nombraron los bucles controlados por fin de fichero junto a los demás tipos de bucles, diferenciados por sus VCB's. Esta nueva variable de control de bucle necesita, al igual que la VCB centinela, una lectura anticipada para poder ser evaluada correctamente y una lectura al comienzo de cada nueva iteración, lo que físicamente coincide con el final en la construcción del bucle.

El recorrido es una operación sistemática que se realiza siempre de la misma manera. Se presenta a continuación una propuesta de pseudocódigo para realizarlo.

```
<RECORRIDO>
inicio
  abrir Fichero_N /* No se indica si se abre para leer o
                  escribir, pues dependerá del proceso a
                  realizar sobre el  registro */
     Leer registro de Fichero_N(registro)
    Mientras(no FF)  //FF, fin de fichero
     <Procesar registro>
     Leer registro de Fichero_N(registro)
    Fin mientras
    Cerrar Fichero_N //Sólo se cierra si ha sido abierto
fin
```

La instrucción *Leer registro* de *Fichero_N(registro)*, es similar a *leer (registro)*. Mientras que, en esta última, se entiende que se lee desde la

entrada estándar (el teclado), en la primera se especifica la lectura desde un fichero en particular.

Los archivos deben estar abiertos el menor tiempo posible, puesto que la apertura bloquea de forma exclusiva y no puede ser usado por ningún otro programa.

Recordad que habrá que añadir el código para el control de errores en la apertura o la lectura cuando el algoritmo se implemente en un lenguaje particular.

7.3. Consulta

La operación de consultar supone buscar secuencialmente en el archivo información referente a un registro, hasta encontrar aquél cuyo valor del campo de búsqueda coincida con el valor buscado o hasta que se acabe el fichero. Si existen varios registros con el mismo valor del campo de búsqueda, se obtendrá el primero de ellos, salvo que se especifique como precondición buscarlos todos.

El archivo donde se busca, puede estar desordenado, en cuyo caso, si se detecta el final de fichero, puede afirmarse que el registro buscado no existe. Por el contrario, si el archivo está ordenado, no es necesario recorrerlo hasta el final para asegurar, en determinado momento del proceso dependiente del orden establecido, que el registro no existe en dicho archivo. Este hecho es similar al planteado en la implementación del algoritmo de búsqueda secuencial en arrays ordenados o desordenados.

7.4. Partición

Como se ha señalado anteriormente, el proceso de partición consiste en dividir los registros de un archivo entre varios, según alguna especificación establecida con anterioridad. Se estudian a continuación algoritmos de partir ficheros en razón de distintas y sencillas especificaciones.

PARTICIÓN POR CONTENIDO

El objetivo es dividir un archivo, según una condición de partición tal que determinará que registros se almacenan en un fichero y cuáles en otro. Por ejemplo, suponga que se desea partir un fichero de nombre `Fichero` en otros dos: *`Fichero1`* y *`Fichero2`*, de manera que pasen a *`Fichero1`* los registros de *`Fichero`* cuyo valor del *`CampoRegistro`* sea igual al valor prefijado de *`Campo`* y al *`Fichero2`* los registros que no cumplan esta condición.

Las líneas referentes a este proceso en el algoritmo generalizado podrían ser:

```
.......
abrir Fichero para leer
abrir Fichero1 para escribir
abrir Fichero2 para escribir
```

```
Si no hay errores de apertura en ninguno de ellos
   leer registro de Fichero
   mientras (no FF de Fichero)
      Si (campoRegistro == campo)
         Escribir registro de Fichero en Fichero1
      En otro caso
         Escribir registro de Fichero en Fichero2
      Finsi
      Leer registro de Fichero
   Fin mientras
   Cerrar Fichero, Fichero1, Fichero2
Finsi
.......
```

PARTICIÓN EN SECUENCIAS

Se trata de distribuir los registros de un fichero en otros ficheros, obedeciendo a secuencias alternativas que pueden ser de igual o diferentes longitudes.

Por ejemplo, suponga que se desea partir un fichero de nombre *Fichero* en otros dos, *Fichero1* y *Fichero2*, de manera que pasen a *Fichero1* los registros que ocupan las posiciones pares y a *Fichero2* los que ocupan posiciones impares, es decir, la longitud de las secuencias para la distribución de los registros es 1.

Se propone el siguiente algoritmo generalizado:

```
.......
abrir Fichero para leer
abrir Fichero1 para escribir
abrir Fichero2 para escribir
Si no hay errores de apertura
   leer registro de Fichero
   numReg = 1
   Mientras (no FF de Fichero)
      Si (numReg es par)
         Escribir registro de Fichero en Fichero1
      En otro caso
         Escribir registro de Fichero en Fichero2
      Finsi
      Leer registro de Fichero
      numReg = numReg + 1
   Fin mientras
   Cerrar Fichero, Fichero1, Fichero2
```

```
finsi
.......
```

7.5. Fusión o mezcla

La *fusión* o *mezcla* de archivos, conocida también como **intercalación**, consiste en agrupar en un sólo archivo los registros de otros archivos, manteniendo o no el orden establecido según las especificaciones. Esta operación no afecta a los ficheros de origen que intervienen en el proceso de fusión.

ARCHIVOS DE ORIGEN DESORDENADOS

La mezcla de archivos desordenados consistirá en almacenar secuencias de registros de determinada longitud procedentes de ambos archivos intercaladas en el archivo destino. Puede que el archivo destino se desee igualmente desordenado o que las especificaciones requieran ordenarlo a la vez que se realiza la fusión. Para el primero de los casos se propone, a continuación, un algoritmo para mezcla.

