PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR SEDE AMBATO

Facultad de Ingeniería
Escuela de Sistemas

TESIS DE GRADO PREVIA LA OBTENCION DEL TITULO DE INGENIERO DE SISTEMAS

"Ingeniería de Software para medicina utilizando técnicas Orientadas a Objetos"

Jorge Andrade Ortiz
Patricio Erazo Calderón

DIRECTOR DE TESIS: Ms.C. ROXANA MERIÑO M.

Ambato, 1998

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR SEDE AMBATO

Facultad de Ingeniería
Escuela de Sistemas

TESIS DE GRADO PREVIA LA OBTENCION DEL TITULO DE INGENIERO DE SISTEMAS

FACULTAD DE INGENIERIA
ESCUELA DE SISTEMAS

"Ingeniería de Software para medicina utilizando técnicas Orientadas a Objetos"

Director: ----------
Ing. Ms.C. Roxana Meriño M.

Revisores: ----------
Ing. Patricio Chambers

Ing. Wigberto Sánchez

Jorge Andrade Ortíz
Patricio Erazo Calderón

Dedicatoria

A mi dulce esposa, la luz de mi camino y al hijo que está por nacer.

Jorge Andrade

Dedico este trabajo a mi esposa y a mis hijos por que ellos son la razón que impulso ha seguir adelante.

Patricio Erazo

AGRADECIMIENTO

Queremos presentar un agradecimiento muy especial a la Pontificia Universidad Católica del Ecuador Sede Ambato en cuyas aulas forjamos nuestro porvenir. Un agradecimiento muy especial a nuestros distinguidos profesores en especial a la Ing. Ms.C. Roxana Meriño, al Ing. Patricio Chambers y al Ing. Wigberto Sanchez, por sus valiosos conocimientos

INDICE

INDICE GENERAL

INTRODUCCION

Hoy en día el creciente y vertiginoso desarrollo de la ciencia de la computación ha hecho que esta forme parte de la vida cotidiana de cada una de las otras ciencias y sus aplicaciones, sin embargo, en nuestro país poco o ninguno ha sido el avance en el desarrollo de software para la ciencia médica, trayendo como consecuencia, que el gran número de problemas susceptibles a la automatización sean resueltos, en el mejor de los casos; por medio de software importado de países desarrollados con sus elevadísimos costos, más aún, hoy en día con el advenimiento de sistemas operativos como WINDOWS 95 y WINDOWS 98 la automatización de sistemas se aprovecha enormemente de la programación visual, la tecnología multimedia (fotos, sonidos, etc.), redes locales, Internet, etc. en fin, el paradigma actual es: utilización de tecnología de punta, como las técnicas orientadas a objetos, para explotar al máximo las bondades del sistema operativo para desarrollar software con calidad total, capaz de competir eficientemente con el producido en países desarrollados.

El alcance de este proyecto prevé la inclusión y adaptación de la Ingeniería de Software y de las técnicas avanzadas de programación; específicamente la Programación Orientada a Objetos, en la solución de problemas relacionados con la medicina. Se ejemplificará a través de dos aplicaciones que solucionan diferentes problemas propios del medio médico como son un sistema para la Digipuntura y un sistema automático para Laboratorios Clínicos, como herramienta de desarrollo se utilizará Borland Delphi que es un ambiente de desarrollo de aplicaciones orientado a objetos que trabaja bajo el Sistema Operativo Windows 95' y tiene como lenguaje de programación, el "Object Pascal".

El sistema de Digipuntura

Este programa permite la prevención y tratamiento con fines curativos de múltiples enfermedades casuales o patológicas, el alivio rápido y efectivo de un dolor físico a causa de un golpe o padecimiento y el regulamiento energético de sus funciones vitales, con la simple presión de un dedo. Todo de una manera sencilla, dinámica y efectiva basado en un método de 18 puntos.

Puede ser usado en el hogar o la oficina y por cualquier persona de cualquier edad, sin distinción de status ni de capacidad, puesto que tanto el sistema como sus demás componentes son totalmente inofensivos y el vocabulario utilizado es básico y efectivo. Así puede usarse tanto por un niño, como por una secretaria de oficina o por cualquier persona que tenga acceso a una IBM PC o compatible.

En el plano profesional puede ser usado para diágnosis, anestesia, tratamiento para adicciones y balance de la energía.

Junto con el programa se ha creado un dispositivo electrónico llamado DIGITEC que le asistirá en la búsqueda y detección de los 18 puntos en los cuales se basa todo el principio de estabilidad energética corporal de este sistema de digipuntura.

Sistema Automático de laboratorio

Este programa permite la administración de la información de laboratorios clínicos de manera sencilla, dinámica y efectiva.

Puede ser usado por cualquier profesional de la salud vinculado a laboratorios clínicos, ya que su interfaz de usuario es intuitiva y gráfica. El programa permite llevar el control de cualquier laboratorio clínico. Cuenta con un conjunto de listas de pruebas para los exámenes más comunes, que pueden ser enriquecidas por el laboratorista y permite llevar el control de exámenes en una base de datos.

Estos proyecto sentará las bases para incentivar la elaboración de software médico, evidenciando cómo el uso de la ingeniería de software y de las técnicas orientadas a objeto, facilita su desarrollo.

CAPITULO I

INGENIERÍA DE SOFTWARE

En los cambios que el mundo ha experimentado en la presente década las ciencias computacionales desempeñaron y desempeñan un rol importantísimo. El mundo ha sido testigo del enorme impacto de ese "mágico" aparato, la computadora, de como éste se ha convertido en una herramienta importante para una gran cantidad de actividades humanas, en tan solo unos años un instrumento de uso exclusivo para el desarrollo de las ciencias a pasado a ser parte integrante del trabajo empresarial, del control automatizado de procesos, en la investigación científica y no solo eso; sino que ha incursionado en la oficina y en el hogar. La continua evolución en cuanto a hardware se refiere, la propagación e impacto de la red de redes, INTERNET, son pruebas fehacientes del imparable avance de esta ciencia. En cuanto ha software se refiere uno de los aspectos relevantes ocurrido en las últimas décadas fue el cambio de enfoque de la programación, lo que muchos llaman "El Paradigma de la Programación". La forma de programar a migrado de la Programación en modo texto, bajo DOS, a la Programación Visual, bajo WINDOWS, de la Programación Estructurada a la Programación Orientada a Objetos (OOP).

El sistema operativo Windows 95 es la versión más difundida del sistema operativo Microsoft Windows establecida en el mercado. Es el sucesor del sistema operativo MS-DOS, de la versión 3.1 de Windows y de la versión 3.x de Windows para trabajo en grupo. Windows 95 ha sido diseñado para proporcionar a los administradores de redes y a los profesionales de soporte de sistemas una gran diversidad de potentes herramientas y

utilidades, que permiten una mejor administración de las computadoras personales y reducen los costos de mantenimiento. Adicionalmente Windows 95 presenta ciertas características novedosas y una interfaz de usuario mejorada que ayuda a los usuarios a mejorar su productividad. Todas estas características de este nuevo sistema operativo sólo pueden ser aprovechadas en un cien por ciento cuando las diferentes aplicaciones que trabajan bajo el mismo hayan sido creadas para éste sistema.

El concepto de programación orientada objetos, OOP (Object Oriented Programming), no es nuevo. La programación orientada a objetos se basa en la idea natural de la existencia de un mundo lleno de objetos, de modo que la resolución del problema se realiza en términos de objetos. En consecuencia, un lenguaje se dice que está basado en objetos si soporta objetos como característica fundamental de un lenguaje.

OOP entraña una metodología de diseño de programas, que ha tenido un gran impacto en la ingeniería del software, por esa razón la utilizaremos en el presente proyecto. El diseño y la programación OOP ayuda considerablemente a los ingenieros de los programas. La herramienta de desarrollo de aplicaciones visual DELPHI presenta características OOP. Siendo ésta una de las características esenciales del sistema operativo Windows 95 de tal forma que un sistema desarrollado en DELPHI aprovechará eficientemente las ventajas de éste nuevo Sistema Operativo.

En el Ecuador, existen muchas ramas del conocimiento y de la industria que no han sido automatizadas utilizando las bondades de la computadora y de las nuevas técnicas de desarrollo de aplicaciones. Una de esas ramas es la medicina. Las instituciones médicas tanto

estatales como particulares, carecen parcial o totalmente de la automatización de sus procesos y es la medicina por su diversidad de tareas y su responsabilidad social, un ejemplo crucial de necesidad de automatización.

A nivel internacional existen diversos sistemas o programas que cubren determinadas necesidades del área médica, pero además de no estar personalizado o adaptado a nuestro medio, tienen precios muy elevados. Este trabajo pretende exponer de forma teórica y práctica, cómo pueden ser diseñadas las aplicaciones utilizando la Ingeniería de software y las técnicas orientadas a objeto y su aplicación en el área médica.

En el presente proyecto se aplicarán estas tecnologías en el desarrollo de DOS SISTEMAS AUTOMATIZADOS PARA MEDICINA con objetivos específicos dentro del área médica:

- Sistema automático para la Dígipuntura. (Técnica científica oriental de medicina alternativa)
- Sistema automático para laboratorios clínicos.

El uso de las técnicas orientadas a objetos en la ingeniería de software permite desarrollar de forma eficiente, atractiva y novedosa; aplicaciones médicas para dar solución a problemas reales de esta medio.

1.1 HACIA DONDE APUNTA LA INGENIERIA

En la década del sesenta, con la introducción de la computación en diversas aplicaciones, se generó una demanda de software difícil de satisfacer. Las técnicas de programación de los analistas en los Centros de Cálculos distaban de ser una metodología

coherente, pareciéndose más bien a la "magia negra". Los costos de desarrollo y mantenimiento del software crecieron descontroladamente, produciéndose lo que ha sido llamado "la crisis del software".

Para remediar esta situación fue necesario un cambio en el enfoque de la programación. Se dejó de pensar en los programas como una secuencia de instrucciones y se comenzó a representarlos como una colección de módulos interactuantes. Se diseñaron lenguajes de programación que tuvieron como base la simplicidad y la estructura. Surgió así la disciplina de ***Ingeniería de Software***. Fueron propuestas toda una gama de metodologías para el diseño y la programación estructurada como la base para el desarrollo de aplicaciones.

Sin embargo, esta crisis se ha mantenido hasta nuestros días. La construcción de sistemas ambiciosos es muy cara, es difícil que resulten altamente confiables y a veces imposibles de manipular. Es el momento de revolucionar la forma en que construimos software de manera similar a como lo hacen los ingenieros de hardware cada 5 años. Esta revolución, según plantean los especialistas, es la Programación Orientada a Objetos (POO).

La POO es una metodología de propósito general que puede aplicarse eficientemente para afrontar cualquier proyecto software. Su objetivo es ayudar a los programadores, en menos tiempo, a hacer programas más estables, claros y mantenibles.

La POO es una nueva tecnología de programación, un nuevo estilo en el diseño e implementación de programas, introduciendo conceptos que son radicalmente diferentes a los de la programación procedural convencional. La POO es una nueva forma de pensar acerca de programas y sus estructuras, y una nueva forma de escribir programas.

La programación Orientada a Objetos tiene sus orígenes en SIMULA, un lenguaje desarrollado para aplicaciones de simulación en 1967. Sin embargo pasó mucho tiempo para ser tomada en cuenta, antes se impuso la programación estructurada como solución a la llamada crisis del software.

La POO no modela los problemas convirtiéndolos en algo familiar a las computadoras, sino todo lo contrario; acerca a la computadora a los problemas del mundo real. Se basa en un mundo de objetos individuales que interactúan entre sí y sus fronteras solo están determinadas por la naturaleza del problema. Esto es lo que caracteriza la sencillez de los conceptos de la POO.

Incorporar esta nueva filosofía en el planteamiento de la ingeniería y acoplarla al desarrollo de software para medicina, requiere que se consideren por lo menos las siguientes etapas previas al ciclo de vida del sotware ya conosido ampliamente:

1.2 PLANEACION

Todo proceso de ingeniería debe partir de la planeación, es indiscutible que para la realización de un proyecto la presión de tiempo es factor muy importe ya que influye directamente en el desarrollo del mismo de forma que si el proyecto se planifica y se programa al azar, los riesgos sólo se consideran, una vez que se convierten de repente en problemas reales.

El primer peso de la planificación es la ESTIMACION, la cual nos dará los elementos y la información necesaria para llevar a cabo las restantes actividades. La estimación más importante es la del esfuerzo y la del tiempo, las cuales nos darán parámetros importantes para continuar con la planificación.

La planeación o planificación debe realizare tanto de los recursos como de los Costes del Proyecto Software.

Sin duda la estimación conlleva varios factores de riesgo que podrían dar una visualización errónea del proyecto, los mismos que son: La complejidad del proyecto, el tamaño del Proyecto, el grado de estructuración del proyecto.

Otro paso importante en la planeación es definir el ámbito del Sistema, para lo cual se procede a enunciar el problema de la forma más clara y comprensible, evitando ambigüedades, es decir se delimitará el ámbito del problema estableciendo la función, el rendimiento, las restricciones, las interfases y la fiabilidad.

1.3 ESTIMACION DE RECURSOS

El primer paso para la determinación de los Recursos Humanos es determinar los requisitos o habilidades necesarias para iniciar el proyecto de desarrollo de software, además de la disponibilidad, duración de las tareas y fecha de comienzo.

1.4 ESTIMACIÓN DE LA REUSABILIDAD

Todo sistema de información debe ser reusable y mejorable, para lograr este objetivo se plantea la realización del análisis e implementación orientada a objetos, donde se estandariza el concepto de reusabilidad por medio de criterios como la herencia y la definición de tipos de datos abstractos.

1.5 ESTIMACIÓN DEL PROYECTO SOFTWARE

Generalmente se propone la estimación del producto software a raíz del coste; como no es lo único daremos otras estimaciones que influyen también en el proyecto y que proporcionan parámetros importantes para la evaluación del mismo.

Los datos de LDC (líneas de código) y de PF (puntos de función) se emplean de dos formas durante la estimación del proyecto de software:

1. Como variables de estimación utilizadas para cambiar cada elemento del software.
2. Como métricas de base recogidas de proyectos anteriores, utilizadas junto con las variables de estimación para desarrollar proyecciones de coste y esfuerzo.

Para calcular la estimación de las LDC se tomará en cuenta dos parámetros importantes que son: a) La Complejidad del Proyecto y b) El tipo de Lenguaje utilizado en la implementación.

En lo referente al punto a, el proyecto constará de 2 partes: La elaboración del módulo y la instalación modular. En la fase de elaboración modular, el nivel de complejidad mayor lo tiene el desarrollo del software. Basado en el hecho de trabajar en un LOO (lenguaje orientado a objetos).

Haciendo un estimativo de ambos parámetros se pueden señalar como valor de referencia 500 LDC es decir O, 5 KLDC. (para efectos de ejemplificar los modelos de cálculo, se ha tomado este valor de referencia sacado del estimativo LDC para el software de Acupuntura) Y por otro lado los PF (puntos de función) que se los obtiene utilizando una relación

empírica basada en medidas cuantitativas del dominio de información y valores subjetivos de la complejidad del software. Para optener los PF se puede proceder de la siguiente manera: Primero se asigna valores del 0 al 10 a cada uno de los parámetros de evaluación para asociarles un peso relativo y luego se multiplica por el número de veces que incurre dicho parámetro en el sistema. Los parámetyros de evaluación son: número de entradas del usuario, número de salidas del usuario, número de peticiones al usuario, número de temas y número de interfases externas. La sumatoria de estos resultados nos da un valor llamado Cuenta-Total.

Se calculan los puntos de función por medio de la siguiente relación:

PF = cuenta - total x [0,65 + 0,01 x $\sum$ (Fi)]

donde (Fi) son valores que van desde 1 hasta 14 y representan los valores de ajuste de complejidad cuyo resultado viene de las respuestas de la siguiente tabla con 14 preguntas:

Evaluar cada factor en una escala del 0 al 5 donde:

0 = Sin influencia

1= Incidental

2= Moderado

3= Medio

4= Significativo

5= Esencial

Para efecto del estudio tomaremos como referente el sistema de Acupuntura, así tenemos lo siguiente:

TABLA PARA CALCULAR VALORES DE AJUSTE DE COMPLEJIDAD

¿Requiere el sistema copias de seguridad y de recuperación fiables?	= 0
¿Se requiere comunicaciones de Datos	= 5
¿Existen funciones de procesamiento distribuidos?	= 0
¿Es crítico el rendimiento?	= 3
¿Será ejecutado en un entorno operativo conocido y fuertemente utilizado?	= 0
¿Requiere el sistema entrada de datos interactiva?	= 1
¿Requiere la entrada de datos interactiva que las transacciones de entrada? Se lleven a cabo sobre múltiples operaciones?	=1
¿Se actualizan los archivos maestros de forma interactiva?	= 0
¿ Son complejas las entradas, salidas, archivos y peticiones?	= 0
¿ Es complejo el procesamiento interno?	= 5
¿ Se ha diseñado el código para ser reutilizable?	= 4
¿ Están incluidas en el diseño la conversión y la instalación?	= 5
¿ Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?	= 0
¿ Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizado por el usuario?	= 0
Total o Sumatoria de (fi)	24

Por lo tanto partiendo de la fórmula anterior obtenemos:

cuenta - total = 19;

$\sum(Fi) = 24$, entonces $\rightarrow$ PF = 19x[0,65 + 0,01x24];

PF = 16, 91.

Por tanto partiremos de éstos puntos en la evaluación de las diferentes métricas que intervienen en un proyecto software y que son indispensables evaluarlas, ya que proporciona un índice real de avance o retroceso del proyecto, Evaluaremos entonces:

La productividad , l a calidad y por último el coste y la documentación.

PRIMERO: PRODUCTIVIDAD

Para este cálculo partimos del tiempo aproximado necesario para la *programación del software* que en este caso y de acuerdo a la planificación previa (anteproyecto del Software) es de 40 días. Su envergadura implica la utilización de alrededor O, 5 KLDC Y considerando que, las tareas las realizan dos personas podemos decir que: el trabajo es de dos_ personas_ mes:

Por 1o tanto:

$$\text{Productividad} = \frac{\text{KLDC}}{\text{persona\_ mes x número\_ meses}} = \frac{500}{2 \times 1} = 250 \frac{\text{LDC}}{\text{persona - mes}}$$

$$\text{Productividad} = \frac{\text{PF}}{\text{persona \_ mes}} = \frac{16{,}91}{2} = 8.4 \text{ PF / pm}$$

SEGUNDO CALIDAD:

Se estima la calidad del Software a sabiendas que se producirán un estimado de 10 errores en total por cada KLDC; (se puede sugerir que la métrica de calidad se la realice también luego de las fase de implementación, durante la prueba ya que se debería tener un valor más ajustado a la realidad de esta métrica).

$$\text{Calidad} = \frac{\text{Errores}}{\text{KLDC}} = \frac{10 \text{ err.}}{500 \text{ LDC}} = 0.02 \text{ err / LDC.}$$

$$\text{Calidad} = \frac{\text{Errores}}{\text{PF}} = \frac{10 \text{ err.}}{16,91 \text{ PFs}} = 0.591 \text{ err / PF.}$$

TERCERO: ESTIMACION DEL COSTE Y LA DOCUMENTACION:

De la misma manera se puede estimar el coste y la documentación a partir de los parámetros del KLDC y PF tenemos.

1. Se calcula el costo aproximado en Sucres de la utilización de máquina por día Y de la adquisición de documentación:

Costo de máquina por hora = $1000.

Número de horas utilizadas de máquina (planificadas) = 200 horas.

Unidad de disco 3.5" -- $3000.

Documentación adquirida y gastos varios = $200000.

$$\text{Coste} = \frac{\text{S/. Sucres}}{\text{KLDC}} = \frac{\text{S/. 403000}}{\text{500 LDC}} = \text{S/. 806 sucres / LDC.}$$

$$\text{Coste} = \frac{\text{S/. Sucres}}{\text{PF}} = \frac{\text{S/. 25.000}}{\text{16,91}} = \text{S/. 1478,41sucres / PF.}$$

Se deja la métrica de la documentación para la fase de prueba porque en esta instancia del desarrollo del proyecto la documentación esta casi completa

Como conclusión de las métricas obtenidas a partir de los parámetros LDC y PF podemos realizar una comparación en función de estimaciones informales del número medio de líneas de código requeridas para construir un punto de función en un lenguaje Orientado a Objetos que es de 30.

Lenguajes Orientados a Objetos ⇒ 30 LDC por PF

En nuestro sistema tenemos los siguientes resultados:

X= 500 / 30 = 16, 12 Puntos de Función para nuestro Sistema lo que coincide aproximadamente con lo obtenido en el cálculo de puntos de función previo es 16, 91 es decir:

16, 12 ≈ 16,91

La estimación del *Esfuerzo* (E) y el tiempo de *desarrollo* (D) en meses cronológicos se realiza por el modelo COCOMO básico.

Hay que tomar en cuenta que el Lenguaje de implementación es un Lenguaje O-O, y que la cantidad de líneas de código podría variar de alguna manera (específicamente reducirse o reciclarse mediante clases) por lo que se toma un estimativo de rango de las líneas.

En este caso tenemos:

Esfuerzo:

$E = a_b (KZDC)\ b_b;$

Desarrollo (tiempo de desarrollo en meses):

$D = c_b (E)\ d_b$

Tomando en cuenta que el Proyecto de Software es de tipo orgánico se utilizan los valores respectivos especificados en la siguiente tabla:

Proyecto Software	a_b	b_b	c_b	d_b
Orgánico	2,4	1,05	2,5	0,38

$$E = a_b (0,5)^{1.05} = 1,1 \text{ persona - mes}$$

$$D = c_b (E)^{0,38} = 2,5 \text{ meses}$$

Lo que se aproxima mucho a la realidad, ya que conocemos que el Proyecto es grupal e implica el trabajo de dos personas por mes. En este caso este modelo de ingeniería nos hace ver que el trabajo planeado está dentro del nivel esperado en el esfuerzo del grupo de trabajo; y nos da un valor estimado de 2, 5 meses de tiempo para la programación del software.

1.6 ANALISIS DE RIESGOS

Identificación de Riesgos.- Se tratará de identificar los riesgos eventuales del proyecto, así como los riesgos técnicos:

Indisponibilidad de tiempo.- Ya que podrían presentarse durante el desarrollo del software eventos de importancia que tal vez abarquen un intervalo de tiempo considerado que desfase la agenda de trabajo.

