MÓDULO 3 | PROGRAMACION COBOL– Archivos

Contenido

1 ARCHIVOS o FICHEROS DE ENTRADA / SALIDA

Hasta el momento hemos trabajado con pequeñas cantidades de datos cargados manualmente por el usuario y almacenados en la memoria de la maquina.

Si la cantidad de datos a manipular fuera mayor, o fueran producto de una ejecución en serie de distintos programas sería engorroso tener que cargar los datos a mano, y el margen de error sería mucho más amplio que en el caso anterior, por lo tanto cargar datos en forma manual no es la forma adecuada de trabajar en estos casos.

La forma correcta de trabajar en estos casos es utilizar soportes donde se conserven grabados los datos que genera o necesita para ejecutar un determinado programa.

Un archivo es una colección de información que almacenamos en un soporte, generalmente el disco rígido, para poder manipularla en cualquier momento.

Esta información se almacena como un conjunto de registros, conteniendo todos ellos los mismos campos.

Por ejemplo, si quisiéramos almacenar en un archivo los datos referentes los artículos que comercializa un almacén, podríamos determinar lo siguientes atributos para cada registro:

CODIGO	DENOMINACION	PRECIO	CANT. DISPONIBLE
001	ACEITE	$ 4,50	9

Cada campo almacenará el tipo de dato correspondiente, es decir “Denominación” será alfanumérico mientras que “Precio” será numérico decimal. El conjunto de campos descritos forma un registro, y el conjunto de registros forma un archivo que se almacenará en un disco con un nombre, por ejemplo *ARTICULO.DAT*

Existen distintas formas de organizar un archivo, por lo tanto tambien hay diversas formas de acceder a la información contenida. A lo largo de este Módulo veremos cuáles son estas formas y cómo se manipula la información en cada caso

Un **archivo** o fichero es una **colección de información** que almacenamos en un **soporte** para poder manipularla en cualquier momento.

Esta información se almacena como **un conjunto de registros**, conteniendo todos ellos generalmente los mismos campos.

Campo elemental almacena un **dato numérico o de caracteres**.

1.1 Organizaciones básicas:

Existen tres organizaciones básicas de los registros de un fichero, de cuya combinación se derivan multitud de organizaciones posibles. Estas son:

- Secuencial.
- Relativa.
- Secuencial Indexada.

1.2 Modos de acceso:

En cuanto a los modos o tipos de acceso, distinguimos dos:

- Acceso secuencial.
- Acceso aleatorio o directo.

Se habla de *acceso secuencial* cuando se van accediendo a posiciones sucesivas, esto es tras acceder a la posición N, se accede a la posición N+1, y se habla de *acceso aleatorio o directo* cuando se accede directamente a la posición deseada, sin necesidad de acceder a las posiciones que le preceden.

Según el tipo el tipo de organización empleada para crear un fichero el modo de acceso puede ser:

ORGANIZACIÓN	CREACIÓN	RECUPERACIÓN
Secuencial	Secuencial	Secuencial
Relativa	Secuencial o aleatoria	Secuencial o aleatoria
Secuencial indexada	Secuencial o aleatoria	Secuencial o aleatoria

1.3 Archivos de organización secuencial

El orden en que los registros están posicionados físicamente coincide con el orden en que han sido grabados. Cada registro tiene un único predecesor y un único sucesor (salvo el primero y el último que, por razones obvias, no tienen predecesor ni sucesor, respectivamente).

Una vez establecido este orden mediante la grabación sucesiva de registros en el archivo, no puede ser modificado de ninguna manera.

El único ***modo de acceso*** permitido para estos archivos es ***secuencial***, es decir solo pueden leerse los registros desde el principio hasta el final en el orden en que están grabados, y no pueden grabarse de otra forma que no sea a continuación del último.

La declaración de un archivo secuencial se realiza de la siguiente forma:

```
SELECT [Nombre-lógico] ASSIGN TO DISK [nombre-físico]
                       ORGANIZATION IS SEQUENTIAL
                       ACCESS MODE IS SEQUENTIAL
                       FILE STATUS IS [nombre-file-status]
```

Ejemplo:

```
FILE-CONTROL.

    SELECT E1DQENT0                ASSIGN TO E1DQENT0
                                   FILE STATUS IS WSS-FS-E1DQENT0
                                   ORGANIZATION IS SEQUENTIAL.
```

1.4 Archivos de organización relativa

Los registros son identificados por su ***posición relativa*** al inicio del archivo. El primer registro tiene el número de registro relativo 1, el décimo tiene el 10, y así consecuentemente. El archivo puede ser pensado como una serie de casilleros numerados consecutivamente, y que pueden o no contener un registro.