Siendo *Fichero1* y *Fichero2* los archivos desordenados a fusionar, *Fichero* el destino, también desordenado y *Longitud* el tamaño de las secuencias:

```
 ............
abrir Fichero para escribir
abrir Fichero1 para leer
abrir Fichero2 para leer
Si no existen errores de apertura
  leer registro de Fichero1
  leer registro de Fichero2
  mientras (no FF de Fichero1 O no FF de Fichero2)
    Para( cont=1; mientras no FF de Fichero1 Y
          cont<=longitud; cont++)
       Escribir registro de Fichero1 en Fichero
       leer registro de Fichero1
    Fin para
    Para( cont=1; mientras no FF de Fichero2 Y
          cont<=longitud; cont++)
       Escribir registro de Fichero2 en Fichero
       Leer registro de Fichero2
    Fin para
  Fin mientras
  Cerrar Fichero, Fichero1, Fichero2
finsi
 ............
```

Si se trata de fusionar ficheros desordenados generando a partir de ellos un fichero ordenado, bastaría con fusionar ambos ficheros y, a continuación, aplicar al fichero destino un método de ordenación de archivos.

ARCHIVOS DE ORIGEN ORDENADOS

Para fusionar archivos ordenados y conservar dicho orden en el fichero destino, pueden aplicarse diferentes algoritmos, se presenta a continuación uno de los más comunes:

Mezcla controlada por fin de fichero

El método de mezcla que se expone, se desarrolla para el caso en que sean dos los ficheros a fusionar y que estén ordenados ascendentemente por el valor del campo establecido como clave, aunque el método puede extrapolarse para fusionar cualquier número de archivos ordenados.

Explicación del método: suponiendo que se desea fusionar dos archivos secuenciales, *F1* y *F2*, ordenados ambos ascendentemente por el campo considerado clave, el algoritmo de mezcla consiste en recorrer simultáneamente los ficheros *F1* y *F2*, escribiendo el registro menor de ambos en el fichero destino *F* y leyendo a continuación del archivo de procedencia del registro copiado. Deberá tenerse en cuenta que el final de fichero puede no producirse a la vez en ambos archivos, por tanto, habrá que pasar al fichero destino los registros que queden en el fichero no finalizado.

El algoritmo de mezcla es igual al que se usa para actualizar, que se expone en el apartado 7.7, sin considerar los casos de error, es decir, sólo habrá modificaciones si hubiese dos registros con la misma clave, o altas, por lo que podrá implementarse sin problemas después de haber estudiado el algoritmo mencionado.

7.6. Clasificación u Ordenación

La clasificación de archivos se enfrenta a un problema: al no ser directamente accesibles los datos, no es posible realizar los intercambios necesarios para dejarlos ordenados, tal como se hace en el caso de ordenación de arrays. Por tanto, se han de buscar estrategias alternativas.

El criterio de clasificación u ordenación podrá ser por uno o más campos de forma ascendente o de forma descendente, aplicando, como se estudió en el capítulo 8, una ordenación híbrida o externa.

HÍBRIDA

Recuerde que la ordenación híbrida puede aplicarse a datos ubicados en almacenamiento externo que se cargan en memoria interna en un array, siendo aquí donde realmente se realiza la ordenación. Este método es aplicable sólo si la totalidad de la información a ordenar cabe íntegramente en la memoria central, en caso

contrario, no será posible utilizarlo. Hay que tener en cuenta que el tamaño máximo de un array dependerá del modelo de memoria que se utilice para compilar el programa. Este método de ordenación es mucho más rápido que los métodos de ordenación externa, por lo que es el más recomendable, si se puede utilizar.

El proceso consiste en cargar la información del archivo en un array y ordenar éste por alguno de los métodos conocidos de ordenación de arrays. Terminada la ordenación, deberá volver la información del array ya ordenado al archivo. El array tendrá la misma estructura que los registros del archivo y tantas casillas como registros tenga éste. Pueden usarse arrays dinámicos para los que se pedirá tanta memoria como registros tenga el archivo.

Módulo generalizado para ordenar un fichero

```
<OrdenHibrido FICHERO>
inicio
   <calcular número de registros del FICHERO>
   pedir memoria para el ARRAY
   <Volcar FICHERO en ARRAY>
   <Ordenar ARRAY>
   <Volcar ARRAY en FICHERO>
   Devolver memoria del ARRAY//si no es automática su devolución
Fin Ordenar
```

Si el lenguaje de programación que se utiliza tiene alguna funcionalidad que permita calcular el número de registros, se usará, pero en caso contrario, habrá que diseñar un subprograma que realice esta operación. El algoritmo para resolverlo no se detalla por ser demasiado simple, consiste en abrir el fichero para leer e ir contando los registros hasta fin de fichero.

El algoritmo para ordenar el array será cualquiera de los estudiados en el capítulo 9 o cualquiera que implemente el lenguaje que se use .

Los algoritmos para realizar el volcado del archivo al array y viceversa, son similares, y generalmente los lenguajes de compilación incorporan mecanismos para ello.

El fichero a ordenar será abierto y cerrado por el mismo módulo. Recuerde que es importante mantener los archivos abiertos el menor tiempo posible.

EXTERNA

La clasificación externa se realiza si el archivo no cabe en memoria central. Tiene la desventaja de ser mucho más lenta que la ordenación interna e, incluso, que la híbrida, por la gran cantidad de operaciones de acceso a memoria secundaria que se necesita realizar hasta tener ordenado los datos.

Para clasificar fuera de memoria se requiere un número de archivos auxiliares dependiente del método que se vaya a implementar. El método más común es el conocido **método de mezcla directa**.

Método de clasificación por mezcla directa

Explicación del método: consiste en realizar sucesivamente una partición y una fusión de archivos, con lo que se producen secuencias ordenadas de longitud cada vez mayor. Para realizar las sucesivas operaciones de particiones y mezclas, se requieren dos ficheros auxiliares con la misma estructura que el original.

La primera partición se hace en secuencias separadas por una distancia o longitud 1, y la fusión correspondiente produce sobre el archivo inicial secuencias ordenadas de longitud 2. Con cada nueva partición y mezcla se duplica la longitud de las secuencias ordenadas. El proceso finaliza cuando la longitud de la secuencia ordenada es igual o excede a la longitud del archivo a ordenar.

