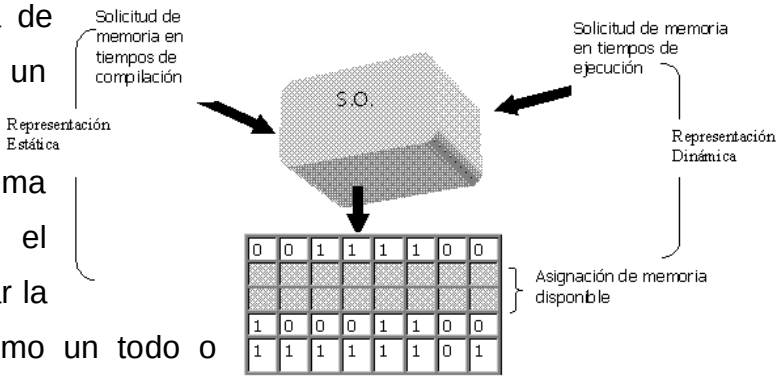
INTRODUCCION



En programación, una estructura de datos es una forma de organizar un conjunto de datos elementales (un dato elemental es la mínima información que se tiene en el sistema) con el objetivo de facilitar la manipulación de estos datos como un todo o individualmente.

Una estructura de datos define la organización e interrelacionamiento de estos, y un conjunto de operaciones que se pueden realizar sobre él. Las operaciones básicas son:

- Alta, adicionar un nuevo valor a la estructura.
- Baja, borrar un valor de la estructura.
- Búsqueda, encontrar un determinado valor en la estructura para realizar una operación con este valor, en forma SECUENCIAL o BINARIO (siempre y cuando los datos estén ordenados)...

Otras operaciones que se pueden realizar son:

- Ordenamiento, de los elementos pertenecientes a la estructura.
- Apareo, dadas dos estructuras originar una nueva ordenada y que contenga a las apareadas.

Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De esta forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

FUNDAMENTOS DE ESTRUCTURA DE DATOS

DEFINICIÓN DE BIT, BYTE, CARÁCTER Y PALABRA

Bit: es una síntesis de dos términos en inglés: Binary digit, que en español significan dígito binario, o lo que es lo mismo, número (dígito) con dos posibles valores (binario). El término surge de usar las dos primeras letras de **Bi**nary con la última de digi**t**.: bit. Es la unidad de información más sencilla posible en el sistema binario.

Byte: Unidad de información que consta de 8 bits equivalente a un único caracter, como una letra, número o signo de puntuación.

Caracter: Es un elemento tomado de un conjunto de símbolos. Un ejemplo de un conjunto de símbolos es {0,1,2,3,4,5,6,7,8,9,A,B,C....Y,z,¡,-,+,*} en el cual se incluyen dígitos, los caracteres del alfabeto y algunos caracteres especiales. Un compilador de lenguaje reconoce un conjunto particular de caracteres.

Palabra: Conjunto de bits que, como unidad elemental, puede manipular una computadora. La longitud en bits de una palabra en una computadora puede ser de 8, 16, 32, etc., y depende del microprocesador de su unidad central de proceso.

MANEJO Y OPERACIONES DE BITS

Ahora el ser humano digitaliza su entorno. Pero, ¿qué significa digitalizar? Digitalizar es traducir información como textos, imágenes o sonidos, a un formato que puedan entender los microprocesadores, y éstos sólo están capacitados para manejar los valores unos y ceros. En efecto, para tu microprocesador todo lo que ves en estos momentos en la pantalla se maneja con unos o ceros. Esto es porque la computadora maneja un sistema binario, que se llama así porque sólo acepta dos valores (0 y 1).

Tal sencillez tiene su razón de ser: los microprocesadores son circuitos electrónicos plasmados en un material llamado silicio (algo parecido al vidrio) que procesan diminutos impulsos eléctricos, el más pequeño de los cuales es conocido por el nombre

de **bit**. Como impulso eléctrico, el microprocesador sólo puede detectar cuando un bit tiene carga eléctrica --su valor sería, en este caso, 1-- o cuando no la tienen --su valor sería 0 - En este ejemplo manejamos los valores unos y ceros de manera un tanto arbitraria, ya que la presencia o ausencia de carga eléctrica en un bit puede ser interpretada como una gran diversidad de valores: cierto y falso, hombre o mujer, T o J, etc.

La eficacia de las computadoras no se basa en la complejidad de su fundamento lógico, que como vimos se reduce a manejar dos posibles valores, sino de la velocidad con la que se aplica dicha lógica: los microprocesadores actuales pueden procesar varios millones de bits en un sólo segundo.

Un bit puede representar solamente dos valores. Dos bits, cuatro posibles valores y ocho bits 256 posibles combinaciones de unos y ceros.

Una unidad de medida muy utilizada en la informática es el **byte**, que consiste en la agrupación de **ocho** bits.

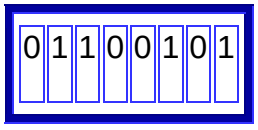
Ejemplo de combinaciones posibles por número de bits

Posibles combinaciones de unos y ceros usando dos bits 4: 00, 01, 11, 10 Posibles combinaciones de unos y ceros usando ocho bits 256:

00000000, 00000001, 00000011, 00000111 […] 11111111

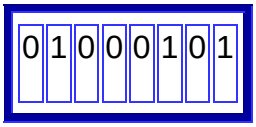
Usando grupos de 8 bits (es decir, bytes) es posible representar a todos los caracteres que conforman el abecedario, incluyendo las mayúsculas y los signos especiales, como el de moneda o los acentos, de tal suerte que cuando se oprime la "e" en el teclado, el microprocesador recibe un paquete de 8 bits con la siguiente combinación de valores:

Valor de la letra "e" minúscula en bits:



Pero si en cambio se presiona la misma tecla en mayúsculas, el paquete de bits que se estará mandando al microprocesador sería el siguiente:

Valor de la letra "E" mayúscula en bits:



Mediante combinaciones de bits y bytes es posible representar una cantidad infinita de cosas: desde bibliotecas completas hasta juegos y películas, todo un universo de información que puede estar en diversas formas; textos, imágenes y sonidos.

Se dice que algunas computadoras tienen aritmética decimal en lugar de binaria. Cuatro bits proporcionan 16 combinaciones, utilizadas para los 10 dígitos de 0 a 9, con 6 combinaciones sin utilizar. El número 1944 se muestra abajo codificado en decimal y en binario, utilizando 16 bits en cada ejemplo:

Decimal: 0001 1001 0100 0100 binario: 0000011110011000

MANEJO Y OPERACIONES CON CARACTER

Como es bien conocido cualquier tipo de información no siempre es interpretada numéricamente. Elementos tales como nombres, cargos y direcciones deben ser también representados en alguna forma en un computador. Este tipo de información es generalmente representado e forma de hileras (strings) de caracteres. Por ejemplo, en algunos computadores los 8 bits 00100110 son utilizados para representar el carácter ‘&’. Un patrón diferente de 8 bits se utiliza para representar el carácter ‘A’,otro para establecer ‘B’.

Si se utilizan 8 bits para representar un carácter, se pueden representar hasta 256 caracteres diferentes, puesto que existen 256 patrones o combinaciones diferentes de 8 bits.

Al igual que en el caso de enteros, no existe nada intrínseco con respecto a alguna hilera de bits en particular que la haga apropiada para representar un carácter

específico. La asignación de as hileras de bits o caracteres pueden ser enteramente arbitraria, pero debe ser consistente.

En efecto, los computadores difieren con respecto al número de bits utilizados para representar un carácter. Algunos computadores utilizan 7 bits (y por consiguiente solo es posible tener hasta 128 caracteres), algunos utilizan 8 (hasta 256 caracteres) y otros utilizan 10 (hasta 1 024 posibles caracteres). El número de bits necesarios para representar un carácter en un computador en particular es denominado el **tamaño del byte** y el grupo de bits correspondientes se denomina un **byte.**

Un ejemplo de una estructura de datos construida a partir de una estructura de datos primitiva es la cadena la cual es una secuencia finita de símbolos tomados de un conjunto de caracteres. El conjunto de caracteres que se emplea para generar cadenas se llama alfabeto. El conjunto de cadenas que se puede derivar del alfabeto A = {C,D,1} incluye los siguientes: 'CD1', 'CD', '1D111', y así sucesivamente, incluyendo la cadena nula o vacía. Por lo general, el inicio y final de una cadena lo delimitamos por comillas.

Las cadenas son un tipo importante de dato que se usan ampliamente. En primera instancia, las cadenas son el medio básico para escribir programas y transmitirlos a la computadora. Segundo, son el medio principal de intercambio de información con los usuarios. Tercero, las cadenas se usan para almacenar información en archivos. Cuarto, se usan lenguajes de programación para nombres de variables, etiquetas y procedimientos. Y en un contexto más general, son una vía de comunicación entre los seres humanos.

