MÓDULO 2 | PROGRAMACION COBOL– Estructuras Avanzadas: TABLAS

Contenido

1 ESTRUCTURAS DE DATOS: TABLAS

Una de las estructuras de datos más utilizadas en la programación son los ***arrays*** o ***tablas***. También se conoce como ***Vectores*** (a las tablas o arrays de una dimensión), como *Matrices* (a las tablas o arrays de dos dimensiones), como ***Poliedros*** (a las tablas o arrays de tres o más dimensiones).

1.1 CONCEPTOS BASICOS DE TABLAS

Tabla: Estructura de datos constituida por un número fijo de elementos, todos ellos del mismo tipo y ubicados en direcciones de memoria físicamente contiguas.

1.1.1 CARACTERÍSTICAS DE UNA TABLA

- es una estructura de datos definida en la **memoria principal** (**no** es un fichero/archivo).
- tiene un nombre que identifica a toda la tabla, para acceder a todos los elementos de la tabla a la vez.
- todos y cada uno de los elementos de una tabla tienen **el** mismo nombre**.**
- todos y cada uno de los elementos de una tabla son del **mismo tipo** y su definición es identica.
- por esto último, (todos los elementos se llaman igual) es imprescindible utilizar **un índice** o **apuntador** para poder referirnos e identificar a cualquier elemento individualmente
- al acceder al contenido de una tabla, el índice siempre debe tener un valor
- comprendido entre **1 y el número máximo de elementos** que contiene**.**

Elemento: Cada uno de los datos consecutivos que forman parte de la tabla.

Nombre de la Tabla: Es el identificador usado para referenciar la tabla y de forma global a todos los elementos que la forman.

Tipo de una Tabla: Marca el tipo de dato básico que es común a todos y cada uno de los elementos o componentes que forman dicha estructura (entero, real, carácter o lógico).

Índices: Valor numérico entero y positivo a través del cual podemos acceder directa e individualmente a los distintos elementos o componentes que la forman, marca la posición relativa de cada elemento dentro de la misma.

Tamaño de la Tabla: Longitud o número máximo de elementos que la forman.

Acceso a los elementos o componentes de una Tabla: Los elementos de una tabla reciben el mismo trato que cualquier otra variables simple, con un tipo de datos que coincide con el tipo de la tabla. Para acceder o referenciar a un elemento en particular es suficiente con indicar el nombre de la tabla seguido del índice correspondiente entre paréntesis.

Dimensión de la Tabla: Viene definida por el número de índices que necesitamos para acceder a cualquiera de los elementos que forman su estructura.

1.1.2 DEFINICIÓN DE UNA TABLA

Definir una tabla en MEMORIA PRINCIPAL implica establecer:

- NOMBRE SIMBÓLICO DE LA TABLA (para referencia global)
- NÚMERO DE ELEMENTOS QUE CONTIENE
- NOMBRE SIMBÓLICO DEL ELEMENTO (igual para todos).
- DEFINIR LA ESTRUCTURA DEL ELEMENTO (la misma para todos).
    - LOS CAMPOS EN QUE SE SUBDIVIDE.
    - LOS ATRIBUTOS Y LONGITUD DE CADA CAMPO.

1.1.3 BENEFICIOS DE LAS TABLAS

Es muy normal, en programación, copiar el contenido de ficheros en tablas cargadas en memoria Principal. Las ventajas más importantes que reporta cargar un fichero en una tabla para su posterior proceso son:

- el **acceso** a los datos de una tabla es **rapidísimo** en comparación a los datos en un fichero.
- **sólo** tenemos **abierto** el fichero durante la carga. de esta forma s**e permite el acceso** al fichero a otros usuarios y procesos.

1.2 CLASIFICACIÓN DE TABLAS

Según la estructura del elemento se clasifican en:

Simples: el elemento no está subdividido en campos**.**
Compuestas: el elemento está formado por varios campos.

