

Nivel: Quinto año

Guía de aprendizaje, para el
nivel de undécimo año, en la
especialidad técnica de
desarrollo de software

2015

San José, Costa Rica

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Índice

Diagnóstico, estructuras de control	1
Diagnóstico, ciclos	3
Diagnóstico, corrida de códigos	4
Introducción a la estructuras de datos	8
Concepto	8
Tipos	9
Estructuras de datos contiguas	10
Cadenas	10
Arreglos (Arrays)	11
Vectores	12
Búsqueda en vectores	14
Matrices	21
Registros	22
Prácticas generales de Datos contiguos	23
<i>Primer proyecto</i>	23
Estructuras Dinámicas y punteros	24
Estructuras lineales	25
Listas enlazadas	25
Pilas	34
Colas	37
Estructuras no lineales	39
Arboles binarios	39
Árbol binario de búsqueda	42
Prácticas generales de Estructuras no lineales	47
<i>Segundo proyecto</i>	49
Introducción al Algebra relacional	50
Visión formal e informal de una relación	50
Dominios	51
Clave candidata, clave primaria y clave alternativa de las relaciones	56
Claves foráneas de las relaciones	57
Operaciones del modelo relacional	61
Reglas de integridad	63
El álgebra relacional	77
Operaciones conjuntistas	80
Unión	80
Intersección	83
Diferencia	84
Producto cartesiano	85
Operaciones específicamente relacionales	86
Selección	86
Proyección	88
Combinación	89

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Secuencias de operaciones del álgebra relacional	92
Prácticas generales Algebra relacional	98
Sentencias SQL	100
DDL (Definición)	104
Create table	106
Drop table	109
Alter table	108
DML (Manipulación)	109
Insert	109
Delete	110
Update	110
Select	110
Where	110
Funciones de agregación	111
Subconsultas	112
Consultas a más de una tabla	114
Prácticas generales de SQL Server	115
<i>Tercer proyecto</i>	116
TSQL	117
Declare	117
Operadores	120
If	121
Case	122
Bucle while	123
Uso de TRY CATCH	125
Procedimientos almacenados	126
Triggers	132
Cursos	136
Prácticas generales de TSQL	148
<i>Cuarto proyecto</i>	149
Diagramas de base de datos	150
Introducción al diseño de bases de datos	150
Diagrama entidad/relación	150
Diagrama relacional	161
Diagrama físico	162
Prácticas generales de Diagramas de SQL Server	162
<i>Quinto proyecto</i>	165
Formas Normales	166
I Forma normal	166
II Forma normal	167
III Forma normal	168
Orientación a objetos	170
Clases	171
Objetos	170
Modelos de objetos	171
Principio de abstracción	171

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Principio de encapsulamiento	172
Principio de modularidad	172
Principio de jerarquía	172
Principio de paso de mensajes	173
Principio de polimorfismo	173
Relaciones entre objetos	173
De asociación	174
De Todo/Parte	174
De generalización/Especialización	174
Objetos y clases n C#	174
Prácticas generales de orientación a objetos	188
<i>Sexto proyecto</i>	188
Aplicaciones web	189
Html	190
Master Pages	190
Web form heredado de una master page	191
Componentes Asp	191
Eventos en el servidor	191
Ajax	193
Sesiones	194
Datagrid	196
<i>Séptimo proyecto</i>	196
Conexión con base de datos	197
<i>Octavo proyecto</i>	200

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Diagnóstico de estructuras de control

Pseudocódigo, utilizando solamente:

Si, si no

En caso de

1. Realice el pseudocódigo, para poder resolver los siguientes problemas:

- a. Seleccionar de tres números, el mayor, el menor y el del medio, mostrarlos en pantalla ordenados de mayor a menor.
- b. Permita calcular el total a pagar por la compra de N camisas. Si se compran entre 1 a 4 camisas se aplica un descuento del 12.5%, si se compra una cantidad comprendida entre 5 y 8 camisas se aplica un descuento del 20% y si se compran cantidades mayores, se aplica un descuento del 31.5% sobre el total de la compra. Debe imprimirse en pantalla la compra final sin descuento, monto del descuento y la compra final con descuento.
- c. En una playa de estacionamiento cobran 2.5 dólares por hora o fracción. Diseñe un algoritmo que determine cuánto debe pagar un cliente por el estacionamiento de su vehículo, conociendo el tiempo de estacionamiento en horas y minutos.
- d. Diseñe un algoritmo que determine si un número es o no par.
- e. Una tienda ha puesto en oferta la venta al por mayor de cierto producto, ofreciendo un descuento del 15% por la compra de más de 3 docenas y 10% en caso contrario. Además por la compra de más de 3 docenas se obsequia una unidad del producto por cada docena en exceso sobre 3. Diseñe un algoritmo que determine el monto de la compra, el monto del descuento, el monto a pagar y el número de unidades de obsequio por la compra de cierta cantidad de docenas del producto.
- f. Diseñe un algoritmo que lea un número de tres cifras y determine si es o no capicúa. Un número es capicúa si es igual al revés del número.
- g. Diseñe un algoritmo que califique el puntaje obtenido en el lanzamiento de tres dados en base a la cantidad seis obtenidos, de acuerdo a lo siguiente: tres seis, excelente; dos seis, muy bien; un seis, regular; ningún seis, pésimo.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

- h. Una compañía dedicada al alquiler de automóviles cobra un monto fijo de \$30 para los primeros 300 km de recorrido. Para más de 300 km y hasta 1000 km, cobra un monto adicional de \$ 0.15 por cada kilómetro en exceso sobre 300. Para más de 1000 km cobra un monto adicional de \$ 0.10 por cada kilómetro en exceso sobre 1000. Los precios ya incluyen el 18% del impuesto general a las ventas, IGV. Diseñe un algoritmo que determine el monto a pagar por el alquiler de un vehículo y el monto incluido del impuesto.
- i. El promedio de prácticas de un curso se calcula en base a cuatro prácticas calificadas de las cuales se elimina la nota menor y se promedian las tres notas más altas. Diseñe un algoritmo que determine la nota eliminada y el promedio de prácticas de un estudiante
- j. Diseñe un algoritmo que lea tres longitudes y determine si forman o no un triángulo. Si es un triángulo determine de qué tipo de triángulo se trata entre: equilátero (si tiene tres lados iguales), isósceles (si tiene dos lados iguales) o escaleno (si tiene tres lados desiguales). Considere que para formar un triángulo se requiere que: "el lado mayor sea menor que la suma de los otros dos lados".
- k. Diseñe un algoritmo que lea tres números enteros y determine el menor valor positivo. Si los números positivos son iguales, dar como menor a cualquiera de ellos.
- l. Diseñe un algoritmo para obtener el grado de eficiencia de un operario de una fábrica de tornillos, de acuerdo a las siguientes condiciones, que se le imponen para un período de prueba:
 - Menos de 200 tornillos defectuosos.
 - Más de 10000 tornillos producidos.

El grado de eficiencia se determina de la siguiente manera:

- Si no cumple ninguna de las condiciones, grado 5.
 - Si sólo cumple la primera condición, grado 6.
 - Si sólo cumple la segunda condición, grado 7.
 - Si cumple las dos condiciones, grado 8.
- m. Se cuenta con los votos obtenidos por Juan, Pedro y María en una elección democrática a la presidencia de un club. Para ganar la elección se debe obtener como mínimo el 50% de los votos más 1. En caso que no haya un ganador se repite la elección en una segunda vuelta. Van a la segunda vuelta los dos candidatos que obtengan la más alta votación. Se anula la elección en caso de producirse un empate doble por el segundo lugar o un empate triple. Diseñe un algoritmo que determine el resultado de la elección.
 - n. Diseñe un algoritmo que lea un número entero de 3 cifras, y forme el mayor número posible con las cifras del número ingresado. El número formado debe tener el mismo signo que el número ingresado.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Ciclos

- Desde / hasta
- Mientras
- Repetir

2. Realice el pseudocódigo, para poder resolver los siguientes problemas:

- o. Hacer un pseudocódigo que imprima los números del 1 al 100.
- p. Hacer un pseudocódigo que imprima los números del 100 al 0, en orden decreciente.
- q. Hacer un pseudocódigo que imprima los números pares entre 0 y 100.
- r. Hacer un programa que imprima la suma de los 100 primeros números.
- s. Hacer un pseudocódigo que imprima los números impares hasta el 100 y que imprima cuantos impares hay
- t. Imprimir y contar los múltiplos de 3 desde la unidad hasta un número que introducimos por teclado
- u. Hacer un pseudocódigo que imprima los números del 1 al 100. Que calcule la suma de todos los números pares por un lado, y por otro, la de todos los impares.
- v. Imprimir y contar los números que son múltiplos de 2 o de 3 que hay entre 1 y 100.
- w. Introducir dos números por teclado. Imprimir los números naturales que hay entre ambos números empezando por el más pequeño, contar cuantos hay y cuántos de ellos son pares. Calcular la suma de los impares.
- x. Imprimir diez veces la serie de números del 1 al 10. (Bucle anidado).
- y. Calcular el factorial de un número.
- z. Comprobar si un número mayor o igual que la unidad es primo.
- aa. Introducir un número menor de 5000 y pasarlo a número romano.
- bb. Realizar la tabla de multiplicar de un número entre 0 y 10.

3. Recursividad

- cc. Realice el pseudocódigo que permita de manera recursiva calcular la secuencia Fibonacci.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

4. Corridas de pseudocódigo

dd. Si A = 21 Y B = 89, ¿Cuál es el valor de C al salir del siguiente algoritmo?

C = A / B

C = C + C

Si C > 45 y C < 150 entonces

C = (C * C) / 2

Si no

Si C > 5000 entonces

C = C / 7

Si no

C = 0

Fin si

Fin si

C = (C * C) + su edad

Mostrar(C)

ee. Cuando Z = 34, ¿cuál es el valor de salida de F, del siguiente algoritmo?

Z = Z / Z

F = 0

Mientras Z > 0 hacer

F = F + 3

Z = Z / 2

Si Z = 0,0078125 entonces

Z = Z - 1

Fin si

Mostrar (F)

ff. Cuando H = 90, ¿Cuáles son los valores de salida de tanto G como L?

G=0

L=0

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

$$H = (H/(H-(H/2)))$$

Desde i = 1 hasta H hacer

$$G = H + i$$

$$L = I - i$$

Fin desde

Si G > L entonces

$$G = H$$

$$L = H$$

Si no

$$G = 0 * G$$

$$L = 1 * L$$

Fin si

Mostrar (G)

Mostrar (L)

gg. Si X = 0 y U = 1, ¿cuál es el valor de salida de T del siguiente algoritmo?

$$T = 0$$

$$X = (X + U) * (X-U)$$

Si X > 0 entonces

$$T = T + X$$

Si no

$$T = T (T+T)$$

Fin si

En caso de T

>100

$$T = T + 0$$

<=100 y > 500

$$T = T - T$$

== 1000

$$T = T(X * T * 0)$$

Fin caso

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

hh. Si $G = 0$, $R = 7$, $F = 9$, ¿cuál es el valor de salida de G del siguiente algoritmo?

$G = 9$

Desde ($i = 0$ hasta F , $i++$)

$G = G + (R - F)$

$G = G -. (G - 1)$

Fin Desde

Si $G > 1000$

$G = R * F$

Si no

$R = G * R$

Fin si.

ii. Si $X = 8$, $Y = \text{al doble menos el triple de } X$, $Z = \text{a la potencia de } X$, ¿cuál es el valor de salida de Y del siguiente algoritmo?

Mientras ($x > 0$)

Si $Y > 0$

$Y = Z * X$

$Z = Z + Y$

Si no

$Y = Y(Y - X)$

$Z = Z * X$

Fin Si

$X = X - 1$

Fin mientras

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

JJ. Sea $R = 43$, $T = R * 5$, $S = T - T$, ¿cuál es el valor de salida de las tres variables del siguiente algoritmo?

Si ($T - R - S > 100$) y ($S + R + T < 1000$)

$R = ((R*2) + T) - (S*S*S)$

$T = T + R$

$R = 7;$

Mientras $R \geq 0$

$T = ((T+2) * T - S) * R$

$R = R-1$

Fin mientras

Si no

$R = 7;$

Mientras $R \geq 0$

$T = ((T+2) * T - S) * R$

$R = R-1$

Fin mientras

Fin si

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Introducción a la estructuras de datos

En la práctica, la mayor parte de información útil no aparece aislada en forma de datos simples, sino que lo hace de forma organizada y estructurada. Los diccionarios, guías, encyclopedias, etc., son colecciones de datos que serían inútiles si no estuvieran organizadas de acuerdo con unas determinadas reglas. Además, tener estructurada la información supone ventajas adicionales, al facilitar el acceso y el manejo de los datos. Por ello parece razonable desarrollar la idea de la agrupación de datos, que tengan un cierto tipo de estructura y organización interna.

Como tendremos ocasión de ver, la selección de una estructura de datos frente a otra, a la hora de programar es una decisión importante, ya que ello influye decisivamente en el algoritmo que vaya a usarse para resolver un determinado problema. El objetivo de este capítulo no es sólo la descripción de las distintas estructuras, sino también la comparación de las mismas en términos de utilidad para la programación. De hecho, se trata de dar una idea, acerca de los pros y contras de cada una de ellas con el propósito final de justificar la ya citada ecuación de:

PROGRAMACION = ESTRUCTURAS DE DATOS + ALGORITMOS

CONCEPTO

Empecemos recordando que un dato de tipo simple, no está compuesto de otras estructuras, que no sean los bits, y que por tanto su representación sobre el ordenador es directa, sin embargo existen unas operaciones propias de cada tipo, que en cierta manera los caracterizan. Una estructura de datos es, a grandes rasgos, una colección de datos (normalmente de tipo simple) que se caracterizan por su organización y las operaciones que se definen en ellos. Por tanto, una estructura de datos vendrá caracterizada tanto por unas ciertas relaciones entre los datos que la constituyen (p.a., el orden de las componentes de un vector de números reales), como por las operaciones posibles en ella. Esto supone que podamos expresar formalmente, mediante un conjunto de reglas, las relaciones y operaciones posibles (tales como insertar nuevos elementos o como eliminar los ya existentes). Por el momento y a falta de otros, pensemos en un vector de números, como el mejor ejemplo de una estructura de datos.

Llamaremos dato de tipo estructurado a una entidad, con un solo identificador, constituida por datos de otro tipo, de acuerdo con las reglas que definen cada una de las estructuras de datos. Por ejemplo: una cadena está formada por una sucesión de caracteres, una matriz por datos simples organizados en forma de filas y columnas y un archivo, está constituido por registros, éstos por campos, que se componen, a su vez, de datos de tipo simple. Por un abuso de lenguaje, se tiende a hacer sinónimos, el dato estructurado con su estructura correspondiente.

Para muchos propósitos es conveniente tratar una estructura de datos como si fuera un objeto individual y afortunadamente, muchos lenguajes de programación permiten manipular estructuras completas como si se trataran de datos individuales, de forma que los datos estructurados y simples se consideran a menudo por el programador de la misma manera. Así a partir de ahora un dato puede ser tanto un entero como una matriz, por nombrar dos ejemplos.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Tipos de datos

Los datos de tipo simple tienen una representación conocida en términos de espacio de memoria. Sin embargo, cuando nos referimos a datos estructurados esta correspondencia puede no ser tan directa; por ello vamos a hacer una primera clasificación de los datos estructurados en: contiguos y enlazados. Las estructuras contiguas o físicas son aquellas que al representarse en el hardware del ordenador, lo hacen situando sus datos en áreas adyacentes de memoria; un dato en una estructura contigua se localiza directamente calculando su posición relativa al principio del área de memoria que contiene la estructura. Los datos se relacionan por su vecindad o por su posición relativa dentro de la estructura. Las estructuras enlazadas son estructuras cuyos datos no tienen por qué situarse de forma contigua en la memoria; en las estructuras enlazadas los datos se relacionan unos con otros mediante punteros (un tipo de dato que sirve para ‘apuntar’ hacia otro dato y por tanto para determinar cuál es el siguiente dato de la estructura). La localización de un dato no es inmediata sino que se produce a través de los punteros que relacionan unos datos con otros.

Los datos estructurados se pueden clasificar, también, según la variabilidad de su tamaño durante la ejecución del programa en: estáticos y dinámicos. Las estructuras estáticas son aquellas en las que el tamaño ocupado en memoria, se define con anterioridad a la ejecución del programa que los usa, de forma que su dimensión no puede modificarse durante la misma (p.a., una matriz) aunque no necesariamente se tenga que utilizar toda la memoria reservada al inicio (en todos los lenguajes de programación las estructuras estáticas se representan en memoria de forma contigua). Por el contrario, ciertas estructuras de datos pueden crecer o decrecer en tamaño, durante la ejecución, dependiendo de las necesidades de la aplicación, sin que el programador pueda o deba determinarlo previamente: son las llamadas estructuras dinámicas. Las estructuras dinámicas no tienen teóricamente limitaciones en su tamaño, salvo la única restricción de la memoria disponible en el computador.

Estas dos clasificaciones nos ayudarán a exponer los distintos tipos de datos estructurados, incidiendo en las ventajas e inconvenientes para su almacenamiento y tratamiento, en términos de la eficacia de una determinada aplicación ya sea de economía espacial (no emplear más memoria de la necesaria) o temporal (emplear el menor tiempo posible en las operaciones).

Informática en Desarrollo

CEDES DON BOSCO

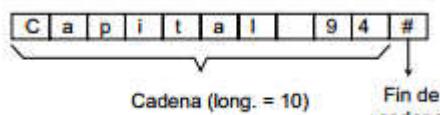
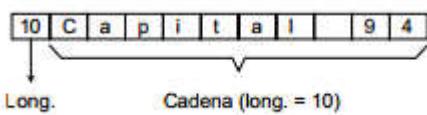
Nivel: Quinto año

ESTRUCTURAS DE DATOS CONTIGUAS

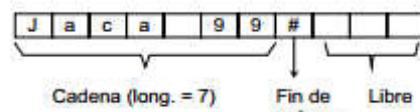
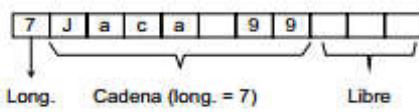
Vamos a estudiar una serie de agrupaciones de datos que son utilizadas en todos los lenguajes de programación, y que tienen en común la ubicación de sus datos en zonas de memoria adyacentes.

Cadenas

La cadena es quizás la estructura más simple y se define como una secuencia de caracteres que se interpretan como un dato único. Su longitud puede ser fija o variable por lo que, además de saber que están constituidas por caracteres alfanuméricos, hemos de conocer su longitud. En una variable tipo cadena se puede almacenar una palabra, una frase, una matrícula de coche, una temperatura, etc. La longitud de una cadena se puede determinar bien indicando al principio de la misma el número de caracteres que contiene, bien situando un carácter especial denominado fin-de-cadena. Los siguientes ejemplos muestran los dos métodos de representar la cadena “Capital 94”:



En el segundo caso el carácter elegido como fin-de-cadena ha sido el #. La cadena que no contiene ningún carácter se denomina cadena vacía y su longitud es 0, que no tiene que ser confundida por una cadena formada sólo por blancos (o espacios), cuya longitud es igual al número de blancos que contiene. De esta manera, una variable de tipo cadena de tamaño 10 puede guardar cadenas de 10 caracteres, pero también de menos si indicamos dónde terminan los caracteres de la cadena. Por ejemplo la cadena “Jaca 99”:



Sobre datos de tipo cadena se pueden realizar las siguientes operaciones:

Asignación: Guardar una cadena en una variable tipo cadena. Como en toda asignación a una variable, la cadena que se guarda puede ser una constante, una variable tipo cadena o una expresión que produzca un dato tipo cadena. Por

Ejemplo:

nombre ← “Pepe”

nombre ← mi-nombre-de-pila

Concatenación: Formar una cadena a partir de dos ya existentes, yuxtaponiendo los caracteres de ambas. Si se denota por // al operador “concatenación”, el resultado de:

Nivel: Quinto año

“ab” // “cd” es “abcd”

Nótese que las constantes de tipo cadena se escriben entre comillas, para no confundirlos con nombres de variables u otros identificadores del programa.

Extracción de subcadena: Permite formar una cadena (subcadena) a partir de otra ya existente. La subcadena se forma tomando un tramo consecutivo de la cadena inicial. Si NOMBRE es una variable de tipo cadena que contiene “JUAN PEDRO ORTEGA” y denotamos por (n:m) la extracción de m caracteres tomados a partir del lugar n, entonces NOMBRE(6:5) es una subcadena que contiene “PEDRO”.

Un caso particular de extracción que se utiliza a menudo es el de extraer un único carácter. Por ello se suele proporcionar un método directo: el nombre seguido por el lugar que ocupa dentro de la cadena. Así, en el ejemplo anterior, NOMBRE (6) = “P” = NOMBRE (6:1)

Obtención de longitud: La longitud de una cadena es un dato de tipo entero, cuyo valor es el número de caracteres que contiene ésta. En el primero de los dos métodos anteriores de representación de cadenas, la longitud se obtiene consultando el número de la primera casilla; en el segundo método la longitud es el número de orden que ocupa el carácter de fin-de-cadena, menos uno.

Comparación de cadenas: Consiste en comparar las cadenas carácter a carácter comenzando por el primero de la izquierda, igual que se consulta un diccionario. El orden de comparación viene dado por el código de E/S del ordenador (ASCII habitualmente). Así la expresión booleana: “José” < “Julio”, se evaluará como verdadera. Nótese que en los códigos de E/S, las mayúsculas y las minúsculas son diferentes, dando lugar a resultados paradójicos en la comparación, así pues, si el código de E/S es ASCII, donde las mayúsculas tienen códigos inferiores a las minúsculas, se cumpliría que “Z” < “a”.

Arreglos

Es un conjunto de datos del mismo tipo almacenados en la memoria del ordenador en posiciones adyacentes. Sus componentes individuales se llaman elementos y se distinguen entre ellos por el nombre del array seguido de uno o varios índices o subíndices. Estos elementos se pueden procesar, bien individualmente, determinando su posición dentro del array, bien como array completo. El número de elementos del array se especifica cuando se crea éste, en la fase declarativa del programa, definiendo el número de dimensiones o número de índices del mismo y los límites máximo y mínimo que cada uno de ellos puede tomar, que llamaremos rango. Según sea este número, distinguiremos los siguientes tipos de arrays:

- unidimensionales (vectores)
- bidimensionales (matrices)
- multidimensionales

Por ello hablaremos de arrays de dimensión 1, 2 o n, cuyo producto por el rango (o rangos) especifica el número de elementos que lo constituyen. Este dato lo utiliza el compilador para

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

reservar el espacio necesario para almacenar en memoria todos sus elementos ocupando un área contigua. Cada elemento ocupa el mismo número de palabras, que será el que corresponda al tipo de éstos. No debemos olvidar que, al nivel físico, la memoria es lineal; por ello los elementos se colocan en la memoria linealmente según un orden prefijado de los índices.

En el ejemplo de la figura el número de dimensiones es 2, su rango (3;5) y el número total de elementos es 15.

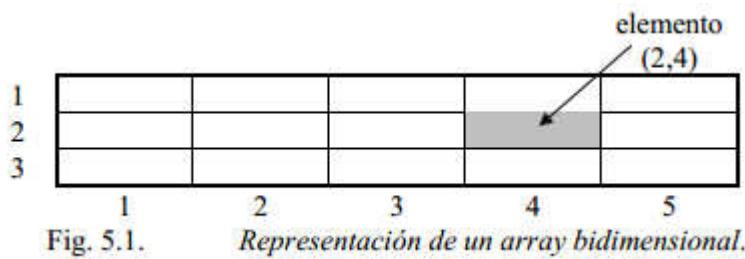


Fig. 5.1. *Representación de un array bidimensional.*

Vectores

Por motivos de simplicidad y mayor frecuencia de uso, a la hora de revisar las operaciones con arrays nos centraremos en los vectores, que además presentan la ventaja de ser estructuras ordenadas (sólo existe orden total cuando tenemos estructuras de una dimensión o lineales). Las notaciones algorítmicas que utilizaremos son: nombre_vector = vector [inf .. sup] de tipo nombre_vector nombre válido del vector inf .. sup límites inferior y superior del rango (valor entero que puede tomar el índice) tipo tipo de los elementos del vector {entero, real, carácter}

La declaración de un vector supone una mera reserva de espacio, pudiéndose asumir que, hasta que asignemos valores por cualquier mecanismo a sus distintos elementos, estamos ante una estructura vacía.

NUMEROS = vector [1..10] de real significa que NUMEROS es un vector, que podrá contener como elementos, al menos 10 números de tipo real, cuyos índices varían desde el 1 hasta el 10.

FORTRAN REAL X(10)	BASIC DIM X(10) AS SINGLE
PASCAL x: array[1..10] of real	C float x[10]

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Es importante señalar que podemos implementar arrays cuyos elementos sean a su vez cadenas o elementos de otro tipo, así podemos pensar en situaciones como la siguiente, durante la fase declarativa tipo: palabra = cadena[16] COCHES = vector [1..9] de palabra lo que nos permitirá manipular en un solo vector hasta 9 cadenas contenido como máximo 16 caracteres cada una. Con lo cual COCHES puede contener una información tal como:

1	Alfa Romeo
2	Fiat
3	Ford
4	Lancia
5	Renault
6	Seat
7	
8	
9	

Ya hemos dicho que las operaciones sobre arrays se pueden realizar con elementos individuales o sobre la estructura completa mediante las correspondientes instrucciones y estructuras de control del lenguaje. Las operaciones sobre elementos del vector son:

Asignación: Tiene el mismo significado que la asignación de un valor a una variable no dimensionada, ya que un vector con su índice representa la misma entidad que una variable no dimensionada.

A[20] ← 5 asigna el valor 5 al elemento 20 del vector A

A[17] ← B asigna el valor de la variable B al elemento 17 del vector A

Acceso secuencial o recorrido del vector: Consiste en acceder a los elementos de un vector para someterlos a un determinado proceso, tal como introducir datos (escribir) en él, visualizar su contenido (leer), etc.. A la operación de efectuar una acción general sobre todos los elementos de un vector se la denomina recorrido y para ello utilizaremos estructuras repetitivas, cuyas variables de control se utilizan como subíndices del vector, de forma que el incremento del contador del bucle producirá el tratamiento sucesivo de los elementos del vector.

Ejemplo 1:

Escribir un algoritmo para recorrer secuencialmente un vector H de 10 elementos (haciendo la lectura y escritura de cada elemento) primero con un bucle desde y luego con un bucle mientras.

El pseudocódigo correspondiente será el siguiente:

desde i ← 1 hasta 10 hacer

 leer (H[i])

 escribir (H[i])

fin_desde

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

i ← 1

mientras i <= 10 hacer

leer (H[i])

escribir (H[i])

i ← i+1

fin-mientras

Ejemplo 2:

Supongamos que queremos procesar los primeros elementos de un vector PUNTOS, previamente declarado, realizando las siguientes operaciones desde 1 hasta LIMITE (donde este valor debe necesariamente ser menor que el límite superior del rango): a) lectura del array; b) cálculo de la suma de los valores del array; c) cálculo de la media de los valores. Escribir el algoritmo correspondiente.

inicio

escribir 'número de datos'

leer numero

suma← 0

escribir 'datos del array'

desde i=1 hasta numero hacer

leer PUNTOS[i]

suma← suma + PUNTOS[i]

fin_desde

media ← suma/numero

escribir 'la media es', media

fin

BÚSQUEDA EN UN VECTOR: Esta operación, relacionada con la recuperación de información, consiste en encontrar un determinado valor dentro del vector, obteniendo su posición en el mismo en caso que éste exista o declarar la búsqueda como fallida en caso de no encontrarlo. La experiencia a la hora de buscar un dato entre una colección de ellos nos dice que este proceso varía sensiblemente según que éstos estén o no ordenados; por esta razón presentaremos dos métodos básicos, según que el vector esté desordenado u ordenado.

Búsqueda secuencial: Consiste en recorrer el vector, con sus datos no necesariamente ordenados, del principio hacia el final. Si se encuentra el valor buscado se da por finalizada la

Nivel: Quinto año

búsqueda; en caso contrario, tras haber recorrido todo el vector se indica que el elemento en cuestión no se encuentra almacenado en el vector. Supongamos que deseamos localizar un determinado valor, que, obviamente, supondremos es del mismo tipo que los elementos del vector de rango

n. El algoritmo es el siguiente: algoritmo búsqueda_secuencial

A: vector donde se busca (contiene n elementos)

t: valor buscado

inicio

encontrado ← Falso

i ← 1

mientras (i <= n) y (encontrado = Falso) hacer

si t = A[i]

entonces

escribir 'Se encontró el elemento buscado en la posición', i

encontrado ← Verdadero

si_no

i ← i + 1

fin_si

fin_mientras

si i = n+1 {también puede utilizarse si encontrado = Falso}

entonces

escribir 'No se encuentra el elemento'

si_no

escribir 'El elemento se encuentra en la posición', i

fin_si

fin

Una manera más eficaz de realizar una búsqueda secuencial consiste en modificar el algoritmo utilizando un elemento centinela. Este elemento se agrega al vector al final del mismo. El valor del elemento centinela es el del argumento. El propósito de este elemento centinela, A[n+1], es significar que la búsqueda siempre tendrá éxito. El elemento A[n+1] sirve como centinela y se le asigna el valor de t antes de iniciar la búsqueda. En cada paso se evita la comparación de i con n

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

y por consiguiente este algoritmo será preferible al método anterior. Si el índice alcanzase el valor $n+1$, supondrá que el argumento no pertenece al vector original y en consecuencia la búsqueda no tiene éxito.

algoritmo búsqueda_centinela

A: vector donde se busca (contiene n elementos); t: valor buscado

inicio

encontrado ← Falso

i ← 1

A[n+1] ← t

mientras A(i) <> t hacer

i ← i + 1

fin_mientras

si i = n+1

entonces

escribir 'No se encuentra el elemento'

si_no

entonces

escribir 'El elemento se encuentra en la posición', i

fin_si

fin

Notemos que, a pesar de esta mejora, el número de comparaciones que hemos de efectuar es del orden de la magnitud del vector y si ésta es muy grande el tiempo necesario para la búsqueda puede ser alto. Por ello, puede valer la pena tener ordenado el vector pues, como veremos ahora, las búsquedas sobre vectores ordenados son sensiblemente más rápidas.

Búsqueda binaria: Se aplica a vectores cuyos datos han sido previamente ordenados y es un ejemplo del uso del 'divide y vencerás' para localizar el valor deseado. Más adelante, trataremos los algoritmos (y su coste) para ordenar un vector; sin embargo ahora supondremos que esta ordenación ya ha tenido lugar. El algoritmo de búsqueda binaria se basa en los siguientes pasos:

- 1) Examinar el elemento central del vector; si éste es el elemento buscado, entonces la búsqueda ha terminado

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

2) En caso contrario, se determina si el elemento buscado está en la primera o en la segunda mitad del vector (de aquí el nombre de binario) y a continuación se repite este proceso, utilizando el elemento central del subvector correspondiente.

Ejemplo 3:

Considerar el siguiente vector ordenado de nueve elementos enteros, donde queremos buscar el valor 2983

1	2	3	4	5	6	7	8	9
2473	2545	2834	2892	2898	2983	3005	3446	3685

central

Para buscar el elemento 2983, se examina el número central 2898, situado en la quinta posición, que resulta ser distinto. Al ser 2983 mayor que 2898, se desprecia la primera mitad del vector, quedándonos con la segunda:

6	7	8	9
2983	3005	3446	3685

central

Se examina ahora el número central 3005, situado en la posición 7, que resulta ser distinto. Al ser 2983 menor que 3005, nos quedamos con la primera mitad del vector:

6
2983

central

Finalmente encontramos que el valor buscado coincide con el central. Nótese que si el valor buscado hubiera sido, por ejemplo, el 2900, la búsqueda habría finalizado con fracaso al no quedar mitades donde buscar.

Vamos a dar el algoritmo de búsqueda binaria para encontrar un elemento K, en un vector de elementos X(1), X(2),..., X(n), previamente clasificados en orden ascendente (ordenado en orden creciente si los datos son numéricos, o alfabéticamente si son caracteres). El proceso de búsqueda debe terminar normalmente conociendo si la búsqueda ha tenido éxito (se ha encontrado el elemento) o bien no ha tenido éxito (no se ha encontrado el elemento), y normalmente se deberá devolver la posición del elemento buscado dentro del vector. Las variables enteras BAJO, CENTRAL, ALTO, indican los límites inferior, central y superior del intervalo de búsqueda, en cada subvector que sucesivamente se está considerando, durante la búsqueda binaria.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

algoritmo búsqueda_binaria

X: vector de N elementos

K: valor buscado

inicio

BAJO \leftarrow 1

ALTO \leftarrow N

CENTRAL \leftarrow ent((BAJO + ALTO) / 2)

mientras BAJO < ALTO y X[CENTRAL] \neq K hacer

si K < X[CENTRAL]

entonces

ALTO \leftarrow CENTRAL - 1

si_no

BAJO \leftarrow CENTRAL + 1

fin_si

CENTRAL \leftarrow ent ((BAJO + ALTO) / 2)

fin_mientras

si K = X(CENTRAL)

entonces escribir 'valor encontrado en', CENTRAL

si_no escribir 'valor no encontrado'

fin_si

fin

La función ent, entenderemos que obtiene un entero redondeándolo por defecto.

El funcionamiento del algoritmo de búsqueda binaria, en un vector de enteros, se ilustra en la Figura 5.2 para dos búsquedas: los enteros 8 y 11 que finalizan respectivamente con éxito (localizado el elemento) y sin éxito (no encontrado el elemento)

Aunque dejamos para un capítulo posterior el análisis de la complejidad algorítmica, con el único objeto de ilustrar la diferencia entre los dos tipos de búsqueda, digamos que para un vector de 1000 elementos, la búsqueda secuencial supone efectuar un promedio de unas 500 comparaciones, mientras que la binaria, resuelve el problema con sólo 10 en el peor de los casos. Otra cuestión, es el esfuerzo que nos suponga ordenar el vector, pero esto lo veremos más adelante.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

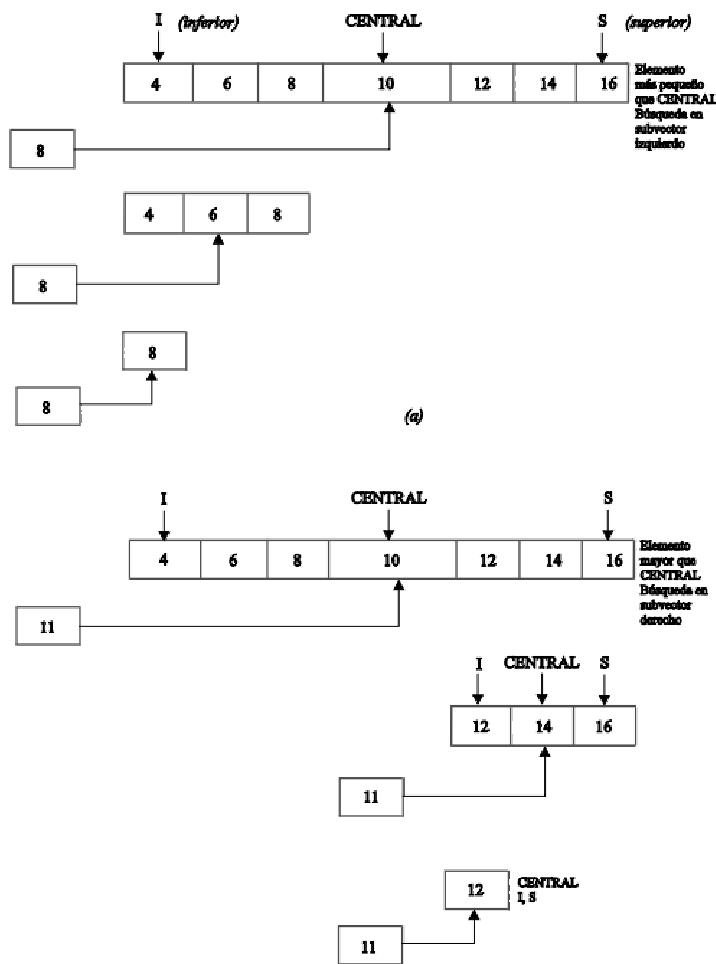


Fig. 5.2. Ejemplo de búsqueda binaria: (a), con éxito; (b), sin éxito.

Insertar datos en un vector

La operación insertar consiste en colocar un nuevo elemento, en una determinada posición del vector; ello supone no perder la información, que pudiera hallarse anteriormente, en la posición que va a ocupar, el valor a insertar.

Es condición necesaria para que esta operación pueda tener lugar, la comprobación de espacio de memoria suficiente en el vector, para el nuevo valor; dicho de otro modo, que el vector no tenga todos sus elementos ocupados por datos. Cada vez que insertamos un dato en una determinada posición hemos de correr todos los elementos posteriores del vector, hacia abajo, poniendo especial cuidado en que no se pierda ninguno de estos datos.

Ejemplo 4:

Consideremos el ya visto vector COCHES de 9 elementos que contiene 7 marcas de automóviles, en orden alfabético, en el que se desea insertar dos nuevas marcas OPEL y CITRÖEN manteniendo el orden alfabético del vector. Como Opel está comprendido entre Lancia y Renault, se deberán desplazar hacia abajo los elementos 5 y 6, que pasarán a ocupar la posición

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

relativa 6 y 7. Posteriormente debe realizarse la misma operación con Citröen que ocupará la posición 2. El algoritmo que realiza esta operación para un vector de n elementos es el siguiente, suponiendo que hay espacio suficiente en el vector, y que conocemos la posición, que designaremos por P, que debe ocupar el elemento a insertar. Por ejemplo, la primera inserción, supone que Opel debe ocupar la posición P = 5.

1. i n ← {Indicar el número de posiciones del vector que ya están ocupadas}

2. mientras $i >= P$ hacer

{mover el elemento actual i-ésimo hacia abajo, a la posición $i+1$ }

COCHES[i+1] ← COCHES[i]

{decrementar contador}

$i \leftarrow i - 1$

fin_mientras

3. {insertar el elemento en la posición P}

COCHES[P] ← "Opel"

4. {actualizar el contador de elementos del vector}

5. $n \leftarrow n + 1$ {el vector acaba con un elemento más ocupado por datos}

6. fin

Los contenidos sucesivos del vector COCHES son los siguientes:

1	Alfa Romeo
2	Fiat
3	Ford
4	Lancia
5	Renault
6	Seat
7	
8	
9	

1	Alfa Romeo
2	Fiat
3	Ford
4	Lancia
5	Opel
6	Renault
7	Seat
8	
9	

1	Alfa Romeo
2	Citröen
3	Fiat
4	Ford
5	Lancia
6	Opel
7	Renault
8	Seat
9	

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Eliminar datos de un vector

La operación de borrar es distinta, según el elemento a eliminar se encuentre al final del vector (no presenta ningún problema) o se borre un elemento del interior de mismo vector. En este último la eliminación provocara el movimiento hacia arriba de los elementos posteriores a él para reorganizar el vector. El algoritmo de borrado del dato almacenado en el elemento j-ésimo del vector COCHES es el siguiente: algoritmo borrado {borrar el elemento j-ésimo}

```
inicio
desde i = j hasta N - 1
{llevar elemento i + 1 hacia arriba}
COCHES[i] ← COCHES[i+1]
fin-desde
{actualizar contador de elementos}
{ahora COCHES tendrá ocupado por datos un elementos menos, n - 1}
N ← N - 1
Fin
```

Matrices

Llamaremos matriz o tabla a un array bidimensional, esto es, un conjunto de elementos del mismo tipo en el que sus componentes vienen definidos por dos subíndices, el primero referido a la fila y el segundo a la columna. Además de las matrices, que utilizamos en álgebra lineal, un ejemplo típico de esta estructura es el tablero de ajedrez, que se representa por un array T[8,8]. Se puede representar la posición o casilla (recuadro) de cada tablero del siguiente modo:

T[i,j] = 0 si no existe ninguna pieza en la fila i-esima y columna j-esima
T[i,j] = 1 si la casilla contiene un peón blanco
= 2 para un caballo blanco
= 3 para alfil blanco
= 4 para una torre blanca
= 5 para una reina blanca
= 6 para un rey blanco con los correspondientes números negativos para las piezas negras.

Puesto que la memoria del ordenador no está organizada en forma rectangular sino en forma de una enorme fila para almacenar una matriz hemos de recurrir a guardar las filas una a continuación de otra. Así, si en una matriz con un número de columnas C, que es el número de elementos que tiene cada fila, para localizar el elemento de fila i y columna j, tendríamos que movernos, desde la posición de inicio de la matriz, hasta la posición: C*(i-1) + j.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Afortunadamente el propio software del sistema se encarga de convertir los términos en forma de filas y columnas, en localizaciones concretas dentro de la memoria, de forma que podemos pensar conceptualmente en forma de tabla, aunque se almacenen en forma de fila, dentro de la máquina.

Arreglos Multidimensionales

Dependiendo del tipo de lenguaje, pueden existir arrays de tres o más dimensiones (por ejemplo FORTRAN 77 admite hasta siete dimensiones). Para el caso de tres dimensiones, la estructura puede visualizarse como un cubo, y para mayor número de dimensiones ésta visualización no es posible.

El tratamiento de estos arrays es similar al de las matrices, cada conjunto de índices individualiza un elemento de la estructura, que se almacena en memoria de forma secuencial.

Registros

Hasta ahora nos hemos referido a estructuras formadas por datos simples del mismo tipo; sin embargo, es interesante poder manejar una especie de arrays heterogéneos en los que sus elementos puedan ser de tipos diferentes. Llamaremos registro a una estructura de datos, formada por yuxtaposición de elementos que contienen información relativa a un mismo ente. A los elementos que componen el registro los llamamos campos, cada uno de los cuales es de un determinado tipo, simple o estructurado. Los campos dentro del registro aparecen en un orden determinado y se identifican por un nombre. Para definir el registro es necesario especificar el nombre y tipo de cada campo. Por ejemplo consideremos un registro, referido a Empleado, que está constituido por tres campos: Nombre (cadena), Edad (entero) y Porcentaje de impuestos (real).

Nombre	Edad	Porcentaje de impuestos

Las operaciones básicas que se ejecutan con los registros son: asignación del registro completo a una variable de tipo registro, (definida con sus mismos campos) y selección de un campo, que se realiza especificando el nombre del campo. Puesto que esta estructura, está especialmente ligada a las transferencias con los periféricos de almacenamiento, volveremos sobre ella cuando nos refiramos a los archivos.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Prácticas generales de datos contiguos:

1. En Visual Studio 2013, realice un programa que lea una palabra de la pantalla, luego muestre la cantidad de letras que tiene dicha palabra.
2. En Visual Studio 2013, realice un programa que lea una frase de la pantalla, divida la misma cada vez que se encuentra la letra “a”, guárde lo en un vector de string, muestre todos los componentes del vector en pantalla.
3. En Visual Studio 2013, realice un programa que lea una frase de la pantalla, luego se le pedirá al usuario una letra, debe buscar la primera coincidencia de la letra en la frase, debe mostrar la posición de dicha letra en la frase.
4. En Visual Studio 2013, realice un programa que cree un vector de enteros, rellene el vector con los números que digite el usuario.
5. En Visual Studio 2013, realice un programa que cree un vector de enteros, rellene el vector con una estructura tipo “for”.
6. En Visual Studio 2013, realice un programa que cree un vector de enteros, rellene el vector con los números que digite el usuario, cree un segundo vector del mismo tamaño, e ingrese todos los enteros del primer vector en el segundo pero de mayor a menor.
7. En Visual Studio 2013, realice un programa que cree un vector de enteros, rellene el vector con los números que digite el usuario, no permita que el usuario repita números.
8. En Visual Studio 2013, realice un programa que cree una matriz de N números, rellena la misma con datos que el usuario digite.
9. En Visual Studio 2013, realice un programa que cree una matriz de N números, rellene la matriz utilizando estructuras tipo “for”.
10. (Extra) En Visual Studio 2013, realice un programa que cree una matriz de N números, otra segunda matriz con N números, guarde en una tercera matriz el resultado de la multiplicación de las dos primeras matrices.

Primer Proyecto

Buscaminas, en Windows Form y con matrices, a divertirse...

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Estructuras dinámicas y punteros

Hasta este momento hemos venido trabajando con variables, dimensionadas o no, que son direcciones simbólicas de posiciones de memoria, de forma que existe una relación bien determinada entre nombres de variables y posiciones de memoria durante toda la ejecución del programa. Aunque el contenido de una posición de memoria asociada con una variable puede cambiar durante la ejecución, es decir el valor asignado a la variable puede variar, las variables por si mismas no pueden crecer ni disminuir, durante la ejecución. Sin embargo, en muchas ocasiones es conveniente poder disponer de un método por el cual, podamos adquirir posiciones de memoria adicionales, a medida que las vayamos necesitando durante la ejecución y al contrario liberarlas cuando no se necesiten.