Al conjunto de todas las posibles cadenas que se pueden derivar de un alfabeto se le llama vocabulario V, el cual se deriva de un alfabeto A y se denota algunas veces como V_A=A. Un alfabeto no sólo contiene letras del alfabeto {a,b,c.....x,y,z}; también contiene cualquier símbolo válido. Si el alfabeto es {0,1} entonces las cadenas que se obtienen se llaman comúnmente cadenas de bits.

Cada cadena tiene un atributo llamado longitud, el cual es el número de caracteres en la cadena.

Las operaciones definidas sobre las cadenas sondiferentes a las definidas para los enteros. Las tres operaciones principales sobre cadenas son:

- Longitud
- Concatenación
- Subcadena

LONGITUD DE CADENA:

El operador de longitud da el número de caracteres de una cadena. Esta tiene un operando de tipo cadena y su resultado es de tipo entero.

Concatenación de cadenas:

La operación de concatenación se efectúa sobre un par de cadenas, juntándolas de extremo a extremo en una nueva cadena. El operador de concatenación tiene dos operandos, ambos de tipo cadena y produce un resultado de tipo cadena.

SUBCADENAS:

La operación subcadena tiene como único operando una cadena de la cual genera una nueva cadena como resultado. Para especificar completamente la operación de subcadena, deben especificarse no sólo la cadena operando, sino también el punto de inicio y el número de caracteres que debe tomarse para formar una nueva cadena.

PRESENTACIÓN DE NÚMEROS ENTEROS Y REALES

ENTEROS BINARIOS Y DECIMALES

El método más ampliamente utilizado para la interpretación de la asignación de bits como enteros no negativos es el **sistema de números binarios.** En este sistema cada posición del bit representa una potencia de 2. El bit que está en la posición más hacia la derecha representa $2^{0,}$, lo cual es igual a 1; el de la siguiente posición a la izquierda representa 2^1, el cual es 2; el bit de la posición siguiente representa 2^2, el cual es 4; y así sucesivamente.

Un entero es representado como una suma de potencias de 2. una hilera de puros ceros representa el numero cero. Si un 1 aparece en alguna posición de un bit en particular, la potencia de 2 representada por la posición de este bit es incluida en la suma, pero si aparece un cero la potencia de 2 no es incluida en la suma. Por ejemplo,

el grupo de bits 00100110 tiene unos en la posición 1,2 y 5 (contados de derecha a izquierda con la posición más hacia la derecha interpretada como la posición cero).

Existen dos métodos ampliamente utilizados para representar números binarios negativos. En el primer método, denominado de **complemento unitario** un número negativo es representado cambiando cada bit de su valor absoluto a la posición opuesta correspondiente. Por ejemplo, puesto que 00100110 representa el número 38, 11011001 es utilizado para representar a –38. Una hilera de bits que empiece con un 0 representa un número positivo, mientras que una hilera de bits que empiece con 1 representa un número negativo.

El segundo método de representar números binarios negativos es denominado método del **complemento doble.** En esta notación, se le agrega un 1 a la representación e complemento unitario de un número negativo. Por ejemplo, puesto que 11011001 representa –38 en la notación de complemento unitario 11011010 es utilizado para representar –38 e la notación de complemento doble.

El sistema de números binarios de ninguna manera es el método mediante el cual se pueden utilizar bits para representar enteros. Por ejemplo, una hilera de bits puede ser utilizada para representar enteros en el sistema de números decimales, en la siguiente forma: 4 bits se pueden utilizar para representar un dígito decimal entre 0 y 9 en la notación binaria descrita anteriormente. Una hilera de bits de longitudes arbitrarias puede ser dividida en grupos consecutivos de 4 bits, donde cada grupo representa un dígito decimal. La hilera entonces representa el número que está formado por estos dígitos decimales en una notación decimal convencional. Por ejemplo, en este sistema, la hilera de bits 00100110 se puede separar en 2 hileras de 4 bits cada uno: 0010 y 0110.el primero de estos grupos representa el dígito decimal 2 y el segundo representa el dígito decimal 6, de tal manera que la hilera completa representa el entero 26. esta representación es llamada **decimal codificado en binario.**

NÚMEROS REALES:

El método común y corriente utilizado por los computadores para representar números reales es la denotación de punto flotante. Existen muchas variedades de notación de punto flotante y cada una de ellas tiene sus características individuales. El concepto

clave es de que un número real es representado por un número denominado **mantisa** multiplicada por una **base** elevada a una potencia entera, denominada **exponente.** La base generalmente es fija y la mantisa y el exponente varían de acuerdo a la representación de diferentes números reales.

En la notación de punto flotante un número real está representado por una hilera de 32 bits formada por una mantisa de 24 bits seguida de un exponente de 8 bits. La base se fija como 10. Tanto la mantisa como el exponente son enteros binarios de complemento doble.

Algunos números reales y representaciones de puntos flotantes son:

0 00000000000000000000000000000000

.5 00000000000000000000010111111111

-387. 53 11111111011010001001111111111110

La ventaja de la notación de punto flotante es que esta puede ser utilizada para representar números con valores absolutos extremadamente grandes o extremadamente pequeños. El número positivo más pequeño que puede ser representado es 10^{-128} el cual es verdaderamente pequeño. No necesariamente cada número comprendido entre el más grande y el más pequeño puede ser representado. Nuestra representación permite solamente 23 bits significativos. Es decir un número tal como 10 millones uno, el cual requiere 24 dígitos binarios significativos en la mantisa, tendría que ser aproximado a 10 millones (1×10^7), el cual requiere solamente un dígito significativo.

ANÁLISIS DE ALGORITMOS

CONCEPTO COMPLEJIDAD ALGORITMOS

La resolución práctica de un problema exige por una parte un algoritmo o método de resolución y por otra un programa o codificación de aquel en un ordenador real.

Ambos componentes tienen su importancia, pero la del algoritmo es absolutamente esencial, mientras que la codificación puede muchas veces pasar a nivel de anécdota.

A efectos prácticos o ingenieriles, nos deben preocupar los recursos físicos necesarios para que un programa se ejecute.

Aunque puede haber muchos parámetros, los más usuales son el tiempo de ejecución y la cantidad de memoria (espacio).

Ocurre con frecuencia que ambos parámetros están fijados por otras razones y se plantea la pregunta inversa: ¿cual es el tamaño del mayor problema que puedo resolver en T segundos y/o con M bytes de memoria?

En lo que sigue nos centramos casi siempre en el parámetro tiempo de ejecución, si bien las ideas desarrolladas son fácilmente aplicables a otro tipo de recursos.

Para cada problema determinaremos una medida N de su tamaño (por número de datos) e intentaremos hallar respuestas en función de dicho N.

El concepto exacto que mide N depende de la naturaleza del problema.

Así, para un vector se suele utilizar como N su longitud; para una matriz, el número de elementos que la componen; para un grafo, puede ser el número de nodos (a veces es mas importante considerar el número de arcos, dependiendo del tipo de problema a resolver), en un archivo se suele usar el número de registros, etc.

Es imposible dar una regla general, pues cada problema tiene su propia lógica de coste.

TIEMPO DE EJECUCIÓN

Una medida que suele ser útil conocer es el tiempo de ejecución de un programa en función de N, lo que denominaremos T(N).

Esta función se puede medir físicamente (ejecutando el programa, reloj en mano), o calcularse sobre el código contando instrucciones a ejecutar y multiplicando por el tiempo requerido por cada instrucción.

Así, un trozo sencillo de programa como:

```
S1; for i=0 to N step 2;
```

Requiere

```
T(N)= t1 + t2*N
```

Siendo t1 el tiempo que lleve ejecutar la serie “S1” de sentencias, y t2 el que lleve la serie “S2”.

Prácticamente todos los programas reales incluyen alguna sentencia condicional, haciendo que las sentencias efectivamente ejecutadas dependan de los datos concretos que se le presenten.

Esto hace que mas que un valor T(N) debamos hablar de un rango de valores

$$Tmin(N) \Leftarrow T(N) \Leftarrow Tmax(N)$$

los extremos son habitualmente conocidos como “caso peor” y “caso mejor”. Entre ambos se hallara algún “caso promedio” o más frecuente. Cualquier fórmula T(N) incluye referencias al parámetro N y a una serie de constantes “Ti” que dependen de factores externos al algoritmo como pueden ser la calidad del código generado por el compilador y la velocidad de ejecución de instrucciones del ordenador que lo ejecuta.