Según su dimensión se clasifican en:

Unidimensionales: Se necesita un índice para acceder al elemento.
Bidimensionales: Se necesitan dos índices para acceder al elemento.
Tridimensionales: Se necesitan tres índices para acceder al elemento.
Multidimensionales: Se múltiples índices para acceder al elemento.

Según los valores de los elementos:
Ordenadas: Sus elementos están ordenados.
Desordenadas: Sus elementos están desordenados.

Según el número de elementos que tienen contenido:

Completa**:** Todos los elementos tiene contenido.
Incompleta: Algunos elementos están vacíos.

1.2.1 TABLAS UNIDIMENSIONALES

También se les llama **vectores.** Los elementos se almacenan en posiciones contiguas adyacentes en la memoria principal de un ordenador.
Los elementos se almacenan en posiciones contiguas adyacentes en la memoria principal de un ordenador.

Ejemplos

TABLA-UNIDIMENSIONAL			
ELEM(1)	ELEM(2)	ELEM(3)	ELEM(4)

Otra forma típica de representarla gráficamente una tabla.
Tabla-Numeros contenidos) =>

Num(1)	Num(2)	Num(3)	Num(4)	Num(5)	Num(6)
8	14	26	38	110	212

Acceso individual a los elementos de la tabla:
Es suficiente su nombre seguido de **1 índice**.

Nombre-Elemento(índice)
Nombre-Tabla(índice) Válido en algunos lenguajes, en COBOL no.

Acceso a los elementos de la Tabla-Numeros (un solo índice):
Accediendo al elemento **Num(1)** tomaríamos el valor que contiene **8**
Accediendo al elemento **Num(2)** tomaríamos el valor que contiene **14**
Accediendo al elemento **Num(3)** tomaríamos el valor que contiene **26**
Accediendo al elemento **Num(4)** tomaríamos el valor que contiene **38**
Accediendo al elemento **Num(5)** tomaríamos el valor que contiene **110**
Accediendo al elemento **Num(6)** tomaríamos el valor que contiene **212**

NOTA: *Algunos lenguajes consideran que el primer elemento se direcciona con valor de índice 0 (cero) y otros con valor de índice 1. PARA NOSOTROS EL VALOR DEL PRIMER ÍNDICE SERÁ EL VALOR 1.*

1.2.2 TABLAS UNIDIMENSIONALES COMPUESTAS

También en estas tablas todos los elementos son iguales tanto en tipo como en nombre. Pero en una tabla compuesta, cada uno de los elementos de la tabla se subdivide en varios campos distintos, tanto en sus nombres como en sus definiciones, a los que se puede acceder por su nombre particular y un índice.
Representación gráfica

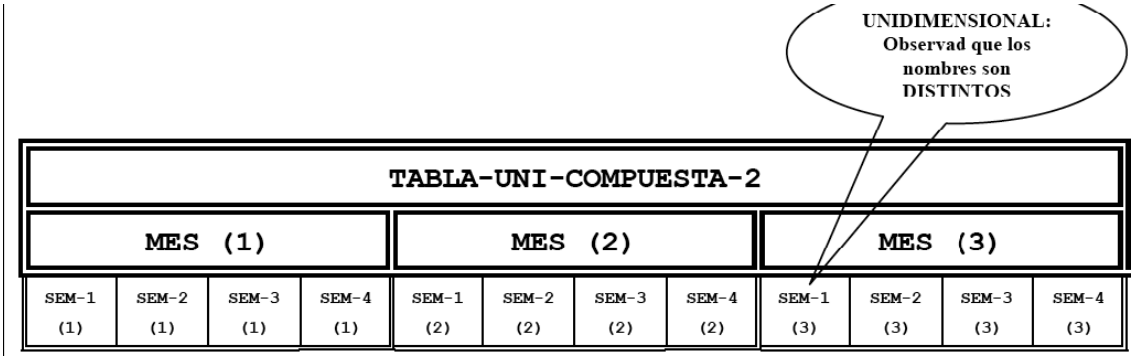
Ejemplo 1:

TABLA-UNI-COMPUESTA-1									
ELEMEN(1)		ELEMEN(2)		ELEMEN(3)		ELEMEN(4)		ELEMEN(5)	
COD(1)	CAN(1)	COD(2)	CAN(2)	COD(3)	CAN(3)	COD(4)	CAN(4)	COD(5)	CAN(5)

Acceso a sus elementos de la Tabla-Uni-Compuesta-1 (un solo índice):

- Accediendo a **COD(2)** tomaríamos el valor de esta variable.
- Accediendo a **CAN(5)** tomaríamos el valor de esta variable.
- Accediendo a **ELEMEN(4)** tomaríamos el valor de COD(4) y de CAN(4) simultáneamente.

OJO: Los accesos a campos compuestos siempre son gestionados por el sistema como alfanumérico aunque cada variable individual fuese numérica.



Acceso a sus elementos de la Tabla-Uni-Compuesta-2 (un solo índice):

- Con **SEM-4(1)** accedemos al valor de la semana 4 del mes 1.
- Con **SEM-1(3)** accedemos al valor de la semana 1 del mes 3.
- Con **MES(2)** accedemos a la vez a las 4 semanas del mes 2.

OJO: Los accesos a campos compuestos siempre son gestionados por el sistema como alfanumérico aunque cada variable individual fuese numérica.

1.2.3 TABLAS BIDIMENSIONALES

Cada elemento o alguno de sus campos es, a su vez, una tabla. También se les llama **matrices.**

Representación gráfica

Indices	Colum. 1	Colum. 2	Colum. 3	Colum. 4	Colum. ...	Colum. C
Fila 1	elem(1, 1)	elem(1, 2)	elem(1, 3)	elem(1, 1)	elem(1, ...)	elem(1, C)
Fila 2	elem(2, 1)	elem(2, 2)	elem(2, 3)	elem(2, 1)	elem(2, ...)	elem(2, C)
Fila ...	elem(..., 1)	elem(... , 2)	elem(... , 3)	elem(..., 1)	elem(..., ...)	elem(... , C)
Fila F	elem(F, 1)	elem(F, 2)	elem(F, 3)	elem(F, 1)	elem(F, ...)	elem(F, C)

Acceso individual a los elementos

Para acceder a uno de los elementos de esta tabla bidiemnsional es necesario utilizar **2 índices**: el 1º marca la fila, el 2º la columna.

Nombre-tabla (índice-fila, índice-columna)

Ejemplo: La tabla se llama **NOTAS**:

10 COLUMNAS > Alumnos del 1 al 10
04 FILAS > ASIGNATURAS (Matemáticas, Física, Historia y Filosofía)

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
F1	9	4	8	10	7	2	5	5	5	9
F2	8	10	7	5	9	4	8	10	0	1
F3	4	8	10	0	1	8	10	7	5	9
F4	7	5	9	4	8	10	0	5	5	5

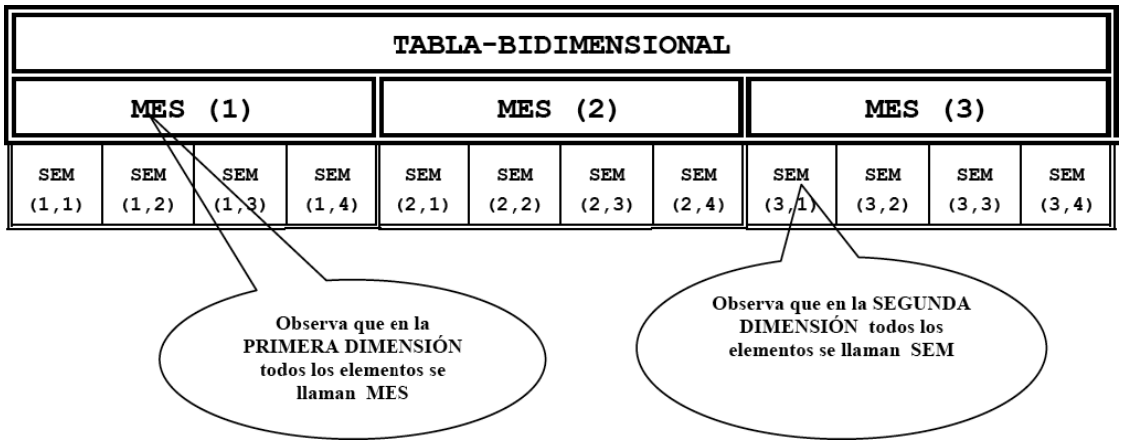
Acceso a los elementos de la tabla Notas (dos índices):

