FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

CURSOS INSTITUCIONALES

MÓDULO BÁSICO DE TECNOLÓGIAS DE LA INFORMACIÓN Y BIOESTADÍSTICA PARA EL ISSSTE

CURSO 2
Herramientas tecnológicas
CI 371

TEMA

Manual de diseño de bases de datos relacionales

28 DE OCTUBRE del 2002 al 15 de enero del 2003

Instructor: **Ing. Rodolfo González Maldonado**
Palacio de Minería
I S S S T E Octubre 2002

INDICE

FUNDAMENTOS

INTRODUCCION

El uso de la computadora en el campo informático se ha diferenciado por su grado de aprovechamiento; generalmente las empresas medianas y microempresas usan sistemas separados que hacen una proliferación de bases de datos, esto sobrecarga el manejo administrativo al mantener datos duplicados; en cambio las grandes empresas han hecho un soporte que incluye toda la empresa y además apoya a otras empresas con las que se relaciona.

En la actualidad cualquier negocio puede invertir lo suficiente para aprovechar la ventaja que ofrece un sistema de base de datos relacional (DBMS). En este caso los datos se centralizan en su almacenamiento con un programa que garantiza la seguridad al acceso y la fiabilidad en su operación; la información se distribuye por una red con acceso desde programas, pueden ser hechos en casi todos los lenguajes de alto nivel, desde la paquetería de oficina y consultas dinámicas hechas a la necesidad del usuario en el momento.

Es conveniente que nuestro sistema de educación tecnológica industrial sea el promotor del apuntalamiento informático de las empresas. La DGETI tiene la gran oportunidad de hacer sus sistemas de gestión educativa apoyados por un DBMS que le permitiría seguridad y gran flexibilidad en obtener datos para análisis de la situación de cada plantel, por especialidades del total de los planteles y en general con los casi 430 planteles que conforman la DGETI en este 1999.

Este curso se ha diseñado para hacer práctica de la metodología de diseño hasta llegar a la implantación en SQL. Al igual que el dominio de cualquier técnica al principio se tienen imperfecciones en los sistemas desarrollados, conforme se usan y luego se ajustan van cumpliendo cabalmente con todos los requerimientos.

He considerado necesario abordar como temas introductorios la teoría relacional, la organización de las empresas, el diseño de sistemas y por último diseño de la base de datos.

Conocer los conceptos de la teoría relacional permiten familiarizarse con los términos usados en las operaciones entre tablas y con esto explicar la actividad del sistema de manejo de bases de datos cuando se hacen las consultas, vistas y operaciones de afectación a uno o más registros de la base de datos.

Se ha avanzado en el soporte de planeación y control que requieren las empresas para cumplir la función de administración al grado de haber empresas con cero comunicación escrita. La función secretarial se ha suplido con una capacitación de todo el personal administrativo en el manejo de las herramientas computacionales de oficina y la

Por: Jorge Gutiérrez Sánchez

organización de un catálogo de formatos donde al ser llenado uno se registra con copia a las oficinas relacionadas mediante correo electrónico.

El diseño de sistemas, usado como una referencia de lo general a lo particular, prepara al diseñador de la base de datos a enfrentar la problemática de las aplicaciones computacionales en las empresas. El ambiente dinámico que las rodea y con el propósito de mantener y en lo posible superar la competencia en un mercado global obliga a reducir las diferencias, que son desventajas para los negocios, y deben ser compensadas con creatividad, organización y labor de equipo.

El diseño de bases de datos relacionales propone la técnica y tratará, en lo posible del tiempo y los recursos puestos a disposición para este curso, hacer la práctica suficiente para enfrentar los retos de una situación real.

Muchos de los sistemas computacionales que hoy existen en las organizaciones son susceptibles de mejorarse con el enfoque de sistemas que nos sustenta. Estas técnicas han sido utilizadas durante mucho tiempo en grandes empresas, con presupuestos muy altos para el área informática, hoy llegan a nuestras posibilidades por el gran abaratamiento del hardware que hace posible tener computadoras poderosas en escritorio, pudiendo correr los programas del sistema de manejo de las bases de datos relacionales.

Algunas versiones de DBMS libres de derecho para aplicaciones no comerciales se ofrecen en el sistema operativo Linux, esta sería una solución para muchos de nuestros centros educativos que por razones económicas no cuentan con el software necesario.

Agradezco la oportunidad de su atención y espero que este material sea base para un trabajo final que se pueda aprovechar en clase en las materias que se relacionan con la especialidad de computación.

El autor.
Nov. 1999.

TEORIA RELACIONAL

El modelo relacional define el almacenamiento en tablas, a las cuales se les llama ***relaciones***. Cada relación tiene un número definido de columnas, llamadas ***atributos*** y un número variable de renglones, llamados ***tuplas***.

El número de atributos determina el ***grado*** de una relación; el número de tuplas determina su ***cardinalidad***. El conjunto de valores posibles de un atributo es llamado ***dominio***.

Un aspecto importante en la definición del modelo relacional es considerar las relaciones como ***conjuntos de tuplas*** y como consecuencia: No puede haber dos tuplas idénticas en la misma relación, no hay un orden establecido en las tuplas que contiene una relación e igualmente no hay un orden establecido en los atributos de una relación. Por lo tanto podemos cambiar un atributo de un lugar a otro en la tabla y la información no se altera; igualmente pasa con las tuplas, se puede cambiar una tupla de una posición a otra en la tabla y el significado de la información no se altera.

Desde el momento que consideramos las relaciones como conjuntos, podemos derivar propiedades matemáticas útiles y elegantes en el álgebra relacional.

R

A	B	C
a	1	a
b	1	b
a	1	d
b	2	f

El esquema de esta relación será:

R (A , B , C)

Esta relación tiene grado 3 y cardinalidad 4.

El álgebra relacional tiene un conjunto de operaciones con una o dos relaciones y producen una relación como resultado. Las operaciones básicas del álgebra relacional pueden ser ***unitarias*** y ***binarias***, las primeras requieren de un operando y las segundas de dos operandos, que inclusive pueden ser la misma relación, si esto tiene significado. Otra característica del álgebra relacional es la posibilidad de construir expresiones complejas con estas operaciones básicas, esto le da potencia para construir consultas, de acuerdo a la diferente naturaleza de los datos, aplicadas a una situación práctica.

Las operaciones unitarias son ***selección*** y ***proyección***. Las binarias, son ***unión, diferencia, producto cartesiano*** y ***composición***.

La **selección** SL_F R. La *relación operando R* donde la selección se aplica y *F* una fórmula que expresa el *predicado* de la selección. Esta operación produce una

relación resultado del mismo grado de la relación operando y su cardinalidad depende de las tuplas que satisfagan el predicado. La fórmula contiene nombres de atributos o constantes como operandos, comparaciones aritméticas y operadores lógicos. Por ejemplo la relación cuyo esquema es R (A, B, C), una fórmula válida sería: *F* = (A = B OR A>C) AND NOT A>7.

La ***proyección*** $PJ_{Attr}\ R$ donde *Attr* denota un subconjunto de los atributos de la relación operando *R*, lo cual produce una relación resultado con solo estos atributos. Las tuplas son una copia de los valores de la relación operando, eliminando los duplicados resultantes, por lo que la cardinalidad de la relación resultado siempre será menor o igual a la cardinalidad de la relación operando.

La ***unión*** ***R UN S*** tiene significado solo entre dos relaciones operando *R* y *S* con el mismo esquema; produce una relación resultado cuyo esquema es el mismo de los operandos y las tuplas son aquellas que aparecen en *R* o en *S* o en ambas, eliminando tuplas duplicadas.

La ***diferencia*** ***R DF S*** tiene significado entre dos relaciones operando *R* y *S* con el mismo esquema; produce una relación resultado con esquema de los operandos y las tuplas resultantes serán aquellas que estando en *R* no están en *S*. Esta operación no es conmutativa, de manera que es distinto *R DF S* que *S DF R*.

El ***producto cartesiano*** ***R CP S*** produce una relación cuyo esquema incluye todos los atributos de *R* y *S*. Las tuplas resultantes serán una combinación de cada tupla de la relación *R* con cada una de las tuplas de *S*. En caso de tener atributos con el mismo nombre se adopta un prefijo calificador del atributo con el nombre de su respectiva relación origen de la forma *R.A* y *S.A*.

Las siguientes operaciones se derivan de las anteriores, solo que por su importancia se les destaca con un nombre propio

La *composición* $R\ JN_F\ S$ de dos relaciones operandos *R* y *S* donde la fórmula *F* especifica la composición predicado La fórmula se estructura por comparaciones entre los atributos de las dos relaciones. La relación resultado contiene todos los atributos de ambas relaciones operando, con prefijo calificador en caso de nombres de atributo duplicados, y las tuplas resultantes serán aquellas que cumplan las condiciones especificadas en la fórmula. La composición se estructura de una selección en una relación resultante de un producto cartesiano entre dos relaciones.

$$R\ JN_F\ S = SL_F\ (R\ CP\ S)$$

De esta operación se derivan dos la composición natural y la semicomposición.

La ***composición natural*** ***R NJN S*** de dos relaciones *R* y *S* es una equicomposición en la cual todos los atributos con el mismo nombre de las dos relaciones son comparados y sus valores deben ser del mismo valor; la relación resultante tiene como atributos la suma de atributos de las dos relaciones eliminando los atributos que se repitan.

La ***semicomposición*** ***R SJ*** $_F$ ***S*** donde la fórmula *F* especifica la composición predicado y la relación resultante tiene como atributos los mismos que la relación R y las tuplas son el resultado de la composición de $R\ JN_F\ S$.

$$R\ SJ_F\ S\ =\ PJ_{Attr\,(R)}\,(R\ JN_F\ S)$$

La ***semicomposición natural*** ***R NSJ S.*** Tiene como resultado una relación producto de una semicomposicion y como predicado la misma consideración de la composición natural.

Una última consideración del álgebra relacional es la generalización de la unión, de acuerdo a su propiedad conmutativa y asociativa, en la cual se extiende a más de dos relaciones. $UN\,(R_1, R_2, \ldots, R_n) = R_1\ UN\ R_2\ UN\ \ldots\ UN\ R_n$

R

A	B	C
A	1	a
B	1	b
A	1	d
B	2	f

S

A	B	C
a	1	a
a	3	f

T

B	C	D
1	a	1
3	b	1
3	c	2
1	d	4
2	a	3

A	B	C
A	1	a
A	1	d

SL $_{A=a}$ R

A	B
a	1
b	1
b	2

PJ $_{A,B}$ R

A	B	C
a	1	a
b	1	b
a	1	d
b	2	f
a	3	f

R UN S

R.A	R.B	R.C	S.A	S.B	S.C
a	1	a	a	1	a
b	1	b	a	1	a
a	1	d	a	1	a
b	2	f	a	1	a
a	1	a	a	3	f
b	1	b	a	3	f
a	1	d	a	3	f
b	2	f	a	3	f

R CP S

A	B	C
b	1	b
a	1	d
b	2	f

R DF S

A	R.B	R.C	T.B	T.C	D
a	1	a	1	a	1
a	1	a	2	a	3
b	1	b	3	b	1
a	1	d	1	d	4

R JN $_{R.C=T.C}$ T

A	B	C	D
A	1	a	1
A	1	d	4

R NJN T

A	B	C
a	1	a
b	1	b
a	1	d

R SJ$_{R.C=T.C}$ T

A	B	C
a	1	a
a	1	d

R NSJ T

DISEÑO DE SISTEMAS

LA EMPRESA

La aplicación más frecuente de las bases de datos se hace a los sistemas de información de las organizaciones, a ellas dedico un rápido análisis.

El primer elemento que define "el ser" de una organización es su objetivo, se expresa generalmente en metas. En segundo lugar las tácticas y la estrategia, y no por estar en segundo término son menos importantes, se ha reconocido que determinan el éxito o fracaso una empresa y definen la estructura.

Este marco establece las políticas y prácticas estándar, la descripción de puestos y requerimientos del puesto. Por último se determinan los procedimientos e instrucciones detalladas con los cuales se precisa:

- Qué se va a hacer.
- Quién lo hará.
- Cuándo se hará
- Cómo se hará.

La administración estratégica que incluye la planeación estratégica, su ejecución y evaluación. Esta es parte vital que contempla toda la empresa; en la planeación estratégica se contemplan siete pasos:

Identificar la misión, los objetivos y estrategias.

Analizar el ambiente

Identificar las oportunidades y amenazas

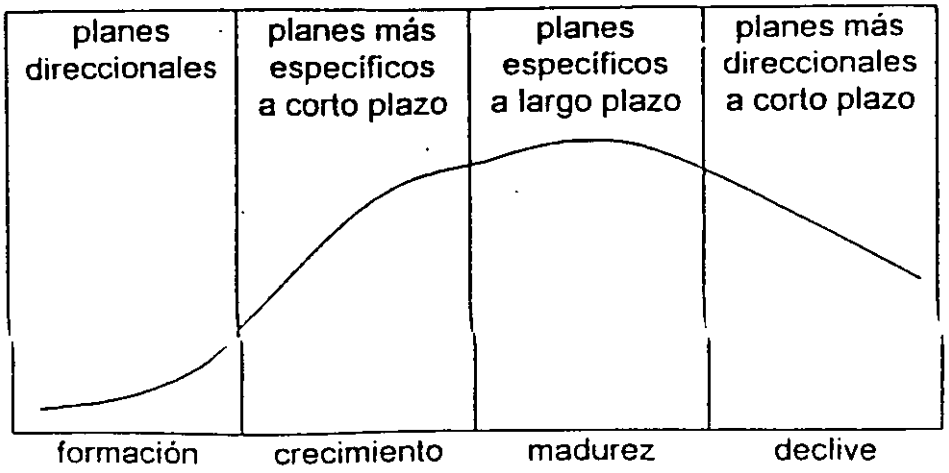
Analizar los recursos de la organización

Identificar las fortalezas y debilidades

Revalorar las misiones y objetivos de la organización

formular estrategias.

En la empresa deberá identificarse la etapa, estando en formación los planes serán direccionales para adaptarse a la situación cambiante; durante el crecimiento los planes son más específicos a corto plazo para aprovechar la situación ventajosa; en la madurez los planes son a largo plazo hasta que se presenta un declive donde se hacen planes más abiertos a corto plazo tratando de corregir la tendencia.



Etapas del ciclo de vida de la empresa y el tipo de planeación.

LA ORGANIZACIÓN DE LAS EMPRESAS

En la organización se tienen dos consideraciones complementarias entre ellas: los niveles organizacionales y la estructura funcional. Los niveles generalmente reconocidos en toda organización son tres: nivel estratégico, táctico y operacional.

El nivel estratégico lo compone la dirección de la empresa: director y gerentes; en ellos recae la responsabilidad de todas las operaciones realizadas; de ellos emana la autoridad hacia los demás niveles: la responsabilidad se comparte, la autoridad se delega. Su responsabilidad: cumplir con eficacia y eficiencia. El nivel táctico lo forman los mandos medios y supervisores, estos últimos están en contacto directo con el nivel operativo de la organización. El nivel operacional lo componen las personas ejecutoras de las actividades en forma directa, en este nivel no se dirige el trabajo de otros.

Los niveles estratégico y táctico hacen la administración, esto es, dirigen la actividad de otras personas Las funciones son clasificadas según la administración tradicional como planeación, organización, dirección y control Estas funciones difieren en grado según el nivel que una persona ocupa, en promedio los supervisores ocupan más de la mitad de su tiempo a la dirección, una cuarta parte a la organización y lo demás a planeación y control En general los administradores además de hacer la administración tradicional ocupan gran tiempo en comunicación, administración de recursos humanos y relaciones públicas. Estos roles son determinantes en sus requerimientos de información.

Las empresas modernas han ampliado los roles del nivel operativo, se les ha facultado, es decir, se les ha dado poder de decisión; esto requiere una buena capacitación y un sistema de comunicación que le permite a la administración tener el control.

Una vez definidos los niveles se establece el principio de excepción: deja a los niveles inferiores la solución de los problemas ordinarios y al nivel táctico o estratégico los problemas grandes y excepcionales.

La estructura funcional consiste en agrupar las actividades relacionadas y disponerlas para personal de la misma especialidad. Esto incluye definir:

- Las actividades por departamento
- Estandarización de cada actividad
- Coordinación de actividades dentro del departamento
- Facultades y responsabilidad del personal en la toma de decisiones
- Niveles y tramos de control
- Mecanismos de coordinación entre los distintos departamentos

En las empresas grandes se hacen separaciones por tipo de producto o región estableciendo divisiones; cada una tiene su propia estructura con niveles y departamentos, como si fueran empresas independientes con autonomía en sus operaciones.

En últimas fechas se ha reconocido la utilidad de formar verdaderos equipos, algunos de ellos informales y otros formalmente constituidos interdepartamentalmente. Esta modalidad da origen a interacción directa entre personal de niveles tácticos ubicados en diferentes departamentos, haciendo una estructura transversal con su propio responsable, que viene a ser líder de proyecto. El personal que participa en estos equipos tiene dos jefes, flexibilizando el principio de autoridad. Por su temporalidad pocas veces se rigen por un manual de organización haciendo vulnerable la estabilidad de la organización. Una organización que opera con estas características tiene una estructura tipo matricial.

La parte final de la organización de una empresa consiste en escoger las personas idóneas a cada puesto administrativo y con esto se le da vida a la organización, al nombrar los administradores desde el nivel más alto hasta los supervisores, con su correspondiente autoridad y responsabilidad y contratando el personal operativo en cada puesto de trabajo.

La actuación del personal genera datos que de registrarse y organizarse producen la información necesaria para la toma de decisiones, reduciendo el riesgo que esto implica.

EL SISTEMA DE INFORMACION

La importancia del sistema de información para una organización se puede comparar en el cuerpo humano con su sistema de circulación, el beneficio que nos ha traído la informática ha sido una ventaja estratégica de las organizaciones más modernizadas. El aprovechamiento de las redes de computadoras en todos los rincones de la empresa ha venido a cambiar la forma de manejar el sistema de información.

Vemos organizaciones operando muchos programas pequeños con su propio conjunto de archivos, algunos sin posibilidad de aprovecharlos para una aplicación diferente para la que fueron hechos, proliferando el almacenamiento de datos, muchos de ellos inconsistentes con las demás aplicaciones. Esto produce más carga de trabajo, poca comunicación y una rápida obsolescencia de la información.

Una modalidad en el manejo de información usando redes locales se ha visto usando archivos del tipo Dbase, esto trae ciertas ventajas, ya que se comparten los datos pero los efectos indeseables pueden poner en riesgo la confianza en el almacenamiento, puesto que no se puede proteger los datos de una intromisión, al operar en red existe el riesgo de perder información si en varias computadoras al mismo tiempo pueden dar de alta y baja información para una misma tabla compartida por la red, al cierre de cada sesión la computadora que se desconecta coloca fin de archivo intermedio a los registros de la tabla. En sistemas con grandes volúmenes de datos y muchos indices hacen una sobrecarga del tráfico de datos en la red; cada indice debe estar presente en la estación que use una tabla, normalmente se envía una parte del índice llamado página, la red debe hacer pasar páginas según el recorrido que se haga sobre la tabla en cada computadora, esto la mantiene ocupada gran parte del tiempo alentando los procesos que deben esperar a que les lleguen los datos necesarios.

La solución a todos estos inconvenientes ha estado disponible hace más de quince años en las grandes computadoras y hasta esta década la tenemos en equipos de escritorio Me refiero a los sistemas de manejo de bases de datos relacionales (DBMS).

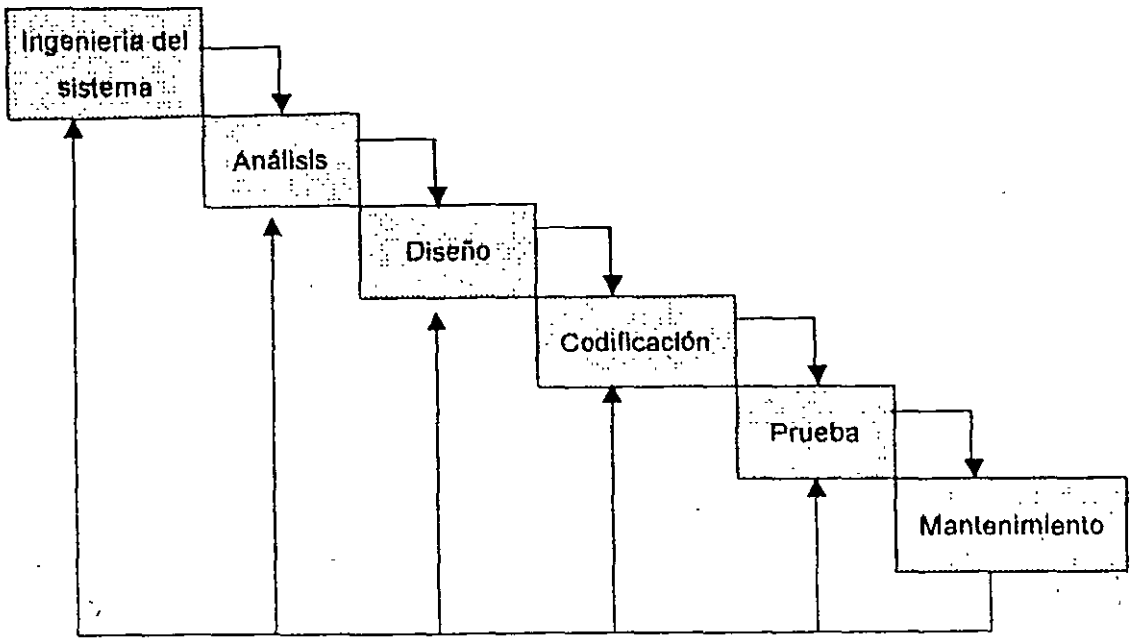
En la actualidad existen sistemas de información ajustables flexiblemente que cubren una empresa al 100% y se pueden poner en operación en menos de tres meses en organizaciones de 50C o más personas, con el consiguiente fortalecimiento de la organización. Este tipo de sistemas así como otros que las empresas van desarrollando particularmente tienen como plataforma un sistema de manejo de base de datos, el cual se comunica en lenguaje estructurado para consulta (SQL).

El SQL es un estandar apropiado a la administración de datos y contiene instrucciones para definición de datos y sus reglas de integridad (DDL), instrucciones para manejo de datos (DML) y otro grupo de instrucciones para la administración y seguridad El tráfico en la red se reduce drásticamente ya que el proceso de organización, control de acceso y búsqueda se hace en una máquina servidor de base de datos, las peticiones vienen diferentes máquinas con sistemas operativos diferentes, aplicaciones, paquetería o programas de consulta que se les considera clientes La petición viaja como una cadena de caracteres muy reducida por cierto, la respuesta que se hace llegar puede ser desde un dato de la suma de ventas de un dia hasta la totalidad de una tabla sin necesidad de enviar índices. Todos los procesos de recuperación o cálculo los realiza el programa que se le ha dado en llamar el motor de la base de datos.

Este entorno es llamado Arquitectura Cliente – Servidor, que a su vez esta soportado por lo que se conoce como un sistema abierto.

La ingeniería del Software contempla el ciclo de vida clásico para la realización de sistemas computacionales, llamado "modelo en cascada".

La ingeniería del Software contempla el ciclo de vida clásico para la realización de sistemas computacionales, llamado "modelo en cascada".



Para la ingeniería de sistemas el método de investigación más aceptado es el enfoque de sistemas, en él se enfatiza el sistema total en lugar de ir primero a los componentes. Una empresa es un sistema abierto expuesto al entorno, el sistema es un conjunto de componentes que interactúa para lograr un objetivo común, el cual se logra por la función que realizan las partes y la interacción entre ellas. El entorno también actúa sobre el sistema, solo que para fines de diseño no se tiene el control de su actividad, a lo más se espera controlar su efecto.

El análisis debe contemplar toda la empresa y su entorno específico. El entorno específico es una parte del entorno que tiene que ver directamente con la realización de las metas de una organización

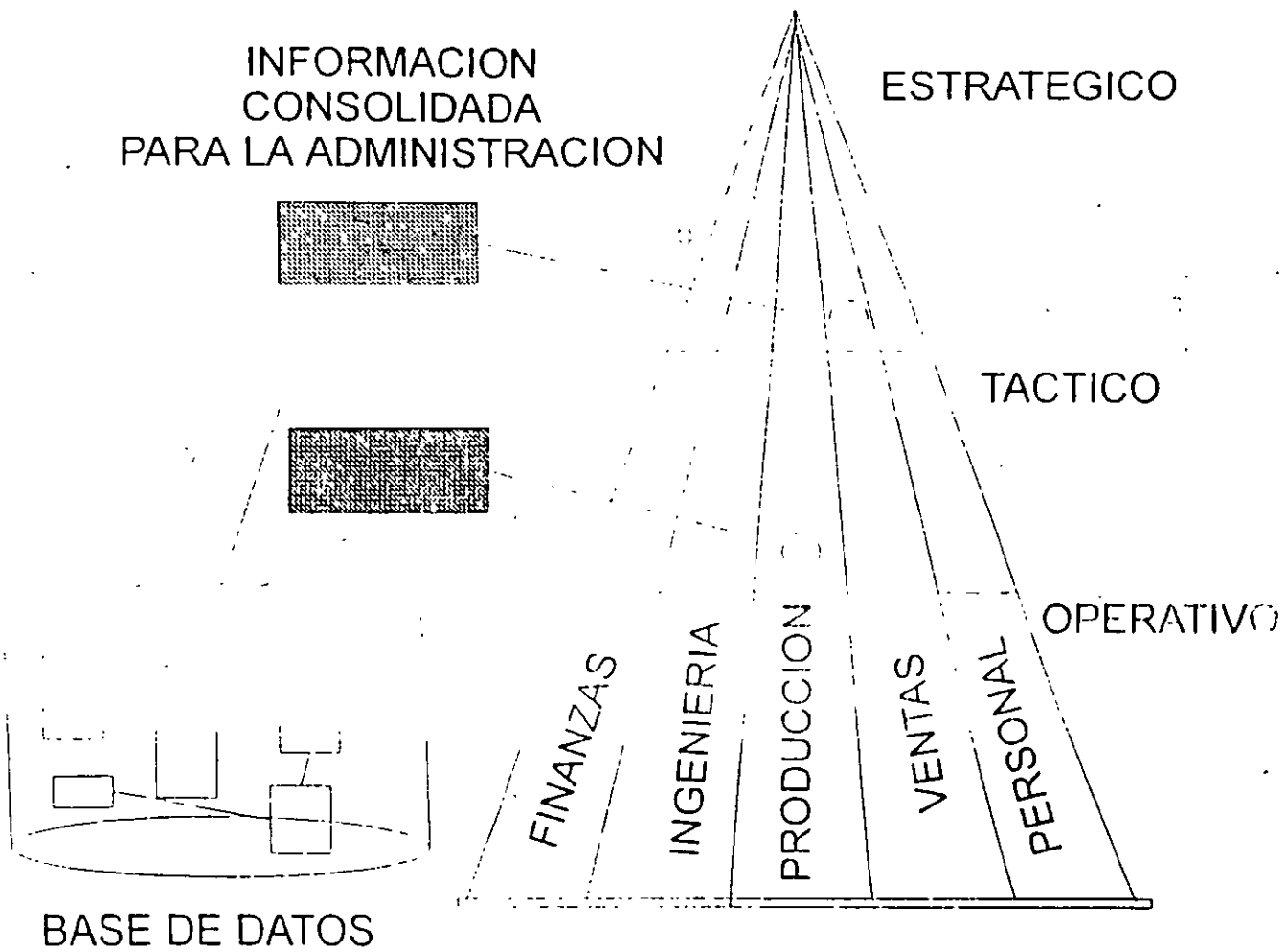
El paquete de información que debe estudiar este análisis incluye el programa de la administración estratégica, el organigrama, manual de organización y los procedimientos e instrucciones de las diferentes operaciones de los departamentos.