Dado que es fácil cambiar de compilador y que la potencia de los ordenadores crece a un ritmo vertiginoso (en la actualidad, se duplica anualmente), intentaremos analizar los algoritmos con algun nivel de independencia de estos factores; es decir, buscaremos estimaciones generales ampliamente válidas.

ESTRUCTURAS

Una *estructura de datos* es una colección de datos que pueden ser caracterizados por su organización y las operaciones que se definen en ella.

Dentro de ellas encontramos distintos tipos, los tipos de datos más frecuentes en los diferentes lenguajes son:

Tipos de datos		
simples	estándar	entero (*integer*)
		real (*real*)
		carácter (*char*)
		lógico (*boolean*)

	definidos por el programador	subrango (*subrange*)
	(no estándar)	enumerativo (*enumerated*)
estructurados	simples o estáticos	arrays (vectores/matrices)
		registros
		ficheros
		conjuntos
		cadenas (*string*)
	compuestos o dinámicos	listas (pilas/colas)
		listas enlazadas
		árboles
		grafos

Las estructuras estáticas son aquellas en las que el tamaño de memoria ocupado se define antes de que el programa se ejecute y no puede modificarse durante la ejecución

Las estructuras dinámicas son aquellas en las que no se debe definir previamente el tamaño de memoria

Los datos simples tienen en común que cada variable representa un elemento, en los estructurados un identificador puede representar múltiples datos individuales, pudiendo cada uno de estos ser referenciados independientemente.

CLASIFICACIÓN DE ESTRUCTURAS DE DATOS

Una estructura de datos es una clase de datos que se puede caracterizar por su organización y operaciones definidas sobre ella. Algunas veces a estas estructuras se les llama tipos de datos.

ESTRUCTURAS LÓGICAS DE DATOS

Las estructuras de datos son muy importantes en sistemas computacionales. En un programa, cada variable pertenece a alguna estructura de datos explícita o implícitamente definida, la cual determina el conjunto de operaciones validas para ella. Las estructuras de datos que se discuten aquí son estructuras de datos lógicas. Cada estructura de datos lógica puede tener varias representaciones físicas diferentes para sus almacenamientos posibles.

ESTRUCTURAS PRIMITIVAS Y SIMPLES

Son primitivos aquellas que no están compuestas por otras estructuras de datos por ejemplo, enteros, **booleanos** y caracteres. Otras estructuras de datos se pueden construir de una o mas primitivas. Las estructuras de datos simples que consideramos se construyen a partir de estructuras primitivas y son: cadenas, arreglos y registros. A estas estructuras de datos las respaldan muchos lenguajes de programación.

ESTRUCTURAS LINEALES Y NO LINEALES

Las estructuras de datos simples se pueden combinar de varias maneras para formar estructuras mas complejas. Las dos cases principales de estructuras de datos son las lineales y las no lineales, dependiendo de la complejidad de las relaciones lógicas que representan. Las estructuras de datos lineales incluyen pilas, colas y listas ligadas lineales. Las estructuras de datos no lineales incluyen grafos y árboles.

ORGANIZACIÓN DE ARCHIVOS

Las técnicas de estructuración de datos aplicadas a conjuntos de datos que los sistemas operativos manejan como “cajas negras” comúnmente se llaman Organización de Archivos. Un archivo tiene nombre, contenido, dirección donde se guarda y alguna información administrativa, por ejemplo, quién la elaboró y cuán grande es. Las cuatro clases básicas de organización de archivos son secuencial, relativo, secuencial indexado, y multillave.

PRIMITIVAS

Enteros

Una estructura de datos primitiva son los enteros. Un entero es un miembro del siguiente conjunto de números:

{...,-(n+1), -n,...-2,-1,0,1,2...n,n+1,...}

Las operaciones fundamentales sobre enteros son: suma, resta, multiplicación, división, exponenciación y otras. Todas estas operaciones trabajan sobre un par de números considerados como operadores binarios.

Booleanos

También llamado lógico. Es un elemento que puede tener uno de dos valores: verdadero o falso. Los tres operadores booleanos básicos son **not, and**, y **or** (negación, conjunción, y disyunción)

TIPOS DE DATOS

El primer objetivo de toda computadora es el manejo de la información o datos. Un dato es la expresión general que describe los objetos con los cuales opera una computadora. La mayoría de las computadoras pueden trabajar con varios tipos de datos. Los algoritmos y los programas correspondientes operan sobre datos.

Existen dos clases de tipos de datos: **Simples y Compuestos**.

Los tipos de datos simples son los siguientes:

Numéricos (Integer, Real)

Lógicos (Boolean)

Carácter (Char, String)

DATOS NUMÉRICOS

El tipo numérico es el conjunto de valores numéricos. Estos pueden representarse en dos formas distintas:

Tipo Numérico Entero (Integer)

Tipo Numérico Real (Real)

Enteros: Es un subconjunto finito de los números enteros. Los enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser positivos y negativos, por ejemplo, 5,6,-15,-1340.

Los números enteros máximos y mínimos de una computadora de 16 bits suelen ser –32768 a +32767. los números enteros fuera de este rano no se suelen representar como enteros, sino como reales, aun que existen excepciones (FORTRAN, Quick/Qbasic, C, C++,etc).

Reales: Consiste en un subconjunto de los números reales. Lo números reales siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consta de un entero y una parte decimal, por ejemplo, 0.08,3789.25,-8.12,3.0

DATOS LÓGICOS (BOOLEANOS)

Es aquel dato que sólo puede tomar uno de dos valores: **cierto** o **verdadero** (true) y **falso** (false).

Este tipo de datos se utiliza para representar las alternativas (sí/no) a determinadas condiciones.

DATOS TIPO CARÁCTER Y TIPO CADENA

El tipo carácter el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato tipo carácter contiene sólo un carácter.

Los caracteres que reconocen las diferentes computadoras no son estándar; sin embargo, la mayoría reconoce los siguientes caracteres alfabéticos y numéricos:

- Caracteres Alfabéticos (A,B,C....X,Y,Z) (a,b,c,.....z)
- Caracteres Numéricos (1,2,3.....,9,0)
- Caracteres especiales (+,-,*,/,`,<,>,°,$,&....)

Una cadena (string) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación.

ORGANIZACIÓN FÍSICA DE LA MEMORIA

El elemento básico de una memoria semiconductora es la celda de memoria. Aunque se utilizan diversas tecnologías electrónicas, todas las celdas de memoria comparten ciertas propiedades:

Presentan dos estados estables (o semi-estables), que pueden emplearse para representar el 1 y el 0 binarios.

- Puede escribirse en ellas (a menos una vez) para fijar su contenido.
- Pueden leerse para detectar su estado.

La figura 1.1 describe el funcionamiento de una celda de memoria. Lo mas común es que la celda tenga tres terminales par transportar señales eléctricas. El Terminal de selección, como su nombre lo indica, selecciona la celda para la operación de escritura de lectura. El Terminal de control indica el tipo de operación. Para la escritura, el tercer Terminal proporciona la señal que fija el estado de la celda.

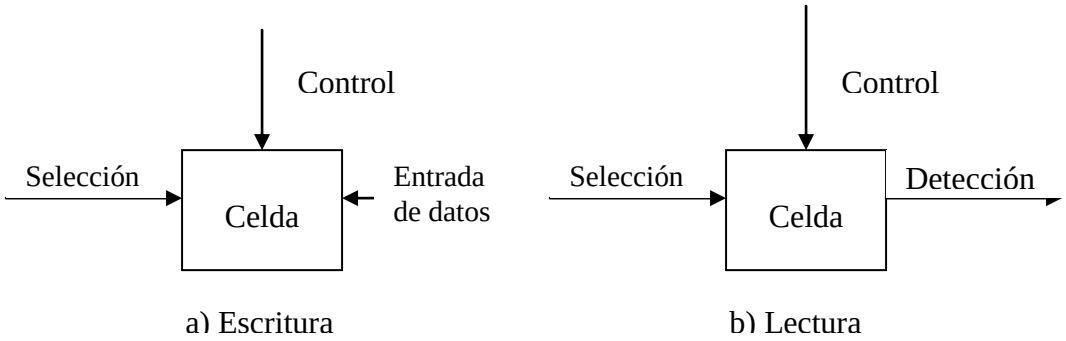


FIGURA 1.1 Funcionamiento de una celda de memoria

Para construir memorias grandes se requiere una organización más compleja. Lo que se necesita es un diseño en que el número de patas crezca en relación logarítmica con la capacidad de la memoria, en lugar de hacerlo de forma lineal.

DEFINICION DE ARREGLOS

Los arreglos (array) son la forma mas simple de las estructuras. Un arreglo unidimecional es un conjunto ordenado, finito de elemento homogéneos.