Los modos de acceso permitidos para los archivos relativos son secuencial ***SEQUENTIAL***, ***RANDOM*** o ***DYNAMIC***. Funcionan de la misma manera que los archivos de organización indexada, pero en este caso no se especifica el valor de la clave sino el número relativo del registro para poder posicionarse.

La declaración de un archivo relativo se realiza de la siguiente forma:

```
SELECT [Nombre-lógico]  ASSIGN TO DISK [nombre-físico]
                        ORGANIZATION IS RELATIVE
                        ACCESS MODE IS DYNAMIC
                        RELATIVE KEY IS [nombre-clave]
                        FILE STATUS IS [nombre-file-status]
```

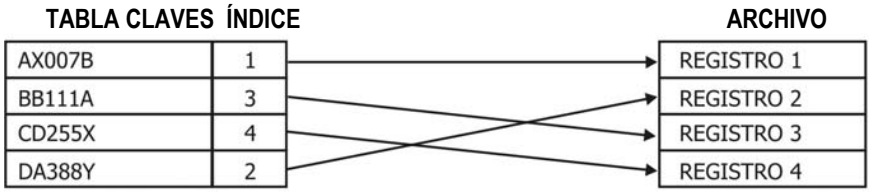
Ejemplo:

```
SELECT JBFCAIM0                 ASSIGN TO JBFCAIM0
                                ORGANIZATION IS INDEXED
                                ACCESS MODE  IS SEQUENTIAL
                                RECORD KEY   IS WSV-REG-PRINC
                                FILE STATUS  IS WSS-FS-JBFCAIM0.
```

1.5 Archivos de organización indexada

La organización secuencial indexada es un modelo de almacenamiento de datos que se basa en una **tabla de claves** que actúa como índice y en un **archivo de datos**. Para clarificar un poco, pensemos en un libro, el cual consta de un índice ordenado por número de página y el contenido del libro en si. De esta forma podremos acceder rápidamente a la información, buscando en el índice el número de página donde se encuentra la misma, y a continuación ir directamente a esa página.

Es así que en cada registro del archivo existe un campo (simple o compuesto) cuyo contenido forma la ***clave del registro***, siendo siempre el o los mismos campos en todos los registros del archivo.



En la tabla se encuentran clasificadas de menor a mayor las claves de acceso a los registros del archivo.

Este modelo de almacenamiento asocia a cada una de las claves el primer registro libre del archivo de datos. Es decir, COBOL automáticamente genera dos archivos, uno con las parejas de datos ***clave-número de registro***, que siempre se tendrá ordenado por la clave de menor a mayor, y otro con los datos.

Un archivo indexado puede además hacer uso de ***claves alternativas*** que permiten acceder al archivo mediante diferentes órdenes lógicos de los registros.

Los modos de acceso permitidos para los archivos indexados son ***SEQUENTIAL***, ***RANDOM*** o ***DYNAMIC***.

Cuando se leen o escriben archivos indexados en forma *SEQUENTIAL*, el orden se da por los valores de las claves de cada registro.

El acceso *RANDOM* permite acceder a un registro en particular, especificando el valor del campo clave.

El acceso *DYNAMIC* se realiza posicionándose en algún registro especificado por el valor de su clave, y realizar a partir de allí una lectura secuencial.

La declaración de un archivo indexado se realiza de la siguiente forma:

```
SELECT [Nombre-lógico]  ASSIGN TO DISK [nombre-físico]
                        ORGANIZATION IS INDEXED
                        ACCESS MODE IS RANDOM
                        RECORD KEY IS [nombre-clave]
                        FILE STATUS IS [nombre-file-status]
                        ALTERNATE RECORD KEY IS
                                  [nombre-clave-secundaria]
                        [WITH DUPLICATES]
```

Ejemplo:

```
SELECT E2DQEMB1                 ASSIGN TO E2DQEMB1
                                ORGANIZATION IS INDEXED
                                ACCESS MODE IS DYNAMIC
                                RECORD KEY IS WSV-FEMB1-CLAVE
                                FILE STATUS IS WSS-FS-E2DQEMB1.
```

2 DECLARACIÓN

Esta sección es opcional, y puede ser omitida si el programa no utiliza archivos. El nombre de la misma, ***INPUT-OUTPUT SECTION***, deberá figurar cuando se especifique alguno de sus párrafos.
Su Formato es el siguiente:

```
INPUT-OUTPUT SECTION.
 FILE-CONTROL.
  { entrada de control de los ficheros } ...
I-O CONTROL.
{ entrada de control de E/S } ...
```