La técnica más difundida para la etapa de análisis es HIPO de IBM la cual contempla la jerarquía más toda la documentación de entrada – proceso – salida. Esto en detalle identifica todos los reportes que se esperan entregar y los datos que se tienen disponibles para idear el proceso que permite hacer la transformación de la entrada en salida.

De esta etapa se identifican dos elementos los procesos y los datos, esto definirá los programas y la base de datos.

En el desarrollo de un sistema de información se requiere un grupo interdisciplinario que identifique los elementos que actúan sobre el sistema y de las partes relevantes que la componen para después al momento de aceptación verifique el cumplimiento de los requerimientos de la empresa, haga las pruebas y por último apruebe su liberación.

LA EMPRESA Y EL SISTEMA DE INFORMACION
con un enfoque sistémico



DISEÑO DE LA BASE DE DATOS

DISEÑO CONCEPTUAL

Una vez identificados los elementos que interactúan en la empresa se determinan las entidades a estudiar y de ellos se escogen los atributos, su relación con otras entidades y los atributos producto de esta relación que sean relevantes a la organización, esto es el análisis entidad – relación

DISEÑO LOGICO

En esta etapa se escoge la forma de implantar en un sistema computacional, las opciones que se han manejado Jerárquica y en red, han dejado de usarse para quedar como opción única las bases de datos relacionales. En estos últimos años se ha fomentado el diseño de las bases de datos orientada a objetos (OODB), no he tenido la oportunidad de ver una empresa cuyo soporte informático total sea un sistema basado en este tipo.

Al escoger una base de datos relacional, se diseñan los esquemas que definen cada tabla Las tablas representan los conjuntos de entidades o relaciones entre dos entidades

Una etapa final de análisis es la determinación de las reglas del negocio para ser usadas como reglas de dominio en ciertos atributos y restricciones de integridad.

En este curso nos apoyaremos en las instrucciones correspondientes del Lenguaje Estructurado de Consulta (SQL) para hacer el diseño, planear su presentación, estructurar su integridad, controlar los permisos de acceso, definir las instrucciones complejas que serán tratadas como unitarias.

DISEÑO FISICO

Su implantación física queda a cargo del sistema manejador de la base de datos y está fuera del alcance de este trabajo Es indispensable mantener el enfoque propuesto por Ted Codd en su articulo de Computerworld en 1985 para escoger el programa llamado frecuentemente "motor de la base de datos".

DOCE REGLAS DE CODD PARA DBMS RELACIONAL.

1.- Regla de información

Toda la información de una base de datos relacional está representada explícitamente a nivel lógico y exactamente mediante un modo: guardando valores en tablas.

2.- Regla de acceso garantizado

Todos y cada uno de los datos, valor atómico, de una base de datos relacional se garantiza que sean lógicamente accesibles recurriendo a una combinación de nombre de tabla, valor de clave primaria y nombre de la columna.

3.- Tratamiento de valores nulos

Los valores nulos, son distintos a una cadena de blancos y distinta de cero, se soportan en los DBMS para representar la falta de información y la información inaplicable.

4.- Catálogo en linea basado en el modelo relacional.

La descripción de la base de datos se representa a nivel lógico del mismo modo que los datos ordinarios, de modo que los usuarios autorizados puedan aplicar a su interrogación del mismo lenguaje que aplican a los datos regulares.

5.- Regla de lenguaje completo

Un sistema relacional puede soportar varios lenguajes y varios modos de uso terminal. Debe haber al menos un lenguaje de alto nivel, con una sintaxis bien definida, que de soporte a los puntos siguientes:

- Definición de datos
- Definición de vista
- Manipulación de datos
- Restricciones de integridad
- Autorización
- Fronteras de actualización (Inicio, regreso y terminación).

6.- Regla de actualización de vista

Todas las vistas que sean teóricas actualizables son también actualizables por el sistema.

7.- Inserción, actualización y supresión

La ampliación de la regla dos a la inserción, actualización y supresión de datos nos permite hacer cualquiera de estas operaciones a un dato o una fila completa, sin tener que llegar por medio de un recorrido o movimiento de otros datos.

8.- Independencia fisica de los datos

Los programas de aplicación permanecen lógicamente inalterados, a pesar de los cambios en la representación del almacenamiento o en el método de acceso.

9.- Independencia lógica de los datos

Los programas de aplicación y las actividades terminales permanecen lógicamente inalterados, cuando se efectúen cambios sobre las tablas de base, no afectarán la capacidad del usuario para trabajar con los datos

10.- Independencia de integridad

Las restricciones de integridad específicas para una base de datos relacional deben ser definibles en el sublenguaje de datos relacional y almacenables en las tablas en el catálogo, no en los programas de aplicación.

11.- Independencia de distribución

El programa administrador del DBMS debe ser capaz de manipular los datos distribuidos, localizados en otros sistemas informáticos.

12.- Regla de no sub-versión

El lenguaje de bajo nivel, distinto al lenguaje de alto nivel, no puede ser utilizado para alterar o suprimir las reglas de integridad.

DISEÑO CONCEPTUAL

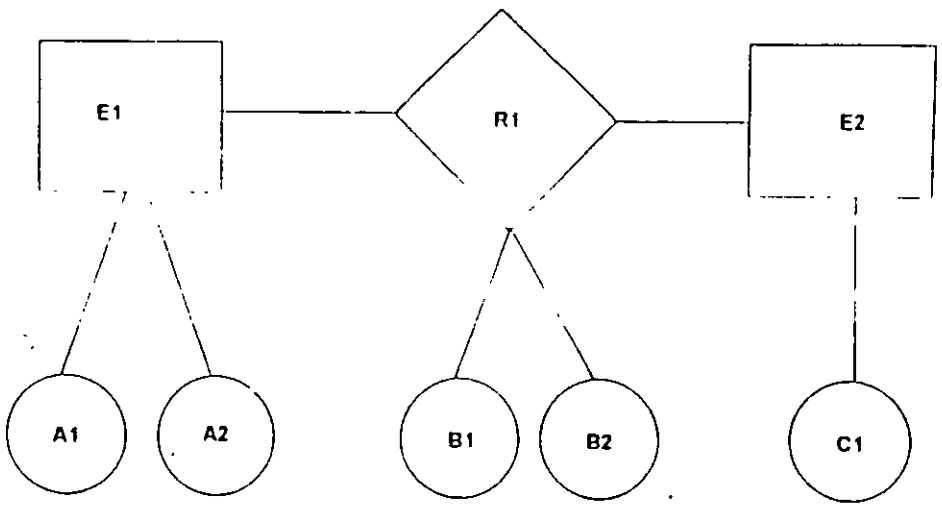
El primer paso consiste en determinar que información es relevante para los usuarios en una empresa. El diseño de sistemas arroja un diccionario de datos, requerimientos de entrada, procesos a realizar y los reportes de salida que se espera obtener.

Para llegar a esto fue necesario conocer la empresa y en detalle como se realizan los procedimientos y su relación con el sistema de información que permite la comunicación entre los usuarios del sistema.

El enfoque de sistemas se aplica considerando el todo de la empresa. El diseño consiste en identificar los conjuntos que forman una entidad, identificar todas las entidades y las relaciones entre entidades. Una vez identificados obtenemos los atributos necesarios para producir los resultados o salidas definidos en el diseño del sistema.

MODELO ENTIDAD – RELACION

Para aplicar el modelo Entidad - Relación usamos una simbología simple y útil para identificar las entidades con sus atributos y las relaciones entre entidades y los atributos de esta relación.



Los cuadrados representan una entidad, el rombo una relación y los círculos representan los atributos correspondientes a una entidad (A1, A2, C1) o correspondientes a una relación entre entidades (B1, B2).

La relación entre el conjunto de elementos que forma una entidad con el conjunto que forman otra entidad pueden ser:

de uno con uno	1
de uno con uno o ninguno	C
de uno con muchos	M

Viendo esto como una relación en un sentido y en el sentido contrario se podrá observar otro tipo de relación dando la siguiente combinación:

relaciones	
de uno con uno	1:1
de uno con uno o ninguno	1:C
de uno con muchos	1:M
de uno o ninguno con uno	C:1
de uno o ninguno con uno o ninguno	C:C
de uno o ninguno con muchos	C:M
de muchos con uno	M:1
de muchos con uno o ninguno	M:C
de muchos con muchos	M:M

Para terminar el diseño debemos observar los dominios de los atributos, en caso que sigan cierta regla, y establecer las restricciones de integridad.

A continuación se hace necesario aplicar esta simbología a casos prácticos con los cuales se encuentren las situaciones que generan dudas, que al resolverse permiten dejar bien cimentado su uso. Las hojas siguientes presentan casos de estudio en los cuales existe una carencia de detalles y se trazan las líneas generales del sistema, esperando una respuesta creativa y que de oportunidad para discutir el diseño propuesto en grupos de trabajo.

CASO DE ESTUDIO

SISTEMA DE CONSTRUCCION:

Una constructora tiene como ventaja competitiva su sistema de "precios unitarios", ya que en él guarda la técnica propia de hacer los trabajos que le encargan.

En orden de importancia estudiaremos primero los contratos, estos se realizan con un cliente, para cada uno se tiene un presupuesto, en el se separan especialidades por tipo de obra, como son albañilería, plomería, carpintería, etc. y se definen los conceptos a detalle y como se van a cobrar por unidad de obra (litro, metro, kg, pza, lote).

En el contrato se incluye un programa de construcción apoyado en el presupuesto, se detallan las diferentes operaciones con su duración y secuencia según el orden que lleva el proceso de construcción y cuanto se gasta por periodo.

El presupuesto resultó de un análisis de cada concepto según el proceso de construcción y el cual incluye maquinaria, equipo, mano de obra y tiempos ocupados; materiales en cantidad, rendimiento y el precio que representa todo esto.

Un precio unitario determina el costo por concepto de construcción, puede ser usado para definir otro precio unitario más complejo.

Para facilitar el costeo de mano de obra se especifican cuadrillas de trabajo donde se identifica la categoría y número de personas que la forman y el rendimiento que realizan. Igualmente en los materiales elaborados se especifican las cantidades de materiales básicos que lo forman.

Para cobrar se usa una estimación que usa los conceptos del presupuesto y las cantidades que se han ejecutado con los precios que se establecieron. En caso de volúmenes superiores al presupuesto se hacen en estimaciones extra; en caso de conceptos fuera de presupuesto se debe hacer un análisis y convenir el precio como extensiones del presupuesto

CASO DE ESTUDIO

SISTEMA DEPORTIVO:

En un club deportivo se tiene organizada por deportes la actividad en roles de juego de torneos continuos, donde los equipos y las canchas se determinan previas al inicio del semestre.

Se han separado torneos por categorías y sexo y tipo de horario; una restricción para los jugadores es que solo pueden pertenecer a un equipo por el tipo de horario, no importando si es el mismo o diferente deporte.

Los árbitros son programados según su número requerido por cada deporte correspondiente.

Para ajustar las actividades de la semana siguiente se tiene un programa de juntas semanales por deporte, categoría y horario. En ellas se resuelven los castigos que ameritan los reportes de los árbitros. Aquí se dan a conocer los resultados de la jornada pasada y se confirma el rol de la semana siguiente.

Se debe contemplar el control de cuotas de inscripción y las mensualidades por equipo, los árbitros son pagados por el club en fecha posterior a realizar sus actividades y de lo cual debe haber un control.

Para la promoción de membresía y superación deportiva se realizan entrenamientos de los diferentes deportes donde tanto asisten jugadores inscritos como público en general de acuerdo a su edad, en ellos los entrenadores determinan las posibilidades de cada jugador para un alto rendimiento. A los jugadores con potencial se les ofrecen becas parciales o hasta totales según su nivel.

CASO DE ESTUDIO

SISTEMA HOSPITALARIO:

En un centro médico se atienden consultas externas, así como, atención hospitalaria. Los médicos y enfermeras son de diferentes especialidades y de alto prestigio, todos con un historial de amplia experiencia.

La atención externa se realiza en consultorios, de acuerdo a las horas previstas. Las citas se apartan en una oficina central por la cual queda establecido el compromiso de día y hora. Los cargos por atención se hacen según las consultas realizadas y en casos de falta repetida a una cita apartada causa un recargo.

Los pacientes internos o externos cuentan con un expediente, donde se registran análisis, consultas, diagnósticos, intervenciones, curaciones y medicación prescrita. La atención hospitalaria cuenta con camas, salas de curaciones, terapias y operaciones. La programación de actividades se lleva con un estricto control para poder atender las emergencias en salas libres.

Normalmente para el tipo de operaciones más frecuentes se tienen equipos de médicos y enfermeras, donde se tiene un responsable de la operación.

Una parte muy costosa del control se tiene en los bancos de sangre, vacunas, medicamentos especiales, prótesis y materiales ortopédicos.

Los pacientes tienen una cuenta con todos sus cargos por la atención recibida y por los materiales ocupados y las medicinas aplicadas. Los pagos se pueden recibir en todo tiempo ya que existe el servicio de cajera permanente.

Los honorarios al personal médico y los sueldos a las enfermeras se controlan de acuerdo al pago contratado y las actividades extra que se presentan durante la semana.

Para la aceptación de nuevos médicos, según su especialidad se les practica una evaluación, si es competente entra directo en funciones, de otra manera trabaja como asistente durante medio año, para volver a presentar otra evaluación.

CASO DE ESTUDIO

SISTEMA ESCOLAR:

En un instituto de educación personalizada se ofrecen estudios superiores en varias carreras y estudios del nivel medio superior del tipo bachillerato tecnológico en varias especialidades. Los horarios de clase cubren todo el día y cada alumno lleva el número de materias con los maestros que desea; los grupos se forman con un mínimo de quince alumnos. En talleres y laboratorios cada grupo lo atienden varios maestros, para realizar trabajos diferentes y supervisarlos adecuadamente.

Cada carrera tiene un plan de estudios, contiene un perfil del egresado perfectamente detallado que constituye un contrato de servicio, una lista de materias, con sus objetivos, unidades y temas.

Los maestros están clasificados en varias categorías: titulares, asociados y asistentes. Cada uno cuenta con un horario, donde se incluyén las materias con los grupos, hora y salón, también asesorías extraclase y actividades extraescolares.

Los maestros se organizan en academias según la afinidad de materias agrupadas. Para las categorías de titular y asociado, además impartir clases, se les da tiempo para preparar materiales auxiliares de enseñanza: videos, acetatos, guías de estudio, textos por cada materia. Estos materiales se controlan para su duplicación y entrega a los maestros que lo requieren. Un maestro puede pertenecer a varias academias.

Un control necesario son las asistencias de alumnos a clase La actividad extraescolar de tipo cultural, técnica y deportiva se organiza en clubs, los maestros participan como asesores La promoción más efectiva son sus egresados, de los cuales e tiene un seguimiento de sus trabajos y cursos de superación. De los servicios más ocupados son las fichas bibliográficas donde relacionan las materias, los temas y prácticas con los libros de su biblioteca

Para la administración la escuela cuenta con departamentos donde se separan las actividades relativas a maestros, empresarios, alumnos y padres de familia. Cada departamento tiene a su vez oficinas para tratar los servicios por separado. Las colegiaturas y los pagos a maestros se hacen contra un control por asistencia, reporte de avance o terminación de actividad.

CASO DE ESTUDIO

SISTEMA DE VENTAS:

Esta empresa fundamenta su éxito en las ventas a crédito; sus clientes pueden tener varias cuentas pendientes y cada una con vencimientos en diferentes fechas. Los pagos se reciben por separado para cada cuenta.

Ciertos clientes prefieren pagar en su domicilio, a ellos se les envía un cobrador. Al cliente con atraso de más de dos meses en un pago también se les envía. Un control importante para la cobranza es manejar rutas, y tener un reporte de cuentas con pagos por vencer en el mes próximo, en este mes, los vencidos el anterior, dos mese antes, y más de dos meses. Hay cobradores, muy buenos, que manejan más de una ruta de cobro.

La tienda tiene departamentos y cada uno tiene control de las existencias de las mercancías, hace sus pedidos y devoluciones. En caso de un cliente que devuelve mercancía gestiona la aplicación de un crédito para abonar la cuenta o la devolución del dinero.

Otro servicio son los apartados, en los cuales se entrega la mercancía al terminar de pagar. Los abonos regulares se programan por mes. Cuando un cliente de apartados es regular y cumple se le admite para darle crédito.

Los proveedores manejan pago anticipado, contra entrega y a crédito de uno a varios meses. Los proveedores reciben pedidos, luego envían la mercancía, por lo regular no llega toda en un solo entrego. Es de vital importancia que se controlen los pedidos para la temporada, fuera de ella no tiene efectividad.

La fuerza de ventas es su publicidad, se hace promocionando uno o varios departamentos. Su impacto se mide según responden las ventas registradas.

CASO DE ESTUDIO

SISTEMA DE MANUFACTURA:

Una fábrica tiene varias líneas de producción según el tipo de producto. Al personal de producción se grupa por línea de producción. Una línea de producción puede tener uno o varios procesos. Cada proceso ocupa maquinaria, materias primas, materiales y mano de obra directa.

Se tiene un registro de los estándares producidos por unidad de tiempo y por cantidad según el rendimiento de materia prima y materiales por unidades producidas. De acuerdo al tipo de producto se ocupan ciertas líneas de producción Usando diferentes partes se hacen ensambles pueden usarse en uno o varios productos.

En temporadas altas de producción se tienen varios turnos de trabajo, el cual se va rolando cada quince días. Se programa la producción de acuerdo al programa tentativo de ventas, ajustandolo con las ventas reales y según la carga de trabajo para cada línea de producción.

Una actividad de apoyo es el almacén, en él se tienen materias primas, materiales, partes, subensambles y producto terminado. Se clasifica por lote de fabricación para hacerle pruebas de calidad y para hacerle seguimiento en el mercado. Esto ha servido de apoyo al comercio, en casos de robo, la mercancía tiene el número de lote y la fabrica registra los envíos para cada cliente.

En la empresa además del departamento de producción tiene otros de apoyo, con personal propio adscrito por departamento. Ultimamente se han formado equipos temporales para la solución interdisciplinaria de un problema, son formados con personal de varios departamentos y el nombramiento de un líder.

El departamento administrativo lleva el control de compras, ventas y pagos realizados Para lo cual requiere un reporte diario de entradas y salidas de almacén, el cual confronta con los contratos suscritos con los clientes mayores.

CASO DE ESTUDIO

SISTEMA TURÍSTICO:

Un hotel está asociado con un sistema de reservaciones por computadora el cual es usado por una red de agencias turísticas. La contratación se puede hacer con anticipo, pago de renta de cuarto o el servicio todo incluido pagado por anticipado.

El hotel tiene cuartos de diferente categoría, los huéspedes tienen una cuenta donde se integra la renta diaria, servicios personales como son limpieza de ropa, peluquería, masaje, teléfono, servicio médico, etc., servicio en restaurantes, renta de vehículos y recorridos turísticos.

Para el registro de un nuevo huésped se consultan las reservaciones, la disponibilidad de cuartos para los dias que desea permanecer, su antigüedad y frecuencia como cliente.

Como servicio incluido se organizan actividades deportivas, turísticas y sociales mediante un sistema interno de reservación. En estos servicios pueden participar personas no huéspedes mediante su acreditación y pago.

El personal de servicio tiene una alta especialización, atiende desde la recepción, restaurantes, áreas de actividades, servicios personales, mantenimiento, limpieza y la administración del hotel.

Los proveedores son muchos, de diferente tipo de mercancías y servicios, sus pagos se confrontan con las requisiciones de compra de cada jefe de departamento dentro del hotel

CASO DE ESTUDIO

SISTEMA BANCARIO:

Esta institución cuenta con una red de cajeros automáticos en los cuales los clientes pueden hacer consultas, retiros y depósitos. Los depósitos pueden ser en documento o en moneda nacional.

Las cuentas llevan el registro de movimientos del mes, el saldo inicial y final. El banco ofrece un servicio de depósito con intereses y para los que necesitan dinero se les extiende una línea de crédito a solicitud y solvencia del cliente. Los depositantes a plazo pueden disponer al momento de su vencimiento, el cual puede extenderse por otro periodo, de acuerdo a los estipulado en el contrato teniendo un día perdido para su disposición.

Se tienen campañas de colocación de dinero, en ellas se ofrece al cliente un préstamo por un año con una tasa fija de interés con posibilidad de pagar por anticipado en cualquier momento. En casos de prestamos directos se hace lo mismo con un ajuste del periodo requerido por el cliente, el préstamo se va cargando a la cuenta a su vencimiento.

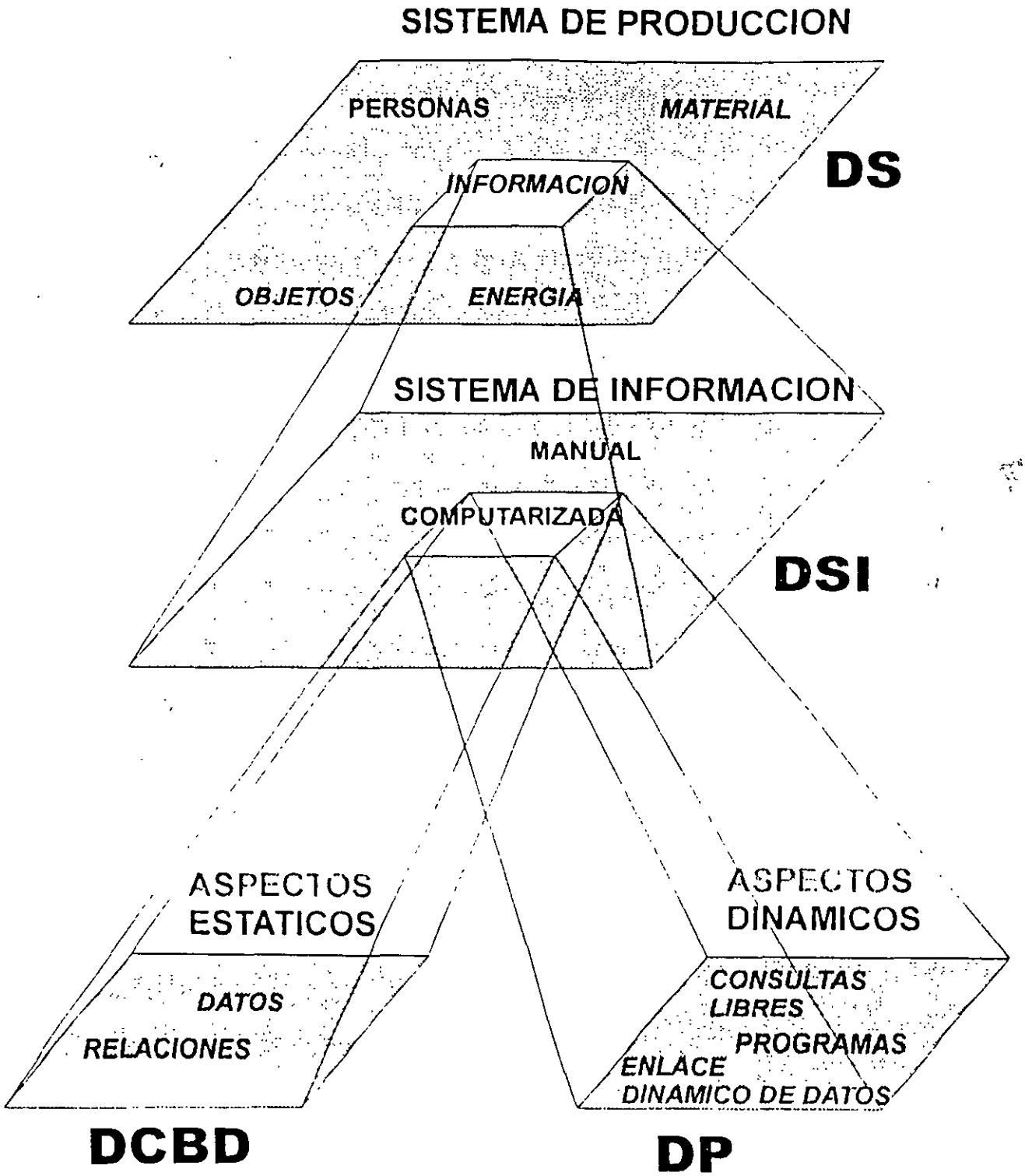
Para créditos mayores se registra una o más propiedades inmuebles como garantía única de cada contrato de préstamo. Los registros pasados se tienen presentes para aprobar líneas de crédito o prestamos.

En apoyo de las empresas se ofertan acciones entre los clientes para que compren o vendan según su deseo recibiendo utilidades variables, generalmente mayores que las del banco.

El banco puede requerir dinero, para esto pertenece a una asociación de bancos a los cuales se les pide el dinero por el tiempo necesario, los accionistas del propio banco exigen que se paguen estos prestamos cuando existe dinero sobrante, manteniendo las reservas reglamentadas en porcentaje del total prestado

Se ha observado que los clientes tienen varias cuentas para llevar su control de ventas, reservas financieras y depósitos a plazo.