- Finito.- Quiere decir que existe un número específico de elementos.
- Ordenado.- Quiere decir que existe un orden, un primero, un segundo, un tercero, etc.
- Homogéneo.- Un arreglo puede ser de enteros o caracteres, pero no una mezcla de los dos es decir, todos sus elementos deben ser del mismo tipo.

Si un arreglo tiene la característica de que puede almacenar a N elementos del mismo tipo, deberá tener la facilidad de permitir el acceso a cada uno de ellos. Así, se distinguen dos partes en los arreglos:

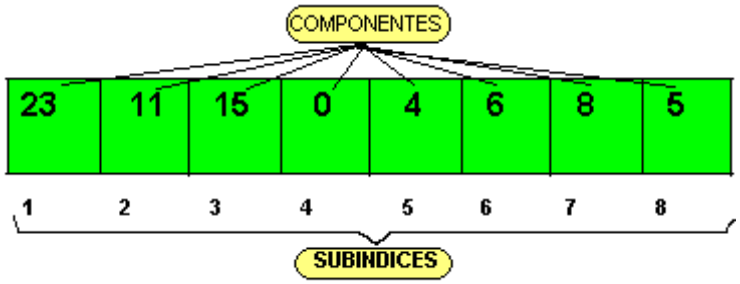
- los componentes
- los índices

Los componentes hacen referencia a los elementos que forman el arreglo. Es decir, a los valores que se almacenan en cada una de las casillas del mismo (véase la figura 1.3). Considerando el ejemplo anterior, cada una de las 50 calificaciones será un componente de un arreglo "Calificaciones". Los índices especifican cuantos elementos tendrá el arreglo y además de que modo podrán accesarse esos componentes.

Los índices permiten hacer referencia a los componentes del arreglo en forma individual. Es decir, distinguir entre los elementos del mismo. Por lo tanto, para hacer referencia a un elemento de un arreglo se utiliza:

- el nombre del arreglo
- el índice del elemento

En siguiente figura se presenta un arreglo y se indican sus componentes y sus índices.



ARREGLOS

En general en cualquier lenguaje de programación, se define un arreglo de la siguiente manera:

```
ident_arreglo = ARREGLO[liminf..limsup] de tipo
```

Con los valores liminf y limsup se declara en tipo de los índices asi como el numero de los elementos que tendrá el arreglo. El numero total de componentes.

(NTC) que tendrá el arreglo que puede calcularse con la formula:

```
NTC = limsup - liminf + 1
```

Con tipo se declara el tipo de datos para todos los componentes del arreglo. El tipo de los componentes no tiene que ser necesariamente el mismo que el tipo de los índices.

OBSERVACIONES:

El tipo de índice puede ser cualquier tipo ordinario (carácter, entero, enumerado).

El tipo de los componentes puede ser cualquier tipo (entero, real, cadena de caracteres, registro, arreglo, etc.)

Se utilizan los corchetes "[ ]" para indicar el índice de una arreglo. Entre los [ ] se debe escribir un valor ordinal (puede ser variable, una constante o una expresión tan compleja como se quiera, pero que de como resultado un valor ordinal).

ARREGLOS MULTIDIMENCIONALES

El tipo componente de un arreglo puede ser otro arreglo de ese, por ejemplo:

```
Típo Arr = Arreglo [ 1..2] es Arreglo de [1..8]
```

El numero de filas o de columnas es igual al limite superior menos el limite inferior mas 1. Este número es denominado el rango de la dimensión.

La posición de cada uno de los elementos esta determinada por dos subíndices, que determinan la fila y la columna, el siguiente ejemplo esquematiza a el arreglo y los subíndices.

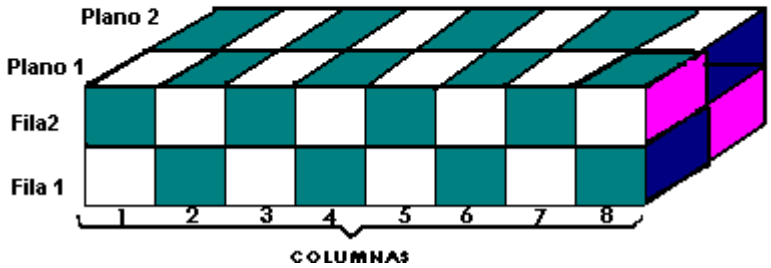
" "

Aunque en la práctica parezca que la computadora almacena la información en 2 dimensiones, la memoria es lineal, es decir un arreglo unidimensional. El método que

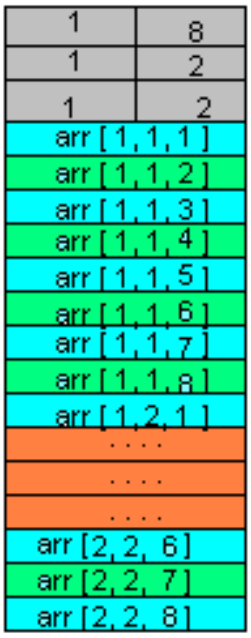
utiliza la computadora es ocupar el primer conjunto de posiciones para la primera fila, la segunda fila ocupa el segundo conjunto.

Bajo este mismo método se representan los arreglos de más de una dimensión, por ejemplo un arreglo de tres dimensiones gráficamente se representan de la siguiente forma:

REPRESENTACION DE UN ARREGLO
MULTIDIMENCIONAL



Los elementos de este arreglo tridimensional en forma unidimensional se representan de la forma:



INSERCIÓN Y BORRADO DENTRO DE UN ARREGLO

Las dos operaciones básicas que accesan un arreglo son extracción y almacenamiento. La operación de extracción es una función que acepta un arreglo a y un elemento de su tipo índice i y regresa un elemento del tipo base del arreglo. El resultado de esta operación se representa por la excreción a[i].

La operación de almacenamiento acepta un arreglo a, un elemento de su tipo índice, y un elemento de su tipo base x.

Las operaciones de asignación dependen del lenguaje de programación que se utilices esta operación se define mediante la regla que dice que después de haber ejecutado la declaración de asignación, el valor de [a]i es x. Antes de que se haya asignado un valor a un elemento del arreglo, su valor no está definido y al referirse a él en una expresión es ilegal.

El tipo índice de un arreglo puede ser cualquier tipo ordinal y el tipo base puede ser cualquier tipo válido.

REGISTROS

Cuando se hablo de los arreglos se hizo mención de que se trataba de una colección de datos, que era un tipo estructurado de datos, y que con ellos se podrían solucionar un gran número de problemas. Sin embargo, en la práctica a veces se necesitan estructuras que permitan almacenar distintos tipos de datos (característica con la cual no cuentan los arreglos). Para ilustrar este problema se incluye el siguiente ejemplo. Una compañía tiene por cada empleado los siguientes datos:

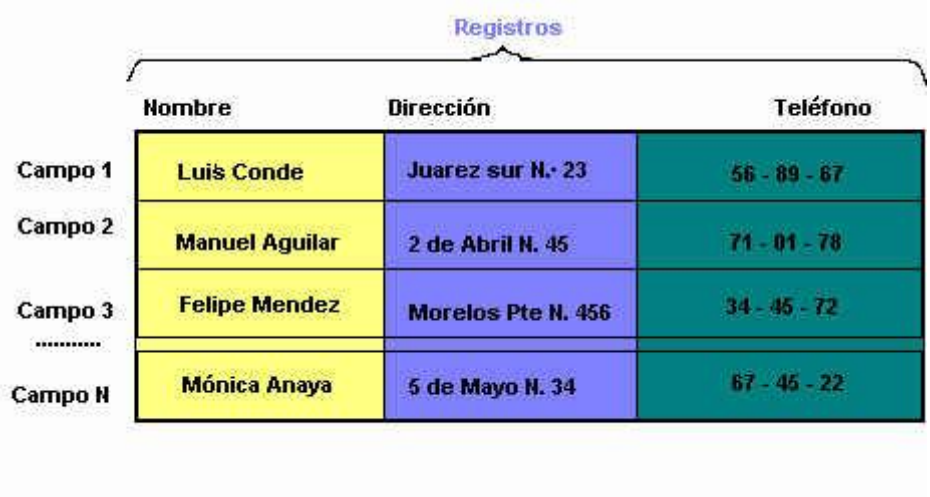
- Nombre (cadena de caracteres)
- Dirección (cadena de caracteres)
- Edad (entero)
- Sexo (carácter)
- Antigüedad (entero)

Si se quisiera almacenar estos datos no posible usar un arreglo, ya que sus componentes deben ser todos del mismo tipo. La estructura que puede guardar esta información es la que se conoce como registro o estructura.

Un registro es un dato estructurado, donde cada uno de sus componentes se denomina campo. Los Campos de un registro pueden ser todos de diferentes tipos por lo tanto, también podrán ser registros o Arreglos. Cada campo se identifica por un nombre único (el identificador de campo).