- Accediendo al elemento **Notas(4, 2)** tomaríamos el valor **10**
  (Nota en **Filosofía** del **alumno 2**)
- Accediendo al elemento **Notas(3, 8)** tomaríamos el valor **7**
  (Nota en **Historia** del **alumno 8**)
- Accediendo al elemento **Notas(2, 10)** tomaríamos el valor **1**
  (Nota en **Física** del **alumno 10**)

OJO: En este ejemplo los índices de fila y de columna nunca deben ser mayores de 4 y 10.

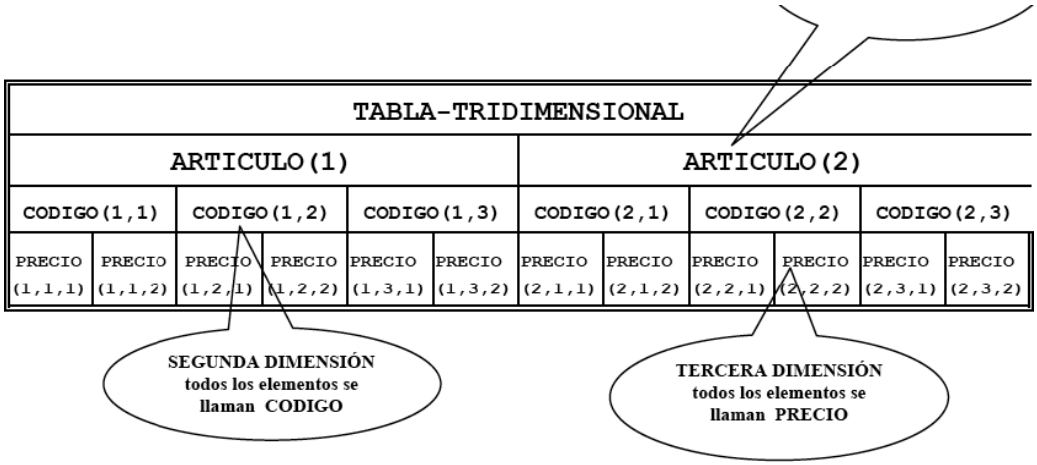
1.2.3.1 EJEMPLO DE TABLAS BIDIMENSIONALES

Es una tabla con información de 3 meses y cada mes es una nueva tabla con información de los gastos acumulados de sus 4 semanas.



1.2.4 TABLAS TRIDIMENSIONALES:

Es una tabla con información sobre 2 artículos. Cada articulo se subdivide en otra tabla con 3 códigos y cada código tiene dos precios posibles.