Indisponibilidad de máquina.- el diseño e implementación requiere un tiempo de utilización de máquina del 90%, de los cuales con una estimación optimista es posible obtener un 60% en todo el lapso previsto para esta fase.

Riesgos de Clientes y Requisitos.- puede ser que el sistema no satisfaga los requisitos potenciales de usuarios finales

ESTIMACION DE RIESGOS.- De los riesgos anotados se realiza la siguiente estimación:

Evaluación de riesgo.- Se trata de obtener un punto de referencia para evaluar si el riesgo que tenemos excede a este punto de manera que el impacto que tenga afecte en algo nuestro proyecto, momento en el que se debe tomar las medidas preventivas necesarias Así: El riesgo de utilización de maquina se estima razonable y tolerante en un 70%, es decir el tiempo mínimo de utilización de máquina para que el desarrollo del software avance es de 168 horas (De las 200 prevista), el análisis arrojó resultados de que solamente el 60% del tiempo estarán disponibles es decir se podrán utilizar sólo 120 horas.

Supervisión de Riesgos.- Del análisis anterior desprendemos que el presente proyecto funcionaría con un riesgo de 10 % este riesgo puede y debe ser reducido a cero por medio de un plan de supervisión de riesgos que en este caso estaría dado por la utilización de un número extra de horas máquina y horas _persona_mes, lo cual aumentaría en un 10% el coste del software, aquí es necesaria entonces la toma de decisiones gerenciales que varían dependiendo de la importancia del proyecto y consecuentemente la tolerancia al riesgo y el financiamiento del proyecto, es decir a mayor riesgo menor costo, y a mayor costo menor riesgo.

CAPITULO 2

PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos lleva ya varios años y ha asomado su cabeza en lenguajes de programación como Ada, SmallTalk, C++, Pascal de Borland (varias versiones) y por último en Delphi. Este término mágico, "objeto", ha adquirido tal fama, que la sola mención de "objeto", en especial si va en la misma oración que las palabras "ingeniero de software", hace que los analistas de sistemas tengan mucho que decir.

La complejidad del Software en los últimos años a aumentado considerablemente debido a esto se ha dado énfasis al estudio del ciclo de vida del software y a la ingeniería del software.

2.1. CICLO DE VIDA DEL SOFTWARE

El ciclo de vida del software es como un mapa. Una serie de pasos que deben seguirse en orden. Mediante este enfoque disciplinado, el desarrollador puede generar un producto mejor terminado, así como emplear menos tiempo en darle mantenimiento al producto. Esto no es la panacea, sino otra herramienta en la lucha contra el software deficiente.

El ciclo de vida del software se divide en seis áreas principales.

2.1.1 ANALISIS

Durante la fase de análisis se trata de definir lo mejor posible el problema por resolver. Se debe asegurar de que una solución basada en software no constituirá un problema, sino que ayudará a resolverlo.

En esta fase se debe determinar además que tipo de recursos se necesitarán para terminar el proyecto. Aquí se define el sistema como un todo y se señalan las funciones que éste debe realizar.

2.1.2 DISEÑO

Una vez establecidos los requerimientos del sistema o sea las funciones que va a realizar el software, se puede pasar a realizar un diseño detallado de la solución de software. Se debe planear y documentar como funcionará todo. Aquí se abordan aspectos como diseño de pantallas, colocación de botones, contenido de reportes y tipos de datos.

2.1.3 CODIFICACION

La codificación está presente a lo largo del resto del ciclo de desarrollo y continúa mientras se hacen cambios al código durante el resto del ciclo de vida del producto.

2.1.4 PRUEBA

El propósito real de ésta fase consiste en probar el producto, como se construyó, frente a los requisitos acordados en la especificación de desarrollo. Aquí es donde se sabrá si se hizo un buen trabajo al trasladar esos requisitos a la realidad.

2.1.5 MANTENIMIENTO Y OPERACION

Se deberá dar un seguimiento al producto final en lo que se refiere al servicio que éste presta a los clientes, a los puntos no documentados que ellos encuentren.

2.2. DISEÑO ORIENTADO A OBJETOS

Al observar el mundo real, se observa que el lenguaje tiene dos componentes principales: sustantivos (objetos) y verbos (operaciones). Para que las aplicaciones se apeguen a la realidad, el lenguaje de computación debe ser el mismo. La mayoría de los lenguajes tienen una gran cantidad de operaciones que se pueden realizar, pero tienen un conjunto reducido de sustantivos para describir los objetos. Incluso aquellos lenguajes que tienen capacidad para manejar objetos son, por lo regular, planos (no pueden heredar los atributos de sus padres). Esto, por supuesto, no es un reflejo del mundo real. En el mundo real las cosas son en tres dimensiones. Se necesita un lenguaje que pueda manejar esto. Delphi suministra un conjunto de sustantivos que permiten describir objetos.

De acuerdo con Grady Booch un objeto es "una entidad que tiene un estado; es decir, que tiene algún valor... el comportamiento de un objeto está definido por las acciones que sufre y viceversa... cada objeto es en realidad una instancia de cierta clase de objetos".

La meta del diseño orientado a objetos consiste en que cada módulo del sistema represente un objeto o una clase de objetos del mundo real. Grady Booch dijo: "Un programa que instrumenta un modelo de realidad pudiera entonces ser visto como un conjunto de objetos que interactúan entre sí". Se puede diseñar un sistema adoptando esta mentalidad orientada a objetos mediante los siguientes pasos:

1. Identificar los objetos y sus atributos.
2. Identificar las operaciones que afectan a cada objeto y las operaciones que cada objeto debe iniciar.
3. Establecer la visibilidad de cada objeto en relación con los otros objetos.
4. Establecer la interfaz para cada objeto.
5. Instrumentar cada objeto.

2.2.1 LOS OBJETOS

Al identificar los objetos en el espacio de un determinado problema, se piensa regularmente en términos de los sustantivos del problema. En un sistema de control de calefacción, se tendría una fuente de calor, temperatura, sensor, termostato, solenoide o algo parecido. Estos sustantivos se convierten en los objetos principales del sistema. Los objetos podrían ser muy grandes y constar de objetos pequeños. Por ejemplo, un automóvil es un objeto grande. Se lo puede dividir en objetos más pequeños como el motor, la transmisión y el chasis.

2.2.2 LAS OPERACIONES

En relación con este aspecto, es necesario identificar las operaciones que, cada uno de los objetos definidos anteriormente, realizan o las que se realizan en ellos. Por ejemplo, se puede ajustar un termostato, activar un solenoide o se puede leer la temperatura. También se podría definir que operaciones vendrían primero sobre un objeto. Un automóvil bien diseñado se pondría en marcha él mismo en cuanto se moviera la palanca a la posición de avance.

2.2.3 LA VISIBILIDAD

Aquí se define la topología del diseño. Se necesita un mapa que describa qué objetos son visibles y pueden ser vistos por otros objetos. En el sistema de control de temperatura, el sensor de temperatura necesita ser visto por el termostato.

2.2.4 LAS INTERFACES

Aquí se define cómo será la interfaz de ciertos objetos con otros objetos. Este paso es muy importante para diseñar un sistema verdaderamente modular. Se debe definir con exactitud cómo determinados objetos se comunican con otros objeto. Aquí se podría usar declaraciones de llamadas a funciones o procedimientos, para definir la interfaz para un objeto dado. Se puede hacer esto si, y sólo si, el lenguaje que se está empleando maneja un formato legible (como Delphi).

2.2.5 INSTRUMENTACION DE LOS OBJETOS

En este paso, se instrumenta cada objeto en la solución. Esto significa escribir las interfaces de código para cada objeto. Se podría optar por no escribir el código completo del objeto, dejando el cuerpo de los códigos para después. Si el objeto es complejo (un objeto conformado por otros más pequeños), se necesitará descomponer el objeto en aquellos que lo componen. Con cada uno de ellos deberá recorrer los mismos pasos para determinar sus operaciones, visibilidad e interfaz. Una vez creados los esqueletos funcionales con interfaces bien definidas, se puede hacer el cuerpo de la codificación en cualquier momento.

Siguiendo estos pasos, se puede diseñar un sistema coherente y bien pensado. Ya que Object Pascal tiene las facilidades para permitir codificar de ésta manera. Hay varios componentes en Object Pascal que hacen que esté orientado a objetos.

2.2.6 CLASES

Delphi suministra una palabra reservada *class* (clase) que permite, al desarrollador, definir un objeto. Cuando se crea un nuevo proyecto en Delphi, al observar las declaraciones en unit1, se encontrará una declaración de clase para la propia forma.

```
type
 TForm1 = class(TForm)
 private
  { Private declarations }
 public
```

```
{ Public declarations }
end;
```

Así es como se define un objeto. A partir de la sección de interfaz, se debe usar el nombre de tipo (TForm1) y luego la clase base de la cual se deriva. Todos los objetos deben derivarse de TObject o de alguno de sus objetos hijos.

La sección public (pública) se reserva para aquellas declaraciones a las que se desea que todo el mundo tenga acceso. Hay una sección private (privada) en la que se puede declarar variables, procedimientos y funciones que se utilizan sólo dentro de una clase. La sección protected (protegida) ofrece lo mejor de las secciones pública y privada en una sola. Los componentes declarados como protegidos son sólo accesibles a los descendientes del tipo que declara.

Al igual que con los componentes privados, se pueden ocultar detalles de instrumentación a los usuarios finales. Sin embargo, a diferencia de los componentes privados, los protegidos aún están disponibles a los programadores que deseen derivar nuevos objetos de sus objetos sin el requisito de que los objetos derivados se declaren en la misma unidad.

2.2.7 PROPIEDADES

La palabra reservada property permite declarar propiedades. Una definición de propiedad en una clase declara un atributo nombrado para objetos de la clase y las acciones asociadas con la lectura y escritura del atributo.

2.2.8 HERENCIA

La herencia es una de las características más poderosas de un lenguaje orientado a objetos como Delphi. Este permite que clases hijas tomen las propiedades de sus padres. Heredan los campos, propiedades, métodos y eventos de su clase padre. Además de tener los atributos de sus padres, la clase hija puede agregar nuevos componentes a los que hereda. Esto le permite tomar una clase que tenga casi todas las piezas fundamentales que se necesitan y agregar nuevos objetos para adaptar esa clase exactamente a sus necesidades.

Incluso si se compila un nuevo proyecto, se podría ir a View | Browser y explorar la base objeto completa en Delphi.

CAPITULO 3

GENERALIDADES DE DELPHI

3.1. EL PROGRAMA DELPHI

Delphi es un lenguaje de programación visual. Esto significa que se diseñan los programas por medios visuales. Como se conoce, los programas Windows contienen objetos gráficos (botones, barras de desplazamiento, cajas de edición, etc.). Delphi es usado para la creación de programas Windows, así se pueden crear ventanas con botones, barras de desplazamiento y otros objetos. Los objetos son puestos usando el Mouse.

La programación visual el muy fácil, y Delphi está diseñado para que así lo sea.

La programación visual de Delphi es sólo una parte del trabajo. Una vez que los objetos son puestos en las ventanas del programa se debe escribir el código respectivo para cada objeto. Por ejemplo, si durante la fase de programación visual se pone un botón en la ventana de la aplicación. Más tarde cuando se ejecute el programa y se haga clic en el botón un cierto código se ejecutará (el que haya sido escrito en tiempo de diseño).

El código escrito en Delphi es escrito con el lenguaje de programación Object Pascal.

Se puede considerar a Delphi como un ambiente de programación diseñado para crear aplicaciones Windows que usan bases de datos relacionales.

Debido al hecho de que Delphi tuvo la fortuna de haber nacido tarde en el mundo de los desarrolladores visuales éste se ha beneficiado de un gran número de herramientas y tecnologías pioneras tales como:

- Cliente Servidor
- Programación Orientada a Objetos (OOP)
- Tecnología RAD

El término RAD corresponde a las siglas de Rapid Aplication Development (Desarrollo Rápido de Aplicaciones) y fue acuñado para una nueva casta de ambientes de desarrollo de software. En este nuevo mundo RAD, los programadores emplean herramientas que son más intuitivas y visuales. Resulta difícil observar un fragmento de código que genera una ventana y visualizar la propia ventana, en comparación con la facilidad de crear la ventana real con un par de clics del ratón.

En éste mundo nuevo de interfaces más visuales y sencillas, el primer jugador en aparecer en el ambiente de Windows fue Visual Basic (el cual será referido en lo sucesivo como VB). VB hizo descender la programación desde la complejidad de los códigos a la construcción visual de la interfaz de usuario con el ratón. Aunque VB tuvo éxito en el mercado, el lenguaje en sí mismo no promovió en realidad un buen diseño. VB en sus inicios carecía de los mecanismos para generar un código altamente estructurado, compacto y bien refinado. Carecía del rigor de un lenguaje orientado a objetos. Delphi es el siguiente paso en los ambientes de desarrollo RAD. Corrige la mayoría de deficiencias encontradas en VB, sin necesidad de agregar otras nuevas.

Aún cuando Delphi presenta una similitud Visual con VB, la principal diferencia entre los dos productos consiste en el lenguaje que existe detrás del Ambiente de Desarrollo Integrado (IDE). Delphi utiliza Object Pascal como su lenguaje base. El compilador Pascal de Borland (a partir del Turbo Pascal 1.0) ha sido uno de los más rápidos en el medio. Borland incorporó al lenguaje extensiones basadas en objetos para dar apoyo a las buenas prácticas de programación y un código más eficiente. Object Pascal es un verdadero lenguaje basado en objetos con un compilador sólido tras él.

3.2 LAS VENTAJAS DE DELPHI

Son muchos los aspectos que han hecho sobresalir a Delphi por encima de sus competidores. Para aquellos que no están familiarizados con VB, un archivo .exe que es generado por VB, no es en verdad ejecutable por sí mismo. Ese archivo .exe es en realidad un código PostCompilado. Esto significa que el archivo ejecutable es realmente como una gran macro o un archivo script. A fin de ejecutar esa gran macro, el archivo .exe debe estar acompañado de un programa denominado VBRUNxxx.DLL. Esto representa una seria molestia; si, por alguna razón, no aparece el archivo VBRUNxxx.DLL o si los usuarios lo borran porque no creen que sea necesario, la aplicación no se ejecutará.

Cuando Delphi genera un archivo .exe, éste es realmente ejecutable. No se requiere de archivos adicionales para que funcione. Esto proporciona aplicaciones más limpias, rápidas y fáciles de distribuir y de mantener.

Delphi proporciona un compilador optimizado que le da una rápida aplicación, sin que para optimizar el programa tenga el programador que hacer esfuerzo adicional al que haya hecho para escribirlo originalmente.

3.3 DIFERENCIAS ENTRE DELPHI Y DELPHI 2

Aunque Delphi 2 IDE pudiera parecerse en mucho a Delphi, bajo la cubierta existen grandes diferencias. Delphi 2 es capaz de manejar controles personalizados OLE (OCX). En el mundo de 16 bits de Windows 3.1, Microsoft creó los VBX (Visual Basic eXtensions). Estos eran controles visuales que representaban los bloques de construcción del ambiente RAD de VB. Estos VBX se incorporaron al ambiente de desarrollo integrado, donde fueron colocados en una caja de herramientas que pudiera usar el creador. Estos controles cumplían un propósito importante; eran objetos tales como botones y cuadros de texto que proporcionaban una retroalimentación visual durante el proceso de diseño. Esto es lo que hizo lo "visual" en Visual Basic. Los VBX fueron en realidad diseñados como un producto adicional.

La siguiente generación de los VBX son los OCX, que vienen a ser como sus hermanos mayores. El VBX fue un control de 16 bits. El OCX es un control personalizado, bien planeado, de 32 bits con base en OLE. Aun cuando Delphi 2 permite agregar un control OCX a la paleta de herramientas (tal como Delphi 1 permitía la adición de VBX), esos son prácticamente todos los atributos que comparten.

El control OCX se diseñó para trabajar en el mundo de 32 bits de Windows 95 y Windows NT. Le permite interactuar en la misma forma que lo hacia VBX. En ese respecto su comportamiento es similar.

El control OCX no es la única forma de utilizar OLE en Delphi 2. Delphi 2 también le permite crear automatización OLE de clientes y servidores. Estos objetos de automatización le dan la facilidad de crear programas que realizan tareas en el trasfondo y pasan los resultados a su aplicación. Estos pueden ser los motores de trabajo de su aplicación y se pueden compartir con varias aplicaciones que se ejecuten a un tiempo. El control de contenedor OLE también ofrece la capacidad para colocar un objeto OLE en una aplicación. Un ejemplo sería incluir una hoja de cálculo de Excel en la forma principal. Esto le da el poder de utilizar las capacidades de una aplicación en lugar de tener que reinventarlas cada vez que se escribe una aplicación.

Otra gran ventaja de Delphi 2 es la capacidad para escribir aplicaciones multitareas. En el mundo de multitareas de Windows 95 y NT, ésta es una característica necesaria. Al hacerse más complejas las aplicaciones, se hace necesario ejecutarlas en partes por separado, cada una de las cuales realizan funciones específicas. Estas partes se denominan hilos de ejecución, Delphi 2 maneja la creación, uso y control de hilos.

También en el área de bases de datos Delphi 2 tiene una ventaja significativa en el desempeño sobre Delphi 1.0. Con manejadores de 32 bits asi como un ejecutable de 32 bits, la velocidad de ejecución es significativamente menor en Delphi 2. Esto ayuda a Delphi 2 a superar a los constructores de proceso frontal como PowerBuilder.

3.4 CLIENTE/SERVIDOR DELPHI

Hay tres versiones diferentes de Delphi: Delphi Desktop, Delphi Developer y Delphi Cliente/Servidor (al que se llamará en lo sucesivo CS). Estas versiones de Delphi ofrecen diferentes niveles de conectividad con el mundo exterior. Delphi Desktop viene con la capacidad de conectarse con dBase y Páradox a través del dispositivo de base de datos de Borland. La edición Developer contiene conectividad ODBC (para conectar cualquier fuente de datos con un manejador ODBC), y la edición CS viene con vínculos SQL (SQL Links). Este producto ofrece manejadores de alta velocidad, de 32 bits, para conectarse con servidores SQL de bases de datos como SyBase y Oracle.

La edición CS se diseñó para competir con otros productos para el desarrollo de aplicaciones cliente/servidor. El principal competidor de Delphi CS es PowerBuilder. Por lo regular, estos productos se utilizan en ambientes corporativos para crear procesos frontales con base en Windows, para bases de datos basadas en redes.

Otra excelente característica de Delphi CS es el generador visual de consultas (Visual Query Builder). Si alguna vez se ha utilizado Query o Access de Microsoft, se sabe de la alegría de construir una consulta de datos mediante arrastrar y soltar campos en una ventana y jugar a conectar los puntos. Esto es una gran ayuda para los legos en bases de datos.

3.5 VCL SUSTITUYE A VBX

Delphi posee su propia versión nativa de un VBX/OCX, denominada componente visual [Visual Component (VC)]. Se escribe un componente visual en Delphi y se agrega a la Biblioteca de Componentes Visuales VCL (Visual Component Library). La VCL es el depósito de todos los componentes visuales que puede utilizar el creador de software para crear aplicaciones Delphi. Todos los componentes en la VCL se exhiben en la barra de herramientas para que sean de fácil acceso para el usuario. Lo mejor de los componentes visuales consiste en que se pueden construir componentes propios y después incorporarlos a la barra de herramientas. Esto promueve, además, la reutilización del código, un componente vital en la programación de alta productividad.

3.6 ALGO MAS SOBRE DELPHI

Hace algún tiempo escoger herramientas para desarrollar aplicaciones de bases de datos para aplicaciones Windows fue muy simple, y muy pobre; se podía construir aplicaciones usando una herramienta script que interpretaba bases de datos o usando un lenguaje de programación de bajo nivel con una librería de rutinas de acceso a bases de datos. Las herramientas visuales permitían obtener rápidamente prototipos de las aplicaciones pero el lenguaje de interpretación de scripts era bajo en poder, desempeño y fiabilidad para la distribución. En febrero de 1995, Borland lanza Delphi 1.0, un nuevo producto diseñado desde abajo que combinaba herramientas de diseño de programas visuales orientado a objetos, un alto desempeño de un compilador optimizado y un poderoso componente de base de datos Cliente/Servidor. La arquitectura orientada a objetos de Delphi promueve el código reutilizable entre aplicaciones reduciendo tiempo y esfuerzo de desarrollo. El lenguaje Object

Pascal de Delphi es limpio, legible (mantenible) y poderoso, con características tales como manejo de excepciones estructurado para mayor estabilidad y fiabilidad. El Borland Database Engine es el corazón de la arquitectura de base de datos de Delphi que tiene un soporte de base de datos independiente de los vendedores, soporta SQL con manejadores ODBC.

Cada dos o tres años, parece que alguien tiene una nueva "bala de plata" una singular herramienta o tecnología que le ayudará en el desarrollo de software en tiempo récord. A mediados de los 80 las herramientas CASE fueron la nueva panacea. Más adelante en los 80 la OOP fue la panacea para curar todos los males. Ultimamente, esta siendo Cliente/Servidor. ¿Que es lo correcto?.

La opinión general en todo esto es que no hay bala de plata, no hay una sola herramienta que pueda hacerlo todo. La idea es que si una herramienta produce aplicaciones rápidamente, es probable que genere ejecutables insuficientes. Si ésta permite construir fácilmente formas usando un paradigma visual, ésta probablemente requiera entrar en código de bajo nivel. Una herramienta que corre en cliente servidor, probablemente no es totalmente compatible con Paradox. Así es como van las cosas.

No se trata de decir que Delphi es la bala de plata de la programación que el mundo ha estado esperando. Sin embargo éste representa un colosal avance en la dirección en la cual las herramientas visuales han ido desde que llegaron a escena. Delphi es la primera herramienta RAD con un compilador optimizado. Para esto añade una tecnología de base de datos escalable, obteniendo una potente mezcla de tecnología de programación nunca antes vista en las computadoras personales. Puede que no sea la bala de plata pero está muy cerca.

Con todas estas ventajas, Delphi no produce un buen software sin un apropiado diseño. Se debe hacer un estudio para saber que es lo que los usuarios necesitan, Delphi no piensa por sí solo, es sólo una herramienta.

El punto principal es aprender a contarle a Delphi lo que se necesita, aprender a manejar Delphi para desarrollar aplicaciones robustas.

¿Porque se querría usar Delphi en lugar de otras herramientas como PowerBuilder o Visual Basic?