Se aplicará la estrategia comentada con el siguiente ejemplo, pero no se implementará el pseudocódigo del algoritmo por ser una mera adaptación de los de partición y fusión descritos anteriormente.

Ejemplo:

Se dispone originalmente de un fichero **F** formado por registros de enteros, cuyos valores se señalan a continuación. El criterio de ordenación será ascendente. Se usarán los ficheros auxiliares **Faux1**, **Faux2**, creados con la misma estructura que el original, sobre los que se irán volcando las sucesivas particiones.

Fichero F: 16, 18, 6, 74, 14, 12, 41, 40, 51, 85, 74, 26, 58, 27, 12

Partición en secuencias de longitud 1

Fichero **Faux1: 16, 6, 14, 41, 51, 74, 58, 12**

Fichero **Faux2: 18, 74, 12, 40, 85, 26, 27**

Mezcla de series de longitud 1

Fichero F: 16, 18, 6, 74, 12, 14, 40, 41, 51, 85, 26, 74, 27, 58, 12

Observe que el fichero original está ahora formado por pares ordenados de elementos.

Partición en secuencias de longitud 2

Fichero **Faux1: 16, 18, 12, 14, 51, 85, 27, 58**

Fichero **Faux2: 6, 74, 40, 41, 26, 74, 12**

Mezcla de series de longitud 2

Fichero F: 6, 16, 18, 74, 12, 14, 40, 41, 26, 51, 74, 85, 12, 27, 58

Observe que el fichero original está ordenado por grupos de cuatro elementos, es decir, cada cuatro elementos están ordenados ascendentemente entre ellos.

Partición en secuencias de longitud 4

Fichero **Faux1: 6, 16, 18, 74, 26, 51, 74, 85**

Fichero **Faux2: 12, 14, 40, 41, 12, 27, 58**

Mezcla de series de longitud 4

Fichero F: 6, 12, 14, 16, 18, 40, 41, 74, 12, 26, 27, 51, 58, 74, 85

Ahora los grupos ordenados son de ocho elementos.

Partición en secuencias de longitud 8

Fichero **Faux1: 6, 12, 14, 16, 18, 40, 41, 74**

Fichero **Faux2: 12, 26, 27, 51, 58, 74, 85**

Mezcla de series de longitud 8

Fichero F: 6, 12, 12, 14, 16, 18, 26, 27, 40, 41, 51, 58, 74, 74, 85

El proceso acaba cuando la nueva longitud que debe aplicarse para la próxima partición es mayor o igual que el número de registros del fichero original **F**. El número de registros es 15 y la longitud nueva que debe aplicarse para partir sería 16, como es mayor que el número de registros ha acabado el proceso de ordenación. Compruébese que efectivamente el archivo queda ordenado.

7.7. Actualización

Los archivos de situación, llamados también **maestros**, reflejan la situación actual de una entidad sobre algún aspecto de su actividad. En el desarrollo habitual de esa actividad, los datos sufren cambios, por lo que es necesario actualizar estos ficheros cada cierto tiempo para que reflejen fielmente el nuevo estado.

El fichero maestro se actualiza periódicamente de acuerdo a los procedimientos de la empresa. La actualización se realiza con los movimientos o transacciones relativos a los datos del fichero efectuados durante un periodo.

Las transacciones pueden ser suministradas en línea al proceso de actualización lo que supondría una actualización interactiva, aunque lo habitual es registrarlas en un fichero de movimientos por un periodo de tiempo que va desde la última a la nueva

actualización. Este periodo lo fija la política de la empresa y puede ser un día, una semana, un mes, etc.

Cuando se actualiza un fichero, al igual que ocurre en el mantenimiento de información en arrays, se pueden realizar las siguientes operaciones:

1. Incluir nuevos registros, operación que se conoce como altas.
2. Suprimir registros o bajas.
3. Cambiar el contenido de algún o algunos registros, modificaciones.

Por la propia naturaleza del fichero secuencial, las actualizaciones no pueden realizarse con eficiencia sobre sí mismo, salvo si se trata de modificar algún registro o añadir nuevos registros al final. Por esta razón, los movimientos suelen registrarse en un segundo fichero, conocido como de **movimientos**, clasificado igual que el fichero primario o maestro. El proceso de actualización consistirá en crear un nuevo archivo maestro, llamado generalmente **maestro actualizado**, generado a partir del antiguo maestro y del fichero de movimientos, añadiendo las altas, suprimiendo las bajas y copiando los registros que no han sufrido ninguna transacción. Cuando se termine la sincronización, tan sólo queda renombrar como maestro al maestro actualizado.

Es habitual generar simultáneamente un **fichero de errores** para su posterior proceso. Este fichero estará formado por los registros que no han podido almacenarse en el nuevo archivo actualizado por ser inconsistentes. Por ejemplo, altas de registros ya existentes y bajas o modificaciones de registros inexistentes son errores típicos.

Para realizar la actualización los ficheros de entrada, maestro y movimientos, y los que se generan, maestro actualizado y errores, deberán estar ordenados con el mismo criterio.

DESCRIPCIÓN DEL PROCESO

Para el desarrollo del método de actualización se considerarán las siguientes especificaciones:

- Los registros de los archivos no tienen repetidos los valores del campo por el que se establece el criterio de ordenación, es decir, en el maestro cada valor de este campo, conocido como clave de ordenación, es único.
- Cada uno de los registros tendrá como mucho un movimiento, pudiendo existir registros que no hayan sufrido ninguno.

El proceso consiste en recorrer simultáneamente los archivos maestro y movimientos, leyendo inicialmente un registro de cada uno de ellos. Se comparan los valores de sus respectivos campos clave y se analiza la situación que se presenta, que dará lugar a un tratamiento particular.

Es importante conocer cuáles y cuántas situaciones pueden darse en el proceso de emparejamiento de los ficheros. La fórmula 2^n-1, siendo *n* el número de ficheros que hay que emparejar, dice el número de situaciones distintas que se pueden encontrar durante la sincronización.