Las variables y estructuras de datos que reúnen estas condiciones se llaman dinámicas y se representan con la ayuda de un nuevo tipo de dato, llamado puntero, que se define como un dato que indica la posición de memoria ocupada por otro dato. Puede concebirse como una flecha, que “apunta”, al dato en cuestión (Ver Figura 5.3).

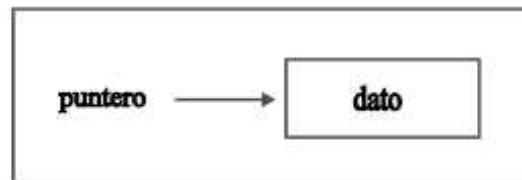


Fig. 5.3 Representación de un puntero

Los punteros proporcionan los enlaces de unión entre los elementos, permitiendo que, durante la ejecución del programa, las estructuras dinámicas puedan cambiar sus tamaños. En las estructuras dinámicas estos elementos, llamados nodos, son normalmente registros de al menos dos campos, donde por lo menos uno de ellos, es un puntero (es decir contiene información que permite localizar al siguiente - siguientes- nodo de la estructura). Ver Figura 5.4.



Fig. 5.4. Representación de un nodo

La utilización de punteros permite que sea relativamente fácil añadir indeterminadamente nuevos datos, insertar nuevos nodos en otros ya existentes y en general modificar estas estructuras. Dependiendo de las relaciones entre los nodos de la estructura hablaremos de estructuras lineales y no lineales: si partiendo del nodo inicial es posible dirigirse sucesivamente

Nivel: Quinto año

a todos los nodos visitando cada uno una única vez diremos que es una estructura lineal; en caso de no ser posible el recorrido en estas condiciones se habla de estructura no lineal.

Estructuras lineales

Una lista en su sentido amplio, es un conjunto de datos del mismo tipo (simple o estructurado), cada elemento de la cual tiene un único predecesor (excepto el primero) y un único sucesor (excepto el último) y cuyo número de elementos es variable. Se distinguen dos tipos de listas: contiguas y lineales. La lista contigua es una estructura intermedia entre las estáticas y las dinámicas, ya que sus datos se almacenan en la memoria del ordenador en posiciones sucesivas y se procesan como vectores. Con esta disposición secuencial, el acceso a cualquier elemento de la lista y la adición por los extremos de nuevos elementos es fácil, siempre que haya espacio para ello. Para que una lista contigua pueda variar de tamaño y por tanto, dar la impresión de que se comporta como una estructura dinámica, hay que definir un vector dimensionado con tamaño suficiente para que pueda contener todos los posibles elementos de la lista. Como se observará, una lista contigua es un vector que tiene posiciones libres por delante y detrás y el índice del mismo hace de puntero. Es un caso relativamente banal de estructura dinámica, al cual no prestaremos mayor atención.

Listas enlazadas

Estas listas están formadas por un conjunto de nodos, en los que cada elemento contiene un puntero con la posición o dirección del siguiente elemento de la lista, es decir su enlace. Se dice entonces, que los elementos de una lista están enlazados por medio de los campos enlaces. Cada nodo está constituido por dos partes: información o campos de datos (uno o varios) y un puntero (con la dirección del nodo siguiente). Al campo o campos de datos del nodo lo designaremos como INFORMACIÓN del nodo y al puntero por SIGUIENTE del nodo. Con esta organización de los datos, es evidente que no es necesario que los elementos de la lista estén almacenados en posiciones físicas adyacentes para estar relacionados entre sí, ya que el puntero indica únicamente la posición del dato siguiente en la lista.

Una forma alternativa de ver las listas enlazadas, es considerarlas como una estructura que contienen un dato dado como la cabeza, y el resto como la cola. La notación usual es la siguiente: (A | B), donde la lista tiene el elemento A en la cabeza y el elemento B como cola. Recursivamente, (A | (B | (C D))) es la lista con el elemento A en la cabeza, mientras que la cola está formada por una lista con el elemento B en la cabeza y una lista que contiene los elementos C y D en la cola, etc.

Nótese que para tener definida una lista enlazada, además de la estructura de cada uno de sus nodos, necesitamos una variable externa a la propia lista, con un puntero que marque la posición de la cabeza (inicio, primero) de la lista. Esta variable es quien normalmente da nombre a la lista, pues nos dice donde localizarla, sea cual sea su tamaño. Para detectar el último elemento de la lista se emplea un puntero nulo, que por convenio, se suele representar de diversas formas: por un enlace con la palabra reservada nil (NULO), por una barra inclinada (/) (Ver Figura 5.5) o por un signo especial, tomado de la toma de tierra en electricidad. Una lista enlazada sin ningún

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

elemento se llama lista vacía y las distinguiremos asignando a su puntero de cabecera el valor nil.

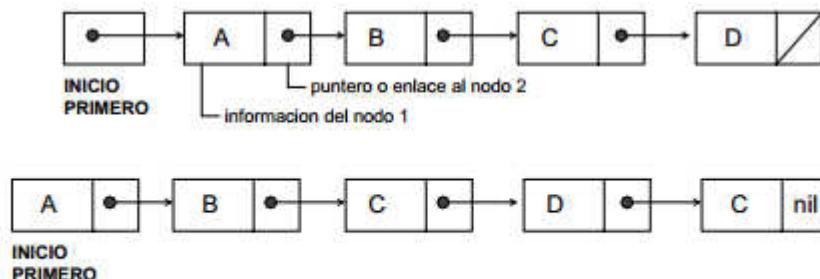
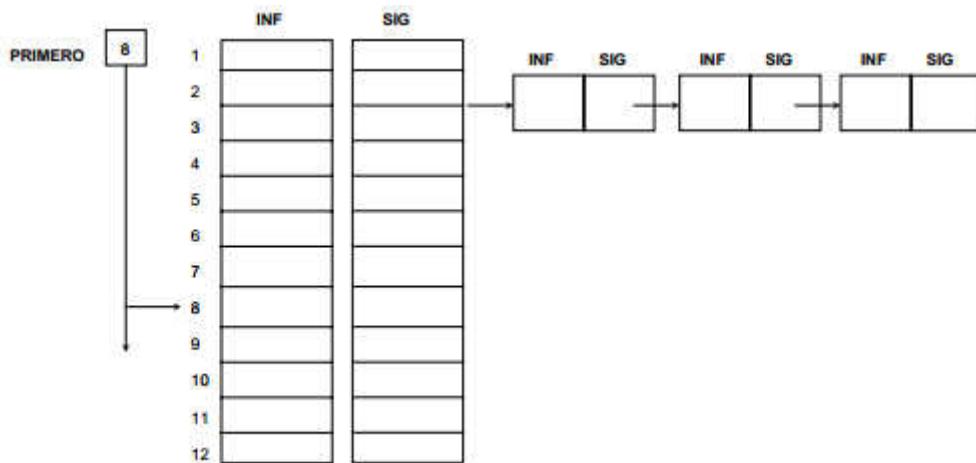


Fig. 5.5. Ejemplos de listas enlazadas

Creación de una lista

La implementación de una lista enlazada, depende del lenguaje de programación, ya que no todos soportan el puntero como tipo de datos; afortunadamente los más modernos como C, Pascal, etc., si lo hacen, por lo que en lo que sigue vamos a asumir el uso de punteros para su manejo. La alternativa, al uso de punteros, es recurrir a vectores paralelos en los que se almacenan los datos correspondientes a INFORMACION y SIGUIENTE, con una variable que apunte al índice que contiene la cabeza de la lista, de forma que nos podemos imaginar el siguiente esquema de equivalencia, entre ambas estructuras físicas:



Una vez definida la estructura de sus nodos, cosa que dejamos aparte, pues dependerá de cada lenguaje de programación, crear una lista consiste en llenar un primer nodo con la información correspondiente a un elemento y cuyo campo de enlace sea NULO. Además hay que definir la variable con el puntero externo que debe contener la dirección de este nodo inicial. Fijarse que ello supone que el lenguaje de programación que utilicemos debe tener una función que nos de la posición de memoria en la que se ha almacenado este nodo. A partir de este nodo inicial la lista empieza a modificarse, creciendo y disminuyendo, insertando y borrando nodos.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Procesamiento de listas enlazadas

El objetivo de lo que sigue, no es tanto la descripción de las operaciones de procesamiento del ejemplo más significativo de estructura dinámica, como la comparación de éstas con las estáticas, en términos de utilidad para la programación. Se trata por tanto solamente de dar una idea, acerca de los pros y contras de utilizar vectores o listas enlazadas como estructuras lineales. En esta línea, vamos ahora a ver como efectuamos, con listas enlazadas, las mismas cuatro operaciones (recorrido, búsqueda, inserción y eliminación) que ya hemos llevado a cabo utilizando vectores, como estructura de datos.

Vamos a utilizar las notaciones algorítmicas siguientes: PRIMERO es un puntero externo al primer nodo de una lista enlazada;

P es un puntero a un nodo cualquiera de la lista;

NODO(P) el nodo apuntado por P.

INFO(P) campo de datos del nodo apuntado por P;

SIG(P) campo puntero del nodo apuntado por P (apunta al nodo siguiente).

Recorrido de una lista

Recorrer un vector fue fácil con una estructura repetitiva, al poder utilizar su índice para poder ir del principio al final de la estructura. Sin embargo, en el caso de las listas, al carecer de índice, puede pensarse que no es tan fácil acceder directa o aleatoriamente a sus nodos, ya que para ello es necesario acceder al primer nodo mediante el puntero externo, al segundo, después de acceder al primero, etc. Afortunadamente, esto se resuelve, usando una variable puntero, auxiliar X, que apunte en cada momento al nodo procesado, de forma que con solo hacer la asignación: $X \leftarrow SIG(X)$, esto es guardar en X el campo puntero del nodo (Ver Figura 5.6) se tiene el efecto de avanzar hacia el siguiente nodo de la lista.

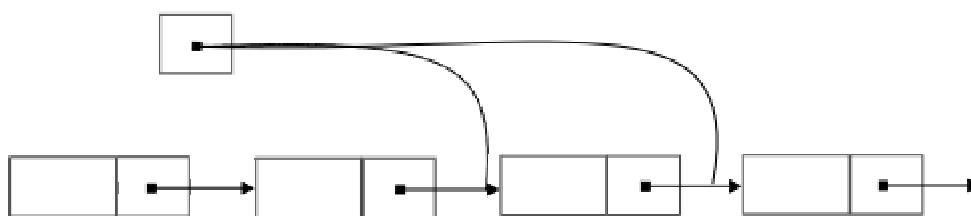


Fig. 5.6 Asignación $X \leftarrow SIG(X)$

Basándonos en lo anterior, el recorrido de una lista, es posible utilizando el puntero temporal P. Así el siguiente algoritmo, nos permite leer la lista completa.

1. inicio
2. $P \leftarrow PRIMERO$

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

3. mientras P <> nil hacer
4. escribir INFO(P)
5. P ← SIG(P)
6. fin_mientras
7. fin

El puntero P contiene el valor del primer elemento de la lista. El bucle recorre toda la lista hasta que encuentra un nodo cuyo siguiente sea NULO, en cuyo caso se efectuaría: ($P \leftarrow NULO$), que corresponde al último de la lista. La línea 4 escribe el valor de cada elemento y la línea 5 avanza el puntero P, dándole el valor del campo SIG del nodo actual.

Ejemplo 5:

Escribir un algoritmo que recorra una lista enlazada para calcular el número de elementos que la componen (que en principio es indeterminado):

```
inicio
    N ← 0 {contador de elementos}
    P ← PRIMERO
    mientras P <> nil hacer
        N ← N + 1
        P ← SIG(P)
    fin_mientras
fin
```

Búsqueda en una lista

En el caso de trabajar con un vector, la búsqueda variaba sensiblemente según que éste estuviera o no ordenado. En el caso de una lista enlazada, la búsqueda debe hacerse mediante un recorrido de la misma, elemento a elemento, hasta o bien encontrar el elemento deseado o bien detectar el final de la lista, aunque en el caso de que la lista este ordenada podemos dar por terminada la búsqueda sin éxito, cuando en este recorrido, nos encontramos con un nodo, cuya información es posterior al valor buscado. Veamos estos dos casos, con listas desordenadas y ordenadas.

Dada una lista enlazada cualquiera cuyo primer nodo está apuntado por PRIMERO, el siguiente procedimiento busca un elemento t obteniendo un puntero POS que lo apunta (si no se encuentra el elemento POS debe ser NULO).

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

procedimiento BusquedaDesordenada(PRIMERO, t, REF POS)

{Elemento buscado: t}

{puntero al lugar que ocupa: POS}

inicio

P \leftarrow PRIMERO

POS \leftarrow NULO

mientras P \neq NULO hacer

si t= INFO [P]

entonces

POS \leftarrow P

P \leftarrow NULO

sino

P \leftarrow SIG [P]

fin-si

fin-mientras

si POS = NULO entonces escribir 'búsqueda sin éxito' {esto es opcional}

fin

En caso de que la lista está ordenada en orden ascendente, el algoritmo de búsqueda en las condiciones arriba indicadas es el siguiente.

procedimiento BusquedaOrdenada (PRIMERO, t,REF POS)

inicio {se supone la lista ordenada}

P \leftarrow PRIMERO

POS \leftarrow NULO

mientras P \neq nulo hacer

si INFO[P] < t

entonces

P \leftarrow SIG[P]

si_no si t = INFO [P]