Como se ha dicho, Delphi integra satisfactoriamente el desarrollo de aplicaciones visuales con un compilador optimizado. Lo cual no es verdadero para otras herramientas, incluyendo PowerBuilder y VB. El compilador de Delphi no es nada flojo ya que es el heredero de una larga línea de compiladores Pascal de la Borland. Estos compiladores se han ganado una gran reputación al producir ejecutables que usan pocos recursos y tienen un alto desempeño. El compilador Object Pascal de Delphi no es la excepción.

Un compilador optimizado no es suficiente, sin embargo. Los diseñadores de éstos días quieren una plataforma de desarrollo de aplicaciones que sea bastante extensa para cumplir con sus necesidades y lo suficientemente ágil para permitirle resolver cualquier problema de programación. Ellos quieren un marco de trabajo con objetos, herramientas que le permitan codificar en lenguaje ensamblador, si el caso lo requiere. Necesitan normalmente generar archivos .EXE, una herramienta que produzca .DLLs y manejadores de dispositivos, si es

necesario. Ellos requieren rápidos desarrollados de aplicaciones de bases de datos pero que no les obligue a involucrarse en la ingeniería de las bases de datos.

Delphi es todas estas cosas y más. Borland tomó todos los mejores elementos encontrados en las modernas herramientas de desarrollo Windows y las tejió en un solo producto.

Hay un número de buenas razones para escoger Delphi sobre otros ambientes de desarrollo. Delphi ofrece éstas características:

- Un sofisticado marco de trabajo de objetos.
- Un compilador de código nativo rápido (Característica de Pascal)
- Depurador integrado
- Acceso abstracto a la base de datos.
- Herramientas sofisticadas TWO-WAY (que quiere decir que se pueden modificar los objetos visualmente o textualmente).

Delphi además añade otras herramientas:

- Acceso directo al API de Windows
- Soporte en línea para el lenguaje ensamblador
- Generación de componentes personales VCL y OCX
- Creación de DLLs y otros tipos de objetos auxiliares de Windows
- Completa implementación OOP.

Muchas herramientas de desarrollo fingen la orientación a objetos, no son verdaderas herramientas OOP. Una herramienta realmente OOP debe cumplir con las siguientes características:

- **Herencia:** Nuevos objetos pueden ser creados heredando sus atributos y métodos de otros objetos ya existentes.
- **Polimorfismo:** Es la característica que permite tener operaciones con el mismo nombre asociadas a diferentes objetos pero actuando en forma diferente. Esto es, el método Show, por ejemplo, ejecuta radicalmente tareas diferentes en un botón y en un Grid pero la llamada es la misma.
- **Encapsulación:** Los datos y el código del programa pueden ser localizados en una sola entidad. Esto es, un objeto puede almacenar tanto los elementos de datos como los procedimientos (métodos). Los elementos procedurales dentro de un objeto deben tener acceso automático a los elementos de datos dentro del objeto.
- **Metodología Primaria:** La orientación a objetos debe ser la metodología primaria para construir el código del programa, no algo a ser añadido después. De esta forma puede ser usado productivamente.

En Orientación a Objetos podemos distinguir tres tipos de herramientas de desarrollo:

1. **Herramientas Orientadas a Objetos Propiamente dichas:** Cumplen todas las características de la orientación a objetos. Los principales representantes de éstos son:
    - Pascal para Objetos (Delphi)
    - C++

- SmallTalk

2. **Lenguajes basados en Objetos:** Poseen Objetos pero no tienen clases ni herencia. Los representantes de éstos son:
    - Visual FoxPro
    - Clipper
    - Páradox
3. **Lenguajes basados en Clases:** Poseen clases pero no poseen herencia entre clases, permite la creación de objetos dentro de las clases. Ejemplo: Visual Basic.

Muchas herramientas reclaman escalabilidad. La escalabilidad de Delphi provee las siguientes características:

- Soporte para tablas locales y remotas
- Soporte heterogéneo para consultas en múltiples aplicaciones DBMS
- Provee una plataforma independiente para acceso a bases de datos.
- Viene con manejadores nativos para la mayoría de DBMS's Cliente/Servidor
- Provee un completo soporte ODBC (Conexión abierta a bases de datos).

En una palabra, Delphi es la herramienta consumada para desarrollo de Bases de Datos.

3.7 COMPARACION CON OTRAS HERRAMIENTAS

C++

- La rica sintaxis del Object Pascal rivaliza con la sintaxis de C
- El lenguaje Pascal se ha ganado el favoritismo por años
- Concisas librerías de Pascal en relación con C. No hay 15 caminos para ejecutar una función, normalmente sólo hay uno. Según Ken Henderson el hecho de que un comité domine los estándares de C ha hecho que éste caiga en la burocracia
- C es un lenguaje confuso, Pascal es transparente y comprensible.

VISUAL BASIC Y ACCESS

Debido a que los módulos de Access se programan con VB, vamos a concentrarnos sólo en éste último:

- Una de las más grandes diferencias entre Visual Basic y Delphi es que Delphi genera ejecutables en código nativo, es un compilador verdadero, o sea que sus ejecutables están en lenguaje máquina. No así Visual Basic quien genera archivos ejecutables semiinterpretados. Los ejecutables de Delphi están compuestos de lenguaje máquina no de seudocódigo que debe ser interpretado en tiempo de corrida, en este último caso no hay una toquenización automática del código que se escribe. Esto hace que los ejecutables de Delphi sean rápidos y robustos
- Se ha llamado a Delphi como el "VB Killer", pero una definición más elegante sería definir a Delphi como: "Visual Basic: La nueva Generación".

DIALECTOS XBASE: DBASE, CLIPPER, FOXPRO Y OTROS

- FoxPro no soporta totalmente la OOP
- El código generado por los Xbase no es código de máquina, por lo tanto son menos rápidos y menos vigorosos.

3.8 EL AMBIENTE DE DESARROLLO INTEGRADO (IDE)

Se ha presentado un breve panorama del porqué se ha elegido Delphi. Ahora pasamos al estudio del ambiente Delphi. El ambiente Delphi fue diseñado de una manera organizada y muy funcional, es un ambiente en el que están presentes todas las herramientas necesarias para diseñar, ejecutar o probar una aplicación, además que se encuentran bien relacionadas para facilitar el desarrollo de programas. En Delphi 2, el IDE consta de un editor de código, un depurador de errores, una barra de herramientas, un editor de imágenes y las herramientas de base de datos, todos ellos operando de manera integrada. La integración ofrece al desarrollador un conjunto de herramientas que operan en armonía y se complementan entre sí. El resultado es un desarrollo de aplicaciones complejas más rápido y libre de errores.

CAPITULO 4

COMPONENTES BASICOS DE DELPHI

4.1. EL ESCRITORIO DE DELPHI

El primer paso consiste en iniciar Delphi. Una vez cargado éste su escritorio deberá verse como en la figura 3.1.

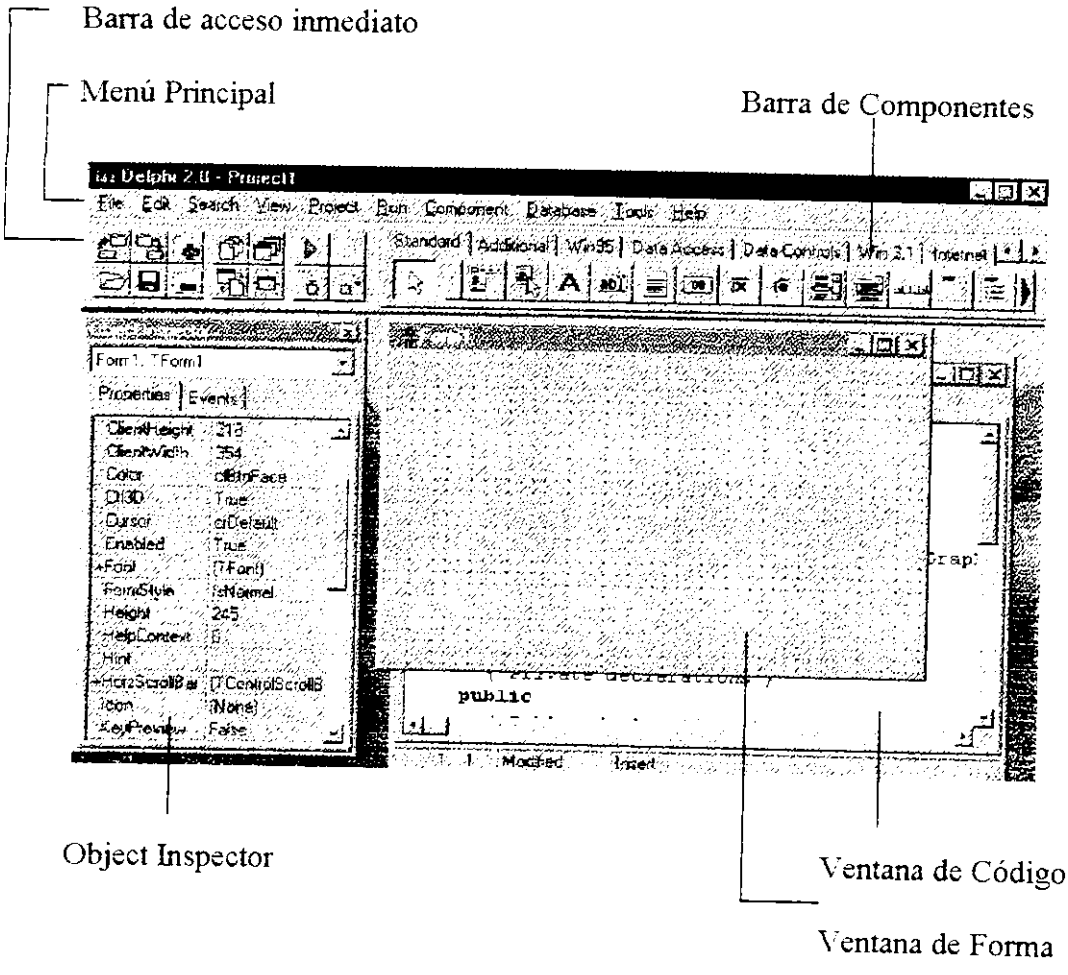


Figura 3.1. Escritorio de Delphi

LA BARRA DE ACCESO DIRECTO

La barra de acceso inmediato (*Speed Bar*) que se muestra en la Figura 3.2 fue diseñada para ayudar a obtener con facilidad y rapidez las funciones que más se utilizan. La configuración por omisión ofrece 14 de las funciones que Borland considera de uso más frecuente. Estas funciones también se hallan disponibles en el menú de Delphi, y sólo se las incluye en la barra de acceso inmediato para acelerar su acceso a ellas.

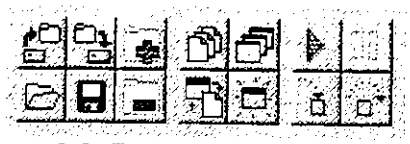


Figura 3.2. Barra de acceso inmediato

LA PALETA DE COMPONENTES

La paleta de componentes representa el "inventario visual" de la biblioteca de componentes visuales (VCL por las siglas de Visual Component Library). Esta paleta le permite conformar categorías de componentes visuales en grupos homogéneos. Por omisión, los componentes se agrupan sobre líneas funcionales, esto es, todos los componentes de acceso de datos aparecen juntos y así sucesivamente. Estos grupos o páginas son denotados mediante pestañas etiquetadas. Las páginas por omisión son:

Standard (Estándar)

Additional (Adicional)

Win95 (Windows 95)

Data Access (Acceso de datos)

Data Controls (Controles de datos)

Win 3.1 (Windows 3.1)

Dialogs (Diálogos)

System (Sistema)

Qreport (Reportes rápidos)

OCX (Controles personalizados OLE)

Samples (Ejemplos)

LA FORMA

Prácticamente, la base de toda aplicación Delphi es la forma. Es posible que se conozca a esta como una ventana: el tipo de ventana que se puede observar en Word, Paradox u otras aplicaciones basadas en Windows. En Delphi, Form representa los cimientos sobre los que se colocarán otros componentes de Delphi. Es el respaldo de una aplicación Windows. En la figura 3.3 se muestra una típica forma Delphi en blanco.

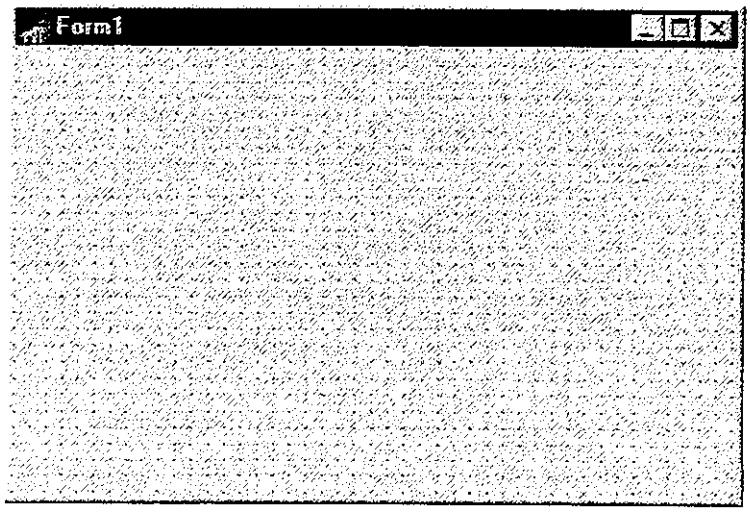


Figura 3.3. Una forma de Delphi

Una forma Delphi tiene las mismas propiedades que se encontrarán en cualquier otra ventana de Windows 95. Tiene un menú de control (en la esquina superior izquierda de la forma). la barra de título a lo largo de la parte superior y los botones para maximizar, minimizar y cerrar (o eliminar) en la esquina superior derecha de la forma. En caso necesario, se puede ocultar o limitar el uso de éstos atributos.

LA VENTANA EDIT

Una de las piezas esenciales del ambiente Delphi es la ventana Edit. Esta ventana provee el mecanismo para que el desarrollador escriba su código Delphi. El editor de código de Delphi es un gran editor con todas sus características (como se ve en la figura 3.4). Incluye señalamiento de sintaxis, a color (lo que le ayuda a localizar más rápido esos errores en el código), comandos de edición de tipo Brief (Brief es el editor de programación para los puristas) y le da la capacidad de "deshacer" a diferentes niveles.

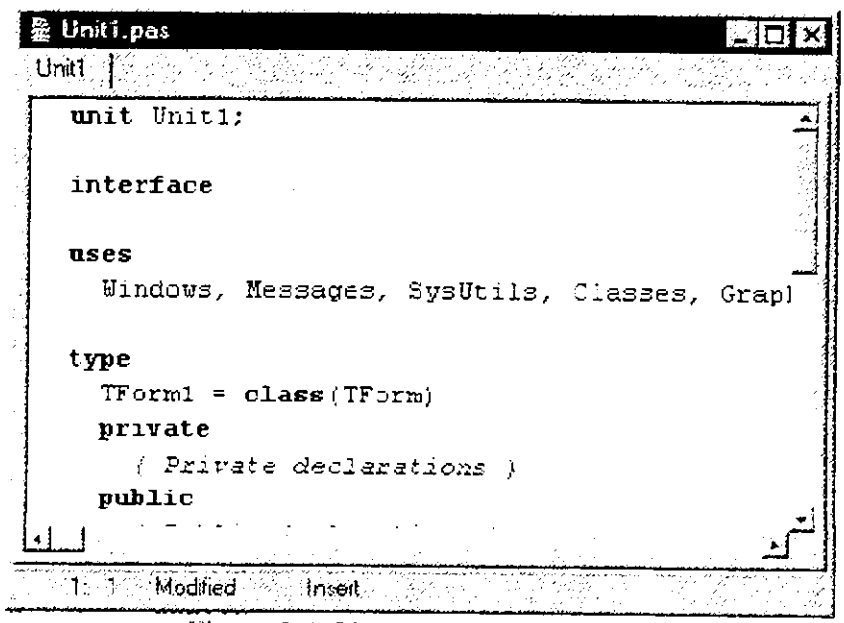


Figura 3.4. Ventana de edición de código

Nótese que la barra de título presenta el nombre del archivo que se visualiza en ese momento. A lo largo de la parte superior de la ventana se observará las pestañas que indican las páginas disponibles actualmente. Puede haber varios archivos fuente para aplicaciones Delphi, y las pestañas le ayudan a navegar a través de ellos. A lo largo de la parte inferior de la ventana Edit hay otros tres elementos de interés. En la posición más hacia la izquierda, está el indicador de línea/columna. Este le sirve para ubicar en que parte del código se encuentra. El segundo elemento es el indicador de modificaciones. Al iniciar un nuevo proyecto, el código que Delphi le presenta no está almacenado. Debe ser guardado. Debido a que éste código a cambiado desde la última vez que se guardó en disco (o sea, hasta ahora nunca), aparece la palabra "modified" (modificado) junto al indicador de línea/columna. Este la muestra en todo momento si el código que se ve no es lo que está guardado en disco. El último elemento es el indicador de inserción y reescritura (insert/overwrite), una característica estándar en la mayoría de los editores, que muestra si se está insertando texto o escribiendo sobre uno ya existente.

EL OBJECT INSPECTOR

El Object Inspector es un elemento esencial. Ofrece una interfaz fácil de usar para cambiar las propiedades de los elementos de Delphi, así como para controlar los eventos ante los que un objeto reacciona.

LA FICHA DE PROPIEDADES

La parte correspondiente a propiedades del Object Inspector (Figura 3.5) le permite ver y modificar las propiedades de un objeto. Al hacer clic en la ventana vacía de la forma se puede observar sus atributos en la ficha properties (propiedades) del Object Inspector. Un punto de interés: cuando se vea una propiedad con un signo de adición (+) a la izquierda, significa que esa propiedad tiene anidadas en sí otras subpropiedades.

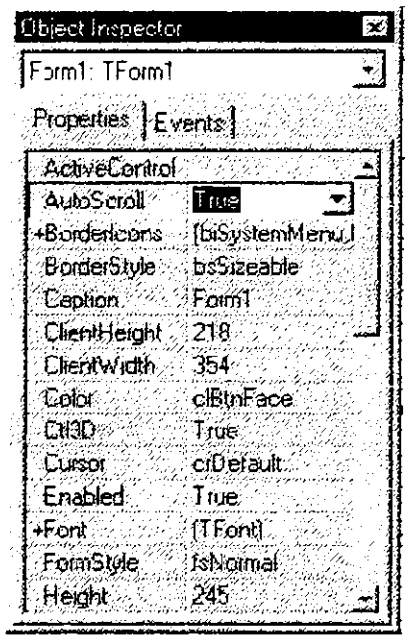


Figura 3.5. La Ficha de Propiedades

Por ejemplo, después de seleccionar la forma, el Object Inspector tiene una propiedad Font (Fuente) con un signo de adición al lado izquierdo. Si se hace un doble clic en ésta propiedad, se mostrarán más propiedades como Color, Heigth, Name (color, altura, nombre) y otras. Este formato es un medio limpio, sencillo y eficiente para modificar los atributos de un objeto.

FICHA DE EVENTOS

La ficha Events (eventos) es la otra mitad en la vida del Object Inspector (Figura 3.6). Se relaciona con el programador y los diferentes eventos a los que puede responder este objeto. Por ejemplo, si se necesita que una aplicación haga algo especial cuando se cierra la ventana, se puede emplear el evento OnClose (al cierre) de la forma, para hacerlo.

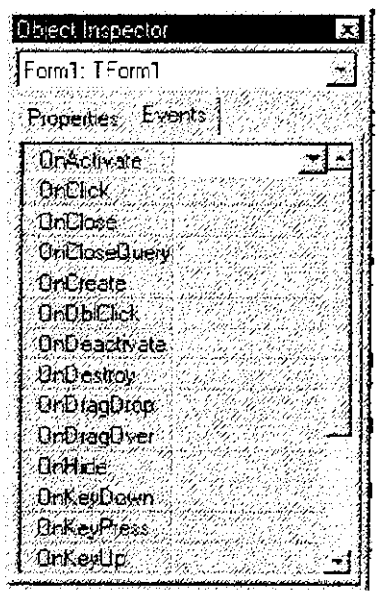


Figura 3.6. La Ficha de Eventos

4.2. LAS BASES DE DATOS EN DELPHI

Delphi tiene integrado un poderoso manejador de bases de datos. Hay componentes visuales que permiten accesar las tablas y métodos para manipular los registros.

Una de las principales herramientas que viene incluida con Delphi es el DATABASE DESKTOP, el cual permite crear una base de datos para ser usada en una aplicación.

EL DATABASE DESKTOP

El programa Database Desktop permite crear bases de datos y ejecutar otras operaciones relativas a bases de datos (Figura 3.7).

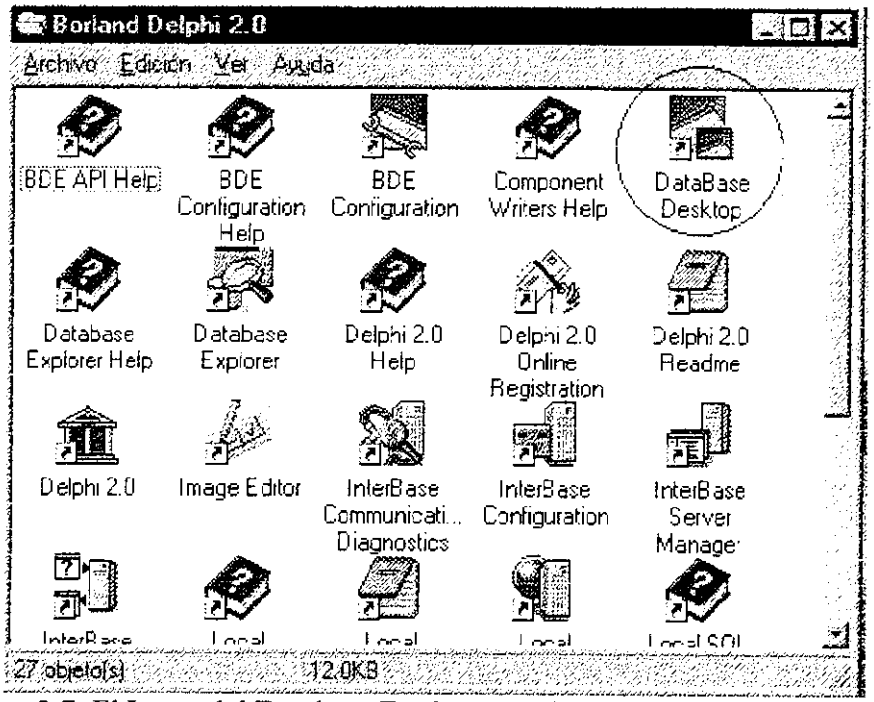


Figura 3.7. El Icono del Database Desktop en el grupo de programas de Delphi

Al hacer doble clic en el icono Database Desktop del grupo de programas de Delphi se presenta la ventana de la figura 3.8.