2.1 FILE-CONTROL

Este párrafo es utilizado en forma exclusiva por entradas ***SELECT***, mediante la cual se asigna un *nombre lógico* a cada archivo utilizado y mediante cláusulas se definen los atributos del mismo.

```
      1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123...

      ENVIRONMENT DIVISION.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
          SELECT OPTIONAL ARCHSEC ASSIGN TO DISK `C:\ARCHIVO1.DAT`
                 FILE STATUS IS WSS-FS-SEC.
          SELECT ARCHIDX ASSIGN TO DISK `C:\ARCHIVO2.DAT`
                 ORGANIZATION IS INDEXED
                 ACCESS MODE IS SEQUENTIAL
                 RECORD KEY IS REG-COD-CLI
                 ALTERNATE RECORD KEY IS REG-APE-CLI
                                      WITH DUPLICATES
                 FILE STATUS IS WSS-FS-IDX.
          SELECT ARCHREL ASSIGN TO DISK `C:\ARCHIVO3.DAT`
                 ORGANIZATION IS RELATIVE
                 ACCESS MODE IS RANDOM
                 RELATIVE KEY IS WSV-NUM-REG
                 FILE STATUS IS WSS-FS-REL.
```

2.1.1 Cláusulas

SELECT
Permite crear el descriptor de un archivo, el nombre lógico que tendrá ese archivo dentro del programa.

ASSIGN TO
Se utiliza para especificar el tipo de dispositivo, el más utilizado en PC es “DISK”. En ese caso, a continuación, se indica la ruta del archivo físico que almacenará la información. Otro tipo de dispositivo utilizado generalmente es PRINTER, que indica que la salida será impresa.

ORGANIZATION IS
Permite especificar el tipo de organización del archivo declarado. Las opciones para esta cláusula son: SEQUENTIAL, INDEXED, RELATIVE o LINE SEQUENTIAL. Si se omitiera esta cláusula, COBOL asumirá por default que la organización es secuencial.

ACCESS MODE IS
Permite especificar el tipo de acceso al archivo declarado. Las posibilidades son: SEQUENTIAL, RANDOM o DYNAMIC.

FILE STATUS IS
Permite asociar una variable en la cuál se almacenará el estado del archivo (abierto, cerrado o fin de archivo, entre otros). Cada vez que se realice una operación con el archivo, el sistema actualizará su valor. La variable asociada debe estar definida dentro de la WORKING STORAGE SECTION como alfanumérica de dos posiciones.

RECORD KEY IS
Se utiliza sólo si el archivo es indexado. Permite indicar el nombre de la clave por la cuál accederemos a los registros del archivo. La variable que indique la clave deber ser alfanumérica y estar especificada en la File Description (FD) del archivo.

RELATIVE KEY IS
Se utiliza sólo si el archivo es RELATIVE. Permite asociar una variable numérica sin signo que indicará el número de registro del archivo. La variable acá indicada debe estar declarada en la WORKING STORAGE SECTION..

ALTERNATE RECORD KEY IS

Es opcional. Se utiliza sólo con archivos indexados. Identifica una o mas claves secundarias para acceder a los registros del archivo.

WITH DUPLICATES

Es opcional. Indica que la clave no es única y que puede haber más de un registro con el mismo valor de campo clave. Por ejemplo, en un archivo de cuentas bancarias si utilizáramos el nombre del titular de la cuenta para realizar una búsqueda, puede pasar que un mismo cliente tenga más de una cuenta abierta en el banco.

OPTIONAL

Le indica al compilador que en caso de que el archivo no exista físicamente al momento de intentar abrirlo, lo cree. Esta cláusula puede ser omitida.

2.2 FILE SECTION

En esta sección debe describirse toda la información referente a los archivos utilizados en el programa. La misma se debe especificar dentro de la DATA DIVISION. El formato de esta sección es el siguiente:

```
      1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123...
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT ARCH ASSIGN TO "C:\DATOS.DAT".
       DATA DIVISION.
       FILE SECTION.
       FD ARCH
           RECORD CONTAINS 132 CHARACTERS.
           LABEL RECORD IS OMMITED
           DATA RECORD  IS REG-ARCH.
       01 REG-ARCHIVO.
           02  CODIGO      PIC X(002).
           02  DESCRIPCION PIC X(130).
```

Ejemplo de la declaración del archivo ARCH

2.2.1 Cláusulas

FILE DESCRIPTION

Es la única cláusula obligatoria, y puede declararse como *FD*. Se utiliza para dar la descripción del formato del archivo lógico declarado mediante la cláusula *SELECT*. Habrá tantos *FD* como *SELECT* hayamos declarado.