entonces

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
POS ← P
P ← NULO
fin-si
fin-si
fin-mientras
si POS = NULO entonces escribir 'búsqueda sin éxito'
fin
```

Como podemos observar, la búsqueda en listas sin ordenar resulta muy parecida a la que se da en el caso de vectores. Sin embargo para listas enlazadas ordenadas, al no poder conocer el tamaño que tiene la estructura en cada momento, no podemos explotar todas las posibilidades que nos proporcionaban los vectores ordenados con la búsqueda binaria.

Inserción de un elemento

Insertar, al igual que borrar, consiste básicamente en modificar los punteros de la lista. Vamos a hacer el siguiente planteamiento general: Sea una lista enlazada en la que se desea insertar un nuevo nodo, cosa que puede hacerse bien al principio de la lista, bien a continuación de un nodo específico. La inserción al principio de la lista es directa (Ver Figura 5.7)

En cualquier inserción necesitamos disponer de un nodo vacío, donde depositar la información que queremos añadir, previamente al propio proceso de inserción.

Llamaremos DISPO a una lista de nodos disponibles, para guardar el nodo antes de ser insertado en la lista.

Veamos como se efectúa la inserción de un nodo, con la información ELEMENTO, al principio de la lista.

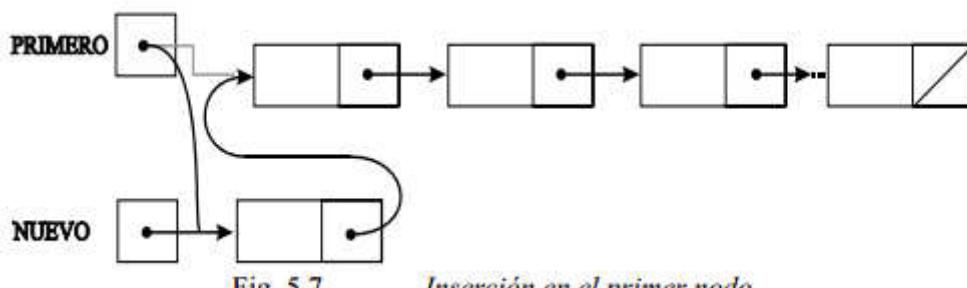


Fig. 5.7.

Inserción en el primer nodo

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

algoritmo inserción

NUEVO \leftarrow DISPO {obtiene un nuevo nodo si es posible; si no, da NULO}

si NUEVO = NULO {comprobación de desbordamiento} entonces

escribir "desbordamiento"

sino

INFO(NUEVO) \leftarrow ELEMENTO

{Colocamos en el campo INFORMACIÓN del nuevo nodo los

datos}

SIG(NUEVO) \leftarrow PRIMERO

{El puntero del nuevo nodo apunta a la cabeza de la lista original}

PRIMERO \leftarrow NUEVO

{El nuevo nodo es ahora la nueva cabeza de la lista}

fin_si

fin

La inserción de un nuevo nodo (cuya información llamaremos NOMBRE) a continuación de un nodo dado apuntado por P, exige la utilización de un puntero auxiliar Q, aparte de los ya utilizados P y NUEVO. Veamos su algoritmo (Ver Figura 5.8) asumiendo que habrá disponibilidad de nodo (no hay desbordamiento) con la misma nomenclatura que en la Figura 5.7.:

1. NUEVO \leftarrow DISPO
2. INFO(NUEVO) \leftarrow NOMBRE
3. Q \leftarrow SIG(P)
4. SIG(P) \leftarrow NUEVO
5. SIG(NUEVO) \leftarrow Q

Después de los pasos 1 y 2, tendremos la estructura mostrada en la Figura 5.8 (a).

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

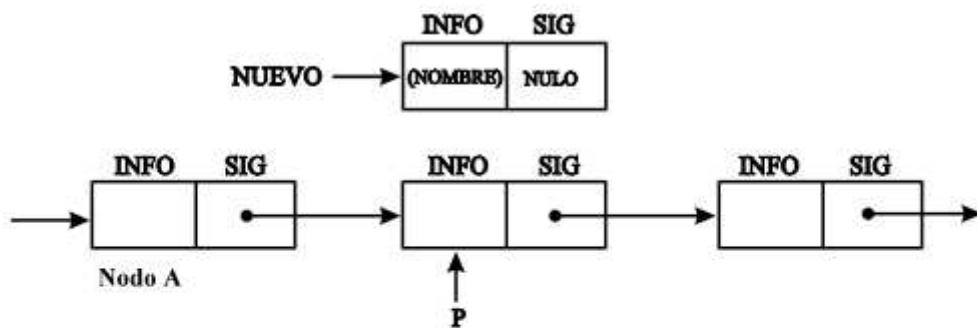


Fig. 5.8 (a)

Tras los pasos 3 y 4, se tiene la estructura mostrada en la Figura 5.8 (b)

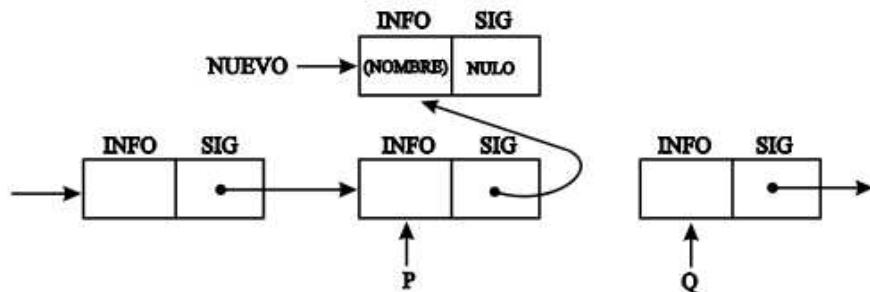


Fig. 5.8 (b)

Finalmente tras la ejecución de la última instrucción, la situación será la mostrada en la Figura 5.8 (c)

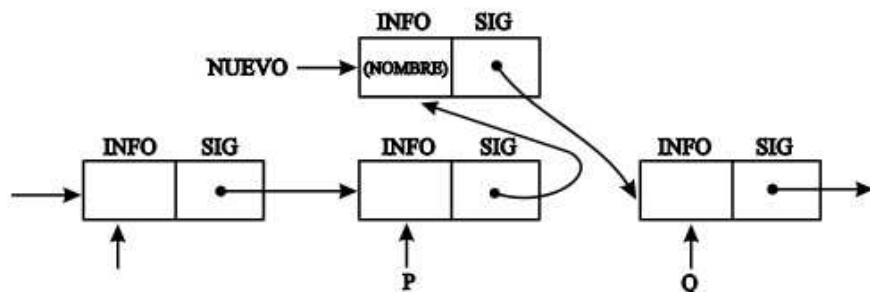


Fig. 5.8 (c)

Nótese que el puntero **Q** puede evitarse ya que los pasos 3, 4 y 5 son reemplazables por:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

$SIG(NUEVO) \leftarrow SIG(P)$

$SIG(P) \leftarrow NUEVO$

Sin embargo, recurrir a Q nos será útil durante el posible proceso de búsqueda del nodo a partir del cual insertar.

Eliminación de un elemento de la lista

Esta operación consiste en hacer que el nodo anterior, al nodo que quiere eliminarse, se enlace con el posterior a éste, con lo cual el nodo que nos interesa quedará fuera de la lista. Consideremos la lista enlazada de la Figura 5.9(a). El algoritmo que sigue elimina de la lista enlazada el elemento siguiente al apuntado por P, utilizando un puntero auxiliar Q:

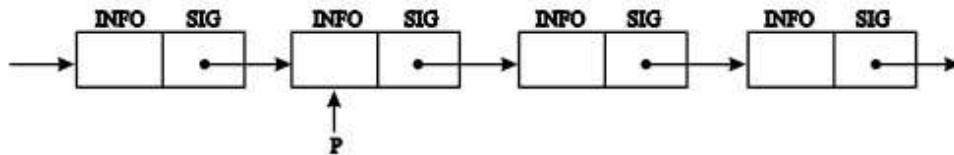


Fig. 5.9 (a) Lista donde se desea eliminar el elemento apuntado por $SIG(P)$

1. $Q \leftarrow SIG(P)$
2. $SIG(P) \leftarrow SIG(Q)$
3. LIBERARNODO(Q)

Tras cada uno de estos tres pasos, la lista sufre los siguientes pasos:

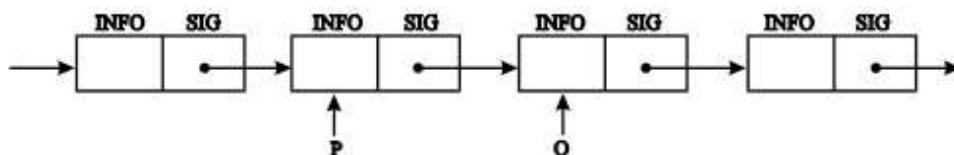


Fig. 5.9 (b) Situación tras el paso 1

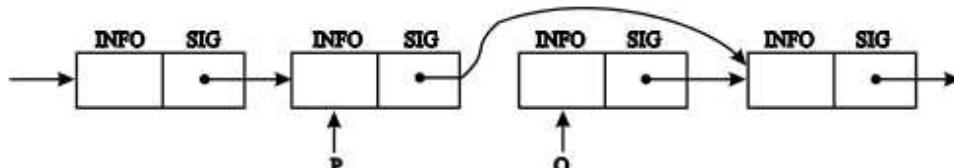


Fig. 5.9 (c) Situación tras el paso 2

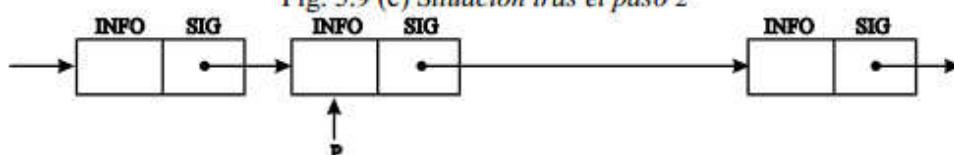


Fig. 5.9 (d) Situación tras el paso 3

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Notemos que a su vez, el proceso de borrado dará lugar a la existencia de nodos libres o disponibles, al liberar el nodo que se elimina; este espacio de memoria podrá ser utilizado al invocar DISPO.

Pilas

Para introducir esta estructura, recordemos la forma en que se apilan los platos en los restaurantes: una pila de platos se soporta sobre un muelle, cuando se retira un plato, los demás suben. Vamos a trasladar esta idea a la informática. Una pila (stack) es una estructura lineal a cuyos datos sólo se puede acceder por un solo extremo, denominado tope o cima (top). En esta estructura sólo se pueden efectuar dos operaciones: añadir y eliminar un elemento, acciones que se conocen por meter (push), y sacar (pop). Si se meten varios elementos en la pila y después se sacan de ésta, el último elemento en entrar será el primero en salir. Por esta razón se dice que la pila es una estructura en la que el último en entrar es el primero en salir, en inglés, LIFO (last in first out).

Cuando se almacena una pila en la memoria de un ordenador, los elementos realmente no se mueven arriba y abajo, a medida que se meten o sacan de la pila.

Simplemente es la posición del tope de la pila la que varía. Un puntero, denominado, puntero de pila, indica la posición del tope o, lo que es lo mismo, el primer elemento disponible en la cima. Otro puntero se emplea para determinar la base de la pila que mantiene el mismo valor mientras existe la pila. La Figura 5.10 muestra el uso del puntero de la pila y la base de ésta. Si se realiza la secuencia de operaciones: sacar, sacar y meter 5.9, el estado resultante de la pila aparece en la Figura 5.11. Para representar una pila vacía, el puntero de la pila tiene el mismo valor que la base de la pila.

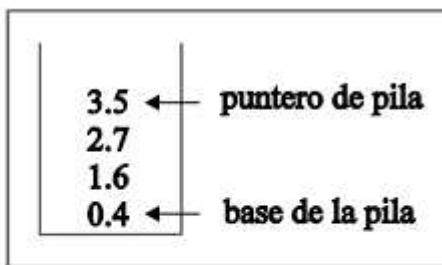


Fig. 5.10 Pila

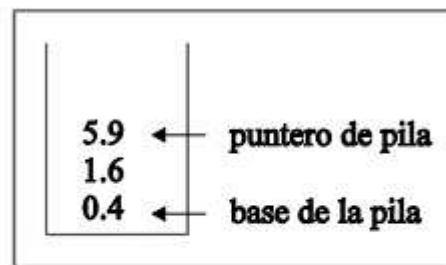


Fig. 5.11. Estado de la pila de la Fig. 5.10 tras las operaciones:
sacar, sacar, meter 5.9

La pila es una de las estructuras más importantes en computación, se usa para calcular expresiones, para pasar de un lenguaje de ordenador a otro y para transferir el control de una parte de un programa a otra. Como ejemplo de uso de la pila, considérese un programa que llama a la función raíz cuadrada de un número. El subprograma recibe el argumento cuya raíz ha de calcularse y retorna la raíz ya calculada. Este argumento se saca de la pila para utilizarlo y una vez calculada la raíz, cuando la función termina, mete el resultado en la pila. El último dato

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

entrado, argumento, es el primer dato sacado e igual ocurre con el resultado de la función que es el primer dato utilizado, cuando el control retorna al programa.

Procesamiento de una pila

Para trabajar con pilas, hay que contar con procedimientos para meter y sacar elementos y para comprobar si la pila está vacía (puede utilizarse una variable o función booleana VACIA, de modo que cuando su valor sea verdadero la pila está vacía, y falso en caso contrario).

Las pilas pueden implementarse utilizando memoria estática (por medio de vectores) o dinámica (utilizando punteros). Vamos a estudiar la implementación de una pila utilizando un vector S, de tamaño LONGMAX, dejando al lector la implementación dinámico de una pila como caso particular de lista enlazada.

Utilizaremos las siguientes notaciones:

p = CIMA puntero de la pila

LONGMAX longitud máxima de la pila

S(i) elemento i-ésimo de la pila S

X elemento a añadir/quitar de la pila

VACIA función booleana “pila vacía”

PUSH subprograma para añadir o meter elementos

POP subprograma para eliminar o sacar elementos

Veamos cuál es el algoritmo de los procedimientos PUSH, POP y de la función booleana VACIA

Meter el dato x en la pila (push):

inicio

si p = LONGMAX

entonces

escribir ‘pila llena’

sino

$p \leftarrow p + 1$

$S(p) \leftarrow x$

fin_si

fin

Sacar un dato de la pila (pop) y ponerlo en x:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

inicio

{este subprograma pone el valor de x en la cima de la pila}

si $p = 0$

entonces

escribir 'pila vacía'

sino

$x \leftarrow S(p)$

$p \leftarrow p - 1$

fin_si

fin

Función Pila vacía (vacía)

inicio

si $p = 0$

entonces

VACIA \leftarrow cierto

sino

VACIA \leftarrow falso

fin_si

fin

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Colas

A pesar de los orígenes no europeos de muchas de las ideas asociadas con los ordenadores, esa importante institución británica, la cola, ha encontrado su lugar en las ciencias de la computación. Todo el mundo sabe como funciona una cola, los recién llegados se sitúan al final, mientras que la desaparición se hace por el principio, sin que esté permitido “colarse”. Vamos a definir cola, como una estructura lineal, en la que los datos entran por la parte de atrás y salen por la de delante. Una cola es una estructura en la que el primer dato en entrar es el primer dato en salir, es decir, una estructura FIFO (first in, first out).

Hay varias formas de implementar una cola en la memoria de un ordenador, bien con vectores, bien en listas enlazadas. En cualquier caso se necesitan dos variables que representan a los punteros FRENTE ($f = \text{front}$) y FINAL ($r = \text{rear}$). El estado de COLA VACIA se manifiesta cuando f y r son ambas nulos en la implementación dinámica o cuando coinciden en el caso estático.

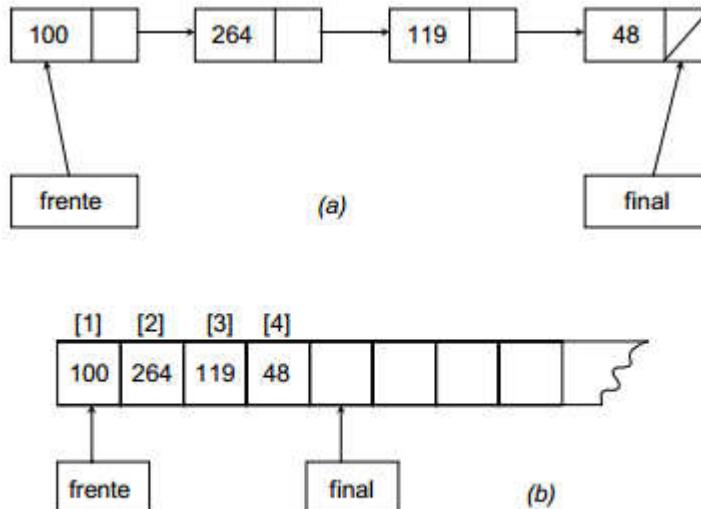


Fig.5.12 *Ejemplos de Implementación de una cola:
a) con una lista enlazada. b) con un vector*

Las colas se utilizan algo menos que las pilas, por lo que no insistiremos en las operaciones que facilitan su procesamiento. Sin embargo digamos que facilitan la interconexión y el almacenamiento de datos en tránsito tanto en redes de ordenadores, como entre un procesador y un periférico (así por ejemplo, en los trabajos para imprimir, decimos que están en cola de impresión, por orden de llegada).

Como se puede observar, la cola puede considerarse en la implementación dinámica como un caso particular de lista enlazada. Se deja al lector la escritura de los algoritmos de inserción y borrado de un elemento y la comprobación de cola vacía, tanto en el caso estático como en el dinámico.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Prácticas generales de Estructuras lineales

1. Investigue en internet, la forma en que se implementa una lista enlazada en el lenguaje c#, escriba un ejemplo explicativo.
2. Investigue en internet, la forma en que se implementa una Pila en el lenguaje c#, escriba un ejemplo explicativo.
3. Investigue en internet, la forma en que se implementa una cola en el lenguaje c#, escriba un ejemplo explicativo.
4. Busque un ejemplo de inserción de un dato en una lista enlazada, en cualquier lenguaje, escriba y explique el código.
5. Desarrolle un programa que utilice punteros, en c#.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Estructuras no lineales (Arboles)

Existen en Informática varias estructuras no lineales, en las que un elemento puede estar relacionado con más de uno sea por delante o detrás de él (por ejemplo, los grafos); no obstante, dentro del carácter introductorio de este curso, nosotros nos restringiremos a la más sencilla de ellas, la de árbol.

A todos nos son familiares expresiones como “árbol genealógico” o “recorrer un árbol”. En este sentido, un árbol es una estructura que implica una jerarquía, en la que cada elemento está unido a otros por debajo de él. Comparada con las estructuras lineales anteriores, el árbol tiene la particularidad de que cada elemento puede tener más de un “siguiente”, aunque un solo antecedente o padre.

Definiremos árbol, de forma recursiva, como un conjunto finito de uno o más nodos, de tal manera, que exista un nodo especial denominado raíz y los nodos restantes están divididos en conjuntos denominados subárboles, que también responden a la estructura de un árbol. Por extensión a la idea de árbol genealógico se habla de nodos padres y nodos hijo y un nodo, en la parte inferior del que no cuelgue ningún subárbol (no tiene ningún hijo) se denomina nodo terminal u hoja (véase Figura 5.13).

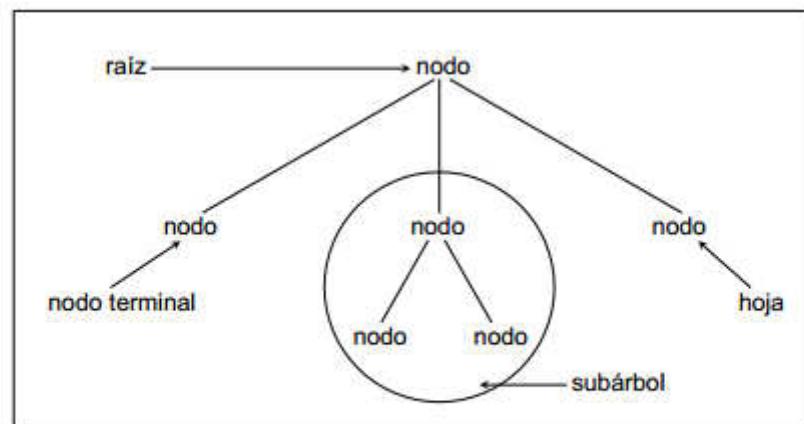


Fig. 5.13. Conceptos relacionados con los árboles

Arboles Binarios

Hay un tipo especial de árbol muy usado en computación, denominado árbol binario, en el que de cada nodo pueden colgar, a lo más, dos subárboles. Estos se denominan subárbol derecho y subárbol izquierdo, y también son árboles binarios. La Figura 5.14 representa un árbol binario.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

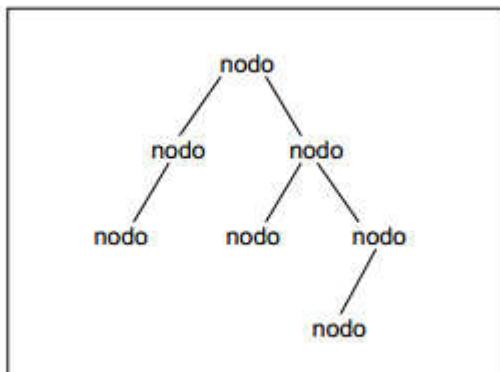


Fig. 5.14. Un árbol binario

La forma usual de representar los árboles supone el uso de punteros, aunque también se puede hacer con vectores. En un árbol binario cada nodo está constituido por una parte de datos (INFORMACION) y dos punteros que indican la posición de sus hijos. Uno o ambos de los punteros pueden tener un valor nulo si del nodo no cuelgan subárboles. Cada nodo de un árbol será un registro que contiene al menos tres campos:

- un campo **INFORMACION** de datos de un cierto tipo.
 - un puntero al nodo del subárbol izquierdo (que puede ser nulo).
 - un puntero al nodo del subárbol derecho (que puede ser nulo).

La Figura 5.15 ilustra el uso de los punteros para construir el mismo árbol representado en la Figura 5.14. Nótese cómo los punteros de los nodos terminales son punteros nulos.

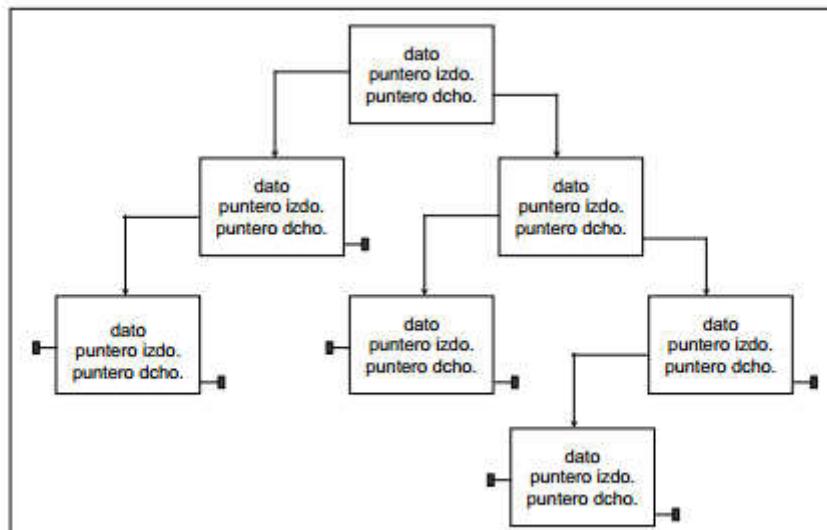


Fig. 5.15. Punteros empleados para construir un árbol binario

Nivel: Quinto año

Recorrido de un árbol binario

Recorrer un árbol supone acceder a sus elementos de forma sistemática lo que supone llevar a cabo tres actividades:

1. Visitar el nodo raíz
2. Recorrer el subárbol izquierdo
3. Recorrer el subárbol derecho

Estas tres acciones repartidas en diferentes órdenes proporcionan diferentes recorridos del árbol, llamados: pre-orden, post-orden, in-orden. Su nombre refleja el momento en que se visita el nodo raíz. En el “in-orden” el raíz está en el medio del recorrido, en el “pre-orden”, el raíz está el primero y en el “post-orden”, el raíz está el último:

Recorrido pre-orden

1. Visitar el raíz
2. Recorrer el subárbol izquierdo en pre-orden
3. Recorrer el subárbol derecho en pre-orden

Recorrido in-orden

1. Recorrer el subárbol izquierdo en in-orden
2. Visitar el raíz
3. Recorrer el subárbol derecho en in-orden

Recorrido post-orden

1. Recorrer el subárbol izquierdo en post-orden
2. Recorrer el subárbol derecho en post-orden
3. Visitar el raíz

Obsérvese que todas estas definiciones de recorrido tienen naturaleza recursiva.

Ejemplo 6:

Determinar cual sería el resultado de los tres posibles recorridos para cada uno de los tres árboles mostrados en la Figura 5.16

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

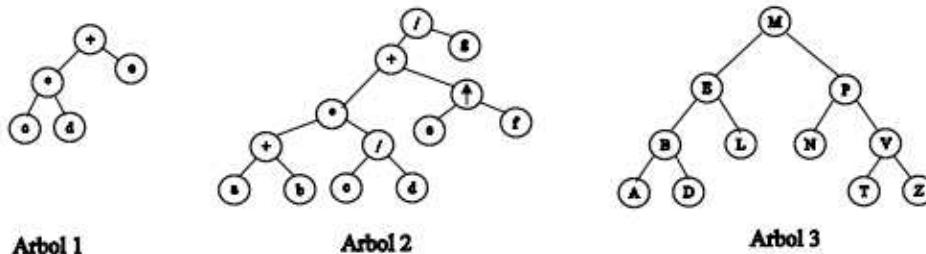


Fig. 5.16 Recorrido de árboles binarios

Árbol 1

Pre-orden
In-orden
Post-orden

+ * cde
c * d + e
cd * e +

Árbol 2

Pre-orden
In-orden
Post-orden

/ + * + ab / cd ↑ efg
a + b * c/d + e ↑ f/g
ab + cd/*ef ↑ + g

Árbol 3

Pre-orden
In-orden
Post-orden

mebadlpnvzt
abdelmnptvz
adblentzvpm

La característica esencial de los árboles es que cada uno de sus nodos puede estar conectado a subárboles, que a su vez tiene estructura arbórea. En otras palabras, siempre que se está en un árbol, la estructura inferior tiene carácter de árbol. En este sentido un árbol es una estructura de datos recursiva, que puede manipularse mediante programas recursivos. Esta propiedad de los árboles es la que los hace más interesantes desde un punto de vista informático y por ello se utilizan ampliamente, como por ejemplo: los módulos de muchos programas se enlazan como si de árboles se tratara, la estructura que emplean muchos sistemas operativos para manejar los ficheros son árboles, algunos ordenadores se refieren a su memoria como si ésta estuviera fragmentada en forma de árbol; asimismo acabamos de ver cómo los árboles se usan para representar operaciones aritméticas.

Árbol binario de búsqueda

Ya hemos hecho notar que una de las aplicaciones que se dan más frecuentemente en informática, es la de manejar una colección de datos sobre los cuales se efectúan de forma constante operaciones de búsqueda inserción y borrado (pensemos por ejemplo en el trabajo habitual de un servicio de reserva en una agencia de viajes).

A lo largo de este capítulo hemos visto que cada una de estas tres operaciones elementales se resuelven de forma distinta, según la estructura elegida para organizar esta colección de datos sea un vector o una lista enlazada. Sabemos que la inserción y el borrado se mejoran sensiblemente, si elegimos una lista en vez de un vector, por el contrario, para la búsqueda parece claramente preferible trabajar con un vector ordenado. Para superar esta situación, vamos a describir una variante del árbol binario con la que podemos localizar, insertar y borrar

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

con mayor eficacia. Ello nos dará una nueva posibilidad a la hora de programar, al poder seleccionar nuevas estructuras, que nos permitan utilizar nuevos y mejores algoritmos.

Llamaremos árbol binario de búsqueda a un árbol binario construido de acuerdo con el procedimiento siguiente:

1. El primer elemento se utiliza para crear el nodo raíz.
2. Los valores del árbol deben ser tales que pueda existir un orden (entero, real, lógico o carácter e incluso definido por el usuario si se tiene un orden total con él).
3. En cualquier nodo, todos los valores del subárbol izquierdo son menores o iguales que el valor del nodo. De modo similar todos los valores del subárbol derecho deben ser mayores que los valores del nodo.

Para este tipo de árbol, es sencillo probar que su recorrido “in-orden” obtiene los valores debidamente ordenados, lo que nos será de gran utilidad. Así, por ejemplo, en la Figura 5.17 se muestra un árbol binario de búsqueda.

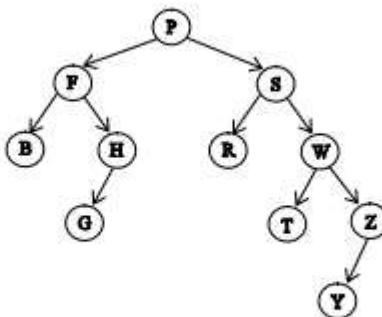


Fig. 5.17 Árbol binario de búsqueda

El recorrido in-orden del árbol de la figura 5.17 es: B F G H P R S T W Y Z. Ello nos permitirá almacenar y procesar un conjunto ordenado, con bastante facilidad, sin que tengamos que proceder a largas operaciones de readaptación. El paso de un conjunto cualquiera a un árbol binario de búsqueda es afortunadamente fácil, como muestra el ejemplo siguiente:

Ejemplo 7:

Supongamos que se dispone de un vector que contiene los siguientes caracteres: D F E B A C G para expresarlo mediante un árbol binario de búsqueda, vamos a seguir el algoritmo:

1. Nodo raíz del árbol: D.
2. El siguiente elemento se convierte en el descendiente derecho, dado que F alfabéticamente es mayor que D.

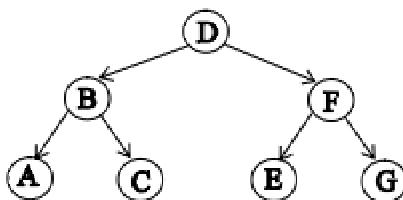
Nivel: Quinto año

3. A continuación, se compara E con el raíz. Dado que E es mayor que D, pasará a ser un hijo de F y como E < F será el hijo izquierdo.

4. El siguiente elemento B se compara con el raíz D y como B < D y es el primer elemento que cumple esta condición, B será el hijo izquierdo de D.

5. Se repiten los pasos hasta el último elemento.

El árbol binario de búsqueda resultante sería:



Nótese que en caso de que el vector estuviese previamente ordenado (ascendente o descendente) el árbol de búsqueda que obtendríamos sería en realidad una lista.

Una vez definida esta nueva estructura tratemos de constatar su utilidad, cuando nos enfrentamos a esta hipotética aplicación, en la que las tres operaciones de búsqueda, inserción y borrado son muy frecuentes. Por motivos de espacio, no vamos a bajar hasta la especificación completa de los algoritmos correspondientes a cada una de estas tres operaciones, sólo presentaremos un esbozo del diseño correspondiente.

Búsqueda de un elemento

La búsqueda en un árbol binario ordenado es dicotómica, ya que a cada examen de un nodo, se elimina aquel de los subárboles que no contiene el valor buscado (valores todos inferiores o todos superiores). El algoritmo de búsqueda del elemento -llamado clave- se realiza comparándolo con la raíz del árbol. Si no es el mismo, se pasa el subárbol izquierdo o derecho según sea el resultado de la comparación y se repite la búsqueda en ese subárbol, de forma recursiva.

La terminación del procedimiento se producirá cuando:

- se encuentra la clave
- no se encuentra la clave; y se llega a encontrar un subárbol vacío.

Así por ejemplo, si buscamos "W" en el árbol de la figura 5.17, se visitarán los nodos "P", "S", "W". Si, en el mismo árbol, buscamos "X", se visitarán los nodos "P", "S", "W", "Z", "Y" y se alcanza un subárbol vacío bajo la "Y" sin encontrar la clave.

Insertar un elemento

Para insertar un elemento en un árbol hay que comprobar en primer lugar que aquel no se encuentre ya en el árbol, dado que en este caso no precisa ser insertado (de hecho ésta es una

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

comprobación que también deberíamos hacer trabajando con vectores o listas enlazadas, lo que supone que cada inserción se acompaña, en cierta manera, de un proceso de búsqueda). Si el elemento no existe, la inserción se realiza en un nodo en el que al menos uno de los dos punteros IZQ o DER tenga valor nil, con lo cual el nuevo elemento de inserta como una nueva hoja del árbol, sea cual sea su valor (Ver Figura 5.18).

La inserción no varía mucho del propio proceso de búsqueda, pues realmente vamos a insertar el nodo en la posición que ésta ocuparía, si ya se encontrara en el árbol. Para ello, se desciende en el árbol a partir del nodo raíz, dirigiéndose de izquierda a derecha de un nodo, según que el valor a insertar sea inferior o superior al valor del campo INFO de este nodo. Cuando se alcanza un nodo del árbol en que no se puede continuar, el nuevo elemento se engancha a la izquierda o derecha de este nodo, en función de que su valor sea inferior o superior al del nodo alcanzado.

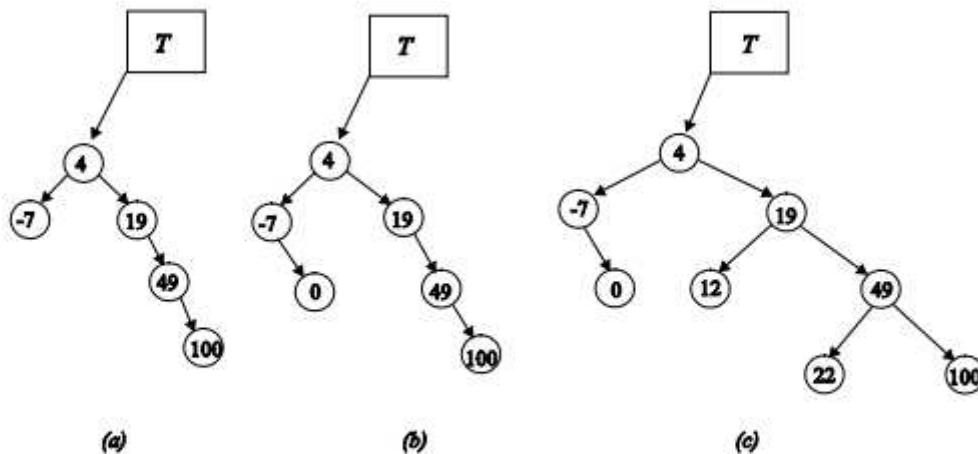


Fig. 5.18 Ejemplo de Inserciones en un árbol de búsqueda binaria: (a), insertar 100; (b), insertar (o); (c), insertar 22 y 12.

Eliminación de un elemento

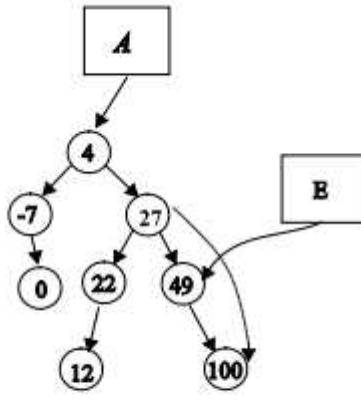
La eliminación de un elemento debe hacerse conservando el orden de los elementos del árbol. Se consideran diferentes casos según la posición del elemento o nodo en el árbol a eliminar:

- si el elemento es una hoja, se suprime simplemente; si el elemento no tiene más que un descendiente, se sustituye entonces por ese descendiente (ver Figura 5.19);
- si el elemento tiene dos descendientes, la situación es un poco más complicada ya que la simple sustitución de un nodo por uno de sus hijos conllevaría la pérdida de estructura de árbol binario. En este caso el nodo a eliminar se debe sustituir por un descendiente más a la derecha o a la izquierda, de modo que siga conservando la ordenación (ver Figura 5.20).

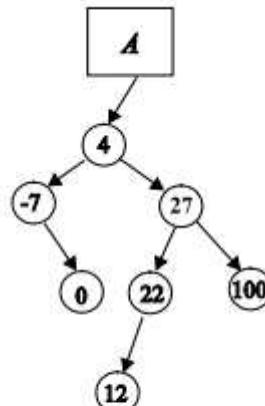
Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



(a)

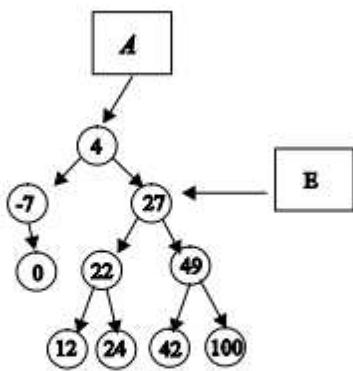


(b)

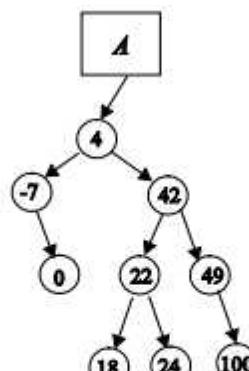
Fig. 5.19 Ejemplo de Eliminación de un nodo (49) con un solo subárbol
a) antes de la eliminación b) después de la eliminación

Para poder realizar estas acciones, será preciso conocer la siguiente información del nodo a eliminar:

- Su posición en el árbol;
- La dirección de su padre;
- La dirección relativa a su ascendencia, es decir si el nodo a eliminar es un hijo derecho o izquierdo.



(a)



(b)

Fig. 5.20: Ejemplo de eliminación de un nodo (27) con dos subárboles no nulos. Se ha sustituido por el elemento 42 que es un nodo sucesor siguiente en orden ascendente.

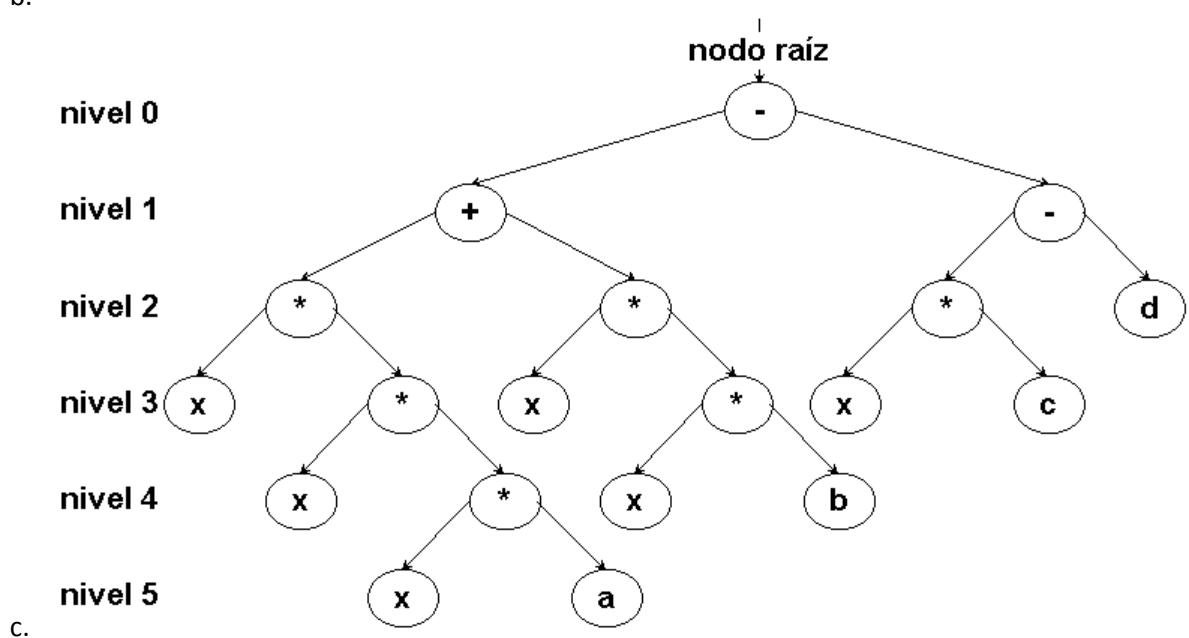
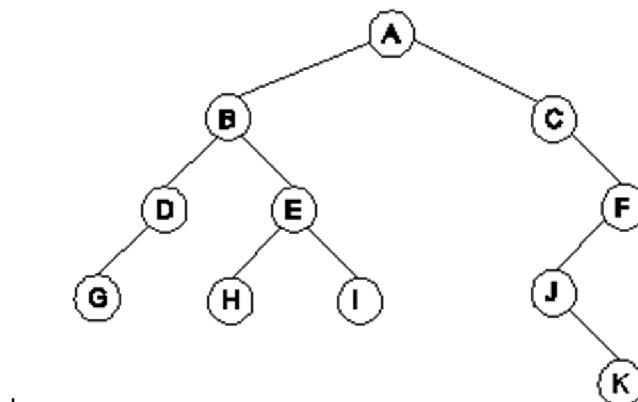
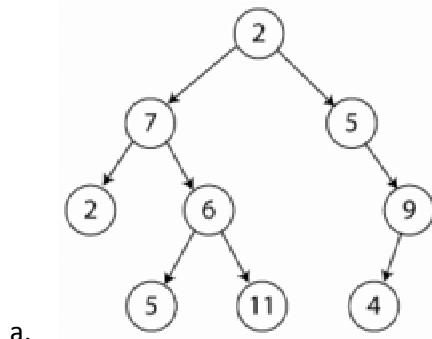
Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Prácticas generales de Estructuras no lineales

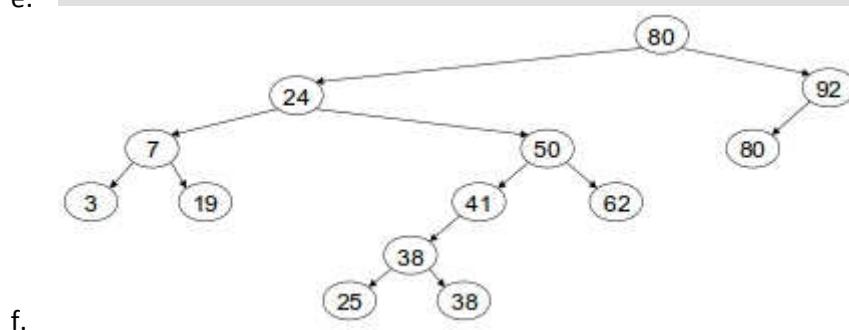
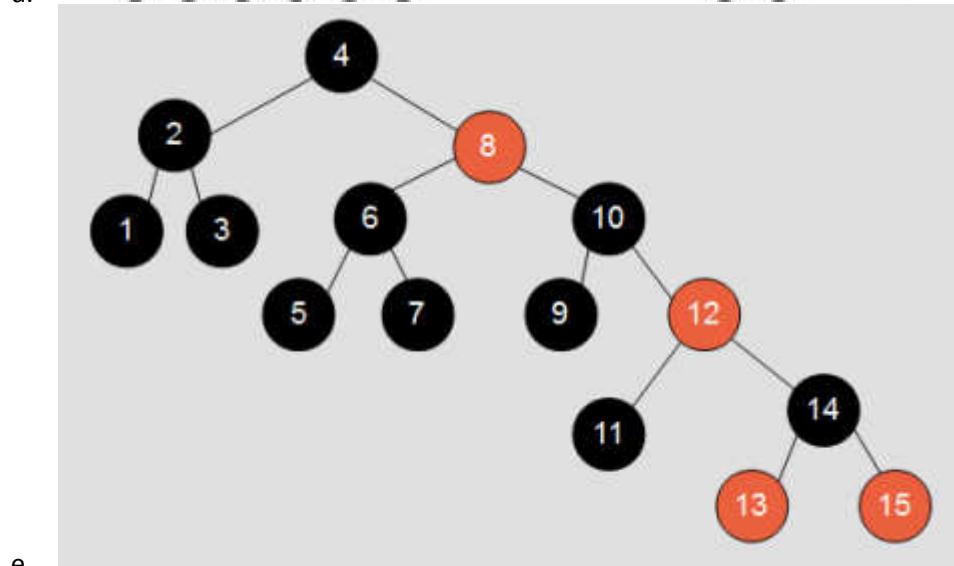
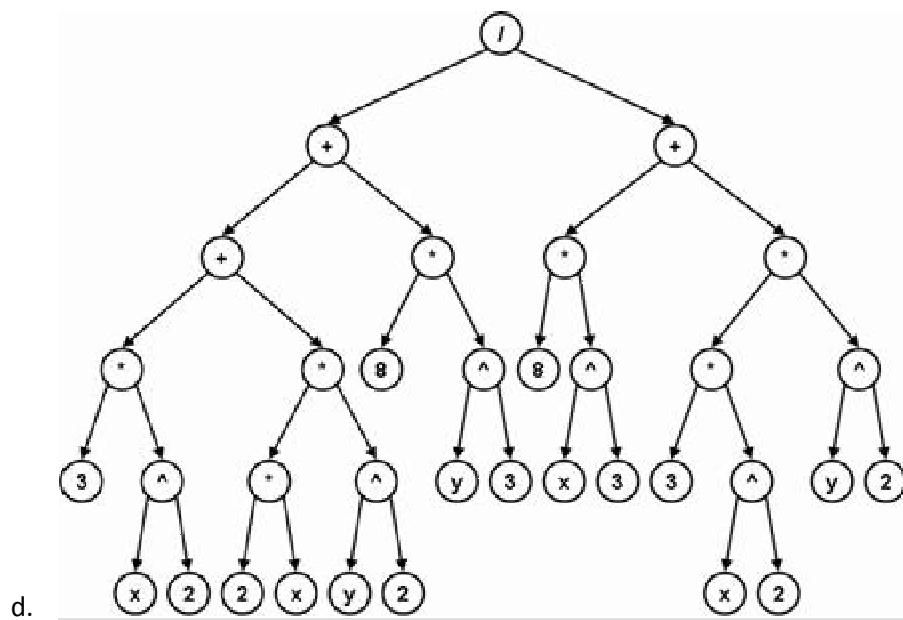
- Escriba el preorden, inorder y postorden de los siguientes árboles binarios



Informática en Desarrollo

CEDES DON BOSCO

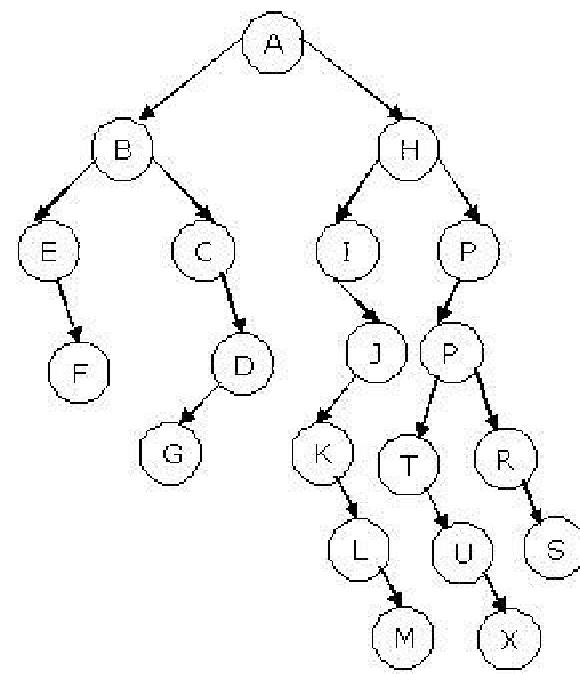
Nivel: Quinto año



Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



g.

2. Confeccione los siguientes arboles binarios de búsqueda

- a. 56,78,3,4,21,76,23,0,90,34,21,55
- b. t,g,y,e,r,w,a,i,n,h,m,f
- c. IV,V,III,I,X,XX,XI,IIV,VI,XXI
- d. M, Y, T, E, R
- e. T, Y, M, E, R
- f. R,E,M, Y, T
- g. C, Q, R, N F, L, A, K, E S

Fin del tema de estructuras de datos, inicio del Segundo proyecto, ajedrez...

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Introducción al Algebra relacional

Esta unidad didáctica está dedicada al estudio del modelo de datos relacional y del álgebra relacional.

El concepto de modelo de datos se ha presentado en otra unidad didáctica. En ésta se profundiza en un modelo de datos concreto: el modelo relacional, que actualmente tiene una gran relevancia. Sus conceptos fundamentales están bien asentados y, además, los sistemas de gestión de bases de datos relacionales son los más extendidos en su utilización práctica. Por estos motivos pensamos que es importante conocerlo.

El estudio del modelo relacional sirve, además, de base para los contenidos de otra unidad, dedicada al lenguaje SQL. Este lenguaje permite definir y manipular bases de datos relacionales. Los fundamentos del modelo relacional resultan imprescindibles para conseguir un buen dominio del SQL.

En esta unidad se analizan también las operaciones del álgebra relacional, que sirven para hacer consultas a una base de datos. Es preciso conocer estas operaciones porque nos permiten saber qué servicios de consulta debe proporcionar un lenguaje relacional. Otra aportación del álgebra relacional es que facilita la comprensión de algunas de las construcciones del lenguaje SQL que se estudiarán en otra unidad didáctica de este curso. Además, constituye la base para el estudio del tratamiento de las consultas que efectúan los SGBD internamente (especialmente en lo que respecta a la optimización de consultas).

Este último tema queda fuera del ámbito del presente curso, pero es relevante para estudios más avanzados sobre bases de datos.

Visión informal de una relación

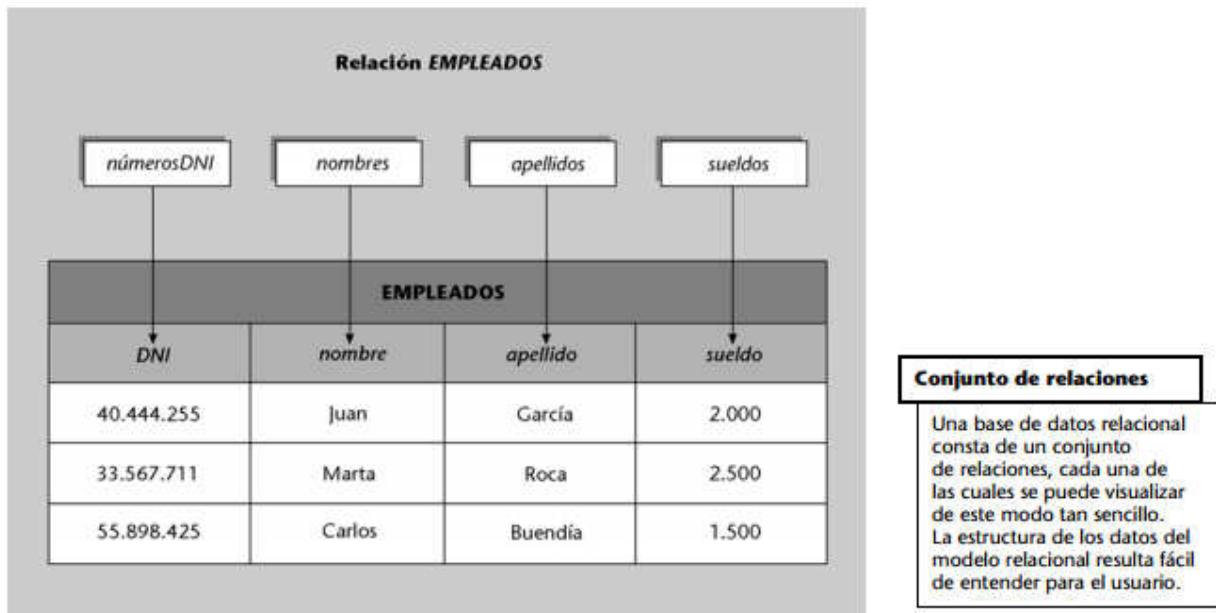
En primer lugar, presentaremos el concepto de relación de manera informal. Se puede obtener una buena idea intuitiva de lo que es una relación si la visualizamos como una tabla o un fichero. En la figura 1 se muestra la visualización tabular de una relación que contiene datos de empleados. Cada fila de la tabla contiene una colección de valores de datos relacionados entre sí; en nuestro ejemplo, son los datos correspondientes a un mismo empleado. La tabla tiene un nombre (EMPLEADOS) y también tiene un nombre cada una de sus columnas (DNI, nombre, apellido y sueldo). El nombre de la tabla y los de las columnas ayudan a entender el significado de los valores que contiene la tabla. Cada columna contiene valores de un cierto dominio; por ejemplo, la columna DNI contiene valores del dominio númerosDNI.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Figura1



Si definimos las relaciones de forma más precisa, nos daremos cuenta de que presentan algunas características importantes que, en la visión superficial que hemos presentado, quedan ocultas. Estas características son las que motivan que el concepto de relación sea totalmente diferente del de fichero, a pesar de que, a primera vista, relaciones y ficheros puedan parecer similares.

Visión formal de una relación

A continuación definimos formalmente las relaciones y otros conceptos que están vinculados a ellas, como por ejemplo dominio, esquema de relación, etc.

Un **dominio D** es un conjunto de valores atómicos. Por lo que respecta al modelo relacional, *atómico* significa indivisible; es decir, que por muy complejo o largo que sea un valor atómico, no tiene una estructuración interna para un SGBD relacional.

Los dominios pueden ser de dos tipos:

- 1) Dominios predefinidos, que corresponde a los tipos de datos que normalmente proporcionan los lenguajes de bases de datos, como por ejemplo los enteros, las cadenas de caracteres, los reales, etc.
- 2) Dominios definidos por el usuario, que pueden ser más específicos. Toda definición de un dominio debe constar, como mínimo, del nombre del dominio y de la descripción de los valores que forman parte de éste.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Un relación se compone del esquema (o intención de la relación) y de la extensión.

Dominio definido por el usuario

Por ejemplo, el usuario puede definir un dominio para las edades de los empleados que se denomine *dom_edad* y que contenga los valores enteros que están entre 16 y 65.

Si consideramos la representación tabular anterior (figura 1), el esquema correspondería a la cabecera de la tabla y la extensión correspondería al cuerpo:

Figura 2

Empleados			
DNI	nombre	apellido	sueldo
40.444.255	Juan	García	2.000
33.567.711	Marta	Roca	2.500
55.898.425	Carlos	Buendía	1.500

► Esquema

► Extensión

El esquema de la relación consiste en un nombre de relación *R* y un conjunto de atributos $\{A_1, A_2, \dots, A_n\}$.

Nombre y conjunto de atributos de la relación EMPLEADOS

Si tomamos como ejemplo la figura 1, el nombre de la relación es EMPLEADOS y el conjunto de atributos es {DNI, nombre, apellido, sueldo}.

Tomaremos la convención de denotar el esquema de la relación de la forma siguiente: *R* (*A*₁, *A*₂, ..., *A*_{*n*}), donde *R* es el nombre la relación y *A*₁, *A*₂, ..., *A*_{*n*} es una ordenación cualquiera de los atributos que pertenecen al conjunto $\{A_1, A_2, \dots, A_n\}$.

Denotación del esquema de la relación EMPLEADOS

El esquema de la relación de la figura 1 se podría denotar, por ejemplo, como EMPLEADOS (DNI, nombre, apellido, sueldo), o también, EMPLEADOS (nombre, apellido, DNI, sueldo), porque cualquier ordenación de sus atributos se considera válida para denotar el esquema de una relación.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Un atributo A_i es el nombre del papel que ejerce un dominio D en un esquema de relación. D es el **dominio de A_i** y se denota como dominio (A_i).

Dominio del atributo DNI

Según la figura 1, el atributo DNI corresponde al papel que ejerce el dominio númerosDNI en el esquema de la relación EMPLEADOS y, entonces, dominio(DNI) = númerosDNI.

Conviene observar que cada atributo es único en un esquema de relación, porque no tiene sentido que un mismo dominio ejerza dos veces el mismo papel en un mismo esquema. Por consiguiente, no puede ocurrir que en un esquema de relación haya dos atributos con el mismo nombre. En cambio, sí que se puede repetir un nombre de atributo en relaciones diferentes. Los dominios de los atributos, por el contrario, no deben ser necesariamente todos diferentes en una relación.

Ejemplo de atributos diferentes con el mismo dominio

Si tomamos como ejemplo el esquema de relación PERSONAS(DNI, nombre, apellido, telcasa, teltrabajo), los atributos telcasa y teltrabajo pueden tener el mismo dominio: dominio(telcasa)=teléfono y dominio(teltrabajo)=teléfono.

En este caso, el dominio teléfono ejerce dos papeles diferentes en el esquema de relación: el de indicar el teléfono particular de una persona y el de indicar el del trabajo.

La extensión de la relación de esquema $R(A_1, A_2, \dots, A_n)$ es un conjunto de tuplas t_i ($i = 1, 2, \dots, m$), donde cada tupla t_i es, a su vez un conjunto de pares $t_i = \{<A_1:v_{i1}>, <A_2:v_{i2}> \dots >A_n:v_{in}\}$ y, para cada par $<A_j:v_{ij}>$, se cumple que v_{ij} es un valor de dominio(A_j), o bien un valor especial que denominaremos *nulo*.

Algunos autores...

... denominan *tablas*, *columnas* y *filas* a las relaciones, los *atributos* y las *tuplas*, respectivamente.

Para simplificar, tomaremos la convención de referirnos a una tupla $t_i = \{<A_1:v_{i1}>, <A_2:v_{i2}>, \dots, <A_n:v_{in}\}$ que pertenece a la extensión del esquema denotado como $R(A_1, A_2, \dots, A_n)$, de la forma siguiente: $t_i = \{v_{i1}, v_{i2}, \dots, v_{in}\}$

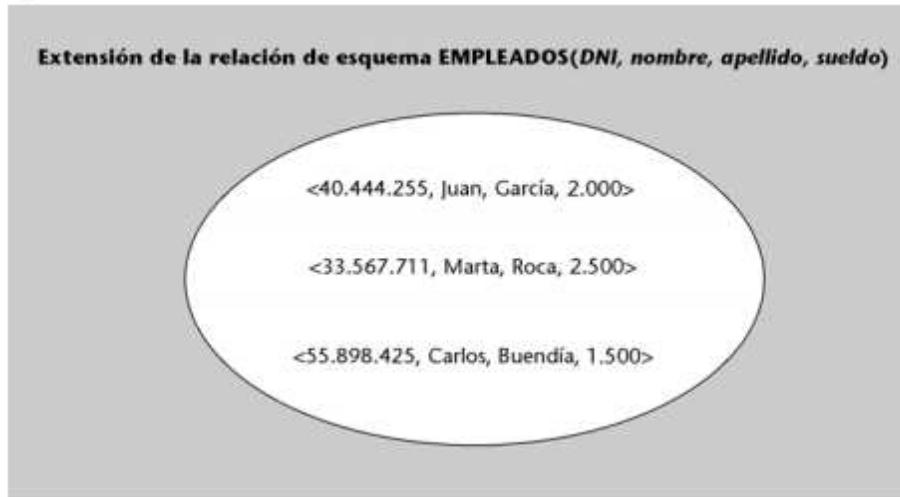
Si denotamos el esquema de la relación representada en la figura 1 como EMPLEADOS(DNI, nombre, apellido, sueldo), el conjunto de tuplas de su extensión será el de la figura siguiente:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Figura 3



Esta figura...

... nos muestra la extensión de *EMPLEADOS* en forma de conjunto, mientras que las figuras anteriores nos la mostraban en forma de filas de una tabla. La representación tabular es más cómoda, pero no refleja la definición de extensión con tanta exactitud.

Si en una tupla $t_i = \langle v_{i1}, v_{i2}, \dots, v_{in} \rangle$, el valor v_{ij} es un valor nulo, entonces el valor del atributo A_j es desconocido para la tupla t_i de la relación, o bien no es aplicable a esta tupla.

Ejemplo de valor nulo

Podríamos tener un atributo telcasa en la relación EMPLEADOS y se podría dar el caso de que un empleado no tuviese teléfono en su casa, o bien que lo tuviese, pero no se conociese su número. En las dos situaciones, el valor del atributo telcasa para la tupla correspondiente al empleado sería el valor nulo.

El grado de una relación es el número de atributos que pertenecen a su esquema.

Grado de la relación EMPLEADOS

El grado de la relación de esquema EMPLEADOS(DNI, nombre, apellido, sueldo), es 4.

La cardinalidad de una relación es el número de tuplas que pertenecen a su extensión.

Cardinalidad de la relación EMPLEADOS

Observando la figura 3 se deduce que la cardinalidad de la relación EMPLEADOS es 3.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Diferencias entre relaciones y ficheros

A primera vista, relaciones y ficheros resultan similares. Los registros y los campos que forman los ficheros se parecen a las tuplas y a los atributos de las relaciones, respectivamente.

A pesar de esta similitud superficial, la visión formal de relación que hemos presentado establece algunas características de las relaciones que las hacen diferentes de los ficheros clásicos. A continuación describimos estas características:

1) Atomicidad de los valores de los atributos: los valores de los atributos de una relación deben ser atómicos; es decir, no deben tener estructura interna.

Esta característica proviene del hecho de que los atributos siempre deben tomar un valor de su dominio o bien un valor nulo, y de que se ha establecido que los valores de los dominios deben ser atómicos en el modelo relacional.

El objetivo de la atomicidad de los valores es dar simplicidad y uniformidad al modelo relacional.

2) No-repetición de las tuplas: en un fichero clásico puede ocurrir que dos de los registros sean exactamente iguales; es decir, que contengan los mismos datos.

En el caso del modelo relacional, en cambio, no es posible que una relación contenga tuplas repetidas. Esta característica se deduce de la misma definición de la extensión de una relación. La extensión es un conjunto de tuplas y, en un conjunto, no puede haber elementos repetidos.

3) No-ordenación de las tuplas: de la definición de la extensión de una relación como un conjunto de tuplas se deduce también que estas tuplas no estarán ordenadas, teniendo en cuenta que no es posible que haya una ordenación entre los elementos de un conjunto.

La finalidad de esta característica es conseguir que, mediante el modelo relacional, se puedan representar los hechos en un nivel abstracto que sea independiente de su estructura física de implementación. Más concretamente, aunque los SGBD relacionales deban proporcionar una implementación física que almacenará las tuplas de las relaciones en un orden concreto, esta ordenación no es visible si nos situamos en el nivel conceptual.

Ejemplo de no-ordenación de las tuplas

En una base de datos relacional, por ejemplo, no tiene sentido consultar la “primera tupla” de la relación EMPLEADOS.

4) No-ordenación de los atributos: el esquema de una relación consta de un nombre de relación R y un conjunto de atributos {A1, A2, ..., An}. Así pues, no hay un orden entre los atributos de un esquema de relación, teniendo en cuenta que estos atributos forman un conjunto.

Como en el caso anterior, el objetivo de esta característica es representar los hechos en un nivel abstracto, independientemente de su implementación física.

Ejemplo de no-ordenación de los atributos

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

El esquema de relación EMPLEADOS(DNI, nombre, apellido, sueldo) denota el mismo esquema de relación que EMPLEADOS(nombre, apellido, DNI, sueldo).

Clave candidata, clave primaria y clave alternativa de las relaciones

Toda la información que contiene una base de datos debe poderse identificar de alguna forma. En el caso particular de las bases de datos que siguen el modelo relacional, para identificar los datos que la base de datos contiene, se pueden utilizar las claves candidatas de las relaciones. A continuación definimos qué se entiende por clave candidata, clave primaria y clave alternativa de una relación. Para hacerlo, será necesario definir el concepto de superclave.

Una superclave de una relación de esquema $R(A_1, A_2, \dots, A_n)$ es un subconjunto de los atributos del esquema tal que no puede haber dos tuplas en la extensión de la relación que tengan la misma combinación de valores para los atributos del subconjunto.

Una superclave, por lo tanto, nos permite identificar todas las tuplas que contiene la relación.

Algunas superclaves de la relación EMPLEADOS

En la relación de esquema EMPLEADOS(DNI, NSS, nombre, apellido, teléfono), algunas de las superclaves de la relación serían los siguientes subconjuntos de atributos: {DNI, NSS, nombre, apellido, teléfono}, {DNI, apellido}, {DNI} y {NSS}.

Una clave candidata de una relación es una superclave C de la relación que cumple que ningún subconjunto propio de C es superclave.

Es decir, C cumple que la eliminación de cualquiera de sus atributos da un conjunto de atributos que no es superclave de la relación. Intuitivamente, una clave candidata permite identificar cualquier tupla de una relación, de manera que no sobre ningún atributo para hacer la identificación.

Claves candidatas de EMPLEADOS

En la relación de esquema EMPLEADOS(DNI, NSS, nombre, apellido, teléfono), sólo hay dos claves candidatas: {DNI} y {NSS}.

Habitualmente, una de las claves candidatas de una relación se designa clave primaria de la relación. La clave primaria es la clave candidata cuyos valores se utilizarán para identificar las tuplas de la relación.

Por ejemplo, ...

... si se almacena información sobre los empleados de una empresa, es preciso tener la posibilidad de distinguir qué datos corresponden a cada uno de los diferentes empleados.

Observad que...

... toda relación tiene, por lo menos, una superclave, que es la formada por todos los atributos de su esquema. Esto se debe a la propiedad que cumple toda relación de no tener tuplas repetidas. En el ejemplo de EMPLEADOS(DNI, NSS, nombre, apellido, teléfono) esta superclave sería: {DNI, NSS, nombre, apellido, teléfono}.

Notad que, ...

... puesto que toda relación tiene por lo menos una superclave, podemos garantizar que toda relación tiene como mínimo una clave candidata.

Relación con una clave candidata

Si una relación sólo tiene una clave candidata, entonces esta clave candidata debe ser también su clave primaria. Ya que todas las relaciones tienen como mínimo una clave candidata, podemos garantizar que, para toda relación, será posible designar una clave primaria.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

El diseñador de la base de datos es quien elige la clave primaria de entre las claves candidatas.

Las claves candidatas no elegidas como primaria se denominan **claves alternativas**.

Utilizaremos la convención de subrayar los atributos que forman parte de la clave primaria en el esquema de la relación. Así pues, R(A₁, A₂, ..., A_i, ..., A_n) indica que los atributos A₁, A₂, ..., A_i forman la clave primaria de R.

Elección de la clave primaria de EMPLEADOS

En la relación de esquema EMPLEADOS(DNI, NSS, nombre, apellido, teléfono), donde hay dos claves candidatas, {DNI} y {NSS}, se puede elegir como clave primaria {DNI}. Lo indicaremos subrayando el atributo DNI en el esquema de la relación EMPLEADOS(DNI, NSS, nombre, apellido, teléfono). En este caso, la clave {NSS} será una clave alternativa de EMPLEADOS.

Es posible que una clave candidata o una clave primaria conste de más de un atributo.

Clave primaria de la relación DESPACHOS

En la relación de esquema DESPACHOS(edificio, número, superficie), la clave primaria está formada por los atributos edificio y número. En este caso, podrá ocurrir que dos despachos diferentes estén en el mismo edificio, o bien que tengan el mismo número, pero nunca pasará que tengan la misma combinación de valores para edificio y número.

Claves foráneas de las relaciones

Hasta ahora hemos estudiado las relaciones de forma individual, pero debemos tener en cuenta que una base de datos relacional normalmente contiene más de una relación, para poder representar distintos tipos de hechos que suceden en el mundo real. Por ejemplo, podríamos tener una pequeña base de datos que contuviese dos relaciones: una denominada EMPLEADOS, que almacenaría datos de los empleados de una empresa, y otra con el nombre DESPACHOS, que almacenaría los datos de los despachos que tiene la empresa.

Debemos considerar también que entre los distintos hechos que se dan en el mundo real pueden existir lazos o vínculos. Por ejemplo, los empleados que trabajan para una empresa pueden estar vinculados con los despachos de la empresa, porque a cada empleado se le asigna un despacho concreto para trabajar.

En el modelo relacional, para reflejar este tipo de vínculos, tenemos la posibilidad de expresar conexiones entre las distintas tuplas de las relaciones. Por ejemplo, en la base de datos anterior, que tiene las relaciones EMPLEADOS y DESPACHOS, puede ser necesario conectar tuplas de EMPLEADOS con tuplas de DESPACHOS para indicar qué despacho tiene asignado cada empleado.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

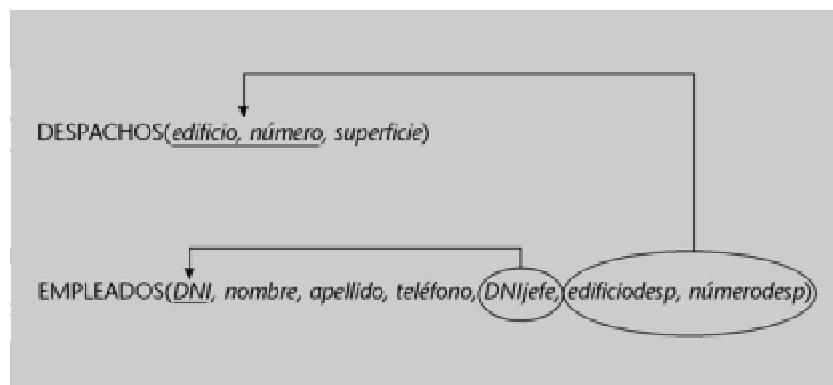
En ocasiones, incluso puede ser necesario reflejar lazos entre tuplas que pertenecen a una misma relación. Por ejemplo, en la misma base de datos anterior puede ser necesario conectar determinadas tuplas de **EMPLEADOS** con otras tuplas de **EMPLEADOS** para indicar, para cada empleado, quién actúa como su jefe.

El mecanismo que proporcionan las bases de datos relacionales para conectar tuplas son las claves foráneas de las relaciones. Las claves foráneas permiten establecer conexiones entre las tuplas de las relaciones. Para hacer la conexión, una clave foránea tiene el conjunto de atributos de una relación que referencian la clave primaria de otra relación (o incluso de la misma relación).

Claves foráneas de la relación **EMPLEADOS**

En la figura siguiente, la relación **EMPLEADOS(DNI, nombre, apellido, teléfono, DNIjefe, edificiodesp, númerodesp)**, tiene una clave foránea formada por los atributos **edificiodesp** y **númerodesp** que se refiere a la clave primaria de la relación **DESPACHOS(edificio, número, superficie)**.

Esta clave foránea indica, para cada empleado, el despacho donde trabaja. Además, el atributo **DNIjefe** es otra clave foránea que referencia la clave primaria de la misma relación **EMPLEADOS**, e indica, para cada empleado, quien es su jefe.



Las claves foráneas tienen por objetivo establecer una conexión con la clave primaria que referencian. Por lo tanto, los valores de una clave foránea deben estar presentes en la clave primaria correspondiente, o bien deben ser valores nulos. En caso contrario, la clave foránea representaría una referencia o conexión incorrecta.

Ejemplo

En la relación de esquema **EMPLEADOS(DNI, nombre, apellido, teléfono, DNIjefe, edificiodesp, númerodesp)**, la clave foránea **{edificiodesp, númerodesp}** referencia la relación **DESPACHOS(edificio, número, superficie)**. De este modo, se cumple que todos los valores que

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

no son nulos de los atributos *edificiodesp* y *númerodesp* son valores que existen para los atributos *edificio* y *número* de DESPACHOS, tal y como se puede ver a continuación:

- Relación DESPACHOS:

DESPACHOS		
<i>edificio</i>	<i>número</i>	<i>superficie</i>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

- Relación EMPLEADOS

EMPLEADOS					
<i>DNI</i>	<i>nombre</i>	<i>apellido</i>	<i>DNIjefe</i>	<i>edificiodesp</i>	<i>númerodesp</i>
40.444.255	Juan	García	NULO	Marina	120
33.567.711	Marta	Roca	40.444.255	Marina	120
55.898.425	Carlos	Buendía	40.444.255	Diagonal	120
77.232.144	Elena	Pla	40.444.255	NULO	NULO

Supongamos que hubiese un empleado con los valores <55.555.555, María, Casagran, NULO, París, 400>. Puesto que no hay ningún despacho con los valores París y 400 para edificio y número, la tupla de este empleado hace una referencia incorrecta; es decir, indica un despacho para el empleado que, de hecho, no existe.

Es preciso señalar que en la relación EMPLEADOS hay otra clave foránea, {DNIjefe}, que referencia la misma relación EMPLEADOS, y entonces se cumple que todos los valores que no son nulos del atributo DNIjefe son valores que existen para el atributo DNI de la misma relación EMPLEADOS.

A continuación estableceremos de forma más precisa qué se entiende por clave foránea.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Una clave foránea de una relación R es un subconjunto de atributos del esquema de la relación, que denominamos CF y que cumple las siguientes condiciones:

- 1) Existe una relación S (S no debe ser necesariamente diferente de R) que tiene por clave primaria CP .
- 2) Se cumple que, para toda tupla t de la extensión de R , los valores para CF de t son valores nulos o bien valores que coinciden con los valores para CP de alguna tupla s de S .

Y entonces, se dice que la clave foránea CF referencia la clave primaria CP de la relación S , y también que la clave foránea CF referencia la relación S .

Conviene subrayar que, ...

... tal y como ya hemos mencionado, el modelo relacional permite representar toda la información mediante valores explícitos que contienen las relaciones, y no le hace falta nada más. De este modo, las conexiones entre tuplas de las relaciones se expresan con los valores explícitos de las claves foráneas de las relaciones, y no son necesarios conceptos adicionales (por ejemplo, apuntadores entre tuplas), para establecer estas conexiones. Esta característica da simplicidad y uniformidad al modelo.

De la noción que hemos dado de clave foránea se pueden extraer varias consecuencias:

- 1) Si una clave foránea CF referencia una clave primaria CP , el número de atributos de CF y de CP debe coincidir.

Ejemplo de coincidencia del número de atributos de CF y CP

En el ejemplo anterior, tanto la clave foránea {edificiodesp, númerodesp} como la clave primaria que referencia {edificio, número} tienen dos atributos. Si no sucediese así, no sería posible que los valores de CF existieran en CP .

- 2) Por el mismo motivo, se puede establecer una correspondencia (en concreto, una biyección) entre los atributos de la clave foránea y los atributos de la clave primaria que referencia.

Ejemplo de correspondencia entre los atributos de CF y los de CP

En el ejemplo anterior, a edificiodesp le corresponde el atributo edificio, y a númerodesp le corresponde el atributo número.

- 3) También se deduce de la noción de clave foránea que los dominios de sus atributos deben coincidir con los dominios de los atributos correspondientes a la clave primaria que referencia. Esta coincidencia de dominios hace que sea posible que los valores de la clave foránea coincidan con valores de la clave primaria referenciada.

Ejemplo de coincidencia de los dominios

En el ejemplo anterior, se debe cumplir que dominio (edificiodesp) = dominio(edificio) y también que dominio(númerodesp) = dominio(número).

Nivel: Quinto año

Observad que, de hecho, esta condición se podría relajar, y se podría permitir que los dominios no fuesen exactamente iguales, sino que sólo fuesen, y de alguna forma que convendría precisar, dominios “compatibles”. Para simplificarlo, nosotros supondremos que los dominios deben ser iguales en todos los casos en que, según Date (2001), se aceptarán dominios “compatibles”.

Lectura recomendada

Encontraréis explicaciones detalladas sobre la coincidencia de dominios en la obra siguiente:

C.J. Date (2001).
Introducción a los sistemas de bases de datos (7^a ed., cap. 19). Prentice Hall.

Ejemplo de atributo que forma parte de la clave primaria y de una clave foránea

Puede suceder que algún atributo de una relación forme parte tanto de la clave primaria como de una clave foránea de la relación. Esto se da en las relaciones siguientes: EDIFICIOS (nombredificio, dirección), y DESPACHOS(edificio, número, superficie), donde {edificio} es una clave foránea que referencia EDIFICIOS.

En este ejemplo, el atributo edificio forma parte tanto de la clave primaria como de la clave foránea de la relación DESPACHOS.

Creación de las relaciones de una base de datos

Hemos visto que una base de datos relacional consta de varias relaciones. Cada relación tiene varios atributos que toman valores de unos ciertos dominios; también tiene una clave primaria y puede tener una o más claves foráneas. Los lenguajes de los SGBD relacionales deben proporcionar la forma de definir todos estos elementos para crear una base de datos.

Más adelante se verá con detalle la sintaxis y el significado de las sentencias de definición de la base de datos para el caso concreto del lenguaje SQL.

Operaciones del modelo relacional

Las operaciones del modelo relacional deben permitir manipular datos almacenados en una base de datos relacional y, por lo tanto, estructurados en forma de relaciones. La manipulación de datos incluye básicamente dos aspectos: la actualización y la consulta.

La actualización de los datos consiste en hacer que los cambios que se producen en la realidad queden reflejados en las relaciones de la base de datos.

Ejemplo de actualización

Si una base de datos contiene, por ejemplo, información de los empleados de una empresa, y la empresa contrata a un empleado, será necesario reflejar este cambio añadiendo los datos del nuevo empleado a la base de datos.

Nivel: Quinto año

Existen tres operaciones básicas de actualización:

- a) Inserción, que sirve para añadir una o más tuplas a una relación.
- b) Borrado, que sirve para eliminar una o más tuplas de una relación.
- c) Modificación, que sirve para alterar los valores que tienen una o más tuplas de una relación para uno o más de sus atributos.

La consulta de los datos consiste en la obtención de datos deducibles a partir de las relaciones que contiene la base de datos.

Ejemplo de consulta

Si una base de datos contiene, por ejemplo, información de los empleados de una empresa, puede interesar consultar el nombre y apellido de todos los empleados que trabajan en un despacho situado en un edificio que tiene por nombre Marina.

La obtención de los datos que responden a una consulta puede requerir el análisis y la extracción de datos de una o más de las relaciones que mantiene la base de datos.

Según la forma como se especifican las consultas, podemos clasificar los lenguajes relationales en dos tipos:

1) Lenguajes basados en el álgebra relacional. El álgebra relacional se inspira en la teoría de conjuntos. Si queremos especificar una consulta, es necesario seguir uno o más pasos que sirven para ir construyendo, mediante operaciones del álgebra relacional, una nueva relación que contenga los datos que responden a la consulta a partir de las relaciones almacenadas. Los lenguajes basados en el álgebra relacional son lenguajes procedimentales, ya que los pasos que forman la consulta describen un procedimiento.

2) Lenguajes basados en el cálculo relacional. El cálculo relacional tiene su fundamento teórico en el cálculo de predicados de la lógica matemática. Proporciona una notación que permite formular la definición de la relación donde están los datos que responden la consulta en términos de las relaciones almacenadas. Esta definición no describe un procedimiento; por lo tanto, se dice que los lenguajes basados en el cálculo relacional son lenguajes declarativos (no procedimentales).

El lenguaje SQL, en las sentencias de consulta, combina construcciones del álgebra relacional y del cálculo relacional con un predominio de las construcciones del cálculo. Este predominio determina que SQL sea un lenguaje declarativo.

El estudio del álgebra relacional presenta un interés especial, pues ayuda a entender qué servicios de consulta debe proporcionar un lenguaje relacional, facilita la comprensión de algunas de las construcciones del lenguaje SQL y también sirve de base para el tratamiento de las consultas que efectúan los SGBD internamente.

Nivel: Quinto año

Este último tema queda fuera del ámbito del presente curso, pero es necesario para estudios más avanzados sobre bases de datos.

Reglas de integridad

Una base de datos contiene unos datos que, en cada momento, deben reflejar la realidad o, más concretamente, la situación de una porción del mundo real.

En el caso de las bases de datos relacionales, esto significa que la extensión de las relaciones (es decir, las tuplas que contienen las relaciones) deben tener valores que reflejen la realidad correctamente.

Suele ser bastante frecuente que determinadas configuraciones de valores para las tuplas de las relaciones no tengan sentido, porque no representan ninguna situación posible del mundo real.

Un sueldo negativo

En la relación de esquema EMPLEADOS(DNI, nombre, apellido, sueldo), una tupla que tiene un valor de -1.000 para el sueldo probablemente no tiene sentido, porque los sueldos no pueden ser negativos.

Denominamos integridad la propiedad de los datos de corresponder a representaciones plausibles del mundo real.

Como es evidente, para que los datos sean íntegros, es preciso que cumplan varias condiciones.

El hecho de que los sueldos no puedan ser negativos es una condición que se debería cumplir en la relación EMPLEADOS.

En general, las condiciones que garantizan la integridad de los datos pueden ser de dos tipos:

1) Las restricciones de integridad de usuario son condiciones específicas de una base de datos concreta; es decir, son las que se deben cumplir en una base de datos particular con unos usuarios concretos, pero que no son necesariamente relevantes en otra base de datos.

Restricción de integridad de usuario en EMPLEADOS

Éste sería el caso de la condición anterior, según la cual los sueldos no podían ser negativos.

Observad que esta condición era necesaria en la base de datos concreta de este ejemplo porque aparecía el atributo sueldo, al que se quería dar un significado; sin embargo, podría no ser necesaria en otra base de datos diferente donde, por ejemplo, no hubiese sueldos.

2) Las reglas de integridad de modelo, en cambio, son condiciones más generales, propias de un modelo de datos, y se deben cumplir en toda base de datos que siga dicho modelo.

Ejemplo de regla de integridad del modelo de datos relacional

Nivel: Quinto año

En el caso del modelo de datos relacional, habrá una regla de integridad para garantizar que los valores de una clave primaria de una relación no se repitan en tuplas diferentes de la relación. Toda base de datos relacional debe cumplir esta regla que, por lo tanto, es una regla de integridad del modelo.

Los SGBD deben proporcionar la forma de definir las restricciones de integridad de usuario de una base de datos; una vez definidas, deben velar por su cumplimiento.

Las reglas de integridad del modelo, en cambio, no se deben definir para cada base de datos concreta, porque se consideran preestablecidas para todas las bases de datos de un modelo. Un SGBD de un modelo determinado debe velar por el cumplimiento de las reglas de integridad preestablecidas por su modelo.

A continuación estudiaremos con detalle las reglas de integridad del modelo relacional, reglas que todo SGBD relacional debe obligar a cumplir.

4.1. Regla de integridad de unicidad de la clave primaria

La regla de integridad de unicidad está relacionada con la definición de clave primaria. Concretamente, establece que toda clave primaria que se elija para una relación no debe tener valores repetidos.

Ejemplo

Tenemos la siguiente relación:

DESPACHOS		
<u>edificio</u>	<u>número</u>	<u>superficie</u>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

En esta relación, dado que la clave primaria está formada por edificio y número, no hay ningún despacho que repita tanto edificio como número de otro despacho. Sin embargo, sí se repiten valores de edificio (por ejemplo, Marina); y también se repiten valores de número (120). A pesar de ello, el edificio y el número no se repiten nunca al mismo tiempo.

A continuación explicamos esta regla de forma más precisa.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

La regla de integridad de unicidad de la clave primaria establece que si el conjunto de atributos CP es la clave primaria de una relación R , entonces la extensión de R no puede tener en ningún momento dos tuplas con la misma combinación de valores para los atributos de CP .

Un SGBD relacional deberá garantizar el cumplimiento de esta regla de integridad en todas las inserciones, así como en todas las modificaciones que afecten a atributos que pertenecen a la clave primaria de la relación.

Ejemplo

Tenemos la siguiente relación:

DESPACHOS		
<i>edificio</i>	<i>número</i>	<i>superficie</i>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

En esta relación no se debería poder insertar la tupla <Diagonal, 120, 30>, ni modificar la tupla <Marina, 122, 15>, de modo que pasara a ser <Marina, 120, 15>.

Regla de integridad de entidad de la clave primaria

La regla de integridad de entidad de la clave primaria dispone que los atributos de la clave primaria de una relación no pueden tener valores nulos.

Ejemplo

Tenemos la siguiente relación:

DESPACHOS		
<i>edificio</i>	<i>número</i>	<i>superficie</i>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

En esta relación, puesto que la clave primaria está formada por *edificio* y *número*, no hay ningún despacho que tenga un valor nulo para *edificio*, ni tampoco para *número*.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Esta regla es necesaria para que los valores de las claves primarias puedan identificar las tuplas individuales de las relaciones. Si las claves primarias tuviesen valores nulos, es posible que algunas tuplas no se pudieran distinguir.

Ejemplo de clave primaria incorrecta con valores nulos

En el ejemplo anterior, si un despacho tuviese un valor nulo para edificio porque en un momento dado el nombre de este edificio no se conoce, por ejemplo <NULLO, 120, 30>, la clave primaria no nos permitiría distinguirlo del despacho <Marina, 120, 10> ni del despacho <Diagonal, 120,10>. No podríamos estar seguros de que el valor desconocido de edificio no es ni Marina ni Diagonal.

A continuación definimos esta regla de forma más precisa.

La regla de integridad de entidad de la clave primaria establece que si el conjunto de atributos CP es la clave primaria de una relación R , la extensión de R no puede tener ninguna tupla con algún valor nulo para alguno de los atributos de CP .

Un SGBD relacional tendrá que garantizar el cumplimiento de esta regla de integridad en todas las inserciones y, también, en todas las modificaciones que afecten a atributos que pertenecen a la clave primaria de la relación.

Ejemplo

En la relación DESPACHOS anterior, no se debería insertar la tupla <Diagonal, NULO, 15>.

Tampoco debería ser posible modificar la tupla <Marina, 120, 10> de modo que pasara a ser <NULLO, 120, 10>.

Regla de integridad referencial

La regla de integridad referencial está relacionada con el concepto de clave foránea.

Concretamente, determina que todos los valores que toma una clave foránea deben ser valores nulos o valores que existen en la clave primaria que referencia.

Ejemplo

Si tenemos las siguientes relaciones:

- Relación DESPACHOS:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

DESPACHOS		
<u>edificio</u>	<u>número</u>	<u>superficie</u>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

- Relación EMPLEADOS:

EMPLEADOS				
<u>DNI</u>	<u>nombre</u>	<u>apellido</u>	<u>edificiodesp</u>	<u>númerodesp</u>
40.444.255	Juan	García	Marina	120
33.567.711	Marta	Roca	Marina	120
55.898.425	Carlos	Buendía	Diagonal	120
77.232.144	Elena	Pla	NULO	NULO

donde edificiodesp y númerodesp de la relación EMPLEADOS forman una clave foránea que referencia la relación DESPACHOS. Debe ocurrir que los valores no nulos de edificiodesp y númerodesp de la relación EMPLEADOS estén en la relación DESPACHOS como valores de edificio y número. Por ejemplo, el empleado <40.444.255, Juan García, Marina, 120> tiene el valor

Marina para edificiodesp, y el valor 120 para númerodesp, de modo que en la relación DESPACHOS hay un despacho con valor Marina para edificio y con valor 120 para número.

La necesidad de la regla de integridad relacional proviene del hecho de que las claves foráneas tienen por objetivo establecer una conexión con la clave primaria que referencian. Si un valor de una clave foránea no estuviese presente en la clave primaria correspondiente, representaría una referencia o una conexión incorrecta.

Referencia incorrecta

Supongamos que en el ejemplo anterior hubiese un empleado con los valores <56.666.789, Pedro, López, Valencia, 325>. Ya que no hay un despacho con los valores Valencia y 325 para edificio y número, la tupla de este empleado hace una referencia incorrecta; es decir, indica un despacho para el empleado que, de hecho, no existe.

A continuación explicamos la regla de modo más preciso.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

La regla de **integridad referencial** establece que si el conjunto de atributos *CF* es una clave foránea de una relación *R* que referencia una relación *S* (no necesariamente diferente de *R*), que tiene por clave primaria *CP*, entonces, para toda tupla *t* de la extensión de *R*, los valores para el conjunto de atributos *CF* de *t* son valores nulos, o bien valores que coinciden con los valores para *CP* de alguna tupla *s* de *S*.

En el caso de que una tupla *t* de la extensión de *R* tenga valores para *CF* que coincidan con los valores para *CP* de una tupla *s* de *S*, decimos que *t* es una tupla que referencia *s* y que *s* es una tupla que tiene una clave primaria referenciada por *t*.

Un SGBD relacional tendrá que hacer cumplir esta regla de integridad. Deberá efectuar comprobaciones cuando se produzcan las siguientes operaciones:

- Inserciones en una relación que tenga una clave foránea.
- Modificaciones que afecten a atributos que pertenecen a la clave foránea de una relación.
- Borrados en relaciones referenciadas por otras relaciones.
- Modificaciones que afecten a atributos que pertenecen a la clave primaria de una relación referenciada por otra relación.

Ejemplo

Retomamos el ejemplo anterior, donde *edificiodesp* y *númerodesp* de la relación *EMPLEADOS* forman una clave foránea que referencia la relación *DESPACHOS*:

- Relación *DESPACHOS*:

DESPACHOS		
<i>edificio</i>	<i>número</i>	<i>superficie</i>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

- Relación *EMPLEADOS*:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

EMPLEADOS				
<u>DNI</u>	<u>nombre</u>	<u>apellido</u>	<u>edificiodesp</u>	<u>númerodesp</u>
40.444.255	Juan	García	Marina	120
33.567.711	Marta	Roca	Marina	120
55.898.425	Carlos	Buendía	Diagonal	120
77.232.144	Elena	Pla	NULO	NULO

Las siguientes operaciones provocarían el incumplimiento de la regla de integridad referencial:

- Inserción de <12.764.411, Jorge, Puig, Diagonal, 220> en EMPLEADOS.
- Modificación de <40.444.255, Juan, García, Marina, 120> de EMPLEADOS por <40.444.255, Juan, García, Marina, 400>.
- Borrado de <Marina, 120, 10> de DESPACHOS.
- Modificación de <Diagonal, 120, 10> de DESPACHOS por <París, 120, 10>.

Un SGBD relacional debe procurar que se cumplan las reglas de integridad del modelo. Una forma habitual de mantener estas reglas consiste en rechazar toda operación de actualización que deje la base de datos en un estado en el que alguna regla no se cumpla. En algunos casos, sin embargo, el SGBD tiene la posibilidad de aceptar la operación y efectuar acciones adicionales compensatorias, de modo que el estado que se obtenga satisfaga las reglas de integridad, a pesar de haber ejecutado la operación.

Esta última política se puede aplicar en las siguientes operaciones de actualización que violarían la regla de integridad:

- a) Borrado de una tupla que tiene una clave primaria referenciada.
- b) Modificación de los valores de los atributos de la clave primaria de una tupla que tiene una clave primaria referenciada.

En los casos anteriores, algunas de las políticas que se podrán aplicar serán las siguientes: restricción, actualización en cascada y anulación. A continuación explicamos el significado de las tres posibilidades mencionadas.

4.3.1. Restricción

La política de restricción consiste en no aceptar la operación de actualización.

Más concretamente, la **restricción en caso de borrado**, consiste en no permitir borrar una tupla si tiene una clave primaria referenciada por alguna clave foránea.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

De forma similar, la **restricción en caso de modificación** consiste en no permitir modificar ningún atributo de la clave primaria de una tupla si tiene una clave primaria referenciada por alguna clave foránea.

Ejemplo de aplicación de la restricción

Supongamos que tenemos las siguientes relaciones:

- Relación CLIENTES:

CLIENTES	
<u>numcliente</u>	...
10	-
15	-
18	-

- Relación PEDIDOS_PENDIENTES

PEDIDOS_PENDIENTES		
<u>numped</u>	...	<u>numcliente*</u>
1.234	-	10
1.235	-	10
1.236	-	15

* numcliente referencia CLIENTES.

- Si aplicamos la restricción en caso de borrado y, por ejemplo, queremos borrar al cliente número 10, no podremos hacerlo porque tiene pedidos pendientes que lo referencian.
- Si aplicamos la restricción en caso de modificación y queremos modificar el número del cliente 15, no será posible hacerlo porque también tiene pedidos pendientes que lo referencian.

Actualización en cascada

La política de actualización en cascada consiste en permitir la operación de actualización de la tupla, y en efectuar operaciones compensatorias que propaguen en cascada la actualización a las tuplas que la referenciaban; se actúa de este modo para mantener la integridad referencial.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Más concretamente, la **actualización en cascada en caso de borrado** consiste en permitir el borrado de una tupla t que tiene una clave primaria referenciada, y borrar también todas las tuplas que referencian t .

De forma similar, la **actualización en cascada en caso de modificación** consiste en permitir la modificación de atributos de la clave primaria de una tupla t que tiene una clave primaria referenciada, y modificar del mismo modo todas las tuplas que referencian t .

Ejemplo de aplicación de la actualización en cascada

Supongamos que tenemos las siguientes relaciones:

- Relación EDIFICIOS:

EDIFICIOS	
<u>nombreedificio</u>	...
Marina	-
Diagonal	-

- Relación DESPACHOS:

DESPACHOS		
<u>edificio*</u>	<u>número</u>	<u>superficie</u>
Marina	120	10
Marina	122	15
Marina	230	20
Diagonal	120	10

* {edificio} referencia EDIFICIOS.

- b) Si aplicamos la actualización en cascada en caso de modificación, y queremos modificar el nombre del edificio Marina por Mar, también se cambiará Marina por Mar en los despachos Marina 120, Marina 122 y Marina 230, y nos quedará:

- Relación EDIFICIOS:

EDIFICIOS	
<u>nombreedificio</u>	...
Mar	-

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

- Relación DESPACHOS:

DESPACHOS		
<u>edificio*</u>	<u>número</u>	<u>superficie</u>
Mar	120	10
Mar	122	15
Mar	230	20

* edificio referencia EDIFICIOS.

Anulación

Esta política consiste en permitir la operación de actualización de la tupla y en efectuar operaciones compensatorias que pongan valores nulos a los atributos de la clave foránea de las tuplas que la referencian; esta acción se lleva a cabo para mantener la integridad referencial.

Puesto que generalmente los SGBD relacionales permiten establecer que un determinado atributo de una relación no admite valores nulos, sólo se puede aplicar la política de anulación si los atributos de la clave foránea sí los admiten.

Más concretamente, la **anulación en caso de borrado** consiste en permitir el borrado de una tupla t que tiene una clave referenciada y, además, modificar todas las tuplas que referencian t , de modo que los atributos de la clave foránea correspondiente tomen valores nulos.

De forma similar, la **anulación en caso de modificación** consiste en permitir la modificación de atributos de la clave primaria de una tupla t que tiene una clave referenciada y, además, modificar todas las tuplas que referencian t , de modo que los atributos de la clave foránea correspondiente tomen valores nulos.

Ejemplo de aplicación de la anulación

El mejor modo de entender en qué consiste la anulación es mediante un ejemplo. Tenemos las siguientes relaciones:

- Relación VENDEDORES:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

VENDEDORES	
<u>numvendedor</u>	***
1	-
2	-
3	-

- Relación CLIENTES
 - Relación **CLIENTES**:

CLIENTES		
<u>numcliente</u>	***	<i>vendedorasig*</i>
23	-	1
35	-	1
38	-	2
42	-	2
50	-	3

* *{vendedorasig}* referencia VENDEDORES.

- a) Si aplicamos la anulación en caso de borrado y, por ejemplo, queremos borrar al vendedor número 1, se modificarán todos los clientes que lo tenían asignado, y pasarán a tener un valor nulo en *vendedorasig*. Nos quedará:

- Relación VENDEDORES:

VENDEDORES	
<u>numvendedor</u>	***
2	-
3	-

- Relación CLIENTES:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

CLIENTES		
<u>numcliente</u>	...	<u>vendedorasig*</u>
23	-	NULO
35	-	NULO
38	-	2
42	-	2
50	-	3

* vendedorasig referencia VENDEDORES.

b) Si aplicamos la anulación en caso de modificación, y ahora queremos cambiar el número del vendedor 2 por 5, se modificarán todos los clientes que lo tenían asignado y pasarán a tener un valor nulo en vendedorasig. Nos quedará:

- Relación VENDEDORES:

VENDEDORES	
<u>numvendedor</u>	...
5	-
3	-

- Relación CLIENTES:

CLIENTES		
<u>numcliente</u>	...	<u>vendedorasig*</u>
23	-	NULO
35	-	NULO
38	-	NULO
42	-	NULO
50	-	3

* vendedorasig referencia VENDEDORES.

Nivel: Quinto año

Selección de la política de mantenimiento de la integridad referencial

Hemos visto que en caso de borrado o modificación de una clave primaria referenciada por alguna clave foránea hay varias políticas de mantenimiento de la regla de integridad referencial.

El diseñador puede elegir para cada clave foránea qué política se aplicará en caso de borrado de la clave primaria referenciada, y cuál en caso de modificación de ésta. El diseñador deberá tener en cuenta el significado de cada clave foránea concreta para poder elegir adecuadamente.

Aplicación de políticas diferentes

Puede ocurrir que, para una determinada clave foránea, la política adecuada en caso de borrado sea diferente de la adecuada en caso de modificación. Por ejemplo, puede ser necesario aplicar la restricción en caso de borrado y la actualización en cascada en caso de modificación.

Regla de integridad de dominio

La regla de integridad de dominio está relacionada, como su nombre indica, con la noción de dominio. Esta regla establece dos condiciones.

La primera condición consiste en que un valor no nulo de un atributo A_i debe pertenecer al dominio del atributo A_i ; es decir, debe pertenecer a dominio(A_i).

Esta condición implica que todos los valores no nulos que contiene la base de datos para un determinado atributo deben ser del dominio declarado para dicho atributo.

Ejemplo

Si en la relación EMPLEADOS(DNI, nombre, apellido, edademp) hemos declarado que dominio(DNI) es el dominio predefinido de los enteros, entonces no podremos insertar, por ejemplo, ningún empleado que tenga por DNI el valor “Luis”, que no es un entero.

Recordemos que los dominios pueden ser de dos tipos: predefinidos o definidos por el usuario. Observad que los dominios definidos por el usuario resultan muy útiles, porque nos permiten determinar de forma más específica cuáles serán los valores admitidos por los atributos.

Ejemplo

Supongamos ahora que en la relación EMPLEADOS(DNI, nombre, apellido, edademp) hemos declarado que dominio(edademp) es el dominio definido por el usuario edad. Supongamos también que el dominio edad se ha definido como el conjunto de los enteros que están entre 16 y 65. En este caso, por ejemplo, no será posible insertar un empleado con un valor de 90 para edademp.

Nivel: Quinto año

La segunda condición de la regla de integridad de dominio es más compleja, especialmente en el caso de dominios definidos por el usuario; los SGBD actuales no la soportan para estos últimos dominios. Por estos motivos sólo la presentaremos superficialmente.

Esta segunda condición sirve para establecer que los operadores que pueden aplicarse sobre los valores dependen de los dominios de estos valores; es decir, un operador determinado sólo se puede aplicar sobre valores que tengan dominios que le sean adecuados.

Ejemplo

Analizaremos esta segunda condición de la regla de integridad de dominio con un ejemplo concreto. Si en la relación EMPLEADOS(DNI, nombre, apellido, edademp) se ha declarado que dominio(DNI) es el dominio predefinido de los enteros, entonces no se permitirá consultar todos aquellos empleados cuyo DNI sea igual a 'Elena' (DNI = 'Elena'). El motivo es que no tiene sentido que el operador de comparación = se aplique entre un DNI que tiene por dominio los enteros, y el valor 'Elena', que es una serie de caracteres.

De este modo, el hecho de que los operadores que se pueden aplicar sobre los valores dependan del dominio de estos valores permite detectar errores que se podrían cometer cuando se consulta o se actualiza la base de datos. Los dominios definidos por el usuario son muy útiles, porque nos permitirán determinar de forma más específica cuáles serán los operadores que se podrán aplicar sobre los valores.

Ejemplo

Veamos otro ejemplo con dominios definidos por el usuario. Supongamos que en la conocida relación EMPLEADOS (DNI, nombre, apellido, edademp) se ha declarado que dominio(DNI) es el dominio definido por el usuario númerosDNI y que dominio(edademp) es el dominio definido por el usuario edad. Supongamos que númerosDNI corresponde a los enteros positivos y que edad corresponde a los enteros que están entre 16 y 65. En este caso, será incorrecto, por ejemplo, consultar los empleados que tienen el valor de DNI igual al valor de edademp.

El motivo es que, aunque tanto los valores de DNI como los de edademp sean enteros, sus dominios son diferentes; por ello, según el significado que el usuario les da, no tiene sentido compararlos.

Sin embargo, los actuales SGBD relacionales no dan apoyo a la segunda condición de la regla de integridad de dominio para dominios definidos por el usuario. Si se quisiera hacer, sería necesario que el diseñador tuviese alguna forma de especificar, para cada operador que se desease utilizar, para qué combinaciones de dominios definidos por el usuario tiene sentido que se aplique.

El lenguaje estándar SQL no incluye actualmente esta posibilidad.

Nivel: Quinto año

El álgebra relacional

Como ya hemos comentado en el apartado dedicado a las operaciones del modelo relacional, el álgebra relacional se inspira en la teoría de conjuntos para especificar consultas en una base de datos relacional.

Para especificar una consulta en álgebra relacional, es preciso definir uno o más pasos que sirven para ir construyendo, mediante operaciones de álgebra relacional, una nueva relación que contenga los datos que responden a la consulta a partir de las relaciones almacenadas. Los lenguajes basados en el álgebra relacional son procedimentales, dado que los pasos que forman la consulta describen un procedimiento.

La visión que presentaremos es la de un lenguaje teórico y, por lo tanto, incluiremos sólo sus operaciones fundamentales, y no las construcciones que se podrían añadir a un lenguaje comercial para facilitar cuestiones como por ejemplo el orden de presentación del resultado, el cálculo de datos agregados, etc.

Una característica destacable de todas las operaciones del álgebra relacional es que tanto los operandos como el resultado son relaciones. Esta propiedad se denomina **cierre relacional**.

Implicaciones del cierre relacional

El hecho de que el resultado de una operación del álgebra relacional sea una nueva relación tiene implicaciones importantes:

1. El resultado de una operación puede actuar como operando de otra operación.
2. El resultado de una operación cumplirá todas las características que ya conocemos de las relaciones: no-ordenación de las tuplas, ausencia de tuplas repetidas, etc.

Las operaciones del álgebra relacional han sido clasificadas según distintos criterios; de todos ellos indicamos los tres siguientes:

1) Segundo se pueden expresar o no en términos de otras operaciones.

a) Operaciones primitivas: son aquellas operaciones a partir de las cuales podemos definir el resto. Estas operaciones son la unión, la diferencia, el producto cartesiano, la selección y la proyección.

b) Operaciones no primitivas: el resto de las operaciones del álgebra relacional que no son estrictamente necesarias, porque se pueden expresar en términos de las primitivas; sin embargo, las operaciones no primitivas permiten formular algunas consultas de forma más

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

cómoda. Existen distintas versiones del álgebra relacional, según las operaciones no primitivas que se incluyen. Nosotros estudiaremos las operaciones no primitivas que se utilizan con mayor frecuencia: la intersección y la combinación.

2) Según el número de relaciones que tienen como operandos:

a) Operaciones binarias: son las que tienen dos relaciones como operandos. Son binarias todas las operaciones, excepto la selección y la proyección.

b) Operaciones unarias: son las que tienen una sola relación como operando.

La selección y la proyección son unarias.

3) Según se parecen o no a las operaciones de la teoría de conjuntos:

a) Operaciones conjuntistas: son las que se parecen a las de la teoría de conjuntos. Se trata de la unión, la intersección, la diferencia y el producto cartesiano.

b) Operaciones específicamente relacionales: son el resto de las operaciones; es decir, la selección, la proyección y la combinación.

Como ya hemos comentado anteriormente, las operaciones del álgebra relacional obtienen como resultado una nueva relación. Es decir que si hacemos una operación del álgebra como por ejemplo `EMPLEADOS_ADM U U EMPLEADOS_PROD` para obtener la unión de las relaciones `EMPLEADOS_ADM` y `EMPLEADOS_PROD`, el resultado de la operación es una nueva relación que tiene la unión de las tuplas de las relaciones de partida.

Esta nueva relación debe tener un nombre. En principio, consideramos que su nombre es la misma expresión del álgebra relacional que la obtiene; es decir, la misma expresión `EMPLEADOS_ADM U EMPLEADOS_PROD`. Puesto que este nombre es largo, en ocasiones puede ser interesante cambiarlo por uno más simple. Esto nos facilitará las referencias a la nueva relación, y será especialmente útil en los casos en los que queramos utilizarla como operando de otra operación. Usaremos la operación auxiliar redenominar con este objetivo.

La operación *redenominar*, que denotaremos con el símbolo `:=`, permite asignar un nombre *R* a la relación que resulta de una operación del álgebra relacional; lo hace de la forma siguiente:

$$R := E,$$

siendo *E* la expresión de una operación del álgebra relacional.

En el ejemplo, para dar el nombre `EMPLEADOS` a la relación resultante de la operación `EMPLEADOS_ADM U EMPLEADOS_PROD`, haríamos: `EMPLEADOS := EMPLEADOS_ADM U EMPLEADOS_PROD`.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Cada operación del álgebra relacional da unos nombres por defecto a los atributos del esquema de la relación resultante, tal y como veremos más adelante.

En algunos casos, puede ser necesario cambiar estos nombres por defecto por otros nombres. Por este motivo, también permitiremos cambiar el nombre de la relación y de sus atributos mediante la operación redenominar.

Utilizaremos también la operación *redenombrar* para cambiar el esquema de una relación. Si una relación tiene el esquema $S(B_1, B_2, \dots, B_n)$ y queremos cambiarlo por $R(A_1, A_2, \dots, A_n)$, lo haremos de la siguiente forma:

$$R(A_1, A_2, \dots, A_n) := S(B_1, B_2, \dots, B_n).$$

A continuación presentaremos un ejemplo que utilizaremos para ilustrar las operaciones del álgebra relacional. Después veremos con detalle las operaciones.

Supongamos que tenemos una base de datos relacional con las cuatro relaciones siguientes:

- 1) La relación EDIFICIOS_EMP, que contiene datos de distintos edificios de los que una empresa dispone para desarrollar sus actividades.
- 2) La relación DESPACHOS, que contiene datos de cada uno de los despachos que hay en los edificios anteriores.
- 3) La relación EMPLEADOS_ADM, que contiene los datos de los empleados de la empresa que llevan a cabo tareas administrativas.
- 4) La relación EMPLEADOS_PROD, que almacena los datos de los empleados de la empresa que se ocupan de tareas de producción.

A continuación describimos los esquemas de las relaciones anteriores y sus extensiones en un momento determinado:

- Esquema y extensión de EDIFICIOS_EMP:

EDIFICIOS_EMP	
<u>edificio</u>	<u>supmediadesp</u>
Marina	15
Diagonal	10

- Esquema y extensión de DESPACHOS:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

DESPACHOS		
<u>edificio</u>	<u>número</u>	<u>superficie</u>
Marina	120	10
Marina	230	20
Diagonal	120	10
Diagonal	440	10

- Esquema y extensión de EMPLEADOS_ADM:

EMPLEADOS_ADM				
<u>DNI</u>	<u>nombre</u>	<u>apellido</u>	<u>edificiodesp</u>	<u>númerodesp</u>
40.444.255	Juan	García	Marina	120
33.567.711	Marta	Roca	Marina	120

- Esquema y extensión de EMPLEADOS_PROD:

EMPLEADOS_PROD				
<u>DNI</u>	<u>nombreemp</u>	<u>apellidoemp</u>	<u>edificiodesp</u>	<u>númerodesp</u>
33.567.711	Marta	Roca	Marina	120
55.898.425	Carlos	Buendía	Diagonal	120
77.232.144	Elena	Pla	Marina	230
21.335.245	Jorge	Soler	NULO	NULO
88.999.210	Pedro	González	NULO	NULO

Se considera que los valores nulos de los atributos edificiodesp y númerodesp de las relaciones EMPLEADOS_PROD y EMPLEADOS_ADM indican que el empleado correspondiente no tiene despacho.

Operaciones conjuntistas

Las operaciones conjuntistas del álgebra relacional son la unión, la intersección, la diferencia y el producto cartesiano.

Unión

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

La unión es una operación que, a partir de dos relaciones, obtiene una nueva relación formada por todas las tuplas que están en alguna de las relaciones de partida.

La unión es una operación binaria, y la unión de dos relaciones T y S se indica $T \cup S$.

La unión de las relaciones EMPLEADOS_ADMIN y EMPLEADOS_PROD proporciona una nueva relación que contiene tanto a los empleados de administración como los empleados de producción; se indicaría así: EMPLEADOS_ADMIN U EMPLEADOS_PROD.

Sólo tiene sentido aplicar la unión a relaciones que tengan tuplas similares.

Por ejemplo, se puede hacer la unión de las relaciones EMPLEADOS_ADMIN y EMPLEADOS_PROD porque sus tuplas se parecen. En cambio, no se podrá hacer la unión de las relaciones EMPLEADOS_ADMIN y DESPACHOS porque, como habéis podido observar en las tablas, las tuplas respectivas son de tipo diferente.

Más concretamente, para poder aplicar la unión a dos relaciones, es preciso que las dos relaciones sean compatibles. Decimos que dos relaciones T y S son relaciones compatibles si:

- Tienen el mismo grado.
- Se puede establecer una biyección entre los atributos de T y los atributos de S que hace corresponder a cada atributo A_i de T un atributo A_j de S , de modo que se cumple que $\text{dominio}(A_i) = \text{dominio}(A_j)$.

Ejemplo de relaciones compatibles

Las relaciones EMPLEADOS_ADMIN y EMPLEADOS_PROD tienen grado 5. Podemos establecer la siguiente biyección entre sus atributos:

- A DNI de EMPLEADOS_ADMIN le corresponde DNIdemp de EMPLEADOS_PROD.
- A nombre de EMPLEADOS_ADMIN le corresponde nombreemp de EMPLEADOS_PROD.
- A apellido de EMPLEADOS_ADMIN le corresponde apellidoemp de EMPLEADOS_PROD.
- A edificiodesp de EMPLEADOS_ADMIN le corresponde edificiodesp de EMPLEADOS_PROD.
- A númerodesp de EMPLEADOS_ADMIN le corresponde edificiodesp de EMPLEADOS_PROD.

Además, supondremos que los dominios de sus atributos se han declarado de forma que se cumple que el dominio de cada atributo de EMPLEADOS_ADMIN sea el mismo que el dominio de su atributo correspondiente en EMPLEADOS_PROD.

Por todos estos factores, podemos llegar a la conclusión de que EMPLEADOS_ADMIN y EMPLEADOS_PROD son relaciones compatibles.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

A continuación, pasaremos a definir los atributos y la extensión de la relación resultante de una unión.

Los atributos del esquema de la relación resultante de $T \cup S$ coinciden con los atributos del esquema de la relación T .

La extensión de la relación resultante de $T \cup S$ es el conjunto de tuplas que pertenecen a la extensión de T , a la extensión de S o a la extensión de ambas relaciones.

No-repetición de tuplas

Notad que en caso de que una misma tupla esté en las dos relaciones que se unen, el resultado de la unión no la tendrá repetida. El resultado de la unión es una nueva relación por lo que no puede tener repeticiones de tuplas.

Ejemplo de unión

Si queremos obtener una relación R que tenga a todos los empleados de la empresa del ejemplo anterior, llevaremos a cabo la unión de las relaciones EMPLEADOS_ADM y EMPLEADOS_PROD de la forma siguiente: $R := \text{EMPLEADOS_ADM} \cup \text{EMPLEADOS_PROD}$.

Entonces la relación R resultante será la reflejada en la tabla siguiente:

R				
DNI	nombre	apellido	edificiodesp	númerodesp
40.444.255	Juan	García	Marina	120
33.567.711	Marta	Roca	Marina	120
55.898.425	Carlos	Buendía	Diagonal	120

R				
DNI	nombre	apellido	edificiodesp	númerodesp
77.232.144	Elena	Pla	Marina	230
21.335.245	Jorge	Soler	NULO	NULO
88.999.210	Pedro	González	NULO	NULO

El hecho de que los atributos de la relación resultante coincidan con los atributos de la relación que figura en primer lugar en la unión es una convención; teóricamente, también habría sido posible convenir que coincidiesen con los de la relación que figura en segundo lugar.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Intersección

La intersección es una operación que, a partir de dos relaciones, obtiene una nueva relación formada por las tuplas que pertenecen a las dos relaciones de partida.

La intersección es una operación binaria; la intersección de dos relaciones T y S se indica $T \cap S$.

La intersección de las relaciones EMPLEADOS_ADM y EMPLEADOS_PROD obtiene una nueva relación que incluye a los empleados que son al mismo tiempo de administración y de producción: se indicaría como $\text{EMPLEADOS_ADM} \cap \text{EMPLEADOS_PROD}$.

La intersección, como la unión, sólo se puede aplicar a relaciones que tengan tuplas similares. Para poder hacer la intersección de dos relaciones, es preciso, pues, que las relaciones sean compatibles.

A continuación definiremos los atributos y la extensión de la relación resultante de una intersección.

Los atributos del esquema de la relación resultante de $T \cap S$ coinciden con los atributos del esquema de la relación T .

La extensión de la relación resultante de $T \cap S$ es el conjunto de tuplas que pertenecen a la extensión de ambas relaciones.

Ejemplo de intersección

Si queremos obtener una relación R que incluya a todos los empleados de la empresa del ejemplo que trabajan tanto en administración como en producción, realizaremos la intersección de las relaciones EMPLEADOS_ADM y EMPLEADOS_PROD de la forma siguiente:

$R := \text{EMPLEADOS_ADM} \cap \text{EMPLEADOS_PROD}$.

Entonces, la relación R resultante será:

R				
DNI	nombre	apellido	edificiodesp	númerodesp
33.567.711	Marta	Roca	Marina	120

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Observad que se ha tomado la convención de que los atributos de la relación que resulta coincidan con los atributos de la relación que figura en primer lugar.

Diferencia

La diferencia es una operación que, a partir de dos relaciones, obtiene una nueva relación formada por todas las tuplas que están en la primera relación y, en cambio, no están en la segunda. La diferencia es una operación binaria, y la diferencia entre las relaciones T y S se indica como $T - S$.

La diferencia EMPLEADOS_ADM menos EMPLEADOS_PROD da como resultado una nueva relación que contiene a los empleados de administración que no son empleados de producción, y se indicaría de este modo: EMPLEADOS_ADM – EMPLEADOS_PROD.

La diferencia, como ocurría en la unión y la intersección, sólo tiene sentido si se aplica a relaciones que tengan tuplas similares. Para poder realizar la diferencia de dos relaciones es necesario que las relaciones sean compatibles.

A continuación definimos los atributos y la extensión de la relación resultante de una diferencia.

Los atributos del esquema de la relación resultante de $T - S$ coinciden con los atributos del esquema de la relación T .

La extensión de la relación resultante de $T - S$ es el conjunto de tuplas que pertenecen a la extensión de T , pero no a la de S .

Ejemplo de diferencia

Si queremos obtener una relación R con todos los empleados de la empresa del ejemplo que trabajan en administración, pero no en producción, haremos la diferencia de las relaciones

EMPLEADOS_ADM y EMPLEADOS_PROD de la forma siguiente:

$R := \text{EMPLEADOS_ADM} - \text{EMPLEADOS_PROD}$

Entonces la relación R resultante será:

R				
DNI	nombre	apellido	edificiodesp	númerodesp
40.444.255	Juan	García	Marina	120

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Se ha tomado la convención de que los atributos de la relación resultante coincidan con los atributos de la relación que figura en primer lugar.

Producto cartesiano

El producto cartesiano es una operación que, a partir de dos relaciones, obtiene una nueva relación formada por todas las tuplas que resultan de concatenar tuplas de la primera relación con tuplas de la segunda.

El producto cartesiano es una operación binaria. Siendo T y S dos relaciones que cumplen que sus esquemas no tienen ningún nombre de atributo común, el producto cartesiano de T y S se indica como $T \times S$.

Si calculamos el producto cartesiano de EDIFICIOS_EMP y DESPACHOS, obtendremos una nueva relación que contiene todas las concatenaciones posibles de tuplas de EDIFICIOS_EMP con tuplas de DESPACHOS.

Si se quiere calcular el producto cartesiano de dos relaciones que tienen algún nombre de atributo común, sólo hace falta redenominar previamente los atributos adecuados de una de las dos relaciones.

A continuación definimos los atributos y la extensión de la relación resultante de un producto cartesiano.

Los atributos del esquema de la relación resultante de $T \times S$ son todos los atributos de T y todos los atributos de S *.

* Recordad que T y S no tienen ningún nombre de atributo común.

La extensión de la relación resultante de $T \times S$ es el conjunto de todas las tuplas de la forma $\langle v_1, v_2, \dots, v_n, w_1, w_2, \dots, w_m \rangle$ para las que se cumple que $\langle v_1, v_2, \dots, v_n \rangle$ pertenece a la extensión de T y que $\langle w_1, w_2, \dots, w_m \rangle$ pertenece a la extensión de S .

Ejemplo de producto cartesiano

El producto cartesiano de las relaciones DESPACHOS y EDIFICIOS_EMP del ejemplo se puede hacer como se indica (es necesario redenominar atributos previamente): EDIFICIOS(nombreedificio, supmediadesp) := EDICIOS_EMP(edificio, supmediadesp).

R := EDIFICIOS \times DESPACHOS.

Entonces, la relación R resultante será:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

R				
<i>nombreedificio</i>	<i>supmediadesp</i>	<i>edificio</i>	<i>número</i>	<i>superficie</i>
Marina	15	Marina	120	10
Marina	15	Marina	230	20
Marina	15	Diagonal	120	10
Marina	15	Diagonal	440	10
Diagonal	10	Marina	120	10

R				
<i>nombreedificio</i>	<i>supmediadesp</i>	<i>edificio</i>	<i>número</i>	<i>superficie</i>
Diagonal	10	Marina	230	20
Diagonal	10	Diagonal	120	10
Diagonal	10	Diagonal	440	10

Conviene señalar que el producto cartesiano es una operación que raramente se utiliza de forma explícita, porque el resultado que da no suele ser útil para resolver las consultas habituales.

A pesar de ello, el producto cartesiano se incluye en el álgebra relacional porque es una operación primitiva; a partir de la cual se define otra operación del álgebra, la combinación, que se utiliza con mucha frecuencia.

Operaciones específicamente relacionales

Las operaciones específicamente relacionales son la selección, la proyección y la combinación.

Selección

Podemos ver la selección como una operación que sirve para elegir algunas tuplas de una relación y eliminar el resto. Más concretamente, la selección es una operación que, a partir de una relación, obtiene una nueva relación formada por todas las tuplas de la relación de partida que cumplen una condición de selección especificada.

La selección es una operación unaria. Siendo C una condición de selección, la selección de T con la condición C se indica como $T(C)$.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Para obtener una relación que tenga todos los despachos del edificio Marina que tienen más de 12 metros cuadrados, podemos aplicar una selección a la relación DESPACHOS con una condición de selección que sea $\text{edificio} = \text{Marina}$ y $\text{superficie} > 12$; se indicaría $\text{DESPACHOS}(\text{edificio} = \text{Marina} \text{ y } \text{superficie} > 12)$.

En general, la condición de selección C está formada por una o más cláusulas de la forma:

$A_i \theta v$, o bien:

$A_i \theta A_j$, donde A_i y A_j son atributos de la relación T, θ es un operador de comparación* y v es un valor. Además, se cumple que:

- En las cláusulas de la forma $A_i \theta v$, v es un valor del dominio de A_i
- En las cláusulas de la forma $A_i \theta A_j$, A_i y A_j tienen el mismo dominio.

Las cláusulas que forman una condición de selección se conectan con los siguientes operadores booleanos: "y" (\wedge) y "o" (\vee).

A continuación definimos los atributos y la extensión de la relación resultante de una selección.

Los atributos del esquema de la relación resultante de $T(C)$ coinciden con los atributos del esquema de la relación T .

La extensión de la relación resultante de $T(C)$ es el conjunto de tuplas que pertenecen a la extensión de T y que satisfacen la condición de selección C . Una tupla t satisface una condición de selección C si, después de sustituir cada atributo que hay en C por su valor en t , la condición C se evalúa en el valor cierto.

Ejemplo de selección

Si queremos obtener una relación R con los despachos de la base de datos del ejemplo que están en el edificio Marina y que tienen una superficie de más de 12 metros cuadrados, haremos la siguiente selección:

$R := \text{DESPACHOS}(\text{edificio} = \text{Marina} \text{ y } \text{superficie} > 12)$.

La relación R resultante será:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

R		
edificio	número	superficie
Marina	230	20

Proyección

Podemos considerar la proyección como una operación que sirve para elegir algunos atributos de una relación y eliminar el resto. Más concretamente, la proyección es una operación que, a partir de una relación, obtiene una nueva relación formada por todas las (sub) tuplas de la relación de partida que resultan de eliminar unos atributos especificados.

La proyección es una operación unaria. Siendo $\{A_i, A_p, \dots, A_k\}$ un subconjunto de los atributos del esquema de la relación T , la proyección de T sobre $\{A_i, A_p, \dots, A_k\}$ se indica como $T[A_i, A_p, \dots, A_k]$.

Para obtener una relación que tenga sólo los atributos nombre y apellido de los empleados de administración, podemos hacer una proyección en la relación EMPLEADOS_ADMIN sobre estos dos atributos. Se indicaría de la forma siguiente: EMPLEADOS_ADMIN [nombre, apellido].

A continuación definiremos los atributos y la extensión de la relación resultante de una proyección.

Los atributos del esquema de la relación resultante de $T[A_i, A_p, \dots, A_k]$ son los atributos $\{A_i, A_p, \dots, A_k\}$.

La extensión de la relación resultante de $T[A_i, A_p, \dots, A_k]$ es el conjunto de todas las tuplas de la forma $\langle t.A_i, t.A_p, \dots, t.A_k \rangle$, donde se cumple que t es una tupla de la extensión de T y donde $t.A_p$ denota el valor para el atributo A_p de la tupla t .

Eliminación de las tuplas repetidas

Notad que la proyección elimina implícitamente todas las tuplas repetidas. El resultado de una proyección es una relación válida y no puede tener repeticiones de tuplas.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Ejemplo de proyección

Si queremos obtener una relación R con el nombre y el apellido de todos los empleados de administración de la base de datos del ejemplo, haremos la siguiente proyección:

$R := \text{EMPLEADOS_ADM}[\text{nombre, apellido}]$.

Entonces, la relación R resultante será:

R	
nombre	apellido
Juan	García
Marta	Roca

Combinación

La combinación es una operación que, a partir de dos relaciones, obtiene una nueva relación formada por todas las tuplas que resultan de concatenar tuplas de la primera relación con tuplas de la segunda, y que cumplen una condición de combinación especificada.

La combinación es una operación binaria. Siendo T y S dos relaciones cuyos esquemas no tienen ningún nombre de atributo común, y siendo B una condición de combinación, la combinación de T y S según la condición B se indica $T[B]S$.

Para conseguir una relación que tenga los datos de cada uno de los empleados de administración junto con los datos de los despachos donde trabajan, podemos hacer una combinación de las relaciones `EMPLEADOS_ADM` y `DESPACHOS`, donde la condición de combinación indique lo siguiente: `edificiodesp = edificio` y `númerodesp = número`. La condición de combinación hace que el resultado sólo combine los datos de un empleado con los datos de un despacho si el `edificiodesp` y el `númerodesp` del empleado son iguales que el `edificio` y el `número` del despacho, respectivamente. Es decir, la condición hace que los datos de un empleado se combinen con los datos del despacho donde trabaja, pero no con datos de otros despachos.

La combinación del ejemplo anterior se indicaría de la forma siguiente:

`EMPLEADOS_ADM[edificiodesp = edificio, númerodesp = número]DESPACHOS`.

Si se quiere combinar dos relaciones que tienen algún nombre de atributo común, sólo hace falta redenominar previamente los atributos repetidos de una de las dos.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

En general, la condición B de una combinación T[B]S está formada por una o más comparaciones de la forma

$A_i \theta A_j$,

donde A_i es un atributo de la relación T, A_j es un atributo de la relación S, θ es un operador de comparación ($=, \neq, <, \leq, >, \geq$), y se cumple que A_i y A_j tienen el mismo dominio. Las comparaciones de una condición de combinación se separan mediante comas.

A continuación definimos los atributos y la extensión de la relación resultante de una combinación.

Los atributos del esquema de la relación resultante de $T[B]S$ son todos los atributos de T y todos los atributos de S *.

* Recuérdese que T y S no tienen ningún nombre de atributo común.

La extensión de la relación resultante de $T[B]S$ es el conjunto de tuplas que pertenecen a la extensión del producto cartesiano $T \times S$ y que satisfacen todas las comparaciones que forman la condición de combinación B. Una tupla t satisface una comparación si, después de sustituir cada atributo que figura en la comparación por su valor en t , la comparación se evalúa al valor cierto.

Ejemplo de combinación

Supongamos que se desea encontrar los datos de los despachos que tienen una superficie mayor o igual que la superficie media de los despachos del edificio donde están situados. La siguiente combinación nos proporcionará los datos de estos despachos junto con los datos de su edificio (observad que es preciso redenominar previamente los atributos):

$EDIFICIOS(nombreedificio,supmediadesp) := EDIFICIOS_EMP(edificio, supmediadesp)$,

$R := EDIFICIOS[nombreedificio = edificio, supmediadesp \leq superficie] DESPACHOS$.

Entonces, la relación R resultante será:

R				
nombreedificio	supmediadesp	edificio	número	superficie
Marina	15	Marina	230	20
Diagonal	10	Diagonal	120	10
Diagonal	10	Diagonal	440	10

Supongamos ahora que para obtener los datos de cada uno de los empleados de administración, junto con los datos del despacho donde trabajan, utilizamos la siguiente combinación:

$R := EMPLEADOS_ADM[edificiodesp = edificio, númerodesp = número]DESPACHOS$.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

La relación R resultante será:

R							
DNI	nombre	apellido	edificiodesp	númerodesp	edificio	número	superficie
40.444.255	Juan	García	Marina	120	Marina	120	10
33.567.711	Marta	Roca	Marina	120	Marina	120	10

La relación R combina los datos de cada empleado con los datos de su despacho.

En ocasiones, la combinación recibe el nombre de θ-combinación, y cuando todas las comparaciones de la condición de la combinación tienen el operador “=”, se denomina equicombinación.

Según esto, la combinación del último ejemplo es una equicombinación.

Observad que el resultado de una equicombinación siempre incluye una o más parejas de atributos que tienen valores idénticos en todas las tuplas.

En el ejemplo anterior, los valores de edificiodesp coinciden con los de edificio, y los valores de númerodesp coinciden con los de número.

Puesto que uno de cada par de atributos es superfluo, se ha establecido una variante de combinación denominada combinación natural, con el fin de eliminarlos.

La combinación natural de dos relaciones T y S se denota como $T \cdot S$ y consiste básicamente en una equicombinación seguida de la eliminación de los atributos superfluos; además, se considera por defecto que la condición de combinación iguala todas las parejas de atributos que tienen el mismo nombre en T y en S .

Observad que, a diferencia de la equicombinación, la combinación natural se aplica a relaciones que tienen nombres de atributos comunes.

Ejemplo de combinación natural

Si hacemos:

$R := EDIFICIOS_EMP * DESPACHOS$, se considera que la condición es $edificio = edificio$ porque $edificio$ es el único nombre de atributo que figura tanto en el esquema de $EDIFICIOS_EMP$ como en el esquema de $DESPACHOS$.

El resultado de esta combinación natural es:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

R			
<i>edificio</i>	<i>supmediodesp</i>	<i>número</i>	<i>superficie</i>
Marina	15	120	10
Marina	15	230	20
Diagonal	10	120	10
Diagonal	10	440	10

Notad que se ha eliminado uno de los atributos de nombre edificio.

En ocasiones, antes de la combinación natural es necesario aplicar la operación redenominar para hacer coincidir los nombres de los atributos que nos interesa igualar.

Ejemplo de combinación natural con redenominación

Por ejemplo, si queremos obtener los datos de cada uno de los empleados de administración junto con los datos del despacho donde trabajan pero sin repetir valores de atributos superfluos, haremos la siguiente combinación natural, que requiere una redenominación previa:

$D(\text{edificiodesp}, \text{númerodesp}, \text{superficie}) := \text{DESPACHOS}(\text{edificio}, \text{número}, \text{superficie}),$

$R := \text{EMPLEADOS_ADM} * D.$

Entonces, la relación R resultante será:

R					
<i>DNI</i>	<i>nombre</i>	<i>apellido</i>	<i>edificiodesp</i>	<i>númerodesp</i>	<i>superficie</i>
40.444.255	Juan	García	Marina	120	10
33.567.711	Marta	Roca	Marina	120	10

Secuencias de operaciones del álgebra relacional

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

En muchos casos, para formular una consulta en álgebra relacional es preciso utilizar varias operaciones, que se aplican en un cierto orden. Para hacerlo, hay dos posibilidades:

- 1) Utilizar una sola expresión del álgebra que incluya todas las operaciones con los paréntesis necesarios para indicar el orden de aplicación.
- 2) Descomponer la expresión en varios pasos donde cada paso aplique una sola operación y obtenga una relación intermedia que se pueda utilizar en los pasos subsiguientes.

Ejemplo de utilización de secuencias de operaciones

Para obtener el nombre y el apellido de los empleados, tanto de administración como de producción, es necesario hacer una unión de EMPLEADOS_ADM y EMPLEADOS_PROD, y después hacer una proyección sobre los atributos nombre y apellido. La operación se puede expresar de las formas siguientes:

a) Se puede utilizar una sola expresión:

R := (EMPLEADOS_ADM U EMPLEADOS_PROD) [nombre, apellido].

b) O bien podemos expresarlo en dos pasos:

- EMPS := EMPLEADOS_ADM U EMPLEADOS_PROD;
- R := EMPS[nombre, apellido]

En los casos en que una consulta requiere efectuar muchas operaciones, resulta más sencilla la segunda alternativa, porque evita expresiones complejas.

Otros ejemplos de consultas formuladas con secuencias de operaciones

Veamos algunos ejemplos de consultas en la base de datos formuladas con secuencias de operaciones del álgebra relacional.

1) Para obtener el nombre del edificio y el número de los despachos situados en edificios en los que la superficie media de estos despachos es mayor que 12, podemos utilizar la siguiente secuencia de operaciones:

- A := EDIFICIOS_EMP(supmediadesp > 12);
- B := DESPACHOS * A;
- R := B[edificio, número]

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

2) Supongamos ahora que se desea obtener el nombre y el apellido de todos los empleados (tanto de administración como de producción) que están asignados al despacho 120 del edificio Marina. En este caso, podemos utilizar la siguiente secuencia:

- A := EMPLEADOS_ADM U EMPLEADOS_PROD;
- B := A(edificiodesp = Marina y númerodesp = 120);
- R := B[nombre, apellido].

3) Si queremos consultar el nombre del edificio y el número de los despachos que ningún empleado de administración tiene asignado, podemos utilizar esta secuencia:

- A := DESPACHOS [edificio, número];
- B := EMPLEADOS_ADM[edificiodesp, númerodesp];
- R := A - B.

4) Para obtener el DNI, el nombre y el apellido de todos los empleados de administración que tienen despacho, junto con la superficie de su despacho, podemos hacer lo siguiente:

- A[DNI, nombre, apellido, edificio, número] := EMPLEADOS_ADM[DNI, nombre, apellido, edificiodesp, númerodesp];
- B := A * DESPACHOS;
- R := B[DNI, nombre, apellido, superficie].

5.4. Extensiones: combinaciones externas

Para finalizar el tema del álgebra relacional, analizaremos algunas extensiones útiles de la combinación.

Las combinaciones que se han descrito obtienen las tuplas del producto cartesiano de dos relaciones que satisfacen una condición de combinación. Las tuplas de una de las dos relaciones que no tienen en la otra relación una tupla como mínimo con la cual, una vez concatenadas, satisfagan la condición de combinación, no aparecen en el resultado de la combinación, y podríamos decir que sus datos se pierden.

Por ejemplo, si hacemos la siguiente combinación natural (con una redenominación previa):

D(edificiodesp, númerodesp, superficie) := DESPACHOS (edificio, número, superficie),
R := EMPLEADOS_PROD * D.

Puesto que se trata de una combinación natural, se considera que la condición de combinación es edificio = edificio y número = número, y la relación R resultante será:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

R					
<i>DNtemp</i>	<i>nombreemp</i>	<i>apellidoemp</i>	<i>edificiodesp</i>	<i>númerodesp</i>	<i>superficie</i>
33.567.711	Marta	Roca	Marina	120	10
55.898.425	Carlos	Buendía	Diagonal	120	10
77.232.144	Elena	Pla	Marina	230	20

Notad que en esta relación R no están los empleados de producción que no tienen despacho asignado (con valores nulos en edificiodesp y númerodesp), y tampoco los despachos que no tienen ningún empleado de producción, porque no cumplen la condición de combinación.

Conviene destacar que las tuplas que tienen un valor nulo para alguno de los atributos que figuran en la condición de combinación se pierden siempre, porque en estos casos la condición de combinación siempre se evalúa a falso.

En algunos casos, puede interesar hacer combinaciones de los datos de dos relaciones sin que haya pérdida de datos de las relaciones de partida. Entonces, se utilizan las combinaciones externas.

Las combinaciones externas entre dos relaciones *T* y *S* consisten en variantes de combinación que conservan en el resultado todas las tuplas de *T*, de *S* o de ambas relaciones. Pueden ser de los tipos siguientes:

- 1) La combinación externa izquierda entre dos relaciones *T* y *S*, que denotamos como $T[C]_lS$, conserva en el resultado todas las tuplas de la relación *T*.
- 2) La combinación externa derecha entre dos relaciones *T* y *S*, que denotamos como $T[C]_dS$, conserva en el resultado todas las tuplas de la relación *S*.
- 3) Finalmente, la combinación externa plena entre dos relaciones *T* y *S*, que denotamos como $T[C]_pS$, conserva en el resultado todas las tuplas de *T* y todas las tuplas de *S*.

Estas extensiones también se aplican al caso de la combinación natural entre dos relaciones, *T* * *S*, concretamente:

- a) La combinación natural externa izquierda entre dos relaciones *T* y *S*, que se indica como $T*I$ *S*, conserva en el resultado todas las tuplas de la relación *T*.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

b) La combinación natural externa derecha entre dos relaciones T y S, que se indica como $T *D S$, conserva en el resultado todas las tuplas de la relación S.

c) Finalmente, la combinación natural externa plena entre dos relaciones T y S, que se indica como $T *P S$, conserva en el resultado todas las tuplas de T y todas las tuplas de S.

Las tuplas de una relación T que se conservan en el resultado R de una combinación externa con otra relación S, a pesar de que no satisfacen la condición de combinación, tienen valores nulos en el resultado R para todos los atributos que provienen de la relación S.

Ejemplos de combinaciones naturales externas

1) Si hacemos la siguiente combinación natural derecha (con una redenominación previa):

$D(\text{edificiodesp}, \text{númerodesp}, \text{superficie}) := \text{DESPACHOS}(\text{edificio}, \text{número}, \text{superficie})$,

$R := \text{EMPLADOS_PROD} *D D$,

la relación R resultante será:

R					
<i>DNIemp</i>	<i>nombreemp</i>	<i>apellidoemp</i>	<i>edificiodesp</i>	<i>númerodesp</i>	<i>superficie</i>
33.567.711	Marta	Roca	Marina	120	10
55.898.425	Carlos	Buendía	Diagonal	120	10
77.232.144	Elena	Pla	Marina	230	20
NULO	NULO	NULO	Diagonal	440	10

Ahora obtenemos todos los despachos en la relación resultante, tanto si tienen un empleado de producción asignado como si no. Notad que los atributos DNI, nombre y apellido para los despachos que no tienen empleado reciben valores nulos.

2) Si hacemos la siguiente combinación natural izquierda (con una redenominación previa):

$D(\text{edificiodesp}, \text{númerodesp}, \text{superficie}) := \text{DESPACHOS}(\text{edificio}, \text{número}, \text{superficie})$,

$R := \text{EMPLEADOS_PROD} *I D$,

entonces la relación R resultante será:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

R					
<i>DNIemp</i>	<i>nombreemp</i>	<i>apellidoemp</i>	<i>edificiodesp</i>	<i>númerodesp</i>	<i>superficie</i>
33.567.711	Marta	Roca	Marina	120	10
55.898.425	Carlos	Buendía	Diagonal	120	10
77.232.144	Elena	Pla	Marina	230	20
21.335.245	Jorge	Soler	NULO	NULO	NULO
88.999.210	Pedro	González	NULO	NULO	NULO

Esta combinación externa nos permite obtener en la relación resultante a todos los empleados de producción, tanto si tienen despacho como si no. Observad que el atributo superficie para los empleados que no tienen despacho contiene un valor nulo.

3) Finalmente, si hacemos la siguiente combinación natural plena (con una redenominación previa):

$D(edificiodesp, númerodesp, superficie) := DESPACHOS(edificio, número, superficie),$

$R := \text{EMPLEADOS_PROD} * P D,$

entonces la relación R resultante será:

R					
<i>DNIemp</i>	<i>nombreemp</i>	<i>apellidoemp</i>	<i>edificiodesp</i>	<i>númerodesp</i>	<i>superficie</i>
33.567.711	Marta	Roca	Marina	120	10
55.898.425	Carlos	Buendía	Diagonal	120	10
77.232.144	Elena	Pla	Marina	230	20
21.335.245	Jorge	Soler	NULO	NULO	NULO
88.999.210	Pedro	González	NULO	NULO	NULO
NULO	NULO	NULO	Diagonal	440	10

En este caso, en la relación resultante obtenemos a todos los empleados de producción y también todos los despachos.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Prácticas generales Algebra relacional

1. Dada la relación que corresponde a la siguiente representación tabular:

Figura 4

Relación DESPACHOS		
edificios	números	superficies
edificio	➔ número	superficie
Marina	140	10
Diagonal	250	15
Diagonal	110	NULO

- Indicad qué conjunto de atributos tiene.
- Decid qué dominio tiene cada uno de sus atributos.
- Escribid todas las distintas formas de denotar su esquema de relación.
- Elegid una de las formas de denotar su esquema de relación y utilizadla para dibujar el conjunto de tuplas correspondiente a su extensión.

2. Indicad cuáles son todas las superclaves de las siguientes relaciones:

- DESPACHOS(edificio, número, superficie), que tiene como única clave candidata la siguiente: edificio, número.
- EMPLEADOS(DNI, NSS, nombre, apellido), que tiene las siguientes claves candidatas: DNI y NSS.

3. Decid, para cada una de las siguientes operaciones de actualización, si se podría aceptar su aplicación sobre la base de datos que se ha utilizado en esta unidad:

- Insertar en EDIFICIOS_EMP la tupla < Nexus, 30 >.
- Insertar en DESPACHOS la tupla < Diagonal, NULO, 15 >.
- Insertar en EMPLEADOS_ADM la tupla < 55.555.555, María, Puig, Diagonal, 500 >.
- Modificar en DESPACHOS la tupla < Marina, 230, 20 > por < Marina, 120, 20 >.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

- e) Borrar en EMPLEADOS_PROD la tupla <88.999.20, Pedro, González, NULO, NULO>.
 - f) Modificar en EMPLEADOS_ADMIN la tupla <40.444.255, Juan, García, Marina, 120> por <33.567.711, Juan, García, Marina, 120>.
 - g) Borrar en EDIFICIOS_EMP la tupla <Marina, 15> si para la clave foránea edificio de DESPACHOS se ha seleccionado la política de restricción en caso de borrado.
 - h) Borrar en EDIFICIOS_EMP la tupla <Marina, 15> si para la clave foránea edificio de DESPACHOS se ha seleccionado la política de actualización en cascada en caso de borrado.
4. Escribid secuencias de operaciones del álgebra relacional que resuelvan las siguientes consultas en la base de datos que hemos utilizado en esta unidad:
- a) Obtener los despachos con una superficie mayor que 15. Concretamente, se quiere saber el nombre del edificio, el número y la superficie de estos despachos, junto con la superficie media de los despachos del edificio donde están situados.
 - b) Obtener el nombre del edificio y el número de los despachos que no tienen asignado a ningún empleado (ni de producción ni de administración).
 - c) Obtener el nombre y el apellido de los empleados (tanto de administración como de producción), que no tienen despacho.
 - d) Obtener el nombre y el apellido de todos los empleados (tanto de administración como de producción) que tienen despacho asignado, junto con la superficie de su despacho y la superficie media de los despachos del edificio al que pertenece su despacho.
 - e) Obtener los despachos con una superficie mayor que la superficie del despacho Diagonal, 120. Concretamente, se quiere saber el nombre del edificio y el número de estos despachos.
 - f) Obtener todos los despachos de la empresa (tanto si tienen empleados como si no), junto con los empleados que tienen asignados (en caso de que los tengan). Concretamente, se quiere conocer el nombre del edificio, el número de despacho y el DNI del empleado.

Nivel: Quinto año

Sentencias SQL

El SQL es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la base de datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los sistemas relacionales comerciales.

Empezamos con una breve explicación de la forma en que el SQL ha llegado a ser el lenguaje estándar de las bases de datos relacionales:

1) Al principio de los años setenta, los laboratorios de investigación Santa Teresa de IBM empezaron a trabajar en el proyecto System R. El objetivo de este proyecto era implementar un prototipo de SGBD relacional; por lo tanto, también necesitaban investigar en el campo de los lenguajes de bases de datos relacionales.

A mediados de los años setenta, el proyecto de IBM dio como resultado un primer lenguaje denominado SEQUEL (Structured English Query Language), que por razones legales se denominó más adelante SQL (Structured Query Language).

Al final de la década de los setenta y al principio de la de los ochenta, una vez finalizado el proyecto System R, IBM y otras empresas empezaron a utilizar el SQL en sus SGBD relacionales, con lo que este lenguaje adquirió una gran popularidad.

2) En 1982, ANSI (American National Standards Institute) encargó a uno de sus comités (X3H2) la definición de un lenguaje de bases de datos relacionales. Este comité, después de evaluar diferentes lenguajes, y ante la aceptación comercial del SQL, eligió un lenguaje estándar que estaba basado en éste prácticamente en su totalidad. El SQL se convirtió oficialmente en el lenguaje estándar de ANSI en el año 1986, y de ISO (International Standards Organization) en 1987. También ha sido adoptado como lenguaje estándar por FIPS (Federal Information Processing Standard), Unix X/Open y SAA (Systems Application Architecture) de IBM.

3) En el año 1989, el estándar fue objeto de una revisión y una ampliación que dieron lugar al lenguaje que se conoce con el nombre de SQL1 o SQL89.

En el año 1992 el estándar volvió a ser revisado y ampliado considerablemente para cubrir carencias de la versión anterior. Esta nueva versión del SQL, que se conoce con el nombre de SQL2 o SQL92, es la que nosotros presentaremos en esta unidad didáctica.

Como veremos más adelante, aunque aparezca sólo la sigla SQL, siempre nos estaremos refiriendo al SQL92, ya que éste tiene como subconjunto el SQL89; por lo tanto, todo lo que era válido en el caso del SQL89 lo continuará siendo en el SQL92.

De hecho, se pueden distinguir tres niveles dentro del SQL92:

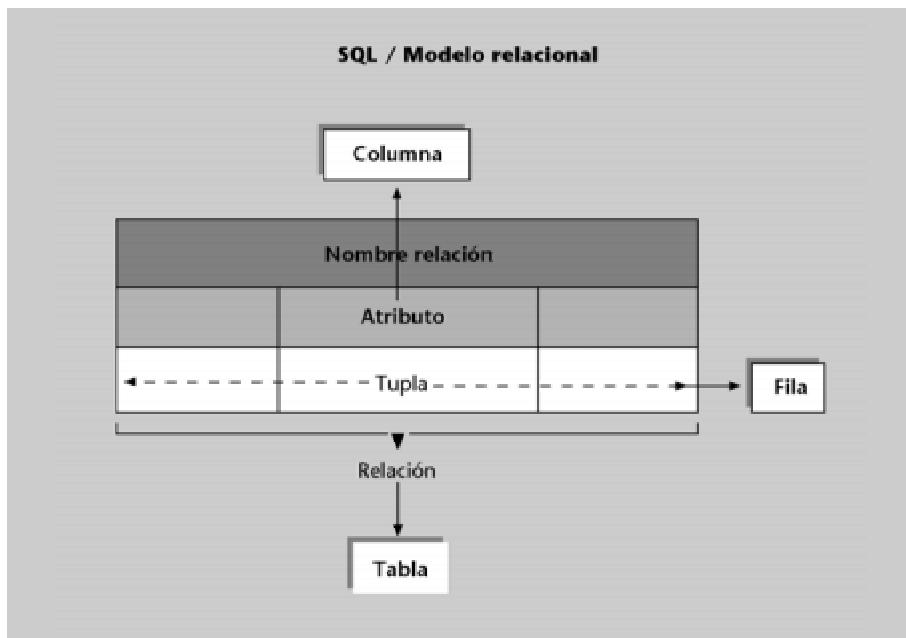
Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

- 1) El nivel introductorio (entry), que incluye el SQL89 y las definiciones de clave primaria y clave foránea al crear una tabla.
- 2) El nivel intermedio (intermediate), que, además del SQL89, añade algunas ampliaciones del SQL92.
- 3) El nivel completo (full), que ya tiene todas las ampliaciones del SQL92.

El modelo relacional tiene como estructura de almacenamiento de los datos las relaciones. La intención o esquema de una relación consiste en el nombre que hemos dado a la relación y un conjunto de atributos. La extensión de una relación es un conjunto de tuplas. Al trabajar con SQL, esta nomenclatura cambia, como podemos apreciar en la siguiente figura:



Hablaremos de tablas en lugar de relaciones.

- Hablaremos de columnas en lugar de atributos.
- Hablaremos de filas en lugar de tuplas.

Sin embargo, a pesar de que la nomenclatura utilizada sea diferente, los conceptos son los mismos.

Con el SQL se puede definir, manipular y controlar una base de datos relacional.

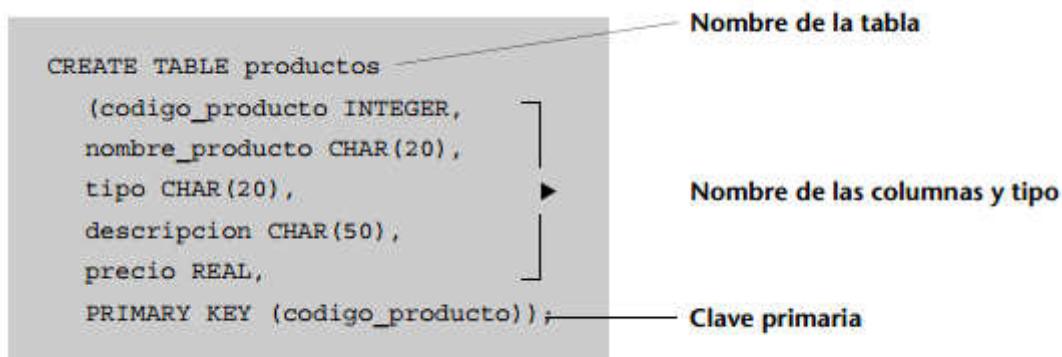
A continuación veremos, aunque sólo en un nivel introductorio, cómo se pueden realizar estas acciones:

- 1) Sería necesario crear una tabla que contuviese los datos de los productos de nuestra empresa:

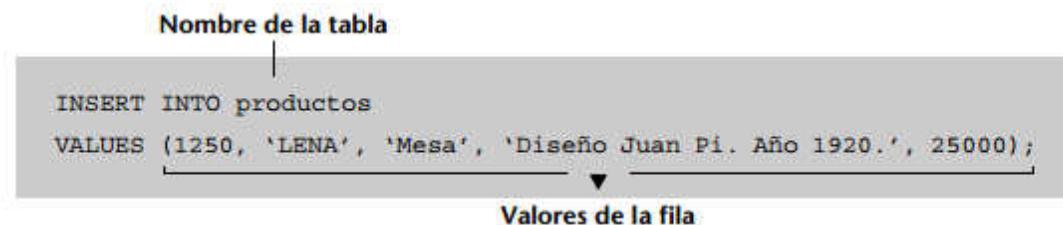
Informática en Desarrollo

CEDES DON BOSCO

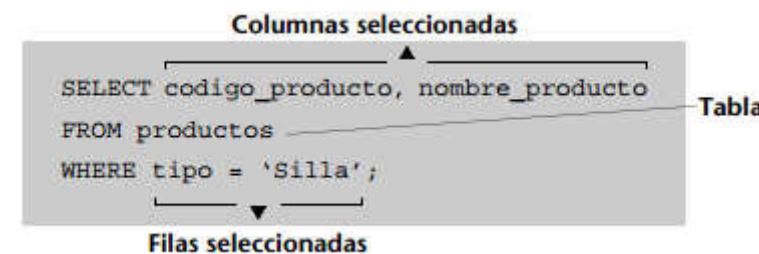
Nivel: Quinto año



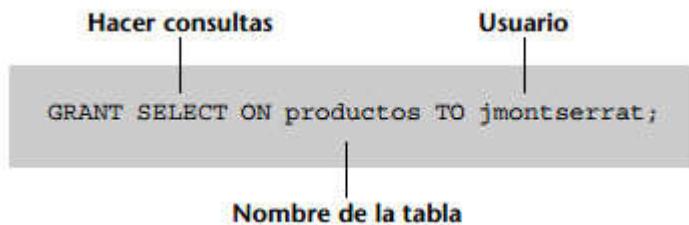
- 2) Insertar un producto en la tabla creada anteriormente:



- 3) Consultar qué productos de nuestra empresa son sillas:



- 4) Dejar acceder a uno de nuestros vendedores a la información de la tabla productos:



Y muchas más cosas que iremos viendo punto por punto en los siguientes apartados.

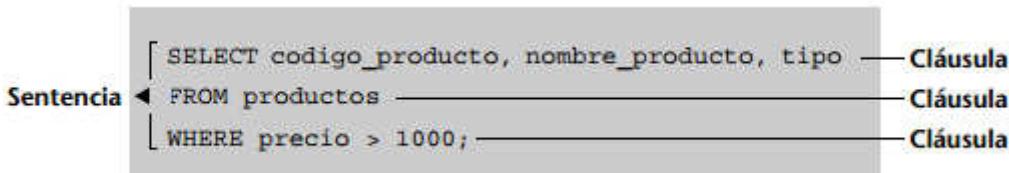
Fijémonos en la estructura de todo lo que hemos hecho hasta ahora con SQL.

Las operaciones de SQL reciben el nombre de sentencias y están formadas por diferentes partes que denominamos cláusulas, tal y como podemos apreciar en el siguiente ejemplo:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



Esta consulta muestra el código, el nombre y el tipo de los productos que cuestan más de 1.000 euros.

Los tres primeros apartados de este módulo tratan sobre un tipo de SQL denominado SQL interactivo, que permite acceder directamente a una base de datos relacional:

- En el primer apartado definiremos las denominadas sentencias de definición, donde crearemos la base de datos, las tablas que la compondrán y los dominios, las aserciones y las vistas que queramos.
- En el segundo aprenderemos a manipular la base de datos, ya sea introduciendo, modificando o borrando valores en las filas de las tablas, o bien haciendo consultas.
- En el tercero veremos las sentencias de control, que aseguran un buen uso de la base de datos.

Sin embargo, muchas veces querremos acceder a la base de datos desde una aplicación hecha en un lenguaje de programación cualquiera, que nos ofrece mucha más potencia fuera del entorno de las bases de datos. Para utilizar SQL desde un lenguaje de programación necesitaremos sentencias especiales que nos permitan distinguir entre las instrucciones del lenguaje de programación y las sentencias de SQL. La idea es que trabajando básicamente con un lenguaje de programación anfitrión se puede cobijar SQL como si fuese un huésped.

Por este motivo, este tipo de SQL se conoce con el nombre de SQL hospedado.

Para trabajar con SQL hospedado necesitamos un precompilador que separe las sentencias del lenguaje de programación de las del lenguaje de bases de datos. Una alternativa a esta forma de trabajar son las rutinas SQL/CLI* (SQL/Call-Level Interface), que resolviendo también el problema de acceder a SQL desde un lenguaje de programación, no necesitan precompilador.

Antes de empezar a conocer el lenguaje, es necesario añadir un último comentario.

Aunque SQL es el lenguaje estándar para bases de datos relacionales y ha sido ampliamente aceptado por los sistemas relacionales comerciales, no ha sido capaz de reflejar toda la teoría del modelo relacional establecida por E.F. Codd; esto lo iremos viendo a medida que profundicemos en el lenguaje.

Los sistemas relacionales comerciales y los investigadores de bases de datos son una referencia muy importante para mantener el estándar actualizado. En estos momentos ya se dispone de una nueva versión de SQL92 que se denomina SQL: 1999 o SQL3. SQL: 1999 tiene a SQL92 como subconjunto, e incorpora nuevas prestaciones de gran interés. En informática, en general, y particularmente en bases de datos, es necesario estar siempre al día, y por eso es muy

Nivel: Quinto año

importante tener el hábito de leer publicaciones periódicas que nos informen y nos mantengan al corriente de las novedades.

Sentencias de definición (DML)

Para poder trabajar con bases de datos relacionales, lo primero que tenemos que hacer es definirlas. Veremos las órdenes del estándar SQL92 para crear y borrar una base de datos relacional y para insertar, borrar y modificar las diferentes tablas que la componen.

En este apartado también veremos cómo se definen los dominios, las aserciones (restricciones) y las vistas.

La sencillez y la homogeneidad del SQL92 hacen que:

- 1) Para crear bases de datos, tablas, dominios, aserciones y vistas se utilice la sentencia CREATE.
- 2) Para modificar tablas y dominios se utilice la sentencia ALTER.
- 3) Para borrar bases de datos, tablas, dominios, aserciones y vistas se utilice la sentencia DROP.

La adecuación de estas sentencias a cada caso nos dará diferencias que iremos perfilando al hacer la descripción individual de cada una.

Para ilustrar la aplicación de las sentencias de SQL que veremos, utilizaremos una base de datos de ejemplo muy sencilla de una pequeña empresa con sede en Barcelona, Girona, Lleida y Tarragona, que se encarga de desarrollar proyectos informáticos. La información que nos interesaría almacenar de esta empresa, que denominaremos BDUOC, será la siguiente:

- 1) Sobre los empleados que trabajan en la empresa, querremos saber su código de empleado, el nombre y apellido, el sueldo, el nombre y la ciudad de su departamento y el número de proyecto al que están asignados.
- 2) Sobre los diferentes departamentos en los que está estructurada la empresa, nos interesa conocer su nombre, la ciudad donde se encuentran y el teléfono.

Será necesario tener en cuenta que un departamento con el mismo nombre puede estar en ciudades diferentes, y que en una misma ciudad puede haber departamentos con nombres diferentes.

- 3) Sobre los proyectos informáticos que se desarrollan, querremos saber su código, el nombre, el precio, la fecha de inicio, la fecha prevista de finalización, la fecha real de finalización y el código de cliente para quien se desarrolla.
- 4) Sobre los clientes para quien trabaja la empresa, querremos saber el código de cliente, el nombre, el NIF, la dirección, la ciudad y el teléfono.

Creación y borrado de una base de datos relacional

El estándar SQL92 no dispone de ninguna sentencia de creación de bases de datos. La idea es que una base de datos no es más que un conjunto de tablas y, por lo tanto, las sentencias que nos ofrece el SQL92 se concentran en la creación, la modificación y el borrado de estas tablas.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

En cambio, disponemos de una sentencia más potente que la de creación de bases de datos: la sentencia de creación de esquemas denominada CREATE SCHEMA.

Con la creación de esquemas podemos agrupar un conjunto de elementos de la base de datos que son propiedad de un usuario. La sintaxis de esta sentencia es la que tenéis a continuación:

```
CREATE SCHEMA { [nombre_esquema] } | [AUTHORIZATION usuario]
[lista_de_elementos_del_esquema];
```

La instrucción

CREATE DATABASE

Muchos de los sistemas relacionales comerciales (como ocurre en el caso de Informix, DB2, SQL Server y otros) han incorporado sentencias de creación de bases de datos con la siguiente sintaxis:

CREATE DATABASE

La nomenclatura utilizada en la sentencia es la siguiente:

- Las palabras en negrita son palabras reservadas del lenguaje:
- La notación [...] quiere decir que lo que hay entre los corchetes se podría poner o no.
- La notación {A| ... |B} quiere decir que tenemos que elegir entre todas las opciones que hay entre las llaves, pero debemos poner una obligatoriamente.

La sentencia de creación de esquemas hace que varias tablas (lista_de_elementos_del_esquema) se puedan agrupar bajo un mismo nombre (nombre_esquema) y que tengan un propietario (usuario). Aunque todos los parámetros de la sentencia CREATE SCHEMA son opcionales, como mínimo se debe dar o bien el nombre del esquema, o bien el nombre del usuario propietario de la base de datos. Si sólo especificamos el usuario, éste será el nombre del esquema.

La creación de esquemas puede hacer mucho más que agrupar tablas, porque lista_de_elementos_del_esquema puede, además de tablas, ser también dominios, vistas, privilegios y restricciones, entre otras cosas.

Para borrar una base de datos encontramos el mismo problema que para crearla.

El estándar SQL92 sólo nos ofrece la sentencia de borrado de esquemas

DROP SCHEMA, que presenta la siguiente sintaxis:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
DROP SCHEMA nombre_esquema {RESTRICT|CASCADE};
```

Donde tenemos lo siguiente:

- La opción de borrado de esquemas RESTRICT hace que el esquema sólo se pueda borrar si no contiene ningún elemento.
- La opción CASCADE borra el esquema aunque no esté completamente vacío.

Creación de tablas

Como ya hemos visto, la estructura de almacenamiento de los datos del modelo relacional son las tablas. Para crear una tabla, es necesario utilizar la sentencia CREATE TABLE. Veamos su formato:

```
CREATE TABLE nombre_tabla
  ( definición_columna
  [, definición_columna...]
  [, restricciones_tabla]
);
```

Donde definición_columna es:

```
nombre_columna {tipo_datos|dominio} [def_defecto] [restric_col]
```

El proceso que hay que seguir para crear una tabla es el siguiente:

- 1) Lo primero que tenemos que hacer es decidir qué nombre queremos poner a la tabla (nombre_tabla).
- 2) Después, iremos dando el nombre de cada uno de los atributos que formarán las columnas de la tabla (nombre_columna).
- 3) A cada una de las columnas le asignaremos un tipo de datos predefinido o bien un dominio definido por el usuario. También podremos dar definiciones por defecto y restricciones de columna.
- 4) Una vez definidas las columnas, sólo nos quedará dar las restricciones de tabla.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Tipos de datos

Para cada columna tenemos que elegir entre algún dominio definido por el usuario o alguno de los tipos de datos predefinidos que se describen a continuación:

Tipos de datos predefinidos	
Tipos de datos	Descripción
CHARACTER (longitud)	Cadenas de caracteres de longitud fija.
CHARACTER VARYING (longitud)	Cadenas de caracteres de longitud variable.

Tipos de datos predefinidos	
Tipos de datos	Descripción
BIT (longitud)	Cadenas de bits de longitud fija.
BIT VARYING (longitud)	Cadenas de bits de longitud variables.
NUMERIC (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
DECIMAL (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
INTEGER	Números enteros.
SMALLINT	Números enteros pequeños.
REAL	Números con coma flotante con precisión predefinida.
FLOAT (precisión)	Números con coma flotante con la precisión especificada.
DOUBLE PRECISION	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas. Están compuestas de: YEAR año, MONTH mes, DAY día.
TIME	Horas. Están compuestas de HOUR hora, MINUT minutos, SECOND segundos.
TIMESTAMP	Fechas y horas. Están compuestas de YEAR año, MONTH mes, DAY día, HOUR hora, MINUT minutos, SECOND segundos.

Ejemplos de asignaciones de columnas

Veamos algunos ejemplos de asignaciones de columnas en los tipos de datos predefinidos

DATE, TIME y TIMESTAMP:

- La columna fecha_nacimiento podría ser del tipo DATE y podría tener como valor '1978-12-25'.
- La columna inicio_partido podría ser del tipo TIME y podría tener como valor '17:15:00.000000'.
- La columna entrada_trabajo podría ser de tipo TIMESTAMP y podría tener como valor '1998-7-8 9:30:05'.

Los tipos de datos NUMERIC y DECIMAL

NUMERIC y DECIMAL se describen igual, y es posible utilizar tanto el uno como el otro para definir números decimales.

El tratamiento del tiempo

El estándar SQL92 define la siguiente nomenclatura para trabajar con el tiempo:

YEAR	(0001..9999)
MONTH	(01..12)
DAY	(01..31)
HOUR	(00..23)
MINUT	(00..59)
SECOND	(00..59.precision)

De todos modos, los sistemas relacionales comerciales disponen de diferentes formatos, entre los cuales podemos elegir cuando tenemos que trabajar con columnas temporales.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Creación de dominios

Además de los dominios dados por el tipo de datos predefinidos, el SQL92 nos ofrece la posibilidad de trabajar con dominios definidos por el usuario.

Para crear un dominio es necesario utilizar la sentencia CREATE DOMAIN:

```
CREATE DOMAIN nombre_dominio [AS] tipos_datos
    [def_defecto] [restricciones_dominio];
```

donde restricciones_dominio tiene el siguiente formato:

```
[CONSTRAINT nombre_restricción] CHECK (condiciones)
```

Modificación y borrado de tablas

Para modificar una tabla es preciso utilizar la sentencia ALTER TABLE.

Veamos su formato:

```
ALTER TABLE nombre_tabla {acción_modificar_columna|
    acción_modif_restricción_tabla};
```

En este caso, tenemos que:

- acción_modificar_columna puede ser:

```
{ADD [COLUMN] columna def_columna |
ALTER [COLUMN] columna {SET def_defecto|DROP DEFAULT}|
DROP [COLUMN] columna {RESTRICT|CASCADE}}}
```

- acción_modif_restricción_tabla puede ser:

```
{ADD restricción|
DROP CONSTRAINT restricción {RESTRICT|CASCADE}}}
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Si queremos modificar una tabla es que queremos realizar una de las siguientes operaciones:

- 1) Añadirle una columna (ADD columna).
- 2) Modificar las definiciones por defecto de la columna (ALTER columna).
- 3) Borrar la columna (DROP columna).
- 4) Añadir alguna nueva restricción de tabla (ADD restricción).
- 5) Borrar alguna restricción de tabla (DROPCONSTRAINT restricción).

Para borrar una tabla es preciso utilizar la sentencia DROP TABLE:

```
DROP TABLE nombre_tabla {RESTRICT|CASCADE};
```

Sentencias de manipulación (DDL)

Una vez creada la base de datos con sus tablas, debemos poder insertar, modificar y borrar los valores de las filas de las tablas. Para poder hacer esto, el SQL92 nos ofrece las siguientes sentencias: INSERT para insertar, UPDATE para modificar y DELETE para borrar. Una vez hemos insertado valores en nuestras tablas, tenemos que poder consultarlos. La sentencia para hacer consultas a una base de datos con el SQL92 es SELECT FROM. Veamos a continuación estas sentencias.

Inserción de filas en una tabla

Antes de poder consultar los datos de una base de datos, es preciso introducirlos con la sentencia INSER TINTO VALUES, que tiene el formato:

```
INSERT INTO nombre_tabla [(columnas)]  
(VALUES ({v1|DEFAULT|NULL}, ..., {vn/DEFAULT/NULL})|<consulta>);
```

Inserción de múltiples filas

Para insertar más de una fila con una sola sentencia, tenemos que obtener los valores como resultado de una consulta realizada en una o más tablas.

Los valores v1, v2, ..., vn se deben corresponder exactamente con las columnas que hemos dicho que tendríamos con el CREATE TABLE y deben estar en el mismo orden, a menos que las volvamos a poner a continuación del nombre de la tabla. En este último caso, los valores se deben disponer de forma coherente con el nuevo orden que hemos impuesto. Podría darse el caso de que quisieramos que algunos valores para insertar fuesen valores por omisión, definidos previamente con la opción DEFAULT. Entonces pondríamos la palabra reservada DEFAULT. Si se trata de introducir valores nulos, también podemos utilizar la palabra reservada NULL.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Borrado de filas de una tabla

Para borrar valores de algunas filas de una tabla podemos utilizar la sentencia DELETE FROM WHERE. Su formato es el siguiente:

```
DELETE FROM nombre_tabla
[WHERE condiciones];
```

En cambio, si lo que quisiéramos conseguir es borrar todas las filas de una tabla, entonces sólo tendríamos que poner la sentencia DELETE FROM, sin WHERE.

Modificación de filas de una tabla

Si quisiéramos modificar los valores de algunas filas de una tabla, tendríamos que utilizar la sentencia UPDATE SET WHERE. A continuación presentamos su formato:

```
UPDATE nombre_tabla
SET columna = {expresión|DEFAULT|NULL}
[, columna = {expr|DEFAULT|NULL} ...]
WHERE condiciones;
```

Consultas a una base de datos relacional

Para hacer consultas sobre una tabla con el SQL es preciso utilizar la sentencia SELECT FROM, que tiene el siguiente formato:

```
SELECT nombre_columna_a_seleccionar [[AS] col_renombrada]
[, nombre_columna_a_seleccionar [[AS] col_renombrada] ...]
FROM tabla_a_consultar [[AS] tabla_renombrada];
```

La opción AS nos permite renombrar las columnas que queremos seleccionar o las tablas que queremos consultar que en este caso, es sólo una. Dicho de otro modo, nos permite la definición de alias. Fijémonos en que la palabra clave AS es opcional, y es bastante habitual poner sólo un espacio en blanco en lugar de toda la palabra.

La cláusula WHERE nos permite obtener las filas que cumplen la condición especificada en la consulta.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Para definir las condiciones en la cláusula WHERE, podemos utilizar alguno de los operadores de los que dispone el SQL, que son los siguientes:

Operadores de comparación		Operadores lógicos	
=	Igual	NOT	Para la negación de condiciones
<	Menor	AND	Para la conjunción de condiciones
>	Mayor	OR	Para la disyunción de condiciones
<=	Menor o igual		
>=	Mayor o igual		
<>	Diferente		

Si queremos que en una consulta nos aparezcan las filas resultantes sin repeticiones, es preciso poner la palabra clave DISTINCT inmediatamente después de SELECT. También podríamos explicitar que lo queremos todo, incluso con repeticiones, poniendo ALL (opción por defecto) en lugar de DISTINCT. El formato de DISTINCT es:

```
SELECT DISTINCT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
[WHERE condiciones];
```

Funciones de agregación

El SQL nos ofrece las siguientes funciones de agregación para efectuar varias operaciones sobre los datos de una base de datos:

Funciones de agregación	
Función	Descripción
COUNT	Nos da el número total de filas seleccionadas
SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor medio de una columna

En general, las funciones de agregación se aplican a una columna, excepto la función de agregación COUNT, que normalmente se aplica a todas las columnas de la tabla o tablas seleccionadas. Por lo tanto, COUNT (*) contará todas las filas de la tabla o las tablas que cumplan las condiciones. Si se utilizase COUNT(distinct columna), sólo contaría los valores que no fuesen nulos ni repetidos, y si se utilizase COUNT (columna), sólo contaría los valores que no fuesen nulos.

Ejemplo de utilización de la función COUNT (*)

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Veamos un ejemplo de uso de la función COUNT, que aparece en la cláusula SELECT, para hacer la consulta “¿Cuántos departamentos están ubicados en la ciudad de Lleida?”:

```
SELECT COUNT(*) AS numero_dep
FROM departamentos
WHERE ciudad_dep = 'Lleida';
```

Subconsultas

Una subconsulta es una consulta incluida dentro de una cláusula WHERE o HAVING de otra consulta. En ocasiones, para expresar ciertas condiciones no hay más remedio que obtener el valor que buscamos como resultado de una consulta.

```
SELECT codigo_proyec, nombre_proyec
FROM proyectos
WHERE precio = (SELECT MAX(precio)
                 FROM proyectos);
```

Otros predicados

1) Predicado BETWEEN

Para expresar una condición que quiere encontrar un valor entre unos límites concretos, podemos utilizar BETWEEN:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna BETWEEN limite1 AND limite2;
```

2) Predicado IN

Para comprobar si un valor coincide con los elementos de una lista utilizaremos IN, y para ver si no coincide, NOT IN:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] IN (valor1, ..., valorN);
```

3) Predicado LIKE

Para comprobar si una columna de tipo carácter cumple alguna propiedad determinada, podemos usar LIKE:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna LIKE característica;
```

Ordenación de los datos obtenidos en respuestas a consultas

Si se desea que, al hacer una consulta, los datos aparezcan en un orden determinado, es preciso utilizar la cláusula ORDER BY en la sentencia SELECT, que presenta el siguiente formato:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
ORDER BY columna_según_la_cual_se_quiere_ordenar [DESC]
[, col_ordenación [DESC] ...];
```

Consultas con agrupación de filas de una tabla

Las cláusulas siguientes, añadidas a la instrucción SELECT FROM, permiten organizar las filas por grupos:

- a) La cláusula GROUP BY nos sirve para agrupar filas según las columnas que indique esta cláusula.
- b) La cláusula HAVING especifica condiciones de búsqueda para grupos de filas; lleva a cabo la misma función que antes cumplía la cláusula WHERE para las filas de toda la tabla, pero ahora las condiciones se aplican a los grupos obtenidos.

Presenta el siguiente formato:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
GROUP BY columnas_según_las_cuales_se_quiere_agrupar
[HAVING condiciones_por_grupos]
[ORDER BY columna_ordenación [DESC] [, columna [DESC] ...]];
```

Consultas a más de una tabla

Muchas veces queremos consultar datos de más de una tabla haciendo combinaciones de columnas de tablas diferentes. En el SQL es posible listar más de una tabla que se quiere consultar especificándolo en la cláusula FROM.

Combinación

La combinación consigue crear una sola tabla a partir de las tablas especificadas en la cláusula FROM, haciendo coincidir los valores de las columnas relacionadas de estas tablas.

```
SELECT proyectos.codigo_proyecto, proyectos.precio, clientes.nif
FROM clientes, proyectos
WHERE clientes.codigo_cli = proyectos.codigo_cliente AND clientes.
codigo_cli = 20;
```

La unión

La cláusula UNION permite unir consultas de dos o más sentencias SELECT FROM. Su formato es:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
UNION [ALL]
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Nivel: Quinto año

Prácticas generales de SQL Server

Actividad

1. Seguro que siempre habéis querido saber dónde teníais aquella película de vídeo que nunca encontrabais. Por ello os proponemos crear una base de datos para organizar las cintas de vídeo y localizarlas rápidamente cuando os apetezca utilizarlas. Tendréis que crear la base de datos y las tablas; también deberéis decidir las claves primarias e insertar filas.

Para almacenar las cintas de vídeo, tendremos que crear las siguientes tablas:

- a) Las cintas: querremos saber su código, la estantería donde se encuentran, el estante y la fila, suponiendo que en un estante haya más de una fila. Tendremos que poner nosotros el código de las cintas, con un rotulador, en el lomo de cada una.
- b) Las películas: querremos saber su código, título, director principal (en el caso de que haya más de uno) y el tema. El código de las películas también lo tendremos que escribir nosotros con un rotulador para distinguir películas que tienen el mismo nombre.
- c) Los actores: sólo querremos saber de ellos un código, el nombre y el apellido y, si somos aficionados al cine, otros datos que nos pueda interesar almacenar. El código de los actores, que inventaremos nosotros, nos permitirá distinguir entre actores que se llaman igual.
- d) Películas que hay en cada cinta: en esta tabla pondremos el código de la cinta y el código de la película. En una cinta puede haber más de una película, y podemos tener una película repetida en más de una cinta; se debe tener en cuenta este hecho en el momento de elegir la clave primaria.
- e) Actores que aparecen en las películas: en esta tabla indicaremos el código de la película y el código del actor. En una película puede participar más de un actor y un actor puede aparecer en más de una película; hay que tener presente este hecho cuando se elige la clave primaria.

Esperamos que, además de practicar sentencias de definición, manipulación y control del SQL, esta actividad os resulte muy útil.

Con la actividad anterior hemos practicado sentencias de definición y control del SQL. Mediante las sentencias de manipulación hemos insertado filas y, si nos hubiésemos equivocado, también habríamos borrado y modificado alguna fila. Con los ejercicios de autoevaluación practicaremos la parte de sentencias de manipulación que no hemos tratado todavía: las consultas.

Los ejercicios que proponemos se harán sobre la base de datos relacional BDUOC que ha ido apareciendo a lo largo de esta unidad.

1. Obtendréis los códigos y los nombres y apellidos de los empleados, ordenados alfabéticamente de forma descendente por apellido y, en caso de repeticiones, por nombre.
2. Consultad el código y el nombre de los proyectos de los clientes que son de Barcelona.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

3. Obtened los nombres y las ciudades de los departamentos que trabajan en los proyectos número 3 y número 4.
4. De todos los empleados que perciben un sueldo de entre 50.000 y 80.000 euros, buscad los códigos de empleado y los nombres de los proyectos que tienen asignados.
5. Buscad el nombre, la ciudad y el teléfono de los departamentos donde trabajan los empleados del proyecto GESCOM.
6. Obtened los códigos y los nombres y apellidos de los empleados que trabajan en los proyectos de precio más alto.
7. Averiguad cuál es el sueldo más alto de cada departamento. Concretamente, es necesario dar el nombre y la ciudad del departamento y el sueldo más elevado.
8. Obtened los códigos y los nombres de los clientes que tienen más de un proyecto contratado.
9. Averiguad los códigos y los nombres de los proyectos cuyos empleados asignados tienen un sueldo superior a 30.000 euros.
10. Buscad los nombres y las ciudades de los departamentos que no tienen ningún empleado asignado.

Tercer proyecto, análisis y desarrollo de una base de datos para diversas empresas, trabajo en parejas

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

TSQL

SQL es un lenguaje de consulta para los sistemas de bases de datos relacionales, pero que no posee la potencia de los lenguajes de programación. No permite el uso de variables, estructuras de control de flujo, bucles ... y demás elementos característicos de la programación. No es de extrañar, SQL es un lenguaje de consulta, no un lenguaje de programación.

Sin embargo, SQL es la herramienta ideal para trabajar con bases de datos. Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. Transact SQL es el lenguaje de programación que proporciona Microsoft SQL Server para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación.

Con Transact SQL vamos a poder programar las unidades de programa de la base de datos SQL Server, están son:

- Procedimientos almacenados
- Funciones
- Triggers
- Scripts

Variables en Transact SQL

Declarar variables es Transact SQL

Una variable es un valor identificado por un nombre (identificador) sobre el que podemos realizar modificaciones.

En Transact SQL los identificadores de variables deben comenzar por el carácter @, es decir, el nombre de una variable debe comenzar por @. Para declarar variables en Transact SQL debemos utilizar la palabra clave declare, seguido del identificador y tipo de datos de la variable.

```
-- Esto es un comentario de línea simple

/*
Este es un comentario con varias líneas.
Conjunto de Líneas.
*/
declare @nombre varchar(50)-- declare declara una variable
                      -- @nombre es el identificador de la
                      -- variable de tipo varchar
set @nombre = 'www.devjoker.com' -- El signo = es un operador
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
print @Nombre -- Imprime por pantalla el valor de @nombre.  
-- No diferencia mayúsculas ni minúsculas
```

Asignar variables en Transact SQL

En Transact SQL podemos asignar valores a una variable de varias formas:

- A través de la instrucción set.
- Utilizando una sentencia SELECT.
- Realizando un FETCH de un cursor.

El siguiente ejemplo muestra como asignar una variable utilizando la instrucción SET.

```
DECLARE @nombre VARCHAR(100)  
-- La consulta debe devolver un único registro  
  
SET @nombre = (SELECT nombre  
FROM CLIENTES  
WHERE ID = 1)  
  
PRINT @nombre
```

- El siguiente ejemplo muestra como asignar variables utilizando una sentencia SELECT.

```
DECLARE @nombre VARCHAR(100),  
@apellido1 VARCHAR(100),  
@apellido2 VARCHAR(100)  
  
  
SELECT @nombre=nombre ,  
@apellido1=Apellido1,  
@apellido2=Apellido2
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
FROM CLIENTES
```

```
WHERE ID = 1
```

```
PRINT @nombre
```

```
PRINT @apellido1
```

```
PRINT @apellido2
```

- Un punto a tener en cuenta cuando asignamos variables de este modo, es que si la consulta SELECT devuelve más de un registro, las variables quedarán asignadas con los valores de la última fila devuelta.
- Por último veamos como asignar variables a través de un cursor.

```
DECLARE @nombre VARCHAR(100),  
        @apellido1 VARCHAR(100),  
        @apellido2 VARCHAR(100)
```

```
DECLARE CDATOS CURSOR
```

```
FOR
```

```
SELECT nombre , Apellido1, Apellido2
```

```
FROM CLIENTES
```

```
OPEN CDATOS
```

```
FETCH CDATOS INTO @nombre, @apellido1, @apellido2
```

```
WHILE (@@FETCH_STATUS = 0)
```

```
BEGIN
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```

PRINT @nombre
PRINT @apellido1
PRINT @apellido2
FETCH CDATOS INTO @nombre, @apellido1, @apellido2

```

END

CLOSE CDATOS

DEALLOCATE CDATOS

Veremos los cursores con más detalle más adelante

La siguiente tabla ilustra los operadores de Transact SQL .

Tipo de operador	Operadores
Operador de asignación	=
Operadores aritméticos	+ (suma) - (resta) * (multiplicación) / (división) ** (exponente) % (modulo)
Operadores relacionales o de comparación	= (igual a) <> (distinto de) != (distinto de) < (menor que) > (mayor que) >= (mayor o igual a) <= (menor o igual a) !> (no mayor a) !< (no menor a)
Operadores lógicos	AND (y lógico) NOT (negación) OR (o lógico) & (AND a nivel de bit) (OR a nivel de bit) ^ (OR exclusivo a nivel de bit)
Operador de concatenación	+

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Otros

ALL (Devuelve TRUE si el conjunto completo de comparaciones es TRUE)
ANY(Devuelve TRUE si cualquier elemento del conjunto de comparaciones es TRUE)
BETWEEN (Devuelve TRUE si el operando está dentro del intervalo)
EXISTS (TRUE si una subconsulta contiene filas)
IN (TRUE si el operando está en la lista)
LIKE (TRUE si el operando coincide con un patrón)
NOT (Invierte el valor de cualquier operador booleano)
SOME(Devuelve TRUE si alguna de las comparaciones de un conjunto es TRUE)

Estructuras de control en Transact SQL

Estructura condicional IF

La estructura condicional IF permite evaluar una expresión booleana (resultado SI - NO), y ejecutar las operaciones contenidas en el bloque formado por BEGIN END.

```
IF (<expresion>)
BEGIN
    ...
END
ELSE IF (<expresion>)
BEGIN
    ...
END
ELSE
BEGIN
    ...
END
```

Ejemplo de la estructura condicional IF.

```
DECLARE @Web varchar(100),
        @diminutivo varchar(3)

SET @diminutivo = 'DJK'

IF @diminutivo = 'DJK'
BEGIN
    PRINT 'www.devjoker.com'
END
ELSE
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
BEGIN
    PRINT 'Otra Web (peor!)'
END
```

La estructura IF admite el uso de subconsultas:

```
DECLARE @coPais int,
        @descripcion varchar(255)
set @coPais = 5
set @descripcion = 'España'
IF EXISTS(SELECT * FROM PAISES
            WHERE CO_PAIS = @coPais)
BEGIN
    UPDATE PAISES
        SET DESCRIPCION = @descripcion
        WHERE CO_PAIS = @coPais
END

ELSE
BEGIN
    INSERT INTO PAISES
        (CO_PAIS, DESCRIPCION) VALUES
        (@coPais, @descripcion)
END
```

Estructura condicional CASE

La estructura condicional CASE permite evaluar una expresión y devolver un valor u otro.

La sintaxis general de case es:

```
CASE <expresion>
    WHEN <valor_expresion> THEN <valor_devuelto>
    WHEN <valor_expresion> THEN <valor_devuelto>
    ELSE <valor_devuelto> -- Valor por defecto
END
```

Ejemplo de **CASE**.

```
DECLARE @Web varchar(100),
        @diminutivo varchar(3)
SET @diminutivo = 'DJK'
SET @Web = (CASE @diminutivo
                WHEN 'DJK' THEN 'www.devjoker.com'
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```

WHEN 'ALM' THEN 'www.aleamedia.com'
ELSE 'www.devjoker.com'
END)
PRINT @Web

```

Otra sintaxis de CASE nos permite evaluar diferentes expresiones:

```

CASE
WHEN <expresion> = <valor_expresion> THEN <valor_devuelto>
WHEN <expresion> = <valor_expresion> THEN <valor_devuelto>
ELSE <valor_devuelto> -- Valor por defecto
END

```

El mismo ejemplo aplicando esta sintaxis:

```

DECLARE @Web varchar(100),
@diminutivo varchar(3)
SET @diminutivo = 'DJK'

SET @Web = (CASE
WHEN @diminutivo = 'DJK' THEN 'www.devjoker.com'
WHEN @diminutivo = 'ALM' THEN 'www.aleamedia.com'
ELSE 'www.devjoker.com'
END)
PRINT @Web

```

Otro aspecto muy interesante de CASE es que permite el uso de subconsultas.

```

DECLARE @Web varchar(100),
@diminutivo varchar(3)
SET @diminutivo = 'DJK'

SET @Web = (CASE
WHEN @diminutivo = 'DJK' THEN (SELECT web
FROM WEBS
WHERE id=1)
WHEN @diminutivo = 'ALM' THEN (SELECT web
FROM WEBS
WHERE id=2)
ELSE 'www.devjoker.com'
END)
PRINT @Web

```

Bucle WHILE

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

El bucle WHILE se repite mientras expresión se evalúe como verdadero.

Es el único tipo de bucle del que dispone Transact SQL.

```
WHILE <expresión>
BEGIN
...
END
```

Un ejemplo del bucle WHILE.

```
DECLARE @contador int
SET @contador = 0
WHILE (@contador < 100)
BEGIN
    SET @contador = @contador + 1

    PRINT 'Iteración del bucle ' + cast(@contador AS varchar)
END
```

Podemos pasar a la siguiente iteración del bucle utilizando CONTINUE.

```
DECLARE @contador int
SET @contador = 0
WHILE (@contador < 100)
BEGIN
    SET @contador = @contador + 1
    IF (@contador % 2 = 0)
        CONTINUE
    PRINT 'Iteración del bucle ' + cast(@contador AS varchar)
END
```

El bucle se dejará de repetir con la instrucción BREAK.

```
DECLARE @contador int
SET @contador = 0
WHILE (1 = 1)
BEGIN
    SET @contador = @contador + 1
    IF (@contador % 50 = 0)
        BREAK
    PRINT 'Iteración del bucle ' + cast(@contador AS varchar)
END
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

También podemos utilizar el bucle WHILE conjuntamente con subconsultas.

```

DECLARE @coRecibo int
WHILE EXISTS (SELECT *
               FROM RECIBOS
               WHERE PENDIENTE = 'S') -- Ojo, la subconsulta se
ejecuta
                                         -- una vez por cada iteracion
                                         -- del bucle!
BEGIN
    SET @coRecibo = (SELECT TOP 1 CO_RECIBO
                      FROM RECIBOS WHERE PENDIENTE = 'S')
    UPDATE RECIBOS
    SET PENDIENTE = 'N'
    WHERE CO_RECIBO = @coRecibo
END

```

Estructura GOTO

La sentencia goto nos permite desviar el flujo de ejecución hacia una etiqueta. Fue muy utilizada en versiones anteriores de SQL Server conjuntamente con la variable de sistema @@ERROR para el control de errores.

Actualmente, se desaconseja el uso GOTO, recomendándose el uso de TRY - CATCH para la gestión de errores.

```

DECLARE @divisor int,
        @dividendo int,
        @resultado int
SET @dividendo = 100
SET @divisor = 0
SET @resultado = @dividendo/@divisor

IF @@ERROR > 0
    GOTO error

PRINT 'No hay error'
RETURN
error:
PRINT 'Se ha producido una division por cero'

```

Control de errores en Transact SQL

Uso de TRY CATCH

A partir de la versión 2005, SQL Server proporciona el control de errores a través de las instrucciones TRY y CATCH.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Estas nuevas instrucciones suponen un gran paso adelante en el control de errores en SQL Server, un tanto precario en las versiones anteriores.

La sintaxis de TRY CATCH es la siguiente:

```
BEGIN TRY
  ...
END TRY
BEGIN CATCH
  ...
END CATCH
```

El siguiente ejemplo ilustra el uso de TRY - CATCH.

```
BEGIN TRY

  DECLARE @divisor int ,
          @dividendo int,
          @resultado int

  SET @dividendo = 100
  SET @divisor = 0

  -- Esta linea provoca un error de division por 0
  SET @resultado = @dividendo/@divisor
  PRINT 'No hay error'
END TRY
BEGIN CATCH
  PRINT 'Se ha producido un error'
END CATCH
```

Procedimientos almacenados en Transact SQL

Un procedimiento es un programa dentro de la base de datos que ejecuta una acción o conjunto de acciones específicas.

Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

CEDES DON BOSCO

Nivel: Quinto año

En Transact SQL los procedimientos almacenados pueden devolver valores (numérico entero) o conjuntos de resultados.

Para crear un procedimiento almacenado debemos emplear la sentencia CREATE PROCEDURE.

```
CREATE PROCEDURE <nombre_procedure> [<param1 <tipo>, ...]
```

```
AS
```

```
-- Sentencias del procedure
```

Para modificar un procedimiento almacenado debemos emplear la sentencia ALTER PROCEDURE.

```
ALTER PROCEDURE <nombre_procedure> [<param1 <tipo>, ...]
```

```
AS
```

```
-- Sentencias del procedure
```

El siguiente ejemplo muestra un procedimiento almacenado, denominado spu_addCliente que inserta un registro en la tabla "CLIENTES".

```
CREATE PROCEDURE spu_addCliente @nombre varchar(100),
```

```
                          @apellido1 varchar(100),
```

```
                          @apellido2 varchar(100),
```

```
                          @nifCif varchar(20),
```

```
                          @fxNacimiento datetime
```

```
AS
```

```
INSERT INTO CLIENTES
```

```
(nombre, apellido1, apellido2, nifcif, fxnacimiento) VALUES
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
(@nombre, @apellido1, @apellido2, @nifCif, @fxNaciento)
```

Para la ejecutar un procedimiento almacenado debemos utilizar la sentencia EXEC. Cuando la ejecución del procedimiento almacenado es la primera instrucción del lote, podemos omitir el uso de EXEC.

El siguiente ejemplo muestra la ejecución del procedimiento almacenado anterior.

```
DECLARE @fecha_nacimiento datetime  
  
set @fecha_nacimiento = convert(datetime, '13/05/1975', 103)  
  
EXEC spu_addCliente 'Pedro', 'Herrarte', 'Sanchez',  
'00000002323', @fecha_nacimiento
```

Siempre es deseable que las instrucciones del procedure esten dentro de un bloque TRY CATCH y controlados por una transacción.

```
ALTER PROCEDURE spu_addCliente @nombre varchar(100),  
                                @apellido1 varchar(100),  
                                @apellido2 varchar(100),  
                                @nifCif varchar(20),  
                                @fxNaciento datetime  
  
AS  
  
BEGIN TRY  
  
    BEGIN TRAN  
  
    INSERT INTO CLIENTES  
  
        (nombre, apellido1, apellido2, nifcif, fxnacimiento) VALUES  
  
        (@nombre, @apellido1, @apellido2, @nifCif, @fxNaciento)
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

COMMIT

END TRY

BEGIN CATCH

ROLLBACK

PRINT ERROR_MESSAGE()

END CATCH

Si queremos que los parámetros de un procedimiento almacenado sean de entrada-salida debemos especificarlo a través de la palabra clave **OUTPUT**, tanto en la definición del procedure como en la ejecución.

El siguiente ejemplo muestra la definición de un procedure con parámetros de salida.

```
CREATE PROCEDURE spu_ObtenerSaldoCuenta @numCuenta varchar(20),  
                                              @saldo decimal(10,2) output  
AS  
BEGIN  
SELECT @saldo = SALDO  
FROM CUENTAS  
WHERE NUMCUENTA = @numCuenta  
END
```

Y para ejecutar este procedure:

```
DECLARE @saldo decimal(10,2)  
EXEC spu_ObtenerSaldoCuenta '200700000001', @saldo output  
PRINT @saldo
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Un procedimiento almacenado puede devolver valores numéricos enteros a través de la instrucción RETURN. Normalmente debemos utilizar los valores de retorno para determinar si la ejecución del procedimiento ha sido correcta o no. Si queremos obtener valores se recomienda utilizar parámetros de salida o funciones escalares (se verán más adelante en este tutorial).

El siguiente ejemplo muestra un procedimiento almacenado que devuelve valores.

```
CREATE PROCEDURE spu_EstaEnNumerosRojos @numCuenta varchar(20)  
AS  
BEGIN  
IF (SELECT SALDO FROM CUENTAS  
      WHERE NUMCUENTA = @numCuenta) < 0  
    BEGIN  
      RETURN 1  
    END  
  ELSE  
    RETURN 0  
END
```

El siguiente ejemplo muestra como ejecutar el procedure y obtener el valor devuelto.

```
DECLARE @rv int  
EXEC @rv = spu_EstaEnNumerosRojos '200700000001'  
PRINT @rv
```

Otra característica muy interesante de los procedimientos almacenados en Transact SQL es que pueden devolver uno o varios conjuntos de resultados.

El siguiente ejemplo muestra un procedimiento almacenado que devuelve un conjunto de resultados.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
CREATE PROCEDURE spu_MovimientosCuenta @numCuenta varchar(20)  
AS  
BEGIN  
SELECT @numCuenta,  
        SALDO_ANTERIOR,  
        SALDO_POSTERIOR,  
        IMPORTE,  
        FXMOVIMIENTO  
FROM MOVIMIENTOS  
INNER JOIN CUENTAS ON MOVIMIENTOS.IDCUENTA = CUENTAS.IDCUENTA  
WHERE NUMCUENTA = @numCuenta  
ORDER BY FXMOVIMIENTO DESC  
END
```

La ejecución del procedimiento se realiza normalmente.

```
EXEC spu_MovimientosCuenta '200700000001'
```

Nivel: Quinto año

Triggers en Transact SQL

Un trigger(o desencadenador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.

SQL Server proporciona los siguientes tipos de triggers:

- *Trigger DML*, se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.
- *Trigger DDL*, se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Trigger DML.

Los trigger DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.

La sintaxis general de un trigger es la siguiente.

```
CREATE TRIGGER <Trigger_Name, sysname, Trigger_Name>
ON <Table_Name, sysname, Table_Name>
AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE>
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.

SET NOCOUNT ON;

-- Insert statements for trigger here

END
```

Antes de ver un ejemplo es necesario conocer las tablas inserted y deleted.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Las instrucciones de triggers DML utilizan dos tablas especiales denominadas **inserted** y **deleted**. SQL Server 2005 crea y administra automáticamente ambas tablas. La estructura de las tablas **inserted** y **deleted** es la misma que tiene la tabla que ha desencadenado la ejecución del trigger.

La primera tabla (**inserted**) solo está disponible en las operaciones **INSERT** y **UPDATE** y en ella están los valores resultantes después de la inserción o actualización. Es decir, los datos insertados. **Inserted** estará vacía en una operación **DELETE**.

En la segunda (**deleted**), disponible en las operaciones **UPDATE** y **DELETE**, están los valores anteriores a la ejecución de la actualización o borrado. Es decir, los datos que serán borrados. **Deleted** estará vacía en una operación **INSERT**.

¿No existe una tabla **UPDATED**? No, hacer una actualización es lo mismo que borrar (**deleted**) e insertar los nuevos (**inserted**). La sentencia **UPDATE** es la única en la que **inserted** y **deleted** tienen datos simultáneamente.

No puede modificar directamente los datos de estas tablas.

El siguiente ejemplo, graba un histórico de saldos cada vez que se modifica un saldo de la tabla cuentas.

CREATE TRIGGER TR CUENTAS

ON CUENTAS

AFTER UPDATE

AS

BEGIN

-- SET NOCOUNT ON impide que se generen mensajes de texto

-- con cada instrucción

SET NOCOUNT ON;

INSERT INTO HCO_SALDOS

(IDCUENTA, SALDO, FXSALDO)

SELECT IDCUENTA, SALDO, getdate()

FROM INSERTED

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

END

La siguiente instrucción provocará que el trigger se ejecute:

UPDATE CUENTAS

SET SALDO = SALDO + 10

WHERE IDCUENTA = 1

Una consideración a tener en cuenta es que el trigger se ejecutará aunque la instrucción DML (UPDATE, INSERT o DELETE) no haya afectado a ninguna fila. En este caso inserted y deleted devolverán un conjunto de datos vacío.

Podemos especificar a qué columnas de la tabla debe afectar el trigger.

ALTER TRIGGER TR_CUENTAS

ON CUENTAS

AFTER UPDATE

AS

BEGIN

-- SET NOCOUNT ON impide que se generen mensajes de texto

-- con cada instrucción

SET NOCOUNT ON;

IF UPDATE(SALDO) -- Solo si se actualiza SALDO

BEGIN

INSERT INTO HCO_SALDOS

(IDCUENTA, SALDO, FXSALDO)

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
SELECT IDCUENTA, SALDO, getdate()  
FROM INSERTED  
END  
END
```

Los trigger están dentro de la transacción original (Insert, Delete o Update) por lo cual si dentro de nuestro trigger hacemos un RollBack Tran, no solo estaremos echando atrás nuestro trigger sino también toda la transacción; en otras palabras si en un trigger ponemos un RollBack Tran, la transacción de Insert, Delete o Update volverá toda hacia atrás.

```
ALTER TRIGGER TR CUENTAS  
ON CUENTAS  
AFTER UPDATE  
AS  
BEGIN  
-- SET NOCOUNT ON impide que se generen mensajes de texto  
-- con cada instrucción  
SET NOCOUNT ON;  
INSERT INTO HCO_SALDOS  
(IDCUENTA, SALDO, FXSALDO)  
SELECT IDCUENTA, SALDO, getdate()  
FROM INSERTED  
ROLLBACK  
END
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

En este caso obtendremos el siguiente mensaje de error:

La transacción terminó en el desencadenador. Se anuló el lote.

Podemos activar y desactivar Triggers a través de las siguientes instrucciones.

-- Desactiva el trigger TR CUENTAS

DISABLE TRIGGER TR CUENTAS ON CUENTAS

GO

-- activa el trigger TR CUENTAS

ENABLE TRIGGER TR CUENTAS ON CUENTAS

GO

-- Desactiva todos los trigger de la tabla CUENTAS

ALTER TABLE CUENTAS DISABLE TRIGGER ALL

GO

-- Activa todos los trigger de la tabla CUENTAS

ALTER TABLE CUENTAS ENABLE TRIGGER ALL

Cursos en Transact SQL

Un cursor es una variable que nos permite recorrer con un conjunto de resultados obtenido a través de una sentencia SELECT fila a fila.

Cuando trabajemos con cursos debemos seguir los siguientes pasos.

- Declarar el cursor, utilizando DECLARE
- Abrir el cursor, utilizando OPEN
- Leer los datos del cursor, utilizando FETCH ... INTO
- Cerrar el cursor, utilizando CLOSE
- Liberar el cursor, utilizando DEALLOCATE

La sintaxis general para trabajar con un cursor es la siguiente.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

-- Declaración del cursor

DECLARE <nombre_cursor> **CURSOR**

FOR

<sentencia_sql>

-- apertura del cursor

OPEN <nombre_cursor>

-- Lectura de la primera fila del cursor

FETCH <nombre_cursor> **INTO** <lista_variables>

WHILE (@@FETCH_STATUS = 0)

BEGIN

-- Lectura de la siguiente fila de un cursor

FETCH <nombre_cursor> **INTO** <lista_variables>

...

END -- Fin del bucle WHILE

-- Cierra el cursor

CLOSE <nombre_cursor>

-- Libera los recursos del cursor

DEALLOCATE <nombre_cursor>

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

El siguiente ejemplo muestra el uso de un cursor.

-- Declaracion de variables para el cursor

DECLARE @Id **int**,

 @Nombre **varchar**(255),

 @Apellido1 **varchar**(255),

 @Apellido2 **varchar**(255),

 @NifCif **varchar**(20),

 @FxNacimiento **datetime**

-- Declaración del cursor

DECLARE cClientes **CURSOR FOR**

SELECT Id, Nombre, Apellido1,
 Apellido2, NifCif, FxNacimiento

FROM CLIENTES

-- Apertura del cursor

OPEN cClientes

-- Lectura de la primera fila del cursor

FETCH cClientes **INTO** @id, @Nombre, @Apellido1,
 @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0)

BEGIN

PRINT @Nombre + '' + @Apellido1 + '' + @Apellido2

-- Lectura de la siguiente fila del cursor

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
FETCH cClientes INTO @id, @Nombre, @Apellido1,
@Apellido2, @NifCif, @FxNacimiento
```

END

-- Cierre del cursor

CLOSE cClientes

-- Liberar los recursos

DEALLOCATE cClientes

Cuando trabajamos con cursores, la función @@FETCH_STATUS nos indica el estado de la última instrucción FETCH emitida, los valores posibles son:

Valor devuelto	Descripción
0	La instrucción FETCH se ejecutó correctamente.
-1	La instrucción FETCH no se ejecutó correctamente o la fila estaba más allá del conjunto de resultados.
-2	Falta la fila recuperada.

En la apertura del cursor, podemos especificar los siguientes parámetros:

```
DECLARE <nombre_cursor> CURSOR

[ LOCAL | GLOBAL ]

[ FORWARD_ONLY | SCROLL ]

[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]

[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]

[ TYPE_WARNING ]

FOR <sentencia_sql>
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

El primer conjunto de parámetros que podemos especificar es [LOCAL | GLOBAL]. A continuación mostramos el significado de cada una de estas opciones.

LOCAL

Especifica que el ámbito del cursor es local para el proceso por lotes, procedimiento almacenado o desencadenador en que se creó el cursor.

```
DECLARE cClientes CURSOR LOCAL FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

GLOBAL

Especifica que el ámbito del cursor es global para la conexión. Puede hacerse referencia al nombre del cursor en cualquier procedimiento almacenado o proceso por lotes que se ejecute en la conexión.

```
DECLARE cClientes CURSOR GLOBAL FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

Si no se especifica GLOBAL ni LOCAL, el valor predeterminado se controla mediante la configuración de la opción de base de datos default to local cursor.

El siguiente conjunto de parámetros que podemos especificar es [FORWARD_ONLY | SCROLL]. A continuación mostramos el significado de cada una de estas opciones.

FORWARD_ONLY

Especifica que el cursor sólo se puede desplazar de la primera a la última fila. FETCH NEXT es la única opción de recuperación admitida.

```
DECLARE cClientes CURSOR FORWARD_ONLY FOR
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

SCROLL

Especifica que están disponibles todas las opciones de recuperación (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Si no se especifica SCROLL en una instrucción DECLARE CURSOR la única opción de recuperación que se admite es NEXT. No es posible especificar SCROLL si se incluye también

FAST_FORWARD.

Si se incluye la opción SCROLL, la forma en la realizamos la lectura del cursor varia, debiendo utilizar la siguiente sintaxis: **FETCH [NEXT | PRIOR | FIRST | LAST | RELATIVE | ABSOLUTE] FROM < INTO**

-- Declaracion de variables para el cursor

```
DECLARE @Id int,  
@Nombre varchar(255),  
@Apellido1 varchar(255),  
@Apellido2 varchar(255),  
@NifCif varchar(20),  
@FxNacimiento datetime
```

-- Declaración del cursor

```
DECLARE cClientes CURSOR SCROLL FOR  
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

-- Apertura del cursor

OPEN cClientes

-- Lectura de la primera fila del cursor

FETCH NEXT **FROM** cClientes

INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0)

BEGIN

PRINT @Nombre + '' + @Apellido1 + '' + @Apellido2

-- Lectura de la siguiente fila del cursor

FETCH NEXT **FROM** cClientes

INTO @id,@Nombre,@Apellido1,@Apellido2,@NifCif,@FxNacimiento

END

-- Lectura de la fila anterior

FETCH PRIOR **FROM** cClientes

INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento

PRINT @Nombre + '' + @Apellido1 + '' + @Apellido2

-- Cierre del cursor

CLOSE cClientes

-- Liberar los recursos

DEALLOCATE cClientes

El siguiente conjunto de parámetros que podemos especificar es [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]. A continuación mostramos el significado de cada una de estas opciones.

STATIC

Define un cursor que hace una copia temporal de los datos que va a utilizar. Todas las solicitudes

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

que se realizan al cursor se responden desde esta tabla temporal de tempdb; por tanto, las modificaciones realizadas en las tablas base no se reflejan en los datos devueltos por las operaciones de recuperación realizadas en el cursor y además este cursor no admite modificaciones.

```
DECLARE cClientes CURSOR STATIC FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

KEYSET

Especifica que la pertenencia y el orden de las filas del cursor se fijan cuando se abre el cursor. El conjunto de claves que identifica las filas de forma única está integrado en la tabla denominada keyset de tempdb.

```
DECLARE cClientes CURSOR KEYSET FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

DYNAMIC

Define un cursor que, al desplazarse por él, refleja en su conjunto de resultados todos los cambios realizados en los datos de las filas. Los valores de los datos, el orden y la pertenencia de las filas pueden cambiar en cada operación de recuperación. La opción de recuperación ABSOLUTE no se puede utilizar en los cursores dinámicos.

```
DECLARE cClientes CURSOR DYNAMIC FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

FAST_FORWARD

Especifica un cursor FORWARD_ONLY, READ_ONLY con las optimizaciones de rendimiento habilitadas. No se puede especificar FAST_FORWARD si se especifica también SCROLL o FOR_UPDATE.

```
DECLARE cClientes CURSOR FAST_FORWARD FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

En SQL Server 2000, las opciones de cursor FAST_FORWARD y FORWARD_ONLY se excluyen mutuamente. Si se especifican ambas, se genera un error. En SQL Server 2005, las dos palabras clave se pueden utilizar en la misma instrucción DECLARE CURSOR.

El siguiente conjunto de parámetros que podemos especificar es [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]. A continuación mostramos el significado de cada una de estas opciones.

READ_ONLY

Evita que se efectúen actualizaciones a través de este cursor. No es posible hacer referencia al cursor en una cláusula WHERE CURRENT OF de una instrucción UPDATE o DELETE. Esta opción reemplaza la capacidad de actualizar el cursor.

```
DECLARE cClientes CURSOR READ_ONLY FOR
```

```
SELECT Id, Nombre, Apellido1,  
Apellido2, NifCif, FxNacimiento  
FROM CLIENTES
```

SCROLL_LOCKS

Especifica que se garantiza que las actualizaciones o eliminaciones posicionadas realizadas a través del cursor serán correctas. Microsoft SQL Server bloquea las filas cuando se leen en el cursor para garantizar que estarán disponibles para futuras modificaciones. No es posible especificar SCROLL_LOCKS si se especifica también FAST_FORWARD o STATIC.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
DECLARE cClientes CURSOR SCROLL_LOCKS FOR
SELECT Id, Nombre, Apellido1,
       Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

OPTIMISTIC

Especifica que las actualizaciones o eliminaciones posicionadas realizadas a través del cursor no se realizarán correctamente si la fila se ha actualizado después de ser leída en el cursor. SQL Server no bloquea las filas al leerlas en el cursor. En su lugar, utiliza comparaciones de valores de columna timestamp o un valor de suma de comprobación si la tabla no tiene columnas timestamp, para determinar si la fila se ha modificado después de leerla en el cursor. Si la fila se ha modificado, el intento de actualización o eliminación posicionada genera un error. No es posible especificar OPTIMISTIC si se especifica también FAST_FORWARD.

```
DECLARE cClientes CURSOR OPTIMISTIC FOR
SELECT Id, Nombre, Apellido1,
       Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Por último, queda la opción TYPE_WARNING

TYPE_WARNING

Especifica que se envía un mensaje de advertencia al cliente si el cursor se convierte implícitamente del tipo solicitado a otro.

```
DECLARE cClientes CURSOR TYPE_WARNING FOR
SELECT Id, Nombre, Apellido1,
       Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Podemos especificar multiples parámetros en la apertura de cursor, pero únicamente un parámetro de cada grupo. Por ejemplo:

```
DECLARE cClientes CURSOR LOCAL STATIC TYPE_WARNING FOR
SELECT Id, Nombre, Apellido1,
       Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Para actualizar los datos de un cursor debemos especificar FOR UPDATE después de la sentencia SELECT en la declaración del cursor, y WHERE CURRENT OF <nombre_cursor> en la sentencia UPDATE tal y como muestra el siguiente ejemplo.

-- Declaracion de variables para el cursor

```
DECLARE @Id int,
        @Nombre varchar(255),
        @Apellido1 varchar(255),
        @Apellido2 varchar(255),
        @NifCif varchar(20),
        @FxNacimiento datetime
```

-- Declaración del cursor

```
DECLARE cClientes CURSOR FOR
SELECT Id, Nombre, Apellido1,
       Apellido2, NifCif, FxNacimiento
FROM CLIENTES
FOR UPDATE
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

-- Apertura del cursor

OPEN cClientes

-- Lectura de la primera fila del cursor

FETCH cClientes

INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0)

BEGIN

UPDATE Clientes

SET APELLIDO2 = **isnull**(@Apellido2,") + ' - Modificado'

WHERE CURRENT OF cClientes

-- Lectura de la siguiente fila del cursor

FETCH cClientes

INTO @id, @Nombre, @Apellido1, @Apellido2,
@NifCif, @FxNacimiento

END

-- Cierre del cursor

CLOSE cClientes

-- Liberar los recursos

DEALLOCATE cClientes

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Practicas generales de TSQL

El siguiente es un esquema simple de una base de datos de una tienda, constituido por cuatro tablas

Tabla empleados

id	nombrecompleto	direccion	salario	Comisionporventa	celular
3	Stan Marsh	Colorado	120000	3%	45566
45	Kyle Broflovski	Los angeles	300000	4%	7877
65	Kenny McCormick	Sinaloa	150000	15%	6555
12	Eric Cartman	Hollywood	450000	7%	6777
89	Leopold Stotch	Hatillo	145000	8%	8899
31	Jimbo Kern	Bolivia	170000	2%	5456
22	Liane Cartman	San andreas	180000	4%	4533

Tabla producto

id	nombre	precio
3	Cepillo de dientes	200
4	Aguarrás	1000
21	Tijeras	500
99	Halo 5	56000
38	Tansung Tablet universe 8	230000
81	Patata	100
33	Arena para gato	700

Tabla bodega

producto	existencia
21	8
99	7
4	6
3	1
38	70
81	5
33	9

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Tabla facturas

idfactura	producto	fecha	empleado	cantidad
1	33	12/06/2013	3	5
4	4	15/07/2014	45	6
5	81	23/08/2012	65	3
90	21	15/04/2013	12	6
65	99	22/07/2012	89	8
78	3	15/01/2013	31	2
23	38	16/02/2014	22	5

Realice, las sentencias SQL en su cuaderno, necesarias para:

1. Seleccionar el monto total a pagar por cada factura
2. Seleccionar el salario de cada empleado más la comisión por cada venta en factura
3. Seleccione el nombre del empleado que vendió más (en dinero)
4. Seleccionar el nombre del producto con más existencias
5. Seleccionar el nombre del producto con menos existencias
6. Seleccione los productos vendidos entre enero y diciembre del 2012
7. Seleccione los productos vendidos cuyo vendedor gane mas comisión
8. Realice un trigger capaz de que cuando se introduzca un dato en factura se le reste la cantidad del producto en bodega
9. Realice un cursor capaz de recorrer la tabla facturas y seleccionar los artículos vendidos del último mes, cuya existencia sea menor al 10% de la cantidad vendida en esa factura

** Si tiene duda con alguna sentencia, pase a la siguiente.

Cuarto proyecto, realice los procedimientos almacenados, triggers y cursor necesarios para lograr la lógica de negocio del tercer proyecto, en parejas

Nivel: Quinto año

Diagramas de base de datos

En otras unidades hemos aprendido cómo es una base de datos relacional y hemos estudiado un lenguaje, el SQL, que nos proporciona mecanismos para crear estas bases de datos, así como para actualizarlas y consultarlas.

Sin embargo, todavía debemos resolver algunas cuestiones fundamentales para poder emplear la tecnología de las bases de datos relacionales; por ejemplo, cómo se puede decidir qué relaciones debe tener una base de datos determinada o qué atributos deben presentar las relaciones, qué claves primarias y qué claves foráneas se deben declarar, etc. La tarea de tomar este conjunto de decisiones recibe el nombre de diseñar la base de datos.

Una base de datos sirve para almacenar la información que se utiliza en un sistema de información determinado. Las necesidades y los requisitos de los futuros usuarios del sistema de información se deben tener en cuenta para poder tomar adecuadamente las decisiones anteriores.

Diseño conceptual: el modelo ER

En este apartado trataremos el diseño conceptual de una base de datos mediante el modelo ER. Lo que explicaremos es aplicable al diseño de cualquier tipo de bases de datos –relacional, jerárquica, etc.–, porque, como ya hemos dicho, en la etapa del diseño conceptual todavía no se tiene en cuenta la tecnología concreta que se utilizará para implementar la base de datos.

El modelo ER es uno de los enfoques de modelización de datos que más se utiliza actualmente por su simplicidad y legibilidad. Su legibilidad se ve favorecida porque proporciona una notación diagramática muy comprensiva. Es una herramienta útil tanto para ayudar al diseñador a reflejar en un modelo conceptual los requisitos del mundo real de interés como para comunicarse con el usuario final sobre el modelo conceptual obtenido y, de este modo, poder verificar si satisface sus requisitos.

El modelo ER resulta fácil de aprender y de utilizar en la mayoría de las aplicaciones.

Además, existen herramientas informáticas de ayuda al diseño (herramientas CASE*) que utilizan alguna variante del modelo ER para hacer el diseño de los datos.

El nombre completo del modelo ER es entity-relationship, y proviene del hecho de que los principales elementos que incluye son las entidades y las interrelaciones (entities y relationships). Traduciremos este nombre por ‘entidad-interrelación’.

El origen del modelo ER se encuentra en trabajos efectuados por Peter Chen en 1976. Posteriormente, muchos otros autores han descrito variantes y/o extensiones de este modelo. Así pues, en la literatura se encuentran muchas formas diferentes del modelo ER que pueden variar simplemente en la notación diagramática o en algunos de los conceptos en que se basan para modelizar los datos.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Cuando se quiere utilizar el modelo ER para comunicarse con el usuario, es recomendable emplear una variante del modelo que incluya sólo sus elementos más simples –entidades, atributos e interrelaciones– y, tal vez, algunas construcciones adicionales, como por ejemplo entidades débiles y dependencias de existencia. Éstos eran los elementos incluidos en el modelo original propuesto por Chen. En cambio, para llevar a cabo la tarea de modelizar propiamente dicha, suele ser útil usar un modelo ER más completo que incluya construcciones más avanzadas que extienden el modelo original.

Según la noción de modelo de datos que hemos utilizado en los otros módulos, un modelo de datos tiene en cuenta tres aspectos de los datos: la estructura, la manipulación y la integridad. Sin embargo, el modelo ER habitualmente se utiliza para reflejar aspectos de la estructura de los datos y de su integridad, pero no de su manipulación.

Construcciones básicas

Entidades, atributos e interrelaciones

Por entidad entendemos un objeto del mundo real que podemos distinguir del resto de objetos y del que nos interesan algunas propiedades.

Ejemplos de entidad

Algunos ejemplos de entidad son un empleado, un producto o un despacho. También son entidades otros elementos del mundo real de interés, menos tangibles pero igualmente diferenciables del resto de objetos; por ejemplo, una asignatura impartida en una universidad, un préstamo bancario, un pedido de un cliente, etc.

Las propiedades de los objetos que nos interesan se denominan atributos.

Ejemplos de atributo

Sobre una entidad empleado nos puede interesar, por ejemplo, tener registrados su DNI, su NSS, su nombre, su apellido y su sueldo como atributos.

El término entidad se utiliza tanto para denominar objetos individuales como para hacer referencia a conjuntos de objetos similares de los que nos interesan los mismos atributos; es decir, que, por ejemplo, se utiliza para designar tanto a un empleado concreto de una empresa como al conjunto de todos los empleados de la empresa. Más concretamente, el término entidad se puede referir a instancias u ocurrencias concretas (empleados concretos) o a tipos o clases de entidades (el conjunto de todos los empleados).

El modelo ER proporciona una notación diagramática para representar gráficamente las entidades y sus atributos:

- Las entidades se representan con un rectángulo. El nombre de la entidad se escribe en mayúsculas dentro del rectángulo.

Nivel: Quinto año

- Los atributos se representan mediante su nombre en minúsculas unido con un guion al rectángulo de la entidad a la que pertenecen. Muchas veces, dado que hay muchos atributos para cada entidad, se listan todos aparte del diagrama para no complicarlo.

Cada uno de los atributos de una entidad toma valores de un cierto dominio o conjunto de valores. Los valores de los dominios deben ser atómicos; es decir, no deben poder ser descompuestos. Además, todos los atributos tienen que ser univalueados. Un atributo es univalueado si tiene un único valor para cada ocurrencia de una entidad



Ejemplo de atributo univalueado

El atributo sueldo de la entidad empleado, por ejemplo, toma valores del dominio de los reales y únicamente toma un valor para cada empleado concreto; por lo tanto, ningún empleado puede tener más de un valor para el sueldo.

Como ya hemos comentado anteriormente, una entidad debe ser distinguible del resto de objetos del mundo real. Esto hace que para toda entidad sea posible encontrar un conjunto de atributos que permitan identificarla. Este conjunto de atributos forma una clave de la entidad.

Ejemplo de clave

La entidad empleado tiene una clave que consta del atributo dni porque todos los empleados tienen números de DNI diferentes.

Una determinada entidad puede tener más de una clave; es decir, puede tener varias claves candidatas.

Ejemplo de clave candidata

La entidad empleado tiene dos claves candidatas, la que está formada por el atributo dni y la que está constituida por el atributo nss, teniendo en cuenta que el NSS también será diferente para cada uno de los empleados.

El diseñador elige una clave primaria entre todas las claves candidatas. En la notación diagramática, la clave primaria se subraya para distinguirla del resto de las claves.

Ejemplo de clave primaria

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

En el caso de la entidad empleado, podemos elegir dni como clave primaria. En la figura del margen vemos que la clave primaria se subraya para distinguirla del resto.

Se define interrelación como una asociación entre entidades.

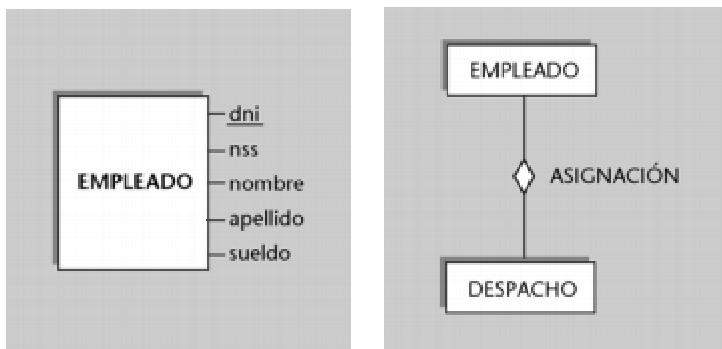
Las interrelaciones se representan en los diagramas del modelo ER mediante un rombo. Junto al rombo se indica el nombre de la interrelación con letras mayúsculas.

Ejemplo de interrelación

Consideremos una entidad empleado y una entidad despacho y supongamos que a los empleados se les asignan despachos donde trabajar. Entonces hay una interrelación entre la entidad empleado y la entidad despacho.

Esta interrelación, que podríamos denominar asignación, asocia a los empleados con los despachos donde trabajan. La figura del margen muestra la interrelación asignación entre las entidades empleado y despacho.

El término interrelación se puede utilizar tanto para denominar asociaciones concretas u ocurrencias de asociaciones como para designar conjuntos o clases de asociaciones similares.



Ejemplo

Una interrelación se aplica tanto a una asociación concreta entre el empleado de DNI '50.455.234' y el despacho 'Diagonal, 20' como a la asociación genérica entre la entidad empleado y la entidad despacho.

En ocasiones interesa reflejar algunas propiedades de las interrelaciones.

Por este motivo, las interrelaciones pueden tener también atributos.

Los atributos de las interrelaciones, igual que los de las entidades, tienen un cierto dominio, deben tomar valores atómicos y deben ser univalueados.

Los atributos de las interrelaciones se representan mediante su nombre en minúsculas unido con un guion al rombo de la interrelación a la que pertenecen.

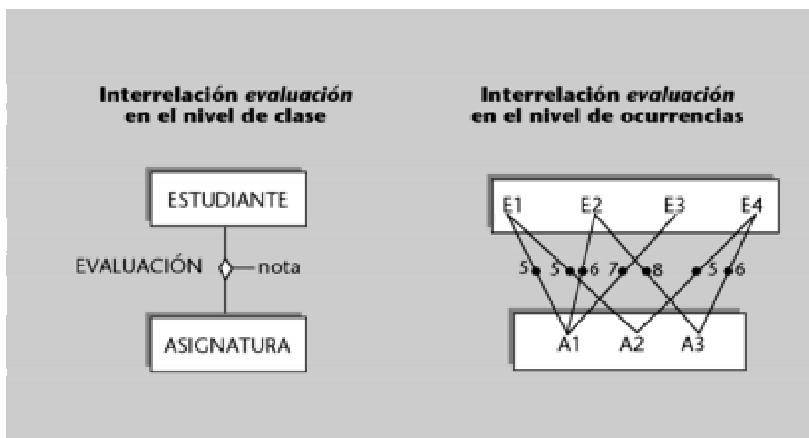
Ejemplo de atributo de una interrelación

Observemos la entidad estudiante y la entidad asignatura que se muestran en la figura siguiente:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



Entre estas dos entidades se establece la interrelación evaluación para indicar de qué asignaturas han sido evaluados los estudiantes. Esta interrelación tiene el atributo nota, que sirve para especificar qué nota han obtenido los estudiantes de las asignaturas evaluadas.

Conviene observar que el atributo nota debe ser forzosamente un atributo de la interrelación evaluación, y que no sería correcto considerarlo un atributo de la entidad estudiante o un atributo de la entidad asignatura. Lo explicaremos analizando las ocurrencias de la interrelación evaluación que se muestran en la figura anterior.

Si nota se considerase un atributo de estudiante, entonces para el estudiante 'E1' de la figura necesitaríamos dos valores del atributo, uno para cada asignatura que tiene el estudiante; por lo tanto, no sería univaluado. De forma similar, si nota fuese atributo de asignatura tampoco podría ser univaluado porque, por ejemplo, la asignatura 'A1' requeriría tres valores de nota, una para cada estudiante que se ha matriculado en ella. Podemos concluir que el atributo nota está relacionado al mismo tiempo con una asignatura y con un estudiante que la cursa y que, por ello, debe ser un atributo de la interrelación que asocia las dos entidades.

Interrelaciones binarias

Conectividad de las interrelaciones binarias

La conectividad de una interrelación expresa el tipo de correspondencia que se establece entre las ocurrencias de entidades asociadas con la interrelación. En el caso de las interrelaciones binarias, expresa el número de ocurrencias de una de las entidades con las que una ocurrencia de la otra entidad puede estar asociada según la interrelación.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

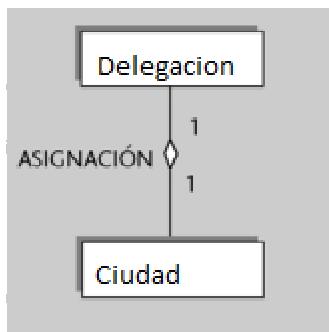
Una interrelación binaria entre dos entidades puede tener tres tipos de conectividad:

- Conectividad uno a uno (1:1). La conectividad 1:1 se denota poniendo un 1 a lado y lado de la interrelación.
- Conectividad uno a muchos (1:N). La conectividad 1:N se denota poniendo un 1 en un lado de la interrelación y una N en el otro.
- Conectividad muchos a muchos: (M:N). La conectividad M:N se denota poniendo una M en uno de los lados de la interrelación, y una N en el otro.

Ejemplos de conectividad en una interrelación binaria

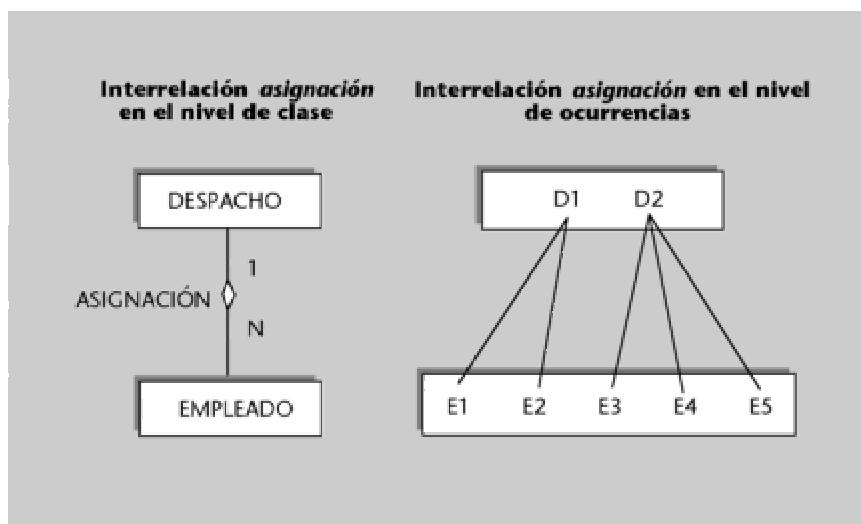
A continuación analizaremos un ejemplo de cada una de las conectividades posibles para una interrelación binaria:

- a) Conectividad 1:1



La interrelación anterior tiene conectividad 1:1. Esta interrelación asocia las delegaciones de una empresa con las ciudades donde están situadas. El hecho de que sea 1:1 indica que una ciudad tiene sólo una delegación, y que una delegación está situada en una única ciudad.

- b) Conectividad 1:N



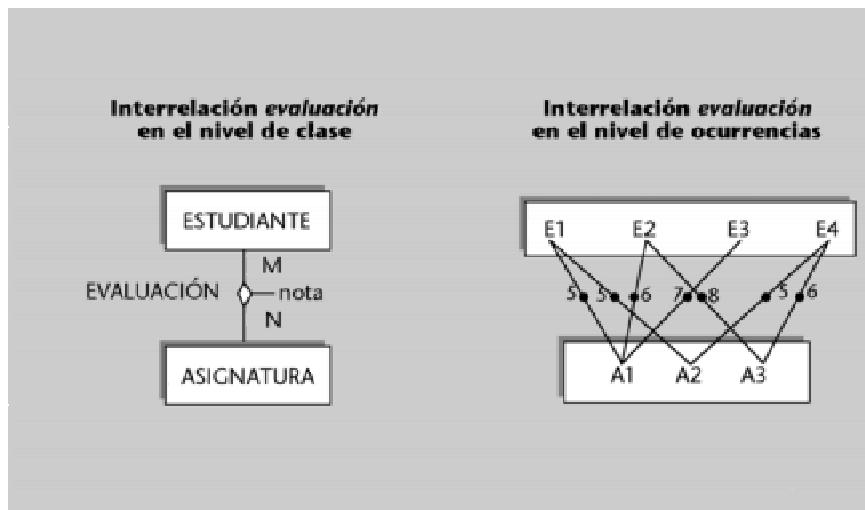
Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

La interrelación asignación entre la entidad empleado y la entidad despacho tiene conectividad 1:N, y la N está en el lado de la entidad empleado. Esto significa que un empleado tiene un solo despacho asignado, pero que, en cambio, un despacho puede tener uno o más empleados asignados.

c) Conectividad M:N



Para analizar la conectividad M:N, consideramos la interrelación evaluación de la figura anterior.

Nos indica que un estudiante puede ser evaluado de varias asignaturas y, al mismo tiempo, que una asignatura puede tener varios estudiantes por evaluar.

Es muy habitual que las interrelaciones binarias M:N y todas las n-arias tengan atributos. En cambio, las interrelaciones binarias 1:1 y 1:N no tienen por qué tenerlos. Siempre se pueden asignar estos atributos a la entidad del lado N, en el caso de las 1:N, y a cualquiera de las dos entidades interrelacionadas en el caso de las 1:1. Este cambio de situación del atributo se puede hacer porque no origina un atributo multivalorado.

Dependencias de existencia en las interrelaciones binarias

En algunos casos, una entidad individual sólo puede existir si hay como mínimo otra entidad individual asociada con ella mediante una interrelación binaria determinada. En estos casos, se dice que esta última entidad es una entidad obligatoria en la interrelación. Cuando esto no sucede, se dice que es una entidad opcional en la interrelación.

En el modelo ER, un círculo en la línea de conexión entre una entidad y una interrelación indica que la entidad es opcional en la interrelación. La obligatoriedad de una entidad a una interrelación se indica con una línea perpendicular.

Si no se consigna ni un círculo ni una línea perpendicular, se considera que la dependencia de existencia es desconocida.

Informática en Desarrollo

CEDES DON BOSCO

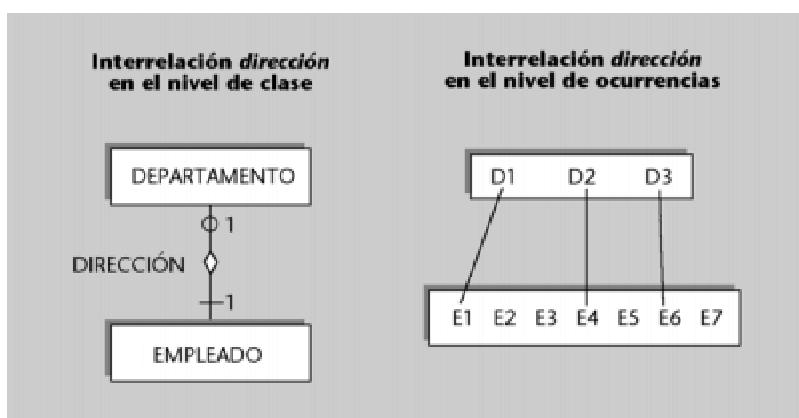
Nivel: Quinto año

Ejemplo de dependencias de existencia

La figura siguiente nos servirá para entender el significado práctico de la dependencia de existencia.

La entidad empleado es obligatoria en la interrelación dirección. Esto indica que no puede existir un departamento que no tenga un empleado que actúa de director del departamento. La entidad departamento, en cambio, es opcional en la interrelación dirección.

Es posible que haya un empleado que no está interrelacionado con ningún departamento: puede haber –y es el caso más frecuente– empleados que no son directores de departamento.



Ejemplo: base de datos de casas de colonias

En este punto, y antes de continuar explicando construcciones más complejas del modelo ER, puede resultar muy ilustrativo ver la aplicación práctica de las construcciones que hemos estudiado hasta ahora. Por este motivo, analizaremos un caso práctico de diseño con el modelo ER que corresponde a una base de datos destinada a la gestión de las inscripciones en un conjunto de casas de colonias. El modelo ER de esta base de datos será bastante sencillo e incluirá sólo entidades, atributos y interrelaciones binarias (no incluirá interrelaciones n-arias ni otros tipos de estructuras).

La descripción siguiente explica con detalle los requisitos de los usuarios que hay que tener en cuenta al hacer el diseño conceptual de la futura base de datos:

- Cada casa de colonias tiene un nombre que la identifica. Se desea saber de cada una, aparte del nombre, la capacidad (el número de niños que se pueden alojar en cada una como máximo), la comarca donde está situada y las ofertas de actividades que proporciona. Una casa puede ofrecer actividades como por ejemplo natación, esquí, remo, pintura, fotografía, música, etc.
- Es necesario tener en cuenta que en una casa de colonias se pueden practicar varias actividades (de hecho, cada casa debe ofrecer como mínimo una), y también puede ocurrir que una misma actividad se pueda llevar a cabo en varias casas. Sin embargo, toda actividad que se registre en la base de datos debe ser ofertada como mínimo en una de las casas.

Informática en Desarrollo

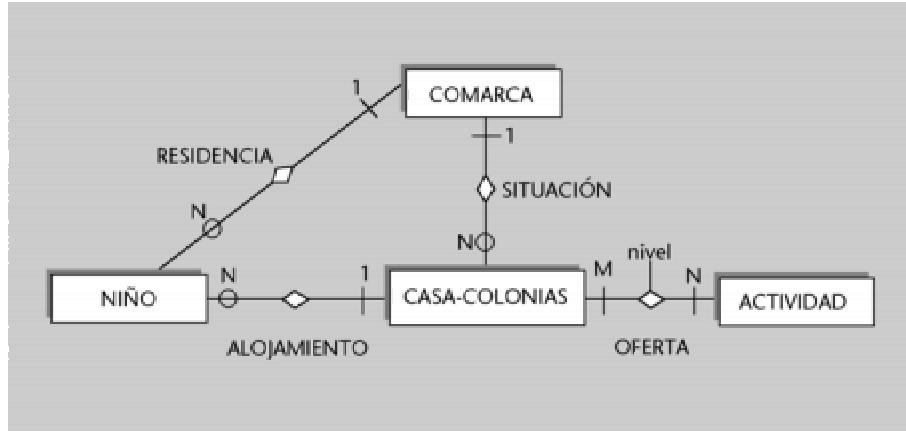
CEDES DON BOSCO

Nivel: Quinto año

- c) Interesa tener una evaluación de las ofertas de actividades que proporcionan las casas. Se asigna una calificación numérica que indica el nivel de calidad que tiene cada una de las actividades ofertadas.
- d) Las casas de colonias alojan niños que se han inscrito para pasar en ellas unas pequeñas vacaciones. Se quiere tener constancia de los niños que se alojan en cada una de las casas en el momento actual. Se debe suponer que hay casas que están vacías (en las que no se aloja ningún niño) durante algunas temporadas.
- e) De los niños que se alojan actualmente en alguna de las casas, interesa conocer un código que se les asigna para identificarlos, su nombre, su apellido, el número de teléfono de sus padres y su comarca de residencia.
- f) De las comarcas donde hay casas o bien donde residen niños, se quiere tener registrados la superficie y el número de habitantes. Se debe considerar que puede haber comarcas donde no reside ninguno de los niños que se alojan en un momento determinado en las casas de colonias, y comarcas que no disponen de ninguna casa.

La figura siguiente muestra un diagrama ER que satisface los requisitos anteriores.

Los atributos de las entidades no figuran en el diagrama y se listan aparte.



Los atributos de las entidades que figuran en el diagrama son los siguientes (las claves primarias están subrayadas):

Nivel: Quinto año

CASA-COLONIAS

nombre-casa, capacidad

ACTIVIDAD

nombre-actividad

NIÑO

código-niño, nombre, apellido, teléfono

COMARCA

nombre-comarca, superficie, número-habitantes

Entidades débiles

Las entidades que hemos considerado hasta ahora tienen un conjunto de atributos que forman su claves primarias y que permiten identificarlas completamente.

Estas entidades se denominan, de forma más específica, entidades fuertes. En este subapartado consideraremos otro tipo de entidades que denominaremos entidades débiles.

Una entidad débil es una entidad cuyos atributos no la identifican completamente, sino que sólo la identifican de forma parcial. Esta entidad debe participar en una interrelación que ayuda a identificarla.

Una entidad débil se representa con un rectángulo doble, y la interrelación que ayuda a identificarla se representa con una doble línea.

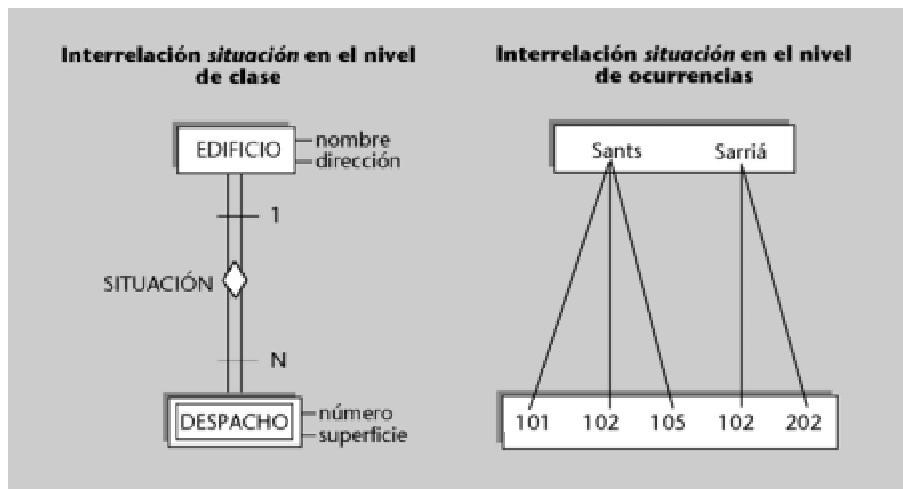
Ejemplo de entidad débil

Consideremos las entidades edificio y despacho de la figura siguiente. Supongamos que puede haber despachos con el mismo número en edificios diferentes. Entonces, su número no identifica completamente un despacho. Para identificar completamente un despacho, es necesario tener en cuenta en qué edificio está situado. De hecho, podemos identificar un despacho mediante la interrelación situación, que lo asocia a un único edificio. El nombre del edificio donde está situado junto con el número de despacho lo identifican completamente.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



En el ejemplo anterior, la interrelación situación nos ha permitido completar la identificación de los despachos. Para toda entidad débil, siempre debe haber una única interrelación que permita completar su identificación. Esta interrelación debe ser binaria con conectividad 1:N, y la entidad débil debe estar en el lado N. De este modo, una ocurrencia de la entidad débil está asociada con una sola ocurrencia de la entidad del lado 1, y será posible completar su identificación de forma única. Además, la entidad del lado 1 debe ser obligatoria en la interrelación porque, si no fuese así, alguna ocurrencia de la entidad débil podría no estar interrelacionada con ninguna de sus ocurrencias y no se podría identificar completamente.

Informática en Desarrollo

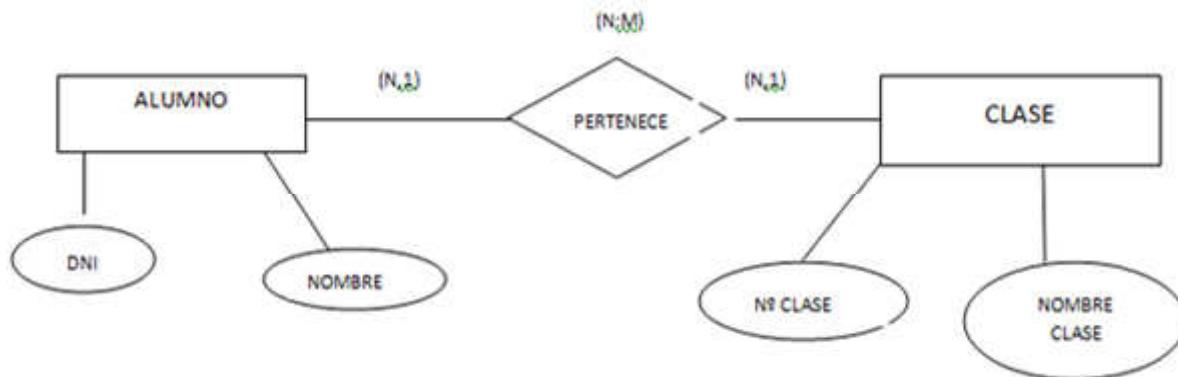
CEDES DON BOSCO

Nivel: Quinto año

Diseño lógico relacional

Para pasar el modelo entidad relación que vimos anteriormente a un modelo relacional, se deben seguir las siguientes reglas:

- Toda entidad se transforma en una tabla
- Todo atributo se transforma en una columna dentro de la tabla a la que pertenece
- El identificador de la entidad se convierte en la clave primaria de la tabla
- Toda relación N:M se convierte en una tabla que tendrá como clave primaria las dos claves primarias de las entidades que se asocian
- En las relaciones 1:N la clave primaria de la entidad con cardinalidad 1 pasa a la tabla de la entidad cuya cardinalidad es N
- En las relaciones N:M existen tres posibilidades: Si la cardinalidad es (0,1) en ambas entidades, se crea tabla. Mientras que si la cardinalidad de una es (0,1) y de la otra es (1,1) se suele pasar la clave primaria de (1,1) a la de (0,1). Si la cardinalidad de ambas es (1,1) se pasa la clave de cualquiera de ellas a la otra.



Para este modelo de entidad-relación el paso a tablas quedaría de la siguiente forma:

Tabla alumno	DNI(clave primaria)	nombre
Tabla clase	Nº clase (clave primaria)	Nombre clase
Tabla pertenece	DNI (clave foránea)	Nº clase (clave foránea)

Clave primaria

Nivel: Quinto año

Diagrama físico

El diseño físico es el proceso de producir la descripción de la implementación de la base de datos en memoria secundaria, a partir del esquema lógico obtenido en la etapa anterior. Para especificar dicha implementación se debe determinar las estructuras de almacenamiento y escoger los mecanismos necesarios para garantizar un acceso eficiente a los datos. Puesto que el esquema lógico utiliza el modelo relacional, la implementación del diseño físico se realizará en SQL.

Prácticas generales de Diagramas de SQL Server

1. Hacer un diseño conceptual de una base de datos mediante el modelo ER que satisfaga los requisitos que se resumen a continuación:

- a) Un directorio de un club de fútbol quiere disponer de una base de datos que le permita controlar datos que le interesan sobre competiciones, clubes, jugadores, entrenadores, etc. De ámbito estatal.
- b) Los clubes disputan cada temporada varias competiciones (liga, copa, etc.) entre sí. Nuestro directorio desea información histórica de las clasificaciones obtenidas por los clubes en las diferentes competiciones a lo largo de todas las temporadas. La clasificación se especificará mediante un número de posición: 1 significa campeón, 2 significa subcampeón, etc.
- c) Los distintos clubes están agrupados en las federaciones regionales correspondientes. Toda federación tiene como mínimo un club. Quiere saber el nombre y la fecha de creación de las federaciones así como el nombre y el número de socios de los clubes.
- d) Es muy importante la información sobre jugadores y entrenadores. Se identificarán por un código, y quiere saber el nombre, la dirección, el número de teléfono y la fecha de nacimiento de todos. Es necesario mencionar que algunos entrenadores pueden haber sido jugadores en su juventud. De los jugadores, además, quiere saber el peso, la altura, la especialidad o las especialidades y qué dominio tienen de ellas (grado de especialidad). Todo jugador debe tener como mínimo una especialidad, pero puede haber especialidades en las que no haya ningún jugador. De los entrenadores le interesa la fecha en que iniciaron su carrera como entrenadores de fútbol.
- e) De todas las personas que figuran en la base de datos (jugadores y entrenadores), quiere conocer el historial de contrataciones por parte de los diferentes clubes, incluyendo el importe y la fecha de baja de cada contratación. En un momento determinado, una persona puede estar contratada por un único club, pero puede cambiar de club posteriormente e, incluso, puede volver a un club en el que ya había trabajado.
- f) También quiere registrar las ofertas que las personas que figuran en la base de datos han recibido de los clubes durante su vida deportiva (y de las que se ha enterado). Considera básico tener constancia del importe de las ofertas. Se debe tener en cuenta que, en un momento

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

determinado, una persona puede recibir muchas ofertas, siempre que provengan de clubes distintos.

2. Haced un diseño conceptual de una base de datos mediante el modelo ER que satisfaga los requisitos que se resumen a continuación:

a) Se quiere diseñar una base de datos para facilitar la gestión de una empresa dedicada al transporte internacional de mercancías que opera en todo el ámbito europeo.

b) La empresa dispone de varias delegaciones repartidas por toda la geografía europea. Las delegaciones se identifican por un nombre, y se quiere registrar también su número de teléfono. En una determinada ciudad no hay nunca más de una delegación. Se desea conocer la ciudad donde está situada cada delegación. Se debe suponer que no hay ciudades con el nombre repetido (por lo menos en el ámbito de esta base de datos).

c) El personal de la empresa se puede separar en dos grandes grupos:

- Administrativos, sobre los cuales interesa saber su nivel de estudios.
- Conductores, sobre los que interesa saber el año en el que obtuvieron el carnet de conducir y el tipo de carnet que tienen.

De todo el personal de la empresa, se quiere conocer el código de empleado (que lo identifica), su nombre, su número de teléfono y el año de nacimiento. Todos los empleados están asignados a una delegación determinada. Se quiere tener constancia histórica de este hecho teniendo en cuenta que pueden ir cambiando de delegación (incluso pueden volver a una delegación donde ya habían estado anteriormente).

d) La actividad de la empresa consiste en efectuar los viajes pertinentes para transportar las mercancías según las peticiones de sus clientes. Todos los clientes se identifican por un código de cliente. Se quiere conocer, además, el nombre y el teléfono de contacto de cada uno de ellos.

e) La empresa, para llevar a cabo su actividad, dispone de muchos camiones identificados por un código de camión. Se quiere tener constancia de la matrícula, la marca y la tara de los camiones.

f) Los viajes los organiza siempre una delegación, y se identifican mediante un código de viaje, que es interno de cada delegación (y que se puede repetir en delegaciones diferentes). Para cada uno de los viajes que se han hecho, es necesario saber:

- Qué camión se ha utilizado (ya que cada viaje se hace con un solo camión).
- Qué conductor o conductores han ido (considerando que en viajes largos pueden ir varios conductores). Se quiere saber también el importe de las dietas pagadas a cada conductor (teniendo en cuenta que las dietas pueden ser diferentes para los diferentes conductores de un mismo viaje).

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

- El recorrido del viaje; es decir, la fecha y la hora en que el camión llega a cada una de las ciudades donde deberá cargar o descargar. Supondremos que un viaje no pasa nunca dos veces por una misma ciudad.
- El número de paquetes cargados y de paquetes descargados en cada ciudad, y para cada uno de los clientes. En un mismo viaje se pueden dejar y/o recoger paquetes en diferentes ciudades por encargo de un mismo cliente. También, en un mismo viaje, se pueden dejar y/o recoger paquetes en una misma ciudad por encargo de diferentes clientes.

3. Haced un diseño conceptual de una base de datos mediante el modelo ER que satisfaga los requisitos que se resumen a continuación:

- a) Es necesario diseñar una base de datos para una empresa inmobiliaria con el objetivo de gestionar la información relativa a su cartera de pisos en venta.
- b) Cada uno de los pisos que tienen pendientes de vender tiene asignado un código de piso que lo identifica. Además de este código, se quiere conocer la dirección del piso, la superficie, el número de habitaciones y el precio. Tienen estos pisos clasificados por zonas (porque a sus clientes, en ocasiones, sólo les interesan los pisos de una zona determinada) y se quiere saber en qué zona está situado cada piso. Las zonas tienen un nombre de zona que es diferente para cada una de una misma población, pero que pueden coincidir en zonas de poblaciones diferentes.

En ocasiones sucede que en algunas de las zonas no tienen ningún piso pendiente de vender.

- c) Se quiere tener el número de habitantes de las poblaciones. Se quiere saber qué zonas son limítrofes, (porque, en caso de no disponer de pisos en una zona que desea un cliente, se le puedan ofrecer los que tengan en otras zonas limítrofes). Es necesario considerar que pueden existir zonas sin ninguna zona limítrofe en algunas poblaciones pequeñas que constan de una sola zona.

- d) Se disponen de diferentes características codificadas de los pisos, como por ejemplo tener ascensor, ser exterior, tener terraza, etc. Cada característica se identifica mediante un código y tiene una descripción. Para cada característica y cada piso se quiere saber si el piso satisface la característica o no. Además, quieren tener constancia del propietario o los propietarios de cada piso.

- e) También necesitan disponer de información relativa a sus clientes actuales que buscan piso (si dos o más personas buscan piso conjuntamente, sólo se guarda información de una de ellas como cliente de la empresa). En particular, interesa saber las zonas donde busca piso cada cliente (sólo en caso de que tenga alguna zona de preferencia).

- f) A cada uno de estos clientes le asignan un vendedor de la empresa para que se ocupe de atenderlo. A veces, estas asignaciones varían con el tiempo y se cambia al vendedor asignado a un determinado cliente. También es posible que a un cliente se le vuelva a asignar un vendedor que ya había tenido con anterioridad. Se quiere tener constancia de las asignaciones de los clientes actuales de la empresa.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

g) Los vendedores, clientes y propietarios se identifican por un código de persona. Se quiere registrar, de todos, su nombre, dirección y número de teléfono. Además, se quiere disponer del número de Seguridad Social y el sueldo de los vendedores, y del NIF de los propietarios.

Puede haber personas que sean al mismo tiempo clientes y propietarios, o bien vendedores y propietarios, etc.

h) Finalmente, para ayudar a programar y consultar las visitas que los clientes hacen a los pisos en venta, se quiere guardar información de todas las visitas correspondientes a los clientes y a los pisos actuales de la empresa. De cada visita hay que saber el cliente que la hace, el piso que se va a ver y la hora concreta en que se inicia la visita. Entendemos que la hora de la visita está formada por la fecha, la hora del día y el minuto del día (por ejemplo, 25-FEB-98, 18:30).

Hay que considerar que un mismo cliente puede visitar un mismo piso varias veces para asegurarse de si le gusta o no, y también que para evitar conflictos no se programan nunca visitas de clientes diferentes a un mismo piso y a la misma hora.

4. Transformad a relacional el diseño conceptual que habéis obtenido en el ejercicio 1.
5. Transformad a relacional el diseño conceptual que habéis obtenido en el ejercicio 2.
6. Transformad a relacional el diseño conceptual que habéis obtenido en el ejercicio 3

Quinto proyecto, crear los tres diagramas de la base de datos hecha en el ejercicio tercero y cuarto, en parejas.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Formas Normales

Dependencia funcional

Uno de los conceptos fundamentales en la normalización es el de dependencia funcional. Una dependencia funcional es una relación entre atributos de una misma tabla. Si x e y son atributos de la relación R , se dice que y es funcionalmente dependiente de x (se denota por $x \rightarrow y$) si cada valor de x tiene asociado un solo valor de y (x e y pueden constar de uno o varios atributos). A x se le denomina determinante, ya que x determina el valor de y . Se dice que el atributo y es completamente dependiente de x si depende funcionalmente de x y no depende de ningún subconjunto de x .

La dependencia funcional es una noción semántica. Si hay o no dependencias funcionales entre atributos, no lo determina una serie abstracta de reglas, sino, más bien, los modelos mentales del usuario y las reglas de negocio de la organización o empresa para la que se desarrolla el sistema de información.

Cada dependencia funcional es una restricción y representa una relación de uno a muchos (o de uno a uno).

En el proceso de normalización debe irse comprobando que cada tabla cumple una serie de reglas que se basan en la clave primaria y las dependencias funcionales. Cada regla que se cumple aumenta el grado de normalización. Si una regla no se cumple, la tabla se debe descomponer en varias tablas que sí la cumplan.

La normalización se lleva a cabo en una serie de pasos. Cada paso corresponde a una forma normal que tiene unas propiedades. Conforme se va avanzando en la normalización, las tablas tienen un formato más estricto (más fuerte) y, por lo tanto, son menos vulnerables a las anomalías de actualización. El modelo relacional sólo requiere un conjunto de tablas en primera forma normal (en caso contrario no se pueden implementar). Las restantes formas normales son opcionales. Sin embargo, para evitar las anomalías de actualización, es recomendable llegar al menos a la tercera forma normal.

Primera forma normal

Una tabla está en primera forma normal (1fn) si, y sólo si, todos los dominios de sus atributos contienen valores atómicos, es decir, no hay grupos repetitivos. Un grupo repetitivo es un atributo que puede tener múltiples valores para cada fila de la relación. Son los atributos que tienen forma de tabla.

Si una tabla no está en 1fn, hay que eliminar de ella los grupos repetitivos.

La forma de eliminar los grupos repetitivos consiste en poner cada uno de ellos como una tabla aparte, heredando la clave primaria de la tabla en la que se encontraban. La clave primaria de esta nueva tabla estará formada por la combinación de la clave primaria que tenía cuando era

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

un grupo repetitivo y la clave primaria que ha heredado en forma de clave ajena. Se dice que conjunto de tablas se encuentra en 1fn si ninguna de ellas tiene grupos repetitivos.

La tabla PRODUCTO que se muestra a continuación no se encuentra en 1fn, ya que tiene un grupo repetitivo:

PRODUCTO(codprod, nombre, VERSIÓN(número, fecha, ventas))

Para pasarla a 1fn se debe eliminar el grupo repetitivo:

PRODUCTO(codprod, nombre)

VERSIÓN(codprod, número, fecha, ventas)

VERSIÓN.codprod es una clave ajena a PRODUCTO

Segunda forma normal

Una tabla está en segunda forma normal (2fn) si, y sólo si, está en 1fn y, además, cada atributo que no forma parte de la clave primaria es completamente dependiente de la clave primaria.

La 2fn se aplica a las tablas que tienen claves primarias compuestas por dos o más atributos. Si una tabla está en 1fn y su clave primaria es simple (tiene un solo atributo), entonces también está en 2fn. Las tablas que no están en 2fn pueden sufrir anomalías cuando se realizan actualizaciones sobre ellas.

Para pasar una tabla en 1fn a 2fn hay que eliminar las dependencias parciales de la clave primaria. Para ello, se eliminan los atributos que son funcionalmente dependientes y se ponen en una nueva tabla con una copia de su determinante. Su determinante estará formado por los atributos de la clave primaria de los que depende.

En la tabla INSCRIPCIÓN que aparece a continuación existe una dependencia funcional parcial de la clave primaria:

INSCRIPCIÓN(estudiante, actividad, precio)

Dependencia funcional parcial: actividad → precio

Esta dependencia existe porque cada actividad tiene un precio, independientemente del estudiante que se inscriba. Las anomalías que se pueden producir si se mantiene esta dependencia dentro de la tabla son varias. Por una parte, no es posible conocer el precio de una actividad si no hay personas inscritas, ya sea porque no se ha inscrito ninguna o porque todas las que lo están cancelan su inscripción. Por otra parte, y que es aún más grave, si se cambia el precio de una actividad y no se cambia para todas las personas inscritas, se tendrá una falta de integridad ya que habrá dos precios para la misma actividad, uno correcto y otro erróneo. Para pasar la tabla a 2fn se debe eliminar el atributo de la dependencia parcial, que se lleva una copia de su determinante:

ACTIVIDAD(actividad, precio)

INSCRIPCIÓN(estudiante, actividad)

Nivel: Quinto año

INSCRIPCIÓN.activity es una clave ajena a ACTIVIDAD

De este modo se evitan las anomalías citadas anteriormente: puede conocerse el precio de las actividades sin haber inscripciones y, puesto que el precio sólo está almacenado una vez, si se cambia éste, será el mismo para todas las inscripciones.

Tercera forma normal

Una tabla está en tercera forma normal (3fn) si, y sólo si, está en 2fn y, además, cada atributo que no forma parte de la clave primaria no depende transitivamente de la clave primaria. La dependencia $x \rightarrow z$ es transitiva si existen las dependencias $x \rightarrow y$, $y \rightarrow z$, siendo x , y , z atributos o conjuntos de atributos de una misma tabla.

Aunque las relaciones en 2fn tienen menos redundancias que las relaciones en 1fn, todavía pueden sufrir anomalías frente a las actualizaciones. Para pasar una relación en 2fn a 3fn hay que eliminar las dependencias transitivas. Para ello, se eliminan los atributos que dependen transitivamente y se ponen en una nueva relación con una copia de su determinante (el atributo o atributos no clave de los que depende).

Ejemplo 7.9 Pasar una tabla en 2 fn a 3 fn.

En la tabla HABITA existe una dependencia funcional transitiva:

HABITA(inquilino, edificio, alquiler)

Dependencia funcional transitiva: edificio \rightarrow alquiler

Esta dependencia existe porque cada edificio tiene un alquiler, independientemente del inquilino que lo habite. Una vez más, mantener esta dependencia dentro de la tabla puede dar lugar a diversas anomalías: no es posible conocer el alquiler de un edificio si no hay inquilinos y si se modifica el precio del alquiler de un edificio sólo para algunos inquilinos se viola una regla del negocio, ya que todos los inquilinos del mismo edificio deben pagar lo mismo. Para pasar la tabla a 3fn se debe eliminar el atributo de la dependencia transitiva, que se lleva una copia de su determinante:

ALQUILER(edificio, alquiler)

HABITA(inquilino, edificio)

HABITA.edificio es una clave ajena a ALQUILER

Descomponiendo la tabla de este modo, se evitan las anomalías que se han citado.

Forma normal de Boyce-Codd

Una tabla está en la forma normal de Boyce-Codd (bcfn) si, y sólo si, todo determinante es una clave candidata.

La 2fn y la 3fn eliminan las dependencias parciales y las dependencias transitivas de la clave primaria. Pero este tipo de dependencias todavía pueden existir sobre otras claves candidatas, si éstas existen. La bcfn es más fuerte que la 3fn, por lo tanto, toda tabla en bcfn está en 3fn.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

La violación de la bcfn es poco frecuente ya que se da bajo ciertas condiciones que raramente se presentan. Se debe comprobar si una tabla viola la bcfn en caso de tener dos o más claves candidatas compuestas que tienen al menos un atributo en común.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Programación Orientada a Objetos

La orientación a objetos es un paradigma de programación que facilita la creación de software de calidad por sus factores que potencian el mantenimiento, la extensión y la reutilización del software generado bajo este paradigma.

La programación orientada a objetos trata de amoldarse al modo de pensar del hombre y no al de la máquina. Esto es posible gracias a la forma racional con la que se manejan las abstracciones que representan las entidades del dominio del problema, y a propiedades como la jerarquía o el encapsulamiento.

El elemento básico de este paradigma no es la función (elemento básico de la programación estructurada), sino un ente denominado objeto. Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones que pueden modificar dicho estado, y determinan las capacidades del objeto.

Java incorpora el uso de la orientación a objetos como uno de los pilares básicos de su lenguaje.

Los objetos

Podemos definir objeto como el "encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios". [Piattini et al., 1996].

Un objeto además de un estado interno, presenta una interfaz para poder interactuar con el exterior. Es por esto por lo que se dice que en la programación orientada a objetos "se unen datos y procesos", y no como en su predecesora, la programación estructurada, en la que estaban separados en forma de variables y funciones.

Un objeto consta de:

- Tiempo de vida: La duración de un objeto en un programa siempre está limitada en el tiempo. La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Los objetos son creados mediante un mecanismo denominado instanciación, y cuando dejan de existir se dice que son destruidos.
- Estado: Todo objeto posee un estado, definido por sus atributos. Con él se definen las propiedades del objeto, y el estado en que se encuentra en un momento determinado de su existencia.
- Comportamiento: Todo objeto ha de presentar una interfaz, definida por sus métodos, para que el resto de objetos que componen los programas puedan interactuar con él.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

El equivalente de un objeto en el paradigma estructurado sería una variable. Así mismo la instanciación de objetos equivaldría a la declaración de variables, y el tiempo de vida de un objeto al ámbito de una variable.

Las clases

Las clases son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común.

Podemos definir una clase como "un conjunto de cosas (físicas o abstractas) que tienen el mismo comportamiento y características... Es la implementación de un tipo de objeto (considerando los objetos como instancias de las clases)". [Piattini et al., 1996].

Una clase no es más que una plantilla para la creación de objetos. Cuando se crea un objeto (instanciación) se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto.

Las clases presentan el estado de los objetos a los que representan mediante variables denominadas atributos. Cuando se instancia un objeto el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece el objeto.

Los métodos son las funciones mediante las que las clases representan el comportamiento de los objetos. En dichos métodos se modifican los valores de los atributos del objeto, y representan las capacidades del objeto (en muchos textos se les denomina servicios).

Desde el punto de vista de la programación estructurada, una clase se asemejaría a un módulo, los atributos a las variables globales de dicho módulo, y los métodos a las funciones del módulo.

Modelo de objetos

Existen una serie de principios fundamentales para comprender cómo se modeliza la realidad al crear un programa bajo el paradigma de la orientación a objetos.

Estos principios son: la abstracción, el encapsulamiento, la modularidad, la jerarquía, el paso de mensajes y el polimorfismo.

a.) Principio de Abstracción

Mediante la abstracción la mente humana modeliza la realidad en forma de objetos. Para ello busca parecidos entre la realidad y la posible implementación de objetos del programa que simulen el funcionamiento de los objetos reales.