Cada uno de los campos esta definido por un tipo de dato que pueden ser diferentes y utilizando los tipos de datos básicos que se mencionaron con en los temas de la primera unidad. A continuación tenemos la representación de un registro con el ejemplo de un directorio telefónico, donde existen solamente 3 campos:

- Nombres
- Dirección
- Teléfono



DEFINICIÓN DE REGISTRO

Un registro se define de la siguiente manera:

```
Ident_registro=REGISTRO
        ID _ campo1: tipo1
        Id _ campo2: tipo2
        ...
```

```
ID _ campo: tipo

(Fin de la definición del registro)
```

Donde: `Ident _ registro` es el nombre del dato tipo registro

`Id _ campo1`, es el nombre del campo 1, Tipo 1 es el tipo del campo ID

Los que siguen son ejemplos de definición de registros, con su correspondiente representación grafica

EJEMPLO

Sea FECHA un registro formado por tres campos numéricos. Su representación queda como se muestra en continuación.

```
FECHA = REGISTRO

    Dia: 1.31
    Mes : 1.12
    Año : 0..2000
    (Fin de la definición del registro FECHA)

FECHA
```

ACCESO A LOS CAMPOS DE UN REGISTRO

Como un registro es un dato estructurado no puede accesarse directamente como un todo, si no que debe especificarse que elemento (campo) que el registro interesa. Para ello, en la mayoría de los lenguajes Se sigue la siguiente sintaxis:

```
Variable _ registro.id _ campo
```

Donde: Variable _ registro es una variable de tipo, registro, Id _ campo es el identificador del campo deseado, Es decir, se usaran dos nombres para hacer referencia a un elemento: el nombre de la variable tipo Registro y el nombre de la componente, separados entre si por un punto.

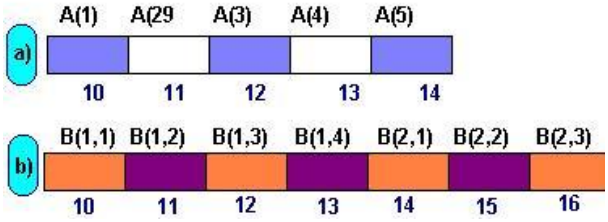
ACCESO DE UN ARREGLO.

Ya sabemos que las localidades de memoria básicamente peden organizarse de forma contigua y ligada, un arreglo también puede organizarse de cualquiera de estas dos formas:

- Organización contigua.
- b) Organización ligada.

ORGANIZACIÓN CONTIGUA.

Los nodos del arreglo ocupan localidades contiguas de la memoria.



Cuando los nodos del arreglo se organizan en la memoria de forma contigua, la recuperación de cualquiera de los nodos se hace básicamente en dos esquemas:

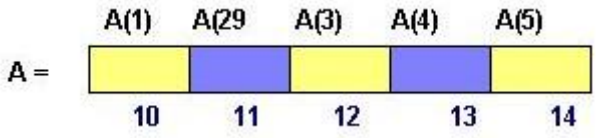
- Función de mapeo de relación 1 a 1.
- Tablas de recuperación.

FUNCIÓN DE MAPEO DE RELACION 1 A 1.

Una función de mapeo de relación 1 a 1 , es una expresión que permite calcular la dirección de un nodo en función de sus índices.

Ejemplo:

Consideremos que el arreglo A es almacenado en forma contigua.

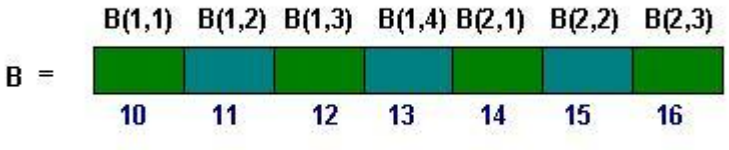


Una expresión que nos permite llegar a cualquier nodo A(i) es : DIR A = 9+i para **A(3) = 9 + 3 = 12**, de forma general:

```
DIR  A(i) = BA+i
```

Donde BA es una dirección base.

b) Consideremos que B está organizado en forma contigua.



Si observamos el orden en que están los nodos de A podemos decir que el almacenamiento de los elementos es por renglones; el primer índice se mantiene constante y el segundo es el que varía; en algunos casos el orden de almacenamiento es por columnas.

Una expresión que permite llegar a cualquier elemento B(i,j) del arreglo es :

```
DIR B(i,j) = 9+4(i-1)+j
para :
DIR B(2,3) = 9+4(2-1)+3 = 16
de forma general:
DIR  B(i,j) = BA+i (i-1) + j
```

Una función de mapeo asociada a este tipo d e arreglos ofrece un ahorro considerable de las localidades de memoria.

CAPITULO II: MODELAMIENTO DE DATOS

Entenderemos por "modelamiento y simulación" a las actividades asociadas con la construcción de modelos de sistemas del mundo real, y su simulación en un computador.

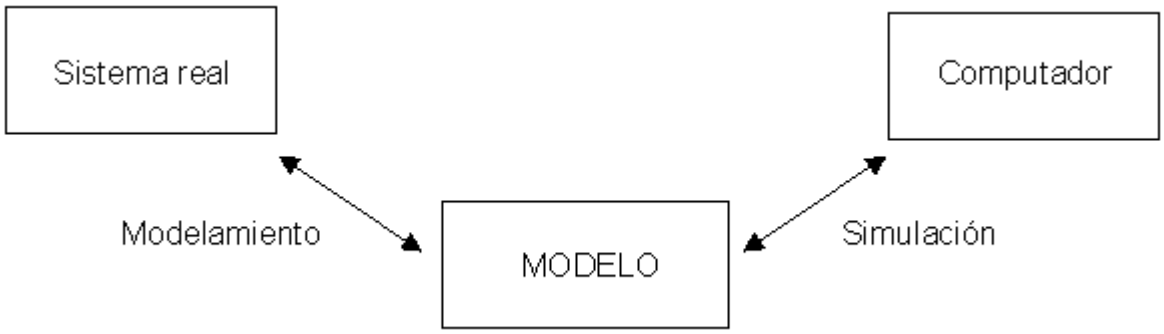
Los modelos son útiles para predecir y/o estudiar el comportamiento de un sistema real, que puede servir para corroborar algunas hipótesis. Por ejemplo, para predecir el movimiento de objetos en la superficie de la Tierra, sometidos a la fuerza de gravedad, las ecuaciones de Newton son un modelo suficientemente bueno. Con ellas se podría predecir cuál sería el movimiento (la posición segundo a segundo) de un objeto que se suelta desde una altura determinada. Sin embargo, si los objetos se mueven a velocidades comparables con la velocidad de la luz, las ecuaciones de Einstein describen mejor su comportamiento.

Muchas veces se usan modelos de sistemas (que incluso puede que no existan todavía) para ver cómo funcionan estos sistemas bajo distintas condiciones (con distintos parámetros) y ver cuáles son las condiciones necesarias para que el sistema sirva o trabaje en forma óptima. Hay muchas razones por las cuales es conveniente experimentar en un modelo y no en la vida real: costos, tiempo, peligro o simplemente imposibilidad. Los experimentos son repetibles. Algunos ejemplos de sistemas de la vida real que pueden ser modelados son: supermercados, hospitales, redes de caminos, represas, redes de computadores y modelos económicos.

En general, los modelos son una simplificación de la vida real. Esto porque el sistema real generalmente es muy complicado, o porque sólo se pretende estudiar una parte del sistema real.

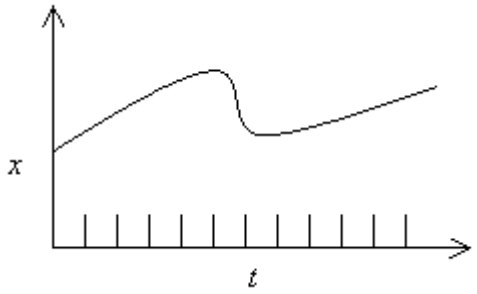
El proceso de definición del modelo de un sistema (real o no real) se llama *modelamiento*. La *simulación* consiste en usar el modelo para generar datos acerca del comportamiento del sistema para ver cómo se comportaría, bajo el supuesto de que el modelo está bien hecho.

En general, cualquiera sea la forma que adopte, el modelo debe ser capaz de proveer instrucciones a alguien o algo, de modo que pueda generar datos que describan el comportamiento del sistema modelado. Así entonces, se tiene un *sistema real*, cuyo *modelamiento* genera un *modelo* que puede ser representado en un *computador* a través de una *simulación*.