RECORD CONTAINS

Es opcional y le indica al compilador cuántos bytes ocupa un registro del archivo. Generalmente se omite esta cláusula, ya que de esta forma el compilador calcula la suma de los tamaños de los campos del registro y ese será el tamaño final del registro físico. En cambio si se indicara un tamaño, el cual debe ser un número entero positivo, éste no puede ser menor a dicha suma. Si el tamaño indicado explícitamente es mayor a la suma de los tamaños de los campos, el compilador llenará el espacio sobrante con espacios en blanco.
La utilidad de especificar un tamaño físico mayor al utilizado por el registro lógico permite que en el futuro se puedan ir agregando campos. Incluso es posible declarar un tamaño variable para el registro de la siguiente manera:

```
RECORD CONTAINS tamaño-mínimo TO tamaño-máximo CHARACTERS
```

LABEL RECORD

Es opcional. Indica al compilador si al momento de crear el archivo físico debe agregar etiquetas especiales al principio y al final del mismo, para identificar correctamente el tipo de archivo que se trata. Es sólo relevante para para los almacenados en disco. Emula las épocas en las cuales se trabajaba con cintas magnéticas y se precisaban registros especiales llamados "de inicio y parada". Puede tomar los valores *OMMITED*, para que no se utilicen las etiquetas, o *STANDARD* para que se generen según el tipo de almacenamiento encontrado.

DATA RECORD

Indica cuál es el nombre lógico del registro para ese archivo. Si esta cláusula se omite, puesto que es opcional, el registro lógico debe hallarse declarado inmediatamente después del párrafo FD. Si se especifica nos da la posibilidad de referenciar más de un registro lógico p para el archivo (separando los identificadores con espacio).

3 SENTENCIAS PARA LA MANIPULACIÓN DE ARCHIVOS

Para acceder a un archivo, los pasos son los siguientes:

1. Especificar la cláusula **SELECT** correspondiente al fichero dentro del párrafo FILE-**CONTROL**.
2. Especificar la cláusula **FD** correspondiente al archivo dentro de la **FILE SECTION**.
3. En la **PROCEDURE DIVISION:**
   - a - Abrirlo utilizando la sentencia **OPEN**.
   - b - Verificar **FILE STATUS**, si es 00 (OK), proseguir.
   - c - Utilizar la sentencia **READ** para leer los datos almacenados.
   - d - Cerrarlo mediante la sentencia **CLOSE**.

OPEN

Esta sentencia es independiente del tipo de archivo que se trate. Permite abrir un archivo para su utilización. Establece una comunicación entre el área de E/S asociada a ese archivo y el dispositivo externo que lo soporta. Existen distintas formas de abrir un archivo ya sea secuencial o indexado:

```
OPEN { INPUT  }
     { OUTPUT } nombre-lógico-del-archivo
     { I-O    }
     { EXTEND }
```

Un archivo que se encuentra abierto, sin importar en qué modo, no puede volver a abrirse sin antes ser cerrado. En caso de realizar esta acción, dará como resultado '42' en la variable de *FILE STATUS* del archivo.

El parámetro **INPUT** indica que el archivo se abrirá como sólo lectura. El puntero se sitúa en el primer registro y si el archivo no existe, se produce un error.

Cuando se especifica la cláusula **OUTPUT**, se abre el fichero especificado para escribir. Si el archivo que se abre ya existe, se destruye su contenido creándose así de nuevo. Si no existe, se crea.
Si se especifica **I-O**, se abre el archivo para leer y escribir. Si no existe, se crea si se especificó la cláusula OPTIONAL en la entrada SELECT correspondiente. Si no la especificamos, se produce un error.

Cuando se especifica el parámetro **EXTEND** significa que se van a agregar datos a un archivo con modo de acceso secuencial. De esta forma podemos añadir mas registros a un archivo existente. Si el archivo no existe, se crea si se especificó la cláusula OPTIONAL en la entrada SELECT correspondiente.

SENTENCIA	ARCHIVOS SEQUENTIAL				ARCHIVOS LINE-SEQUENTIAL			
	MODO DE APERTURA				MODO DE APERTURA			
	INPUT	OUTPUT	I-O	EXTEND	INPUT	OUTPUT	I-O	EXTEND
READ	X		X		X			
WRITE		X		X		X		X
REWRITE			X					

Dependiendo del modo en el cuál abrimos el archivo vamos a tener que utilizar cláusulas y sentencias distintas para manipular sus datos. A continuación una serie de cuadros que resumen la disponibilidad del archivo y las sentencias permitidas en cada caso.