Los seres humanos no pensamos en las cosas como un conjunto de cosas menores; por ejemplo, no vemos un cuerpo humano como un conjunto de células.

Los humanos entendemos la realidad como objetos con comportamientos bien definidos. No necesitamos conocer los detalles de porqué ni cómo funcionan las cosas; simplemente

Nivel: Quinto año

solicitamos determinadas acciones en espera de una respuesta; cuando una persona desea desplazarse, su cuerpo le responde comenzando a caminar.

Pero la abstracción humana se gestiona de una manera jerárquica, dividiendo sucesivamente sistemas complejos en conjuntos de subsistemas, para así entender más fácilmente la realidad. Esta es la forma de pensar que la orientación a objeto intenta cubrir.

b.) Principio de Encapsulamiento

El encapsulamiento permite a los objetos elegir qué información es publicada y qué información es ocultada al resto de los objetos. Para ello los objetos suelen presentar sus métodos como interfaces públicas y sus atributos como datos privados e inaccesibles desde otros objetos.

Para permitir que otros objetos consulten o modifiquen los atributos de los objetos, las clases suelen presentar métodos de acceso. De esta manera el acceso a los datos de los objetos es controlado por el programador, evitando efectos laterales no deseados.

Con el encapsulado de los datos se consigue que las personas que utilicen un objeto sólo tengan que comprender su interfaz, olvidándose de cómo está implementada, y en definitiva, reduciendo la complejidad de utilización.

c.) Principio de Modularidad

Mediante la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.

Esta fragmentación disminuye el grado de dificultad del problema al que da respuesta el programa, pues se afronta el problema como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

d.) Principio de Jerarquía

La mayoría de nosotros ve de manera natural nuestro mundo como objetos que se relacionan entre sí de una manera jerárquica. Por ejemplo, un perro es un mamífero, y los mamíferos son animales, y los animales seres vivos...

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Del mismo modo, las distintas clases de un programa se organizan mediante la jerarquía. La representación de dicha organización da lugar a los denominados árboles de herencia:

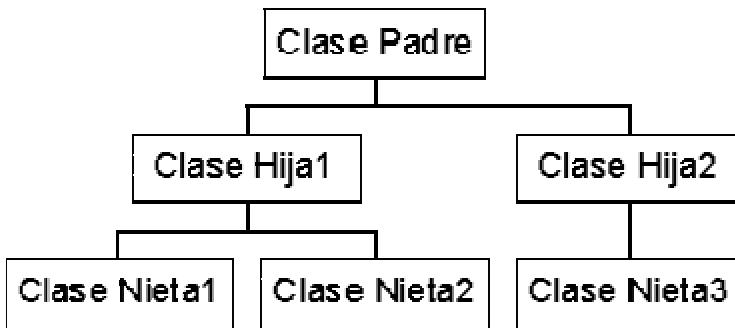


Imagen 1: Ejemplo de árbol de herencia

Mediante la herencia una clase hija puede tomar determinadas propiedades de una clase padre. Así se simplifican los diseños y se evita la duplicación de código al no tener que volver a codificar métodos ya implementados.

Al acto de tomar propiedades de una clase padre se denomina heredar.

e.) Principio del Paso de Mensajes

Mediante el denominado paso de mensajes, un objeto puede solicitar de otro objeto que realice una acción determinada o que modifique su estado. El paso de mensajes se suele implementar como llamadas a los métodos de otros objetos.

Desde el punto de vista de la programación estructurada, esto correspondería con la llamada a funciones.

f.) Principio de Polimorfismo

Polimorfismo quiere decir "un objeto y muchas formas". Esta propiedad permite que un objeto presente diferentes comportamientos en función del contexto en que se encuentre. Por ejemplo un método puede presentar diferentes implementaciones en función de los argumentos que recibe, recibir diferentes números de parámetros para realizar una misma operación, y realizar diferentes acciones dependiendo del nivel de abstracción en que sea llamado.

E. Relaciones entre objetos

Durante la ejecución de un programa, los diversos objetos que lo componen han de interactuar entre sí para lograr una serie de objetivos comunes.

Existen varios tipos de relaciones que pueden unir a los diferentes objetos, pero entre ellas destacan las relaciones de: asociación, todo/parte, y generalización/especialización.

a.) Relaciones de Asociación

Serían relaciones generales, en las que un objeto realiza llamadas a los servicios (métodos) de otro, interactuando de esta forma con él.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Representan las relaciones con menos riqueza semántica.

b.) Relaciones de Todo/Parte

Muchas veces una determinada entidad existe como conjunción de otras entidades, como un conglomerado de ellas. La orientación al objeto recoge este tipo de relaciones como dos conceptos; la agregación y la composición.

En este tipo de relaciones un objeto componente se integra en un objeto compuesto. La diferencia entre agregación y composición es que mientras que la composición se entiende que dura durante toda la vida del objeto componedor, en la agregación no tiene por qué ser así.

Esto se puede implementar como un objeto (objeto compuesto) que cuenta entre sus atributos con otro objeto distinto (objeto componente).

c.) Relaciones de Generalización/Especialización

A veces sucede que dos clases tienen muchas de sus partes en común, lo que normalmente se abstrae en la creación de una tercera clase (padre de las dos) que reúne todas sus características comunes.

El ejemplo más extendido de este tipo de relaciones es la herencia, propiedad por la que una clase (clase hija) recoge aquellos métodos y atributos que una segunda clase (clase padre) ha especificado como "heredables".

Este tipo de relaciones es característico de la programación orientada a objetos.

En realidad, la generalización y la especialización son diferentes perspectivas del mismo concepto, la generalización es una perspectiva ascendente (botón-up), mientras que la especialización es una perspectiva descendente (top-down).

Objetos y clases en C#

Vamos con los detalles. Las clases en C# se definen de forma parecida a los registros (struct), sólo que ahora también incluirán funciones. Así, la clase "Puerta" que mencionábamos antes se podría declarar así:

```
public class Puerta
{
    int ancho; // Ancho en centímetros
    int alto; // Alto en centímetros
    int color; // Color en formato RGB
    bool abierta; // Abierta o cerrada
```

```
public void Abrir()
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

{

abierta = true;

}

public void Cerrar()

{

abierta = false;

}

public void MostrarEstado()

{

Console.WriteLine("Ancho: {0}", ancho);

Console.WriteLine("Alto: {0}", alto);

Console.WriteLine("Color: {0}", color);

Console.WriteLine("Abierta: {0}", abierta);

}

} // Final de la clase Puerta

Como se puede observar, los objetos de la clase "Puerta" tendrán un ancho, un alto, un color, y un estado (abierta o no abierta), y además se podrán abrir o cerrar (y además, nos pueden "mostrar su estado, para comprobar que todo funciona correctamente").

Para declarar estos objetos que pertenecen a la clase "Puerta", usaremos la palabra "new", igual que hacíamos con los "arrays":

Puerta p = new Puerta();

p.Abrir();

p.MostrarEstado();

Vamos a completar un programa de prueba que use un objeto de esta clase (una "Puerta"):

/*-----*/

/* Ejemplo en C# nº 59: */

/* ejemplo59.cs */

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
/*
 * Primer ejemplo de clases *
 */
/* Introduccion a C#, */
/* Nacho Cabanes */
/*-----*/
using System;
public class Puerta
{
    int ancho; // Ancho en centímetros
    int alto; // Alto en centímetros
    int color; // Color en formato RGB
    bool abierta; // Abierta o cerrada

    public void Abrir()
    {
        abierta = true;
    }

    public void Cerrar()
    {
        abierta = false;
    }

    public void MostrarEstado()
    {
        Console.WriteLine("Ancho: {0}", ancho);
        Console.WriteLine("Alto: {0}", alto);
        Console.WriteLine("Color: {0}", color);
    }
}
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
Console.WriteLine("Abierta: {0}", abierta);
```

```
}
```

```
} // Final de la clase Puerta
```

```
public class Ejemplo59
```

```
{
```

```
public static void Main()
```

```
{
```

```
Puerta p = new Puerta();
```

```
Console.WriteLine("Valores iniciales...");
```

```
p.MostrarEstado();
```

```
Console.WriteLine("\nVamos a abrir...");
```

```
p.Abrir();
```

```
p.MostrarEstado();
```

```
}
```

```
}
```

Este fuente ya no contiene una única clase (class), como todos nuestros ejemplos anteriores, sino dos clases distintas:

- La clase "Puerta", que son los nuevos objetos con los que vamos a practicar.
- La clase "Ejemplo59", que representa a nuestra aplicación.

El resultado de ese programa es el siguiente:

Valores iniciales...

Ancho: 0

Alto: 0

Color: 0

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Abierta: False

Vamos a abrir...

Ancho: 0

Alto: 0

Color: 0

Abierta: True

Se puede ver que en C#, al contrario que en otros lenguajes, las variables que forman parte de una clase (los "atributos") tienen un valor inicial predefinido: 0 para los números, una cadena vacía para las cadenas de texto, o "False" para los datos booleanos.

Vemos también que se accede a los métodos y a los datos precediendo el nombre de cada uno por el nombre de la variable y por un punto, como hacíamos con los registros (struct). Aun así, en nuestro caso no podemos hacer directamente "p.abierta = true", por dos motivos:

- El atributo "abierta" no tiene delante la palabra "public"; por lo que no es público, sino privado, y no será accesible desde otras clases (en nuestro caso, desde Ejemplo59).
- Los puristas de la Programación Orientada a Objetos recomiendan que no se acceda directamente a los atributos, sino que siempre se modifiquen usando métodos auxiliares (por ejemplo, nuestro "Abrir"), y que se lea su valor también usando una función. Esto es lo que se conoce como "ocultación de datos". Supondrá ventajas como que podremos cambiar los detalles internos de nuestra clase sin que afecte a su uso.

Normalmente, como forma de ocultar datos, crearemos funciones auxiliares GetXXX y SetXXX que permitan acceder al valor de los atributos (en C# existe una forma alternativa de hacerlo, usando "propiedades", que veremos más adelante):

```
public int GetAncho()
{
    return ancho;
}

public void SetAncho(int nuevoValor)
{
    ancho = nuevoValor;
}
```

Nivel: Quinto año

La herencia. Visibilidad

Vamos a ver ahora cómo definir una nueva clase de objetos a partir de otra ya existente. Por ejemplo, vamos a crear una clase "Portón" a partir de la clase "Puerta". Un portón tendrá las mismas características que una puerta (ancho, alto, color, abierto o no), pero además se podrá bloquear, lo que supondrá un nuevo atributo y nuevos métodos para bloquear y desbloquear:

```
public class Portón: Puerta
```

```
{
```

```
    bool bloqueada;
```

```
    public void Bloquear()
```

```
{
```

```
    bloqueada = true;
```

```
}
```

```
    public void Desbloquear()
```

```
{
```

```
    bloqueada = false;
```

```
}
```

```
}
```

Con "public class Portón: Puerta" indicamos que Portón debe "heredar" todo lo que ya habíamos definido para Puerta. Por eso, no hace falta indicar nuevamente que un Portón tendrá un cierto ancho, o un color, o que se puede abrir: todo eso lo tiene por ser un "descendiente" de Puerta.

No tenemos por qué heredar todo; también podemos "redefinir" algo que ya existía. Por ejemplo, nos puede interesar que "MostrarEstado" ahora nos diga también si la puerta está bloqueada. Para eso, basta con volverlo a declarar y añadir la palabra "new" para indicar al compilador de C# que sabemos que ya existe ese método y que sabemos seguro que lo queremos redefinir:

```
    public new void MostrarEstado()
```

```
{
```

```
    Console.WriteLine("Ancho: {0}", ancho);
```

```
    Console.WriteLine("Alto: {0}", alto);
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
Console.WriteLine("Color: {0}", color);
Console.WriteLine("Abierta: {0}", abierta);
Console.WriteLine("Bloqueada: {0}", bloqueada);
}
```

Aun así, esto todavía no funciona: los atributos de una Puerta, como el "ancho" y el "alto" estaban declarados como "privados" (es lo que se considera si no decimos los contrario), por lo que no son accesibles desde ninguna otra clase, ni siquiera desde Portón.

La solución más razonable no es declararlos como "public", porque no queremos que sean accesibles desde cualquier sitio. Sólo queríamos que esos datos estuvieran disponibles para todos los tipos de Puerta, incluyendo sus "descendientes", como un Portón. Esto se puede conseguir usando otro método de acceso: "protected". Todo lo que declaremos como

"protected" será accesible por las clases derivadas de la actual, pero por nadie más:

```
public class Puerta
{
    protected int ancho; // Ancho en centímetros
    protected int alto; // Alto en centímetros
    protected int color; // Color en formato RGB
    protected bool abierta; // Abierta o cerrada
```

```
public void Abrir()
```

```
...
```

(Si quisiéramos dejar claro que algún elemento de una clase debe ser totalmente privado, podemos usar la palabra "private", en vez de "public" o "protected").

Un fuente completo que declarase la clase Puerta, la clase Portón a partir de ella, y que además contuviese un pequeño "Main" de prueba podría ser:

```
/* Ejemplo en C# nº 60: */
```

```
/* ejemplo60.cs */
```

```
/* */
```

```
/* Segundo ejemplo de */
```

```
/* clases: herencia */
```

```
/* */
```

CEDES DON BOSCO

Nivel: Quinto año

```
/* Introducción a C#, */  
/* Nacho Cabanes */  
/*-----*/  
  
using System;  
// -----  
  
public class Puerta  
{  
  
    protected int ancho; // Ancho en centímetros  
    protected int alto; // Alto en centímetros  
    protected int color; // Color en formato RGB  
    protected bool abierta; // Abierta o cerrada  
  
  
    public void Abrir()  
{  
        abierta = true;  
    }  
  
  
    public void Cerrar()  
{  
        abierta = false;  
    }  
  
    public void MostrarEstado()  
{  
  
        Console.WriteLine("Ancho: {0}", ancho);  
        Console.WriteLine("Alto: {0}", alto);  
        Console.WriteLine("Color: {0}", color);  
        Console.WriteLine("Abierta: {0}", abierta);  
    }  
}
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
} // Final de la clase Puerta
// -----
public class Portón: Puerta
{
    bool bloqueada;

    public void Bloquear()
    {
        bloqueada = true;
    }

    public void Desbloquear()
    {
        bloqueada = false;
    }

    public new void MostrarEstado()
    {
        Console.WriteLine("Ancho: {0}", ancho);
        Console.WriteLine("Alto: {0}", alto);
        Console.WriteLine("Color: {0}", color);
        Console.WriteLine("Abierta: {0}", abierta);
        Console.WriteLine("Bloqueada: {0}", bloqueada);
    }
}

} // Final de la clase Portón
// -----
public class Ejemplo60
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

{

```
public static void Main()
{
    Portón p = new Portón();

    Console.WriteLine("Valores iniciales...");
    p.MostrarEstado();

    Console.WriteLine("\nVamos a bloquear...");
    p.Bloquear();
    p.MostrarEstado();

    Console.WriteLine("\nVamos a desbloquear y a abrir...");
    p.Desbloquear();
    p.Abrir();
    p.MostrarEstado();
}

}
```

Constructores y destructores.

Hemos visto que al declarar una clase, se dan valores por defecto para los atributos. Por ejemplo, para un número entero, se le da el valor 0. Pero puede ocurrir que nosotros deseemos dar valores iniciales que no sean cero. Esto se puede conseguir declarando un "constructor" para la clase.

Un constructor es una función especial, que se pone en marcha cuando se crea un objeto de una clase, y se suele usar para dar esos valores iniciales, para reservar memoria si fuera necesario, etc.

Se declara usando el mismo nombre que el de la clase, y sin ningún tipo de retorno. Por ejemplo, un "constructor" para la clase Puerta que le diera los valores iniciales de 100 para el ancho, 200 para el alto, etc., podría ser así:

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
public Puerta()  
{  
    ancho = 100;  
    alto = 200;  
    color = 0xFFFFFFFF;  
    abierta = false;  
}
```

Podemos tener más de un constructor, cada uno con distintos parámetros. Por ejemplo, puede haber otro constructor que nos permita indicar el ancho y el alto:

```
public Puerta(int an, int al)  
{  
    ancho = an;  
    alto = al;  
    color = 0xFFFFFFFF;  

```

Ahora, si declaramos un objeto de la clase puerta con "Puerta p = new Puerta();" tendrá de ancho 100 y de alto 200, mientras que si lo declaramos con "Puerta p2 = new Puerta(90,220);" tendrá 90 como ancho y 220 como alto.

Un programa de ejemplo que usara estos dos constructores para crear dos puertas con características iniciales distintas podría ser:

```
/*-----*/  
/* Ejemplo en C# nº 61: */  
/* ejemplo61.cs */  
/* */  
/* Tercer ejemplo de clases */  

```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
/* Nacho Cabanes */  
/*-----*/  
using System;  
public class Puerta  
{  
    int ancho; // Ancho en centímetros  
    int alto; // Alto en centímetros  
    int color; // Color en formato RGB  
    bool abierta; // Abierta o cerrada  
  
    public Puerta()  
    {  
        ancho = 100;  
        alto = 200;  
        color = 0xFFFFFFFF;  
        abierta = false;  
    }  
    public void Abrir()  
    {  
        abierta = true;  
    }  
  
    public void Cerrar()  
    {  
        abierta = false;  
    }  
    public void MostrarEstado()  
    {
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
Console.WriteLine("Ancho: {0}", ancho);
Console.WriteLine("Alto: {0}", alto);
Console.WriteLine("Color: {0}", color);
Console.WriteLine("Abierta: {0}", abierta);
}

} // Final de la clase Puerta

public class Ejemplo61
{

    public static void Main()
    {
        Puerta p = new Puerta();
        Puerta p2 = new Puerta(90,220);

        Console.WriteLine("Valores iniciales...");
        p.MostrarEstado();

        Console.WriteLine("\nVamos a abrir...");
        p.Abrir();
        p.MostrarEstado();

        Console.WriteLine("Para la segunda puerta...");
        p2.MostrarEstado();
    }
}
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Nota: al igual que existen los "constructores", también podemos indicar un "destructor" para una clase, que se encargue de liberar la memoria que pudiéramos haber reservado en nuestra clase (no es nuestro caso, porque aún no sabemos manejar memoria dinámica) o para cerrar ficheros abiertos (que tampoco sabemos).

Un "destructor" se llama igual que la clase, pero precedido por el símbolo "~", no tiene tipo de retorno, y no necesita ser "public", como ocurre en este ejemplo:

```
~Puerta()
```

```
{
```

```
// Liberar memoria
```

```
// Cerrar ficheros
```

```
}
```

```
public Puerta(int an, int al)
```

```
{
```

```
ancho = an;
```

```
alto = al;
```

```
color = 0xFFFFFFFF;
```

```
abierta = false;
```

```
}
```

Polimorfismo y sobrecarga

Esos dos constructores "Puerta()" y "Puerta(int ancho, int alto)", que se llaman igual pero reciben distintos parámetros, y se comportan de forma que puede ser distinta, son ejemplos de "polimorfismo" (funciones que tienen el mismo nombre, pero distintos parámetros, y que quizás no se comporten de igual forma).

Un concepto muy relacionado con el polimorfismo es el de "sobrecarga": dos funciones están sobrecargadas cuando se llaman igual, reciben el mismo número de parámetros, pero se aplican a objetos distintos, así:

```
puerta.Abrir();
```

```
libro.Abrir();
```

En este caso, la función "Abrir" está sobrecargada: se usa tanto para referirnos a abrir un libro como para abrir una puerta. Se trata de dos acciones que no son exactamente iguales, que se aplican a objetos distintos, pero que se llaman igual.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Prácticas generales de orientación a objetos

Utilizando Visual studio 2013, desarrolle las clases necesarias para un sistema de matrícula, de un colegio técnico X:

Deberá implementar:

1. Polimorfismo
2. Encapsulamiento
3. Modularidad
4. Herencia
5. Relaciones de asociación

Sexto proyecto, desarrolle las clases necesarias para el sistema que utilice la base de datos de los ejercicios tres, cuatro y cinco, utilizando todos los conceptos de orientación a objetos, proyecto individual.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Aplicaciones web

Uno de los componentes más prácticos que brinda la red Internet, es el servicio de HTTP, mediante el cual por medio de un navegador, los clientes pueden acceder a una computadora con IP publica y visualizar lo que esta computadora deseé mostrarles en formato HTML.

Los tiempos van cambiando y la tecnología avanza siempre sobre las bases de tecnologías antiguas para lograr algo más complejo pero de más utilidad, el formato HTML, brinda un marco de reglas estándar para que navegadores lo puedan interpretar correctamente y mostrar en pantalla las formas, colores, letras y demás componentes que se desean mostrar, esto funcionó muy bien por algún tiempo y sin duda fue toda una revolución.

Las empresas y personas podían tener información que el usuario podía consultar libremente y hasta en algún grado interactuar por medio de botones que mostraban nueva información, sin embargo esto muy pronto resultó insuficiente.

Desde un principio se notó que esta tecnología de interacción por medio de internet era el futuro de la comunicación, pero se antojó rápidamente muy plana y estática, y sin duda tenía la oportunidad de crecer de forma inaudita, pronto algunas casas desarrolladoras, generaron estándares diferentes para interpretar o reinterpretar el HTML, y darle funcionalidades diversas, fue así como tecnologías como “flash”, “JavaScript”, y otras empezaron a ser el nuevo estándar.

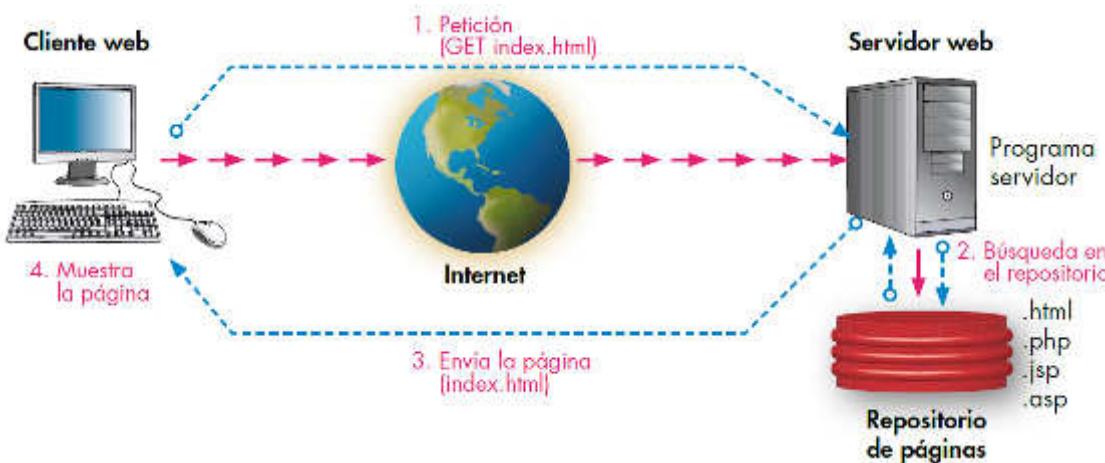
Las páginas web evolucionaron a aplicaciones web, donde la máxima era la interacción con el usuario; cómo se logra esto, respondiendo a las acciones del usuario con un nuevo código HTML, así se logra dar la impresión de interacción entre los componentes, para lograr esto es necesario que cada acción o al menos la mayoría de acciones del usuario sean registradas y enviadas en tiempo real al servidor o la computadora con la IP publica, y está según la acción de usuario responda con el código HTML correspondiente.

Con este último punto se pudo elaborar poderosas aplicaciones web, aplicaciones que tenían bases de datos, lógica de negocio y componentes personalizados, ya que el servidor podía tener acceso a todos los componentes y enviarle HTML puro y personalizado al cliente, para el cual todo este proceso era transparente, al menos mientras la conexión de internet fuese lo suficientemente rápida.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



Si reparamos un momento en este esquema rápidamente llegamos a la descubrir su mayor problema, si cada vez que el usuario interactúa con la página se realiza el proceso de la imagen anterior, se pierde la sensación de fluidez. La ventaja es que no todas las interacciones deben necesariamente hacer este “roundtrip” como se le llama a la acción de pedir información al servidor, algunas de las interacciones pueden ser locales, por ejemplo mostrar la hora del sistema, moverse entre los componentes de la página, redimensionar una imagen, etc.

Con toda la explicación en mente, no está demás comentar que surgieron múltiples tecnologías para lograr esta comunicación fluida entre cliente y servidor vía web, a estas herramientas les llamaremos servidores web, entre los que podemos enumerar: IIS de Microsoft, Apache de origen libre, para nuestros efectos trabajaremos con IIS de Microsoft, los cuales generaron un marco de desarrollo llamado ASP.Net, bajo el cual el servidor puede interpretar las páginas con extensión .aspx(en lugar de HTML), en estas páginas aspx podemos utilizar cualquiera de los lenguajes de programación de Microsoft: visual Basic o c#, para programar con el framework que Microsoft nos ofrece, podemos utilizar la mayoría de utilidades que nos ha ofrecido hasta ahora con “Windows form”, obviamente hay algunas diferencias inherentes al paradigma web, las cuales vamos a enumerar y describir a continuación:

Master Pages

Las páginas maestras, tienen consigo el concepto de herencia intrínseco, se trata de una página aspx, en la cual podemos poner tanto HTML como componentes de ASP (que explicaremos más adelante) la única diferencia es que podemos poner un componente único para este tipo de páginas llamado “ContentPlaceContainer”, como podrán inferir de esta página maestra podemos heredar otras páginas, páginas estándares aspx, las cuales tendrán todos los componentes HTML y asp de la página maestra, estos componentes en la página hija(como llamaremos a la página que heredamos de ahora en adelante) no pueden ser modificados desde esta página, la única sección de la página donde podremos modificar o añadir componentes HTML y asp en la página hija será donde este el “ContentPlaceContainer”.

Este concepto nos permite crear formatos estándares para todo nuestro sitio web, que en general se componen de un “Banner”, el cuerpo de la página y un pie de página, en donde generalmente en el cuerpo de la página se pone el “ContentPlaceContainer”.

Informática en Desarrollo

CEDES DON BOSCO

Desarrollo
Sustentable

Nivel: Quinto año

Componentes Asp

Microsoft brinda con su framework, una serie de componentes ya desarrollados, que en realidad son clases con sus atributos y eventos, estos componentes se pueden visualizar en el “Toolbox”, ubicado generalmente en la parte izquierda de la pantalla.

Entre los componentes más comunes se tienen los botones, botones con imágenes, imágenes, “Dropdownlist”, Cajas de texto, “Label”, etc.

Puede ver todos los componentes, arrastrarlos a su área de trabajo y visualizarlos en el diseño.

Diseño y código

El framework en Visual studio nos permite al crear un nuevo formulario web visualizar dos archivos: uno con el diseño(HTML y ASP) y otra con el código de la misma(C#), el archivo con la extensión aspx es el que llamaremos diseño y el archivo cs, es el del código, al darle doble clic al formulario diseño podremos visualizar en el área de trabajo un bosquejo de la página en HTML, en la parte inferior del área de trabajo existen botones que nos permiten trabajar ya sea en diseño o en el código del diseño(no debe confundirse con el código C# que es otra parte de la página), si entramos a la pestaña código, veremos el código HTML que se corresponde con el diseño, y si tenemos algún componente asp en el diseño, podemos darnos cuenta que el mismo esta embebido en el HTML como una etiqueta “<asp>”, en realidad todas estas etiquetas el servidor las transformara a su equivalente de un formulario HTML.

Eventos y propiedades

Como todo objeto, los de asp tienen propiedades y eventos, las propiedades de cada objeto se visualizan al seleccionar el objeto ya en el diseño y generalmente se visualizan en la parte inferior derecha de la pantalla, se pueden modificar diversas propiedades inherentes al objetos, algunas de las cuales se repiten en todos, como color de fondo, color de borde, texto, visibilidad, etc.

Sin embargo cada objeto tiene propiedades diferentes propias de su función, por ejemplo un image, tiene una propiedad llamada imageurl, para especificar la dirección de la imagen.

Podemos establecer estas propiedades en el diseño y también se pueden modificar desde el código (C#) cuando algún evento así lo amerite, por ejemplo, al presionar un botón, podemos programar que la propiedad backcolor de ese botón u otro cambie.

En la misma ventana de propiedades, hay un botón con forma de trueno, que representa los eventos que tiene el objeto seleccionado



Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

¿Qué es un evento?

Un evento es un mensaje (o notificación) que lanza un objeto de una clase determinada cuando algo ha ocurrido. El ejemplo más clásico y fácil de entender es cuando el usuario pulsa con el ratón en un botón de un formulario. Esa acción produce, entre otros, el evento Clic del botón en el que se ha pulsado. De esa forma, el botón notifica que esa acción ha ocurrido, y ya es cuestión nuestra que hagamos algo cuando eso ocurra. Si estamos interesados en interceptar ese evento tendremos que comunicárselo a la clase que define el botón; si no lo estamos, simplemente no es necesario que indiquemos nada.

Como es de suponer, los eventos se pueden definir en cualquier clase, y no solo en clases que tengan algún tipo de interacción con el usuario, aunque lo más habitual es que precisamente se usen con clases que forman parte de la interfaz gráfica que se le presenta al usuario de una aplicación. De esa forma podremos saber que algo está ocurriendo y en qué control, de forma que sepamos en todo momento lo que el usuario quiere hacer o lo que está haciendo.

Añadir “manejadores” de evento

Cuando estamos trabajando con formularios y controles, la asociación entre un evento y el método que usaremos para interceptarlo lo podemos hacer de varias formas.

Para “ligar” un evento con un método (y crearlo si no existe), debemos seleccionar el control que define el evento que queremos usar y en la ventana de propiedades seleccionar “Eventos” (el botón con la imagen de un rayo amarillo). De esa forma tendremos una lista de los eventos definidos por ese control (o al menos de los eventos que el creador del control haya decidido que se muestren en esa ventana de propiedades). En la figura 3 podemos ver algunos de los eventos de un formulario.

Como vemos en la figura 3, si el evento está asociado a un método, se muestra el nombre del mismo; si no hay ninguna asociación, el campo se muestra en blanco. Para crear un nuevo método y asociarlo al evento, simplemente tendremos que hacer una doble pulsación en la caja de textos en blanco. Pero si lo que queremos es usar uno de los métodos existentes, podemos seleccionar el método de la lista desplegable; en esa lista solo se mostrarán los métodos que tengan la misma firma del delegado asociado con ese evento, tal como podemos ver en la figura 4, donde el delegado del evento Clic tiene la misma firma que los mostrados en dicha lista, lo que nos permite usar cualquiera de los existentes o bien crear uno nuevo.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año



Figura 3. Desde la ventana de propiedades podemos seleccionar el evento que queremos interceptar

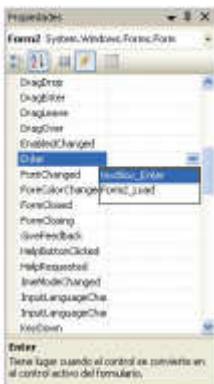


Figura 4. Podemos asociar un evento a un método existente, el cual seleccionamos de una lista desplegable

Lo único que no nos permite el diseñador de formularios es asociar varios métodos de eventos a un mismo evento. En ese caso, tendremos que hacer manualmente esa asociación. La pregunta puede ser ¿dónde hacer esa asociación? La respuesta es: donde queramos, pero siempre que ese código no se repita más de una vez, con idea de no agregar más veces de las necesarias el método que intercepta el evento. Por tanto, lo más recomendable es que escribamos ese código en el constructor del formulario, pero justo después de la llamada al método InitializeComponent.

Esto también es válido para cuando decidimos asociar manualmente los eventos con los métodos que recibirán la notificación.

Ajax

Con las herramientas Ajax, Microsoft ha tratado de ofrecer un ambiente de desarrollo integrado con JavaScript, la finalidad es ofrecer al desarrollador una serie de herramientas para desarrollo a nivel de cliente, utilizando JavaScript un lenguaje bastante popular para los desarrolladores y diseñadores de páginas web pero no muy utilizado por desarrolladores de aplicaciones.

Con JavaScript se pueden desarrollar efectos muy útiles y llamativos con los objetos de una página pero a nivel del clientes, es decir sus eventos no tienen que ir al servidor de datos a preguntar qué hacer ante tal evento, si no que la página ya tiene precargado que acción se debe realizar ante tal evento, JavaScript por si solo es un lenguaje, pero para efectos de este curso se deja al lector la tarea de investigar el funcionamiento del mismo, lo importante es rescatar que Ajax internamente utiliza JavaScript como motor para lograr sus efectos, y lo enmascara para que el desarrollador de aplicaciones pueda usarlo de manera más sencilla.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Las utilidades Ajax están condensadas en un archivo .dll, que se puede descargar de internet, ya sea en la página oficial de Microsoft o cualquier otro sitio que lo ofrezca, una vez descargado presionando el botón derecho del ratón sobre el toolbox, se escoge Elegir elementos, en el botón examinar se busca el .dll descargado y lo escogen, al finalizar presionan aceptar para cargar los elementos, los cuales se verán como nuevos objetos elegibles en la barra de herramientas.

El primer paso para utilizar cualquier elemento de Ajax, es cargar un elemento primario llamado "Scriptmanager", el cual debe cargarse ya sea en la Página maestra, o en la página hija pero no en ambos, se carga seleccionándolo de la barra de herramientas y arrastrándolo a la página.

Una vez con el scriptmanager en la página se pueden arrastrar cualquiera de los demás elementos, los más utilizados son: Tabcontainer, Accordeon, Ajaxfileupload, y muchos otros que se pueden arrastrar y colocar en la página donde se desee, en múltiples páginas de internet se puede encontrar documentación de cada objeto.

Sesiones y variables globales

En múltiples ocasiones hemos utilizado variables en nuestros códigos, son muy útiles nos permiten almacenar información que utilizaremos más adelante en un objeto con un nombre acorde o lo que guardamos en él, las variables como ya hemos visto en la práctica pueden ser objetos de cualquier clase: int, string, clases creadas por nosotros, etc., pues bien qué pasaría si ocupamos que al presionar un botón necesitamos guardar el dato de un textbox en un botón, pues muy sencillo, declaramos una variable string; string variable; y le asignamos el texto del texto; variable = texto.text; excelente, ahora necesitamos que al hacer clic en una imagen aparezca ese texto en la pantalla, pero un momento, esto es otro evento diferente al evento clic del botón anterior, y si intentamos utilizar la variable que declaramos anteriormente en este nuevo evento, pues nos dice que no existe, para resolver ese conflicto es que existen las variables globales.

Variables globales

Para declararlas debemos usar la palabra reservada Public static, y se deben declarar justo después de la declaración de la página al inicio del código de la misma:

```
public partial class FRM_becas : System.Web.UI.Page
{
    public static List<String> _lista_de_docs = new List<string>();
    ...
}
```

En este ejemplo estamos declarando una variable global del tipo lista de cadena, llamada _lista_de_docs, al declararla en esta parte del código, puede ser accesada, leída y reescrita en cualquier evento de la página, obviamente se puede declarar cualquier tipo de variable.

Estas variables globales son muy útiles al compartir información entre eventos dentro de una misma página, pero existe otra situación que podría darse, si ocupamos que la información pase de un formulario a otro, para esto debemos usar las sesiones.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Sesiones

Las variables de sesión son secciones de memoria en el servidor, tienen un tiempo definido y que puede ser alterado en el archivo de configuración de la aplicación web, y como su nombre implica está muy ligado a la información del cliente que accesa la misma, en resumen el servidor toma varios parámetros del cliente, IP, navegador, dirección MAC y con esa información crea un perfil del usuario conectado, todas las sesiones estarán ligadas a ese perfil, por tanto cada usuario que ingrese a la aplicación tendrá su sesión independiente de las demás sesiones de los otros usuarios de la aplicación, no se puede compartir información de sesiones de diferentes perfiles.

```
Clases_entidad.USUARIO CLASE_A_DEVOLVER = new Clases_entidad.USUARIO();
CLASE_A_DEVOLVER.ID = 0;
CLASE_A_DEVOLVER = Seguridad.Clase_inicia_sesion.CORROBORA_LOGIN(Txt_nom_usu.Text, Txt_contrasena.Text);
if (CLASE_A_DEVOLVER.ID != 0)
{
    if ((CLASE_A_DEVOLVER.ROL == 1) || (CLASE_A_DEVOLVER.ROL == 2) || (CLASE_A_DEVOLVER.ROL == 3) || (CLASE_A_DEVOLVER.ROL == 4))
    {
        Session["usuario"] = CLASE_A_DEVOLVER;
        Response.Redirect("~/Paginas/FRM_busqueda_principal.aspx");
    }
}
```

En el ejemplo anterior se declara una clase tipo USUARIO, se rellena con una función y posterior se asigna en una variable tipo sesión, la forma de diferenciar las variables de sesión es por el nombre, el cual está entre paréntesis cuadrados y comillas, se puede almacenar en la sesión cualquier tipo de variable.

En el siguiente formulario donde haya que usar la variable sesión, se debe primero verificar que exista y que tenga algo asignado.

```
if (Session["usuario"] != null)
{
    ...
}
```

Una vez que sabemos que la variable tiene algún objeto asignado, debemos convertir la sesión a ese tipo de objeto.

```
((Clases_entidad.USUARIO)Session["usuario"])
```

Justo antes de la sesión y entre paréntesis ponemos el tipo de objeto al que deseamos convertirlo, y en la misma línea podemos asignárselo a un objeto local del mismo tipo.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Datagrid (grillas)

Una de las utilidades más usadas en cualquier aplicación de información son las grillas, que en general son colecciones de datos presentadas en filas y columnas (como una matriz) la misma puede ser rellenada por un objeto especial de datos llamado DataTable.

```
System.Data.DataTable dt = new System.Data.DataTable();
```

Una vez definido el DataTable, se le deben especificar las columnas que deseamos que tenga la grilla.

```
dt.Columns.Add("Identificación");
```

Dentro del Datatable debemos agregarle filas, cada fila que queremos debe agregarse de manera manual, lo primero es declarar un objeto tipo fila

```
System.Data DataRow dr;
```

Para cada fila se debe agregar la misma, de esta manera

```
dr = dt.NewRow();
```

Rellenamos cada columna que hayamos declarado al principio con el dato respectivo utilizando índices que empiezan obviamente en 0

```
dr[0] = item.IDENTIFICACION.IDENTIFICACION.ToString();
```

Se llenan todas las columnas.

Al final se agrega la fila al datatable

```
dt.Rows.Add(dr);
```

Como último paso le agregamos el datatable a la grilla, obviamente la grilla ya debe estar en nuestra página, para agregarla se selecciona en el cuadro de herramientas y arrastrándola al diseño de la página, le ponemos el nombre representativo que deseemos.

En el código después de declarar y llenar el datatable colocamos solamente dos líneas de código.

```
_GRILLA_A_RELLENAR.DataSource = dt;  
_GRILLA_A_RELLENAR.DataBind();
```

Séptimo proyecto, desarrolla toda la interfaz gráfica de la aplicación de los proyectos anteriores, pero para una aplicación web, proyecto individual.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Conexión con base de datos

La mayoría de aplicaciones actuales interactúan con una base de datos, y parte de esta interacción se da a nivel del código mediante la aplicación, al definir la forma como nos conectamos a la base de datos e interactuamos con sus procedimientos almacenados.

Microsoft ha definido una serie de herramientas que permiten que esa conexión sea transparente y sencilla, utilizando solamente algunas funciones simples de objetos ya predefinidos.

Lo primero que haremos es crear una nueva clase, la cual va a contener toda la lógica de la conexión, esta nueva clase que le pondremos un nombre representativo que nos parezca, debe utilizar las librerías que mencionábamos en los párrafos anteriores:

```
using System.Data.OleDb;
```

Declaramos una serie de variables globales que vamos a utilizar a lo largo de la clase:

```
public OleDbConnection conexion;
    public OleDbCommand comando;
    String strcomando;
    String strconexion;
    OleDbTransaction transaccion;
    bool conecta;
    String xconecta;
```

Utilizamos una función para especificar la cadena de conexión, esta nos sirve para indicarle a la librería a qué tipo de base de datos nos vamos a conectar, el usuario la contraseña el servidor, entre otras:

```
public void parametro()
{
    strconexion = "Provider=SQLOLEDB;Data Source=sistemcr.com;Initial
Catalog=XXXXX;Persist Security Info=True;User
ID=pepito;Password=XXXXXX;Pooling=False";
}
```

Utilizamos otra función para inicializar la conexión, con la cual corroboramos que la cadena de conexión este bien y tengamos un canal de información

```
public bool inicializa()
{
    conexion = new OleDbConnection(strconexion);
    try
    {
        conexion.Open();
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
}
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

Una nueva función se encargara de añadir una nueva consulta a la base de datos, las mismas como se discutirá en clase, deben ser procedimientos validos de la base de datos.

```
public bool annadir_consulta(String _Consulta)
{
    comando = new OleDbCommand(_Consulta, conexion);
    return false;
}
```

Como ya se ha visto anteriormente todos los procedimientos almacenados tienen parámetros, por tanto haremos otra función para añadir parámetros a la consulta

```
public bool annadir_parametro(Object _PARAMETRO, Int16 _TIPO)
{
    OleDbParameter parametro;
    switch (_TIPO)
    {
        case 1:
            parametro = comando.Parameters.Add("@InputParm",
OleDbType.BigInt);
            parametro.Value = _PARAMETRO;
            break;
        case 2:

            parametro = comando.Parameters.Add("@InputParm",
OleDbType.VarChar, 2500);
            parametro.Value = _PARAMETRO;
            break;
        case 3:

            parametro = comando.Parameters.Add("@InputParm",
OleDbType.Decimal, 10);
            parametro.Value = _PARAMETRO;
            parametro.Precision = 10;
            parametro.Scale = 2;
            break;
        case 4:

            parametro = comando.Parameters.Add("@InputParm",
OleDbType.Date);
            parametro.Value = _PARAMETRO;
            break;
        case 5:

            parametro = comando.Parameters.Add("@InputParm",
OleDbType.VarBinary, ((byte[])_PARAMETRO).Length);
            parametro.Value = _PARAMETRO;
            break;
    }
    return false;
}
```

Con esta clase lista podemos probarla en cualquiera de los códigos de los formularios de nuestra aplicación.

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

En la parte del código donde vamos a hacer la transacción con la base de datos (consulta, inserción, actualización, etc.), declaramos un objeto de nuestro tipo conexión recién hecho

```
Conexion conx_detalles = new Conexion();
conx_detalles.parametro();
conx_detalles.inicializa();
```

Nótese que en las dos líneas anteriores de una vez puse la cadena de conexión e inicialice la misma.

Se declaran dos nuevos objetos, una cadena normal donde vamos a escribir la sentencia SQL a ejecutar y un contenedor SQL, que es una matriz por defecto que da el framework para guardar los resultados de la consulta.

```
string CONSULTA;
System.Data.OleDb.OleDbDataReader CONTENEDOR;
```

Pues bien, ahora en la cadena de conexión le asignamos la palabra reservada “EXEC” espacio y el nombre del procedimiento almacenado, espacio y un signo de pregunta “?” por cada parámetro que el procedimiento ocupe, si ocupa más de uno se dividen por comas “,”

```
CONSULTA = "EXEC INSERTA_DEDUCCIONES_MANUALES ?,?";
```

Ahora añadimos la consulta y los parámetros a la conexión:

```
conx_detalles.annadir_consulta(CONSULTA);
conx_detalles.annadir_parametro(_QUINCENA, 1);
conx_detalles.annadir_parametro(_MES, 1);
```

En este ejemplo específico tenemos dos parámetros que añadir, por tanto en la cadena tenemos dos signos de pregunta, y llamamos dos veces a la función añadir parámetros.

Ahora al contenedor que declaramos párrafos atrás le asignamos lo que nos devuelva la clase conexión en su función busca

```
CONTENEDOR = conx_detalles.busca();
```

Usamos un “While” para decir que mientras el contenedor tenga filas que leer, hagamos algo con esas filas

```
while (CONTENEDOR.Read())
{
    respuesta = CONTENEDOR[0].ToString();
}
```

Cerramos la conexión y el contenedor

```
conx_detalles.conexion.Close();
conx_detalles.conexion.Dispose();
```

Informática en Desarrollo

CEDES DON BOSCO

Nivel: Quinto año

```
CONTENEDOR.Close();
```

Dentro del while leemos las filas que devuelva la consulta, obviamente solamente los procedimientos que tienen consultas (select) devolverán filas, los de inserción, actualización o eliminación no devuelven ninguna, por lo tanto si sabemos que el procedimiento nos devolverá algo ponemos código dentro del while para recuperar esas filas, ya sea almacenarlas en una lista de clases global o llenar una única clase, como se puede notar en el ejemplo la variable contenedor se comporta como un vector donde la primera columna que devuelve es la 0, y así incrementándose según la columna que deseamos leer, si la consulta devuelve 7 columnas entonces podemos llegar hasta el índice 6, CONTENEDOR[6], para leer los datos.

Octavo y último proyecto, conecte correctamente la base de datos desarrollada en los proyectos tercero, cuarto y quinto, con la aplicación web creada en los proyectos sexto, y séptimo, de tal manera que la aplicación web esté terminada al 100%, proyecto individual.