ENFOQUES DE DESARROLLO EN APLICACIONES COMPUTACIONALES



DISEÑO LOGICO

Una vez que determinamos usar una base de datos relacional se hace la adaptación de los datos identificados en el diseño conceptual en tablas con un nombre y columnas por cada atributo. A continuación se especifica una o más columnas como llave de cada tabla, se le selecciona desde el momento que garantiza no dar lugar a duplicados. Las relaciones entre tablas se hacen mediante llaves ajenas, esto es, la llave de una tabla es un atributo de la misma, este atributo contenido en otra tabla relaciona las tablas siendo una llave principal y en otra o en otras tablas llave ajena.

Posteriormente se hace el proceso de normalización, su propósito: evitar la redundancia y ajustarse a la situación real puesto que estamos usando un modelo. Por último se definen las reglas de integridad.

La estructuración de la base de datos en forma lógica permite su conversión directa al lenguaje del sistema de manejo de la base de datos, esto aplicado al DBMS genera las estructuras fisicas de almacenamiento. Esto queda totalmente oculto para el administrador y usuarios de la base de datos.

NORMALIZACION

Una vez que tenemos identificadas las entidades y las relaciones entre entidades, convertimos a tablas donde cada columna es un atributo y cada fila corresponde a una entidad del conjunto. Por lo general se presenta el proceso de normalización usando un formalismo matemático, aquí nos ocuparemos de hacerlo en forma práctica.

Una primera propiedad que tenemos en estas tablas es la libertad de colocar las columnas y filas en cualquier orden y vemos que no se altera el significado de la informacion. La aplicacion de las reglas de normalizacion considerar el concepto de llave de una relación Se debe escoger un atributo en cada tabla mediante el cual se asegure que observando su dominio encontramos que solo puede haber un valor para cada fila, con esto nos aseguramos que cada fila representa una entidad distinta de las demás registradas y no permite dos o más registros que sean la misma cosa o entidad. En caso que un atributo no sea suficiente para distinguir cada entidad se podrán tomar más de uno y en grupo deberán asegurar que no existe redundancia.

En la práctica se aplican las tres primeras formas normales y en casos especiales se deberá considerar una cuarta o hasta quinta forma normal.

REGLAS DE NORMALIZACION

1FN) Una relación está en primera forma normal cuando hay un elemento en cada celda, y solo uno. Los valores Nulos son permitidos en cualquier atributo que no sea llave de la relación.

2 FN) Una relación está en segunda forma normal cuando satisface la primera y cuando cada atributo es funcionalmente dependiente de toda la llave, no solo de una parte de ella.

3 FN) Una relación esta en tercera forma normal cuando está en segunda forma normal y ningún atributo es funcionalmente dependiente de otro atributo que no sea llave.

4 FN) Está en cuarta forma normal cuando está en tercera forma normal y no permite que un atributo varie con respecto a otro dando un dependencia múltiple.

5 FN) Una relación esta en quinta forma normal si está en cuarta y al juntar dos tablas produce un resultado irreal, aun cuando sea posible.

Tomada una descripción informal de datos, se observa que están relacionados por filas para un mismo tipo de producto y por columnas se identifica un mismo atributo.

ALIMENTOS

UNIDAD	FRUTA	TIPO	COLOR	PESO	PRESENTACION
KG PZA	MANZANA	FRESCA	VERDE GOLDEN	1 .350	A GRANEL, CHAROLA FAMILIAR C/6
CAJA, PZA	FRESA	MERMELADA	ROJO OBSCURO	.630 , .300 , 1.3, 7.56	LITRO, MEDIOS CUARTOS
SIX PACK	DURAZNO	YOGURT	ROSA PALIDO	1.8	CUARTO

CONVERSIÓN A PRIMERA FORMA NORMAL:

ALIMENTOS

UNIDAD	FRUTA	TIPO	COLOR	PESO	PRESENTACION
KG	MANZANA	FRESCA	VERDE GOLDEN	1	A GRANEL
PZA	MANZANA	FRESCA	VERDE GOLDEN	.350	CHAROLA FAMILIAR C/6
PZA	FRESA	MERMELADA	ROJO OBSCURO	.630	MEDIOS
PZA	FRESA	MERMELADA	ROJO OBSCURO	.300	CUARTOS
PZA	FRESA	MERMELADA	ROJO OBSCURO	1.3	LITRO
CAJA	FRESA	MERMELADA	ROJO OBSCURO	7.56	MEDIOS
SIX PACK	DURAZNO	YOGURT	ROSA PALIDO	1.8	CUARTO

Dando como resultado una tabla con un valor en cada celda. Falta definir la llave de la tabla.

DE PRIMERA FORMA NORMAL

ALIMENTOS

UNIDAD	FRUTA	TIPO	COLOR	PESO	PRESENTACION
KG	MANZANA	FRESCA	VERDE GOLDEN	1	A GRANEL
PZA	MANZANA	FRESCA	VERDE GOLDEN	.350	CHAROLA FAMILIAR C/6
PZA	FRESA	MERMELADA	ROJO OBSCURO	.630	MEDIO
PZA	FRESA	MERMELADA	ROJO OBSCURO	.300	CUARTO
PZA	FRESA	MERMELADA	ROJO OBSCURO	1.3	LITRO
CAJA	FRESA	MERMELADA	ROJO OBSCURO	7.56	MEDIOS
SIX PACK	DURAZNO	YOGURT	ROSA PALIDO	1.8	CUARTOS

PASAR A SEGUNDA FORMA NORMAL:

PRESENTACIONES

UNIDAD	PESO	PRESENTACION	FRUTA
KG	1	A GRANEL	MANZANA
PZA	.350	CHAROLA FAMILIAR C/6	MANZANA
PZA	.630	MEDIO	FRESA
PZA	.300	CUARTO	FRESA
PZA	1.3	LITRO	FRESA
CAJA	7.56	MEDIOS	FRESA
SIX PACK	1.8	CUARTOS	DURAZNO

ALIMENTOS

FRUTA	COLOR
MANZANA	VERDE
	GOLDEN
FRESA	ROJO
	OBSCURO
DURAZNO	ROSA
	PALIDO

Se dividen en dos tablas cada una con la llave de la que son dependientes los atributos que la forman

TOMEMOS LA SIGUIENTE TABLA:

ALUMNOS

MATRICULA	SEM	NOMBRE	ASESOR	CARRERA ASESOR
167897	5	MALDONADO CORONA JUAN	JGS	ING. IND.
114598	3	HERNANDEZ RUIZ MARTIN	PSP	ING. ELEC.
127599	1	VALLE SILIS JORGE	EGC	TEC. AGRO
213097	5	ARREDONDO NUNEZ LUIS	JGS	ING. IND.
232098	3	TIRADO SALGADO FELIX	RSP	ING. ELEC.
222897	3	BASURTO DELGADO MARIA	MJJS	LIC. ADMON
130199	1	MARTINEZ TIRADO JESUS	SCC	ABOGADO

PARA PASAR A LA TERCERA FORMA NORMAL:

ALUMNOS

MATRICULA	SEM	NOMBRE	ASESOR
167897	5	MALDONADO CORONA JUAN	JGS
114598	3	HERNANDEZ RUIZ MARTIN	RSP
127599	1	VALLE SILIS JORGE	EGC
213097	5	ARREDONDO NUÑEZ LUIS	JGS
232098	3	TIRADO SALGADO FELIX	RSP
222897	3	BASURTO DELGADO MARIA	MJJS
130199	1	MARTINEZ TIRADO JESUS	SCC

ASESORES

ASESOR	CARRERA ASESOR
JGS	ING. IND.
RSP	ING. ELEC.
EGC	TEC. AGRO.
MJJS	LIC. ADMON
SCC	ABOGADO

Se tienen dos tablas como resultado de separar la carrera del asesor de la tabla de los alumnos. En una modificación de la carrera de un asesor solo se afecta una celda. antes de pasar a tercera forma normal tendrian que afectar varias celdas, de no hacerlo caeriamos en inconsistencias.

El proceso de normalización hasta la tercera forma normal es de descomposición de una a dos o más. En adelante observaremos las operaciones que se realizan entre las tablas y comparamos con la realidad, si da lugar a filas que no son ciertas en la realidad deberíamos ajustarlo a los hechos.

GRUPOS

GRUPO	MAESTRO
2A	FRM
1A	FRM
1B	ATN
2A	FLC

IMPARTE

MAESTRO	MATERIAS
FRM	COMUNES
ATN	COMUNES
FLC	COMPUTACION

Si en la tabla Imparte deseara registrar las habilidades en distintas especialidades en un descuido solo se flexibiliza haciendo llave los dos atributos maestro y materias y poder registrar un maestro más de una vez.

IMPARTE

MAESTRO	MATERIAS
FRM	COMUNES
ATN	COMUNES
FLC	COMPUTACION
ATN	ELECTRONICA
FRM	ADMINISTRACION

Al relacionar las dos tablas por maestro se tendrían combinaciones irreales, aun cuando posibles. En este caso se debe asegurar que exista una tabla que limite los resultados a los que en realidad hay.

La quinta forma normal tiene el mismo sentido, de ajustar las operaciones a los hechos reales mediante una tabla ex profeso.

INTEGRIDAD REFERENCIAL

El diseño lógico obliga a observar la realidad para asegurarnos que la información almacenada pueda quedar inconsistente dentro del DBMS. Se ha superado usando procesos que se activan al afectar una tabla con una modificación, un borrado o una inserción.

La integridad referencial puede garantizar que las llaves ajenas de una tabla corresponden a la llave principal con la que se relacionan. Deberá establecerse si se pueden aceptar valores nulos en las llaves ajenas; contemplar lo que pasaría con las llaves ajenas si su llave principal fuera borrada y que pasaría con las llaves ajenas cuando la llave principal cambia.

El propio negocio puede dar lugar a establecer restricciones y estas se agregan a las propiamente de integridad referenicial

Las consideraciones que se pueden hacer al insertar, borrar y modificar son hacer las afectaciones en cascada, restringir y volver nulos los valores de las llaves ajenas.

Inserción restringida	IR
Borrado en cascada	BC
Restringir el borrado	BR
Hacer nulos al borrar	NB
Modificación en cascada	MC
Restringir la modificación	RM
Hacer nulos al modificar	NM

SISTEMA DE MANEJO DE BASES DE DATOS (DBMS)

STRUCTURED QUERY LANGUAGE (SQL)

IDENTIFICADORES

Tienen como función darle nombre a los objetos de la base de datos: base de datos, tablas, vistas, columnas, índices, procesos, reglas, etc.

Podrán usarse de uno a treinta caracteres de extensión, por omisión las mayúsculas o minúsculas serán consideradas iguales. El primer carácter debe ser una letra, los símbolos _ ó # . Cuando una tabla inicia con # es considerada temporal.

Dentro de una tabla las columnas deben tener nombre diferente, pero en el sistema cada diferente propietario no puede tener nombres repetidos en el mismo tipo de objeto. En caso de haber nombres duplicados se podrá hacer uso de la sintaxis convencional:

[[database.]owner.]table_name
[[database.]owner.]view_name

TIPOS DE DATOS USADOS EN LAS COLUMNAS

TIPOS	EXPLICACION
int	Valor numérico entero entre 2,147'483,647 y -2,147'483,648 . Almacenado en 4 Bytes
smallint	Valor numérico entero entre 32,767 y –32,768 Alamacenado en 2 Bytes
tinyint	Valor numérico entre 0 y 255
float	Valor numérico decimal entre 1 7 E -308 y 1 7 E +308. Almacenados en 8 Bytes
char(n)	Combinación de letras, símbolos y números. Hasta n=255 caracteres fijos
varchar(n)	Combinación variable de letras, símbolos y numeros. Hasta n=255 caracteres.
text	Colúmna variable de char de hasta 2,147'483,647 letras.
binary(n)	Almacenamiento de hasta 255 Bytes de dígitos binarios. No almacena hexadecimales.
varbinary(n)	Almacenamiento variable de hasta 255 Bytes de dígitos binarios
image	Columna de longitud variable desde 0 hasta 2,147'483,647
bit	Guarda un bit (0,1), cualquier otro numero es tomado como 1. No se acepta valor NULL
money	±922'337,203,685,477.5807 es almacenado en 8 Bytes
datetime	Guarda junto fecha y hora desde el año 1753 y 9999 El tiempo guarda un exactitud de 0.003 milisegundos
sysname	Es un tipo varchar(30) usado en tablas del sistema. No se permite valore NULL.
timestamp	Este valor es agregado en cada celda en cualquier consulta cuando se usa Browse. Cuando se usa la columna o el valor de cada celda deberán llamarse "timestamp".
user_type_name	Se puede definir con un nombre propio cualquier tipo usando los otros del sistema.

FUNCIONES

Para obtener el cálculo de promedios, determinar el mayor, menor o contar de una lista de cantidades de un resultado de una consulta SELECT.

FUNCION	PARAMETROS	RESULTADO
AVG	DISTINCT	Promedio de los [distintos] valores números
COUNT	DISTINCT	El número de [distinto] valores no nulos
COUNT	(*)	Cuenta los renglones
MAX	(expresion)	El valor mayor de la expresión
MIN	(expresion)	El valor menor de la expresión
SUM	DISTINCT	La suma de [distintos] valores

EXPRESIONES

[constant | column_name | function | (subquery) |
[{ arithmetic_operator | bitwise_operator | string_operator}
{ constant | column_name | function | (subquery)} . . .]

Los operadores aritméticos son: + - * / %, este último indica residuo de una división entera.

Los símbolos para operadores entre bits se usan entre dos operadores con excepción del NOT que sólo requiere uno.

&	AND
|	OR
^	EX OR
~	NOT

DEFINICION DE DATOS

CREATE DATABASE

Crea una base de datos registrandola en la tabla maestra.

CREATE DATABASE database_name
[ON [{ DEFAULT | database_device } [= size]
[, database_device [= size]] . . .]

USE

Al entrar en comunicación con el sistema se hace contacto con la base de datos maestra, para dirigirse a otra base de datos se toma esta instrucción para identificar la base con la cual se va a trabajar.

USE database_name

CREATE TABLE

Esta instrucción da origen a una tabla especificando la base de datos, el propietario, el nombre d e la tabla con sus columnas, cada una con su nombre y tipo; hasta 250 columnas puede especificarse. Para permitir valores nulos deberá especificarse NULL en las columnas elegidas.

```
CREATE TABLE [ database. [ owner. ] table _name
        ( column_name   datatype   [NOT NULL | NULL]
        [ , column_name   datatype   [NOT NULL | NULL] . . . ] )
```

CREATE INDEX

La creación permite responder una consulta según el orden especificado por el índice en forma más rápida. En caso de la tabla con muchos índices, su inserción y borrado son más lentos.

La columna tipo bit no se puede usar de índice No se requiere crear un índice para localizar información por cualquier columna según el orden requerido; su creación y destrucción es dinámica y manejada por el propio sistema.

CREATE DEFAULT

Es usado para guardar valores no especificados en la celda de una tabla con un nombre

CREATE DEFAULT [owner] default_name **AS constant_expresion**

sp_bindefault

Proceso del sistema, liga un nombre creado por CREATE DEFAULT con la columna de una tabla o con un tipo de datos definido por el usuario (objname).

sp_binddefault default_name , objname [, futureonly]

futureonly indica los próximos a ser creados, cuando ya existen algunos.

CREATE RULE

Es usado para guardar valores incluidos en un dominio especificado para una columna o un tipo de datos.

CREATE RULE [owner.] rule_name
AS condition_expresion

sp_bindrule

Proceso del sistema, liga una regla con una columna o con un tipo de datos definido por el usuario.

sp_bindrule rulename, objname [, futureonly]

futureonly no puede ser usado en columnas.

DROP DATABASE

Borra una o varias bases de datos.

DROP DATABASE database_name [, database_name]

DROP DEFAULT

Borra un tipo de datos especificado por el usuario para dar valor por omisión

DROP DEFAULT [owner.] default_name
[, [owner.] default_name ...]

DROP INDEX

Borra uno o varios índices de la base de datos.

```
DROP INDEX table_name.index_name
      [ , table_name.index_name . . . ]
```

DROP RULE

Borra uno o varios dominios para aplicar en las columna o tipos.

```
DROP RULE [ owner. ] rule_name
[ , [owner. ] rule_name ]
```

DROP TABLE

Borra una o varias tablas de la base de datos.

```
DROP TABLE [ [ database. ] owner. ] table_name
[ , [ database. ] owner. ] table_name . . . ]
```

MANIPULACION DE DATOS

SELECT

Toma uno o varios datos de una tabla en la base de datos.

```
SELECT [ALL | DISNTINCT] select_list
    [INTO [[database.]owner.]table_name]
    [FROM [[database.]owner.](table_name | view_name) [ HOLDLOCK]
    [[database.]owner.](table_name | view_name) [ HOLDLOCK ] . . . ]
    [WHERE search_conditions]
    [GROUP BY [ALL]
    aggregate_free_expession
    [ , aggregate_free_expession . . . ]
    [HAVING seach_conditions]
    [ORDER BY {[[[database ]owner ](table_name | view_name. )]
    column_name |
    select_list_number | expression} [ASC | DESC]
    [ , {[[[data_base ]owner ] (table_name | view_name )]
    column_name |
    select_list_number | expression} [ASC | DESC]] . . . ]
    [COMPUTE row_aggregate(column_name)
    [ , row_aggregate(column_name) . . . ]
    [BY column_name [, column_name] . . . ]]
    [FOR BROWSE]
```

INSERT

Agrega un nuevo renglón a una tabla. El orden por omisión en las columnas es el usado en CREATE TABLE. Si al especificar las columnas se omite alguna esta tendrá como valor NULL en el renglón insertado, a menos que tenga una regla o valor por omisión.

```
INSERT [ INTO ]
[ [ database. ] owner. ] { table_name | [ ( column_list ) ]
{ VALUES ( constant_expresion [ , constant_expresion ] . . .  |
select_statement }
```

DELETE

Borra renglones en una tabla.

```
DELETE [ [ database. ] owner. ] { table_name | view_name }
[ FROM [ [ database. ] owner. ] { table_name | view_name }
[ , [ database ] owner. ] { table_name | view_name } . . . ]
[ WHERE search_conditions ]

DELETE [ FROM ] [ [ database ] owner. ] { table_name | view_name }
[ WHERE search_conditions ]
```

UPDATE

Permite alterar uno o mas datos en una tabla

```
UPDATE [ [ database. ] owner. ] { table_name | view_name }
SET [ [ [database. ] owner. ] { table_name | view_name } ]
column_name1 = { expression1 | NULL }
[ , column_name2 = { expression2 | NULL } . . . ]
[ WHERE search_conditions ]

UPDATE [ [database ] owner ] { table_name | view_name }
SET [ [ [database ] owner ] { table_name | view_name } ]
column_name1 = { expression1 | NULL }
[ , column_name2 = { expression2 | NULL } . . . ]
FROM [ [database. ] owner. ] { table_name | view_name }
[ , [ [database ] owner. ] { table_name | view_name } ] . . . ]
[ WHERE search_conditions ]
```

CREATE PROCEDURE

Crea y guarda un procedimiento, compilado o evalúa en cada ejecución el plan para efectuarlo en forma óptima. Puede requerir de datos proporcionados por el usuario.

```
CREATE PROCedure [ owner. ] procedure_name [ ; number]
[ [ ( ] @parameter_name datatype [ = default ]
[ , @ parameter_name datatype [ = default ] . . . [ ) ] ]
[ WITH RECOMPILE ]
AS sql_statements
```

DEFINICION DE VISTA

CREATE VIEW

Establece una vista para ver datos de una o más tablas, mediante una consulta SELECT.

```
CREATE VIEW [ owner.] view_name
        [ (column_name [ , column_name ] . . . ]
AS select_statement
```

DROP VIEW

Borra una o varias vistas de una base de datos.

```
DROP VIEW  [ owner.] view_name
[ , [ owner.] view_name . . . ]
```

RESTRICCIONES DE INTEGRIDAD

CREATE TRIGGER

Crea un proceso usado para forzar la integridad referencial. Un disparador se ejecuta automáticamente cuando un usuario trata de modificar los datos en una tabla especifica.

```
 CREATE TRIGGER [ owner  ] trigger_name
ON [ owner.] table_name
FOR { INSERT | UPDATE | DELETE }
  [ [ , INSERT | UPDATE | DELETE ] . . . ]
AS sql_statements  |
IF UPDATE  ( column_name )
[ { AND | OR } UPDATE ( column_name ) . . . ]
```

DROP TRIGGER

Borra uno varios disparadores.

```
DROP TRIGGER [ owner.] trigger_name
```

[, [owner.] trigger_name . . .]

AUTORIZACION

GRANT

Asigna permisos a los usuarios.

```
GRANT { ALL | permission_list }
ON { table_name [ ( column_list ) ] | view_name
[ ( column_list ) ] | stored_procedure_name }
TO { PUBLIC | name_list }

GRANT { ALL | statement_list }
TO { PUBLIC | name_list }
```

REVOKE

Borra uno o varios permisos para usuarios.

```
REVOKE { ALL | permission_list }
ON { table_name [ ( column_list) ] | view_name [ ( column_list ) ]
stored_procedure_name
FROM { PUBLIC | name_list }

REVOKE { ALL | statement_list }
FROM { PUBLIC | name_list }
```

FRONTERAS DE TRANSACCIONES

BEGIN TRANSACTION

Inicia una serie de operaciones sobre la base de datos, todas ellas como si fuera una sola. En caso de suspenderse la actividad del sistema y no se ha terminado se deshace al restablecerse el sistema. Puede terminar con un COMMIT TRANSACTION o ROLLBACK TRANSACTION.

BEGIN TRANsaction [tansaction_name]

COMMIT TRANSACTION

Marca punto final a una serie de operaciones que son tratadas como una sola. Después de su ejecución no se pueden regresar automáticamente con ROLLBACK TRANSACTION.

COMMIT TRANsaction [transaction_name]

ROLLBACK TRANSACTION

Regresa a su estado anterior todas las afectaciones realizadas a partir de un BEGIN TRANSACTION o en caso de contener SAVE TRANSACTION lo que se realizó después de él. Esto permite dejar consistes los datos relacionados entre varias tablas o entre varios renglones de la misma tabla.

ROLLBACK TRANSACTION [transaction_name
| savepoint_name]

SAVE TRANSACTION

Permite dar por terminada una parte del proceso iniciado por un BEGIN TRANSACTION, cuando existe un ROLLBACK TRANSACTION pueden deshacerse todas las operaciones o solo las operaciones a partir del SAVE TRANSACTION.

TABLAS DEL SISTEMA

TABLAS MAESTRAS DEL SISTEMA DE BASE DE DATOS

SYSLOGINS
suid, status, acdate, totcpu, totio, spacelimit, timelimit, resullimit, dbname, name, password

SYSPROCESSES
spid, kpid, status, suid, hostname, program_name, hostprocess, cmd, cpu, physical_io, memusage, blocked, dbid, uid, gid

SYSCURCONFIGS
config, value, comment, status

SYSMESSAGES
error, severity, dlevel, descripcion

SYSUSAGES
dbid, segmap, lstart, size, vstart

SYSDATABASES
name, dbid, suid, mode, status, version, crdate

SYSCONFIGURES
config, value, comment, status

SYSDEVICES
low, high, status, cntrltype, name, phyname

SYSLOCKS
id, dbid, page, type, spid

TABLAS DE ADMINISTRACION PARA CADA BASE DE DATOS

SYSKEYS
id, type, depid, keycnt, size, key1, key2, . . ., key8, depkey1, depkey2, . . ., depkey8

SYSOBJECTS
name, id, uid, type, userstat, sysstat, indexdel, schema, refdate, crdate, expdate, deltrig, instrig, updtrig, seltrig, category, cache

SYSDEPENDS
id, number, depid, depnumber, depdbid, depsitieid, status, selall, resultobj, readobj

SYSSEGMENTS
segment, name, status

SYSLOGS
xactid, op

SYSTYPES
tdefault, usertype, variable, allownulls, type, length, uid, domain, name, printfmt

SYSCOLUMNS
id, number, colid, status, type, length, offset, usertype, cdefault, domain, name, printfmt

SYSINDEXES
name, id, indid, dpages, reserved, used, rows, first, root, distribution, usagecnt, segment, status, rowpage, minlen, maxlen, maxirow, keycnt, keys1, keys2

SYSPROCEDURES
type, id, sequence, status, number

SYSUSERS
suid, uid, gid, name, environ

SYSPROTECTS
id, uid, action, protecttype, columns

SYSCOMMENTS
id, number, colid, texttype, language, text

SYSALTERNATES
suid, altsuid

BIBLIOGRAFIA

VETTER M. Estrategy for Data Modeling. John Wiley Sons. 1987.

Martin James. Organización de las Bases de Datos. Prenice Hall. 1977.

Groff, James R. y Paul N. Wienberg. Aplique SQL. McGraw Hill 1991.

Robbins P Stephen y David A. De Cenzo. Fundamentos de Adminsitración. Prentice Hall Hispanoamericana S.A. 1996.

Pressman S. Roger Ingenieria del Softmare. McGraw Hill. Tercera edición. 1992

FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

CURSOS INSTITUCIONALES

MÓDULO BÁSICO DE TECNOLÓGIAS DE LA INFORMACIÓN Y BIOESTADÍSTICA PARA EL ISSSTE

Herramientas tecnológicas (CI 371)

TEMA
PROGRAMACIÓN

28 de octubre del 2002 al 15 de enero del 2003

Instructor: Ing. Rodolfo González Maldonado
Palacio de Minería
I S S S T E noviembre 2002

INTRODUCCION

El presente material, sirve como soporte y complemento al curso presencial de Programación en Visual Basic

Los temas cubiertos en el curso se detallan en este material, permitiendo así, que el material sea utilizado tanto como material dentro del curso, como de consulta y referencia posterior.