El "sistema real" es la parte del mundo real de nuestro interés. Como regla general, podemos decir que el sistema real es una *fuente de datos conductuales,* los cuales consisten en formas primarias de gráficos *x* versus *t,* donde *x* puede ser cualquier variable de interés, tal como la temperatura de un cuarto, el número de pétalos de una

flor o el Producto Nacional Bruto, y *t* es el tiempo, medido en unidades como segundos, días o años. La siguiente figura muestra un ejemplo.



Un *modelo* es básicamente un conjunto de instrucciones para generar datos conductuales (con cierto comportamiento) de la forma de la figura anterior. Los modelos son algunas veces expresados en forma de ecuaciones diferenciales, notación teórica de autómatas o en formalismo de eventos discretos.

MODELAMIENTO DE DATOS

Desde tiempos remotos, los datos han sido registrados por el hombre en algún tipo de soporte (piedra, papel, madera, etc.) a fin de que quedara constancia de un fenómeno o idea. Los datos han de ser interpretados para que se conviertan en información útil, esta interpretación supone un fenómeno de agrupación y clasificación.

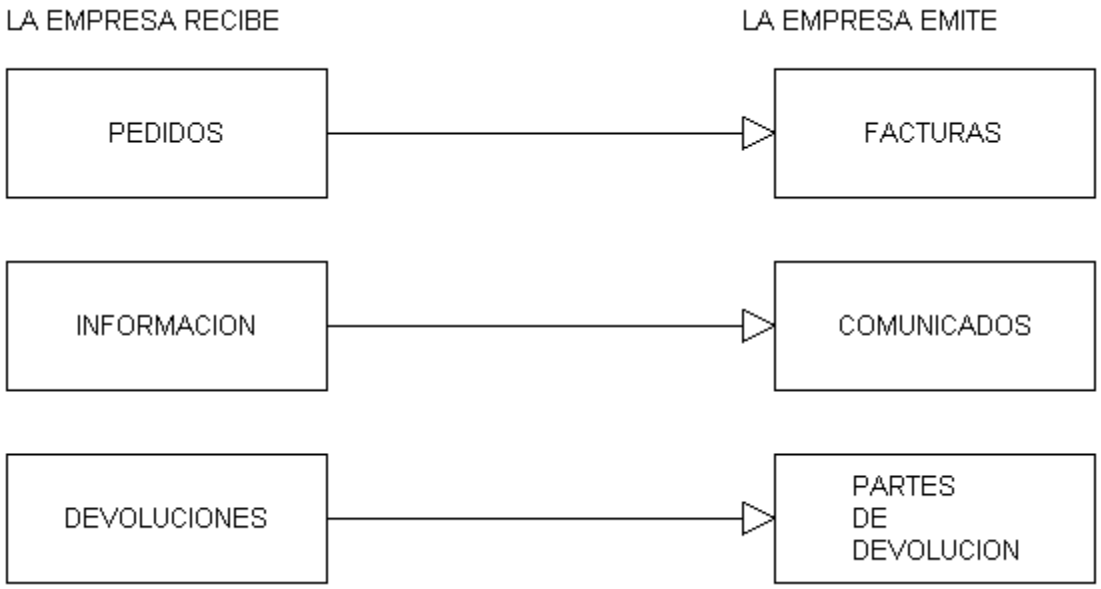
En la era actual y con el auge de los medios informáticos aparece el almacenamiento en soporte electromagnético, ofreciendo mayores posibilidades de almacenaje, ocupando menos espacio y ahorrando un tiempo considerable en la búsqueda y tratamiento de los datos. Es en este momento donde surge el concepto de bases de datos y con ellas las diferentes metodologías de diseño y tratamiento.

El objetivo básico de toda base de datos es el almacenamiento de símbolos, números y letras cadentes de un significado en sí, que con un tratamiento adecuado se convierten en información útil. Un ejemplo podría ser el siguiente dato: 19941224, con el tratamiento correcto podría convertirse en la siguiente información: "Fecha de nacimiento: 24 de diciembre de 1994".

Según van evolucionando los tiempos, las necesidades de almacenamiento de datos van creciendo y con ellas las necesidades de transformar los mismos datos en información de muy diversa naturaleza. Esta información es utilizada diariamente como herramientas de trabajo y como soporte para la toma de decisiones por un gran colectivo de profesionales que toman dicha información como base de su negocio. Por este motivo el trabajo del diseñador de bases de datos es cada vez más delicado, un error en el diseño o en la interpretación de datos puede dar lugar a información incorrecta y conducir al usuario a la toma de decisiones equivocadas. Se hace necesario la creación de un sistema que ayude al diseñador a crear estructuras correctas y fiables, minimizando los tiempos de diseño y explotando todos los datos, nace así la metodología de diseño de bases de datos.

La metodología de diseño de datos divide cada modelo en tres esquemas:

A) Modelo Global: se trata de una representación gráfica legible por el usuario y que nos aporta el flujo de información dentro de una organización. No existen reglas para su construcción y se debe realizar siempre el esquema más sencillo posible para la comprensión por parte del usuario de la base de datos. Por ejemplo:



B) MODELO LÓGICO: se trata de una representación gráfica, mediante símbolos y signos normalizados, de la base de datos. Su objetivo es representar la estructura de los datos y las dependencias de los mismos, garantizando la consistencia y evitando la duplicidad. Este modelo de datos se estudiará con profundidad en los capítulos siguientes.

C) MODELO FÍSICO: se trata del almacén de los datos, es la base de datos en sí misma, el soporte donde se almacenan los datos y de donde se extraen para convertir los datos en información. En función del gestor de bases de datos empleado las reglas de almacenamiento varían.

¿CÓMO SE SABE SI UN MODELO ES BUENO?

Para saber si un modelo es bueno, basta comparar los resultados que arroja con los del sistema real que se quiere estudiar. La validez del modelo depende de "cuan bien el

modelo representa al sistema real" en términos de los datos arrojados por el modelo versus los datos del sistema real. El problema es que muchas veces el sistema no existe, pues puede ser un modelo de algo que se quiere construir. En todo caso, lo principal de un modelo es que los resultados que arroje reflejen de alguna manera lo que se quiere estudiar acerca del sistema que se está modelando. No existe ninguna manera de saber cuál es el mejor modelo para un sistema. Es posible comparar dos modelos y decidir cuál de ellos es mejor bajo algún punto de vista particular, pero en ningún caso se puede saber exactamente cuándo se está frente al mejor modelo para una situación dada.

LOS USUARIOS

En todo sistema de base de datos cabe diferenciar tres tipos diferentes de usuarios, entre todos comparten la información pero acceden a ella de una forma diferente, siempre en función de sus necesidades.

El primer grupo de usuarios es el PED (Procesamiento Electrónico de Datos), normalmente compuestos por los operarios de la organización. Las necesidades básicas de este grupo de usuarios son:

- El foco operativo fundamental se centra en el almacenamiento de los datos, el procesamiento de los mismos y el flujo de datos.
- Generan informes de tipo listados;
- Poseen acceso restringido a la información.

El segundo grupo de usuarios es el SIM (Sistemas de Información de Gestión) y suele estar formado por los mandos medios de la organización. Las necesidades básicas de este grupo de usuarios son:

- El foco operativo se fundamenta en la toma de decisiones, tomando como partida los datos del grupo PED e introduciendo un volumen pequeño de información;
- No poseen acceso medianamente restringido a la información;
- Generan informes de resúmenes de datos del grupo PED y listados de la información que introducen.

El tercer último grupo de usuarios lo forman el STD (Sistema de apoyo a Toma de Decisiones), este grupo se centra en el nivel más alto de la organización y poseen las características siguientes:

- El foco operativo se centra en la decisión, con una entrada mínima de datos;
- No tienen acceso restringido;
- Generan informes globales que les sirven como apoyo a las tomas de decisiones del negocio, estos son los informes más importantes y suelen ir acompañados de resúmenes, gráficas y sobre todo centrados en la evolución y comparación de la información.

Cabe destacar la figura de un cuarto grupo de usuarios, en este caso usuarios avanzados, que está compuesto por los administradores del sistema, cuya opinión es fundamental para seleccionar el soporte de los datos, evitar la duplicación de

información ya existente en otros sistemas y sobre todo puede aportar el conocimiento de sus usuarios, sus necesidades y los problemas ya resueltos.

En general, podemos decir que los objetivos de una base de datos son los siguientes:

- Ayudar en la toma de decisiones;
- Compartir de forma controlada y restringida los datos y el acceso a la información;
- Integrar los datos de una forma lógica, evitando la duplicidad
- Asegurar un rápido acceso a la información y los datos.

DISEÑO

En esta etapa se crea un esquema conceptual de la base de datos. Se desarrollan las especificaciones hasta el punto en que puede comenzar la implementación. Durante esta etapa se crean modelos detallados de las vistas de usuario y sobre todo las relaciones entre cada elemento del sistema, documentando los derechos de uso y manipulación de los diferentes grupos de usuarios.