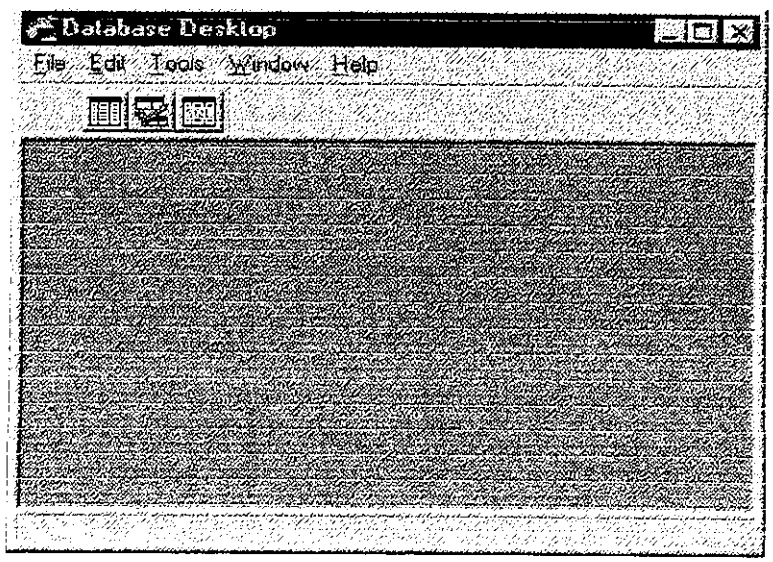


Figura 3.8. El Database Desktop

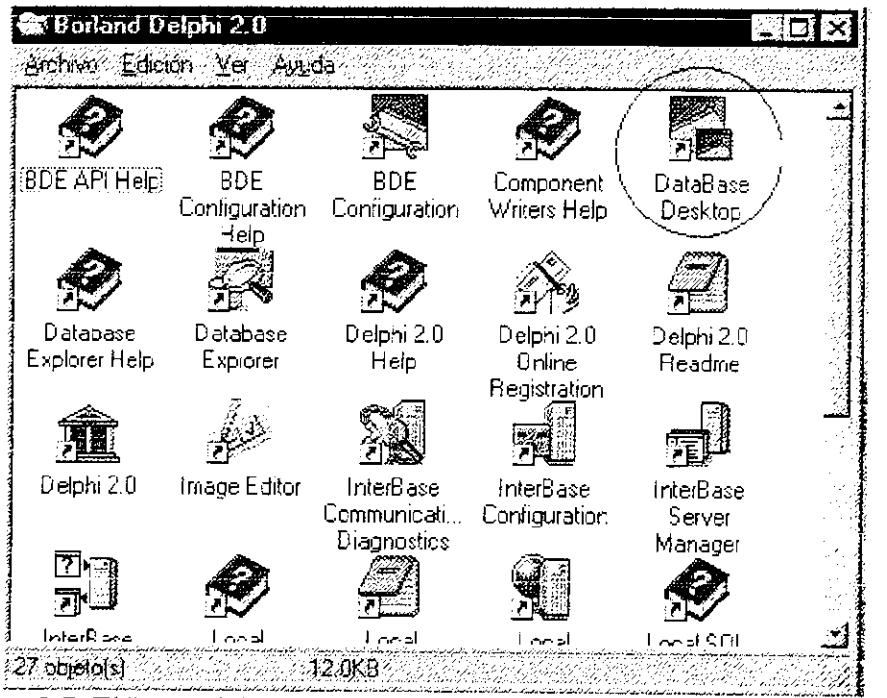


Figura 3.7. El Icono del Database Desktop en el grupo de programas de Delphi

Al hacer doble clic en el icono Database Desktop del grupo de programas de Delphi se presenta la ventana de la figura 3.8.

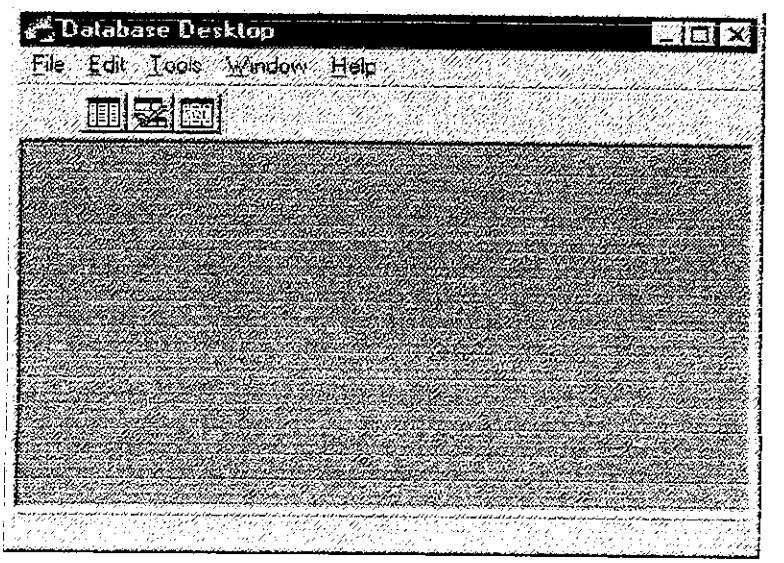


Figura 3.8. El Database Desktop

- En el Database Desktop se debe seleccionar *File* | *Working Directory*. El programa responde desplegando la caja de diálogo *Set Working Directory* como se muestra en la figura 3.9.

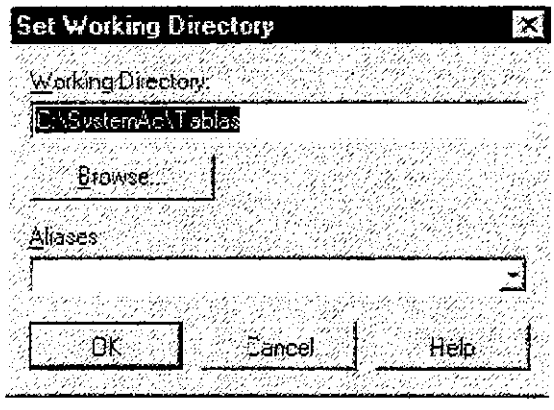


Figura 3.9. Forma de diálogo para la selección del directorio de trabajo

- Hacer clic en el botón *Browse* para escoger el directorio en el cual se almacenarán las tablas de la base de datos. El Database Desktop responde desplegando la caja de diálogo *Directory Browser* como se muestra en la figura 3.10.

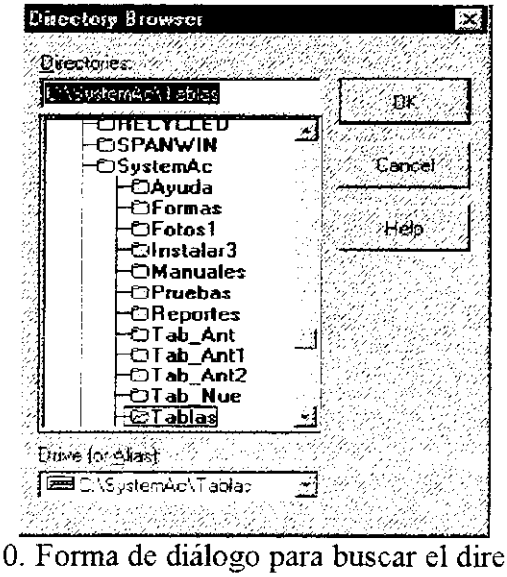


Figura 3.10. Forma de diálogo para buscar el directorio de trabajo

- Se debe escoger el directorio que se haya elegido como directorio de trabajo y hacer clic en el botón OK. El Database Desktop cierra la caja de diálogo *Directory Browser* y despliega la caja de diálogo *Set Working Directory*.
- Al hacer clic en el botón OK de la caja de diálogo *Set Working Directory*. El directorio de trabajo ha quedado establecido.

Establecimiento de un Alias para el directorio de trabajo

Un Alias no es más que un sobrenombre o nombre simplificado para un directorio de trabajo. Crear un alias cumple con dos objetivos principales:

1. Evita teclear continuamente el nombre del directorio de trabajo, ya que cuando se necesita conectarse a una base de datos se lo hace por medio del alias.

2. Si el programa se comunica con la base de datos por medio de un alias es muy fácil redefinir un nuevo directorio de trabajo para la base de datos, sin tener que modificar la aplicación.

Aquí está como asignar un Alias a su directorio de trabajo:

- Seleccionar *Database* ¦ *Explore* desde el menú principal de Delphi. Delphi responde presentando la ventana *Database Explorer* (Figura 3.11).

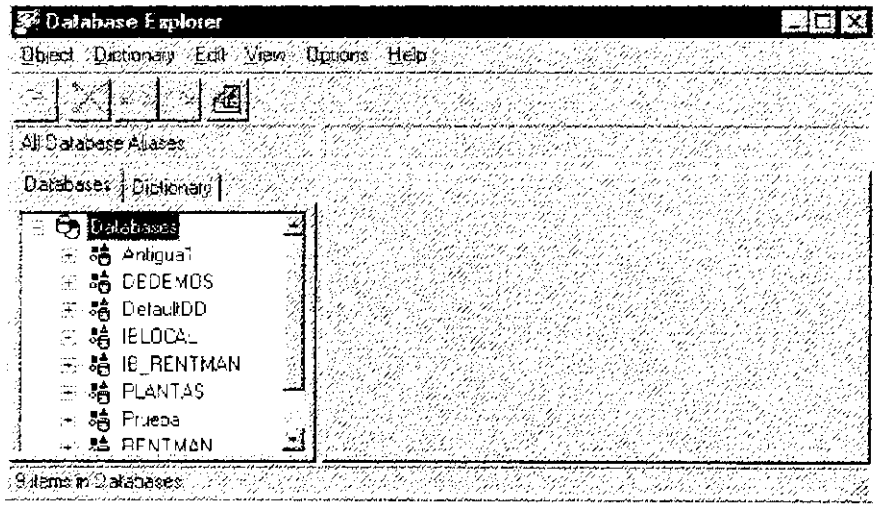


Figura 3.11. El Database Explorer

- Al escoger *New* desde el menú *Object* se presenta la caja de diálogo *New Database Alias* que permite escoger el nombre del manejador de base de datos (Figura 3.12). En este caso se podría seleccionar el valor por defecto.

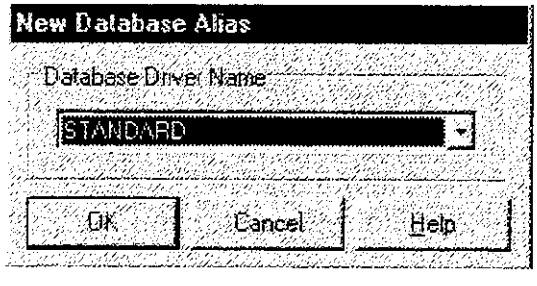


Figura 3.12. Forma de diálogo para seleccionar el manejador de base de datos.

- Delphi presenta la pantalla de la Figura 3.13, en donde se puede ingresar el nombre para el nuevo alias, el directorio de trabajo o PATH de la base de datos y el manejador por defecto, es éste caso Paradox.
- Una vez ingresados los datos se pude salvarlos con la opción *Apply* del menú *Object*.

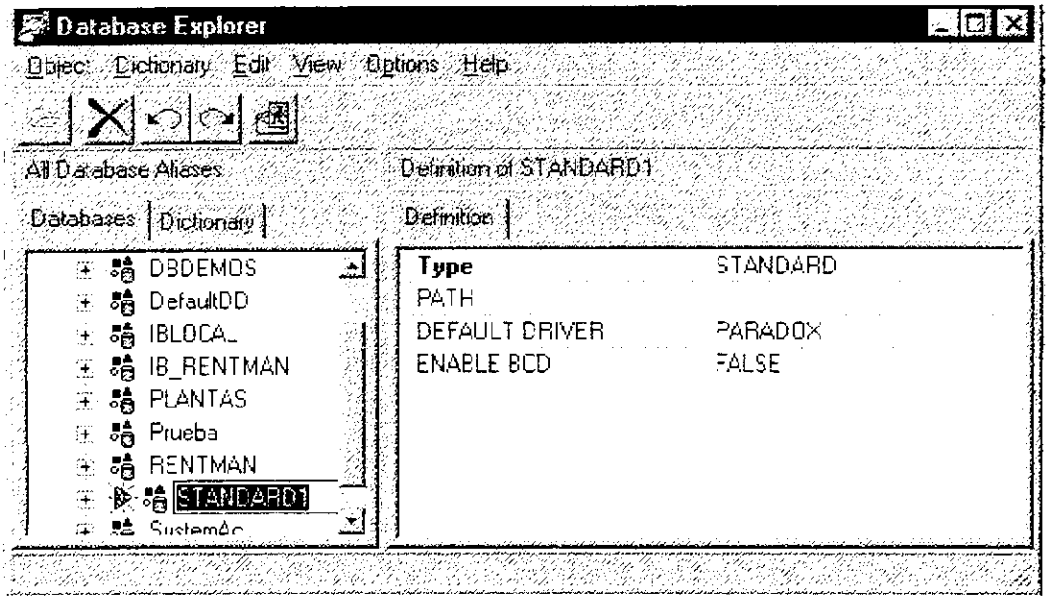


Figura 3.13. Creación de un Alias en el Database Explorer

Creación de tablas con el Database Desktop

- Al seleccionar *File | New | Table* en el Database Desktop, el Database Desktop responde con la caja de diálogo de la figura 3.14.

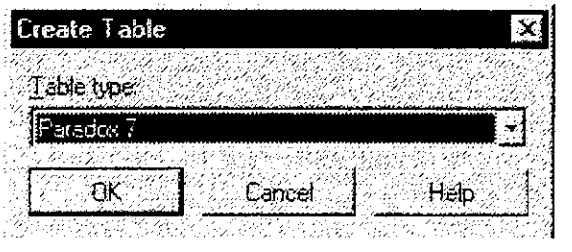


Figura 3.14. Forma de diálogo para escoger el tipo de tabla a crear

- Si se hace clic en el menú desplegable se puede ver que existen varios tipos de tablas para escoger.

- Ahora se puede hacer clic en el botón OK para escoger la tabla por defecto. El Database Desktop responde con la caja de diálogo de la figura 3.15. Allí se puede ingresar los diferentes campos que componen la tabla con su respectivo tipo y tamaño, además se puede establecer reglas de validación de datos, integridad referencial, claves, etc.

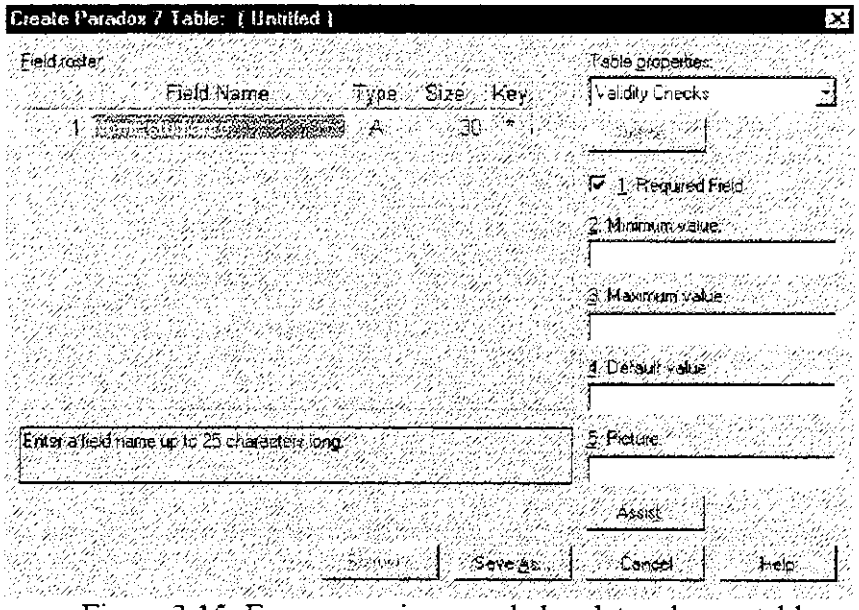


Figura 3.15. Forma para ingreso de los datos de una tabla

- Luego de ingresar toda la estructura de la tabla se puede hacer clic en el botón *Save As* para salvar la tabla. Luego, se debe ingresar el nombre de la tabla y hacer clic en el botón *Guardar* para guardarla en el directorio de trabajo previamente establecido (Figura 3.16).

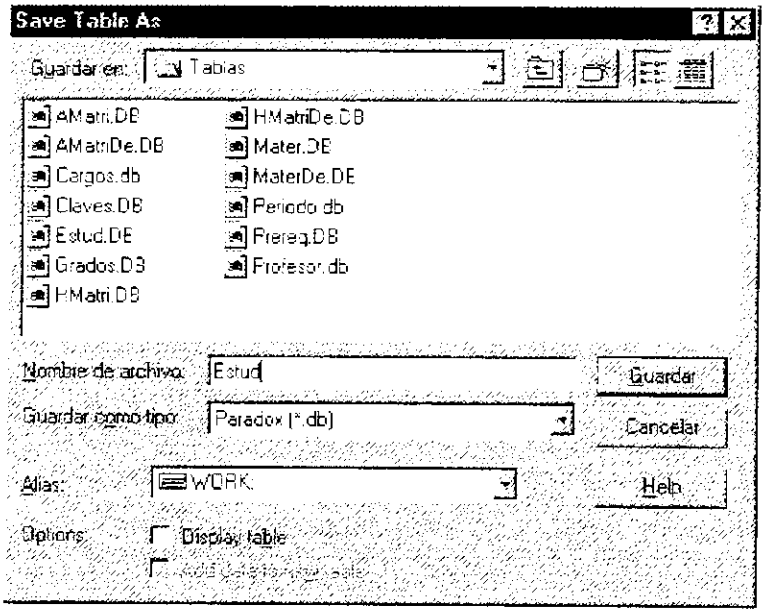


Figura 3.16. Caja de diálogo para Guardar una tabla

EL DATABASE FORM EXPERT

Delphi posee la capacidad de realizar una poderosa y compleja manipulación de bases de datos o actuar simplemente como manipulador frontal. Un proceso frontal de base de datos es un programa que proporciona una interfaz de fácil manejo para accesar y manipular la información contenida en una base de datos. Delphi incluye una herramienta que hace que el desarrollo de procesos frontales se realice casi sin esfuerzo, efectuando automáticamente casi todo el desarrollo. Esta herramienta se denomina DataBase Form Expert (Forma Experta de Base de Datos). Esta crea un proceso frontal de base de datos completamente funcional, sin una sola línea de código. Para accesar a éste simplemente se debe seleccionar *DataBase | Form Expert* desde el menú principal de Delphi y seguir los pasos hasta obtener una forma que se comunica con la(s) tabla(s) seleccionada(s) durante el proceso.

4.3. IMPRESIONES EN DELPHI

A diferencia de Delphi 1, Delphi 2 trae integrado una serie de componentes que permiten la creación rápida de reportes, éstos son los componentes QuickReport que se encuentran en la lengüeta QReport de la paleta de componentes (Figura 3.17).

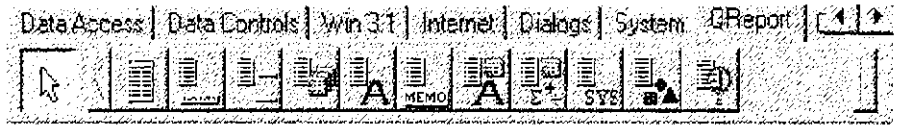


Figura 3.17. Paleta de componentes QReport

Los componentes QuickReport ofrecen la más rápida ruta para reportes de negocios profesionales, en aplicaciones Delphi. A continuación se da una breve descripción de cada uno de ellos:

QuickReport: Es el componente principal que permite generar un reporte.

QRBand: Es una banda en donde se ubican los diferentes elementos de un reporte, ya sean etiquetas, columnas, cabeceras de columna, etc.

QRGroup: Permite agrupar los datos para cálculos.

QRDetailLink: Permite crear relaciones Maestro/Detalle en los reportes.

QRLabel: Permite crear etiquetas en un reporte.

QRMemo: Permite visualizar campos memo en un reporte.

QRDBText: Permite visualizar los datos de una columna determinada de una tabla.

QRDBCalc: Permite ejecutar cálculos a partir de las columnas de una tabla.

QRSysData: Permite acceder a diferentes variables del sistema.

QRShape: Permite realizar formas simples de dibujo como líneas, círculos, etc.

QRPreview: Permite crear formas de visualización previa personalizadas

Crear reportes en QuickReport no es nada del otro mundo. A continuación se presenta los pasos para crear un Reporte simple:

- Se debe crear una nueva forma seleccionando *File . New Form* desde el menú principal de Delphi.
- Luego, se debe soltar un componente QuickReport sobre la forma.
- Desde la lengüeta Data Access de la paleta de componentes se debe soltar un componente Table y un componente DataSource.
- Modificar la propiedad DataBaseName del componente Table para que se conecte con una base de datos existente.
- Modificar la propiedad TableName del componente Table para que se conecte a una tabla.
- Poner la propiedad Active del componente Table a True.
- Modificar la propiedad DataSet del componente DataSource para que apunte al componente Table recientemente creado.
- Cambiar la propiedad DataSource del componente QuickReport para que apunte al Datasource recientemente creado.
- Soltar un componente QRBand desde la paleta de componentes QReport.
- Cambiar la propiedad BandType de éste componente a rbColumnHeader.
- Soltar sobre ésta un componente QRLabel y cambiar la propiedad Caption de éste último a un nombre de columna descriptiva (algo como Nombre, Edad, etc.)

- Soltar otro componente QRBand y poner su propiedad BandType a rbDetail.
- Sobre ésta última banda soltar un componente QRDBText, y modificar sus propiedades DataSource, para que apunte al Datasource antes creado, y DataField a una columna de la tabla antes creada.
- Al hacer doble Clic sobre le componente QuickReport se obtendrá un reporte simple (Figura 3.18).