2 Cláusula OCCURS

Esta cláusula permite definir una estructura de datos denominada **vector**, la misma no puede ser realizada en una descripción con nivel 01, 77, 66 u 88, así como tampoco se puede utilizar la cláusula *VALUE* para inicializar al vector con un determinado valor.
Para ver la sintaxis de esta cláusula nos valdremos del siguiente ejemplo en el cual el campo *DIA-SEMANA*, alfanumérico de 10 posiciones, se repite 7 veces:

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

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 SEMANA.
        02 DIA-SEMANA PIC X(10) OCCURS 7 TIMES.
```

Ejemplo 1 – Definición de OCCURS

Tambien puede utilizarse para repetir estructuras de datos mas amplias, como en el siguiente ejemplo donde un vector con 20 elementos contiene el grupo de campos DATOS-CLIENTE. Es decir que esta estructura podrá almacenar la información de 20 clientes distintos.

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

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 CLIENTES.
           02 DATOS-CLIENTE OCCURS 20 TIMES.
            03 NOMBRE    PIC X(15).
            03 APELLIDO  PIC X(15).
            03 DNI       PIC X(10).
            03 CUENTA.
             04 ENTIDAD  PIC 9(3).
             04 FILLER   PIX X(1) VALUE '-'.
             04 PRODUCTO PIC 9(2).
             04 FILLER   PIX X(1) VALUE '-'.
             04 N-CUENTA PIC 9(3).
             04 FILLER   PIX X(1) VALUE '-'.
             04 DIGVER   PIC 9(01).
```

Ejemplo 2 – Definición de OCCURS

3 Cláusula DEPENDING

Permite definir un vector de longitud variable. La sintaxis es la siguiente: OCCURS **n** TIMES ***DEPENDING*** ON identificador-1
En este caso la cláusula *OCCURS* determina que COBOL reservará memoria suficiente para almacenar ***n*** veces el elemento declarado, pero la cantidad de repeticiones de ese elemento estará determinada por ***identificador-1***. La variable **identificador** debe estar declarada en la Working Storage para poder ser utilizada en la declaración del vector.

En el siguiente ejemplo se define el vector *DATOS-ALUMNOS* que puede almacenar hasta 50 elementos, aunque se encuentra limitado por la variable *NRO-ALUMN,* cuyo valor inicial es 5.

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

         DATA DIVISION.
         WORKING-STORAGE SECTION.
         77 NRO-ALUM       PIC 9(2) VALUE 5.
         01 ALUMNOS.
             02 DATOS-ALUMNOS.
                OCCURS 50 TIMES DEPENDING ON NRO-ALUM.
              03 NOMBRE    PIC X(15).
              03 APELLIDO  PIC X(15).
              03 PROMEDIO  PIC 9(2).
```

Ejemplo 3 – Definición de DEPENDING

4 Cláusulas ASCENDING/DESCENDING y SORT

Estás cláusulas permiten indicar si los elementos del vector están ordenados en forma ascendente o descendente por uno o más campos que formen parte de la estructura. Esta es la sintaxis:

OCCURS **n** TIMES
[{ASCENDING / DESCENDING} KEY IS identificador-1[identificador-2]…]

Los elementos van a estar ordenados por ***identificador-1*** ya sea en forma ascendente o descendente, según la cláusula que se utilice. Luego de cargar los datos se debe ejecutar la sentencia ***SORT*** para ordenar el vector.

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

         DATA DIVISION.
         WORKING-STORAGE SECTION.
         01 ALUMNOS.
          02 DATOS-ALUMNOS.
             OCCURS 50 TIMES ASCENDING KEY IS APELLIDO.
           03 NOMBRE    PIC X(15).
           03 APELLIDO  PIC X(15).
           03 PROMEDIO  PIC 9(2).
 (...)
         PROC DURE DIVISION
 (...)
              SORT DATOS-ALUMNOS ASCENDING APELLIDO
```

Ejemplo 4 – Definición de DEPENDING

5 Cláusula INDEXED BY

Se utiliza para especificar el nombre del índice o los índices cuando en una tabla ordenada se realiza una búsqueda binaria, el compilador genera internamente un campo de datos binario que hace referencia a un único elemento de una tabla. Si el índice es secundario el valor puede repetirse en distintos registros.

Se declara a continuación de la cláusula *OCCURS*, su sintaxis es:
INDEXED BY *índice-1* [*índice-2*]...