Si parte de la información necesaria para crear algún elemento establecido ya se encuentra implementado en otro sistema de almacenamiento hay que documentar que relación existirá entre uno y otro y detallar los sistemas que eviten la duplicidad o incoherencia de los datos.

El diseño consta, como se vio anteriormente, de tres fases: el diseño global o conceptual, el diseño lógico y el modelo físico.

IMPLEMENTACIÓN

Una vez totalmente detallado el modelo conceptual se comienza con la implementación física del modelo de datos, a medida que se va avanzando en el modelo el administrador del sistema va asegurando la corrección del modelo y el validador la utilidad del mismo.

La implementación consiste en el desarrollo de las tablas, los índices de los mismos, las condiciones de validación de los datos, la relación entre las diferentes tablas. Por otro lado, la definición de las consultas y los parámetros a utilizar por cada una de ellas.

Una vez finalizada la implementación física, se asignan las correspondientes medidas de seguridad y se ubica la base de datos en el lugar correspondiente.

EVALUACIÓN Y PERFECCIONAMIENTO

En esta última etapa todos los usuarios del sistema acceden a la base de datos y deben asegurarse el correcto funcionamiento de la misma, que sus derechos son los adecuados, teniendo a su disposición cuanta información necesiten. También deberán asegurarse que el acceso a los datos es cómodo, práctico, seguro y que se han eliminado, en la medida de lo posible, las posibilidades de error.

El administrador se asegura que todos los derechos y todas las restricciones han sido implementados correctamente y que se ha seguido en manual de estilo en la totalidad de la implementación.

El validador se asegurará que todas las necesidades del cliente han sido satisfechas.

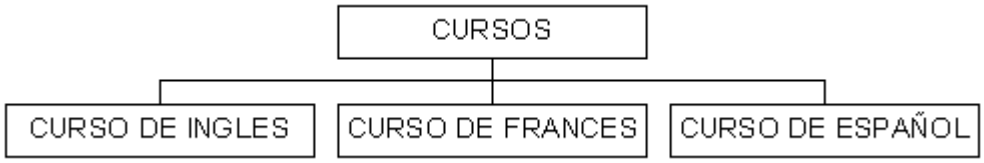
EL MODELO LÓGICO

Anteriormente se expuso el ciclo de vida del desarrollo de una base de datos. Este capítulo se centrará en el diseño del modelo lógico de los datos, por tanto antes de comenzar esta modelación es necesario tener documentado las necesidades, viabilidad y definición de los requisitos, así como tener elaborado el modelo global o conceptual del diseño.

El paso del modelo global o conceptual de datos al modelo lógico supone una abstracción, un mecanismo para la conversión del mundo real a un mundo formado por datos, a su agrupación y clasificación. El proceso de abstracción consiste en identificar los elementos ó conceptos empleados en el modelo global y transformarlo en lo que denominamos entidades en el modelo lógico. La abstracción se puede realizar de las siguientes formas:

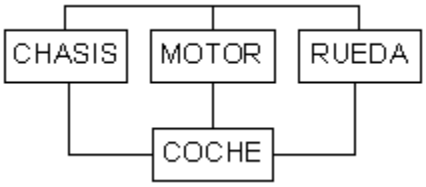
CLASIFICACIÓN

Consiste en generar una única entidad conceptos con características comunes, todos ellos tendrán las mismas características y se diferencian unos de otros por los valores que toman dichas características. Por ejemplo: los conceptos cursos de inglés, cursos de español y cursos de francés se pueden agrupar en una única entidad denominada "CURSOS" que englobe y diferencie cada uno de los diferentes cursos que se imparten.



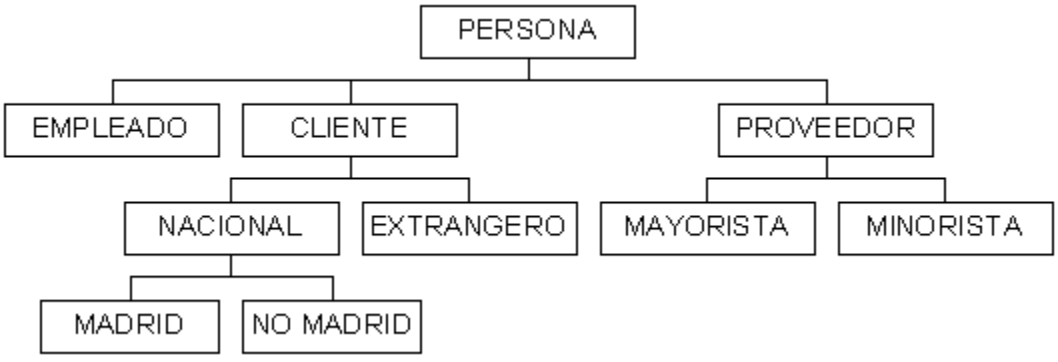
AGREGACIÓN

Consiste en separar cada una de las partes de un concepto para generar distintas entidades, por ejemplo el concepto coche lo podemos definir utilizando las entidades rueda, motor y chasis.



GENERALIZACIÓN

Consiste en ir generando entidades de diferentes niveles de tal forma que cada entidad de nivel superior agrupe las de nivel inferior.



ASOCIACIÓN

Consiste en la generalización de entidades a partir de entidades ya existentes.

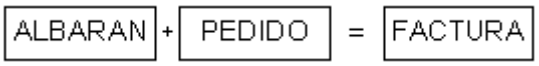


DIAGRAMA ENTIDAD RELACION

Un modelo de datos:

Es una representación conceptual de las estructuras de datos que se almacenarán en una BD. Debe ser independiente del hardware y software.

Su enfoque se centra en mostrar los datos tal cual como los ve el usuario en el “mundo real”.

La notación que se utiliza en su construcción permite ser comprendido por el usuario y también por los diseñadores que crearán la estructura física de la BD. En resumen, un modelo de datos es un “plano” para construir una BD. El modelo más usado para construir modelos de datos para Bases de Datos Relacionales es el Modelo Entidad-Relación. Fue originalmente propuesto por **Peter Chen** en el siguiente artículo: *The Entity-Relationship Model—Toward a Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No 1, March 1976.*

La forma de ver el mundo del modelo ER es en base a Entidades y sus RELACIONES.

Su utilidad se debe a que:

- Mapea bien hacia el modelo relacional. Es decir, es fácilmente transformable hacia una BD Relacional.
- Es independiente del hardware y software.

ENTIDADES

Es todo sobre lo cual se desea recolectar datos y almacenarlos. Usualmente son conceptos reconocibles, concretos o abstractos, tales como persona, lugar, proyecto, etc...

Una ocurrencia de una entidad es una instancia, una ocurrencia individual. Por ejemplo: existe una entidad Alumno, pero miles de instancias de esta entidad.

RELACIONES

Representan una asociación entre dos o más entidades. Ejemplo de relación pueden ser:

- Un alumno inscribe asignaturas.
- Un departamento administra uno o más proyectos. Empleados son asignados a un proyecto.
- Un profesor dicta varias asignaturas.

Las relaciones son clasificadas en términos de grado, conectividad, cardinalidad y existencia.

GRADO

Corresponde al número de entidades asociadas con las relación. La relación binaria es la más común. En algunos casos relaciones de mayor grado se descomponen hacia una relación binaria.

CONECTIVIDAD Y CARDINALIDAD

La conectividad describe en forma “global” la asociación entre entidades. Los tipos básicos de conectividad son: uno-a-uno, uno-a- mucha, muchos-ha-muchos. La cardinalidad cuantifica de manera más precisa la conectividad.

RELACIONES UNO-A-UNO 1:1

en este caso a lo más una instancia de la entidad A se relaciona con una instancia de la entidad B. Por ejemplo: Un proyecto de título es realizado solo por un alumno.

RELACIONES UNO-A-MUCHOS 1:N

Una instancia de la entidad A está relacionada con cero, una o muchas instancias de B, pero una instancia de B se relaciona solo con una de A. Por ejemplo:

- Un departamento tiene muchos proyectos.

- Un proyecto es asignado a un departamento.

RELACIONES MUCHOS-A-MUCHOS N:M

En este caso una instancia de A se relaciona con cero, una o muchas instancias de la entidad B y una instancia de B está relacionada con cero, una o muchas instancias de A. Por ejemplo:

- Empleados no pueden ser asignados a más de 2 proyectos a las vez. (cardinalidad 2)

- Un proyecto debe tener asignados al menos 3 empleados. (cardinalidad 3)

Importante: las relaciones N:M no pueden ser implementadas en forma directa en una BD Relacional. Es necesaria una transformación.