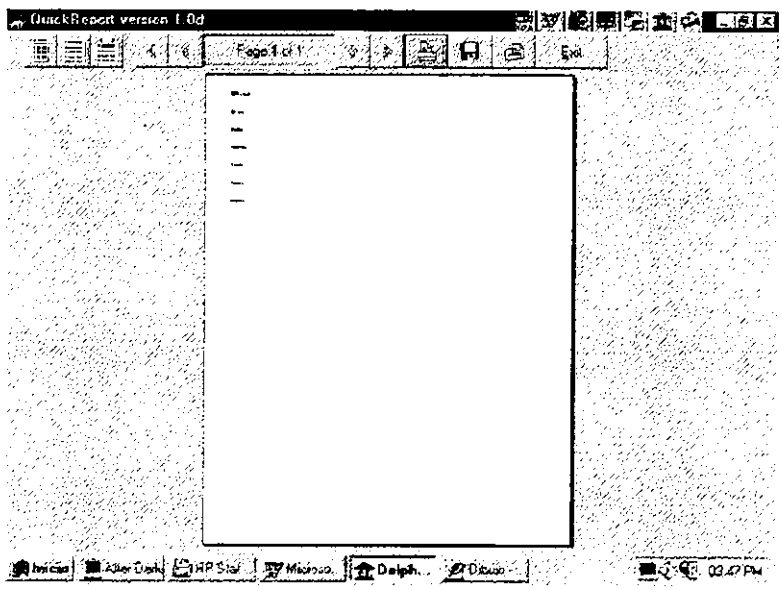


Figura 3.18. Reporte Simple

- Al basarse en éste breve ejemplo se puede crear reportes cada vez mas complejos de acuerdo a la experiencia que se vaya adquiriendo en el manejo de los componentes QuickReport.

CAPITULO 5

DISEÑO DE LA BASE DE DATOS

Este capítulo y el que sigue cubre el diseño de aplicaciones de bases de datos desde un punto de vista pragmático. Ellos introducen varios conceptos de la teoría de bases de datos y diseño de aplicaciones y muestran como aplicar éstos conceptos.

El proceso formal para construir una aplicación de base de datos es como sigue:

1. Definir el propósito y objetivos de la aplicación.
2. Diseñar las bases de datos fundamentales y los procesos de aplicación necesarios para cumplir estos objetivos.
3. Desarrollar el diseño dentro de una aplicación creando las bases de datos requeridas y los objetos de programa.
4. Probar la aplicación para ver si cumple con los objetivos establecidos.
5. Instalar la aplicación para uso productivo.

Este capítulo se enfoca sobre las partes de la lista que se aplican al diseño de las bases de datos. El próximo capítulo se enfoca a aquellos que se relacionan al diseño de la aplicación. El estudio se ha organizado de esta forma solamente para hacer el material más fácil de digerir, ya que los dos objetivos son, en realidad, dependientes uno del otro y así sería inútil tratar de separarlos completamente.

La presente aproximación para diseñar aplicaciones de base de datos difiere de la mayoría de las personas. En lugar de adentrarse en un largo discurso sobre teoría de base de datos se enfoca en el aspecto práctico. No se preocupa de discutir los finos detalles del papel histórico del Dr. E. F. Codd o las matemáticas detrás de la normalización de las bases de datos. Hay bastantes libros que ya hacen esto. Tampoco se hablará de los diferentes tipos de sistemas manejadores de bases de datos, se asume que se ha elegido una base de datos relacional del mismo tipo (Paradox). Tampoco se hacen mención de las herramientas CASE para el diseño de las bases de datos o aplicaciones. Las herramientas CASE son muy buenas pero, desafortunadamente, no todos tienen alcance a ellas o conocen como usarlas. Además las herramientas CASE son como artículos lujosos o cómodos, como un control remoto o un horno de microondas, son muy buenos y son ciertamente ahorradores de tiempo, pero son grandemente innecesarios y tienden a provocar que los diseñadores se vuelvan un poco holgazanes. Consecuentemente esta descripción se apegará a todo lo que viene en la caja de Delphi.

Ahora que se ha mencionado aquello que no será cubierto, se mencionará lo que sí será cubierto. Esta aproximación se concentrará sobre conceptos que se puedan poner rápidamente en práctica.

Una trampa común del diseño de bases de datos y aplicaciones es concentrarse demasiado sobre la teoría y no lo suficiente sobre la implementación. Este género de inbalance conduce a pobres implementaciones de base de datos y sus respectivas aplicaciones. Una implementación de una aplicación es la que el usuario ve, así, es importante poner adecuado énfasis sobre esto. Nadie comprará una casa cuyo piso se cae al minuto que camina sobre ella,

no importa cuan bien diseñada se vea. Lo mismo es verdadero en aplicaciones que se miran bien sobre el papel pero son deficientes en la pantalla. Un buen diseñador se esfuerza por construir aplicaciones que trabajen bien en teoría y en la práctica.

Algo más que se notará en el transcurso de éste capítulo y del que sigue es que no se discute el diseño de bases de datos como si éste fuera un sujeto separado del diseño de las aplicaciones de bases de datos. Esto es debido a que los dos están inseparablemente enlazados.

En la práctica nunca se ha diseñado una base de datos que no fuera proyectada para ser accesada, en alguna forma, por una aplicación final. Se debe recordar que una base de datos es sólo un medio para un fin, esto es, el camino para proveer los servicios que el cliente ha requerido. La aplicación de base de datos es el conducto entre el usuario y la base de datos. Esta, también, es sólo una herramienta para servir a su cliente. Ambas, la base de datos y su aplicación dependiente, deben trabajar juntas para proveer soluciones que los clientes encontrarán aceptables.

En este capítulo se dará información, trucos y atajos para el diseño de las aplicaciones de base de datos. Estas ideas se darán sobre un sólido marco de trabajo teórico. Aquí no se está tirando la teoría de las bases de datos. Al contrario, se está empleando para lo cual fue destinada. Para producir aplicaciones de bases de datos robustas.

5.1. PRIMEROS PASOS EN EL DISEÑO

5.1.1 DEFINIENDO EL PROPOSITO DE LA APLICACION

El primer paso en el diseño de cualquier aplicación –de base de datos u otra– es definir el propósito de la aplicación. La declaración del propósito de una aplicación debe consistir de una sentencia simple describiendo el sujeto, el verbo y el complemento del verbo. El sujeto es siempre la aplicación, tal como "El Sistema...." o "El Sistema médico...." El verbo describe lo que la aplicación se supone que hace, por ejemplo, "El Sistema llevará..." o "El Sistema médico manejará..." El complemento denota el destinatario de las acciones de la aplicación, por ejemplo, "El Sistema médico automatizará las tareas comunes de manejo de información en un laboratório clínico" o "El Sistema médico automatizará la consulta de digipuntura".

La declaración del propósito debe ser lo más simple y concisa posible. No se debe desperdiciar tiempo en lenguaje florido o información innecesaria. Por ejemplo, se debe evitar "para la organización" al final de la declaración porque esto es normalmente implícito.

Una vez definida la declaración del propósito para la aplicación, ésta debe ser mostrada a los potenciales usuarios de la nueva aplicación para ver si ellos están de acuerdo. No hay que sorprenderse si ellos no comprenden a la brevedad nuestra, pero se debe tratar de hacer que aprueben la declaración, asegurándoles que los objetivos específicos de la aplicación se enviarán después.

5.1.2 DEFINIENDO LOS OBJETIVOS DE LA APLICACION

Una vez que el propósito de la aplicación está en su sitio, es tiempo de determinar los objetivos específicos de la aplicación. ¿Qué es, lo que específicamente, está yendo ha hacer la aplicación? Los objetivos deben cubrir no más de tres o cuatro tareas mayores, en lo posible. Es una buena idea para desarrollar estos ítems usar un bosquejo. Estos objetivos no deben definir más que la declaración del propósito de la aplicación, no ir más allá de éste. Se debe seguir el mismo formato de tres partes que se usó con la declaración del propósito, por ejemplo:

El Sistema tiene como propósito manejar la información médica de laborátório. Cubre los siguientes objetivos:

- Registrar pacientes
- Registrar exámenes
- Controlar información de los pacientes y sus exámenes

Hay que asegurarse de cubrir todas las funciones principales de la aplicación, pero no exagerar con detalles innecesarios. También hay que asegurarse de que las tareas que se listan no se sobrepongan (no se debe mencionar una tarea que es cubierta por otra tarea que ya está en la lista).

5.2 DISEÑANDO LOS FUNDAMENTOS DE LAS BASES DE DATOS Y PROCESOS DE LA APLICACION

Esta etapa es donde las bases de datos propiamente dichas son diseñadas. La mayoría de las personas separan la parte de la base de datos de los procesos de aplicación, pero esto no deja de ser un error. Como se ha dicho una base de datos es un medio para lograr un fin – éste es el medio por el cual se satisfacen los requerimientos del cliente para almacenar y recuperar datos. Una base de datos y la aplicación que corre sobre ésta, están inexorablemente unidas. Por añadidura, sus diseños están también enlazados. Esta es la razón por la que se cubre el diseño de la base de datos y el diseño de la aplicación en la misma discusión.

5.2.1 MODELANDO DATOS

La primera cosa que se debe hacer en la construcción de una base de datos es determinar que tablas son necesarias y que es lo que las tablas deben contener. Modelar datos es el término formal para el proceso en el cual se hace esto. Hay varias aproximaciones reconocidas para modelar datos, cada cual con sus propios méritos. La primera es modelado centrado a la aplicación (*aplication-centric*), y la segunda es modelado centrado al género (*Kind-Centric*).

- El modelado centrado a la aplicación busca satisfacer totalmente las necesidades de los datos de una aplicación, con una simple base de datos. En el modelado centrado a la aplicación, cada aplicación tiene su propio, esencialmente privado, conjunto de tablas. Las tablas no se relacionen con el mundo exterior – la aplicación y la base de datos son esencialmente un mundo hacia ellas mismas.

- El modelado centrado al género organiza las tablas dentro de las bases de datos de acuerdo a los géneros que los datos representan. Por ejemplo, la tabla que lista los productos de una compañía puede existir en sus bases de datos de ventas, mientras que las partes del inventario usadas para construir estos productos residen en su base de datos industrial. Un reporte que muestre el monto de ganancias sobre cada producto debe referirse a ambas tablas, aunque ellas residan en bases de datos completamente diferentes.

Generalmente hablando, el modelado centrado al género es la aproximación preferida. Esta es la mejor solución para las bases de datos de una institución médica. Esta acaba con las tareas redundantes de bases de datos, al normalizar los bancos de datos de una organización.

Por otro lado, el modelado centrado en la aplicación tiene sus propias ventajas. Con esta aproximación el acceso del usuario al objeto de base de datos usado por una aplicación puede ser coordinado a través de una base de datos simple. Aplicaciones que usan cientos de tablas que se extienden a través de docenas de bases de datos pueden representar un reto real en lo concerniente a la administración de las bases de datos.

Además, copiar un objeto dependiente de una aplicación de servidor a servidor (por ejemplo, desde un servidor de desarrollo a un servidor de producción o un servidor de respaldos) es complicado, por magnitudes, cuando estos objetos residen en muchas bases de datos diferentes. El tomar una instantánea de los datos para respaldarlos es también difícil porque la mayoría de DBMS's no soportan instantáneas de bases de datos cruzadas. Sybase, por ejemplo, soporta instantáneas de bases de datos simples; una vez se empieza un backup en una base de datos dada, todos los datos escritos en el dispositivo de backup serán consistentes

en sólo un determinado tiempo. Este no es el caso con las bases de datos múltiples. Después de restaurar las base de datos de ventas y manufactura desde un backup, se puede encontrar que ellas no se emparejan – que productos existentes en la base de datos de ventas no son conocidos en la base de datos de manufactura.

Con respecto a Delphi mismo, almacenar objetos de bases de datos en múltiples bases de datos requiere múltiples alias BDE (Borland Database Engine) para accesar a éstos objetos y múltiples conexiones al que es, probablemente, un servidor de base de datos simple. Esto provocará que se tenga que registrar varias veces en el mismo servidor de base de datos – una vez por cada base de datos que se esté accesando.

Finalmente, tan lejos como van las aplicaciones comerciales, es raro que se esté habilitado para utilizar bases de datos o tablas que los clientes ya tienen. Se necesita una solución enlatada – algo que pueda ser instalado y usado desde fuera de la caja sin mirar por otras aplicaciones. Usualmente, una aproximación centrada a la aplicación trabaja mejor para éste tipo de desarrollo.

Lo mejor es usar una combinación de las dos aproximaciones. Es conveniente empezar cualquier diseño de aplicación de base de datos por delinear que es lo que la aplicación necesita hacer y entonces determinar que tablas son necesarias para hacer esto. Luego, se deben chequear las otras base de datos a disposición para ver si cualquiera de ellas tienen las entidades que se necesitan. Si ellas lo tienen, se hace uso de ellas en una de dos formas. Con múltiples alias acarreando el peso del acceso a múltiples bases de datos, o usando una VIEW SQL en la base de datos primaria de la aplicación para cada tabla externa referida. Se debe

usar ésta vista en lugar de la tabla base en la aplicación. La base de datos primaria de la aplicación es una base de datos centrada al género que es, o creada en conjunto con el proyecto o una que ya existe. Usando éste método se elimina la insistencia del BDE por un login separado para cada base de datos porque todos los objetos que existen en éste aparecen en una base de datos. Al mismo tiempo, éste preserva su centramiento al género de la base de datos – una VIEW no copia su tabla base; ésta es sólo referenciada a través de una consulta SQL. Esta es una buena transacción entre las dos metodologías.

Se debe recordar que no todos los servidores soportan referencia a objetos que se hallan en diferentes bases de datos. InterBase, por ejemplo, no soporta estos tipos de referencias externas. Esto significa que una VIEW InterBase en una base de datos no puede referenciar tablas en otras bases de datos. Por contraste, muchas plataformas, entre ellas Sybase SQL Server y Oracle, soportan estos tipos de referencias a objetos externos. Si se intenta crear un modelo de datos híbrido, como el mencionado, primero se debe asegurar que el servidor soporte referencias externas.

5.2.2 TERMINOS DE BASES DE DATOS

Antes de proceder, es una buena idea hablar acerca de los bloques básicos de construcción de bases de datos relacionales. Como ya se mencionó no se trata de entrar en una prolongada discusión sobre la teoría de bases de datos. Por otro lado, un conocimiento del funcionamiento de los bloques básicos es esencial para el apropiado diseño de la base de datos.

Por todo lo hablado estos días acerca de las bases de datos, muchas personas parecen estar confundidas en lo que es exactamente una base de datos. Gracias a los DBMS's basados en PC, muchas personas tienen la impresión que una base de datos y un archivo físico son la misma cosa. Algunas personas piensan que una hoja de trabajo de QPRO es una base de datos. Otros han sido conducidos para creer que es un archivo de WordPerfect. Las extensiones .DBF de DBASE tampoco ayudan. Allí parece haber un poco de confusión sobre lo que es esencialmente un concepto simple.

Una buena forma de pensar en una base de datos es imaginarla como un gabinete de archivos. Cada carpeta representa una tabla y cada hoja en cada carpeta representa un registro. Como se discutió en la sección previa sobre el modelado de datos, una base de datos es usualmente la colección de todos los datos de un género particular. Varios tipos de datos usualmente relacionados existen dentro de cada género. Estos tipos de datos son conocidos como tipo registro o clases de entidades y están incluidos en tablas. En su forma básica, entonces, una base de datos es simplemente una colección de tablas.

Cuando se habla acerca de bases de datos se debe aclarar el concepto de organización física y organización lógica. El hecho de que una base de datos esté compuesta físicamente de un archivo simple o de numerosos archivos nada tiene que ver con su disposición lógica. Algunos formatos de base de datos, por ejemplo dBASE y Paradox, usan numerosas tablas para almacenar sus elementos. Otras como Access usan sólo una. Esta diferencia nada tiene que ver con su uso o funcionamiento como base de datos.

El siguiente punto en la lista es el elemento clave en la construcción de bases de datos, la *tabla*. Se debe pensar en una tabla como una superficie de dos dimensiones dividida en líneas y columnas. Como en una hoja de cálculo, las líneas representan los registros en la tabla y las columnas representan los campos. Se puede asociar cada tabla como una carpeta en un metafórico gabinete de archivos, las bases de datos.

Un *registro* corresponde a los datos de un objeto del mundo real. Este puede ser una factura, un desembolso de una cuenta de ahorros o un listado de un libro de teléfonos. Los registros son el condumio de las bases de datos. Los puristas de las bases de datos prefieren el uso de Línea (Row) en lugar de registro. Aquí se usarán los dos términos intercambiablemente. En su forma más simple, las bases de datos y tablas son sólo mecanismos para organizar registros. En nuestra analogía, un registro es una hoja dentro de una carpeta en nuestro gabinete de archivos de base de datos. Si la carpeta fue llamada "FACTURAS", nosotros debemos esperar que cada elemento allí dentro es una factura del mismo tipo. De acuerdo a nuestra analogía, entonces, cada factura debe ser el equivalente de un registro en la tabla FACTURAS.

Los registros en una base de datos son conocidos como entidades – instancias de un registro particular o clase entidad. Las tablas albergan todas las entidades de una clase entidad particular. Así como una carpeta está en un gabinete de archivos que idealmente contiene todos los ítems de un tipo dado (por ejemplo, la totalidad de facturas de una compañía), así lo hace una tabla en una base de datos, contiene todos los registros de un tipo dado. Estos registros representan los datos reales de la base de datos. Muchos usuarios finales tienden a ver los datos desde un punto de vista centrado a registros.

Un *campo* es un elemento dentro de un registro. Análogo a una columna en una hoja de cálculo, un campo representa una característica de un objeto representado por un registro. Los puristas de las bases de datos prefieren el uso del término *Columna* en lugar de *Campo*. Aquí se usan los dos términos intercambiablemente. Un ejemplo de un campo podría ser el campo dirección de una tabla de clientes. Por sí misma la dirección no tiene sentido. En el contexto de su clase entidad, sin embargo, esta describe la dirección de un cliente.

Un *atributo* es la intersección entre una línea y una columna en una tabla. Esto es, el valor de un campo dado dentro de un registro particular es conocido como un atributo. Lo mismo que las líneas de un tipo particular de dato son conocidas como entidad de clase, una columna de un tipo particular es conocida como atributo de clase (Clase atributo o Attribute Class).

Un *identificador de entidad* es un campo o conjunto de campos que distinguen un registro de otro en la tabla. Este identifica en forma única cada registro en una tabla. Un ejemplo de un identificador de entidad podría ser el campo número de factura en una tabla factura. El número de factura en cada registro de la tabla Factura es único para cada registro. Si se guarda el valor del campo número de factura y va a otra parte en la tabla, se puede siempre regresar al registro original usando sólo el número de factura como la clave.

Una *clave primaria* es el campo o campos en una tabla que es usado como un identificador de entidad. Como identificador, el valor de la clave primaria de una tabla es siempre único para cada registro. Los campos que componen una clave primaria son también normalmente usados para construir un índice en la tabla para acceso rápido a sus líneas.

Una *clave externa* es un campo o campos en una tabla que, aún cuando no son usados como su identificador, se usa a menudo para juntarse con otras tablas. Por ejemplo, usando el ejemplo de la tabla factura, el campo número de cliente en una tabla factura puede ser establecido como una llave externa. El campo número de cliente no identifica un registro individual único de la factura – se puede tener varias facturas para un solo cliente. Sin embargo, la tabla cliente puede ser juntada con la tabla factura para producir un listado de ventas por cliente. Esto se haría probablemente usando el campo número de cliente en cada tabla. El campo número de cliente podría probablemente ser la llave primaria de la tabla *cliente* y una llave externa de la tabla factura.

Una *Join* (unión) es la relación lógica de una tabla con otra usando una llave común. Las tablas implicadas son normalmente referidas como el lado izquierdo y derecho de la unión, refiriéndose a la sintaxis SQL para la construcción de uniones. Una unión normalmente retorna un conjunto de resultados con elementos de ambas tablas. La sintaxis SQL para construir una Join tiene la siguiente forma:

```
SELECT CLIENTE.NumCliente, CLIENTE.NomCliente, PROPIEDAD.Direccion
FROM CLIENTE, PROPIEDAD
WHERE CLIENTE.NumCliente = PROPIEDAD.NumCliente
```

La unión viene en dos formas básicas: Unión interna (*Inner Join*) y unión externa (*Outer Join*). Una inner join no retorna líneas desde ninguna tabla si la condición de unión no se cumple. Una outer join retorna líneas desde la tabla de la izquierda indiferente de, si se cumplió o no la condición de unión. Si la condición no es cumplida, los campos en el conjunto de resultados que residen en la tabla de la derecha son regresados como NULL. El

ejemplo de la sintaxis SQL en párrafo precedente demuestra una inner join. El siguiente es la misma consulta expresada como una outer join.

```
SELECT CLIENTE.NumCliente, CLIENTE.NomCliente, PROPIEDAD.Direccion
FROM CLIENTE LEFT OUTER JOIN PROPIEDAD
ON CLIENTE.NumCliente = PROPIEDAD.NumCliente
```

Note el uso de la sintaxis LEFT OUTER JOIN en la segunda consulta. Esto causa que la consulta retorne las líneas en la tabla de la izquierda indiferente de, si una pareja es hallada en la tabla de la derecha usando los campos NumCliente de las tablas.

5.3 OBTENCIÓN DE PROCESOS OPERACIONALES A PARTIR DE LOS OBJETIVOS

Para continuar diseñando su aplicación, se debe desglosar aún más los objetivos que se perfilaron en el paso previo a los procesos operacionales. Haga una lista tan exhaustiva y detallada como pueda. Continúe refinando la lista hasta obtener procesos que parezcan operar sobre objetos de una sola base de datos, por ejemplo:

El Sistema tiene como propósito manejar la información médica de laboratorios clínicos. incluyendo:

Controlar pacientes actuales

Controlar pacientes anteriores

Controlar exámenes actuales

- Controlar exámenes anteriores

- Generar reportes de exámes de pacientes

5.4 DETERMINACION DE LOS OBJETOS DE BASE DE DATOS REQUERIDOS PARA ABASTECER A LOS PROCESOS

Mirando minuciosamente en los procesos que Ud. ha definido, determine que tablas de bases de datos serán necesarias para abastecerlos. Por ejemplo, en los procesos listados en el ejemplo precedente es evidente que las siguientes tablas serán necesarias:

- Una tabla que almacena los datos generales de los pacientes
- Una tabla para cada exámen de laboratório clínico

5.5 VERIFICACION DE LA PRESENCIA DE ESTOS OBJETOS EN BASES DE DATOS EXISTENTES

En el interés de eludir un diseño de base datos centrado a la aplicación, verifique las otras bases de datos a su disposición para ver si éstos objetos ya existen. Ud. puede necesitar la asistencia de su administrador de base de datos (DBA) para determinar esto.