Para el caso en que sean dos los ficheros a emparejar, un maestro y un fichero de movimientos, se presentan tres situaciones distintas (2*2-1), que son las siguientes:

Registro de movimiento con registro en maestro

Una situación de este tipo significa que existe un registro en el fichero de movimientos cuya clave existe en el fichero maestro. Esta es una circunstancias típica de actualización.

- Si es una baja, se prescinde del registro del maestro, es decir, no se copia el registro de movimiento en el maestro actualizado. La baja vendrá señalada por marca de algún campo en particular establecido de antemano. Por ejemplo, si el registro dispone de un campo nombre puede marcarse la baja con * en este campo.

- Si es una modificación, se graba en el maestro actualizado el registro con las modificaciones oportunas.

- Si es un alta, se trata de un error, pues la clave ya existe.

En cualquiera de los tres casos se leerá un nuevo registro del fichero maestro y del de movimientos.

Registro de maestro sin registro en movimiento

Significa que se ha encontrado un registro del maestro que no ha sufrido ningún tipo de movimiento, por lo que deberá guardarse igual en el maestro actualizado y leer un nuevo registro del maestro.

Registro de movimiento sin registro en maestro

Obedece a una situación en que se encuentra un registro del fichero de movimiento que no tiene campo clave igual en el maestro. Puede tratarse de:

- Un alta, en cuyo caso se copia el registro de movimiento en el fichero maestro actualizado. En las especificaciones estará detallada la forma en que se establece si un registro que no está en el maestro es o no un alta.

- Un error, ya sea baja o modificación, por clave inexistente, en cuyo caso pasará al fichero de errores si está previsto.

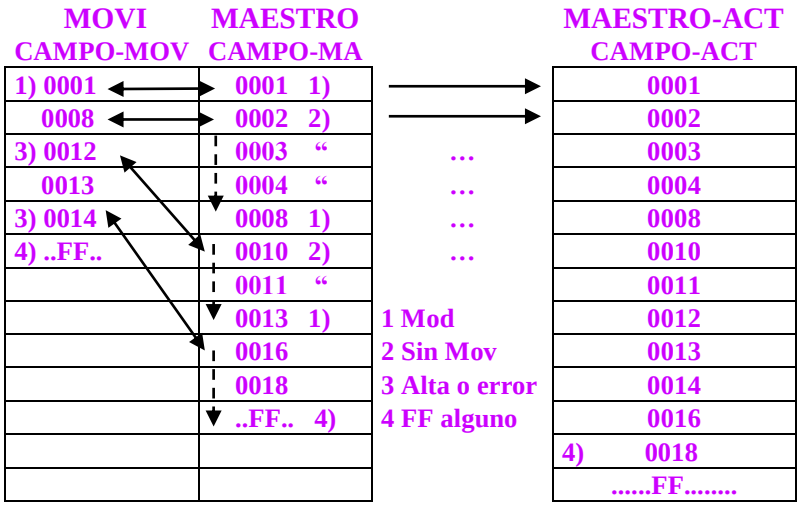
En ambos casos, se leerá el registro siguiente del fichero de movimientos.

Antes de implementar el algoritmo, se aplicará el proceso descrito anteriormente al ejemplo siguiente, donde se observarán todas las situaciones que se pueden presentar en el proceso de sincronización de dos ficheros secuenciales.

Se consideran los archivos: *MAESTRO*, el actual, *MOVI*, el de movimientos y *MAESTRO-ACT*, el que se genera a partir de los dos primeros y la clave de ordenación de los ficheros *CAMPO-MA* y *CAMPO-MOV*, respectivamente.

Hay que generar un maestro actualizado y el de errores, si fuera necesario, con la misma estructura que los anteriores. En el ejemplo, se señalarán en el pseudocódigo los posibles casos de error, donde se ejecutaría el tratamiento de error previsto, por lo que sólo se generará el maestro actualizado.

Los campos clave tienen los valores que aparecen en la figura siguiente y se considera, por simplicidad para entender el método, que se trata de valores numéricos.



Siga la traza del algoritmo sobre la gráfica de los ficheros. Las flechas señalan las lecturas y comparaciones que se realizan sobre el campo clave de los registros de los ficheros iniciales, maestro y movimiento y la escritura en el maestro actualizado. Siguiendo el algoritmo observará las situaciones señaladas como 1) 2) 3) y 4), que marcan sobre la gráfica de los archivos y sobre el pseudocódigo del algoritmo, las distintas situaciones que se presentan en la sincronización. Son las siguientes:

1) **Modificación (o baja si estuviera señalado como tal)**, modificación en el ejemplo**.**

2) Situación en la que un registro en maestro que no ha sufrido ningún movimiento, por lo que deberá grabarse en el maestro actualizado.

3) Se trata de un registro que no aparece en el fichero maestro original, por tanto, se trata de un alta o es un error, alta en el ejemplo.

4) Señala la finalización por encontrar la marca fin de fichero en alguno de los archivos a sincronizar. En el ejemplo, se ha acabado el fichero de movimientos y aún quedan archivos en el maestro, por lo que deberán pasar todos al maestro actualizado. Si finaliza primero el fichero maestro, pasarán al maestro actualizado todos los de movimientos que no sean errores.