El material cubre, el como realizar programas en ambiente Windows bajo el Lenguaje Visual Basic

El presente curso, da las bases a los participantes, de como elaborar un sistema

OBJETIVO

El participante, aprenderá y conocerá los principios de la programación, así como la forma desarrollar sistemas en ambiente Windows

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN AL AMBIENTE DE PROGRAMACIÓN VISUAL BASIC

CÓMO DESARROLLAR UNA APLICACIÓN DE VISUAL BASIC

El primer paso para desarrollar una aplicación en Visual Basic es planear lo que ve el usuario; en otras palabras, diseñar las pantallas. ¿Qué menús se desean? ¿Cómo ve la ventana en la que se ejecuta la aplicación? ¿Cuántas ventanas debe haber? ¿Debe poder modificar el usuario el tamaño de las ventanas? ¿Dónde se colocarán *los botones de ordenes* los botones sobre los que el usuario hará «clic» para activar las aplicaciones? ¿Tendrá la aplicación sitios para introducir texto *(cajas de texto)?* En Visual Basic los objetos que sitúa el diseñador de un programa en una ventana se denominan *controles.*

Lo que hace que Visual Basic sea diferente de cualquier otra herramienta de programación es la facilidad con la que se puede diseñar una pantalla Se puede dibujar, literalmente, la interfaz de usuario, de la misma forma que se utiliza un programa de dibujo Además, una vez se ha terminado de dibujar la interfaz, los botones de órdenes cajas de texto y otros controles que se han colocado en una ventana en blanco, reconocerán automáticamente acciones del usuario, tales como el movimiento del ratón y los clics de los botones Visual Basic incluye una característica de diseño de menús que hace que la creación de menús desplegados o normales sea instantánea.

Tan sólo después de diseñar la interfaz de usuario es cuando se empieza a hacer algo que se parezca a programar. Los objetos en Visual Basic *reconocerán* sucesos como los clics del ratón; la forma en que los objetos respondan dependerá del código que se escriba. Se necesitará escribir código para que los controles respondan a los sucesos. Todo esto hace que Visual Basic sea diferente de la programación convencional.

Los programas en los lenguajes de programación convencionales se ejecutan de arriba abajo. En los antiguos lenguajes de programación la ejecución comienza en la primera línea y se desplaza con el flujo del programa a las distintas partes según se necesite Un programa en Visual Basic funciona de un modo totalmente diferente. El núcleo de un programa en Visual Basic es un conjunto de diferentes partes de código que son *activadas* por, y que solamente responden a, los sucesos que se les han indicado que reconozcan. Esto es un avance fundamental Ahora,

NOTAS:

en lugar de diseñar un programa que haga lo que el programador piense que debe hacer. el usuario tiene el control.

La mayor parte del código de programación en Visual Basic indica al programa el modo de responder a determinados sucesos, como el clic del ratón. en lo que en Visual Basic se denominan *procedimientos de suceso*. Esencialmente. cualquier cosa ejecutable en Visual Basic es, o bien un procedimiento de suceso, o es utilizado por un procedimiento de suceso para. ayudar al procedimiento.

Más aún, Visual Basic hace sencilla la creación de grandes programas mediante las modernas *técnicas modulares de Programación*. Esto significa que se puede dividir un programa en módulos, más sencillos de manejar y, por tanto, menos sensibles a los errores (un *módulo* es una parte relativamente pequeña y manejable de código de programación). Idealmente, los módulos realizan una sola tarea y tienen una interfaz bien definida con el resto del programa, por lo que puede ser codificado y verificado independientemente. De este modo, es posible concentrar la atención en el modo en que cada módulo realiza su labor, y en la forma en que las piezas del programa se comunican con las demás dentro de la aplicación.

Visual Basic también proporciona una sofisticado gestión de errores para la siempre demasiado frecuente tarea de evitar que los nuevos usuarios dinamiten una aplicación. *(Dinamitar una aplicación* es jerga de informática para finalizar un programa abrupta y anómalamente., *Bugs* es también jerga informática que define los errores de programación que normalmente provocan el dinamitado.) Visual Basic dispone de un editor / intérprete inteligente que normalmente detecta, e incluso sugiere a vèces los cambios necesarios para corregir los errores tipográficos y de programación que son frecuentes cuando se empieza a crear una aplicación. También tiene un extenso sistema de ayuda en linea para referencia rápida cuando se desarrolla una aplicación.

Cuando se inicia Visual Basic se presenta una pantalla con *el copyright*. indicando la persona para la cual es válida la licen- cia de la copia de Visual Basie. Después de un breve retraso, automáticamente se muestra el entorno de Visual Basic.

La barra de títulos

La barra de titulos es la barra horizontal situada en la parte superior de la pantalla; contiene el nombre de la aplicación y es común a todas las aplicaciones de Windows de Microsoft Las interacciones entre el usuario y la barra de titulos se gestionan por Windows. no por la aplicación. Todo lo que hay debajo de la barra de títulos y de menús en una aplicación Windows se denomina *área de cliente*. La aplicación es integramente responsable del aspecto, sensación y respuesta de los objetos que sitúen en este área.

NOTAS:

La barra de menús

Seleccionar elementos de los menús desplegables situados en la *barra de menús es* uno de los sistemas más frecuentes para desencadenar la potencia de una aplicación Windows. Lo mismo es cierto para Visual Basic. En Visual Basic, la barra de menús proporciona las herramientas necesarias para desarrollar, probar y archivar la aplicación El menú File (Archivo) contiene las órdenes para trabajar con archivos que incluye la aplicación. El menu Edit (Edición) contiene la mayoría de las herramientas de edición que ayudan a escribir el código que activa la interfaz que se diseña para la nueva aplicación, incluyendo las herramientas de edición de búsqueda y sustitución. El menú View (Ver), proporciona acceso rápido a todas las partes del programa El menú Run (Ejecutar) permite verificar la aplicación según se va desarrollando. El menú Debug (Depurar) da acceso a las herramientas utilizadas para depurar un programa.
El menú Options (Opciones) permite controlar el entorno de Visual Basic. El menú Windows (Ventana) proporciona acceso rápido a las diferentes ventanas que configuran el entorno Visual Basic. Permite controlar los colores utilizados mientras se desarrolla la aplicación. Por último, el menú Help (Ayuda) se utiliza para acceder al detallado sistema de ayuda en línea incluido en Visual Basic.

La barra de herramientas

La barra de herramientas está justo debajo de la barra de menús. Como es cada vez más frecuente en las aplicaciones Windows. Microsoft añade barras de herramientas de iconos para permitir activar las tareas más comunes sin necesidad de utilizar los rnenús. Dado que cada elemento de la barra de herramientas tiene una combinación de teclado para la misma tarea. la elección de un sistema u otro para activarlo es cuestión de gustos.
De izquierda a derecha, aquí se describe lo que realiza cada icono de la barra de herramientas:

New Form (Nuevo formulario). Este botón permite añadir un nuevo formulario (una nueva ventana personalizable) a un proyecto Hace la misma función que New Forrn en el menú File

New Module (Nuevo módulo). Este botón permite abrir un nuevo módulo para código especifico de programación. Su utilización se estudia en el Capítulo 13. Es el mismo elemento que New Module en el menú File (ALT+F+M)

Open Project (Abrir proyecto). Este botón permite abrir un proyecto existente de Visual Basic.

Save Project (Archivar proyecto). Permite archivar un proyecto de Visual Basic

Menu Design (Diseño de menú). Se utiliza para diseñar menús.

Activate Properties Windows (Activar ventana de propiedades). La ventana de propiedades se encuentra en la parte inferior derecha de la pantalla. Se utiliza para modificar la forma. tamaño y color por omisión de un objeto de Visual Basie

NOTAS:

Run (Ejecuta). Este elemento ejecuta las aplicaciones. Una vez diseñada la aplicación, es lo rnisrno que elegir Run dej men-á Run (el atajo es F5).

Break (Pausa). Sirve para detener temporalmente un programa en ejecución (los programas se pueden continuar eventualmente utilizando la herramienta Run o SHIFT+F5). Realiza la misma función que el elemento Break del menú Run o mediante la combinación de teclado CTRL+BREAK

End (Fin) Finaliza un programa en ejecución. En el mismo efecto que el elemento End en el menú Run.

Breakpoint (Punto de pausa). Una herramienta de depuración: véase Capitulo 11. Esto coloca un punto de detención temporal en un lugar específico del programa Disponible también en el menú Debug, el atajo es F9.

Instant Watch (Observación instantánea). También es una herramienta de depuración que se explica en el Capítulo 1 1. Este elemento permite tomar una instantánea de lo que sucede en las diferentes partes dej programa en ejecución. También se encuentra disponible en el menú Debug, y su atajo es SHIFT+F9.

Command (Orden de llamada). Muestra una lista de las llamadas a procedimientos actuales. Es una herramienta de depuración.

Single Step (Paso a paso) . Sirve para desplazarse por el programa en pasos de una línea. Es también una herramienta de depuración que se explica en el Capitulo 11. Está disponible en el menú Debug o mediante el atajo F8.

Prcedure Step (Paso de procedimiento) . Se utiliza cuando el programa se hace más sofisticado y divide tareas específicas en procedimientos diferentes También sirve para desplazarse por el programa paso a paso. pero considera a los procedimientos como un solo paso.

La caja de herramientas

Situada a la izquierda de la pantalla, justo debajo de la barra de herramientas. la caja de herramientas contiene las 22 herramientas básicas para desarrollar aplicaciones.

La ventana del formulario inicial

La ventana del formulario inicial ocupa la mayor parte del centro de la pantalla. En ella es donde se personaliza la ventana que verán los usuarios. La documentación de Visual Basic utiliza el *término form (formulario)* para una ventana personalizable.

NOTAS:

La ventana de proyecto

Dado que es muy común en las aplicaciones de Visual Basic compartir código o formularios personalizados, Visual Basic organiza las aplicaciones en lo que denomina *projects (proyectos).* Cada proyecto puede tener varios formularios, y el código que activa los controles de un formulario es archivado con el formulario en archivos separados El código general de programación compartido por todos los formularios de una aplicación puede ser dividido en varios módulos, que también se archivan separadamente. Situado en la parte derecha de la pantalla y parcialmente tapado por la ventana de formulario, está la *ventana de proyecto* que contiene un listado de todas las ventanas personalizabas y de código general (módulos) que conforman una aplicación.

Opciones del Menú File

La mayoria de los elementos del menú File son útiles únicamente cuando se ha comenzado a desarrollar aplicaciones. A continuación se incluye una breve descripción de cada uno de los elementos, lo que debe ser una ayuda para una primera orientación.

New Project (Proyecto nuevo). La opción New Project archiva el proyecto actual Si se ha realizado algún cambio desde la última vez que se archivó aparece un cuadro de diálogo preguntando si se desea archivar el trabajo:

Open Project (Abrir proyecto). La opción Open Project permite trabajar con una aplicación Visual Basic existente.

Saye Project (Archivar Proyecto). El elemento Saye Project archiva todos los archivos del proyecto actual. Los formularios que conforman la aplicación se archivan en un formato binario que no puede ser leído por otras aplicaciones. También se puede archivar esta información así como la del código en formato ASCII, que puede ser leido por la mayoría de los procesadores de texto. (Los archivos almacenados en formato ASCII son denominados archivos de texto.)

SaveProject As(Archivar proyecto como). Esta opción despliega un cuadro de diálogo que permite archivar el proyecto con un nombre diferente. También se puede utilizar esta opción para hacer copias de seguridad del proyecto en un disco diferente, o para archivar diferentes versiones del proyecto.

New Form (Formulario nuevo). Este elemento añade nuevas ventanas a la aplicación

New MDI Form (Nuevo formulario MDI) Un formulario MDI *(Multiple Document Interface.* Interfaz para documentos múltiples) se utiliza para que ciertas ventanas se comporten subordinadas a la ventana principal.

NOTAS:

New Module (Módulo Nuevo). Se utiliza la opción New Module para añadir código de programación que se comparte por todas las partes de la aplicación que se desarrolla En Visual Basic, el código se une a una ventana (formulario) especifico a menos que se sitúe en un módulo.

Add File (Añadir archivo). La opción Add File abre un cuadro de diálogo que permite incorporar trabajo anterior a una aplicación en curso. Se pueden añadir formularios terminados, código de propósito general (módulos), e incluso archivos que añaden nuevas posibilidades a Visual Basic. A éstos se les denomina *archivos de control personalizados.*

Remove File (Borrar archivo). Esta opción se utiliza para borrar la parte de la aplicación Visual Basic en la que se está trabajando. Esta opción no borra la parte de la aplicación del disco en el que está almacenada. Para eso se necesitará utilizar el Administrador de archivos de Windows, o bien órdenes desde el DOS.

Save File (Archivar). La opción Save File archiva el formulario activo o el módulo (código de propósito general) en disco. La primera vez que se selecciona esta opción, Visual Basic abre una ventana de diálogo idéntica a la de Save File As

Save File As (Guardar archivo como). Esta opción despliega un cuadro de diálogo que permite archivar el formulario actual o el módulo (código de propósito general) en disco, posiblemente con un nombre diferente. Se puede utilizar esta opción para realizar copias de seguridad de una parte especifica del proyecto en un disco diferente, o archivar diferentes versiones del proyecto. También se puede utilizar esta opción en el caso de que se considere que parte de la aplicación actual puede ser útil en ulteriores aplicaciones.

Load Text (Abrir texto). Esta opción extrae en Visual Basic código escrito en otro lenguaje o con otro editor de texto. Este texto puede sustituir otro anterior a unirse a él

Save Text (Archivar texto). La opción Save Text archiva el código del formulario o códigos actuales en formato texto (ASCII). Esto hace que pueda ser leído por otros editores de texto u otros programas. Como indican los puntos suspensivos, la selección de esta opción abre un cuadro de diálogo en el que se introduce el nombre de archivo que se desea que tenga el código archivado.

Print (Imprimir). Esto permite la impresión del formulario, su código o el módulo con los que se está trabajando, o todos los formularios o módulos de la aplicación.

Make EXE File. La opción Make EXE File abre un cuadro de diálogo que permite crear aplicaciones en Visual Basic que se pueden ejecutar en el entorno Windows independientes de Visual Basic.

NOTAS:

El listado Most Recently Used (Más recientemente utilizado): Realiza un seguimiento de los últimos cuatro proyectos abiertos en Visual Basic. Al hacer clic con el ratón sobre uno de los archivos listados, Visual Basic recupera el proyecto automáticamente. Esto permite que la vuelta al trabajo ya realizado sea rápida.

Exit (Salida). Seleccionar esta opción es el modo normal de abandonar Visual Basic. Si se ha realizado algún cambio al proyecto activo, Visual Basic pregunta si se desean archivar los cambios antes de finalizar la sesión.

MODIFICACIÓN DE UN FORMULARIO

Por ahora, nos ceñiremos al formulario denominado Form1 que se encuentra en el centro de la pantalla. Es preciso estar completamente familiarizado con los métodos para cambiar el tamaño y la posición de este formulario antes de continuar. En muchas de las aplicaciones de Visual Basic el tamaño y la forma del formulario al acabar el diseño (normalmente denominado *design time)* es el tamaño y la forma que verá el usuario cuando lo ejecute *(run time).*

Esto no quiere decir que Visual Basic no permita cambiar el tamaño y la posición de los formularios cuando se ejecute el programa. Una de las propiedades esenciales de Visual Basic es la posibilidad de realizar cambios dinámicos en respuesta a los sucesos del usuario.

Propiedades de un Formulario (Properties)

El tamaño y la posición de un formulario son ejemplos de los que Visual Basic llama propiedades *(properties)* de un formulario. Dichas propiedades pueden cambiarse desde la ventana Properties, que se visualiza al pulsar F4.

Obsérvese que justo debajo de la barra de título hay una línea que dice «Form1 Form Esta línea indica el nombre y el tipo de objeto sobre el que se esta trabajando. En este caso, el nombre es el existente por omisión, Form1. (Cuando se tengan otros controles en los formularios, al hacer clic en la flecha hacia abajo que se encuentra en el extremo derecho de la línea se verá una lista de los controles, véase a continuación la línea resaltada que dice Caption en la primera columna y Form1 en la segunda columna. La palabra *Form1* debajo de la línea Form! Form en la parte de arriba del cuadro es la leyenda del formulario. Este cuadro de texto indica *el valor* actual de la propiedad Caption.

Conforme se vaya uno desplazando por la ventana Properties mediante las teclas del cursor o el ratón, el texto de esta caja cambia para reflejar la propiedad correspondiente a la línea que esté resaltada.

NOTAS:

Si la linea resaltada es Caption, lo que se introduzca en el cuadro de texto se convertirá inmediatamente en la nueva leyenda del formulario. Este cuadro de texto se denomina Cuadro de parámetros *(Settings box)* de la ventana Properties. Por ejemplo, tecléese **Primera Ventana** en el cuadro de parámetros correspondiente a Caption. Puede verse que conforme se va tecleando la leyenda del formulario cambia instantáneamente reflejando lo que se está escribiendo.

Sin embargo, si no se procede exactamente en el orden indicado, o si se ha hecho clic sin darse cuenta en un sitio que no se quería, este método no funcionará bien

Las propiedades más comunes de un formulario

Seguidamente se expone una lista con algunas de las propiedades más comunes de los formularios.

Border Style (Estilo de borde). Este es un ejemplo de propiedad que ofrece sólo un pequeño numero de opciones. Debido a ello, la flecha que hay a la derecha del lugar donde se paramétriza esta propiedad está activada (oscurecida). Puede elegirse entre cuatro valores para esta propiedad. El valor por omisión, 2, permite al usuario establecer el tamaño y la forma del formulario por medio de los puntos calientes situados en los límites del formulario. Si se cambia este parámetro a 1 (llamado *fijo),* el usuario no será capaz de modificar el tamaño de la ventana Todo lo que el usuario puede hacer es minimizar o maximizar la ventana (a menos, por supuesto, que estén desactivadas estas opciones cuando se diseñe la aplicación).

Si se pone el parámetro 0, la aplicación no mostrará ningún tipo de borde, por lo que no habrá posibilidad de minimizar, maximizar o emplear botones de control de cuadro. Debido a esto, el formulario creado sin borde no se puede mover ni cambiar de tamaño o de forma. Este parámetro es muy útil cuando se quiere que un formulario permanezca inalterable.

El cuarto parámetro, 3 (llamado *fijo doble)* no se emplea frecuentemente en los formularios ordinarios, pero sí en los cuadros de diálogo. Proporciona un borde no modificable (no tiene puntos calientes) que tiene un grosor dos veces superior al normal

Caption (Leyenda). La propiedad Caption establece el título del formulario También es el título que usa Microsoft Windows para el icono de la aplicación cuando el usuario la minimiza.

Control Box (Cuadro de control). Es otra propiedad que sólo surte efecto cuando el usuario ejecuta la aplicación Como en cualquier aplicación de Microsoft Windows, el cuadro de control se encuentra en la esquina izquierda de la barra de títulos. Al hacer clic sobre el cuadro se presenta una lista de las tareas comunes de la ventana, tales como maximizarla, minimizarla y cerrarla, así como sus teclas equivalentes si existen

NOTAS:

Sólo existen dos opciones, o tener el cuadro de control o no tenerlo. Por tanto sólo hay dos posibles parámetros y, por ello, el cuadro de listas a la derecha del área de parámetros está activado. Es de señalar que si la aplicación no tiene un cuadro de control, un usuario que no tenga ratón estará en apuros. Será incapaz de minimizar, maximizar o cerrar la aplicación. Por lo general no es una buena idea el quitar los cuadros de control.

Enabled (Activado). Esta es otra propiedad que no hay que cambiar de manera fortuita. Si se pone en False, el formulario no responderá a ningún suceso. Normalmente se conmuta esta propiedad de True a False como respuesta a algún suceso. Para hacer que los formularios respondan dinámicamente es preciso escribir código. La ventana Properties se emplea con más frecuencia para establecer las propiedades estáticas de los objetos.

FontName (Nombre de la fuente). La propiedad FontName es otro ejemplo de parametrización con un número finito de opciones. La lista muestra las fuentes (tipos de letra) disponibles. Esta lista depende de las fuentes que Windows reconozca. La fuente que se elija será la que se emplee para presentar información en el formulario, no afecta a la leyenda del formulario.

FontSize (Tamaño de la fuente). Esta propiedad determina el tamaño del texto presentado en el formulario. El tamaño que se puede elegir depende de la fuente que se haya elegido por medio de la propiedad FontName.

FontBold, Fontitalic, FontStrikethrough, FontUnderline, Font Transparent (Negrita, Cursiva, Tachada, Subrayada, Transparente). Al igual que en FontName, las demás propiedades de las fuentes afectan a lo que se presenta en el formulario. Estas propiedades pueden ponerse en True o en False y pueden normalmente combinarse como se quiera.

Por ejemplo, se puede tener un texto en negrita, cursiva y que esté tachado. La propiedad Font Transparent determina si se muestra o no el fondo a través de los caracteres. A menudo la conmutación entre estas propiedades se realizará como respuesta a algún suceso. Como cualquier actividad de Visual Basic que suscita reacción, la conmutación requiere que se escriba código.

Height, Width (Alto, Ancho). Son dos propiedades interesantes, y no sólo porque puedan establecerse de dos formas distintas. Son ejemplo de propiedades que usan la escala *twips* de Microsoft Windows para medir el tamaño de los objetos complicados. Hay 1.440 *twips* en una pulgada (567 en un centímetro). El término viene realmente de *one twentieth of a point*, un veinteavo de punto. Los puntos son una medida corriente en imprenta, este libro está compuesto con tipos de 11 puntos. Moviéndose a lo largo de la barra de propiedades, elíjase la propiedad Height. Obsérvese que el valor actual es 4 425 *twips* (lo que significa que el tamaño

NOTAS:

por omisión del formulario cuando se imprima será ligeramente mayor que 3 pulgadas. o ligeramente menor que 8 centímetros).

Precaución. Los twips miden el tamaño que tendrá el objeto si se imprimiera; no corresponde exactamente al tamaño del objeto en la pantalla. Por ejemplo. en un monitor de 15 pulgadas de diagonal, el tamaño por omisión de un formulario es aproximadamente 4.5 pulgadas.

Obsérvese el extremo derecho de la barra de herramientas

Se puede apreciar que el último cuadro tiene el valor 7.485 x 4.425. (Los números que se verán dependen de la resolución de la pantalla). Esto indica el valor actual en *twips* del ancho y la altura del formulario. Si ahora se arrastra el formulario para modificar sus dimensiones. se verá que conforme se va variando el tamaño del mismo los valores del extremo derecho de la barra de herramientas cambian para reflejar el nuevo tamaño. Si se mira en la ventana Properties. se verá que los valores de estas propiedades también han cambiado.

El empleo del ratón y el arrastre para establecer la altura y la anchura de un formulario es, por supuesto, el método menos preciso que se pueda emplear. Afortunadamente, puesto que Height y Width son propiedades, pueden cambiarse sus valores directamente por medio de la ventana Properties.

Todo lo que hay que hacer es introducir el valor que se desee en el cuadro de parámetros. Por ejemplo, si se cambia la anchura a 3.743 *twips* se cortará la ventana por la mitad (es como reducir a la mitad lo que la pantalla puede presentar)

Al igual que con la propiedad Caption. cualquier cambio que se efectúe en la altura y en la anchura del formulario surte efecto inmediatamente. no hay que esperar a que se ejecute la aplicación.

Nota. A menos que se desactive el borde cambiando la propiedad BorderStyle, el usuario puede modificar el tamaño y la forma de los diversos formularios de la aplicación sin tener en cuenta los tamaños que se establecieran cuando se diseñaron.

Left, Top(Izquierda, Arriba). Los parámetros Left y Top determinan la distancia entre la parte izquierda o superior del formulario y la pantalla. Si el valor de la propiedad Top se hace 0, el formulario que se está diseñando quedará a ras del borde superior Si se hace 0 la propiedad Left, el formulario lindará con el borde izquierdo de la pantalla.

NOTAS:

Estos parámetros funcionan de una manera idéntica a las propiedades Height y Width Pueden cambiarse de las mismas dos maneras Se miden también en *twips* y causan efecto cuando se cambian los valores, no cuando el usuario ejecuta la aplicación

MousePointer(Puntero del ratón). Es una entretenida (y a veces útil) propiedad que establece la forma del puntero del ratón. El valor por omisión es 0, pero como indica la lista desplegable, hay otros 12 valores. Si se pone el valor 4, por ejemplo, el puntero del ratón se convierte en un cuadrado dentro de un cuadrado bastante bonito. El valor que más se empleará es el 11. Este valor cambia el puntero del ratón por un reloj de arena y, como en otras aplicaciones de Microsoft Windows, se emplea para indicar al usuario que tiene que esperar hasta que la computadora termine lo que está haciendo.

Visible (Visible). Esta es otra de las propiedades que es peligroso cambiar por error Si se pone el valor de esta propiedad en False, el formulario no estará visible (y, por tanto, será difícil que el usuario lo pueda manejar).

La ventana de código

Ahora se verá cómo diseñar un formulario para que responda a un clic del ratón Si se hace un doble clic en cualquier parte en blanco de Form1(o cualquier otro nombre que tenga el formulario), se abrirá una nueva ventana. La acción de efectuar un doble clic en el formulario abre la *ventana de código.* Aquí es donde se introduce el código que indica a Visual Basic cómo responder al suceso.

En la parte superior de la pantalla se encuentran dos cuadros marcados con *Object* (Objeto) y *Proc* (Procedimiento). Si se hace clic en la flecha que se encuentra al lado del cuadro Object se despliega el *cuadro de lista de Objetos.*

Este cuadro contiene todos los objetos del formulario. Todavía hay que añadir a este formulario cuadros de texto, botones de órdenes u otros controles, por lo que en este cuadro no aparece ningún objeto salvo el formulario mismo

Si se despliega el *cuadro de lista de procedimientos* (el señalado mediante Proc:) a la derecha se verá la lista de todos los sucesos que puede reconocer el objeto Visual Basic que se encuentra en el cuadro Object. Un formulario puede reconocer más de 20 sucesos: el cuadro de lista de procedimientos tiene 23 elementos en él.

Este código es un ejemplo *de plantilla de procedimiento para un suceso.* Como cualquier plantilla, proporciona un marco en el que trabajar. El suceso Form-Load se ejecuta siempre que Visual Basic lleve a la pantalla un formulario. Se emplea para establecer, por medio de código, las propiedades iniciales del formulario.

NOTAS:

Por ahora lo que se quiere es introducir el código que indique a Visual Basic cómo responder a un clic sobre el formulario. Para ello hay que hacer aparecer la plantilla del procedimiento Form-Clic. Esto se logra yendo al cuadro de lista procedimientos *(Proc)* y haciendo clic en la flecha hacia abajo. Como en cualquier cuadro de lista, se accede rápidamente a cualquier elemento pulsando la primera letra del mismo. Para cerrar el cuadro de lista se pulsa Esc. El cursor se situará sólo en la línea en blanco entre «Sub» y «End Sub.»