EXISTENCIA

Con esto es posible mostrar cuando la existencia de la instancia de una entidad es dependiente de la existencia de una instancia en otra. Una existencia es obligatoria u opcional.

Por ejemplo:

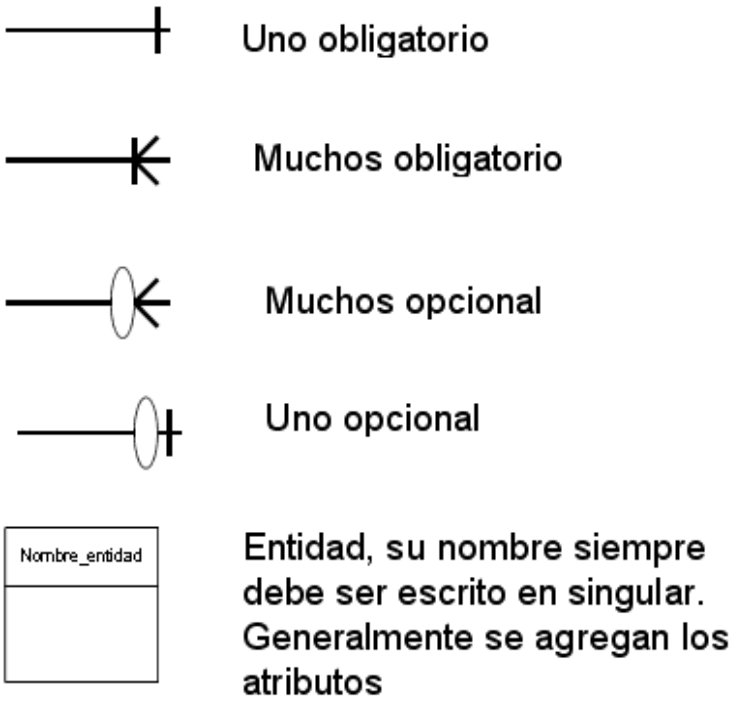
- Cada proyecto debe ser administrado por un departamento. (Obligatoria)

- Un empleado puede o no ser asignado a un proyecto. (Opcional)

ATRIBUTOS

Describen la entidad a la cual pertenecen. Una instancia de un atributo es un valor. Por ejemplo: Calle del Agua 123 es un valor del atributo Dirección. Los atributos pueden ser clasificados como descriptores o identificadores.

NOTACIÓN

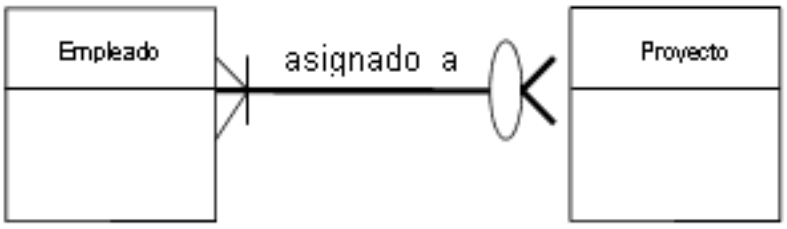


REFINAMIENTO DEL MODELO ER

Entidades deben participar en relaciones: la única excepción es un modelo con solo una entidad. Resolver relaciones N:M: las relaciones N:M no son representables en una BD Relacional. La estrategia que comúnmente se utiliza es utilizar una entidad de asociación. Más allá de un problema de implementación en algunos casos es necesario almacenar datos que pertenecen a la relación.

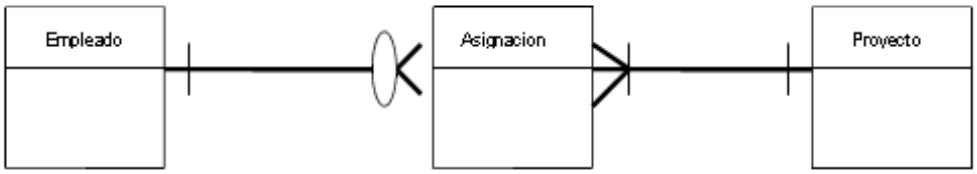
Por ejemplo, si tenemos:

Empleados pueden ser asignados a muchos proyectos. A cada proyecto debe estar asignado más de un empleado.



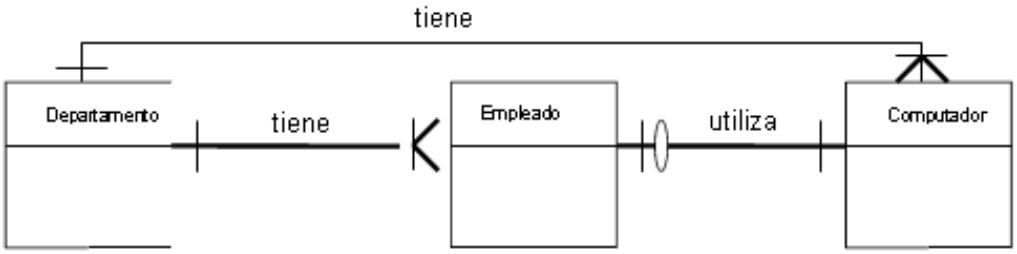
Si un requerimiento fuera: “Determinar cuantas horas de trabajo por proyecto tiene un determinado trabajador“.

A que entidad se pueden asignar los atributos necesarios para representar el periodo de tiempo (fecha inicio y fin) que un empleado se encuentra trabajando en un proyecto.

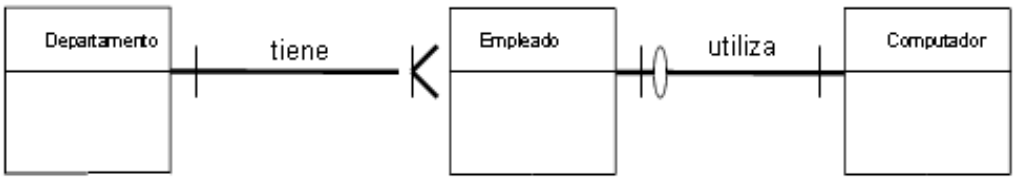


ELIMINAR RELACIONES REDUNDANTES

Una relación redundante es una relación entre dos entidades que es equivalente en significado a otra relación que utiliza entidades intermedias.



Eliminando redundancia



CLAVE PRIMARIA Y CLAVE FORÁNEA

Corresponden al componente más básico sobre el cual se basa la teoría relacional.

La clave primaria identifica de manera única las ocurrencias de una entidad.

La clave foránea permite materializar una asociación entre 2 entidades. Clave Primaria puede estar compuesta de uno o más atributos. Toda entidad debe tener una clave primaria. Para que un atributo califique como clave primaria debe tener las siguientes

propiedades:

- No permitir valores nulos.
- Su valor debe ser único para cada instancia.
- Su valor no debe cambiar ni ser nulo durante la vida de cada instancia.

En el caso de la entidad alumno, la clave primaria es su rol. Una clave compuesta puede ser el par (**ID_Empleado, ID_Proyecto**) para el caso de la entidad asignación. En algunos casos la clave primaria puede ser artificial. Migración de clave primaria Toda entidad que depende de otra para su identificación hereda la clave primaria de su entidad padre. En la entidad que recibe la clave se llamará Clave Foránea. La importancia de la clave foránea es fundamental ya que permite navegar en el modelo de datos.

APLICACIÓN

- Especificación
- Actividades
- Tabla Cliente
- Tabla Camarero
- Tabla Cocinero
- Tabla Mesa
- Tabla Factura
- Tabla DetalleFactura

- Modelo Entidad-Relación
- Consultas

Solución: restaurante.mdb

Especificación

Un restaurante posee una serie de mesas en las que se sientan clientes y que a su vez son atendidas por camareros. Los camareros van tomando nota de los distintos platos que piden los clientes que son preparados por los cocineros. Modelizar el sistema de información para que se puedan emitir facturas de la forma más detallada posible. Considerar que habrá clientes de los cuales conozcamos sus preferencias (es decir que tendrán un **IdCliente**, y otros que quieran permanecer anónimos en el restaurante.

Actividades

Creación de tablas, con sus claves principales y las relaciones entre las mismas.

Tabla Cliente

IdCliente: Cliente Anónimo tiene valor: 1
Nombre
Apellido1
Apellido2
Observaciones

Tabla Camarero

IdCamarero
Nombre
Apellido1
Apellido2

Tabla Cocinero

IdCocinero
Nombre
Apellido1
Apellido2

Tabla Mesa

IdMesa
NumMaxComensales
Ubicacion

Tabla Factura

IdFactura
IdCliente
IdCamarero
IdMesa
FechaFactura

Tabla DetalleFactura

IdFactura
IdDetalleFactura
IdCocinero
Plato
Importe

Modelo Entidad-Relación