PSEUDOCODIGO

```
Inicio
  Abrir para leer MOV y MAESTRO
  Abrir para escribir MAESTRO-ACT
  Si no hay errores de apertura
     Leer reg de MOV
     Leer reg MAESTRO
     Mientras(No FF de MOV y no FF de MAESTRO)
        Si(CAMPO_MOV == CAMPO_MA)
        // 1)Modificación
           Escribir registro actualizado en MAESTRO-ACT
           Leer registro MOV
           Leer registro MAESTRO
           /*Podemos añadir un mientras como en el caso 2),
           pero esta situación es menos habitual */
           /*Baja si estuviera especificado, en cuyo caso no
           se escribe en el actualizado */
        En otro caso
            Si(CAMPO-MOV > CAMPO-MA)   /* 2)Hay registro en
                                       maestro sin movimiento
                                       y debe grabarse en el
                                       maestro actualizado */
              Mientras (CAMPO-MOV >CAMPO-MA y no FF de
               MAESTRO)
                Escribir registro de MAESTRO en MAESTRO-ACT
                Leer registro de MAESTRO
              Fin mientras
            En otro caso
              Si(CAMPO-MOV < CAMPO-MA) /* 3)Es un alta o un
                                          error*/
                //Alta
                Escribir registro de MOV en MAESTRO
                /* Puede ser error y se ejecutaría
                 <Tratamiento de error> */
                Leer registro MOV
```

```
                    //Puede incluirse un mientras como en el
                     //caso 2). Aquí tampoco vale la pena,
                     además se pasa inmediatamente al mientras
                     inicial.
                Fin si
              Fin si
           Fin si
        Fin mientras
        /* 4)En este ejemplo se ha acabado el fichero de
         movimientos y aún quedan archivos en el maestro */
        Mientras(no FF de MAESTRO)
          Escribir registro de MAESTRO en MAESTRO-ACT
          Leer registro de MAESTRO
        Fin mientras
        Mientras(no FF de MOV)
           //Alta
          Escribir registro de MOV en MAESTRO-ACT
          /* Puede ser error y se ejecutaría <Tratamiento de
           error> */
          Leer registro de MOV
        Fin mientras
        Cerrar archivos
    Finsi
Fin
```

Acabada la sincronización tan sólo hay que renombrar como maestro al maestro actualizado. Esta operación puede realizarse desde el propio programa o desde el sistema operativo.

7.8. Actualización interactiva

La actualización interactiva consiste en suministrar directamente desde la entrada estándar los movimientos, para procesarlos sobre el fichero maestro directamente en ese mismo instante. Hay que considerar que las consultas y modificaciones pueden realizarse sin ningún problema, sin embargo, las altas sólo pueden añadirse al final y las bajas deben realizarse por marca. Si el fichero está desordenado, insertar las altas al final no supone problema alguno, pero si el fichero estuviera ordenado, habría que volver a ordenar con cada alta realizada. A estas operaciones habrá que añadir, cada cierto tiempo, la operación de reorganización del fichero.

Puede comprenderse la complejidad y poca eficiencia del mantenimiento de un fichero en estas condiciones.

La actualización interactiva es más frecuente y eficaz aplicada a los ficheros directos que a los secuenciales.

8. FICHEROS DE ORGANIZACIÓN DIRECTA

Para poder organizar los archivos de forma directa es imprescindible ubicarlos sobre un soporte de almacenamiento direccionable, que permita acceder directamente a un elemento cualquiera, sin necesidad de pasar por todos los que le preceden. Con este tipo de organización, los elementos se colocan y acceden indicando el lugar relativo que ocupan dentro del conjunto de direcciones posibles. De esta forma, se pueden leer y escribir registros en cualquier orden y en cualquier lugar. Para que esto sea posible, se establece una relación entre el valor del campo clave y la dirección de almacenamiento del registro. Hay que subrayar que es responsabilidad del programador establecer dicha relación, en caso de que el lenguaje utilizado para programar no incorpore ninguna utilidad al respecto.

Por tanto, los ficheros de organización directa o aleatoria se caracterizan por:

a. La ubicación de los registros se realiza a través de una clave que establece, por un lado, su posición dentro del fichero y, por otro, la posición de memoria donde está almacenado.

b. La dirección de un registro dentro del soporte de almacenamiento se obtiene a partir de la clave del registro aplicando algunas especificaciones:

    1. Si la clave es numérica, se aplica un algoritmo de transformación de clave que permita obtener direcciones dentro del rango de valores de las direcciones físicas de memoria disponibles. De esta forma se establece la relación entre posición lógica (clave) y dirección física (memoria) donde se realiza el almacenamiento.

    2. Si la clave no es numérica, debe aplicársele en primer lugar algún método, generalmente matemático, para obtener a partir de ellas valores enteros positivos, a los que aplicar en segundo lugar los algoritmos de transformación de claves numéricas.

Es necesario contar con algoritmos de transformación de claves eficientes, a los que hay que exigirles al menos tres características:

a. Deben usar eficientemente la memoria disponible.

b. Deben establecer una correspondencia unívoca entre dirección lógica y física.

c. Es deseable que produzcan el menor número de **sinónimos** que son registros que teniendo diferentes claves, cuando se les aplica el método de transformación de clave, generan las mismas direcciones de almacenamiento.

Además, para aplicar un método u otro habrá que tener en cuenta detalles como: longitud de la clave, su rango de valores, soporte físico de almacenamiento que se va a utilizar, previsiones sobre la evolución del fichero, etc.

Aplicado un método de transformación de claves para ubicar registros en memoria, es obvio, que para recuperar la información de tales registros habrá que aplicar la misma fórmula que se utilizó en su almacenamiento.

8.1. Modos de direccionamiento

Cualquier algoritmo para direccionar se aplica sobre claves numéricas. Por tanto, antes de utilizarlos debe convertirse en numérica la clave que no lo es.

TRANSFORMACIONES DE CLAVES ALFABÉTICAS

Aunque se puede usar para ello diversos métodos, es bastante usual utilizar como base de transformación el valor numérico que corresponde a cada letra, según el orden alfabético que ocupa en el abecedario, en cuyo caso no podría distinguirse entre mayúsculas y minúsculas. También es común el uso del código ASCII correspondiente a cada letra, con lo cual sí es posible distinguir mayúsculas y minúsculas. Además, tiene la ventaja de poder utilizar todo el conjunto de caracteres ASCII para formar claves.

Ejemplo: si el valor de la clave es “ABC” su transformación en clave numérica aplicando el orden en el abcedario es 123, pero también tendría la misma transformada la clave “abc”. Aplicando el código ASCII para transformación de clave, a la primera correspondería 656667 y a la segunda 979899.

Se comentan a continuación algunos de los mecanismos más utilizados para direccionar, aplicados todos ellos a claves numéricas.

DIRECCIONAMIENTO DIRECTO

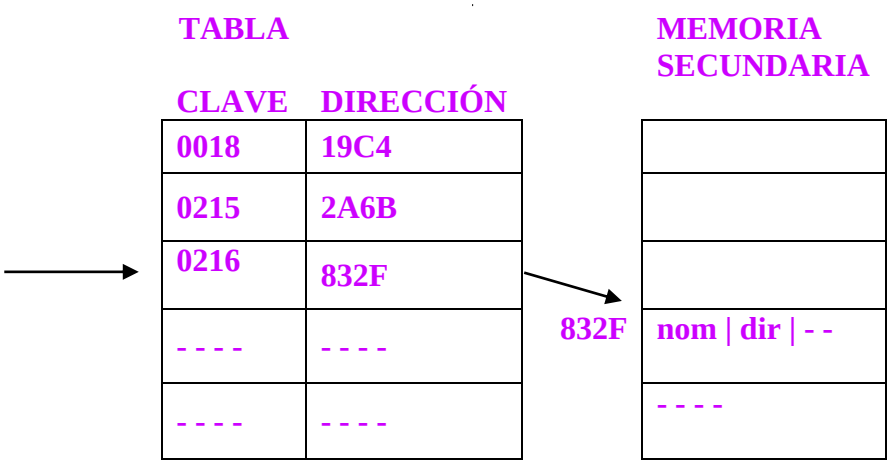
Con este método **la dirección de un registro es directamente la clave**, por lo que, es obvio, que el método sólo será aplicable si la clave es numérica. Aplicando esta forma de direccionamiento el fichero, quedará ordenado por el campo clave. El método incorpora una paradoja, por un lado tiene como ventaja la de ahorrar memoria, puesto que no es necesario que la transformada de clave forme parte del registro. Por otro lado, añade la desventaja de desperdiciar memoria, ya que se generan bastantes huecos correspondientes a las claves que no se transforman porque no existen en el fichero. Además, si las claves del fichero tienen valores muy dispersos, también se generan muchos huecos en el almacenamiento.

Es el método menos utilizado de los que se aplican para direccionar archivos.

DIRECCIONAMIENTO POR TABLA

Se asocia a cada clave una dirección, y se guardan en una tabla las claves y las direcciones correspondientes a cada uno de ellas. La tabla se carga en memoria al abrir el fichero y se mantiene mientras se realicen operaciones sobre dicho fichero.

El proceso de localización de un registro consiste en buscar en primer lugar en la tabla la clave correspondiente y a continuación acceder a la zona de memoria indicada en la tabla junto a la clave, este es el lugar donde se encuentra el registro buscado. Se puede observar en el gráfico que aparece a continuación.



Es conveniente que la tabla esté ordenada para que el acceso sea más rápido, pudiendo aplicar en este caso la búsqueda dicotómica, más rápida que la secuencial. La tabla puede adoptar distintas formas para optimizar su tamaño o velocidad de acceso, pudiendo usar arrays (más lento), listas enlazadas o árboles de búsqueda para su implementación.

La principal dificultad de este método es la cantidad de espacio que ocupa la tabla y el tiempo invertido en la búsqueda. Además, las actualizaciones afectan tanto a la tabla como al área de datos. También es necesario mantener una tabla con los espacios vacíos que pueden utilizarse para almacenar nuevos registros.

DIRECCIONAMIENTO CALCULADO

El método consiste en la aplicación de una fórmula de cálculo matemático sobre las claves para obtener una dirección de almacenamiento. Se puede representar formalmente como:

Dirección = F (clave)

Siendo *F* la fórmula que se aplica sobre la clave.

8.2. Transformaciones calculadas de claves o Hashing

A las técnicas que estudian las distintas formas de cálculo de direcciones y las peculiaridades de cada una de ellas se les denomina técnicas de **Hashing**. Todas tienen como característica común destacable provocar la mínima cantidad de sinónimos y huecos en las transformaciones de claves. Véanse a continuación algunas.

DIVISIÓN POR NÚMERO PRIMO

Consiste en dividir el campo clave, numérico, entre el número primo mayor y más cercano al número total de registros, tomando como dirección el resto de esta división.

Este método tiene la ventaja de no precisar cálculos posteriores ya que, al ser el número primo más o menos igual al número de registros posibles en el fichero, el resto siempre estará en el rango permitido.

Ejemplo: supongamos que el valor del campo clave que se desea almacenar es 754 y se prevé que el archivo tenga 100 registros. El primo más cercano mayor que 100 es 101. El resto de la división entera 754/101 es 47. Por tanto, la clave transformada o dirección de almacenamiento del registro cuyo campo clave tiene como valor 754 es 47.

MÉTODO DEL CUADRADO

Se aplica a una clave numérica no muy larga. Consiste en elevar la clave al cuadrado y elegir unas cuantas cifras intermedias como dirección de almacenamiento. Las cifras a elegir quedan preestablecidas, antes de aplicar el método, y serán siempre las mismas al obtener las transformaciones de cada clave.

Ejemplo: aplicamos el nuevo método de direccionamiento al mismo caso del ejemplo anterior. El cuadrado de 754 es 568516. Si la política es utilizar como clave los dos dígitos que anteceden al último, la clave transformada o dirección de almacenamiento es 51.

MÉTODO DE MULTIPLICACIÓN Y RESTO

El método consiste en multiplicar la clave por un número, dividir a continuación el resultado por otro número y tomar como dirección el resto de esta división. El multiplicador y el divisor serán números establecidos antes de aplicar por primera vez el método y no pueden variar de unos registros a otros.

Ejemplo: supongamos que el multiplicador es 35 y el divisor 200. Siguiendo con el ejemplo anterior, aplicaremos el nuevo método a un registro cuyo campo clave tiene valor 754, de manera que 754 * 35 = 26390 y el resto de 26390/ 200 es 190, que será el valor de la clave transformada.

8.3. Tratamiento de sinónimos

Es probable que al aplicar algunos de los métodos descritos anteriormente, resulten claves transformadas iguales para distintos registros, o sea, sinónimos o colisiones. Como la aparición de sinónimos es relativamente frecuente, antes de grabar un registro se leerá la posición calculada, para comprobar si está ocupada, en cuyo caso habrá que buscar otra dirección a la que enviar el registro sinónimo. Si la dirección calculada está libre, se graba el registro sin más.

Para realizar una lectura, se aplica la fórmula a la clave y se busca en la dirección obtenida. Si en esa posición no está el registro buscado, se intenta su localización en el área destinada a sinónimos.

Si debe buscarse otra dirección, puede aplicarse cualquiera de los métodos de tratamiento de sinónimos que se establezca. Los más usuales son los siguientes:

ASIGNACIÓN CONSECUTIVA

Si la posición calculada para un registro está ocupada, es decir, se ha generado un sinónimo, el método consiste en asignarle la primera posición libre que se encuentre a partir de la que le corresponde.

Ejemplo: si a la clave *ClaveA* le corresponde a la dirección *DirecciónD* y está ocupada, se inspeccionará *DirecciónD+1*, *DirecciónD+2*... hasta encontrar una dirección libre o el fin del trozo asignado para almacenamiento.

En cuanto a la lectura, si al buscar el registro, accede a la posición *DirecciónD*, y la clave no es *ClaveA*, se seguirá buscando a partir de esa posición en *DirecciónD+1*, *DirecciónD+2*...hasta encontrar el registro, el primer espacio libre o el fin de la zona de almacenamiento del fichero. Si se da cualquiera de los dos últimos casos, se puede concluir que el registro no está en el fichero.

Este método tiende a acumular los registros en bloques, lo que amplia la posibilidad de que hayan sinónimos. Por otro lado, cuando el fichero está muy lleno y se producen muchas colisiones, se realizará prácticamente un procesamiento secuencial en gran medida.

ASIGNACIÓN CALCULADA

Con este método, cuando un registro encuentra su posición de almacenamiento ocupada, se le aplica a la dirección obtenida, el mismo método de cálculo aplicado en primera instancia o bien un método de cálculo distinto. Para la lectura se aplicarán los mismos métodos usados para direccionar.

ZONA DE EXCEDENTES

Con esta modalidad el fichero queda dividido en dos zonas, *zona primaria*, donde se almacenan los registros cuyas claves transformadas no son sinónimos y *zona*

de excedentes a la que irán todos los sinónimos que se produzcan. Esta zona se conoce también como de *overflow* o *desbordamiento*.

La zona de excedentes se puede gestionar de dos formas:

Organización Secuencial

Si aplicada una fórmula de cálculo se genera un sinónimo, el registro se almacenará en el primer espacio libre de la zona de memoria reservada para excedentes. Como la escritura en esta zona se realiza de forma secuencial, la lectura se hará del mismo modo.

Organizada mediante listas enlazadas

Cuando se produce un sinónimo, se coloca en la zona de excedentes formando parte de una lista enlazada cuyo primer elemento es el registro sinónimo colocado en la zona primaria. Conviene que la zona primaria sea grande, para que la lista de excedentes sea corta.

8.4. Ventajas e inconvenientes de la organización directa

Este tipo de organización confiere al fichero las siguientes ventajas:

- Tiene la ventaja de ser muy rápido el acceso a los registros, por lo que es eficaz usarlos en aplicaciones cuyas consultas y actualizaciones se dan en tiempo real y el tiempo de acceso sea crítico.

- El acceso a registro es directo mediante su propia clave, sin que deba pasarse por los registros que le preceden.

- La actualización y recuperación de un registro es inmediata, sin necesidad de usar ficheros auxiliares.

- Las supresiones son posibles por marcas y físicamente.

Y los siguientes inconvenientes:

- Es responsabilidad del programador generar el software para su gestión o buscar utilidades en el mercado que solucionen el problema.

- Presenta la desventaja de aprovechar poco el espacio en el soporte de almacenamiento, ya que por una parte los registros no deben estar agrupados en bloques y, por otra, hay que dejar semivacío el soporte para que no se produzcan muchos sinónimos.

- En los ficheros directos, en general no es utilizable el método de acceso secuencial, puesto que el fichero no está ordenado respecto ningún campo.

Su creación exige un análisis y una gran planificación, para dar con un método de cálculo de direcciones apropiado, o para gestionar el uso de tablas de claves y direcciones de dimensiones bastante grandes.

9. FICHEROS DE ORGANIZACIÓN SECUENCIAL INDEXADA

Surgen con el objeto de aprovechar las ventajas de los ficheros de organización secuencial y de organización directa, tratando de paliar al mismo tiempo los inconvenientes de ambos.

En este tipo de organización todos los registros se identifican unívocamente por un identificativo que se conoce como **clave primaria**. Esta clave es un campo perteneciente al registro que no puede repetirse.

9.1. Generación de un fichero secuencial indexado

Un fichero secuencial indexado consta de tres zonas o áreas: primaria, de índices y de excedentes.

ÁREA DE ÍNDICES

Los índices constituyen una información generada y actualizada por el propio sistema permanentemente, y organizada de forma secuencial. Esta zona tiene la estructura de un archivo secuencial con dos campos por registro. Los registros del área de índice se conocen como **entradas**. Cada entrada contiene la dirección de comienzo de un bloque y la clave más alta de ese bloque.

Generalmente, los índices se organizan en tres niveles que se llaman de mayor a menor rango: **índice maestro** que se remite al **índice de cilindros,** que se remite a su vez al **índice de pista**.

Existen compiladores que gestionan automáticamente este tipo de organización, por ejemplo, COBOL, en cuyo caso es transparente al programador su creación y mantenimiento. En aquellos casos en que no exista esta funcionalidad, el área de índices deberá crearse mediante código y podrá implementarse en una tabla, lista, fichero o cualquier estructura de datos que convenga.

ÁREA PRIMARIA O ÁREA DE REGISTRO

En el área principal se almacenan los registros que constituyen el fichero, ordenados de forma ascendente por su clave. Esta zona está dividida en segmentos, en cada uno de los cuales se almacena un cierto número de registros secuencialmente.

El acceso a los segmentos es directo y el acceso a cada uno de los registros dentro de un segmento es secuencial

ÁREA DE EXCEDENTES

Ya comentada para los ficheros de organización directa, en ella se guardan los registros que no caben en el área principal.

Ejemplo: en el gráfico siguiente se observa la configuración del fichero con una sola zona de índices. Observe que la zona primaria está formada por distintos segmentos de la misma longitud, de los que se indica su comienzo en memoria, con los registros almacenados por orden ascendente. En la zona de índices, cada entrada almacena la dirección de comienzo de cada segmento y la clave del último registro de dicho segmento.

ÁREA PRIMARIA	
Blasco Palacios	1000
Cano Godoy	
Chacón Barbero	
Corrales Ríos	
.....	.....
Escobar Torres	1100
Jurado Barroso	
López Ojeda	
Márquez Gallego	
MárquezVargas	
Martín Cano	
Osuna Herrero	1200
.....	.....
Perera Rincón	
Pérez Reina	
Rivas Carmona	1300
.....	.....
Román Álvarez	
Rubio Vizcaya	1400
Ruiz Riejos	
.....	.....
Soriano Fernández	

ÁREA DE ÍNDICES	
Dirección	Clave
1000	De la Rosa Sánchez
1100	Martín Cano
1200	Pérez Reina
1300	Román Álvarez
1400	Soriano Fernández

9.2. Ventajas e inconvenientes de la organización secuencial indexada

Entre sus ventajas se encuentran:

- Simula el proceso de acceso directo a registros sin el inconveniente de los "huecos".

- Se puede actualizar un registro sin necesidad de archivo auxiliar en un tiempo aceptable. Las actualizaciones pueden realizarse:
    - **Por bloques**, en base a un lote de transacciones ocurridas en un periodo determinado.
    - **En línea**, gracias a la posibilidad de acceder directamente al fichero.
- Si tiene una alta tasa de actividad de consulta a todo el archivo, resulta muy rápido por su organización secuencial.

Entre sus inconvenientes se encuentran:

- Las bajas solo se pueden efectuar por marca.
- Las altas van siempre al área de *overflow*, por lo que esta zona puede crecer excesivamente, lo que conlleva una disminución de la rapidez. A veces, puede llegar a crecer tanto que sea necesaria una reorganización del fichero, lo que genera una pérdida de tiempo considerable.
- Necesita un espacio grande de memoria para implementar la tabla de índices.
- No es aconsejable esta organización para ficheros muy volátiles, que tengan un alto grado de supresiones y actualizaciones.

10. INCONVENIENTES DE LAS ESTRUCTURAS DE DATOS ARCHIVOS

Para procesar la información que maneja cualquier entidad se utilizan, por un lado, los archivos de datos donde se almacena dicha información y, por otro, los archivos de programas que actúan sobre ellos. Los datos deben estar estructurados en los archivos de acuerdo a los tratamientos que deban sufrir, posteriormente, por los programas de aplicación. Los programas deben diseñarse teniendo en cuenta los procesos deseados, formatos de entrada y de salida, etc. Además, los programas creados para que realicen una determinada funcionalidad, no pueden utilizarse para una aplicación diferente. Todo esto provoca serios inconvenientes entre los que podemos destacar los siguientes:

- **Redundancia de datos**. Es frecuente que muchos de los datos que maneja una entidad estén repetidos en varios archivos, lo que supone un despilfarro de memoria, además de aumento en la gestión de dichos datos y serias dificultades en su tratamiento.
- **Conflictos serios en el mantenimiento de los archivos**. Efecto directo de la redundancia de datos, son las dificultades en las actualizaciones, que deben realizarse sobre cada uno de los archivos donde se encuentren almacenados los datos a actualizar.

Por otro lado, es posible que se realicen actualizaciones por usuarios diferentes sobre datos idénticos, por lo que pueden aparecer *datos incongruentes*, es decir, valores diferentes en los distintos archivos para datos que deben ser idénticos. Es obvia, la complejidad y la pérdida de tiempo que todo esto supone, además del aumento de errores que puede provocar.

- **Conflictos en la seguridad y confidencialidad de los datos**. Cuando los distintos programas que forman una aplicación deben ser manipulados por usuarios diferentes, es complicado realizar una política segura de acceso a los datos que garantice la confidencialidad y su correcto tratamiento, e impedir que sobre ellos se realicen operaciones no permitidas.

- **Acoplamiento entre los programas y los archivos**. Existe una gran interdependencia entre los programas y los datos. Cualquier cambio en la estructura de un archivo exige modificar todos los programas que acceden a dicho archivo.

Todos estos inconvenientes dan lugar a la implementación de una nueva estructura de datos que los soluciona en gran medida, son las conocidas como *Bases de Datos*.

Conceptualmente, una **Base de Datos** es un sistema organizado constituido por una colección de datos, pertenecientes a diversos archivos que están integrados en una sola estructura global. Los datos son totalmente independientes de los programas de aplicación y el sistema está exento de redundancias perjudiciales. El acceso es inmediato a los datos de la base deseados, así como a los relacionados con ellos y esta operación puede realizarse mediante distintas formas y por múltiples usuarios al mismo tiempo.

Quién organiza y gestiona los datos de un sistema de este tipo es el conocido como **Sistema Gestor de la Base de Datos** (**SGBD**), también llamado en el argot informático **Motor de la Base de Datos**. El SGBD es el software que funciona como intermediario entre la información contenida en la base y los programas que la utilizan.