Un simple procedimiento de suceso

Supongamos que se desea escribir el código necesario para que Visual Basic responda con un mensaje a un clic del ratón. Se pueden usar para introducir el código las teclas normales de edición de Windows.

Si el cursor no se encuentra en la línea en blanco entre Sub y End Sub, hay que llevarlo allí haciendo un clic en la línea en blanco. Se pulsa la BARRA F-SPACIADORA dos veces (esta identación mejorará la legibilidad) y se teclea **Print «Ha hecho clic con el ratón una vez».**

Si se pulsa ahora F5 se ejecuta la aplicación. Si tan pronto como aparezca el formulario se mueve el ratón hasta que su puntero quede dentro del formulario y se hace clic una vez, se vera el texto especificado escrito sobre el formulario.

Supervisión de sucesos múltiples

Visual Basic está siempre pendiente de la pantalla de la aplicación por si ocurriera un suceso, pero, a menos que se haya escrito código para él, nada ocurrirá. Por jemplo, se puede añadir más código para que monitorice (e imprima) algo cuando e1 usuario haga un doble clic.

Para ello, primero hay que terminar el programa anterior seleccionando el menú Run y haciendo clic sobre End (o sobre la herramienta de terminar, la que parece el botón de parada de un reproductor de *cassetes).* Luego se abre la ventana de código (si no está ya abierta) mediante un doble clic en cualquier zona en blanco del formulario. Visual Basic presenta ahora el código para el suceso de un simple clic. Supongamos que se desea escribir código para el suceso de un doble clic. Se lleva el ratón a la flecha que despliega el cuadro de lista de procedimientos y se hace clic, aparecerá una lista de sucesos para el objeto seleccionado a la izquierda.:

Pulsando la letra D se selecciona el suceso doble clic. El procedimiento del suceso Clic está ahora en negrita, que es como Visual Basic indica que ya existe un procedimiento para un determinado suceso.

NOTAS:

Se abre una nueva plantilla de suceso para el doble clic Exactamente como antes, se puede escribir entre las líneas inicial y final de la plantilla. Por ejemplo, **Print «¡Dije un clic, no dos!».**

Se puede observar que en la pantalla aparecen las dos líneas de texto. Esto es debido porque Visual Basic, al supervisar la ejecución de un doble clic, detecta el primer clic y activa también el código correspondiente a este suceso

Si todo lo que se necesita es borrar la pantalla antes de presentar el segundo mensaje, basta con escribir la orden Cls en la primera línea del procedimiento para el doble clic, con lo cual se borrará cualquier texto o gráfico del formulario.

Las órdenes Print y Cls son ejemplos de lo que Visual Basic denomina métodos En pocas palabras, se puede decir que métodos son las declaraciones que afectan a lo que hacen los objetos de Visual Basic (en contraposición a propiedades que afectan a lo que *son)*

Se puede usar otra sintaxis para las órdenes Print y Cls. Esta sintaxis se emplea en otros objetos de Visual Basic y sigue el formato general *Objeto.Método.* Esta sintaxis usa el nombre del objeto seguido de un punto y del nombre del método, seguido (en su caso) de lo que el método ha de hacer:

Objeto.Método Lo que hay que hacer

La caja de herramientas

Como su propio nombre indica, la caja de herramientas es un conjunto de herramientas que se emplean para embellecer un formulario en blanco con los controles necesarios para los sofisticados programas Windows. La versión estándar de Visual Basic viene con 22 herramientas diferentes. Una de las características más apasionantes de Visual Basic es su ampliabilidad.

La edición estándar de Visual Basic viene con tres controles personalizables: un control de rejilla para manejar los datos tabulados, otro para añadir cuadros de diálogo normales y un control OLE *(Object Linking and Embedding)* para comunicarse con otras aplicaciones Windows.

Una vez instalado un control personalizado, forma parte de la versión de Visual Basic exactamente igual que las herramientas suministradas por Microsoft con la versión estándar de Visual Basic.

La caja de herramientas se encuentra normalmente situada en el extremo izquierdo de la pantalla, pero no es necesario que esté visible todo el tiempo.

NOTAS:

Command Button (Botones de órdenes). Estos botones de órdenes reciben a veces el nombre de pulsadores. La idea es que cuando el usuario lleva el cursor del ratón sobre el botón de órdenes y hace clic, ocurrirá algo interesante (y ocurrirá, una vez que se escriba el procedimiento de suceso que diga a Visual Basic cómo responder al clic del ratón) Cuando se hace clic sobre un botón de órdenes, da la impresión de haber sido pulsado. Esta ilusión óptica es debida a la figura usada por Visual Basic para los botones de órdenes y es intrinseca a Microsoft Windows.

Image Controls (Controles de imágenes). Este tipo de control es uno de los dos que pueden emplearse para presentar imágenes. Este control es el que emplea el menor número de recursos de Windows para hacerlo. (Las cajas de figuras pueden hacer más, pero empleando más recursos.) Se pueden emplear los controles de imágenes como botones de órdenes, ya que también reconocen el suceso de hacer clic. Sin embargo, al contrario de los botones, los controles de imágenes no proporcionan al usuario ningún retorno cuando los pulsan, a menos que se programen así en el proyecto.

Text Boxes (Cajas de texto). Estas cajas también reciben a veces el nombre de campos de edición. Se pueden emplear las cajas de texto para presentar uno o para aceptar la entrada del usuario. La mayoría del código que se escribe para las cajas de texto es para procesar la información que los usuarios introducen en ellas. Cuando se introduce información en una de estas cajas, se puede disponer de todas las herramientas ordinarias de edición de Windows como por ejemplo cortar y pegar..Estas cajas permiten el formato de textos y pueden tener barras de desplazamiento para moverse por ellas. Esto último es a veces vital, ya que las cajas pueden contener grandes cantidades de texto. El límite normal de una caja de texto es aproximadamente 32.000 caracteres.

Labels (Etiquetas). Son para presentar información que los usuarios no pueden cambiar Identifican objetos y, ocasionalmente, se emplean para presentar la salida. Se puede tener un control casi completo sobre la forma en la que la etiqueta presenta la información: si el texto esta en negrita, el tamaño que tiene, etc. Si bien las etiquetas responden a 12 sucesos, se emplean normalmente de una forma pasiva, solamente para presentar información.

The Line and Shape Controls (Los controles línea y figura). Estas son las formas más rápidas para dibujar líneas y ciertas figuras como círculos y rectángulos Son similares a las etiquetas en el sentido de que son controles pasivos que se emplean sólo para la presentación. No responden a ningún suceso y, por tanto, no pueden responder a ninguna acción del usuario.

The Pointer (El puntero). El primer elemento de la caja de herramientas no es un control, pero se usa para manejar controles después de crearlos. Hay que hacer clic sobre el puntero cuando se desee seleccionar, mover o cambiar de tamaño un control ya existente Se activa automáticamente después de colocar un control en un formulario.

NOTAS:

Crear controles

La mayoría de los métodos que se emplean en la caja de herramientas son similares a los empleados en un programa de dibujo como Microsoft Paint, que es el que viene con Windows. Para manejar la caja de herramientas se emplea una concitación de apuntar, hacer clic y arrastrar. Por ejemplo, para dibujar un elemento de la caja de herramientas en un formulario·

1. Se lleva el puntero del ratón a la herramienta que se desea usar y se hace clic La herramienta cambia de color (sombreado) para indicar que ha sido seleccionada

2. Se lleva el puntero del ratón al formulario. Piense en él como si fuera un área a pintar en la que se quiere dibujar el control. Observe que el puntero del raton ha cambiado de una flecha a una cruz filiar.

3. Se mantiene pulsado el botón del ratón y se arrastra éste para crear el objeto. Conforme se va arrastrando el puntero del ratón va apareciendo en el formulario el perfil del control.

Una de las esquinas del control viene determinada por el punto donde se pulsó el ratón. en el paso 3, y la otra por el punto en el que se soltó el botón

Es de señalar que cuando se libera el botón del ratón. el control tiene sobresaliendo ocho cajas pequeñas llamadas *manejadores*. (Los controles de línea son unidimensionales y sólo tienen dos manejadores). Se emplean para mover o modificar el tamaño de un control después de creado. El control puntero también queda automáticamente resaltado cuando se libera el botón izquierdo del ratón.

Cuando se manipula un control, se notará que parece como si se moviera o cambiara de tamaño a saltos, no de forma suave. Por omisión se supone que la posición de los controles en un formulario de Visual Basic se sitúan solamente en puntos de la malla. Si no importa tener un control fuera de la malla, se puede suavizar el movimiento del mismo aunque si se hace será más **difícil** el alineado del control en el formulario.

> ***Nota***. La malla de puntos ayuda a situar el control con precisión

En teoría se pueden tener 255 objetos en un único formulario. pero raramente se llegará a esta cantidad. Una de las razones para limitar el número de los controles de un formulario es que Microsoft Windows tiene un límite en el número de controles y ventanas que puede manejar a la vez. Si se empieza a añadir demasiados controles o menús a un formulario. el programa se alentará drásticamente. Si Windows se va acercando a sus límites, es posible que deje colgado a Visual Basic, o bien que cierre la aplicación

NOTAS:

Trabajar con un control ya existente

Cuando se crea un botón de órdenes, éste aparece con una leyenda centrada: Command1, Command2, etc., como pronto se verá, es muy fácil cambiar estas leyendas por medio de la ventana Properties.

En Visual Basic no se ve uno nunca forzado a guardar el control con su tamaño original o en su posición original. Las técnicas necesarias para mover o cambiar el tamaño de los controles son las mismas para todos los controles de Visual Basic. Se puede también cortar y pegar los controles cortándolo y empleando el elemento Copy del menú Edit.

Para trabajar con un control existente es preciso, en primer lugar, seleccionarlo. Esto se hace llevando el puntero del ratón al interior del control y haciendo clic. (También se puede ir pulsando la tecla de tabulación hasta que el foco se posicione sobre el control, se puede ver donde está el foco viendo los manejadores.)

Cambiar el tamaño de un control existente

Se parte de la base de que se ha creado un botón de órdenes, pero no se está de acuerdo con su tamaño. Para cambiar el tamaño de un botón de órdenes ya existente, es necesario emplear los manejadores mencionados anteriormente.

Si no aparecen los manejadores se puede hacer que aparezcan llevando el puntero del ratón sobre el control y haciendo clic una vez. Cuando aparecen los manejadores se sabe que el control está seleccionado. Los manejadores de las cuatro esquinas permiten cambiar la anchura y la altura a la vez. El puntero del ratón, cuando se lleva sobre un manejador de las esquinas, cambia a una diagonal con dos flechas.

Los cuatro manejadores laterales permiten cambiar el **tamaño** sólo en una dirección. En estos manejadores, el puntero del ratón cambia a una delgada doble flecha.

Para cambiar realmente el tamaño de un control: Se lleva el puntero del ratón sobre un manejador y se pulsa y se mantiene pulsado el botón izquierdo del ratón Se arrastra el ratón hasta que el control tenga el tamaño deseado. Por ejemplo, si se quiere estrechar un botón de control por el lado izquierdo manteniendo el lado derecho **fijo,** se lleva el puntero del ratón sobre el manejador que está en el centro del lado izquierdo, se hace clic y se arrastra el ratón hacia la derecha.

Mover un control existente

Para poder mover un control existente, el foco ha de estar sobre el mismo. Obsérvese que cuando se mueve el ratón para que su puntero esté dentro del formulario, se convierte en una flecha. La forma de cruz filiar mencionada anteriormente sólo aparece cuando se crea un control nuevo. Para mover un control existente·

NOTAS:

1. Se lleva el puntero sobre cualquier punto del interior del control. se pulsa y se mantiene pulsado el botón izquierdo del ratón.

2. Se arrastra el ratón hasta que el control esté en la posición que se desee y entonces se libera el botón izquierdo.

Un atajo para crear controles

Ahora que se conoce cómo mover los controles. puede ser preferible emplear un atajo para crearlos. Si se hace un doble clic sobre cualquiera de los iconos de la caja de herramientas. ese control aparecerá en el centro de la pantalla. Si se hacen sucesivos dobles clics, los controles se irán apilando unos encima de los otros. A continuación se pueden emplear las técnicas de la sección anterior para ir llevándolos de la pila al sitio deseado y para cambiar su tamaño al que se desee.

Se desea crear, por ejemplo. una aplicación con cinco botones de órdenes colocados simétricamente. La forma más fácil de hacerlo es haciendo cinco veces doble clic sobre el icono del botón de órdenes. Esta operación apila cinco botones de órdenes en el centro del formulario.

Trabajar con controles múltiples

Ocasionalmente se deseará mover un grupo de controles en la misma dirección. Se puede tener, por ejemplo, tres botones de órdenes alineados y se desea mantenerlos alineados, pero moverlos hacia una pareja de puntos de la malla. Para trabajar con múltiples controles como si fueran una única unidad es preciso en primer lugar indicar a Visual Basic que se desea que temporalmente sean tratados como una unidad. Hay dos formas de hacerlo.

Supóngase un rectángulo imaginario que abarque exclusivamente a los controles que quiera seleccionar. Llévese el puntero a una de las esquinas de este rectángulo imaginario y púlsese el botón izquierdo del ratón.

Manteniendo pulsado el botón izquierdo del ratón se arrastra el rectángulo punteado hasta que comprenda a todos (y sólo a ellos) los controles que se quiera seleccionar. Seguidamente se libera el botón del ratón.

Los controles agrupados mostrarán todos manejadores *grises*. Cuando se seleccione uno cualquiera de los controles y se mueva. Visual Basic moverá los demás de una forma similar

El método anterior sólo sirve si los controles a mover se encuentran en un «lazo» rectangular que excluye a los demás controles. Si los controles se encuentran irregularmente dispuestos en el formulario. es preciso otro método.

NOTAS:

Se selecciona cada uno de los controles llevando el puntero sobre él y pulsando el botón izquierdo del ratón mientras se mantiene pulsada la tecla CTRL (A partir del segundo control, los manejadores se vuelven grises.)

Independientemente del método elegido, si se lleva el puntero del ratón sobre cualquiera de los controles que se han seleccionado y se le arrastra hacia la nueva posición, los demas controles se moverán con él.

Para anular el proceso de selección de múltiples controles se lleva el puntero del ratón fuera de los controles seleccionados y se hace clic.

Borrar controles

Se puede terminar con demasiados controles en el formulario, en especial si se emplea mucho el método del doble clic. Para borrar un control:

1. Se lleva el puntero del ratón sobre el interior del control y se hace clic con el botón izquierdo del ratón para seleccionarlo

2. Se pulsa DEL o se abre el menú Edit y se elige la opción Delete.

Los métodos vistos en la sección anterior para seleccionar múltiples controles y poder moverlos como si fueran uno se pueden emplear si se desea borrar varios controles a la vez. Una vez dicho a Visual Basic los controles que se desean tratar como si fueran uno, la tecla DEL o la opción Delete del menú son aplicables sobre todos los controles del grupo.

Propiedades de los botones de órdenes

Así como se emplea la ventana Properties para personalizar el tamaño y la forma de un formulario en blanco también se puede emplear para personalizar los controles. Es decir, que si no se quieren los valores por omisión de las propiedades de un control, se puede abrir la ventana Properties y cambiarlos.

Las siguientes secciones muestran las propiedades más útiles de los botones de órdenes. También se puede uno desplazar a través de la lista de propiedades y usar la ayuda en línea de la propiedad que tenga el nombre que llame la atención.

La propiedad Caption (Leyenda)

La propiedad Caption de un formulario determina el nombre que se muestra en la barra de título. De forma similar, la propiedad Caption de un botón de órdenes determina el mensaje o leyenda que el usuario verá. Cualquier texto que se empiee como leyenda de un botón de

NOTAS:

órdenes queda centrado automáticamente dentro del botón. Sin embargo, los botones de órdenes no cambian de tamaño para ajustarse al del mensaje, hay que hacerlo manualmente.

Los botones de órdenes siempre empiezan con leyendas como Command1, Command2, etc. El número indica el orden en el que los botones van siendo creados. Se puede crear primero el mensaje empleando la ventana Properties y luego cambiar el tamaño del botón Conforme se va agrandando el control se va mostrando más parte del mensaje, por lo que es fácil juzgar cuándo hay que parar.

La propiedad Name-Control Name (Nombre -Nombre del control)

La propiedad Name es muy importante cuando se empieza a escribir procedimientos de sucesos para los controles. Esta propiedad determina el nombre que Visual Basic usa para los procedimientos de sucesos que se escriban para hacer que el control responda al usuario. La elección de nombres significativos para los controles es un paso importante para que el proceso inevitable de la depuración de una aplicación sea más fácil

Los nombres de los controles *agrandan* el tamaño del archivo creado por Visual Basic al terminar de programar el proyecto. Los límites para los nombres de los controles en Visual Basic son los siguientes:

- El nombre de un control tiene que empezar con una letra. (Después de ella, se puede emplear cualquier combinación de letras, números y caracteres de subrayado.)
- El nombre no puede ser más largo de 40 caracteres.
- El convenio que usa Microsoft es emplear una abreviatura para el tipo de control seguida de una parte significativa. Un ejemplo sería emplear BtnAyuda en lugar de AyudaBoton.

Otras propiedades útiles para los botones de órdenes

Se pueden controlar 28 propiedades de un botón de órdenes. Por ejemplo, los botones de órdenes tienen propiedades BackColor, Height y Width. Lo que sigue es un breve estudio de los más básicos.

BackColor (Color de fondo). Como en los formularios, la propiedad BackColor establece el color de fondo del botón, pero en realidad sólo cambia el color de las cuatro esquinas. Debido a ello, la propiedad BackColor se cambia raramente. Los botones de órdenes carecen de la propiedad ForeColor, por lo que hay que aceptar el color por omisión Esta propiedad rara vez se cambia.

Height, Width (Alto, Ancho). Las propiedades Height y Width expresan la altura y la anchura del botón de órdenes. Observe que las unidades empleadas son las establecidas en las propiedades de *scale* (escala) del *contenedor* circundante. Normalmente el contenedor es el

NOTAS:

formulario. Esto significa que por omisión las medidas de la altura y la anchura de los botones de órdenes se expresan en *twips* (1120 de un punto de impresor o 111.440 de pulgada). Por otro lado, si se establece en 4 la propiedad ScaleMode del formulario circundante, las propiedades Height y Width de los botones de órdenes se expresarán en pulgadas. Se pueden cambiar directamente los valores de estas propiedades desde la ventana Properties o empleando los manipuladores. Como en los formularios, los valores actuales del tamaño del botón de órdenes se presentan en el extremo derecho de la ventana Properties del botón de órdenes.

Left, Top (Izquierda, Arriba). Estas propiedades determinan la distancia entre el botón de órdenes y el extremo izquierdo superior del contenedor (de nuevo, normalmente el formulario), respectivamente. Como con las propiedades Height y Width, estas propiedades usan la escala determinada por el formulario circundante. También se pueden cambiar arrastrándole y, como en los formularios, los valores se presentan en el extremo derecho de la caja de valores de la ventana Properties

Visible. La propiedad Visible determina si el botón estará visible o no. Es bastante corriente hacer que el código haga alternativamente que un botón de órdenes sea visible o invisible dependiendo de la situación. Al igual que la propiedad Visible de los formularios, esta propiedad sólo puede tomar los valores de True (Verdadero) o False (Falso).

Enabled (Habilitación): La propiedad Enabled determina si el botón puede responder a cualquier suceso que acontezca. Si se coloca esta opción en False, Visual Basic no responderá a ningún suceso que tenga que ver con este botón. A diferencia de la propiedad Visible, el botón permanece en el formulario, pero está inerte. La propiedad Enabled se emplea a menudo para mantener la flexibilidad del programa activándola o desactivándola temporalmente por medio del código. Al cambiar esta propiedad se cambia también la apariencia del objeto, normalmente poniendo el texto en gris.

MousePointer(Puntero del ratón). Para proporcionar al usuario una retroalimentación sobre el hecho de que ha llevado el foco al botón de órdenes, es una buena idea el poner un puntero del ratón de forma diferente a la flecha habitual. (Recuérdese que «tener el foco» es la frase normalizada de Microsoft Windows para describir que un control está presto para recibir entradas). En un botón de órdenes están disponibles los mismos 13 valores que para la propiedad MousePointer de un formulario

Propiedades de las fuentes.

Todas las propiedades de las fuentes -FontBold, FontItalic, FontName, FontSize, FontStrikeThrough y FontUnderline- pueden tratarse independientemente para cada botón de órdenes. Estas propiedades permiten controlar la apariencia del texto en el interior de un botón.

NOTAS:

Atajos para establecer propiedades

Supóngase que se desea establecer la propiedad Caption de todos los botones de órdenes del formulario. Si se hace una vez e inmediatamente se selecciona otro botón de órdenes, la propiedad Caption del control nuevo es la que aparece en la ventana Properties. Por lo general, Visual Basic recuerda la propiedad que se acaba de establecer en un control y, si es posible, saca a colación la misma propiedad en el siguiente control que se seleccione en la ventana Properties. (Pero se puede cambiar perfectamente la propiedad.)

De forma similar, si se selecciona un grupo de controles, la ventana Properties mostrará sólo las propiedades comunes del grupo elegido. Si se cambia una de ellas, se cambia para todos los controles.

Sugerencia. La forma más fácil de trabajar con los diferentes controles de un formulario es haciendo clic en la flecha abajo que hay en el extremo derecho de la primera línea de la ventana Properties. Esto hace que se muestre una lista de todos los objetos del formulario.

Procedimientos de suceso simple para botones de órdenes

Escribir un procedimiento de suceso para un botón de órdenes es similar a escribir uno para un formulario. Siempre que se haga un doble clic sobre un control o se use el atajo F7 Visual Basic abre la ventana Code. Visual Basic presenta una plantilla para el procedimiento más común del suceso (normalmente el suceso clic) para ese objeto.

Supóngase que se ha creado un control que contiene el texto «¡Haga clic aquí para AYUDA!» Además, imagine que se ha cambiado el nombre del control (la propiedad Name) por el de BotónAyuda que es más significativo La única diferencia está en el nombre del control que se usa seguido por un carácter de subrayado y seguido por el nombre del suceso. Este es el formato general de la plantilla de un procedimiento de suceso para los controles.

Sub NombreControl-NombreSuceso ()

End Sub

Visual Basic siempre sigue la pista de todos los objetos del proyecto, y se pueden escribir procedimientos de suceso para cualquiera de ellos abriendo la ventana Code, moviéndose a través de la caja con la lista Objetos y seleccionando el objeto que interese.

NOTAS:

Otros sucesos para botones de órdenes

Los botones de órdenes pueden responder a ocho sucesos, pero el clic es con mucho el más común. Otros dos que pueden encontrarse muy útiles son GotFocus (Obtención del foco) y LostFocus (Pérdida del foco).

Los botones de órdenes pueden responder también a la pulsación de determinadas teclas y a los sucesos del ratón.

Algunos puntos finales sobre botones de órdenes

Normalmente, el usuario de la aplicación que se está desarrollando elige un botón de órdenes llevando el puntero del ratón sobre el botón y haciendo clic. Sin embargo, hay veces que se desearía más flexibilidad. Otro de los métodos para activar un botón de órdenes es común a todas las aplicaciones de Microsoft Windows: mover el foco mediante la tecla TAB y luego pulsar la BARRA ESPACIADORA cuando el foco se encuentra donde se quiere. El usuario sabe que el botón ha recibido el foco, ya que pasa a tener un aspecto tridimensional. Lo que realmente pasa es que Visual Basic dibuja una fina caja punteada alrededor del texto del botón y un fino rectángulo alrededor del mismo botón. Ambos métodos generan el procedimiento del suceso clic para dicho botón. En otras palabras, Visual Basic activa el procedimiento del suceso clic en cualquier caso siempre que haya alguno disponible para ese botón.

Además, a veces se desea proporcionar a los usuarios un botón de escape para una acción. Este botón cancela una acción o rescata al usuario de algún tipo de situación en el que no quiere entrar. Se puede activar este botón de órdenes (uno por formulario) pulsando Esc en el teclado. El botón de órdenes que hace esto recibe el nombre de *cancel button* (botón de cancelación) en los manuales de Visual Basic.

Normalmente se emplea la ventana Properties para hacer que un botón de órdenes sea un botón de cancelación, aunque también se puede hacer mediante código. Al desplazarse a través de la lista de propiedades disponibles para el botón de órdenes se podrá ver la propiedad Cancel. Si se pone esta propiedad en True se establece que al pulsar ESC se genera el suceso clic de este botón, sin tener en cuenta el lugar donde estuviera el foco. Al poner en True la propiedad Cancel de un botón se ponen automáticamente en False las de los demás botones del formulario.

Otra posibilidad es establecer un botón de órdenes por omisión para el fomulario. Cuando se pulse INTRO se generará el procedimiento clic del botón elegido (botón por omisión). Si se quiere usar esta opción, hay que poner en True la propiedad Default (Omisión) del botón. Sólo se puede tener un botón de órdenes por omisión por formulario.

NOTAS:

Sugerencia. Se pueden combinar los botones por omisión y de cancelación en un botón de cancelación por omisión. Esta posibilidad es especialmente útil si se está a punto de acometer una acción irreversible.

Teclas de acceso

Muchas de las aplicaciones Windows permiten que al pulsar ALT y cualquier otra tecla. en la *tecla de acceso* se active rápidamente un control o un elemento del menú, el mismo Visual Basic, por ejemplo Estas letras se encuentran subrayadas en la leyenda del control o en el nombre del elemento del menú.

Visual Basic hace que sea muy fácil el colocar una tecla de acceso a cualquier objeto que tenga la propiedad Caption. Al establecer esta propiedad. lo único que hay que hacer es colocar un ampersand (&) delante de la letra que se quiere que sea la letra clave. Cuando se ejecute la aplicación. se puede activar este botón bien pulsando la combinación de teclas ALT+C o haciendo un clic en el boton

Controles de líneas y formas

El control de líneas puede usarse para presentar, en un formulario, líneas de diferentes grosores. El control de formas se puede emplear para presentar rectángulos, cuadrados, óvalos o círculos. También se puede emplear para presentar rectángulos o cuadrados de esquinas redondeadas.

El control de formas

El control de formas tiene 18 propiedades. Normalmente se cambian de forma dinámica mediante código al ejecutarse la aplicación Las propiedades más importantes del control de formas en el momento del diseño son las que se describen en las secciones siguientes.