ARCHIVOS INDEXED Y RELATIVE					
ACCESS MODE	SENTENCIA	MODO DE APERTURA			
		Input	Output	I-O	Extend
SEQUENTIAL	READ	X		X	
	WRITE		X		X
	REWRITE			X	
	START	X		X	
	DELETE			X	
RANDOM	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
DYNAMIC	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

DISPONIBILIDAD DE UN ARCHIVO		
MODO DE APERTURA	SI EL ARCHIVO EXISTE	SI EL ARCHIVO NO EXISTE
INPUT	OPEN exitoso	OPEN no exitoso.
INPUT OPTIONAL	OPEN exitoso	OPEN exitoso; la primer lectura causa la condición AT END, o INVALID KEY.
I-O	OPEN exitoso	OPEN no exitoso.
I-O OPTIONAL	OPEN exitoso	OPEN causa la creación del archivo.
OUTPUT	OPEN exitoso, el archivo no contiene registros.	OPEN causa la creación del archivo.
EXTEND	OPEN exitoso	OPEN no exitoso.
EXTEND OPTIONAL	OPEN exitoso	OPEN causa la creación del archivo.

START

Esta sentencia permite iniciar el proceso de acceso a un archivo cuya organización sea *INDEXED* o *RELATIVE* por cualquiera de sus índices (no necesariamente la primaria, como veremos a continuación) y sin necesidad de que el valor inicial del índice elegido exista en el archivo; y además el ACCESS MODE sea DYNAMIC.

La sintaxis general de ***START*** es:

```
START nombre-lógico-del-archivo [KEY IS {condicional} nombre-índice][[NOT] INVALID KEY sentencia]]
```

Si no se especifica ninguna de las cláusulas opcionales, ***START*** no tiene efecto sobre el archivo (excepto si el archivo está vacío, en ese caso causará una excepción).

Funciona de esta manera: se evalua *condicional* contra el valor actual de *nombre-índice*, el cual debe ser cualquiera de los campos declarados como índices en la *SELECT*, sea primario o cualquiera de los secundarios. El puntero del archivo se posiciona en el primer registro que satisface la condición.

Si ningún registro del archivo satisface la condición, se provoca una excepción de entrada-salida análogo a lo ya visto para la sentencia *READ*, de allí que es opcional la cláusula INVALID KEY y su negación.
La expresión *condicional* a la que se hace referencia en la sintaxis, puede ser cualquiera de las siguientes:

LESS (<) menor que.
EQUAL (=) igual a.
NOT EQUAL (¬=) distinto de.
GREATER (>) mayor que.
GREATER OR EQUAL (>=) mayor o igual que.
LESS OR EQUAL (<=) menor o igual que.
FIRST principio de archivo (RM/COBOL).
LAST final del archivo (RM/COBOL).

```
      1        2        3        4        5        6        7
1234567890123456789012345678901234567890123456789012345678901234567890123...
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EJEMPLO.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT OPTIONAL CLIENTES ASSIGN TO "CLIENTES.DAT"
           ORGANIZATION IS INDEXED
           ACCESS MODE DYNAMIC
           RECORD KEY CODIGO-CLIENTE
           ALTERNATE RECORD KEY APELLIDO-NOMBRE WITH DUPLICATES.
           FILE-STATUS IS WSS-FS-CLI
       DATA DIVISION.
       FILE SECTION.
       FD CLIENTES
       01 REG-CLIE.
         02 CODIGO-CLIENTE PIC 9(5).
         02 APELLIDO-NOMBRE.
           03 APELLIDO PIC X(20).
           03 NOMBRE PIC X(20).
       PROCEDURE DIVISION.
       COMIENZO.
           OPEN INPUT CLIENTES.
           MOVE 1 TO CODIGO-CLIENTE.
           START CLIENTES KEY >= CODIGO-CLIENTE.
       LEER.
           PERFORM UNTIL WSS-FS-CLI-EOF
              READ CLIENTES NEXT AT END PERFORM CERRAR
              END-READ
              DISPLAY CODIGO-CLIENTE
              DISPLAY APELLIDO
              DISPLAY NOMBRE
           END-PERFORM.
       CERRAR.
           CLOSE CLIENTES.
       SALIR.
           STOP RUN.
```

Ejemplo de sentencia START

Aquí comenzamos abriendo el archivo. Asignamos valor 1 para el campo índice primario. Ejecutamos un START el cual dice que debe buscarse el primer registro que no sea menor a CODIGO-CLIENTE, o lo que es lo mismo, que no sea menor a 1. En nuestro caso ese registro podría ser cualquiera del archivo excepto el cero, pero el primero que cumple la condición es precisamente el 1. Si no existiera el 1 sería el 2 y así sucesivamente.

En caso de que el archivo esté vacío ningún registro cumple la condición, por lo que se provocará una excepción que en nuestro caso no está controlada con la cláusula INVALID KEY.

Luego procedemos a la lectura del archivo secuencialmente. Esta lectura se hace siguiendo el valor ascendente de la clave primaria puesto que ese campo es el que se especificó en la sentencia START. Así hasta el final del archivo.