5.6 EMPEZAR A CONSTRUIR EL ARMAZON DE LA BASE DE DATOS

Es tiempo de trasladar su trabajo hecho hasta ahora a los objetos de la base de datos y a los procesos del programa. Una vez que Ud. se decide por una lista formal de tablas y las relaciones de una con otras, es hora de construir el armazón preliminar de la base de datos. Este es preliminar debido a que probablemente cambiará. En las etapas tempranas del desarrollo de la aplicación de base de datos, el diseño de base de datos puede ser lo suficientemente maleable como para que vaya creciendo de acuerdo a las necesidades de la aplicación. Más adelante, esta debe ser un armazón firme capaz de soportar la aplicación que Ud. construya encima de ésta. Cambiar el diseño de la base de datos en etapas tempranas del proyecto es mucho más fácil que hacerlo después de que la aplicación está construida. No tema cambiar el diseño de la base de datos en éste punto inicial. Nada está esculpido en piedra en éste punto.

5.7 CONSTRUIR UN DEPOSITO DE DATOS

Cuando diseñe tablas para la base de datos, piense acerca de los elementos que la tabla necesita almacenar. Usando la lista de tablas que Ud. desarrolló anteriormente, construya una lista de todos los elementos de información que la aplicación necesita rastrear. Consolide estos elementos dentro de una sola lista maestra de todos los campos que existirán en la base de datos. Liste cada campo sólo una vez, y determine el tipo de dato óptimo y el tamaño de cada uno. Piense en su lista maestra como un depósito de campos – Ud. usa los campos en ésta para construir sus tablas. Algunas herramientas se refieren a ésta lista maestra como un

diccionario de datos o una tabla de referencia. En Delphi Ud. implementa éste tipo de depósito de campos usando "*attribute sets*" (conjunto de atributos) en el Database Explorer.

5.8 PENSAR EN LA EFICIENCIA

Pregúntese a Ud. mismo cuál es el tipo de dato más corto que puede usar para determinado campo y aún así almacenar el valor más largo posible. Si el campo nunca necesitará almacenar un valor más grande de que 255, almacénelo como tipo byte o entero corto. Use enteros en lugar de valores de punto flotante, cuando sea posible. Use variables tipo carácter en lugar de tipo carácter fijo cuando la longitud de un campo carácter pueda variar considerablemente de registro a registro.

5.9 TIPOS DE DATOS DE USUARIO

Si la plataforma de base de datos que Ud. usa soporta la definición de tipos de datos de usuario, éste es un buen momento para pensar acerca de ello. Un tipo de dato definido por el usuario es sólo una forma taquigráfica para referirse a un tipo de dato que el servidor ya soporta. InterBase llama a éstos "dominios". Ellos proveen un método fácil y consistente para definir columnas similares. Por ejemplo, supóngase que Ud. tiene varios campos en su repositorio de campos que representen nombres del mismo tipo, digamos, NombreCliente, NombreVendedor y NombreEmpleado. Vamos a suponer que Ud. quiere que todos los tres campos sean campos carácter con una longitud de 40. En lugar de definir todos los tres campos como VARCHAR(40), Ud. podría definir en cambio un tipo de dato de usuario llamado, digamos, NombreTYPE, éste mismo será definido como un dato tipo carácter con

longitud de 40. Ud. podría entonces usar NombreTYPE, en lugar de VARCHAR(40), como el tipo de dato para los tres campos. Esta es una forma más segura y legible de definir campos similares.

5.10 DESCRIBIR LOS DATOS

Es muy útil escribir una línea de descripción de cada campo en el repositorio de campos. Describir los datos contenidos en el campo y los valores aceptables que éste puede tener, si es aplicable. Esto ayuda a comprender el papel que cada campo jugará en la base de datos.

Opcionalmente, Ud. podría incluir la descripción de cada campo como un comentario en la declaración SQL de la tabla (declaración CREATE). Describir los campos de ésta forma le ayudará a eliminar redundancia y a clarificar su comprensión de la base de datos. En proyectos en donde intervienen varios programadores, esto también ayuda a otros a entender lo que Ud. estaba pensando cuando construyó una tabla dada.

5.11 USAR EL REPOSITORIO DE CAMPOS PARA CONSTRUIR LAS DEFINICIONES DE TABLAS

Luego, use éstos atributos para reensamblar sus tablas. Copie las definiciones de campos refinadas desde el repositorio de campos a las estructuras de tablas que delineó previamente. Debería ver que su diseño de base de datos lentamente empieza a tomar forma.

5.12 CODIFICAR SUS DEFINICIONES EN SQL

Algo muy útil en éste punto es codificar las estructuras de las tablas en SQL. Esto es, escriba una declaración SQL CREATE para cada tabla. Esto le fuerza a formalizar la definición de cada tabla. Esto también le ayuda a tomar decisiones con respecto al tipo de dato SQL que será usado. En éste punto Ud. tiene que decidir que campos pueden y cuales no pueden estar vacíos y definirlos como NULL o NOT NULL. No obstante, el hecho de que su servidor probablemente define un valor por defecto para la designación NULL/NOT NULL, especifique esto de cualquier modo – esto se hace para mayor claridad. Al hacer esto también se empieza el trabajo real de creación de las tablas porque Ud. puede más tarde usar esta declaración CREATE para construirlas (si Ud. usa herramientas como el DATABASE DESKTOP para construir sus tablas, escribir la declaración CREATE es opcional, pero aún así una buena práctica).

5.13 ORGANIZAR LAS TABLAS POR TIPO

Divida las tablas en tablas bases y transaccionales. Las tablas bases contienen elementos que son únicos por definición. Un ejemplo de una tabla base podría ser una que liste las capitales de las provincias, ya que hay una sola capital por provincia, la tabla capitales es una tabla base. Las tablas base son a menudo usadas para descifrar códigos, números o abreviaciones con alguno o mas significados. Por ejemplo, una tabla que liste los cargos más importantes de la institución podría ser una tabla base.

Una tabla transaccional recibe datos recolectados desde equipos o tecleados por usuarios. Esta cambia frecuentemente y es la parte más volátil de la base de datos.

5.14 TABLAS MAESTRO/DETALLE

Uno de los atributos de las bases de datos relacionales es que ellas no permiten la duplicación de datos dentro de una tabla individual. Consecuentemente, los datos son, en su lugar, duplicados entre tablas. Esto ocurre porque la llave primaria de una tabla puede ser duplicada como la llave externa de otra para establecer una relación entre ellas. Estas relaciones vienen en dos tipos básicos. Relación uno a uno y relación uno a muchos. Las relaciones uno a uno son bastante raras. Usualmente los datos contenidos en una segunda tabla que es accesada a través de una relación uno a uno pueden, en su lugar, ser incluidas en la primera tabla. Las relaciones uno a muchos, por contraste, son más comunes – ellas forman la base de la mayoría de aplicaciones de base de datos para PC (note que las relaciones muchos a uno son realmente relaciones uno a muchos invertidas). Las tablas involucradas en relaciones uno a muchos son comúnmente referidas como tablas Maestro/Detalle.

Dentro del grupo de tablas transaccionales, determine que tablas dependen de otras para proveer información significativa. Estas tablas son llamadas tablas detalle o tablas hijas, las tablas de las cuales ellas dependen son llamadas tablas hijas o padres. Las tablas detalle dependen de otras tablas para ser realmente útiles – ellas normalmente almacenan datos línea-artículo. Estas son también conocidas como hijas o tablas esclavas.

Algunas veces una tabla juega un doble papel – puede ser hija de una tabla y maestra de otra. Cualquiera que sea el caso. las tablas maestro/detalle son usualmente enlazadas por una combinación de llaves primarias y externas. Además del campo clave por el cual está enlazada a la tabla maestra, una tabla hija usualmente almacena una línea o secuencia del número del mismo tipo. Esto es lo que distingue a los registros hijos unos de otros, los cuales participan en una relación uno a muchos con una tabla maestra.

Un buen ejemplo de una relación maestro/detalle es un par de tablas que almacena información de facturas. Normalmente, una tabla funciona como la tabla maestra y almacena información de la cabecera para cada factura. La segunda tabla actúa como la tabla hija y almacena las líneas de artículos de cada factura.

5.15 VISUALIZAR LA RELACIÓN ENTRE LAS TABLAS

Luego, es útil visualizar el diagrama de relaciones entre tablas. Los diagramas de Entidad – Relación (ER) muestran la forma en que las entidades (registros) en una tabla se relacionan con las entidades en otra. Estas no necesitan ser muy elaboradas o usar símbolos complejos. El propósito de un diagrama ER es visualizar la relación entre las tablas, y Ud. puede hacer esto sin usar símbolos especiales. Aunque hay una especificación de diseño formal para diagramas ER, simplemente dibuje cajas sobre una hoja de papel y conéctelas con líneas. O, Ud. puede querer tomar una aproximación más formal. Como sea que Ud. lo haga asegúrese de comprender la manera en que sus tablas están relacionadas, antes de proceder.

Otra idea útil es crear el diseño esquemático completo de la base de datos. Esto es, yo creo un diagrama simple que muestra las bastas relaciones entre las tablas en la base de datos. Esta visión aérea de la base de datos le ayuda bastante para comprender qué datos contiene la base de datos y como se relacionan las tablas, unas con otras, dentro de la base de datos. La figura 5.1 ilustra tal diagrama.

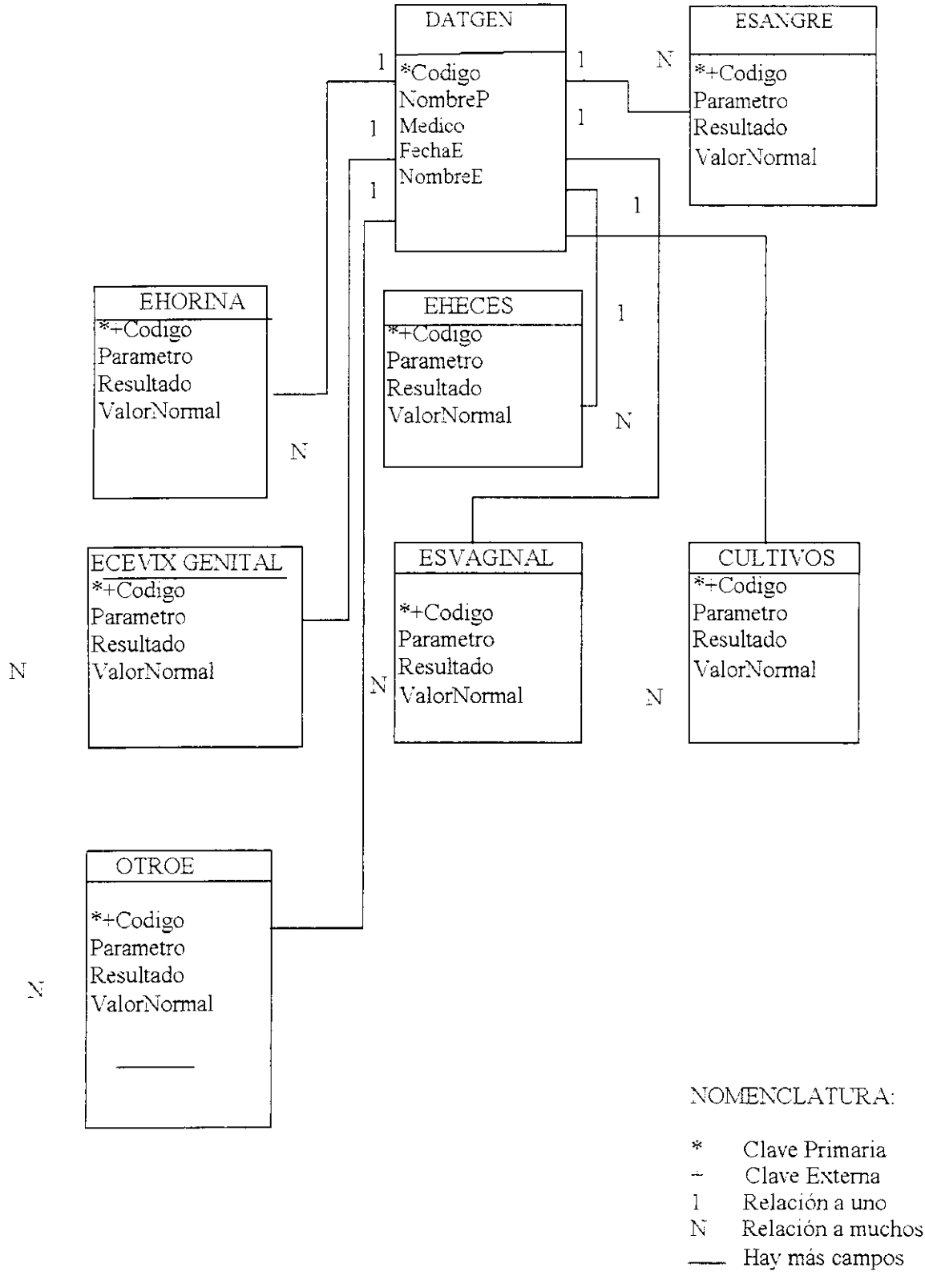


Figura 5.1. Diseño esquemático completo de la base de datos

5.16 DIAGRAMA DE FLUJO DE DATOS

Otra cuestión muy útil es construir el diagrama de flujo de datos para los procesos que componen la aplicación. Los diagramas de flujo de datos son elaborados desde los años 50, cuando las primeras computadoras empezaron a ser usadas en los negocios. Estos son orientados a tareas en lugar de orientados a datos. Usando símbolos especiales, ellos describen los pasos que componen un proceso y las tablas y dispositivos que éste usa.

5.17 NORMALIZAR LA BASE DE DATOS

El proceso de transformar datos en la forma racional discutido en 1970 por el Dr. E. F. Codd en el famoso artículo "Un modelo relacional de datos para dividir grandes bancos de datos", es llamado normalización. Simplemente, normalización es la eliminación de los datos redundantes de cada tabla en una base de datos. El propósito de la normalización es doble: para eliminar copias innecesarias de datos y para proveer la máxima flexibilidad en la estructura de las tablas, ya sea para el uso en la aplicación o para futuros cambios que pueden ser hechos en la base de datos.

Ud. necesitará normalizar las tablas en su base de datos antes de seguir adelante con el diseño de la aplicación. Algunas veces los procesos de normalización producen tablas adicionales que Ud. no introdujo en el diseño original. Saber sobre esto tan pronto como sea posible le ayudará a prevenir esfuerzos innecesarios en el desarrollo.

La Normalización es formalmente dividida dentro de cinco formas o etapas, desde la primera forma normal hasta la quinta forma normal. Estos términos oscuros realmente se refieren a los 5 juegos de criterio relacional que una tabla o cumple o no llega a cumplir. Cada etapa sucesiva se construye sobre una etapa previa. Aunque éstas son técnicamente 5 formas, en la práctica Ud. probablemente usará sólo las tres primeras. Las dos últimas son generalmente contempladas como demasiado especializadas para ser aplicables en un diseño de base de datos trivial.

5.17.1 PRIMERA FORMA NORMAL

Para que una tabla sea considerada normalizada en la primera forma normal, cada uno de sus campos deben ser completamente "atómicos", y éstos no deben contener grupos repetidos. Un campo es atómico cuando éste contiene sólo un elemento de dato. Por ejemplo, un campo dirección que contiene no solamente una dirección de calle, sino también una ciudad, estado y código zip no es atómico. Las columnas diseñadas de ésta forma deben ser separadas en múltiples campos para estar totalmente conforme con la primera forma normal.

Un grupo repetido es un campo que está repetido en la definición de registro por el sólo propósito de almacenar múltiples valores para el atributo. Por ejemplo, podemos tomar como ejemplo almacenar la propiedad que fue rentada por un inquilino en la tabla de datos del inquilino en lugar de separadamente en una tabla propiedades. Esto no permite la posibilidad de que un solo inquilino puede rentar más de una propiedad, al colocar ésta dentro de la tabla inquilino, tendríamos que decidir sobre el máximo número de propiedades que un inquilino

puede rentar y entonces añadir los campos de soporte necesarios a la tabla. Estos campos repetidos podrían ser grupos repetidos.

Algunas herramientas de bases de datos proveen soporte directo para grupos repetidos, y como tal, son inherentemente no relacionales. Obviamente éstas no pueden ser herramientas relacionales si violan la primera forma normal. Un ejemplo de tal herramienta es Advanced Revelation. Sus campos multivaluados son realmente grupos repetidos. Usarlos, lo cual es una práctica popular en aplicaciones AREV, viola la primera forma normal. Otro ejemplo es el lenguaje de programación COBOL. COBOL fue inventado antes de las bases de datos relacionales y por lo tanto acarrea con este bagaje desde los días de las bases de datos pre-relacionales. En particular, la sintaxis "groupname occurs several times" de COBOL es una violación directa de la primera forma normal. Esto no es sorprendente, sin embargo, porque COBOL fue originalmente diseñado para trabajar con redes de bases de datos en lugar de relacionales.

Todos éstos ejemplos de grupos repetidos son una muy pobre práctica en el diseño de bases de datos. Primero, usando el ejemplo de la tabla inquilinos, cada registro de inquilino incluye la repetición del campo propiedad aún cuando éste no sea usado. Segundo, la repetición de grupos presenta un reto cuando se procesan los datos que ellos representan. En particular, formatearlos apropiadamente para la impresión puede ser difícil. No sólo hacen que Ud. tenga que tratar con una serie de registros, sino que Ud. también tiene que tratar con una serie de campos, convirtiendo una tarea de una dimensión en una de dos dimensiones. Tercero, si el número máximo de propiedades que un cliente puede rentar necesita ser incrementado, la estructura de la tabla inquilinos debe ser cambiada.

5.17.2 SEGUNDA FORMA NORMAL

Para que una tabla esté normalizada en la segunda forma normal, cada uno de sus campos no clave deben ser totalmente dependientes de la clave primaria de la tabla y de cada campo de la clave primaria, cuando la clave primaria está compuesta de múltiples campos. Esto significa que cada campo no clave debe ser únicamente identificado por la clave primaria y sus campos componentes.

Vamos a tomar el ejemplo de una tabla factura de nuevo. Si la clave primaria de la clave consiste de dos campos NumLocalidad y NumFactura, almacenar el nombre de la localidad en cada registro podría violar la segunda forma normal. Esto es porque el campo NomLocalidad no podría ser totalmente identificado por la totalidad de la llave primaria. Este podría depender solamente del campo NumLocalidad – el campo NumFactura no tendría efecto sobre éste. En cambio, el campo NomLocalidad podría ser recuperado desde la tabla LOCALIDAD por medio de una JOIN siempre que aquel fuera necesario y no estar permanentemente almacenado en la tabla FACTURA.

5.17.3 TERCERA FORMA NORMAL

Para que una tabla esté normalizada en la tercera forma normal, cada uno de sus campos no clave debe ser totalmente dependiente de la clave primaria de la tabla e independiente de cualquier otro. Así, cumpliendo con los requisitos de la primera forma normal, cada campo no clave en una tabla debe ser independiente de los otros campos no claves.

Vamos a regresar a nuestra tabla de ejemplo. FACTURA. Vamos a decir que la llave primaria de la tabla es la misma (NumLocalidad y NumFactura). Uno de los campos no clave de la tabla podría ser, probablemente, el campo NumCliente. Si, con el campo NumCliente, el campo NomCliente fue también almacenado en la tabla FACTURA, la tabla no cumpliría con el criterio de la tercera forma normal porque los campos NumCliente y NomCliente pueden ser dependientes uno de otro. Si el campo NumCliente fue cambiado, ciertamente el campo NomCliente también debería cambiarse, y viceversa. El campo NomCliente debería en su lugar ser dejado en la tabla CLIENTES y ser accesado por medio de una JOIN, cuando haya la necesidad.

5.17.4 CUARTA FORMA NORMAL

La cuarta forma normal, prohibe que entidades independientes sean almacenadas en la misma tabla, cuando relaciones muchos a muchos existen entre ellas. Regresando al ejemplo de la tabla FACTURA, vamos a asumir que sólo un artículo puede ser listado en cada factura, y que la tabla FACTURA incluye una columna Vendedor y una columna Artículo. La columna vendedor se refiere al vendedor del artículo en la factura, "Vendedor X", por ejemplo. La columna Artículo contiene el código de la compañía para determinado tipo de producto que es comprado regularmente, digamos, papel copia de máquina. Allí pueden estar muchos registros de la tabla factura que se refieren al vendedor X – después de todo, el vendedor X vende muchas cosas, no sólo papel copia de máquina. Así mismo, allí pueden estar numerosos registros de la tabla que listan papel copia de máquina – después de todo, la compañía compra esto a cierto número de vendedores, no solamente al vendedor X. Se dice entonces que hay una relación de muchos a muchos entre las columnas vendedor y artículo. Para cada artículo

en la tabla FACTURA, pueden haber numerosos vendedores, y para cada vendedor en la tabla FACTURA pueden haber numerosos artículos. La cuarta forma normal requiere que Ud. almacene estas entidades en tablas separadas y que Ud. cree una relación de tablas para mantener enlaces con las tablas teniendo relaciones muchos a muchos con otras tablas.

Por supuesto, ya que las dos columnas tienen relaciones muchos a muchos entre ellas, no son realmente independientes, así, ellas ya violan la tercera forma normal. Esta es la razón por la cual la cuarta forma normal es, generalmente, considerada demasiado esotérica para ser de uso práctico en aplicaciones actuales. Las partes de ésta que importan son ya cubiertas por la tercera forma normal.

5.17.5 QUINTA FORMA NORMAL

La quinta forma normal requiere que Ud. esté habilitado para reconstruir sus datos originales partiendo de las tablas normalizadas. Esto significa que si Ud. empezó con tablas que no están normalizadas, Ud. debería poder regresar a ellas, una vez que ellas ya han sido normalizadas. El principal objetivo de esto es asegura que no se hayan perdido datos en el proceso de normalización. Por ejemplo, si Ud. empezó con la tabla FACTURA, previamente mencionada, que incluye ambas columnas, un NumCliente y un NomCliente, al normalizar la base de datos se debería crear una tabla CLIENTE y ser movido NomCliente a ésta. Ud. podría, por supuesto, copiar la columna NumCliente a la nueva tabla. Si Ud. cometió el error de simplemente remover la columna NomCliente desde la tabla FACTURA, Ud. no podría estar habilitado para reconstruir su tabla original no normalizada ya que NomCliente ya no existe en su diseño. Su diseño, entonces, podría violar la quinta forma normal.