Shape (Forma). Determina el tipo de forma que se obtendrá. Se puede elegir entre seis valores diferentes

BackStyle (Estilo de fondo). Esta propiedad determina si el fondo de la forma es transparente o no El valor por omisión es 1, lo que proporciona un borde opaco; BackColor (Color de fondo) rellena la forma y tapa lo que esté detrás de ella. Si se pone el valor en 0 (Transparente) se podrá ver los que este detrás de la forma.

BorderWidth (Ancho del borde). Determina el grosor de la línea. Se mide en pixels y puede variar de 0 a 8 192 (demasiado grande para ser presentado en la mayoría de los formularios).

NOTAS:

BorderStyle (Estilo del borde). A diferencia de los controles de imágenes. BorderStyle tiene seis valores posibles.

> **Nota**. Si se da un valor mayor que 1 a la propiedad BorderWidth. no tendrá efecto lo que se coloque en la propiedad BorderStyle.

FillColor, FillStyle (Color del relleno, Estilo del relleno). La propiedad FiliColor determina el color a emplear para rellenar la forma en la manera establecida por la propiedad FillStyle. El valor de la propiedad FillColor puede establecerse de la misma forma que cualquier propiedad de color. bien directamente por medio del código hexadecimal o empleando la paleta de colores La propiedad FillStyle tiene ocho valores posibles:

El control de líneas

El control de líneas tiene diez propiedades. Normalmente se cambian de forma dinamica mediante código al ejecutarse la aplicación. Las propiedades más importantes del control de formas en el momento del diseño son las propiedades BorderWidth (Ancho del borde) y BorderStyle (Estilo del borde) BorderWidth determina el grosor de la línea. Se mide en pixels y puede variar de 0 a 8 192 (demasiado grande para ser presentado en la mayoria de los formularios.

Cajas de texto y etiquetas

Las cajas de texto son el método principal para aceptar entradas y para presentar salidas en Visual Basic.

Las etiquetas se emplean para presentar información que no se quiere que el usuario pueda cambiarla Probablemente el uso más común de las etiquetas es para identificar, describiendo su contenido. una caja de texto o cualquier otro control. Otro empleo corriente de las etiquetas es para presentar información de ayuda. El icono para las etiquetas es una letra A mayúscula y en negrita

Propiedades estándar de las cajas de texto

Las cajas de texto tienen 40 propiedades. Muchas de ellas ya son familiares al lector Como antes. la propiedad Name (Nombre) se emplea sólo para el código que se escriba. el usuario nunca la verá Las seis propiedades de las fuentes (FontBold, FontItalic. etc.) funcionan de la misma forma que antes Controlan la forma en la que aparece el texto en la caja. La única diferencia estriba en que. puesto que los usuarios pueden llevar el foco a una caja de texto y escribir allí información las propiedades de las fuentes también afectan a lo que el usuario

NOTAS:

coloque dentro de la caja de texto. Como en los botones de órdenes, las propiedades Height. Width. Left y Top emplean la escala determinada por el contenedor circundante.

A diferencia de los botones de órdenes (pero como los formularios) se pueden establecer en una caja de texto tanto las propiedades ForeColor como BackColor. La primera afecta al color del texto a visualizar. BackColor afecta al resto de la caja de texto. Ambas son independientes del color del contenedor circundante.

Al igual que con los botones de órdenes. la propiedad Enabled determina si la caja de texto responderá o no a los sucesos. En particular. si se inhabilita una caja de texto, el usuario no podrá introducir texto en ella. Cuando una caja de texto está inhabilitado, se pone de color gris.

También como antes. es bastante corriente el cambiar la propiedad Visible de True a False, y viceversa. mediante código para hacer que la caja de texto aparezca o desaparezca.

La propiedad MousePointer tiene los mismos 13 valores posibles que con los formularios. A menudo se cambia esta propiedad para resaltar que el foco se encuentra ahora en el control.

Algunas propiedades especiales de las cajas de texto

Hay tres propiedades que no se han visto antes y una que funciona de diferente manera en las cajas de texto que en los formularios. Las tres propiedades nuevas son Text, MultiLine y ScrollBars, y son muy importantes para manejar las cajas de texto. La propiedad BorderStyle trabaja aquí de forma diferente a la de los formularios.

Text (Texto). La propiedad Text es análoga la propiedad Caption de los botones de órdenes o de los formularios. La propiedad Text controla el texto que el usuario ve. Al crear una caja de texto. el valor por omisión de esta propiedad es el del valor por omisión de la propiedad Name del control (Text1. Text2. etc.). Si se necesita una caja de texto que esté vacía al arrancar la aplicación. basta con seleccionar la propiedad texto y borrar el valor original

ScrollBars (Barras de desplazamiento) Esta propiedad determina si una caja de texto tiene barras de desplazamiento horizontal o vertical. Son muy útiles ya que Visual Basic permite aceptar líneas largas o múltiples de datos en una simple caja de texto. El límite normal está en 32.000 caracteres aproximadamente Sin barras de desplazamiento es mucho más difícil para el usuario moverse a través de los datos contenidos en la caja de texto, haciendo que la edición de la información sea mucho más difícil.

MultiLine (Línea múltiple). La propiedad MultiLine determina si la caja de texto puede aceptar más de una línea de texto cuando el usuario ejecute la aplicación y normalmente se combina con el valor de la propiedad ScrollBars. De todos modos. si se pone esta propiedad en True, el usuario puede usar los métodos estándar de Microsoft Windows para moverse a través de la caja de texto. las flechas.

NOTAS:

INICIO. CTRL+INICIO, FIN Y CTRL+END.

Visual Basic formatea automáticamente el texto cuando un usuario teclea más de una línea de información en la caja de texto. a menos que se haya añadido a la caja de texto una barra de desplazamiento horizontal. Los usuarios pueden emplear también la tecla INTRO para separar las líneas. a menos que se haya añadido al formulario un botón de órdenes por omisión (una razón más para tener cuidado con esto). Si se tiene un botón de órdenes por omisión, el usuario deberá pulsar CTRL+INTRO para partir las líneas.

Puesto que los formularios tienen limitada la cantidad de texto que pueden presentar sin desplazarlo, las cajas de texto MultiLine son el método usual de presentar en Visual Basic grandes cantidades de texto. El límite para una caja de texto MultiLine es aproximadamente 32.000 caracteres.

BorderStyle (Estilo de borde). Al igual que en los controles de imágenes. la propiedad BorderStyle de una caja de texto sólo tiene dos valores posibles El valor por omisión es 1. lo que proporciona un borde de ancho sencillo. denominado *fijo sencillo.* Si el valor de esta propiedad se hace 0, el borde desaparece.

MaxLength (Longitud máxima). Esta propiedad determina el número máximo de caracteres que puede aceptar la caja de texto. El valor por omisión es 0. lo que de una forma muy aproximada limita el número máximo de caracteres de una caja de texto multilínea a 32.000 Cualquier otro valor distinto de cero limita a ese valor el número de caracteres que puede un usuario introducir en dicha caja de texto.

PasswordChar (Caracteres de clave). Como puede imaginarse por su nombre, esta propiedad permite indicar lo que la caja de texto presentará. Por convención se emplea como carácter para las claves el asterisco (*). Una vez activada esta propiedad. todo lo que verá el usuario será una fila de asteriscos. Frecuentemente se combina esta propiedad con MaxLength para añadir a los programas una contraseña.

Procedimientos de suceso para las cajas de texto

Las cajas de texto pueden reconocer 12 sucesos Los sucesos como GotFocus y LostFocus funcionan exactamente como antes. Otros tres -KeyDown. KeyUp y KeyPress (Tecla arriba. Tecla abajo y Tecla pulsada)- son para monitorizar exactamente lo que el usuario teclee.

Visual Basic monitorea la caja de texto y llama al procedimiento de este suceso siempre que el usuario haga cualquier cambio en ella. No importa lo que el usuario teclee. Visual Basic lo detectará Uno de los usos más normales de este suceso es alertar a la gente de que no deben introducir texto en una caja de texto especifica. borrando lo que hayan tecleado

NOTAS:

Etiquetas

Existen 31 propiedades para las etiquetas. La mayoría de ellas coinciden con las propiedades de las cajas de texto y de los formularios y muchas de ellas serán ya familiares. Al igual que los formularios (pero a diferencia de las cajas de texto), las etiquetas tienen la propiedad Caption que determina lo que presentará. El valor de esta propiedad es originalmente Label1 para la primera etiqueta del formulario, Label2 para la segunda y así sucesivamente. Cuando se diseña una etiqueta se puede poner más de una línea de texto como leyenda. Se pueden añadir líneas de texto en blanco en la leyenda mediante código.

Como antes, la propiedad Name se emplea sólo en el código que se escriba; el usuario nunca la ve. Las seis propiedades de las fuentes (FontBold, FontItalic, etc.) funcionan de la misma forma que antes, controlando la forma en la que el texto aparece en la etiqueta. Al igual que en los botones de órdenes, las propiedades Height, Width, Left y Top emplean la escala determinada por el contenedor circundante.

Se pueden establecer las propiedades BackColor y ForeColor en una etiqueta igual que en una caja de texto. La propiedad ForeColor afecta al color del texto que se visualiza, BackColor afecta al resto de la etiqueta. Ambas son independientes del contenedor circundante.

La propiedad Enabled no se emplea mucho con las etiquetas. Su papel principal es determinar si el usuario puede mover el foco al control que sigue a la etiqueta en el orden del tabulador. Como antes, es muy corriente el cambiar alternativamente la propiedad Visible de True a False para hacer que la etiqueta aparezca y desaparezca.

La propiedad MousePointer emplea los mismos 13 valores posibles. Raramente se cambia en las etiquetas, pero una posibilidad es cambiar el icono cuando el usuario se mueve desde la etiqueta al control que está siendo etiquetado.

Propiedades útiles de las etiquetas

Hay cinco propiedades de las etiquetas que son especialmente útiles, dos de las cuales no se han visto todavía. Alignment y AutoSize. La tercera, WordWrap trabaja de una forma ligeramente diferente de las cajas de texto. Por ejemplo, WordWrap puede usarse sólo si AutoSize está en True. Por otro lado, la propiedad BorderStyle tiene un uso más atractivo que hace que las aplicaciones sean más cuidadas

Alignment (Alineación). La propiedad Alignment tiene tres valores posibles. El valor normal (por omisión) es 0, lo que significa que el texto de la etiqueta estará justificado por la izquierda (alineado por la izquierda) Si se hace que el valor de esta propiedad sea 1, el texto quedará justificado por la derecha: si fuera 2, el texto quedaría centrado.

NOTAS:

AutoSize, WordWrap (Tamaño automático, Enlace de textos). A diferencia de los botones de órdenes, las etiquetas pueden crecer automáticamente horizontalmente para acompañar al texto que se coloque en ellas. Esta es la función de la propiedad AutoSize. El valor por omisión de esta propiedad es, sin embargo, False, por lo que es preciso ponerlo en True para aprovechar sus ventajas. Si además se pone la propiedad WordWrap en True, la etiqueta crecerá en el sentido vertical acompañando al texto, si bien el tamaño horizontal permanecerá igual. Además, el texto quedará enlazado de forma que las palabras no puedan quedar partidas como se ve aquí.

Procedimientos de suceso para etiquetas

Las etiquetas responden a 12 sucesos. Responden por ejemplo a los sucesos clic, doble clic o Change (Cambio). Los procedimientos de suceso más comunes en las etiquetas son los sucesos del ratón. Se puede emplear el clic del botón derecho del ratón para proporcionar una ayuda sensible al contexto o, por ejemplo, para hacer aparecer un **menú.** Un problema es que las etiquetas no responden a los sucesos de teclas y no detecta si el usuario ha desplazado el foco. Esto restringe dramáticamente el empleo de procedimientos de suceso en las etiquetas. Las etiquetas son en Visual Basic principalmente descriptivas y no interactivas.

Navegar entre controles

La forma mas normal de mover en una aplicación Windows de control en control es empleando el ratón si bien las aplicaciones deben permitir también el uso de la tecla TAB. *Tab Order* (Orden de tabulación) es el término que se emplea en una aplicación Windows para describir una secuencia de controles en los que se puede mover de uno a otro pulsando la tecla TAB. En Visual Basic el orden en el que se van creando los controles es el mismo orden de tabulación. Al diseñar una aplicación, el primer control que se crea es el que recibe el foco cuando se arranca la aplicación. Si se pulsa una vez TAB cuando la aplicación se está ejecutando, el foco se desplaza al segundo control que se diseñó, y así seguidamente. Si se pulsa la tecla TAB cuando el foco está en el último control, el foco se desplazará al primero. (Los controles inhabilitados se saltan)

El orden de tabulación se puede cambiar por medio de la ventana Properties o por medio de código. La propiedad que se necesita es la propiedad Tabindex (Indice de tabulación). Si en un control se pone a 0 esta propiedad, este control pasa a ser automáticamente el primero en el orden de tabulación mientras que los demás se correrán un puesto cada uno, es decir, que el que era el primero es ahora el segundo, el que era el segundo el tercero, y así sucesivamente. Si se cambia un control con un número de orden mayor, sólo quedarán afectados aquellos que tengan un número de orden mayor. Al crear un control y establecer la propiedad Tabindex, los valores de la propiedad Tabindex se mueven para dejar sitio al control nuevo También se puede cambiar Tabindex por medio de código.

NOTAS:

Teclas de acceso para cajas de texto

Las cajas de texto carecen de la propiedad Caption, por lo que es necesario un truco que permita a los usuarios llevar rápidamente el foco a ellas por medio de una tecla de acceso. El truco se basa en lo siguiente: las etiquetas tienen leyenda, por lo que se puede establecer una tecla de acceso a ellas mediante el empleo de un ampersand (&) delante de la letra que se quiera que sea la tecla de acceso. Sin embargo, las etiquetas no responden a los sucesos GotFocus o LostFocus. ¿Qué sucede entonces si se pulsa la tecla de acceso? Si se pulsa la tecla de acceso de un control que no responde a los sucesos del foco, éste se mueve al siguiente control del orden de tabulación que lo acepte.

Esto hace que proporcionar una tecla de acceso a una caja de texto sea fácil. Se crea una etiqueta para la caja de texto, se establece la tecla de acceso para la etiqueta y se crea la caja de texto. (Al hacerlo hay que asegurarse que la caja de texto siga a la etiqueta en el orden de tabulación.)

Cajas de Mensajes

Las cajas de mensajes presentan información en un cuadro de diálogo superpuesto al formulario. Esperan a que el usuario elija un botón antes de volver a la aplicación. Los usuarios no pueden cambiarse a otro formulario de la aplicación mientras que Visual Basic esté presentando una caja de mensajes. Estas cajas deben emplearse para pequeños mensajes o para proporcionar una realimentación transitoria. No se debe emplear, por ejemplo, para una pantalla de ayuda. Un buen ejemplo de dónde una aplicación puede presentar una caja de mensajes es cuando el usuario mueve el foco fuera de una caja de texto antes de introducir información en ella. La forma más simple de orden de una caja de mensajes es la siguiente:

MsgBox(«El mensaje debe ir entre comillas»)

Las cajas de mensajes pueden contener un máximo de 1.024 caracteres y Visual Basic los partirá al llegar a la derecha del cuadro de diálogo. También se pueden partir las líneas manualmente. Se ha escrito, por ejemplo, una aplicación y se piensa que será necesario recordar al usuario que no sucederá nada hasta que haga clic en un botón de órdenes. Se puede añadir un procedimiento del suceso LostFocus como el siguiente:

Cuando se ejecute esta aplicación y se lleve el foco fuera del botón de órdenes se verá una pantalla como la mostrada en la Figura 4-12. Se puede observar en esta figura que la barra de título de la caja de mensaje no es particularmente informativa. Es factible añadir otro título más explicativo Para ello, es necesario emplear la forma completa de la sintaxis de la orden de la caja de mensajes añadiendo dos opciones Esta es la sintaxis completa de la orden MsgBox:

MsgBox Mensaje En La Caja, Tipo De Caja, Título De La Caja

NOTAS:

El elemento *Mensaje En La Caja* ya se ha visto, es el que proporciona el texto de la caja. Para especificar el tipo de caja de mensaje se combinan tres grupos de números diferentes que controlan el tipo de botón que aparecerá, los iconos y, por último, el botón por omisión de la caja. Las siguientes tablas sintetizan esta información.

Los grupos de números de los iconos que aparecen en la caja son los siguientes:

La malla

La malla es muy importante para colocar los controles con precisión por lo que si se domina, será fácil dar un acabado con aspecto profesional a las aplicaciones. La malla se controla desde el cuadro de diálogo Environment Options (Opciones del entorno) mostrada en la Figura 4-13, propiedades que se pueden controlar.

Grid Width, Grid Hight *(Ancho de la malla, Alto de la malla).* Esta propiedad controla el ancho (en twips) entre las marcas de la malla. El valor por omisión es 120 *twips. Si* se cambia este valor a 60, como se muestra en la Figura 4-13, la malla será dos veces más **fina.**

Show Grid *(Mostrar la malla).* Cambiando esta propiedad se puede activar o desactivar la malla El valor por omisión es on (activada). Normalmente hay pocas razones para desactivarla.

Align to Grid (Alinear con la malla). La opción Align to Grid determina si los controles se moverán automáticamente a la marca de la malla más próxima o si se pueden colocar entre marcas de la malla

Normalmente es mejor cambiar la distancia entre las marcas de la malla para que se adapte al diseño pensado que desactivar esta opción

Es posible alinear un elemento con la malla incluso aunque se haya desactivado ésta. Para ello, se selecciona el control haciendo clic una vez sobre él (aparecerán sus manejadores) y se elige Align to Grid (Alinear con la malla) del menú Edit

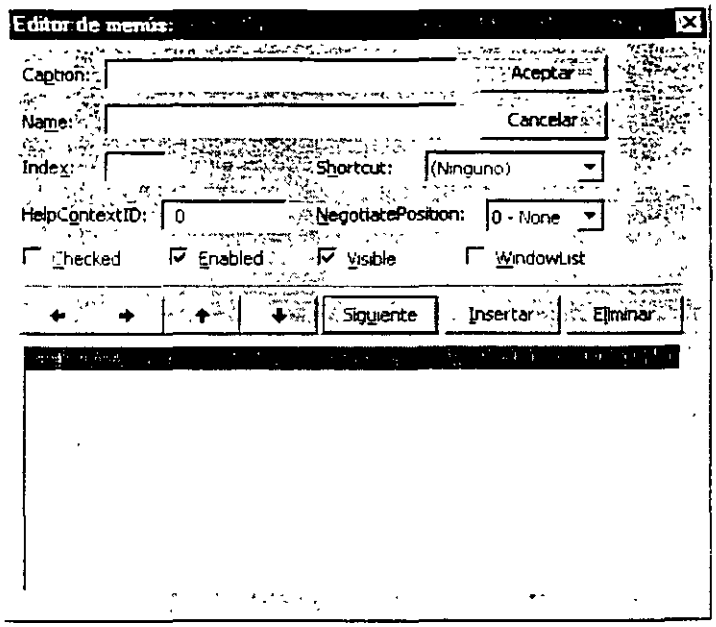
NOTAS:

Uso de MENUS

El manejo o implementación de menus, se realiza a través del asistente de menús, el cual se activa mediante las siguientes opciónes:

- Click Derecho sobre del formulario: Editor de Menus
- Menu Herrameintas: Editor de Menus

La estructura de menus, esta basada en el ALINEAMIENTO, en donde los menus principales, representan los textos con cero alineación. Las opciones de cada menu, son manejadas a través de la identación de un nivel de los textos



PROGRAMACIÓN POR EVENTOS

Un procedimiento de evento es un procedimiento ***Sub*** llamado automáticamente como respuesta a un evento. Para crear un procedimiento de evento, primero deberá determinar el evento (por ejemplo, hacer clic con el mouse o actualizar un registro) cuya propiedad de evento ha de activarlo.

NOTAS:

Crear un Procedimiento de Evento

Pasos a Seguir:

3. Abra un ***formulario***.

4. Haga doble clic en el objeto. Visual Basic mostrará la ventana de código.

5 Seleccione el evento que activará el procedimiento de evento.

6. Escriba el procedimiento de evento entre las líneas ***Sub*** y ***End Sub***.

7 Guarde los cambios en el módulo.

Cuando el formulario esté abierto, cada vez que ocurra el evento especificado se producirá una llamada al procedimiento de evento.

Ejecutar Código de Visual Basic

El código de Visual Basic se puede ejecutar invocando el procedimiento ***Function*** o ***Sub*** que contiene el código. Normalmente, el código se ejecuta como respuesta a un evento. Visual Basic está accionado por eventos: reconoce y responde a eventos, como el hacer clic con botón del mouse, abrir un formulario o modificar datos de un registro.

Todos los eventos tienen una propiedad de evento asociada, que puede llamar a un procedimiento de evento, a una macro o a una función. Los procedimientos de evento sólo pueden ser de tipo ***Sub***.

FUNCIONES Y PROCEDIMIENTOS

Procedimiento Function

Un procedimiento que devuelve un valor y que puede ser utilizado en una expresión. Una función se declara usando la instrucción ***Function*** y terminándola con la instrucción ***End Function***.

NOTAS:

Instrucción Function

Declara el nombre, argumentos y código que forman el cuerpo de un procedimiento ***Function***.

Sintaxis

*[**Static**] [Private] Function nombreDeFunción [(listaArgumentos)] [As tipo]*

[bloqueDeInstrucciones]

[nombreDeFunción = expresión]

*[**Exit** Function]*

[bloqueDeInstrucciones]

[nombreDeFunción = expresión]

***End** Function*

Observaciones

Todos los códigos ejecutables deben encontrarse en procedimientos ***Function*** o procedimientos ***Sub***. No puede definir un procedimiento ***Function*** dentro de otro procedimiento ***Function*** o ***Sub***.

Argumentos

La instrucción ***Function*** usa estos argumentos.

Argumento	*Descripción*
Static	Indica que las variables locales del procedimiento ***Function*** se preservan entre las llamadas. El atributo ***Static*** no afecta las variables que se declaran fuera del procedimiento ***Function***, aunque se usen en el procedimiento.
Private	Indica que el procedimiento ***Function*** es accesible únicamente a otros procedimientos en el módulo en el que existe. Ningún otro procedimiento en ningún otro módulo tiene acceso.

NOTAS:

Argumento	*Descripción*
NombreDeFunción	Nombre de la función. Los nombres de procedimientos ***Function*** siguen las mismas reglas que re***String***en los nombres de otras variables y pueden incluir un carácter de declaración de tipo. Puesto que los nombres de ***Function*** son reconocidos por todos los procedimientos en todos los módulos, ***nombreDeFunción*** no puede ser igual a ningún otro nombre reconocido ***Global***mente en el programa. Observe que el tipo del nombre determina el tipo de datos devuelto por el procedimiento ***Function***. Para especificar el tipo de datos a devolver del procedimiento ***Function***, use una cláusula As después de ***listaArgumentos*** o incluya un carácter de declaración de tipo en el nombre de ***Function***. Por ejemplo, para crear un procedimiento ***Function*** que devuelve una cadena, puede incluir símbolo de moneda en el nombre, usar una cláusula ***As String*** o asignarle un nombre definido como un nombre de cadena con una instrucción ***DefStr***
listaArgumentos	Lista de variables, que representan argumentos, que se pasan al procedimiento ***Function*** al ser invocado. Cuando hay múltiples variables, éstas se separan con comas. A menos que se identifiquen con la palabra reservada ***ByVal***, los argumentos individuales se pasan por referencia, de manera que al cambiar el valor de un argumento dentro del procedimiento ***Function*** se cambia su valor en el procedimiento de llamada. Si un argumento pasado a una función es una expresión, se trata como si usara la palabra reservada **ByVal**: la función no altera ninguna parte de la expresión.
As tipo	Palabra reservada que se usa para declarar el tipo de datos del valor devuelto por el procedimiento ***Function***; tipo puede ser un tipo de datos ***Integer, Long, Single, Double, Currency, String o Variant.***
BloqueDeInstrucciones	Cualquier grupo de instrucciones que se ejecutará dentro del cuerpo del procedimiento ***Function***
Expresión	Valor devuelto de la función. Un procedimiento ***Function*** devuelve un valor asignando ese valor a la función ***nombreDeFunción***. No hay límite para el número de tales asignaciones que pueden aparecer en cualquier lugar del procedimiento. Si no se asigna ningún valor a ***nombreDeFunción***, el procedimiento devuelve un

NOTAS:

Argumento	*Descripción*
	valor predeterminado: una función numérica devuelve 0, una función de tipo de datos ***String*** devuelve una cadena de longitud cero (**) y una función de tipo de datos ***Variant*** devuelve un tipo de datos Vacio
Exit Function	Causa una salida inmediata de un procedimiento ***Function***. La ejecución del programa continúa con la instrucción que sigue después de la instrucción que llamó el procedimiento ***Function***. No hay límite para el número de instrucciones ***Exit Function*** que pueden aparecer en cualquier lugar de un procedimiento ***Function***.

Function y ***End Function*** marcan el comienzo y el final de un procedimiento ***Function***.

Al igual que un procedimiento ***Sub***, un procedimiento ***Function*** es un procedimiento separado que puede llevar argumentos, realizar una serie de instrucciones o cambiar los valores de sus argumentos. Sin embargo, a diferencia de un procedimiento ***Sub***, un procedimiento ***Function*** se puede usar en una expresión de la misma manera que cualquier función intrinseca, como por ejemplo Sqr, Cos o Chr.

Un procedimiento ***Function*** se puede invocar usando el nombre de la función, seguido por la lista de argumentos entre paréntesis, en una expresión. Aunque la función no tenga argumentos, deberá incluir los paréntesis.

Los procedimientos ***Function*** pueden ser recursivos; es decir, pueden invocarse ellos mismos para realizar una tarea determinada. Sin embargo, la recursión puede conducir a un desbordamiento en la pila. La palabra reservada ***Static*** generalmente no se usa con los procedimientos ***Function*** recursivos.

Las variables utilizadas en los procedimientos ***Function*** se agrupan en dos categorías: aquéllas que se declaran explícitamente dentro del procedimiento y las que no. Las variables que se declaran explícitamente en un procedimiento (usando ***Dim*** o el equivalente) siempre son locales al procedimiento. Otras variables utilizadas pero no declaradas explícitamente en un procedimiento también son locales a menos que se declaren explícitamente en algún nivel superior fuera del procedimiento.