Si hubiésemos asignado el valor 3 a CODIGO-CLIENTE el registro tres cumpliría la condición. En nuestro hipotético caso de que no existiera el registro 4 en el archivo y le asignáramos ese valor a CODIGO-CLIENTE antes del START, el registro 5 cumpliría la condición.

START no solo funciona con el índice primario, también sirve con cualquiera de los índices secundarios. En nuestor ejemplo ese índice es el campo APELLIDO-NOMBRE que a su vez está permitido que sea duplicado, es decir que puede haber dos registros cuyos campos APELLIDO-NOMBRE tienen el mismo valor.

El siguiente ejemplo genera un listado del archivo de clientes alfabéticamente por APELLIDO-NOMBRE.

```
      1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123...

      COMIENZO.
          OPEN INPUT CLIENTES.
          MOVE " " TO APELLIDO-NOMBRE.
          START CLIENTES KEY NOT < APELLIDO-NOMBRE
          INVALID KEY PERFORM CERRAR.
      LEER.
          PERFORM UNTIL WSS-FS-CLI-EOF
             READ CLIENTES NEXT AT END PERFORM CERRAR
             END-READ
             DISPLAY CODIGO-CLIENTE.
             DISPLAY APELLIDO.
             DISPLAY NOMBRE.
          END-PERFORM
      CERRAR.
          CLOSE CLIENTES.
      SALIR.
          STOP RUN.
```

Ejemplo START por índice secundario

Aquí lo que hacemos es asignar un espacio vacío al campo APELLIDO-NOMBRE y en el START usar ese campo como *nombre-índice*. Lo que provoca que el archivo sea mostrado en pantalla ordenado alfabéticamente por ese campo, ya que el espacio es menor a la letra A. Incluso si se da el caso de que APELLIDO-NOMBRE esté vacío en alguno de los registros, ése será el primero en ser listado.

Variando la condición y el valor asignado para el campo índice se puede listar el archivo de diferente manera. Y también se puede crear una condición que no puede ser satisfecha por ningún registro. Por ejemplo quitando la palabra NOT en el ejemplo podemos ver que no existe y no puede existir un registro que sea menor a espacio. Así mismo he agregado la cláusula INVALID KEY para el caso de que el archivo esté vacío (un archivo sin registros no satisface ninguna condición en START).

CLOSE

Luego de trabajar con un archivo hay que cerrarlo. La utilidad de esta sentencia está relacionada con las actualizaciones necesarias que debe efectuar el runtime sobre el archivo en cuestión, además de liberar todos los buffers y apuntadores que creó para poder manipular el archivo. Si bien el compilador (mejor dicho, el runtime) lleva la cuenta de los archivos abiertos por el programa y los cierra automáticamente luego que se abandona la ejecución del mismo, no hay que confiar esta tarea al compilador.

No se recomienda mantener archivos abiertos que ya no se van a usar, como tampoco abrir archivos que se sabe no van a ser usados. Esto se debe al modo en que COBOL mantiene los índices de los archivos de datos, si se tiene un archivo (o muchos) abierto y ocurre un imprevisto, como un corte de electricidad, el arbol de índices de los archivos abiertos podría dañarse, a veces irreparablemente (RM/COBOL tiene una utilidad para reparar índices pero no es por completo efectiva).

Por otro lado el propio sistema operativo puede tener un límite en la cantidad de archivos abiertos que un programa cualquiera puede mantener (en MS-DOS esto se especifica en el CONFIG.SYS mediante la directiva FILES).

La sintaxis de la sentencia CLOSE es sencilla:

```
CLOSE nombre-lógico-del-archivo.
```

Hay cuatro operaciones básicas que se pueden realizar sobre un archivo: Leer, escribir, sobreescribir y borrar. Todas referidas a registros individuales. El comportamiento de cada una de estas operaciones en COBOL está ligado al tipo de organización del archivo.

READ

Al ejecutar esta sentencia, se deja disponible un registro lógico en el área de entrada para ser procesado. De esta forma podremos "leer" un registro del archivo siempre que no haya sido abierto en modo OUTPUT.

Dependiendo del tipo de acceso, se modifica su sintaxis.

ACCESS MODE SEQUENTIAL/DYNAMIC

Estos tipos de acceso pueden ser utilizados en archivos ***secuenciales*** o ***indexados***. Ésta es su sintaxis:

```
READ nombre-lógico-del-archivo [NEXT/PREVIOUS [INTO id-registro-WS[AT END sentencia]]].
```

Lo que se indica entre corchetes es opcional.

Si se omite **NEXT** y **PREVIOUS**, se asume ***NEXT***. Eso significa que por cada lectura el puntero avanza al siguiente registro, si indicáramos ***PREVIOUS*** el puntero retrocedería al registro anterior. El orden está dado por el índice empleado para hacer la lectura en el caso de que el *ACCESS MODE* sea *DYNAMIC,* o físicamente en el caso que sea *SEQUENTIAL*.

Si se especifica la opción **INTO**, la sentencia **READ** ademas de dejar disponible un registro lógico en el área de entrada, es decir en el registro de entrada definido en la *FD*, copia el contenido de este registro en el área de datos especificada en la cláusula.

La cláusula **AT END** sirve para indicar qué debe hacerse cuando se encuentra el final del archivo (o el principio para lecturas con *PREVIOUS*) por lo tanto debe ser seguida de una sentencia imperativa (no admite estructuras de decisión). Puede especificarse la cláusula **NOT AT END**, pero en general ninguna de las dos se utiliza. A continuación un ejemplo:

```
READ CLIENTES AT END MOVE "SI" TO ULT-REG.
```

Ejemplo de cláusula *AT END*

ACCESS MODE RANDOM

Para archivos cuyo modo de acceso es ***RANDOM*** la sintaxis utilizada es la siguiente:

```
READ NOMBRE-LÓGICO-DEL-ARCHIVO [INTO IDENTIFICADOR [NOT] INVALID KEY sentencia]].
```

En este caso, el acceso a los registros del archivo indexado se hace en orden ascendente de clave, la cual viene dada por el campo descrito a continuación de la cláusula *RECORD KEY.*

```
          1         2         3         4         5         6         7
7 123456 8901234567890123456789012345678901234567890123456789012345678901234567890123...

         MOVE MOD-COD-CLIEN    TO CTE-COD-CLIEN
         MOVE "SI"             TO ENCONTRADO
         READ CLIENTES INVALID KEY MOVE "NO" TO ENCONTRADO
```

Ejemplo de lectura de archivos secuenciales indexados

En este ejemplo se observa que antes de efecturar la lectura, se carga en el campo clave *CTE-COD-CLIEN* el valor de la clave del registro que se quiere leer.

Cuando se ejecuta una sentencia de este tipo, el contenido del campo especificado como clave se compara con los valores contenidos en los correspondientes campos de los registros del archivo hasta que se encuentre un valor igual, en cuyo caso se recupera el registro y se ejecuta la sentencia que hayamos codificado en la cláusula *NOT INVALID KEY*, o la línea de código consecuente a la lectura. Si no se encuentra un valor igual al de la clave dada, se ejecuta la sentencia descrita a continuación de la frase *INVALID KEY*, si ha sido especificada dicha opción.

No hemos desarrollado en nuestros ejemplos la opción ***WITH [NO] LOCK***, dado que no se utiliza con frecuencia. En un archivo abierto en modo I-O, a menos que se especifique *WITH NO LOCK*, cada registro leído es exclusivo, no compartido, hasta que se ejecute otra sentencia para este archivo.

WRITE

Cuando se ejecuta esta sentencia se efectúa una operación de salida o de escritura de un registro lógico con destino a un archivo de salida. Esta transferencia de información se hace a través del área de memoria asignada al archivo, aunque esta operación es transparente al usuario.

ARCHIVOS SEQUENTIAL/LINE-SEQUENTIAL

La sintaxis a utilizar es la siguiente:

```
WRITE NOMBRE-REGISTRO-FD [FROM IDENTIFICADOR]
```

El archivo asociado debe abrirse en modo *OUTPUT* o *EXTEND*.

Si no se especifica la opción FROM, se debe mover la información que se desea almacenar al campo referenciado por *NOMBRE-REGISTRO-FD*. A continuación se efectúa la operación de escribir.

```
          1         2         3         4         5         6         7
7 123456 8901234567890123456789012345678901234567890123456789012345678901234567890123...

         MOVE IDENTIFICADOR TO NOMBRE-REGISTRO-FD
         WRITE NOMBRE-REGISTRO-FD
```

Ejemplo de la sentencia WRITE

La sintaxis a utilizar es la siguiente:

```
WRITE NOMBRE-REGISTRO-FD [FROM IDENTIFICADOR][[NOT]INVALID KEY SENTENCIA][END-WRITE]
```

Al igual que en la lectura, primero debemos mover un valor a la clave principal del archivo. Dicho valor debe ser único en todo el archivo.

IDENTIFICADOR puede ser el nombre de una entrada en la *Working Storage Section* o en la *Linkage Section*, el nombre de un registro descriptivo *FD* de otro archivo previamente abierto o el nombre de un identificador de una función alfanumérica.

Si el modo de acceso es ***SEQUENTIAL***, los registros deben ser emitidos al sistema en orden ascendente de la clave principal.

Si el modo de acceso es ***RANDOM*** o ***DYNAMIC***, los registros pueden ser emitidos al sistema en cualquier orden.

Si la opción ***INVALID KEY*** se ha especificado, *SENTENCIA* se ejecutará en los siguientes casos:

- Cuando se abre un archivo como *OUTPUT* en modo de acceso SEQUENTIAL y la clave del registro no es mayor que la del registro anterior.
- Cuando el archivo se ha abierto en modo *OUTPUT* o *I-O* y la clave del registro a escribir ya existe en el archivo.
- Cuando el archivo se ha abierto en modo *OUTPUT* o *I-O* y la clave alternativa del registro a escribir ya existe en el archivo.
- Cuando el disco está lleno.

END-WRITE puede ser utilizada solamente si se especifica la frase ***INVALID-KEY***.

REWRITE

Esta sentencia permite modificar lógicamente un registro existente en un archivo de acceso directo. Para poder ejecutarla, el archivo de acceso directo debe estar abierto en modo ***I-O***. Utiliza los mismos parámetros que *WRITE*.

Los archivos *LINE-SEQUENTIAL* no pueden ser reescritos.

La longitud del nuevo registro debe ser la misma que la del registro a reemplazar, aunque no pueden compartir el mismo área de memoria.

ARCHIVOS SEQUENTIAL

Para modificar un registro de un archivo secuencial, primero debemos posicionarnos sobre el mismo utilizando la sentencia *READ*. Luego, al ejecutar *REWRITE* modificaremos el último registro leído.

```
REWRITE NOMBRE-REGISTRO-FD [FROM ID-REGISTRO-WS]
```

ARCHIVOS INDEXED/RELATIVE

```
REWRITE NOMBRE-REGISTRO-FD [FROM IDENTIFICADOR][[NOT]INVALID KEY SENTENCIA][END-REWRITE]
```

Las restricciones son las mismas que las de un ***READ*** en modo de acceso ***RANDOM***, es decir, hay que proveer una clave primaria y ésta debe existir en el archivo. No se puede *reescribir* un registro que no existe en el archivo.

Si se ha especificado la cláusula *INVALID KEY* se usará cuando el valor contenido en el campo de índice primario del registro que se desea reemplazar no exista en el archivo, o bien cuando el valor del campo que referencia cualquiera de los índices secundarios ya exista en el archivo y no se haya especificado *WITH DUPLICATES*.

Si se utiliza la frase *INVALID KEY* al operar con un archivo secuencial, se puede generar una EXCEPTION/ERROR procedural.

Esta sentencia se puede usar con todos los modos de accesos siempre que el archivo no haya sido abierto en modo *INPUT*.

DELETE
Finalmente tenemos la sentencia que elimina un registro del archivo, la cual se puede usar en un archivo abierto en modo ***I-O*** o ***OUTPUT***.

Antes de que se ejecute una sentencia ***DELETE***, es necesario haber cargado en el campo descrito a continuación de la cláusula ***RECORD KEY***, el valor de la clave del registro al que se quiere acceder.

Para archivos en los que el modo de acceso es ***SEQUENTIAL***, la última sentencia ejecutada antes de la ejecución de la sentencia ***DELETE*** debe ser la sentencia ***READ***, la cual localiza el registro a borrar por el valor contenido en el campo que hace referencia a la clave principal.

Para archivos en los que el modo de acceso es ***RANDOM*** o ***DYNAMIC***, el registro a borrar viene dado por el contenido del campo que hace referencia a la clave principal. No es necesario ejecutar previamente una sentencia ***READ***.

Cuando se ejecuta la sentencia ***DELETE***, el contenido del campo especificado como clave, es comparado con los valores contenidos en los correspondientes campos de los registros del archivo, hasta que se encuentre un valor igual, en cuyo caso se borra el registro y se ejecuta la *sentencia-2* si la frase ***NOT INVALID KEY*** ha sido especificada. Si no se encuentra un valor igual a la clave dada, entonces se ejecuta la *sentencia-1* descrita a continuación de la frase ***INVALID KEY***, si esta ha sido especificada.

La frase ***INVALID KEY*** no puede ser especificada con una sentencia ***DELETE*** que haga referencia a un archivo con modo de acceso secuencial.

END-DELETE puede ser utilizada solamente si se especifica la frase ***INVALID KEY***.

Esta es su sintaxis:

```
DELETE NOMBRE-LÓGICO-DEL-ARCHIVO [[NOT] INVALID KEY sentencia].
```

DELETE se usa también para borrar físicamente un archivo, que previamente debe haber sido cerrado. La sintaxis para realizar esta acción es la siguiente:

```
DELETE FILE NOMBRE-LÓGICO-DEL-ARCHIVO.
```