En la práctica, el concepto de retener todos los elementos en una base de datos durante la normalización es tan intuitivo que esto no se da. Ya que Ud. no tiraría ciegamente datos elementales. Sin embargo, la quinta forma normal existe para ayudar a asegurar que no se den éstos casos.

5.17.6 NORMALICE, PERO NO EXAGERE

Una vez que Ud. empieza normalizando sus datos, es importante no ir demasiado lejos. El hecho de exagerar el proceso de normalización puede tener efectos devastadores sobre el desempeño. Por ejemplo, en una frenética normalización, Ud. puede estar tentado para establecer una tabla separada para la información de la empresa en que trabaja el inquilino, la cual está actualmente almacenada en la tabla INQUILINO. Después de todo, Empresa y DirecEmpresa son claramente dependientes una de otra violando, de ésta forma, la tercera forma normal. ¿Pero que beneficio real podría Ud. obtener de hacer esto? Ninguno en absoluto, ya que la información de la Empresa es solamente de interés en el contexto del inquilino con el cual ésta es asociada. Ud. no tiene necesidad para listar, por ejemplo todos los inquilinos trabajando para una determinada empresa. Ni podría Ud. querer cambiar toda la información relativa a la empresa. Así, establecer una tabla separada para la información de la empresa sería un desgaste de tiempo.

Hay veces que la normalización limitada es la única forma de obtener el desempeño requerido. Esto es especialmente cierto cuando se trabaja con grandes cantidades de datos.

5.18 REGLAS DE BASES DE DATOS

Una vez que las tablas están completamente normalizadas, el próximo paso es definir las reglas en el ámbito de base de datos para cada tabla. Estas reglas además definen las relaciones entre las tablas y las características de cada campo.

5.18.1 CLAVES

Los campos clave identifican el método de acceso que Ud. piensa usar para llegar a una línea de la tabla. Las claves que Ud. selecciona en ésta etapa se traducirán más tarde en índices. Escoja todas las claves que Ud. necesite, pero no seleccione campos como claves innecesariamente. Indices no requeridos gastan espacio en disco y aminoran su DBMS. También, no se moleste en seleccionar el primer campo o campos de una clave compuesta como una clave separada. Por ejemplo, si Localidad y NumFactura forman la clave primaria para la tabla FACTURA, no hay necesidad de señalar también localidad como una clave separada. Casi todos los DBMS's soportan búsquedas por claves parciales, así crear un índice sobre la columna localidad podría gastar espacio del disco y recursos del DBMS.

5.18.2 CLAVES PRIMARIAS

Revise, las tablas en su base de datos y especifique una clave primaria para cada una. Recuerde que la clave primaria puede ser una clave compuesta – es decir que puede estar formada de más de un campo. La clave primaria que Ud. selecciona debería ser inherentemente única y debería ser el camino más popular de acceso que Ud. planea tomar en

los datos de la tabla. Esto es, piense acerca de la forma en la cual un dato de una tabla puede ser usado. ¿Cómo se pregunta más a menudo? ¿Qué campos en ésta tabla pueden referenciar otra tabla en una restricción?

Note que la columna o columnas que Ud. identifica como una clave primaria de la tabla se usará para construir una restricción de clave primaria.

5.18.3 CLAVES SECUNDARIAS

Use claves secundarias para establecer caminos de acceso secundarios en una tabla. Esto son caminos a los datos menos populares pero esenciales. No exagere en este punto ya que éstas selecciones de claves se traducirán a índices e índices inútiles gastan espacio en disco.

Señale como claves secundarias a los campos que Ud. piensa que pueden participar en una JOIN. También establezca como claves secundarias aquellos campos que, Ud. piensa, el usuario usará para búsquedas.

5.18.4 RESTRICCIONES

Hay tres tipos básicos de restricciones: restricciones de clave primaria, restricciones de clave externa y restricciones de control (algunas plataformas se refieren a las columnas por defecto como restricciones). Las restricciones de clave primaria aseguran que los valores sean únicos dentro de una tabla. Las restricciones de clave externa aseguran que los valores en una tabla

existan en otra. Las restricciones de control garantizan que el valor de una columna cumpla con un conjunto dado de criterios.

5.18.5 VALORES POR DEFECTO

La siguiente clase de reglas de bases de datos para establecer son los valores por defecto para los campos en cada tabla. Revise cada tabla y determine cuales, sino todos, de los campos deberían tener un valor por defecto asignado a éste.

5.19 CREAR LOS OBJETOS DE LA BASE DE DATOS

Una vez que las dependencias entre las tablas, reglas, claves, etc. han sido todas definidas, su diseño de base de datos está prácticamente completado, Ud. está ahora listo para implementar éste diseño creando los correspondientes objetos de la base de datos. Esto involucra crear una base de datos en la que se almacenen sus objetos de base de datos, si es necesario, entonces crear las tablas, índices, restricciones, reglas, valores por defecto, etc. necesarios para actualizar su diseño.

CAPITULO 6

DISEÑO DE LA APLICACIÓN

El capítulo previo listó los 5 procesos o etapas de desarrollo de una aplicación de base de datos. Para reiterar, los cinco pasos son:

1. Definir el propósito y objetivos de la aplicación.
2. Diseñar los fundamentos de la base de datos y procesos de aplicación necesarios para cumplir con estos objetivos.
3. Desarrollar el diseño en una aplicación para crear las bases de datos requeridas y los objetos del programa.
4. Probar que la aplicación cumpla con los objetivos establecidos.

El capítulo anterior "Diseño de la base de datos", se enfocó sobre los elementos de los cinco procesos que son relativos al diseño de la base de datos. Este capítulo se enfoca sobre lo relativo al diseño de la aplicación. Debido a que, el diseño de las bases de datos y el de la aplicación están estrechamente relacionados, es difícil determinar donde finaliza la una y empieza la otra. Una comparación que viene al caso sería pensar en el diseño de la aplicación de base de datos en términos de una cuadrícula de dos columnas – una para el diseño de base de datos y una para el diseño de la aplicación – cuyas líneas consisten de los cinco procesos formales. La cuestión clave es comprender que el diseño de la base de datos y el diseño de la aplicación están inseparablemente enlazados.

Vamos ahora a retomar donde nos quedamos en al capítulo anterior y continuar con los cinco procesos formales para diseñar aplicaciones de bases de datos.

6.1. DISEÑAR LOS FUNDAMENTOS DE LA BASE DE DATOS Y PROCESOS DE APLICACIÓN

Ahora que los fundamentos de la base de datos han sido diseñados, es hora de seguir adelante y construir los procesos de aplicación necesarios para cumplir con los objetivos de la aplicación. La base de datos actúa como una plataforma sobre la cual el resto de la aplicación es construida. Es importante finalizar ésta tanto como sea posible antes de ir más lejos en la fase de construcción de la aplicación. Los cambios hechos a la base de datos después de que la aplicación es construida pueden afectar a porciones significativas de la aplicación teniendo que ser rediseñada o reescrita.

6.2. DECIDIR EL TIPO DE APLICACIÓN

El primer paso en el proceso de diseño de una aplicación es decidir exactamente que tipo de aplicación Ud. está construyendo. Las aplicaciones pueden clasificarse en dos tipos: Sistemas de Procesos Transaccionales y Sistemas de soporte de decisiones. Generalmente hablando, los sistemas para soporte de decisiones son utilizados para manejar vistas globales de los informes para una porción de datos de una compañía. Los sistemas de procesamiento de transacciones, por otro lado, son responsables de la entrada y manipulación de éstos datos.

Las aplicaciones para soporte de decisiones necesitan solamente accesar a los datos en un modo de sólo lectura. Los sistemas de procesamiento de transacciones necesitan habilitarse para lectura y escritura de datos. A causa de que ésta es la diferencia fundamental entre los dos tipos de aplicaciones, Ud. necesita decidir que tipo de aplicación planea construir, antes de haber iniciado.

6.3. DIAGRAMA O ESQUEMA DE LOS PROCESOS OPERACIONALES DE UNA APLICACIÓN

Usando la lista de procesos operacionales que Ud. desarrolló en el capítulo anterior, trace gráficamente el flujo de datos de cada proceso separado, de la aplicación. Puede usar símbolos formales de diagramas de flujo o solamente declaraciones escritas a mano de qué es lo que pasa proceduralmente en la aplicación.

6.4. VISUALIZAR LOS ENLACES RELATIVOS A LOS PROCESOS

Una vez que Ud. ha trazado los procesos operacionales que componen la aplicación, visualice cualquier enlace relativo a procesos. Por ejemplo, si una pila de reportes es impresa cada vez siguiendo un proceso de cierre, muestre el enlace entre los dos procesos.

Una vez que Ud. ha visualizado el enlace relativo a procesos, construya un diagrama que de una apreciación global de la aplicación, un diagrama de flujo del sistema. Aunque una pieza dada de datos puede no ir a través de todos los procesos operacionales que componen su

aplicación, construya un diagrama que ilustre los procesos que componen la aplicación de cualquier modo. Esto le dará a Ud. una comprensión total del alcance de la aplicación.

6.5. ETIQUETAR LOS ELEMENTOS OPERACIONALES POR TIPO

Después, Ud. debería ir a través de los elementos que componen los procesos operacionales y etiquetar cada uno de acuerdo al tipo que éste representa. Decida que nodos tendrán formas correspondientes o reportes y cuales tendrán un código de soporte no interactivo. Cada nodo debería tener una etiqueta asociada con éste. Dentro de aquellos nodos que tienen formas asociadas, decida que tipos de formas tendrán ellos. La forma será una forma de soporte de decisión, una forma de soporte de transacción o una forma de entrada de datos. Etiquete los elementos que componen sus procesos operacionales, en consecuencia. Es recomendable etiquetar cada nodo con el nombre que se piensa usar para la forma, reporte o código fuente (*unit*) correspondiente.

6.6. DISEÑO JERARQUICO DE FORMAS Y REPORTES

Una vez que Ud. ha determinado, aproximadamente, que formas y reportes son necesarios para su aplicación, diseñe una jerarquía separada para sus formas y reportes, para tomar ventaja del soporte de herencia de las formas visuales de Delphi. Esto es, organice sus formas y reportes en clases generales que se construyan una sobre otra. Por ejemplo, Ud. debería definir una forma de nivel superior para su aplicación desde la cual descenderán todas las otras. Si Ud. más tarde necesita cambiar un atributo de todas las formas en su aplicación, tiene la suerte de estar habilitado para hacer los cambios en sólo un lugar. Lo mismo es cierto para

las formas jerárquicas de los reportes. Después de que Ud. probablemente finalizó el diseño de la mayoría de sus reportes usando los componentes QuickReport de Delphi, puede realizar cambios globales en éstos.

6.7. IDENTIFICAR Y ADQUIRIR CÓDIGO DE SOPORTE DE TERCERAS FUENTES

Una vez culminado el proceso anterior, intente determinar cualquier librería de soporte y código de terceras fuentes que pueda necesitar. En una gran empresa, Ud. puede necesitar hablar con el administrador de recursos acerca de las librerías de soporte utilizadas y que han sido desarrolladas internamente por la organización. El momento que Ud. decide que procesos tendrán formas asociadas y cuales tendrán código no interactivo asociado es un buen momento para determinar que librerías de soporte pueden necesitarse.

6.8. HORARIO O PLAN DE LOS PROCESOS A DESARROLLAR

La siguiente cosa que necesita hacer es planificar el desarrollo de las formas y código de soporte requerido para su aplicación. Hay tres factores básicos que afectan esta planificación. Primero, su cliente puede requerir que ciertas partes de la aplicación sean desarrolladas antes que otras. Por ejemplo, Ud. puede ser comisionado para desarrollar la aplicación en piezas, desplegando cada pieza separadamente. El cliente puede tener una idea acerca de qué piezas querrá ver primero. Segundo, algunas partes de la aplicación pueden ser requisitos de otras. Por ejemplo Ud. puede necesitar desarrollar una serie de rutinas para manejo de Strings antes de que Ud. pueda proseguir con una rutina de desarrollo que analice un archivo de texto. Si

éste es el caso, Ud. necesitará primero desarrollar estos módulos de prerequisito. Tercero, desarrollar algunas partes de la aplicación antes que otras puede ayudarle en su esfuerzo de desarrollo. Por ejemplo construyendo una forma principal de la aplicación antes que las otras formas le ayudará a Ud. en la construcción de las otras formas, ya que Ud. puede probarlas en tiempo de corrida desde el menú principal de la forma.

6.9. LINEAS DE TIEMPO

El proceso de planificación de un proyecto a lo largo de una línea de tiempo es verdaderamente un arte complejo. Se complica más por la perspectiva de desarrollo en equipo.

6.10. CONSTRUIR LA APLICACIÓN

Lo siguiente es el desarrollo real de la aplicación de acuerdo al horario que Ud. ha elaborado. La artesanía del desarrollo del código fuente en una forma estructurada varía considerablemente. Recuerde que el énfasis sobre el diseño del código fuente ha sido aminorado en una gran cantidad por el lanzamiento de herramientas visuales como Delphi. El énfasis en cambio se ha trasladado al diseño de formas.

6.11. DISEÑO DE LA FORMA

La aproximación que Ud. toma para diseñar formas depende en gran medida del tipo de aplicación que se está construyendo y del tipo de formas que Ud. está construyendo para esa

aplicación. Así es muy importante decidir el tipo de aplicación y de formas antes de seguir adelante.

CONCLUSIONES Y RECOMENDACIONES

1. El Análisis, el Diseño y la Programación pueden llevarse a cabo tomando como base el enfoque de la Orientación a Objetos.
2. En la presente aplicación, el proceso de Análisis se lo realiza con la metodología orientada a objetos así como también el diseño y la implementación .
3. Si se deseara, en un futuro, cambiar a Cliente/Servidor la presente aplicación debería trasladar la base de datos actual de Paradox a un DBMS que soporte cliente servidor como Interbase.
4. El proceso de traspaso a Cliente/Servidor requiere una modificación del diseño de base de datos y de la aplicación..
5. Delphi no sólo es una poderosa herramienta para desarrollar aplicaciones de bases de datos sino, también es una excelente herramienta para uso didáctico.
6. Al desarrollar una aplicación como ésta un estudiante queda capacitado para desarrollar aplicaciones comerciales de bases de datos.
7. Al usar el Object Pascal como su lenguaje, Delphi provee de flexibilidad y de una gran variedad de soluciones a los comunes problemas de desarrollo de base de datos.
8. Al ser totalmente orientado a objetos Delphi supera a Visual Basic.
9. Usando los manejadores adecuados, Delphi puede trabajar con tablas de cualquier DBMS como Paradox, Access, FoxPro, etc.
10. Cuando se tiene a disposición un poderoso lenguaje como es Object Pascal los caminos de solución a los problemas son múltiples.
11. Debido a que el Sistema desarrollado en la presente tesis de grado se halla en continuo cambio y evolución, de acuerdo a las nuevas necesidades que las instituciones médicas

van adquiriendo. Se aconseja un profundo estudio del mismo, para de acuerdo a ello realizar actualizaciones pertinentes.

BIBLIOGRAFIA

COAD, P., YOURDON., E. Object oriented analisys. Englewood Cliffs. Nueva Jersey. Yourdon Press. 1990.

GUREWICH, N., GUREWICH, O. Teach yourself Database Programming with Delphi in 21 days. Primera edición. USA. Sams Publishing. 1995. 569 p.

HENDERSON, K. Database developer's guide with Delphi 2. Primera edición. USA. Sams Publishing. 1996. 857 p.

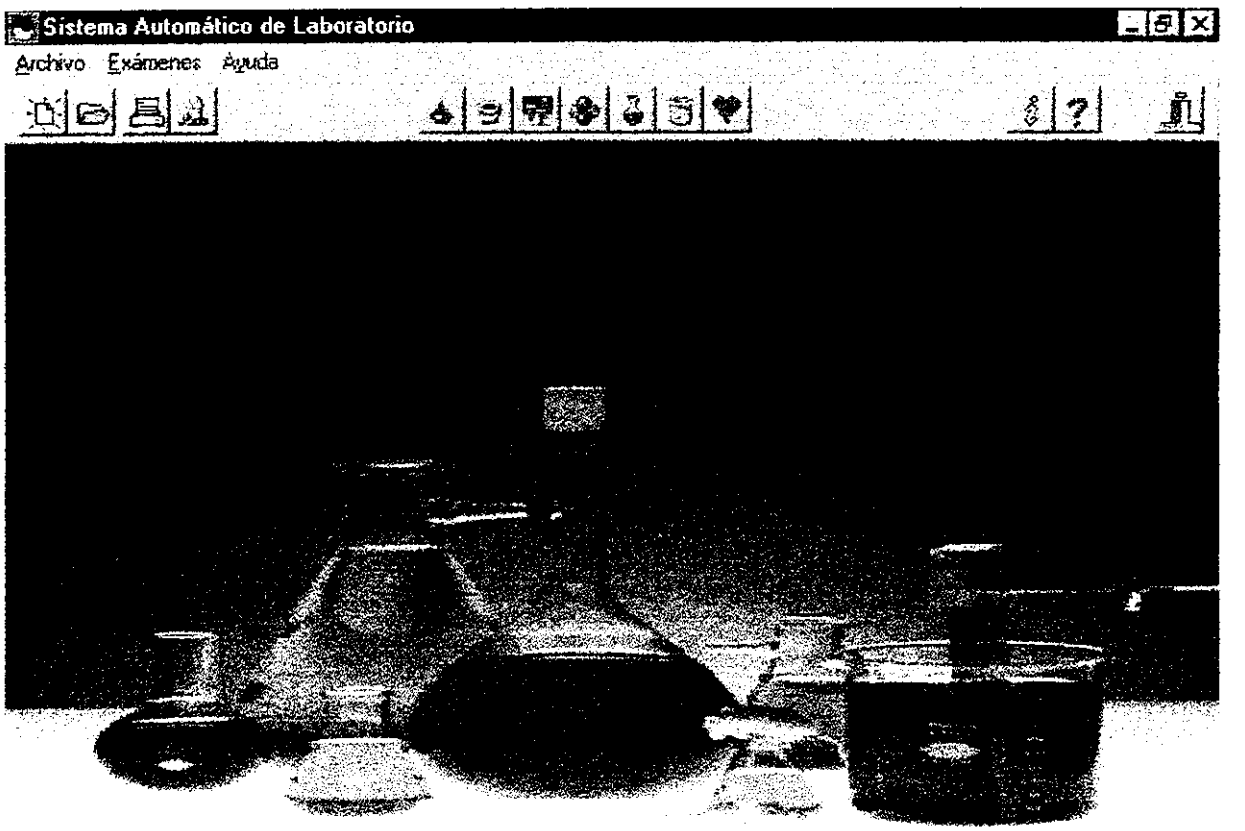
OSIER, D., GROBMAN, S., BATSON, S. Aprendiendo Delphi 2 en 21 días. Primera edición. México D.F. A Simon & Schuster Company. 1996. 734 p.

MANNING, M. M. Delphi 2.0 Guía oficial de Borland. Primera edición. México D.F. A Simon & Schuster Company. 1996. 410 p.