Un procedimiento puede usar una variable que no está declarada explícitamente en el procedimiento, pero podría ocurrir un conflicto de nombres si algún elemento definido en la sección Declaraciones tiene el mismo nombre. Si su procedimiento hace referencia a una variable no declarada que tiene el mismo nombre que otro procedimiento, constante o variable ***Global*** o de nivel del módulo, o un objeto, Visual Basic supone que su procedimiento se está refiriendo a ese nombre a nivel del módulo. Declare explícitamente las variables para evitar esta

NOTAS:

clase de conflicto. Puede usar una instrucción ***Option Explicit*** para forzar la declaración explícita de variables.

Procedimiento Sub

Un procedimiento que ejecuta una operación. A diferencia de un procedimiento ***Function***, un procedimiento ***Sub*** no devuelve un valor. Un procedimiento ***Sub*** se declara con la palabra reservada ***Sub*** y se termina con una instrucción ***End Sub***.

Instrucción Sub

Sintaxis

*[**Static**] [Private] Sub nombreDeSub [(listaDeArgumentos)]*

[bloqueDeInstrucciones]

*[**Exit** Sub]*

[bloqueDeInstrucciones]

***End** Sub*

Todos los códigos ejecutables deben encontrarse en procedimientos ***Function*** o ***Sub***. No puede definir un procedimiento ***Sub*** dentro de otro procedimiento ***Sub*** o ***Function***

Argumentos

La instrucción ***Sub*** usa estos argumentos

Argumento	*Descripción*
Static	Indica que las variables locales del procedimiento ***Sub*** se preservan entre llamadas El atributo ***Static*** no afecta a las variables que se hayan declarado fuera del procedimiento ***Sub***, aunque se usen en el procedimiento
Private	Indica que el procedimiento ***Sub*** es accesible sólo a otros procedimientos del módulo en el que existe. Ningún otro procedimiento en ningun otro módulo tiene acceso a él.

NOTAS:

Argumento	*Descripción*
NombreDeSub	Nombre del procedimiento. Los nombres de procedimientos ***Sub*** siguen las mismas reglas que re***String***en los nombres de otras variables pero no pueden incluir un carácter de declaración de tipo. Puesto que los nombres ***Sub*** son reconocidos por todos los procedimientos en todos los módulos, ***nombreDeSub*** no puede ser igual a ningún otro nombre reconocido ***Global***mente en el programa.
ListaDeArgumentos	Lista de variables que representan argumentos, que se pasan al procedimiento ***Sub*** al ser llamado. Cuando se tienen múltiples variables, éstas se separan con comas. A menos que se identifiquen con la palabra reservada ***ByVal***, los argumentos se pasan por referencia, así, al cambiar el valor de un argumento dentro del procedimiento ***Sub*** se cambia su valor en el procedimiento de llamada.
BloqueDeInstrucciones	Cualquier grupo de instrucciones que se ejecutan dentro del cuerpo de un procedimiento ***Sub***.
Exit **Sub**	Causa una salida inmediata de un procedimiento ***Sub***. La ejecución del programa continúa con la instrucción que sigue después de la instrucción que invocó el procedimiento ***Sub***. No hay límite para el número de instrucciones ***Exit Sub*** que pueden aparecer en cualquier lugar de un procedimiento ***Sub***.

Sub y ***End Sub*** marcan el comienzo y el final de un procedimiento ***Sub***.

Al igual que un procedimiento ***Function*** un procedimiento ***Sub*** es un procedimiento separado que puede llevar argumentos, realizar una serie de instrucciones y cambiar el valor de sus argumentos. Sin embargo, a diferencia del procedimiento ***Function*** que devuelve un valor, un procedimiento ***Sub*** no se puede usar en una expresión

Un procedimiento ***Sub*** se puede llamar usando el nombre del procedimiento seguido por la lista de argumentos.

Advertencia Los procedimientos ***Sub*** pueden ser recursivos; es decir, pueden invocarse a sí mismos para realizar una tarea determinada. Sin embargo, la recursión puede conducir a un desbordamiento en la pila. La palabra reservada ***Static*** generalmente no se usa con los procedimientos ***Sub*** recursivos.

Las variables usadas en los procedimientos ***Sub*** se clasifican en dos grupos: aquéllas que se declaran explícitamente dentro del procedimiento y las que no. Las variables que se declaran

NOTAS:

explícitamente en un procedimiento (usando ***Dim*** o el equivalente) siempre son locales al procedimiento. Otras variables usadas pero no declaradas explícitamente en un procedimiento son también locales a menos que estén declaradas explícitamente en algún nivel superior fuera del procedimiento.

> ***Advertencia:*** Un procedimiento puede usar una variable que no se ha declarado explícitamente en el procedimiento, pero podría ocurrir un conflicto de nombres si algún elemento que se haya definido en la sección Declaraciones tiene el mismo nombre. Si su procedimiento se refiere a una variable no declarada que tiene el mismo nombre que otro procedimiento, constante o variable ***Global*** o a nivel de módulo, u objeto, Visual Basic supone que su procedimiento se está refiriendo a ese nombre de nivel del módulo. Declare explícitamente las variables para evitar esta clase de conflicto. Puede usar una declaración ***Option Explicit*** para forzar la declaración explícita de variables.

DECLARACIÓN DE VARIABLES

Instrucción Const

Declara constantes simbólicas para usar en lugar de valores

Sintaxis

*[**Public**] Const nombre*Constante = *expresión [, nombre*Constante = *expresión]*

Argumentos

La instruccion ***Const*** usa estos argumentos.

Argumento	*Descripción*
Global	Palabra reservada que puede preceder a ***Const*** para declarar constantes a las que todos los procedimientos pueden hacer referencia en todos los modulos.
NombreConstante	Nombre de la constante
Expresión	Expresión que se asigna a la constante. Puede estar compuesta por

NOTAS:

Argumento	Descripción
	literales (como por ejemplo 1,0), otras constantes o cualquiera de los operadores aritméticos o lógicos con la excepción de exponenciación (^). También puede usar una sola cadena literal, como por ejemplo Error al escribir. No puede usar concatenación de cadenas, variables, funciones definidas por el usuario ni funciones intrinsecas de Visual Basic (como por ejemplo Chr[$] en expresiones asignadas a constantes

Sugerencia: Las constantes pueden hacer que sus programas creen su propia ***Documentación*** y sean fáciles de modificar. A diferencia de las variables, las constantes no se pueden cambiar inadvertidamente.

Sugerencia: Use sólo letras mayúsculas para los nombres de constantes a fin de que sean fáciles de reconocer en sus listados de programas.

Advertencia: Antes de poder hacer referencia a una constante, ésta tiene que haber sido definida. Asegúrese de que las constantes ***Public*** no estén colocadas en múltiples módulos de tal manera que se causaría una dependencia entre módulos que no se puede resolver. Para evitar tal dependencia coloque sus definiciones de constantes tipo ***Public*** en un solo módulo.

Instrucción Dim

Se usa a nivel de módulo y a nivel de procedimiento para declarar variables y asignar el espacio de almacenamiento.

Sintaxis

Dim *nombreDeVariable[([indices])][As tipo][, nombreDeVariable[([indices])][As tipo]] . . .*

Argumentos

La instrucción ***Dim*** usa estos argumentos.

Argumento	Descripción
NombreDeVariable	Nombre de una variable

NOTAS:

Índices	*Dim*ensiones de una variable de matriz. Puede declarar múltiples *Dim*ensiones. *Sintaxis* *[inferior* **To** *]superior[.[inferior* **To***] superior]* . La palabra reservada ***To*** proporciona una manera de indicar los límites inferiores y superiores de los índices de una variable matricial.
As tipo	Palabra reservada que se usa para declarar el tipo de datos de una variable. El tipo puede ser un tipo de datos ***Integer, Long, Single, Double, Currency, String*** (para cadenas de longitud variable), ***String*** * longitud (para cadenas de longitud fija), ***Variant***, un tipo definido por el usuario o un tipo de datos de objeto (pero no matrices de objetos). Use una cláusula As tipo separada para cada variable que se está definiendo

Use ***Dim*** en la sección Declaraciones de un módulo para declarar variables que estén disponibles para todos los procedimientos a través de ese módulo

Use ***Dim*** en un procedimiento ***Sub*** o ***Function*** para declarar variables que son locales a ese procedimiento. El colocar instrucciones ***Dim*** al inicio del procedimiento es una práctica de programación de aceptación general.

Use una instrucción ***Dim*** en la sección Declaraciones o en procedimientos ***Sub*** o ***Function*** para declarar el tipo de datos de una variable.

Por ejemplo:

La instruccion siguiente declara la variable como un tipo de datos ***Integer***.

Dim *númeroDeEmpleados As Integer*

También puede usar la instrucción ***Dim*** con paréntesis vacios para declarar matrices dinámicas. Despues de declarar una matriz dinámica, use la instrucción ***ReDim*** dentro de un procedimiento para definir el numero de Dimensiones y elementos de la matriz. Si trata de redeclarar una Dimensión para una variable matricial cuyo tamaño ya se ha declarado, ocurrirá un error

NOTAS:

Las variables se inicializan durante el tiempo de compilación. Las variables numéricas se inicializan con 0, las variables de tipo de datos ***Variant*** con un tipo de datos Vacío. Las cadenas de longitud variable se inicializan como cadenas de longitud cero (**) y las cadenas de longitud fija se rellenan con ceros ANSI (Chr(0)). Los campos de variables de tipos definidos por el usuario se inicializan como si fueran variables separadas.

Instrucción ReDim

Se usa al nivel de procedimiento para declarar variables de una matriz dinámica y asignar o reasignar el espacio de almacenamiento.

Sintaxis

ReDim *[Preserve] nombreDeVariable(índices) [As tipo][, nombreDeVariable(índices) [As tipo]]* .

Argumentos

La instrucción ***ReDim*** usa estos argumentos.

Argumento	*Descripción*
Preserve	Preserva los datos en una matriz existente cuando cambia el tamaño de la última dimensión
NombreDeVariable	Nombre de una variable.
Índices	Dimensiones de una matriz. Puede declarar múltiples dimensiones. *Sintaxis: [inferior* ***To*** *]superior[,[inferior* ***To****] superior]. . .*
As tipo	Palabra reservada usada para declarar el tipo de datos de una variable. El tipo puede ser ***Integer, Long, Single, Double, Currency, String*** (para cadenas de longitud variable), ***String*** * longitud (para cadenas de longitud fija), ***Variant***, o un tipo definido por el usuario. Use una cláusula As tipo separada para cada variable que se está definiendo.

Generalmente la instrucción ***ReDim*** se usa para ajustar el tamaño o para cambiar el tamaño de una matriz dinámica que ya ha sido formalmente declarada usando una instrucción ***Global*** o ***Dim*** con paréntesis vacíos (sin índices de dimensión). Si primero declara una matriz

NOTAS:

dinámica usando una instrucción ***Global*** o ***Dim*** y no incluye índices de dimensión, el máximo número de dimensiones matriciales que puede especificar posteriormente con ***ReDim*** es ocho. Si sus matrices requieren más de ocho dimensiones, puede usar la instrucción ***ReDim*** con un procedimiento para declarar inicialmente una variable de una matriz dinámica local. Así, su matriz puede tener hasta 60 dimensiones.

Puede usar repetidamente la instrucción ***ReDim*** para cambiar el número de elementos de una matriz. Sin embargo, no puede usar ***ReDim*** para cambiar el número de dimensiones de una matriz. Por ejemplo, si declara una matriz que ***consta*** de dos dimensiones (por ejemplo ***ReDim*** A(10,10)), no puede usar ***ReDim*** después para cambiarla a una matriz con tres dimensiones (Por ejemplo ***ReDim*** A(12.12.12.)). De manera similar, no puede declarar una matriz de tipo de datos Integer y luego usar ***ReDim*** para cambiar la matriz a otro tipo de datos.

Si usa la palabra reservada Preserve, cambie el tamaño únicamente de la última Dimensión de la matriz. Por ejemplo, si su matriz tiene una sola Dimensión, puede cambiar el tamaño de esa Dimensión porque es la última y única Dimensión. Sin embargo, si su matriz tiene dos o más Dimensiones, no cambie el tamaño de la primera. Puede cambiar el tamaño solamente de la última Dimensión y preservar el contenido de la matriz.

Por Ejemplo:

A continuación, se muestra cómo puede aumentar el tamaño de la última Dimensión de una matriz dinámica sin borrar ninguno de los datos existentes contenidos en la matriz.

ReDim *x(10 10, 10)*

ReDim *Preserve x(10, 10, 15)*

Instrucción Public

Usado a nivel de modulo para declarar variables globales (variables que se encuentran disponibles para todos los procedimientos en todos los modulos) y asignar espacio de almacenamiento.

Sintaxis

Public nombreDeVariable[([indices])] [As tipo] [, nombreDeVariable[([indices])] [As tipo]]

Argumentos

NOTAS:

La instrucción Public usa estos argumentos.

Argumento	*Descripción*
NombreDeVariable	Nombre de una variable.
Índices	Dimensiones de una matriz. Puede declarar múltiples dimensiones.
As tipo	Palabra reservada usada para declarar el tipo de datos de una variable. El tipo puede ser un tipo de datos ***Integer***, ***Long***, ***Single***, ***Double***, ***Currency***, ***String*** (para cadenas de longitud variable), ***String*** * longitud (para cadenas de longitud fija), ***Variant***, un tipo definido por el usuario o un tipo de datos de objeto (pero no matrices de objetos). Use una cláusula As tipo separada para cada variable que se está definiendo.

Además de declarar variables globales, puede utilizar la instrucción ***Global*** para declarar el tipo de datos de una variable. Por ejemplo, la instrucción siguiente declara la variable como de tipo de datos ***Integer***

Public númeroDeEmpleados As Integer

También puede usar la instrucción Public con paréntesis vacíos para declarar matrices dinámicas. Después de declarar una matriz dinámica, use la instrucción ***ReDim*** dentro de un nivel de procedimiento para definir el número de dimensiones y elementos en la matriz. Si trata de redeclarar una dimensión para una variable matricial cuyo tamaño ya ha sido declarado, ocurrirá un error

Instrucción Static

Se usa a nivel de procedimiento para declarar una variable ***Static*** y asignar el espacio de almacenamiento

Sintaxis

Static nombreDeVariable[([índices])] [As tipo] [, nombreDeVariable[([índices])] [As tipo]]

Use ***Static*** en procedimientos no estáticos para declarar variables ***Static*** explícitamente. Debe usar ***Static*** para declarar una matriz de tamaño fijo en procedimientos no estáticos. En los procedimientos de tipo ***Static***, puede usar ***Static*** o ***Dim*** para declarar las variables de tipo ***Static***.

NOTAS:

También puede usar una instrucción ***Static*** dentro de un procedimiento para declarar el tipo de datos de una variable de tipo ***Static***. Por ejemplo, la declaración siguiente declara una matriz de números enteros de tamaño fijo:

Static númeroEmpleado(200) As Integer

Si trata de usar una instrucción ***Static*** para declarar una variable matricial que ya ha sido declarada, ocurrirá un error y se presentará un mensaje.

Nota: La instrucción ***Static*** y la palabra reservada ***Static*** afectan a la vida útil de las variables en forma diferente Si declara un procedimiento usando la palabra reservada ***Static*** (como por ejemplo en ***Static Sub ContarVentas***), el espacio de almacenamiento para todas las variables locales dentro del procedimiento se asigna una vez y el valor de las variables se preserva por la duración de la ejecución del programa. Para procedimientos no estáticos, el espacio de almacenamiento para las variables se asigna cada vez que se llama el procedimiento y se desasigna al salir del procedimiento La instrucción ***Static*** se usa para declarar variables dentro de procedimientos no estáticos para preservar su valor durante la ejecución del programa.

Instrucción Set

Asigna una referencia de objeto a una variable.

Sintaxis

Set nombreDeVariable = expresiónDeObjeto

Argumentos

La instrucción ***Set*** usa los siguientes argumentos.

Argumento	*Descripción*
NombreDeVariable	Nombre de la variable a la que se está asignando la referencia de objeto. Para que sea válido, ***nombreDeVariable*** se debe haber declarado formalmente (usando una instrucción ***Dim***, ***Global*** o ***Static***) como un tipo de datos de objeto concordante con el objeto que se le está asignando. Los tipos de datos de objeto válidos son Control, ***Database*** (Base de datos) ***DynaSet*** (Hoja de respuestas dinámica), Form (Formulario),

NOTAS:

	QueryDef, ***Report*** (Informe), ***Table*** (Tabla) y ***Snapshot*** (Instantánea)
ExpresiónDeObjeto	Expresión que consiste en el nombre de un objeto, otra variable declarada del mismo tipo de datos de objeto o una función o método que devuelve un objeto.

Por Ejemplo:

El ejemplo siguiente ilustra cómo usar ***Dim*** para declarar formalmente la variable presentación1 como ***DynaSet***. ***Set*** asigna después una referencia a la hoja de respuestas dinámica (***DynaSet***) (creada usando el método ***CreateDynaSet*** en la base de datos Clientes) a la variable presentación1.

Dim *clientes As* ***Database***

Dim *presentación1 As* ***RecordSet***

Set presentación1 = Clientes. ***OpenRecordSet****(ClientesActivos, DB_* ***OPEN****_DYNASET)*

Cuando se usa ***Set*** para asignar una referencia de objeto a una variable, no se crea ninguna copia del objeto para esa variable. En su lugar, se crea una referencia al objeto. Varias variables de objeto pueden referirse al mismo objeto Puesto que estas variables son referencias en lugar de copias del objeto, cualquier cambio en el objeto es también un cambio para todas las variables que se refieren a él.

Instrucción Option Explicit

Se usa para forzar la declaración explícita de todas las variables.

Sintaxis

Option Explicit

Argumentos

La instrucción ***Option Explicit*** se usa en la sección Declaraciones de un módulo para forzar la declaración explícita de todas las variables para ese módulo

Si no utiliza la instrucción ***Option Explicit***, todas las variables no declaradas serán de tipo ***Variant*** (a menos que se haya especificado otro tipo predeterminado con una instrucción

NOTAS:

Deftipo). Cuando se usa la instrucción ***Option Explicit***, todas las variables deben declararse explícitamente antes de ser usadas. Si intenta usar un nombre de una variable sin declarar, ocurrirá un error.

Use ***Option Explicit*** para evitar escribir incorrectamente el nombre de una variable existente o arriesgar una confusión en un código en el que el ámbito de la variable no es claro

Tipos de Datos de Visual Basic

La tabla siguiente muestra los tipos de datos fundamentales que admite Visual Basic, así como el sufijo de declaración de tipo, el tamaño de almacenamiento y el rango de valores de cada uno de ellos.

Tipo de datos	*Sufijo*	*Tamaño de almacenamiento*	*Rango*
Integer	%	2 bytes	De -32.768 a 32.767.
Long(entero largo)	&	4 bytes	De -2.147.483.648 a 2.147 483.647
Single(signo flotante con precisión simple)	!	4 bytes	De -3.402823E38 a -1.401298E-45 para valores negativos: de 1.401298E-45 a 3.402823E38 para valores positivos: y 0.
Double (signo flotante con precisión doble)	#	8 bytes	De -1.79769313486232E308 a -4.94065645841247E-324 para valores negativos. de 4.94065645841247E-324 a 1.79769313486232E308 para valores positivos: y 0.
Currency(entero a escala)	@	8 bytes	De -922.337.203.685.477,5808 a 922.337.203.685.477,5807.
String	$	1 byte por carácter	De 0 a aproximadamente 65.535 bytes. Es necesaria una cantidad adicional de espacio de almacenamiento
Variant	Ninguno	Variable	Cualquier valor numérico hasta el rango del tipo ***Double***, o cualquier

NOTAS:

Tipo de datos	*Sufijo*	*Tamaño de almacenamiento*	*Rango*
			cadena de caracteres.
Definido por el usuario (con *Type*)	**Ninguno**	El requerido por los elementos	El rango de cada elemento es el mismo que el de su tipo de datos fundamental. de entre los anteriores.

NOTAS:

CAPÍTULO 2. FUNDAMENTOS DE PROGRAMACIÓN

CREACIÓN DE ESTRUCTURAS DE DECISIÓN

Instrucción If...Then...Else

Permite la ejecución condicional. basándose en la evaluación de una expresión

Sintaxis 1

If *condición* ***Then*** *entonces* ***[Else*** *sino]*

Sintaxis 2

If *condición1* ***Then***

[bloqueInstruccion-1]

[ElseIf condición2 ***Then***

[bloqueInstruccion-2]]

. . .

[Else

[bloqueInstruccion-n]]

End If

Argumentos de la Sintaxis 1

La instrucción ***If..Then..Else*** en una sola línea suele ser de utilidad para pruebas condicionales sencillas y breves Consta de las siguientes partes.

NOTAS:

Parte	*Descripción*
Condición	Uno de dos tipos de expresiones: Una expresión numérica o de cadena que se evalúa como verdadero (distinto de cero) o falso (0 y valor ***Null***)
entonces. sino	Instrucciones o bifurcaciones efectuadas cuando condición es verdadero (entonces) o falso (sino).

Los campos entonces y sino tienen la siguiente sintaxis:

Sintaxis

*{instrucciones | [**GoTo**] númeroDeLínea | **GoTo** etiquetaDeLínea }*

La sintaxis entonces y sino consta de las siguientes partes.

Parte	*Descripción*
Instrucciones	Una o más instrucciones de Visual Basic. separadas por dos puntos (:)
númeroDeLínea	Un número de línea válido de programa
etiquetaDeLínea	Una etiqueta de línea válida de programa

Observe que ***GoTo*** es opcional con un número de línea, pero es obligatorio con una etiqueta de línea.

Entonces se ejecuta si condición es verdadero o si objeto es del tipo especificado por ***objetoTipo***. Si condición es falso u objeto no es el ***objetoTipo*** especificado, entonces se ejecuta sino. Si la cláusula ***Else*** no está presente. el control pasa a la siguiente instrucción del programa.

Con una condición se pueden tener múltiples instrucciones, pero deben estar en la misma línea y separadas por dos puntos (.) como se muestra en la siguiente instrucción.

__If__ a > 10 __Then__ a = a + 1 . b = b + a · c = c + b

Argumentos Sintaxis 2

La forma de bloque de ***If...Then...Else*** proporciona mayor estructuración y flexibilidad que la forma de línea única y generalmente es más fácil de leer. mantener y depurar. Consta de las siguientes partes.

NOTAS:

Parte	*Descripción*
condición1, condición2	Las mismas condiciones que se usan en la forma de línea única.
BloqueInstrucción1 a n	Una o más instrucciones de Visual Basic en una o más líneas

Al ejecutar un bloque ***If***, Visual Basic comprueba condición1, la primera expresión numérica o expresión ***TypEOF***. Si la expresión es verdadera, entonces se ejecutan las instrucciones que siguen después de ***Then*** Si la primera expresión es falsa entonces Visual Basic comienza a evaluar cada condición ***ElseIf*** en orden. Cuando Visual Basic encuentra una condición verdadera, se ejecutan las instrucciones que siguen después del ***Then*** asociado. Si ninguna de las condiciones ***ElseIf*** es verdadera, se ejecutan las instrucciones que siguen después del ***Else***. Después de ejecutar las instrucciones que siguen a ***Then*** o ***Else*** el programa continúa con la instrucción que sigue después de ***End If***.

Los bloques ***Else*** y ***Else If***, son opcionales. Puede tener tantas cláusulas ***Else If*** como desee en un bloque ***If*** pero ninguno de los dos puede aparecer después de una cláusula ***Else***. Cualquiera de los bloques de instrucciones puede contener instrucciones anidadas del bloque ***If***.

Visual Basic se fija en lo que aparece después de la palabra reservada ***Then*** para determinar si una instrucción ***If*** es o no es un bloque ***If***. Si aparece alguna otra cosa además de un comentario despues de la instrucción ***Then***, la instrucción recibe el mismo tratamiento que una instrucción ***If*** de una sola línea.

Una instrucción de bloque ***If*** debe ser la primera instrucción de una línea. Las partes ***Else***, ***Else If***, y ***End If*** de la instrucción pueden estar precedidas sólo por un número de línea o una etiqueta de línea El bloque debe terminar con una instrucción ***End If***.

Instrucción Select Case

Ejecuta uno de varios bloques de instrucciones dependiendo del valor de una expresión.

Sintaxis

Select Case ***expresiónPrueba***
    *[**Case** listaExpresiones1*
        [bloqueInstrucción-1]]
    *[**Case** listaExpresiones2*
        [bloqueInstrucción-2]]

    *[**Case Else***
        [bloqueInstrucción-n]]

NOTAS:

Si usa la palabra clave ***To*** para indicar un rango de valores, el valor más pequeño debe preceder a ***To***.

Un operador de comparación sólo se puede usar junto con la palabra clave Is.

Si usa ***Case Else***, sus instrucciones asociadas se ejecutan sólo si ***expresiónPrueba*** no concuerda con ninguna otra de las selecciones ***Case***. Aunque no se requiere, se recomienda incluir una instrucción ***Case Else*** en su bloque ***Select Case*** para controlar los valores imprevistos de ***expresiónPrueba***. Cuando no hay ninguna instrucción ***Case Else*** y ninguna expresión enumerada en las cláusulas ***Case*** coincide con ***expresiónPrueba***, la ejecución del programa continúa en la instrucción después de ***End Select***.

Puede usar múltiples expresiones o rangos en cada cláusula ***Case***.

Por Ejemplo:

La siguiente línea es válida:

Case *1* ***To*** *4, 7* ***To*** *9, 11, 13, Is > númeroMáximo*

También puede especificar rangos y múltiples expresiones para las cadenas de caracteres. En el siguiente ejemplo, ***Case*** concuerda con las cadenas que son exactamente iguales a todo, con cadenas comprendidas entre nueces y sopa en orden alfabético y con el valor real de ElementoPrueba$:

Case *"todo", "nueces"* ***To*** *"sopa", elementoPrueba$*

Si ***expresiónPrueba*** coincide con más de una cláusula ***Case***, sólo se ejecutarán aquellas instrucciones que se presentan después de la primera concordancia.

Las instrucciones ***Select Case*** se pueden anidar. Cada instrucción ***Select Case*** debe tener una instrucción ***End Select*** correspondiente

Función IIf

Devuelve uno de dos argumentos, dependiendo de la evaluación de una expresión.

Sintaxis

IIf(expresión, parteVerdadera, parteFalsa)

NOTAS:

Argumentos

La función SiInm (IIf) usa los siguientes argumentos.