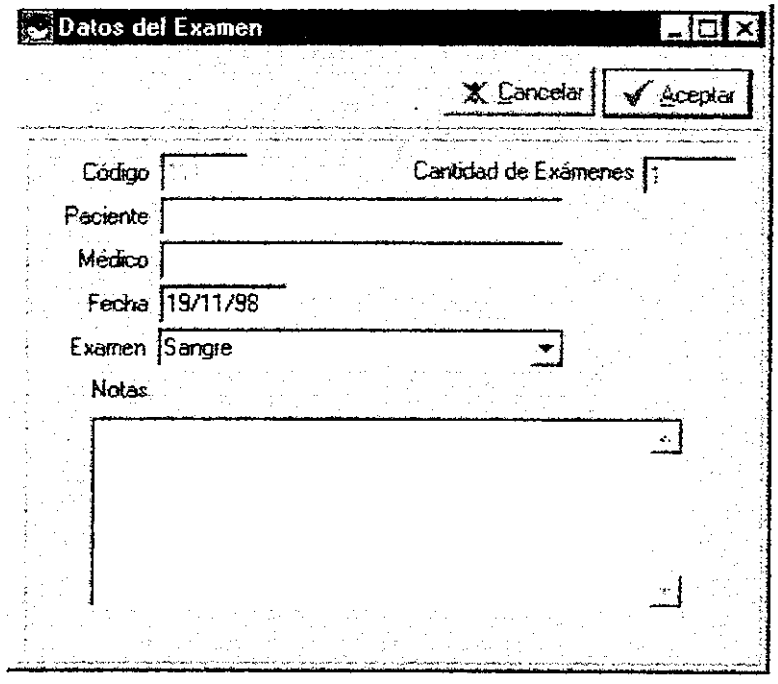
ANEXOS

LISTA DE ANEXOS

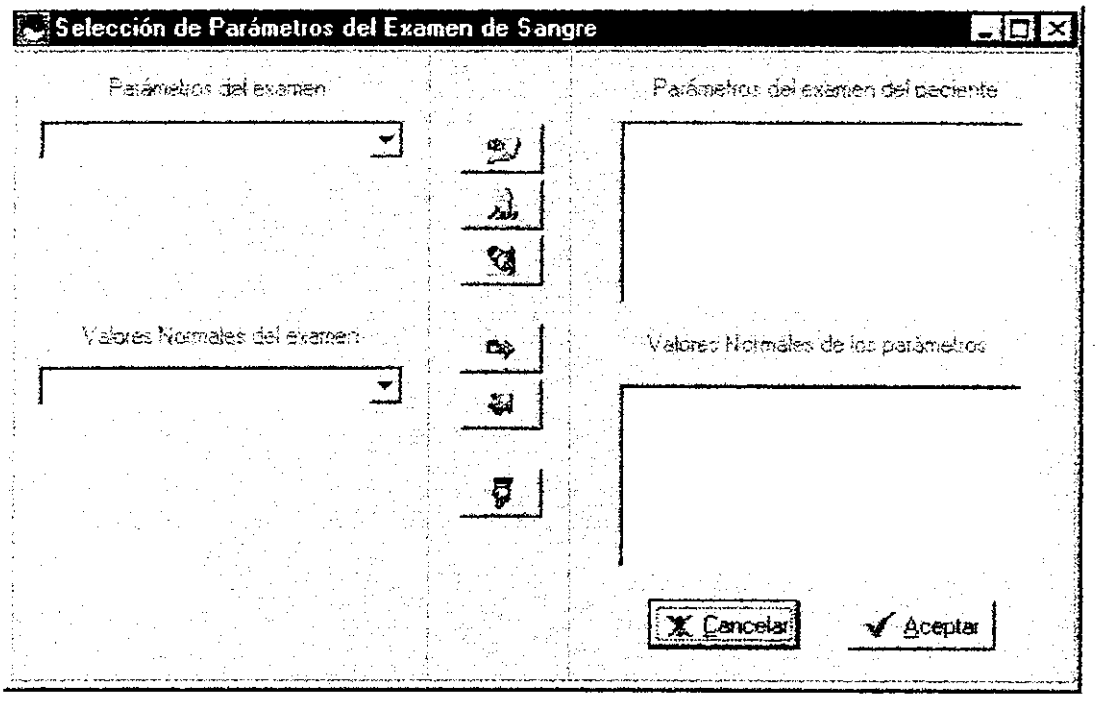
Sistema Automático de laboratorio



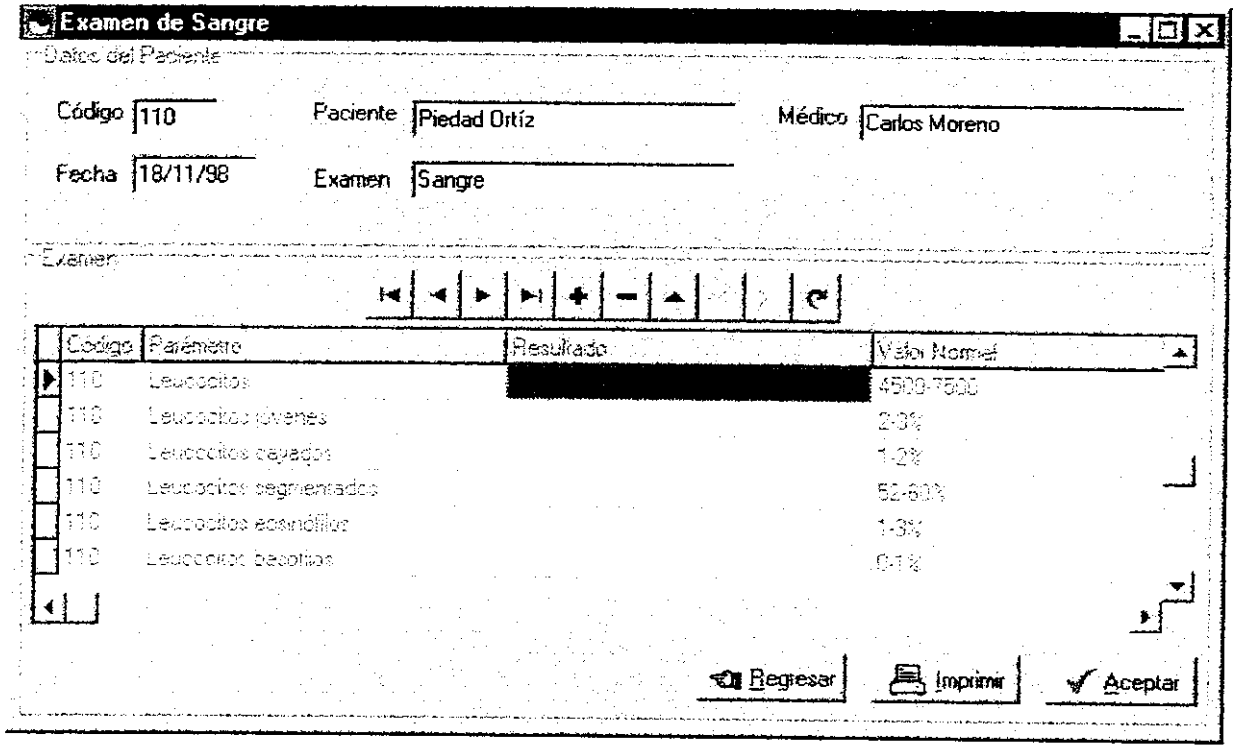
Forma para el ingreso de un nuevo examen



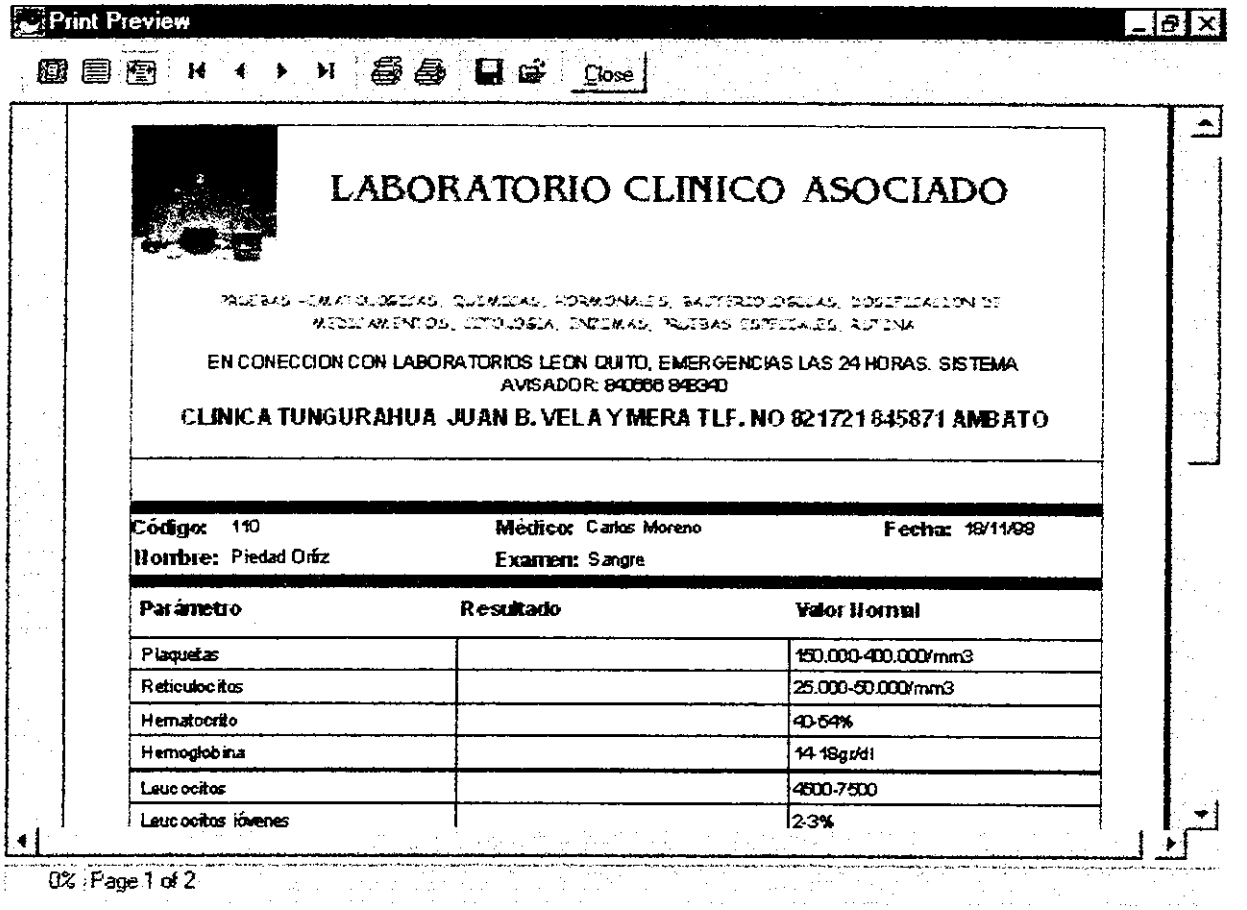
Forma para seleccionar los parámetros del examen



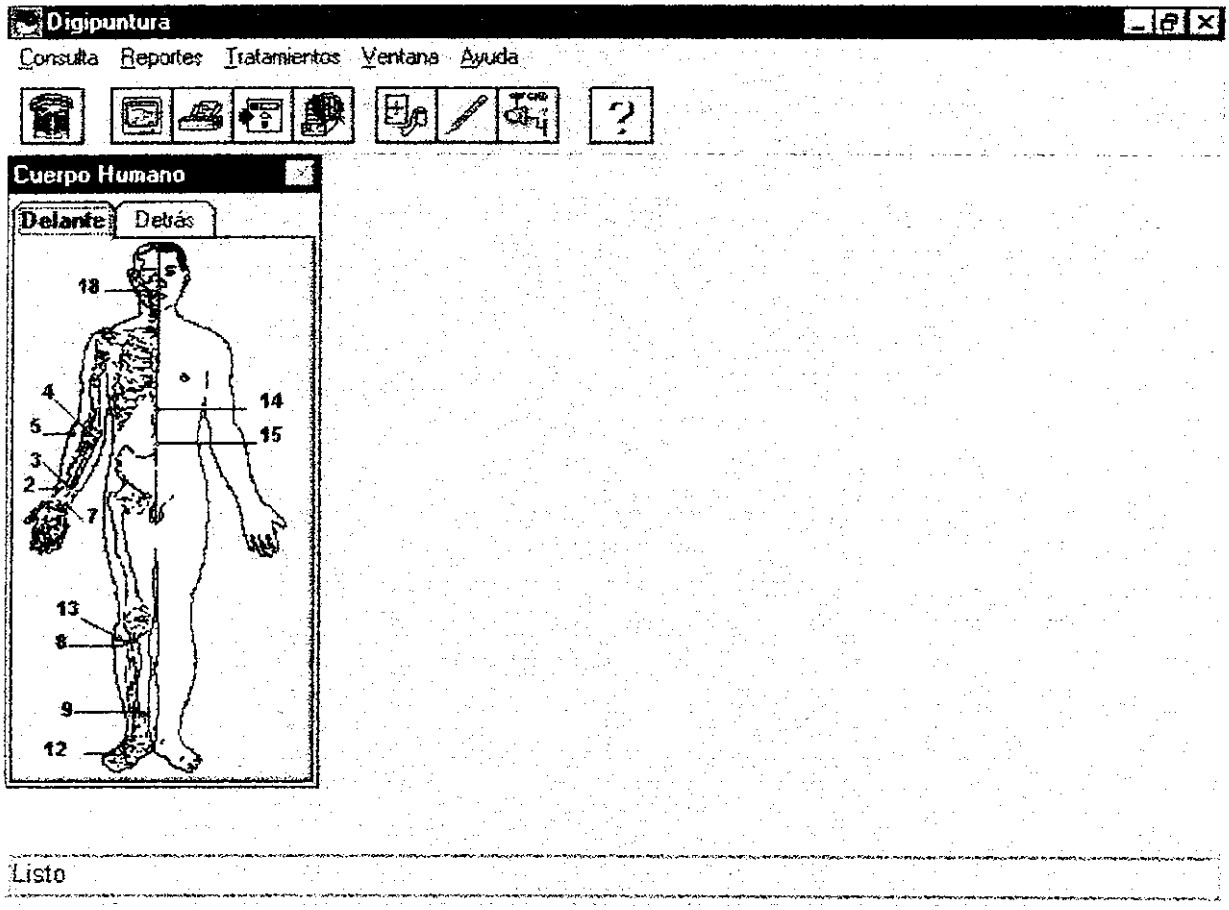
Forma para seleccionar los parámetros del examen



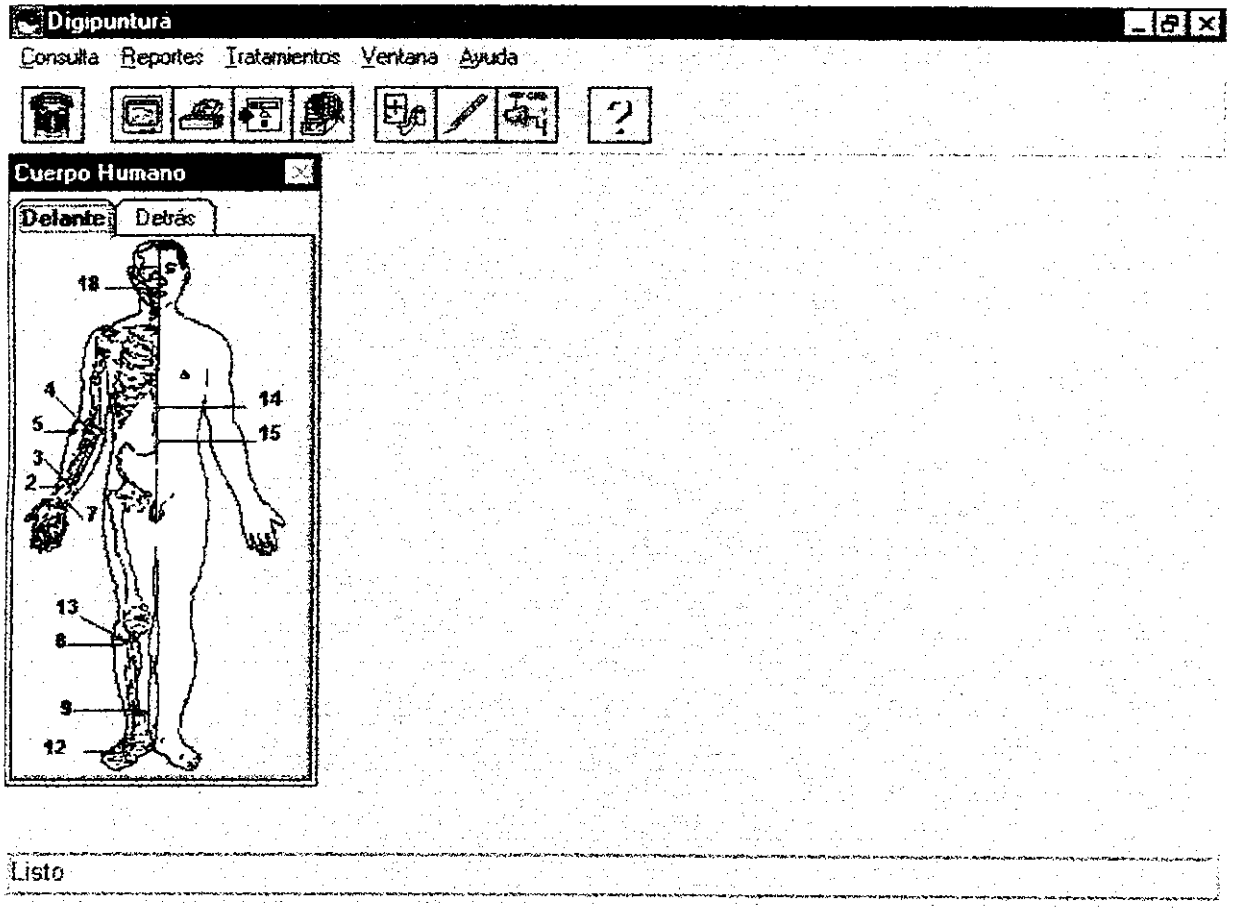
Forma para imprimir los resultados del examen



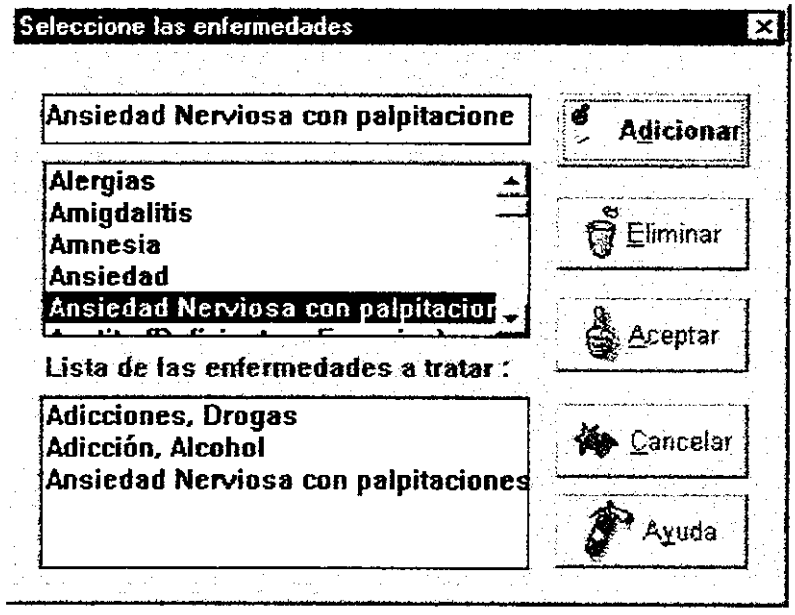
SISTEMA DE DIGIPUNTURA

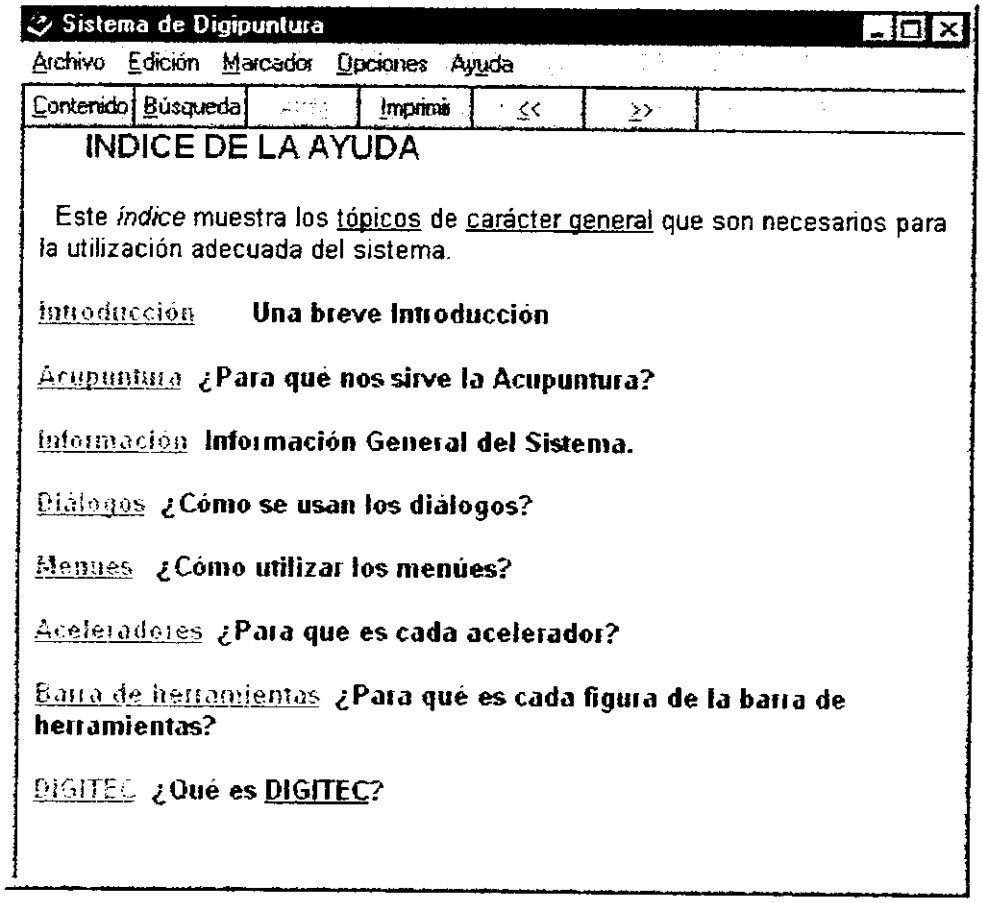


SISTEMA DE DIGIPUNTURA

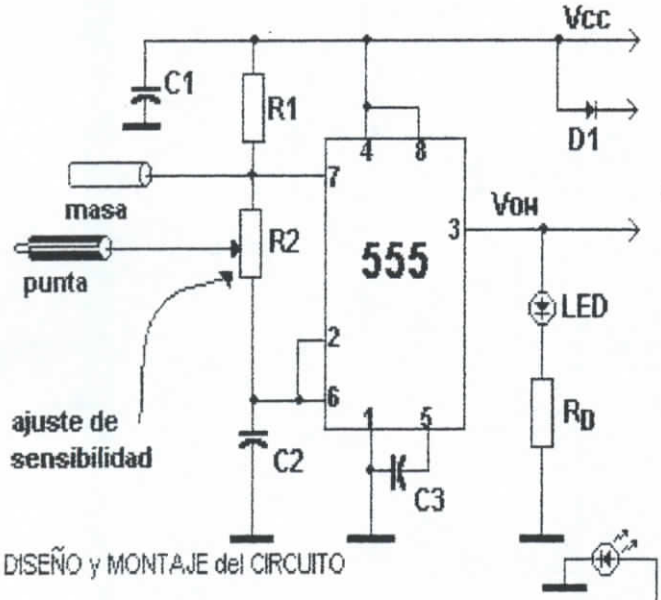


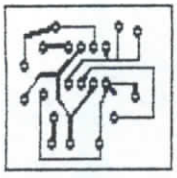
Forma para tratamiento por enfermedades

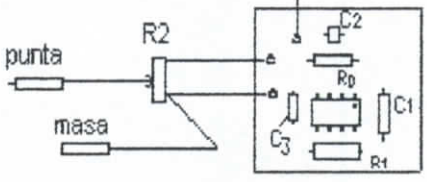


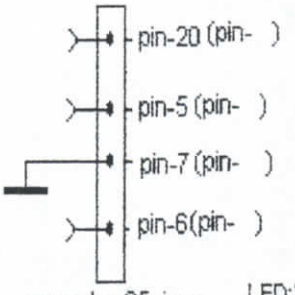

Sistema de Digipuntura
Archivo Edición Marcador Opciones Ayuda
Contenido Búsqueda Imprimir << >>
INDICE DE LA AYUDA
Este índice muestra los tópicos de carácter general que son necesarios para la utilización adecuada del sistema.
Introducción Una breve Introducción
Acupuntura ¿Para qué nos sirve la Acupuntura?
Información Información General del Sistema.
Diálogos ¿Cómo se usan los diálogos?
Menues ¿Cómo utilizar los menúes?
Aceleradores ¿Para que es cada acelerador?
Barra de herramientas ¿Para qué es cada figura de la barra de herramientas?
DIGITEC ¿Qué es DIGITEC?

Diseño electrónico e impreso de DIGITEC









R1= 10 KΩ
R2= 1 MΩ
C1= 10 nf
C2= 100 nf
C3= 10 nf
D1- 1N4148

LED: Valores Maximos

(voltage) V_D - 2v
(corriente) I_D - 10 mA

$$R_D = \frac{V_{OH} - V_D}{I_D}$$

Ej: V_{OH} = 5 v
V_D = 2 v
I_D = 10 mA

$$R_D = \frac{5\text{ v} - 2\text{ v}}{10\text{ mA}} = 300\ \Omega$$