Argumento	*Descripción*
expresión	Expresión que desea evaluar.
ParteVerdadera	Valor o expresión devuelta si expresión es ***True*** (-1)
ParteFalsa	Valor o expresión devuelta si expresión es ***False*** (0)

La función ***IIf*** (si inmediato) podría usarse para evaluar una expresión y devolver uno de los dos valores. Por ejemplo, podría usar la función ***IIf*** para examinar un campo en un formulario y determinar si fue un tipo de datos ***Null***. En caso afirmativo, podría hacer que la función devuelva una cadena de caracteres vacía, de lo contrario, devolvería el contenido del campo.

CREACIÓN DE CICLOS

Instrucción For...Next

Repite un grupo de instrucciones el número de veces especificado.

Sintaxis

For *contador = inicio* ***To*** *fin [Step incremento]*

[bloqueInstrucción]

[Exit For]

[bloqueInstrucción]

Next [contador [, contador]]

Argumentos

La instrucción ***For*** usa estos argumentos

NOTAS:

Argumento	*Descripción*
Contador	Variable numérica usada como contador de bucles. La variable no puede ser un elemento matricial ni un elemento de registro
Inicio	Valor inicial del contador.
Fin	Valor final del contador.
Incremento	La cantidad que cambia el contador cada vez que pasa por el bucle. Si no especifica ***Step***. el incremento adopta 1 (uno) como valor predeterminado.
BloqueInstrucción	Cualquier número de instrucciones o métodos que desea ejecutar tantas veces como se especifique.

El valor ***Step*** controla la ejecución del bucle de la siguiente manera.

Cuando Step es	*El bucle se ejecuta si*
Positivo o 0	contador <= fin
Negativo	contador >= fin

Una vez iniciado el bucle y ejecutadas todas las instrucciones del bucle, el valor de ***Step*** se suma a contador En este momento se ejecutan de nuevo las instrucciones del bucle (basandose en la misma condición que causó que el bucle se ejecutara originalmente) .o se sale del bucle y la ejecución continúa con la instrucción que sigue a la instrucción Next.

Sugerencia: La lectura y depuración del programa podrían resultar más difíciles si cambia el valor de contador estando dentro del bucle.

Los bucles ***For***. .Next se pueden anidar colocando un bucle ***For***...Next dentro de otro. Asigne a cada bucle un nombre de variable distinto como contador. La siguiente instrucción es correcta:

```
For i = 1 To 10
    For j = 1 To 10
        For k = 1 To 10
```

NOTAS:

Next k

Next j

Next i

Una instrucción Next con la forma Next k, j, i equivale a la siguiente secuencia de instrucciones:

Next k

Next j

Next i

Nota: Si omite la variable en una instrucción Next, el valor de incremento de Step se suma a la variable asociada con la instrucción ***For*** más reciente. Si se ejecuta una instrucción Next antes de la instrucción ***For*** correspondiente, ocurre un error.

Instrucción Do...Loop

Repite un bloque de instrucciones siempre que una condición sea verdadera o hasta que la condición se vuelva verdadera.

Sintaxis 1

Do *[{**While** | Until} condición]*

[bloqueInstrucción]

[Exit Do]

[bloqueInstrucción]

Loop

Sintaxis 2

Do

[bloqueInstrucción]

[Exit Do]

NOTAS:

[bloqueInstrucción]

Loop *[{**While** | Until} condición]*

Argumentos

El argumento condición es una expresión numérica o una expresión de cadena que se evalúa como verdadera (distinta de cero) o falsa (0 o datos de tipo Nulo) Las líneas del programa entre las instrucciones ***Do*** y ***Loop*** se repiten mientras o hasta que condición sea verdadero.

Una instrucción ***Exit Do***, que sólo se puede usar dentro de una instrucción ***Do ..Loop***, proporciona una alternativa para salir de una estructura de control ***Do...Loop*** Puede colocar cualquier número de instrucciones ***Exit Do*** dentro del bucle ***Do...Loop*** La instrucción ***Exit Do*** que a menudo se usa con la evaluación de alguna condición (por ejemplo ***If Then***), transfiere el control a la instrucción que sigue inmediatamente después de la instrucción ***Loop***. Cuando las instrucciones ***Do...Loop*** están anidadas, el control se transfiere al bucle ***Do ..Loop*** que se encuentre anidado un nivel más arriba que el bucle en el que ocurre ***Exit Do***

Instrucción While...Wend

Ejecuta una serie de instrucciones en un bucle, siempre y cuando una condición determinada sea verdadera.

Sintaxis

While *condición*

[bloqueInstrucción]

Wend

NOTAS:

Argumentos

El argumento condición es una expresión numérica o expresión de cadena que se evalúa como verdadero (distinto de cero) o falso (0 o tipo de datos Nulo)

Si condición es verdadero, entonces se ejecutan todas las instrucciones del ***bloqueInstrucción*** hasta llegar a una instrucción ***Wend***. El control vuelve a la instrucción ***While*** y condición se comprueba de nuevo. Si condición continua siendo verdadero, el proceso se repite De lo contrario, la ejecución se reanuda en la instrucción que sigue después de la instrucción ***Wend***.

Los bucles ***While***, ***Wend*** pueden estar anidados hasta cualquier nivel Cada ***Wend*** coincide con el ***While*** más reciente.

Advertencia: No se bifurque dentro del cuerpo de un bucle ***While***...***Wend*** sin ejecutar una instrucción ***While***, ya que puede causar errores de tiempo de ejecución o problemas con el programa que son difíciles de localizar

CREACIÓN DE ARREGLOS

Instrucción Option Base

Declara el límite inferior predeterminado para los índices de una matriz

Sintaxis

Option Base *número*

Argumentos

La instrucción ***Option Base*** nunca se requiere Sin embargo, si se usa puede aparecer sólo una vez en un módulo y puede ocurrir sólo en la sección Declaraciones. Si elige usar una instrucción ***Option Base***, debe usarse antes de declarar las dimensiones de cualquier matriz.

El valor de número debe ser 0 ó 1. La base predeterminada es 0

NOTAS:

Sugerencia: La cláusula ***To*** en las instrucciones ***Dim***, ***Global*** ***ReDim*** y ***Static*** proporciona una manera más flexible de controlar el rango de los índices de la matriz. Sin embargo, si no establece explícitamente el límite inferior con una cláusula ***To***, puede usar ***Option Base*** para cambiar el límite inferior predeterminado a 1.

Función LBound

Devuelve el índice más pequeño disponible para la dimensión indicada de una matriz.

Sintaxis

LBound*(matriz [,dimensión])*

Argumentos

Esta función sólo se puede usar en el código de Visual Basic

La función ***LBound*** se usa con la función ***UBound*** para determinar el tamaño de una matriz. Use la función ***UBound*** para buscar el límite superior de una dimensión de matriz.

LBound usa los siguientes argumentos.

Argumento	*Descripción*
matriz	Nombre de una variable matricial.
dimensión	Número entero que indica a qué dimensión corresponde el límite inferior devuelto. Use 1 para la primera dimensión, 2 para la segunda y así sucesivamente. Si dimensión se omite, se supondrá como 1

LBound devuelve los valores de la tabla siguiente para una matriz con las siguientes dimensiones:

Dim *a(1* ***To*** *100, 0* ***To*** *3, -3* ***To*** *4)*

Instrucción	*Valor devuelto*
LBound(A, 1)	1

NOTAS:

LBound(A, 2)	0
LBound(A, 3)	-3

El límite inferior predeterminado para cualquier dimensión es 0 o 1 dependiendo de la configuración de la instrucción ***Option Base***.

Las matrices para las que se establecen dimensiones usando la clausula ***To*** en una instrucción ***Dim***, ***Global***, ***ReDim*** o ***Static*** pueden tener cualquier valor entero como límite inferior

Función UBound

Devuelve el mayor índice disponible para la dimensión indicada de una matriz

Sintaxis

UBound*(matriz [, dimensión])*

Argumentos

Esta función se puede usar sólo en el código de Visual Basic

La función ***UBound*** se usa con la función ***LBound*** para determinar el tamaño de una matriz. Use la función ***LBound*** para conocer el límite inferior de una dimensión matricial.

UBound usa estos argumentos

Argumento	*Descripción*
Matriz	Nombre de una variable matricial
Dimensión	Número entero que indica la dimensión a la que corresponde el límite superior devuelto. Use 1 para la primera dimensión, 2 para la segunda dimensión y así sucesivamente. Si dimensión se omite, se supondrá que es 1

UBound devuelve los valores enumerados en la tabla de abajo para una matriz con estas dimensiones:

Dim *A(1* ***To*** *100, 0* ***To*** *3, -3* ***To*** *4)*

NOTAS:

Instrucción	*Valor devuelto*	*Instrucción*	*Valor devuelto*
UBound(A, 1)	100	*UBound*(A, 3)	4
UBound(A, 2)	3		

Instrucción Erase

Reinicializa los elementos de las matrices fijas y desasigna el espacio de almacenamiento de la matriz dinámica.

Sintaxis

Erase *nombmatriz [. nombmatriz] . . .*

Argumentos

El argumento ***nombmatriz*** es el nombre de la matriz a borrar. Es importante saber si una matriz es fija (ordinaria) o dinámica porque ***Erase*** se comporta en forma diferente dependiendo del tipo de matriz. No se recupera ninguna memoria para las matrices fijas ***Erase*** establece los elementos de una matriz fija del siguiente modo.

Tipo	*Efectos de Erase*
Matriz numérica fija	Establece cada elemento como cero
Matriz de cadena fija (longitud variable)	Establece cada elemento como de longitud cero ("")
Matriz de cadena fija (longitud fija)	Establece cada elemento como cero de ANSI (Chr(0))
Matriz fija de tipo de datos *Variant*	Establece cada elemento como un tipo de datos Vacío
Matriz de tipos definidos por el usuario	Establece cada elemento como si fuera una variable separada

NOTAS:

Para las matrices dinámicas ***Erase*** libera la memoria usada por la matriz. Antes de que su programa pueda referirse nuevamente a la matriz dinámica debe redeclarar las dimensiones de la variable matricial usando una instrucción ***ReDim***.

NOTAS:

CAPÍTULO 3. DEPURACIÓN Y MANEJO DE ERRORES

TÉCNICAS DE DEPURACIÓN

Detener la Ejecución del Código de Visual Basic

Cuando la ejecución del código ha sido interrumpida, puede usar las herramientas de depuración de Visual Basic. La ejecución del código se puede interrumpir como resultado de un error en tiempo de ejecución o si usted detiene la ejecución

Detener la Ejecución en una Línea de Código Específica

Pasos a Seguir:

1. Defina un punto de ruptura en la línea de código o inserte una instrucción ***Stop*** justo antes de la línea de código. El programa borra los puntos de ruptura cuando usted cierra una base de datos. Si desea detener la ejecución en el mismo lugar la próxima vez que ejecute el código, inserte una instrucción ***Stop***

2. Ejecute el código Visual Basic detiene la ejecución del código cuando llega al punto de ruptura o a la instrucción ***Stop***. En este punto puede utilizar las herramientas de depuración de Visual Basic para ejecutar paso a paso el código y examinar las variables.

Detener la Ejecución Mientras se Está Ejecutando el Código

Pasos a Seguir:

- Presione las teclas CTRL+INTERRUMPIR

NOTAS:

Seguir Paso a Paso el Código de Visual Basic

Cuando depura el código de Visual Basic puede ver el efecto de cada instrucción recorriendo el código paso a paso. El recorrido paso a paso de las instrucciones a veces conocido como seguimiento, ejecuta una línea de código a la vez

Visual Basic ofrece dos maneras de seguir paso a paso el código Las dos recorren paso a paso el procedimiento activo: la única diferencia entre ellas es la forma en que tratan las llamadas a otros procedimiento Utilice Recorrer todo para recorrer tambien paso a paso los procedimientos llamados, y Recorrer principal para ejecutar los procedimientos llamados como una unidad, pasando a la línea siguiente del procedimiento activo

Pasos a Seguir:

1. Detenga la ejecución de su código.

2 Presione la tecla F8.haga clic en el botón "***Recorrer todo***" de la barra de herramientas o elija Recorrer todo en el ***menú Ejecutar*** para ejecutar paso a paso un procedimiento.

Recorrer El Código Un Procedimiento a la Vez

1. Detenga la ejecución de su código

2. Presione las teclas MAYÚSCULAS+F8. haga clic en el botón "***Recorrer principal***" en la barra de herramientas de la ***ventana Módulo***, o bien, en el ***menú Ejecutar***, elija ***Recorrer principal***.

Definir o Borrar un Punto de Ruptura

Un punto de ruptura le indica a Microsoft Access que se detenga justo antes de ejecutar una línea de código específica. Una vez que el programa ha llegado al punto de ruptura y se ha detenido la ejecución, podrá examinar lo que ha ocurrido hasta ese momento cambiando entre la ventana Módulo y las otras ventanas de su base de datos

Pasos a Seguir:

1. En la ventana módulo, mueva el punto de inserción a una línea de código que no contenga un punto de ruptura.

NOTAS:

2. Presione la tecla F9, haga clic en el botón "***Punto de ruptura***" en la barra de herramientas o bien, en el ***menú Ejecutar***, seleccione Alternar puntos de ruptura Microsoft Access muestra la línea con formato de negrita.

Compile y ejecute el código. La ejecución se detiene después de la instrucción que precede al primer punto de ruptura Una vez detenida, puede depurar el código con las herramientas de depuración de Visual Basic disponibles en el menú Ejecutar Continuar

- Recorrer todo
- Recorrer principal
- Establecer la instrucción siguiente
- Mostrar la instrucción siguiente
- Restablecer

Utilice la Ventana de ejecución para examinar los valores de las variable Haga clic en el botón "***Llamadas***" de la barra de herramientas (o bien seleccione Llamadas en el ***menú Ver***) para mostrar el cuadro de diálogo ***Llamadas***, en el que puede hacer un seguimiento de las llamadas a funciones ejecutadas por el código.

Borrar un Punto de Ruptura

1. En la ventana Módulo, mueva el punto de inserción a una línea de código que tenga un punto de ruptura.

2 Presione la tecla F9, haga clic en el botón "***Punto de ruptura***" en la barra de herramientas de la ventana ***Módulo*** o bien, en el ***menú Ejecutar*** elija Alternar puntos de ruptura. Microsoft Access muestra la línea sin formato de negrita

Puede establecer o borrar un punto de ruptura en tiempo de diseño, o bien cuando la ejecución del código ha sido interrumpida. La ejecución del código puede interrumpirse a causa de un error en tiempo de ejecución, por un punto de ruptura por una instrucción ***Stop*** o al presionar CTRL+INTERRUMPIR mientras se ejecuta el código.

Los puntos de ruptura se borran al cerrar la base de datos Si desea suspender la ejecución en el mismo punto la siguiente vez que abra la base de datos y ejecute el código, inserte una instrucción ***Stop*** inmediatamente antes de la línea de código en la que desee que se detenga la ejecución.

NOTAS:

Sugerencias Para Depurar Código(Tipos de Errores)

Hay tres tipos de errores que pueden darse al ejecutar una aplicación de Microsoft Access:

Errores	*Situación en que ocurren*
Errores en tiempo de compilación	Son el resultado de incorrecciones en el código. Puede tratarse de una falta de correspondencia en una estructura de control, como una instrucción ***Next*** sin el ***For*** asociado; o de errores de programación que violen las reglas de Visual Basic, como una palabra mal escrita, la falta de un separador o una incompatibilidad en el tipo de datos. Entre los errores en tiempo de compilación se cuentan los errores de sintaxis, como pasar a una función un número incorrecto de argumentos o una falta de correspondencia entre los parentesis abiertos y cerrados
Errores en tiempo de ejecución	Suceden después de que comience la ejecución de la aplicación. Entre ellos se encuentra el intento de efectuar una operación ilegal, como escribir en un archivo inexistente, o dividir entre cero
Errores lógicos	Suceden cuando el programa puede compilarse y ejecutarse, pero no se comporta como se esperaba.

Para encontrar y solucionar los errores de los programas en Visual Basic, pruebe las siguientes técnicas de depuración.

- Imprima el código, si encuentra más fácil leerlo en copia impresa que en la pantalla
- Ejecute la aplicación para encontrar los puntos problemáticos
- Ejecute el código desde la Ventana de ejecución hasta que un error lo detenga, o bien suspenda la ejecución manualmente en el momento en que piensa que se produce el error
- Resuelva tantos errores en tiempo de compilación y de ejecución como sea posible.
- Continúe la ejecución
- Utilice las herramientas y técnicas de depuración de Visual Basic para aislar los errores y hacer un seguimiento de la ejecución del código.

NOTAS:

- Establezca puntos de ruptura para suspender la ejecución en puntos determinados y buscar en ellos tipos o nombres de variables incorrectos. errores en las comparaciones logicas. bucles infinitos, salidas truncadas, problemas con las matrices etcétera. Con los puntos de ruptura también puede controlar el flujo del código o examinar la información en un momento determinado.

- Utilice las instrucciones de impresión de la Ventana de ejecución para comprobar el valor de expresiones o variables determinadas al irse ejecutando el código En la ventana Módulo puede observar el valor de una variable o expresión. e incluso modificarla directamente mientras la ejecución del código está suspendida.

- Siga el código paso a paso haciendo clic en los botones "***Recorrer todo***" o "***Recorrer principal***" de la barra de herramientas para observar la aplicación mientras se ejecuta. Haga clic en el botón "***Llamadas***" cuando el código esté suspendido para activar el cuadro de diálogo Llamadas, que hace un seguimiento de la secuencia en que se llama a los procedimientos.

- Busque primero las soluciones sencillas· muchos errores son originados por pasar por alto cuestiones simples

- Asegúrese de que no utiliza el nombre de un control en su expresion· es lo que se conoce como referencia circular

- Intente dividir el código en partes más pequeñas y manejables. Con ello facilitará las pruebas, y podrá localizar los errores siguiendo un proceso de eliminación.

MANEJO DE ERRORES

Instrucción On Error

Activa una rutina de control de error y especifica la ubicación de esta rutina dentro de un procedimiento: puede usarse para desactivar una rutina de control de error

Sintaxis

*On Error { **GoTo** línea | Resume Next | **GoTo** 0 }*

NOTAS:

Argumentos

Si no usa una instrucción On Error, cualquier error en tiempo de ejecución que ocurra sera irrecuperable: es decir. Visual Basic generará un mensaje de error y finalizará la ejecución del programa.

La instrucción On Error consta de las siguientes partes.

Parte	*Descripción*
***GoTo* linea**	Activa la rutina de control de error que comienza en linea (una etiqueta de línea o un número de linea) Después, si ocurre un error en tiempo de ejecución, el control del programa se bifurca a linea. La linea especificada debe estar en el mismo procedimiento que la instrucción On Error De lo contrario, ocurre un error en tiempo de compilación
Resume Next	Especifica que cuando ocurra un error en tiempo de ejecución, el control se dirija a la instrucción situada inmediatamente después de la instrucción en la que ocurrió el error. En otras palabras, el código continúa su ejecución. Puede usar la función Err en las lineas de código posteriores para obtener el número de error en tiempo de ejecución.
***GoTo* 0**	Desactiva cualquier controlador de error activado en el procedimiento actual.

Un controlador de error está activado cuando una instrucción ***On Error GoTo*** linea hace referencia a dicho controlador Una vez que se ha activado un controlador de error, cualquier error en tiempo de ejecución provoca que el control del programa salte a la rutina de control de error activada y active el controlador de error Un controlador de error permanece activo desde el momento en que se intercepta un error en tiempo de ejecución hasta que se ejecuta una instrucción Resume, ***Exit Sub*** o ***Exit*** Function en el controlador de error.

Si ocurre un error mientras un controlador de error está activo (entre el momento en que surge el error y la ejecución de una instrucción Resume, ***Exit Sub*** o ***Exit*** Function), el controlador de error del procedimiento activo no podrá controlar el error Si el procedimiento de llamada tiene activado un controlador de error, el control se devuelve al procedimiento de llamada y su controlador de error se activa para controlar el error. Si el controlador de error del procedimiento de llamada también está activo, el control se pasa hacia atrás a través de todos los procedimientos de llamada anteriores hasta encontrar un controlador de error inactivo. Si no se encuentra ningún controlador de error inactivo, el error será irrecuperable en el punto donde ocurrió. Cada vez que el controlador de error vuelve a transferir el control al procedimiento de llamada, ese procedimiento pasa a ser el procedimiento activo. Una vez que un error ha sido

NOTAS:

controlado por un controlador de error en cualquier procedimiento. la ejecución del programa se reanuda en el procedimiento activo en el lugar designado por la instruccion Resume

Observe que una rutina de control de error no es un procedimiento *Sub* o ***Function*** Es un bloque de códigos marcado por una etiqueta de linea o número de linea.

Las rutinas de control de error dependen del valor en ***Err*** para determinar la causa del error. La rutina de control de error debe comprobar o guardar este valor antes de que pueda ocurrir cualquier otro error o antes de invocar un procedimiento que podría causar un error. El valor en ***Err*** sólo refleja el error más reciente. La función Error[$] se puede usar para devolver un mensaje de error asociado con cualquier número de error en tiempo de ejecución determinado devuelto por ***Err***

On Error Resume ***Next*** hace que la ejecución del programa continúe con la instrucción situada inmediatamente después de la instrucción que causó el error en tiempo de ejecución Esto permite que su programa continúe a pesar de un error en tiempo de ejecucion y verificar posteriormente la causa del error. ***On Error*** Resume ***Next*** también le permite crear la rutina de control de error en linea con el procedimiento en lugar de transferir el control a otro lugar dentro del procedimiento.

On Error GoTo 0 desactiva el control de error en el procedimiento activo. No especifica la línea 0 como el comienzo del código de control de error. aunque el procedimiento contenga una línea numerada 0. Sin una instrucción ***On Error GoTo*** 0, un controlador de error se desactiva automáticamente cuando se sale de un procedimiento

Para impedir que el código de control de error se ejecute cuando no ha ocurrido ningún error. coloque una instrucción ***Exit Sub*** o ***Exit*** Function inmediatamente antes de la rutina de control de error. como en el siguiente ejemplo:

```
Sub inicializarMatriz(var1, var2, var3 var4)

    On Error GoTo controladorError


    Exit Sub

controladorError:

    ...

    Resume Next
```

NOTAS:

End Sub

Aquí, el código de control de error aparece después de la instrucción ***Exit Sub*** y antes de la instrucción ***End Sub*** para separarla del flujo de ejecución normal del procedimiento. Esta no es la única solución. El código de control de error se puede colocar en cualquier lugar de un procedimiento.

Instrucción Resume

Reanuda la ejecución del programa después de terminar una rutina de control de error.

Sintaxis

Resume {[0] | Next | línea}

Argumentos

Las diferentes formas de la instrucción Resume redirigen el flujo del programa como se describe a continuación.

Instrucción	*Descripción*
Resume [0]	La ejecución del programa se reanuda en la instrucción que causó el error, o bien en la llamada del procedimiento ejecutada más recientemente que contenga la rutina de control de error
Resume Next	La ejecución se reanuda con la instrucción que sigue inmediatamente a la que causó el error o con la instrucción que sigue inmediatamente a la llamada ejecutada más recientemente desde el procedimiento que contiene la rutina de control de error.
Resume línea	La ejecución se reanuda en línea, que es una etiqueta de línea o número de línea. El argumento línea debe estar en el mismo procedimiento que el controlador de error.

El lugar donde se reanuda la ejecución está determinado por la ubicación del controlador de error en el que se ha interceptado el error y no necesariamente por la ubicación del error en sí.

La tabla siguiente resume el lugar donde un programa reanuda su ejecución cuando se usa la instrucción Resume [0].

NOTAS:

Dónde ocurre el error	*Dónde se reanuda el programa*
El mismo procedimiento que el controlador de error	Instrucción que causó el error
Procedimiento diferente al del controlador de error	Última instrucción que se ejecutó en el procedimiento que contiene el controlador de error.

Si usa una instrucción Resume en cualquier parte del programa que no sea una rutina de control de error, ocurrirá un error.

Cuando una rutina de control de error está activa y se llega al final del procedimiento (una instrucción ***End Sub*** o ***End*** Function) antes de que se ejecute la instrucción Resume, surge un error porque se supondrá que se ha cometido inadvertidamente un error lógico. Sin embargo, si se encuentra una instrucción ***Exit Sub*** o ***Exit*** Function mientras un controlador de error está activo, no ocurrirá ningún error porque se considerará como una redirección deliberada del flujo del programa

NOTAS:

Funciones Err, Erl

Devuelven el estado de error.

Sintaxis

Err

LíneaErr

Erl

Argumentos

Cuando ocurre un error, la función ***Err*** devuelve un código de error de tipo de datos ***Integer*** en tiempo de ejecución que identifica el error. La función ***LíneaErr*** (***Erl***) devuelve un tipo de datos ***Integer*** que es el número de línea de la línea en que ocurrió el error o la línea anterior más próxima.

Puesto que ***Err*** y ***LíneaErr*** (Erl) solamente devuelven valores significativos una vez que ha ocurrido un error, generalmente se utilizan en rutinas de control de error para determinar el error y la acción correctiva. ***Err*** y ***LíneaErr*** (Erl) se restablecen al valor 0 después de cualquier forma de la instrucción Resume u ***On Error*** y después de una instrucción ***Exit Sub*** o ***Exit*** Function dentro de una rutina de control de error.

Si establece un controlador de errores usando ***On Error GoTo*** y ese controlador de errores llama a otro procedimiento, existe la posibilidad de que el valor de ***Err*** y ***LineaErr*** (Erl) se restablezca como 0. Para asegurarse de que el valor no cambie, asigne los valores de ***Err*** o ***LíneaErr*** (Erl) a variables antes de llamar a otro procedimiento o antes de ejecutar Resume. On Error, ***Exit Sub*** o ***Exit*** Function.

El valor devuelto por la función ***Err*** puede establecerse directamente usando la instrucción ***Err***. Los valores de ***Err*** y ***LíneaErr*** (***Erl***) se pueden establecer indirectamente usando la instrucción Error.

La función ***LíneaErr*** (***Erl***) devuelve sólo un número de línea, no una etiqueta de línea, ubicado en la línea que produce el error, o antes de ella. Los números de línea mayores que 65.529 se consideran como etiquetas de línea y no pueden ser devueltos por ***LíneaErr*** (***Erl***). Si su programa no tiene números de línea o si no hay un número de línea antes del punto en el que ocurrió el error, ***LíneaErr*** (***Erl***) devolverá 0.

NOTAS: