PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
SEDE AMBATO

Unidad de Ingeniería de Sistemas

Disertación de Grado previa la obtención del título de Ingeniero en Sistemas

"Estudio comparativo de las herramientas de programación visual y su aplicación en el desarrollo de software financiero bajo el sistema operativo Windows"

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
PROGRAMAS ACADEMICOS AMBATO
BIBLIOTECA

Disertación de grado de:

Rubén Eduardo Nogales Portero

Director de Disertación:

Ing. Mst. Wigberto Sánchez Peña

PONTIFICIA UNIVERSIDAD CATOLICA
DEL ECUADOR
AMBATO
INGENIERIA DE SISTEMAS
SECRETARIA

Ambato, 2000

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
SEDE AMBATO

Disertación de Grado previa la obtención del título de Ingeniero en Sistemas

"Estudio comparativo de las herramientas de programación visual y su aplicación en el desarrollo de software financiero bajo el sistema operativo Windows"

Director de Disertación:

Ing. Wigberto Sánchez

Revisores de Disertación:

Ing. Patricio Chambers

Ing. Roxana Meriño

Rubén Eduardo Nogales Portero

Ambato, 2000

DEDICATORIA

A mis adorables padres, a mi hermana, y en general a todas aquellas personas que han iluminado mi camino con la luz de DIOS durante el largo transcurrir de mis estudios.

Rubén Nogales

AGRADECIMIENTO

Deseo presentar mi más profundo agradecimiento a la Pontificia Universidad Católica del Ecuador sede Ambato, a sus profesores que han dado todo de si para sembrar en mi conocimientos de bien, y en especial al Ing. Wigberto Sánchez, Ing. Patricio Chambers, y a la Ing. Roxana Meriño.

Rubén Nogales.

INDICE GENERAL

CAPITULO 1.....1

ANÁLISIS DE MICROSOFT ACCESS.....1

1.1 Introducción.....1

1.2 Creación de Proyectos.....2

1.3 Estudio de las bases de datos en Microsoft Access.....7

1.3.1 Creación de bases de datos.....7

1.3.2 Bases de datos relacionales.....10

1.3.2.1 Relación de uno a uno.....11

1.3.2.2 Relación de uno a varios.....12

1.3.2.3 Relación de varios a varios.....12

1.3.3 Lectura de registros.....14

1.3.4 Bases de datos de Microsoft Jet.....18

1.3.5 Ordenar registros.....19

1.3.6 Filtrar información.....19

1.3.7 Agregar registros.....20

1.4 Estudio de los tipos de datos en Microsoft Access.....21

1.4.1 Los datos de tipo texto.....23

1.4.2 Los datos tipo numérico.....24

1.4.3 Los datos de tipo fecha hora.....26

1.4.4 Los datos de tipo Boléanos.....26

1.5 Manejo de cadenas en Microsoft Access.....27

1.6 Multimedia.....30

1.7 Otras especificaciones de Microsoft Access....31
1.7.1 Costos de Microsoft Access....31
1.7.2 Requerimientos de Hardware....31

CAPITULO 2....33
ANÁLISIS DE VISUAL BASIC....33
2.1 Introducción....33
2.2 Creación de proyectos....34
2.3 Estudio de las bases de datos en Visual Basic....37
2.3.1 Características de las bases de datos en Visual Basic....37
2.3.2 Bases de datos relacionales....40
2.3.2.1 Relación uno a uno....40
2.3.2.2 Relación uno a varios....41
2.3.2.3 Relación muchos a uno....41
2.3.3 Visualización de datos....42
2.4 Estudio de los tipos de datos en Visual Basic....46
2.4.1 Datos de tipo entero....49
2.4.2 Datos de tipo texto....50
2.4.3 Datos de tipo moneda....50
2.5 Manejo de cadenas en Visual Basic....51
2.6 Multimedia....53
2.7 Otras especificaciones de Visual Basic....54
2.7.1 Costos de Visual Basic....54
2.7.2 Requerimientos de Hardware....54

CAPITULO 3.....56

ANÁLISIS DE MICROSOFT VISUAL FOXPRO.....56

3.1 Introducción.....56

3.2 Creación de proyectos.....57

3.3 Estudio de las bases de datos en Visual Foxpro.....63

3.3.1 Bases de datos relacionales.....65

3.3.1.1 Relación de uno a uno.....66

3.3.1.2 Relación de uno a varios.....66

3.3.1.3 Relación de muchos a muchos.....67

3.4 Estudio de los tipos de datos en Visual Foxpro.....71

3.4.1 Tipo de datos carácter.....74

3.4.2 Tipo de datos moneda.....74

3.4.3 Tipo de datos fecha.....75

3.4.4 Tipo de datos fecha hora.....76

3.4.5 Tipo de datos doble.....76

3.4.6 Tipos de datos flotante.....76

3.4.7 Tipos de datos lógicos.....77

3.4.8 Tipos de datos numéricos.....77

3.4.9 Tipos de datos entero.....77

3.4.10 Tipos de datos general.....78

3.4.11 Tipos de datos memo.....78

3.5 Manejo de cadenas en Visual FoxPro.....79

3.6 Multimedia.....81

3.7 Otras especificaciones de Visual FoxPro.....83

3.7.1 Costos de Visual FoxPro....83

3.7.2 Requerimientos de Hardware....83

CAPITULO 4....85

ANÁLISIS DE BORLAND DELPHI....85

4.1 Introducción....85

4.2 Creación de proyectos....86

4.2.1 Opciones del proyecto....91

4.2.1.1 Forms....92

4.2.1.2 Application....92

4.2.1.3 Compiler....93

4.2.1.4 Linker....97

4.2.1.5 Directores/Conditional....99

4.3 Estudio de las bases de datos en Delphi....100

4.3.1 Bases de datos locales....102

4.3.2 Servidor de bases de datos remotas....103

4.3.3 Seguridad de bases de datos....104

4.3.4 Transacciones....105

4.3.5 Diccionario de datos....105

4.3.6 Integridad referencial....106

4.3.7 Seguridad de procedimientos....106

4.3.8 Disparadores....106

4.3.9 Archivos índices en Delphi....108

4.4 Estudio de los tipos de datos en Delphi....110

4.4.1 Tipos de datos integer, cardinal....112
4.4.2 Tipos de datos currency....113
4.4.3 Tipos de datos caracter....113
4.4.4 Tipos de datos string....114
4.4.5 Tipos de datos variant....114
4.5 Manejo de cadenas....115
4.6 Multimedia....119
4.7 Otras especificaciones de Borland Delphi....120
4.7.1 Costos de Borland Delphi....120
4.7.2 Requerimientos de hardware....121

CAPITULO 5....122
INGENIERIA DE BASES DE DATOS....122
5.1 Tema I....122
5.1.1 Aspectos introductorios....122
5.1.1.1 Surgimiento histórico de las bases de datos integradas....122
5.1.1.2 Objetivos de los sistemas de bases de datos integradas....126
5.1.1.3 Arquitectura de un sistema de bases de datos integrada....130
5.2 Tema II....138
5.2.1 Modelo conceptual....138
5.2.1.1 Representación de la información....138
5.2.1.2 Características del Modelo conceptual....146
5.2.1.3 Modelo Entidad-Relación....148
5.3 Tema III....162

5.3.1 Modelo Lógico-Global de los datos relacionales....162
5.3.1.1 Modelo Relacional....162
5.3.1.2 Normalización....168
5.3.1.3 Primera Forma Normal....170
5.3.1.4 Segunda Forma Normal....173
5.3.1.5 Tercera Forma Normal....181
5.3.1.6 Forma Normal Boyce-Cod (FNBC)....183
5.3.1.7 Obtención del modelo relacional a partir del DER....194
5.3.1.7.1 Paso #1....196
5.3.1.7.2 Paso #2....197
5.3.1.7.3 Paso #3....198
5.3.1.7.4 Paso #4....198
5.3.1.7.5 Paso #5....199
5.3.1.7.6 Paso #6....199
5.3.1.8 Metodología para el diseño de bases de datos....200

CAPITULO 6....213
RESULTADOS DEL ANALISIS....213
6.1 Facilidad de Manejo....213
6.1.1 Creación de proyectos....214
6.1.2 Manejo y creación de bases de datos....216
6.1.3 Tipos de datos....219
6.1.4 Manejo de cadenas....221
6.1.5 Multimedia....223

Conclusiones y Recomendaciones.....226

Bibliografía.....231

Anexos.....233

INDICE DE GRAFICOS

Figura # 1	Portada de Microsoft Access	2
Figura # 2	Proyecto de Microsoft Access	3
Figura # 3	Cuadro de referencias a proyectos	5
Figura # 4	Creación de Bases de Datos	7
Figura # 5	Tablas en Access	9
Figura # 6	Formularios en Access	9
Figura # 7	Informe o reporte impreso	10
Figura # 8	Base de datos relacional	14
Figura # 9	Diseño de formularios con bases de datos	15
Figura # 10	Pantalla para ordenar y filtrar datos	21
Figura # 11	Portada de Visual Basic	34
Figura # 12	Emplazamiento de Visual Basic para crear proyectos	35
Figura # 13	Estructura de una base de datos en Visual Basic	39
Figura # 14	Portada de Visual FoxPro	57
Figura # 15	Proyecto de Visual FoxPro	58
Figura # 16	Base de Datos con sus respectivas tablas	64
Figura # 17	Estructura de una tabla	65
Figura # 18	Tablas relacionales en Visual FoxPro	66
Figura # 19	Estructura de una cuadricula en Visual FoxPro	71
Figura # 20	Portada de Delphi	86
Figura # 21	Proyecto en Delphi	87
Figura # 22	Forms y Unit en Delphi	88

Figura # 23	Ventana manipuladora de proyectos.....	89
Figura # 24	Opciones de un proyecto.....	91
Figura # 25	Creación de una tabla en Delphi.....	101
Figura # 26	Arquitectura de una base de datos.....	131
Figura # 27	Representación del modelo conceptual.....	147
Figura # 28	Grafico de Modelo Conceptual ó MER.....	150
Figura # 29	Relación por medio de un atributo.....	151
Figura # 30	Diseño Entidad Relación.....	153
Figura # 31	Generalización de conjuntos.....	155
Figura # 32	Especialización de conjuntos.....	156
Figura # 33	Relación de entidades generalizadas y especializadas.....	157
Figura # 34	Representación ternaria de entidades.....	158
Figura # 35	Agregación de entidades.....	159
Figura # 36	Agrupamiento de entidades.....	160
Figura # 37	Representación de un MER en tablas.....	163
Figura # 38	Representación de tablas con datos.....	166
Figura # 39	Representación de las formas normales.....	170
Figura # 40	Primera fase normal.....	171
Figura # 41	Dependencia funcional.....	174
Figura # 42	Dependencia funcional completa.....	174
Figura # 43	Relación excluida de la 2 FN.....	179
Figura # 44	Relación transitiva.....	181
Figura # 45	Relación a partir de atributos no llaves.....	182
Figura # 46	Representación de llaves candidatas.....	187

Figura # 47 Pasos de un procesos de normalización..... 188

Figura # 48 Modelo relacional a partir del DER..... 195

Figura # 49 DER con el ejemplo de automatización de la clinica..... 210

INDICE DE TABLAS

Tabla #1	Tipos de Datos en Microsoft Access.....	22
Tabla #2	Tipos de Datos en Visual Basic.....	47
Tabla #3	Tipos de Datos en Visual FoxPro.....	72
Tabla #4	Tipos de Datos en Borlan Delphi.....	110

PREFACIO

El avance de la tecnología en la época actual es de primordial importancia, por tal motivo es necesario hacer referencia a la informática; ciencia de la cual se sustentan las distintas ramas para su avance tecnológico. En consideración a esto; es necesario tomar en cuenta el avance que a tomado la informática, y más aun las herramientas que han hecho posible dicha evolución.

En consecuencia se torna indispensable hacer un estudio detallado y minucioso de las herramientas de programación visual, por cuanto las mencionadas herramientas son las artífices del desarrollo del mundo actual, debido a que en base de estas herramientas se a creado software en el cual las empresas han sustentado su información.

En el presente estudio se detalla un análisis de cuatro herramientas de programación visual como son: Microsoft Access, Microsoft Visual Basic, Microsoft Visual FoxPro y Borland Delphi, las mencionadas herramientas han sido elegidas puesto que son las más utilizadas en nuestro medio y las de más fácil acceso para las personas que desarrollan software. Dicho análisis se desarrolla siguiendo un esquema, el mismo que facilitará el estudio en tanto en cuanto su estructura permite una comparación directa de los puntos analizados, el esquema seguido es un análisis de creación de Proyectos en cada una de las herramientas, del mismo modo se analizó la forma para la creación de Bases de Datos, Tipos de Datos y manejo de Cadenas.

En el mismo estudio se hace referencia al análisis y diseño de Bases de Datos, puesto que es indispensable para las personas destinadas a desarrollar software que involucra gran cantidad de datos tener una sólida estructura de estas.

Al término de esta investigación se presenta un resumen de las herramientas de programación visual analizadas, detallando estos resultados en gráficos de barras y de acuerdo a una escala de 0 – 5, los distintos puntos tratados y comparados son los más relevantes e importantes para realizar software de apoyo financiero.

Con el estudio realizado se ha podido determinar que el Borland Delphi es la herramienta recomendada para desarrollar software de apoyo financiero, por tal motivo se ha desarrollado una aplicación tomando como base la mencionada herramienta, la aplicación consiste en un almanaque interactivo en el que el usuario podrá consultar temas relacionados con las *Finanzas, Economía y Bolsa,* e incluso podrá adicionar temas a los ya existentes ó prescindir de algunos de estos. Dicho software tiene una visión general de aspectos muy importantes para el desarrollo de cualquier empresa, y en especial de una financiera por cuanto es posible con algunos conceptos aquí detallados tomar decisiones importantes que involucren perdidas o ganancias.

La presente tesis consta de seis capítulos que a continuación se describen:

En el capítulo uno se da a conocer la historia de Microsoft Access, la forma de crear y manejar proyectos, se hace referencia a la forma de crear Bases de Datos, a las Bases de datos relacionales, como poder leer, modificar, ingresar registros en Microsoft Access, también se estudian los Tipos de Datos que esta herramienta ofrece al programador, la forma de manejar cadenas, se detalla también una forma muy elemental de manejo y utilización de multimedia, y por último se detallan otras especificaciones de esta herramienta de programación como son velocidad de procesamiento, costos, disponibilidad de espacio y memoria, y bajo que sistema operativo puede desenvolverse.

El capítulo dos hace referencia a una breve reseña histórica de Microsoft Visual Basic, la forma de crear Proyectos, se refiere también a un estudio de las Bases de Datos, los comandos para navegar en una tabla, los Tipos de Datos que esta herramienta provee al programador, la forma de manejar Cadenas y los comandos destinados a este fin, se da una descripción rápida de los controles y objetos que se utilizan para la creación de software con multimedia, y los requerimientos de Hardware necesarios para que esta plataforma de programación pueda desarrollarse.

El capítulo tres trata del análisis perpetrado para Microsoft Visual FoxPro, al mismo que se le hace una introducción, un estudio acerca de la creación de proyectos, conceptos de la programación orientada a objetos, explica lo referente a Bases de Datos y Bases de Datos relacionales, da un breve bosquejo de integridad referencial, estudia los Tipos de Datos disponibles en esta herramienta, la manipulación de Cadenas, el tratamiento que se le da a la Multimedia y las especificaciones adicionales de hardware.

En el capítulo cuarto se describe al Borland Delphi, su reseña histórica, la forma de crear proyectos y navegación por los mismos; las opciones de las que se dispone para configurar un proyecto, sus Bases de Datos locales y remotas, seguridades en datos, hace referencia a los tipos de datos mejorados y los ya existentes, los comandos y funciones para manipular cadenas, de la misma forma hace referencia a los objetos utilizados para el manejo de la Multimedia, y especificaciones extras como costos y requerimientos de Hardware.

El capitulo cinco hace referencia a la ingeniería de Bases de Datos, en donde que el tema uno se describen a los aspectos introductorios, el tema dos se refiere al modelo conceptualy el tema tres contiene lo concerniente al modelo relacional.

El capítulo sexto es el final y en este se da a conocer los resultados del análisis especificandos los aspectos e items tratados, haciendo referencia al 100%, mostrando los resultados en una escala de 0 – 5.

La presente tesis también consta de un anexo, en el mismo que se explica la navegación en el software desarrollado y explicado anteriormente, adjunto a este anexo se entrega un CD el mismo que contendrá los programas instaladores de la mencionada aplicación.

Este trabajo esta dirigido a todas aquellas personas que estén orientadas a desarrollar software de tipo comercial, puesto que es un guía rápida para poder elegir la herramienta base para desarrollar software, para crear Bases de Datos y Optimizar Tablas. En fin esta tesis se la puede utilizar como una guía rápida de elección de software.

CAPITULO 1

ANÁLISIS DE MICROSOFT ACCESS

1.1 INTRODUCCIÓN.

Access fue creado por la empresa de Software Microsoft como un gestionador de Bases de Datos relacionales, función que permite al programador un conveniente acceso a los datos almacenados en tablas, permitiendo de esta manera reducir notablemente la redundancia de los mismos, para así poder facilitar a los usuarios la búsqueda, flexibilidad en su uso, y estandarización de estos.

Microsoft al transcurrir de los tiempos fue creando versiones mejoradas de Access, para de esta forma estar acorde con los avances tecnológicos de la época, por tal motivo Microsoft Access 97 en la actualidad contiene un asistente para desarrollar paginas WEB y realizar vínculos con archivos .HTML de internet, de la misma forma Microsoft permite que Access pueda trabajar en forma conjunta con otras aplicaciones, caso concreto del Visual Basic, Excel, Word, y en general con todos los productos de Microsoft Office 97.

Pudiendo de esta manera hablar de una programación orientada a objetos, por su vinculación directa con el Visual Basic.



Fig. #1: Portada de MICROSOFT ACCESS 97

1.2 CREACIÓN DE PROYECTOS.

Microsoft Access es un manipulador de Bases de Datos muy potente, pero que lastimosamente no puede actuar de una forma independiente, es decir que éste depende de otros lenguajes de programación. Para el caso concreto de la creación de proyectos, este manipulador depende de una forma directa del Visual Basic.

Cada Base de Datos edificada en Microsoft Access incluye un proyecto Visual Basic, el mismo que va ha contener todas sus bases de datos; formularios, módulos, consultas, informes, macros que abarca el proyecto, incluyendo estos sus módulos estándar y módulos clase. Todo esto estará contenido en un archivo *.mdb que es un proyecto Visual Basic, como se lo puede observar en la Figura #2.

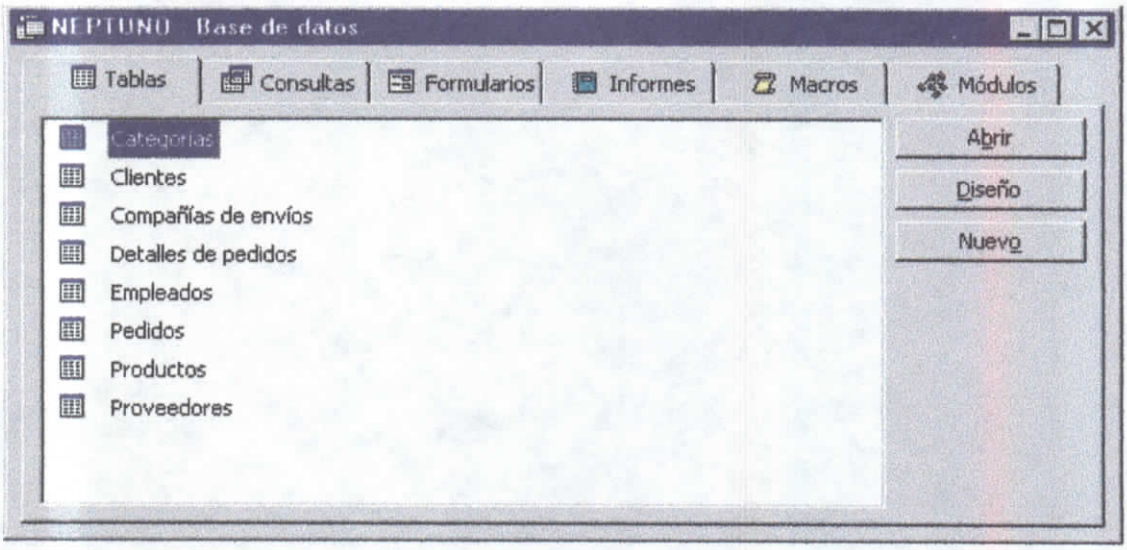


Fig. #2: Pantalla de Microsoft Access por medio de la cual se pueden manipular un proyecto.

Los nombres que se les atribuyen a las Bases de Datos y a los proyectos pueden ser tomados indistintamente, los nombres de las Bases de Datos podrán ser reconocidos por los archivos con extensión *.mdb ó .mda. ó .mde*, mientras que el nombre del proyecto se le asignará en el momento de grabar, esto se lo puede realizar por medio del menú Herramientas, o en la caja de diálogo.

Al primer instante de su creación; por defecto el nombre de la Base de Datos y del proyecto serán los mismos, pero si se cambian el nombre de la Base de Datos; el nombre del proyecto no se cambia inmediatamente, es decir que el nombre del proyecto no tiene efecto sobre el nombre de la Base de Datos.

Microsoft Access dentro de la construcción de proyectos brinda al programador dos facetas; una en tiempo de diseño, y otra en tiempo de ejecución.

En el tiempo de diseño el programador puede modelar las formas de acuerdo a sus necesidades tomando en cuenta que la forma ya en tiempo de ejecución es la parte que visualizara el usuario, o la parte con la que interactuara el usuario de dicha aplicación. El programador para el diseño de las formas dispone de herramientas que Microsoft Access ofrece, y estas se detallan a continuación:.

Etiquetas, cuadros de texto, grupos de opciones, botones de alternar, botones de opción, botones de verificación, cuadros combinados, cuadros de lista, botones, campos de imágenes, controles de fichas, sub formularios, líneas.

En tiempo de diseño estas herramientas serán utilizadas para crear la interfaz, la misma que en tiempo de ejecución se podrán utilizar para poder realizar las tareas asignadas, las funciones programadas para todas y cada una de las herramientas utilizadas para crear la interfaz.

Microsoft Access también pone a disposición de sus usuarios la posibilidad de poder establecer referencias desde proyectos de Visual Basic de una Base de Datos de Microsoft Access a un proyecto en otra Base de Datos de Microsoft Access; a una Base de Bibliotecas; o en su ves a una base de complementos contenida en un archivo con extensión *.mde*. Una vez que se hayan establecido las referencias entre los proyectos, se podrá hacer llamados o referirse a procedimientos o funciones contenidos en el proyecto sobre el cual se ha hecho la referencia como si se estuviese tratando con procedimiento o funciones del mismo proyecto, cave aclarar que si se utilizan Bases de Datos creadas en versiones anteriores; para tener una referencia en primer lugar se las debe convertir a formato de Microsoft Access 97 (versión 8), aquí es posible que se necesite hacer algunos cambios en cuanto a objetos, procedimientos y macros se refiere;

para tener un máximo rendimiento de las referencias, que se vayan hacer en esta versión con versiones anteriores.

Para poder establecer una referencia de un proyecto diseñado en Microsoft Access a otra Base de Datos, se lo podrá llevar a cabo por medio del menú de herramientas; haciendo un clic en referencias, y posteriormente eligiendo la Base de Datos sobre la cual se quiere hacer referencia, previa apertura de la ventana de dialogo modulo, como se muestra en la Fig. 3.

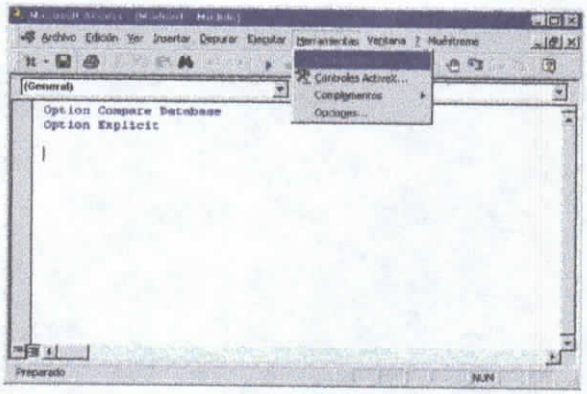


Fig. #3: Ventana de diálogo para establecer una referencia a otra BASE DE DATOS DE MICROSOFT ACCESS

Cuando desee llamar un procedimiento público que está definido dentro de un módulo estándar en esa Base de Datos, se lo puede hacer siempre y cuando no este declarado como privado, entendiéndose que módulo estándar es en donde se pueden editar SUB y FUNCTION que se los desee utilizar dentro de otras Bases de Datos.

No se pueden llamar a procedimientos que estén definidos dentro de un módulo de clase o procedimiento de un módulo estándar que esté precedido por la palabra clave *private*.

Un módulo clase, es un módulo que pude tener una definición para un nuevo objeto, cuando crea una nueva instancia de una clase, crea un nuevo objeto, algunos procedimientos definidos en el módulo, se convierten en propiedades y métodos del objeto, los módulos clase, pueden existir independientemente o en asociación con formularios e informes.

Solo se podrá establecer una *referencia al proyecto* en una Base de Datos de Microsoft Access desde otra Base de Datos de Microsoft Access, si se desea establecer una referencia, a otra Base de Datos con un formato distinto al que Microsoft Access proporciona, Microsoft Access no lo aceptará.

También se puede establecer una referencia desde Bases de Datos de Microsoft Access 97 a otra Base de Datos de Microsoft Access 97, solo bajo las condiciones que deben ser del mismo formato, o en caso contrario realizar la transferencia de formato, como ya se indicó anteriormente, y su clave debe necesariamente estar escrita en Visual Basic.

Si se establece una referencia a un proyecto o biblioteca de tipos de Microsoft Access y luego mueve el archivo que contiene ese proyecto o biblioteca de tipos a una carpeta distinta, Microsoft Access intentará ubicar y restablecer las referencias (si la clave *RefLibPaths* existe en el registro de Microsoft Access), buscando primero en la carpeta de origen, y si no encuentra una entrada que coincida. Microsoft Access buscara el archivo primero en la carpeta actual y luego en todas las carpetas de la unidad. En caso de que no existiera la clave *RefLibPaths* se la pudiera crear usando el editor de registros en windows, bajo la clave del registro.

1.3 ESTUDIO DE LAS BASES DE DATOS EN MICROSOFT ACCESS.

1.3.1 CREACIÓN DE BASES DE DATOS.

Las Bases de Datos en Microsoft Access se han constituido en una herramienta indispensable para el progreso de las empresas a todo nivel y condición, básicamente por la interfaz grafica que presenta, como se lo puede observar en la Fig. 4.

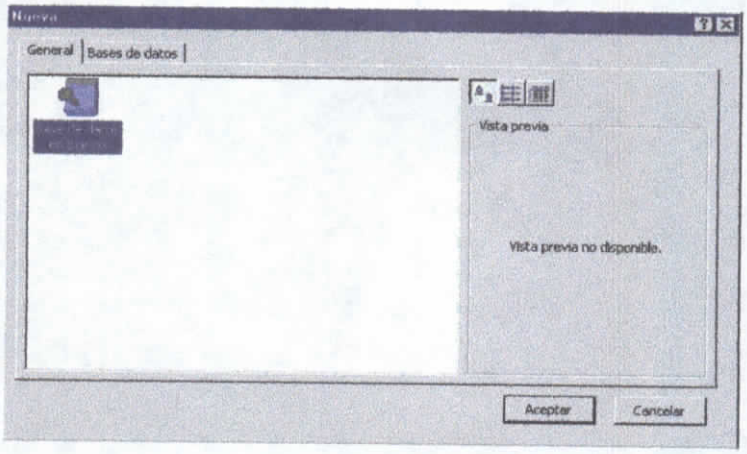


Fig. #4: Cuadro de dialogo de Microsoft Access para poder crear una Base de Datos.

Del mismo modo en Microsoft Access encontraremos una gran facilidad para el manejo de Bases de Datos, especialmente con el manipulador de asistentes, esto debido a que Microsoft Access forma parte de la gran familia del Microsoft Office, y precisamente a Microsoft Access se le a diseñado para realizar cualquier tipo de tareas con sus asistentes.

Microsoft Access brinda una gran facilidad y seguridad para la manipulación de información que se registrarán en las Bases de Datos, tanto así que puede guardar datos relacionados con un tema

en una sola ubicación, por lo que facilita su búsqueda, y por ende con estos datos resumir considerablemente el tiempo de la presentación de dicha información.

Además de todo esto Access también facilita la actualización de los datos contenidos en las tablas, siempre que estas estén relacionadas, ya que tan solo haría falta cambiar la información una sola vez, y Microsoft Access lo actualizara en todos los lugares en donde aparezca dicha información.

Las Bases de Datos de Microsoft Access brinda también la oportunidad de trabajar en Red, compartiendo una base de datos completamente; si dicha Base está ubicada en un servidor de red o si la Bases de Datos está ubicada dentro de una carpeta compartida, en este caso los usuarios podrán utilizar los mismo informes, reportes, formularios, etc. que forman la Base de Datos, en otro caso Access también brinda la oportunidad de compartir solo tablas ubicadas en el servidor de red, y los formulario, informes, reportes, etc. se las encontrará en las estaciones de trabajo, de esta forma lo único que viaja por la red son datos.

Dentro de la construcción de bases de datos, Microsoft Access brinda a los usuario las facilidades que se describen a continuación:

a) **Crear una tabla**.- Una tabla es creada con la finalidad de guardar información referente a un mismo tema en un lugar específico, tomando en cuenta que una tabla, es un lugar físico del disco, dividida en campos y registros, tal como se muestra a continuación en la fig. 5.

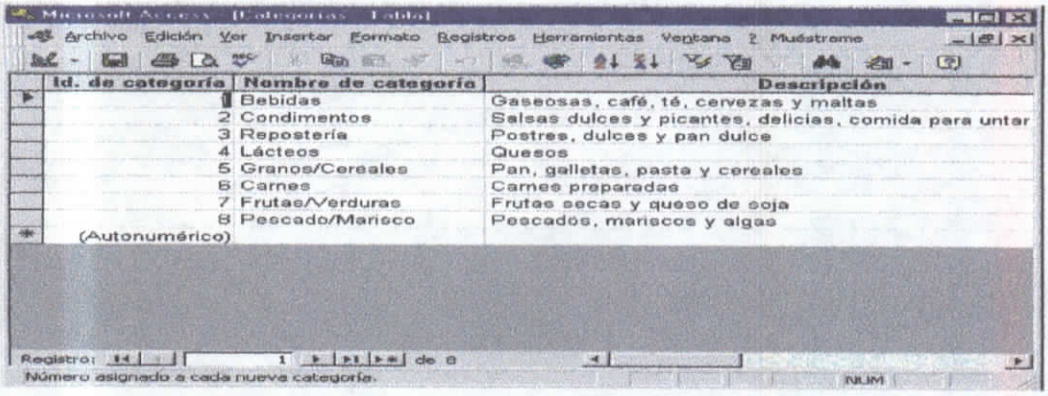


Fig. #5: Descripción de una tabla en Microsoft Access con sus respectivos campos.

b) Crear un Formulario.- Un formulario es una forma sencilla y fácil de accesar y revisar la información contenida dentro de un tabla o consulta, un formulario también es interpretada como la interfaz de visualización de los datos en forma de texto o gráfica, según conveniencia de los usuarios, tal como se puede observar en la Fig. 6.

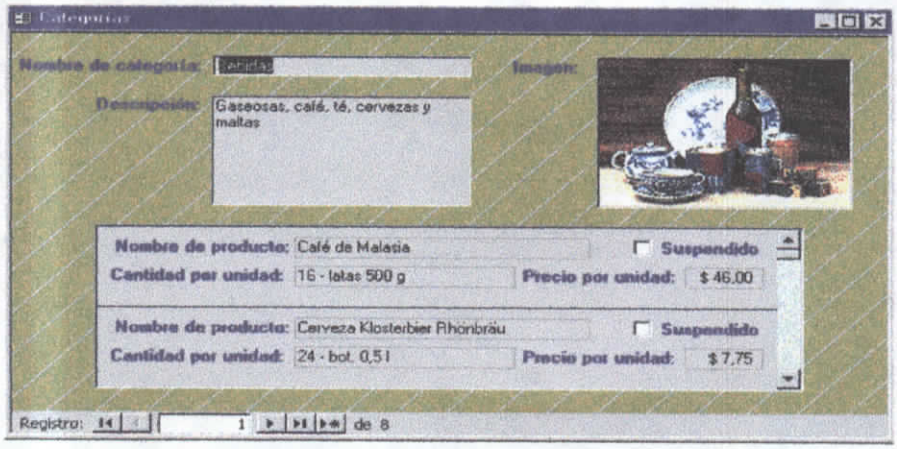


Fig. #6: Formulario de Microsoft Access visualizado mostrando datos contenidos en una tabla en tiempo de ejecución.

c) Crear un informe.- Un informe en Microsoft Access es una forma de presentación de datos ordenados convenientemente y según requerimientos del usuario, estos datos

visualizados en el informe pueden estar contenidos en una o más tablas, tal como se muestran en la Fig. 7.



Fig. #7: Informe Impreso con datos extraídos de diferentes tablas.

1.3.2 BASES DE DATOS RELACIONALES.

Antes de explicar lo que es un Base de Datos relacional, debemos comprender que una Base de Datos tendrá información en diferentes tablas, y esta información será referente a un mismo tema, por tanto la relación se refiere a que se puede usar al mismo tiempo, datos que provienen de varias tablas.

El potencial de un sistema de base de datos relacional como Microsoft Access reside en su capacidad para encontrar y reunir rápidamente información almacenada en tablas independientes mediante consultas, formularios e informes. Para ello, cada tabla debe incluir un campo o conjunto de campos que identifique exclusivamente un registro almacenado en la tabla. Este campo o conjunto de campos se denominan clave principal de la tabla. Una vez designada una clave principal de una tabla, con el fin de garantizar su exclusividad, Microsoft Access impide

que se almacenen valores duplicados o Nulos en los campos de la clave principal, almacenar datos en una tabla relacional resulta muy útil debido a que cada dato se almacena una sola vez, reduciendo las necesidades de almacenamiento en el disco.

Para establecer relaciones entre tablas similares a las que se muestran en la Fig. 8, debemos tener una Tabla principal; que es la que va a contener la clave principal, es decir la que tiene el campo clave, y una o más tablas relacionales, que son las que tienen los campos coincidentes.

Esto nos garantiza tener una integridad referencial, es decir que los cambios que se le realicen a una tabla referencial no vayan a afectar a la relación, esto impide añadir un registro, modificar un registro o eliminar un registro de la tabla referencial, si el valor del campo de relación no existe en la tabla principal.

La integridad referencial se conseguirá solamente si el campo coincidente de la tabla esclava es clave principal de la tabla maestra o el campo índice, si la tabla maestra se encuentra indexada, es decir ordenada por medio de este campo; y además los campos relacionales tienen el mismo tipo de datos, en este punto Microsoft Access brinda la oportunidad de realizar diversos tipos de relaciones entre tablas, como se describen a continuación.

1.3.2.1 Relación de uno a uno.

Consiste en tener un registro de la tabla maestra (principal) relacionada tan solo a un registro de la tabla esclava (relacional) y viceversa. Este tipo de relación no es habitual, debido a que la mayoría de información relacional estaría en una sola tabla, esta

relación se la podría utilizar para dividir una tabla con muchos campos, y por cuestiones de seguridad.

1.3.2.2 Relación de uno a varios.

Se da en el momento que el registro que contiene la clave principal de la tabla maestra está una sola vez, y los registros coincidentes que se encuentran en la tabla relacionada pueden repetirse un incontable número de veces, sin embargo este tipo de relación no puede darse a la inversa. Este tipo de relación es el más común.

1.3.2.3 Relación de varios a varios.

Para esta relación tomemos como ejemplo dos tablas A y B, la tabla A puede tener muchos registros coincidentes en la tabla B, y viceversa, este tipo de relación solo es posible llevarla a cabo si se define una tercera tabla (tabla de unión), cuya clave principal conste de al menos dos campos que serán las claves coincidentes de las tablas A y B.

Para tener un concepto más claro de los tipos de relaciones que se acaban de explicar es necesario hacer referencia al concepto de clave principal, con este fin se pueden citar a tres tipos de claves las mismas que se detallan a continuación:

a) **Claves principales de Auto numérico.-** Un campo Auto numérico puede establecerse para que introduzca automáticamente un número secuencial cuando se agrega un

registro a la tabla. Designar un campo de este tipo como clave principal de una tabla es la forma más sencilla de crear una clave principal. Si no establece una clave principal antes de guardar una tabla recién creada, Microsoft Access le preguntará si desea que cree una clave principal automáticamente. Si contesta afirmativamente, Microsoft Access creará una clave principal de tipo Auto numérico. Existen otras consideraciones adicionales que deben tenerse en cuenta si la tabla se va a utilizar con una réplica de base de datos. Debe tener en cuenta el tamaño del campo.

b) **Claves principales de Campo simple.-** Si tiene un campo que contiene valores exclusivos, como números de identificación, puede designar ese campo como la clave principal. Si el campo seleccionado como clave principal tiene valores duplicados o Nulos, Microsoft Access no establece la clave principal. Puede ejecutar una Consulta de buscar duplicados con el fin de determinar qué registros contienen datos duplicados. Si no puede eliminar fácilmente las entradas duplicadas mediante la edición de los datos, puede agregar un campo Auto numérico y establecerlo como clave principal o bien definir una clave principal de campos múltiples.

c) **Claves principales de Campos múltiples.-** En situaciones en las que no se puede garantizar la exclusividad de un solo campo, puede designar dos o más campos como clave principal. La situación más común en la que surge este problema es en la tabla utilizada para relacionar otras dos tablas en una relación varios a varios.

Con la relación de tablas podemos hacer: una eliminación, modificación, etc. de registros en cascada. Es decir si modificamos o eliminamos un registro en la tabla que contenga la clave

principal, los registros coincidentes de las tablas esclavas por ende se eliminaran o modificaran respectivamente.

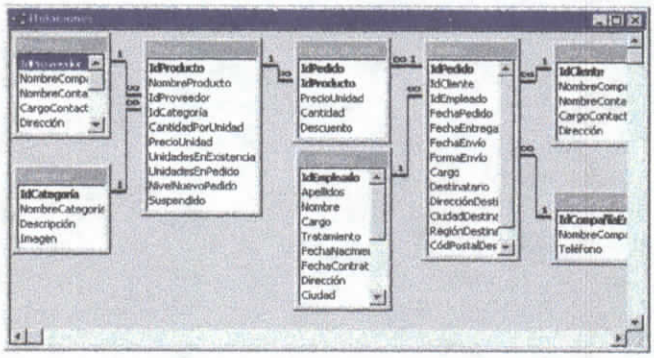


Fig. #8: Estructura de una base de datos mostrando sus Tablas con sus respectivas Relaciones.

1.3.3 LECTURA DE REGISTROS.

Debido a que la información esta distribuida en la tabla, y al tratar de ver solamente la información específica de un registro puede resultar difícil, al darse cuenta de esto Microsoft Access pone a disposición de los usuarios la facilidad de crear formularios, en los mismos que se puede presentar la información en forma sucesiva, es decir la información será mostrada de acuerdo a como el usuario haya definido el formulario destinado a mostrar los datos existentes en los registros de la Tabla como se muestra en la Fig. 9. Debido a que el formulario de Microsoft Access trabaja en forma directa con la tabla, la información es reflejada de forma inmediata, casi instantánea, también se tiene la facilidad de mostrar datos por medio de una consulta, en la misma que se podrá mostrar todo tipo de datos, incluyendo la posibilidad de mostrar datos calculados.

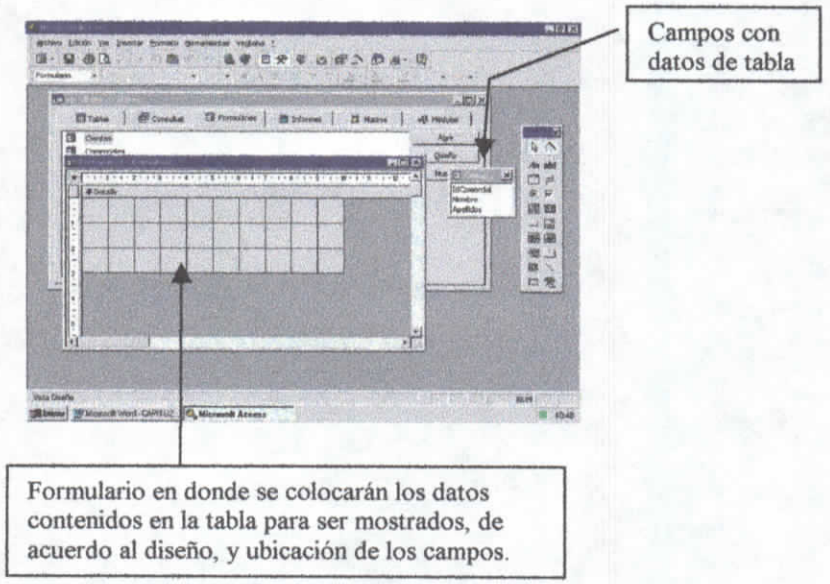


Fig. #9: Pantalla de Microsoft Access por medio de la cual se diseñaran formularios para poder mostrar datos contenidos en las tablas

La rapidez con que son mostrados los datos radica en los archivos Índices, los mismos que especifican el orden de los registros a los que se accede desde las tablas de la base de datos, y determinan si se aceptan o no registros duplicados, permitiendo un acceso eficiente a los datos. En las bases de datos externas, los archivos Índices describen los índices establecidos para tablas externas (sólo espacios de trabajo Microsoft Jet).

El motor de base de datos Microsoft Jet utiliza índices cuando se une con tablas y crea objetos *Recordset*. Los índices determinan el orden en que los objetos *Recordset* de tipo *Table* devuelven registros, pero no determinan el orden en el que el motor de base de datos Microsoft Jet los almacena en la tabla base o el orden en que cualquier otro tipo de objeto *Recordset* los devuelve.

En Microsoft Access si el programador utiliza un objeto Index, puede:

- Utilizar la propiedad *Required* para determinar si los objetos *Field* del índice necesitan valores *no Null* y utilizar la propiedad *IgnoreNulls* para determinar si los valores *Null* tienen entradas de índices.

- Utilizar las propiedades *Primary y Unique* para determinar el orden y exclusividad del objeto *Index*.

El motor de base de datos Microsoft Jet mantiene todos los índices de la tabla base automáticamente. Los índices se actualizan siempre que se añadan, cambien, o eliminen registros de la tabla base. Una vez que se ha creado la base de datos, se puede utilizar el método *CompactDatabase* periódicamente para actualizar las estadísticas del índice.

Al acceder a un objeto *Recordset* de tipo tabla, se puede determinar el orden de los registros mediante la propiedad *Index*. Se puede establecer esta propiedad para el valor de la propiedad *Name* de un objeto *Index* existente en la colección *Indexes*. Esta colección está contenida en el objeto *TableDef*, que subyace al objeto *Recordset* que está rellenando.

No es necesario crear índices para una tabla, sin embargo en las tablas grandes y sin índice el acceso a un registro específico o el proceso de uniones puede tardar mucho tiempo. Por el contrario, si se crean demasiados índices, se puede hacer más lenta la actualización de la base de datos, puesto que Microsoft Access modifica todos los índices relacionados con la tabla.

La propiedad *Attributes* de cada objeto *Field* del índice establece el orden de los registros y por ello determina las técnicas de acceso que se deben utilizar para ese índice.

Cada objeto *Field* de la colección *Fields* de un objeto *Index* es un componente del índice. Para definir un nuevo objeto *Index* es necesario establecer las propiedades antes de añadirlo a una colección, lo que permite que el objeto *Index* esté disponible para su uso posterior.

Puede modificar el valor de la propiedad *Name* de un objeto existente *Index* sólo si la propiedad *Updatable* del objeto *TableDef* que la contiene es *True*.

Cuando establezca una clave principal para una tabla, el motor de base de datos Microsoft Jet la definirá como el índice principal. Un índice principal consta de uno o más campos que identifican exclusivamente todos los registros de una tabla en un orden predefinido. Como el índice principal del campo debe ser único, el motor de base de datos Microsoft Jet establece automáticamente como *True* la propiedad *Unique* del objeto *Index*. Si el índice principal consta de más de un campo, cada campo puede contener valores duplicados, pero la combinación de valores de todos los campos indexados debe ser única. Un índice principal consta de una clave para la tabla y siempre está formado de los mismos campos que la clave principal.

Compruebe que los datos coincidan con los atributos del nuevo índice. Si el índice requiere valores exclusivos, compruebe que no existan duplicados en los registros de datos existentes. Si existen duplicados, el motor de base de datos Microsoft Jet no podrá crear el índice y se producirá un error cuando intente utilizar el método *Append* con el nuevo índice.

Cuando cree una relación que exija la integridad referencial, el motor de base de datos Microsoft Jet creará automáticamente un índice con la propiedad *Foreign*, establecida como clave externa de la tabla de referencia. Una vez establecida una relación de tabla, el motor de base de datos

Microsoft Jet evita adiciones o cambios en la base de datos que violen esa relación. Si establece la propiedad *Attributes* del objeto *Relation* para permitir operaciones de actualización en cascada y eliminación en cascada, el motor de base de datos Microsoft Jet actualiza o elimina automáticamente los registros de las tablas relacionadas.

Para crear un nuevo objeto Index se hace necesario establecer la siguiente secuencia:

1. Utilice el método *CreateIndex* con el objeto *TableDef.*
2. Utilice el método *CreateField* con el objeto Index para crear un objeto *Field* para cada campo (columna) que se vaya a incluir en el objeto Index.
3. Seleccione las propiedades *Index* según sea necesario.
4. Añada el objeto *Field* a la colección *Fields.*
5. Añada el objeto *Index* a la colección *Indexes.*

Nota La propiedad *Clustered* se pasa por alto para las bases de datos que utilizan el motor de base de datos Microsoft Jet, ya que no admite índices agrupados.

1.3.4 BASES DE DATOS MICROSOFT JET.

Microsoft Jet es un sistema de administración de Bases de Datos que recupera y almacena datos en tablas de la Base de Datos respectivamente. El motor Base de Datos de Microsoft jet se puede ver como un componente administrador de datos con el que se crean otro sistema de acceso a datos, como MICROSOFT ACCESS Y VISUAL BASIC.

1.3.5 ORDENAR REGISTROS.

Para ordenar registros en una tabla de Microsoft Access, resulta muy sencillo, simplemente debemos marcar el campo por el cual queremos ordenar, y posteriormente se da un clic en el icono prediseñado por Microsoft Access tal como se puede observar en la Fig. 10 para ordenar en forma ascendente o descendente según su conveniencia, Microsoft Access permite también ordenar por más de un campo, sencillamente seleccionando los campos por los cuales se desee ordenar, esto se lo hace con la tecla SHIFT y CLIC sobre los campos, y se proceden con los mismos pasos anteriores, Microsoft Access realiza este tipo de ordenamientos con una velocidad impresionante, también da la facilidad para que el usuario pueda elegir y crear una tabla dinámica, que no es otra cosa que la creación de una hoja de cálculo en Excel, en donde se puede realizar este tipo de ordenamientos con mucha facilidad y rapidez.

Una forma de tener la tabla ordenada permanentemente se lo puede conseguir mediante la creación de un archivo índice, que es un archivo que está incluido dentro del mismo archivo con extensión *.mdb* seleccionando un campo por medio del cual será el indexado.

1.3.6 FILTRAR INFORMACIÓN.

Microsoft Access brinda la facilidad de mostrar información, solamente de los datos que se requieran, en una consulta, en una tabla, en un formulario, en un informe, o en la utilización de una cláusula SQL utilizando los filtros de datos, para realizar un filtro seleccionamos un registro con la información por la que deseamos filtrar, y posteriormente dando un clic en el botón prediseñado por Microsoft Access (Filtros por Selección), y de la misma forma si se desea quitar

el filtro, nuevamente hacemos un clic en el botón prediseñado para filtros, como se presenta en la Fig. 10.

Un filtro de datos no es más que ordenar y limitar registros de una Tabla, puesto que Microsoft Access puede alternar sus ventanas, para modificar o aplicar filtros, estos se los puede hacer de una forma sencilla y rápida, para poder mostrar los datos seleccionados de acuerdo al filtro aplicado.

1.3.7 AGREGAR REGISTROS.

El formulario de Microsoft Access da la facilidad de agregar nuevos registros, y observar de una forma clara los campos que van ha ser ingresados, para esto tan solo se debe dar un clic en * (asterisco) ubicado en la parte inferior del formulario (Ver Fig. 10) De la misma forma se puede deshacer los cambios hechos en los campos o simplemente se puede eliminar registros, Microsoft Access también brinda la facilidad de poder trabajar directamente en la Tabla, es decir poder ingresar nuevos registros, eliminar registros, siempre y cuando estos cumpla con las reglas establecidas por la integridad referencial, si la tiene al momento de ingresar datos; los campos que no tienen valor alguno se llenaran por el valor predeterminado definido en el momento en que se estructuró la tabla, y si no existen campos con valores predeterminados estos se pondrán con valores nulos, adicionalmente a esto se debe aplicar la instrucción *UpDate* con el propósito de que los cambios realizados se queden registrados en la tabla.

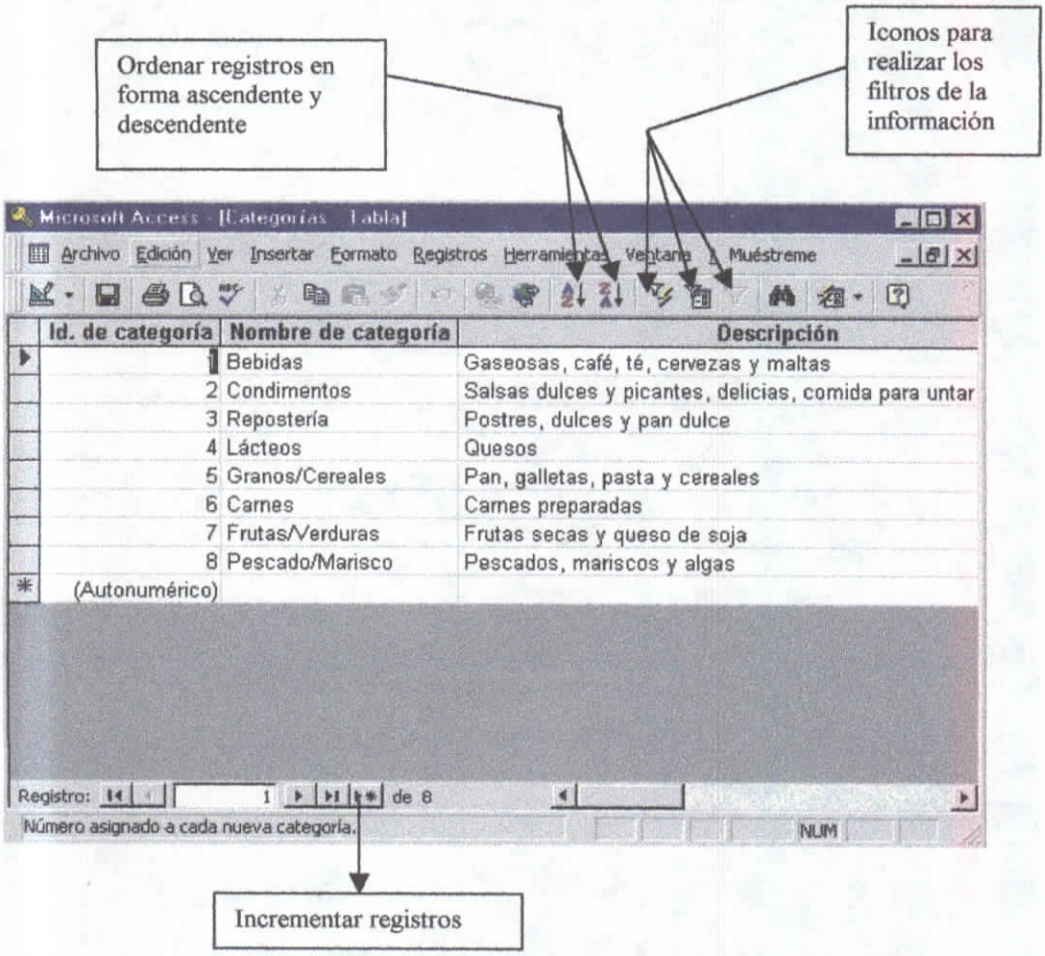


Fig. #10: pantalla por medio de la cual se puede ordenar, filtrar o agregar registros.

1.4 ESTUDIO DE LOS TIPOS DE DATOS EN MICROSOFT ACCESS.

Microsoft Access tiene distintos tipos de datos, los mismos que le sirven para dar propiedades a los campos de una tabla, las cuales permiten especificar el tamaño del campo y el tipo del campo.

En cuanto a la propiedad tamaño del campo Microsoft Access permite especificar la longitud que tendrá el campo al cual se le esta asignando dicha propiedad, la misma que será acorde con la longitud del dato que será introducido en la tabla, en Microsoft Access esta propiedad se le

puede asignar a los datos tipo texto y numéricos, en cambio que los otros tipos de datos pueden tomar esta propiedad por defecto.

Microsoft Access para facilitar aun más el trabajo de los programadores, a creado diferentes tipos de datos que en la tabla 2.1. se detallan sus características y cualidades específicas.

Tabla #1: Tipos de datos que Microsoft Access ofrece a los programadores.

Nombre de Tipo de Dato	Descripción	Rango de Valores	Espacio requerido
Entero	Numérico Entero, no presenta decimales	-32768 a 32767	2 Bytes
Entero Largo	Numérico Entero, no presenta decimales	-2147483648 a 2147783647	4 Bytes
Simple	Numérico, siete posiciones decimales	$-3.4*10^{38}$ a $3.4*10^{38}$	4 Bytes
Doble	Numérico, quince posiciones decimales	$-1.8*10^{308}$ a $1.8*10^{308}$	8 Bytes
Byte	Numérico, no pre. decimales	0 a 255	1 Byte
Autonumérico	Coge un Valor aletoreo de access	Depende del valor	De 4 a 8 Bytes

Texto	Textos y números que no realicen cálculos	Hasta 255 caracteres	Un bit por carácter
Memo	Texto extenso	Limitado por la base de datos	Iimitado
Fecha	Tipo Fecha		8 Bytes
Boléanos	Datos de si o no	TRUE o FALSE	2 Bytes

1.4.1 LOS DATOS DE TIPO TEXTO.

A este tipo de datos, se les puede delimitar con un máximo y un mínimo de caracteres. Microsoft Access por defecto a este tipo de datos les asignara un tamaño de 50 caracteres, pudiendo asignar nuevos valores para estos campos, teniendo un rango que puede variar de 1 a 255 caracteres.

a) **Texto**.- (Predeterminado) Texto o combinaciones de texto y números, así como números que no requieran cálculos, como los números de teléfono. Hasta 255 caracteres o la longitud que indique la propiedad Tamaño del campo (el menor de los dos valores). Microsoft Access no reserva espacio para la parte que no se utiliza de un campo de texto.

b) **Memo**.- Texto extenso, o combinación extensa de texto y números. Hasta 65.535 caracteres. (Si el campo se manipula a través de DAO y sólo se va

a almacenar en él texto y números [no datos binarios], el tamaño está limitado por el de la base de datos).

1.4.2 LOS DATOS DE TIPO NUMÉRICO.

Este tipo de datos determina cual será el rango de números que puede introducirse en un campo, este tipo de datos también puede especificar el número de caracteres decimales, que se pueden asignar al campo.

El rango de valores que se le pueden introducir en este tipo de datos son:

a) Byte.- - Valores entre 0 y 255
- No presentan decimales
- Ocupan un Byte

b) Entero.- - Valores entre -32768 y 32767
- No presente decimales
- Ocupa dos Bytes

c) Entero Largo.- Valores entre -2.147.483.648 y 2.147.783.647
- No presenta decimales
- Ocupa cuatro Bytes

d) Simple.- - Valores entre $-3.4*10^{38}$ y $3.4*10^{38}$

- Presenta siete posiciones de decimales
- Ocupa cuatro Bytes

e) Doble.-
- Valores entre $-1.8*10^{308}$ y $1.8*10^{308}$
- Presenta 15 posiciones de decimales
- Ocupa ocho Bytes

f) Autonumérico.- de 4 bytes (16 bytes si el valor es de tipo Id. Réplica)
- Coge un valor aleatorio de Access
- No se puede repetir este valor.

Microsoft Access permite darle formato a los datos de tipo Numérico/Moneda. Los formatos que Microsoft Access presenta para los datos de tipo Numérico y los datos de tipo moneda son los siguientes:

Número General.
- Moneda
- Fijo
- Estandar
- Porcentaje
- Científico

1.4.3 LOS DATOS DE TIPO FECHA HORA.

Microsoft Access también brinda la posibilidad de tener campos con formato tipo fecha hora, este tipo de datos puede tener los siguientes formatos:

Fecha General	(15/6/99 13.10.30)
Fecha Larga	(martes 15 de junio de 1999)
Fecha Mediana	(15-jun-99)
Fecha Corta	(15/6/99)
Hora Larga	(13.10.30)
Hora Mediana	(1.10)
Hora Corta	(13.10)

1.4.4 LOS DATOS DE TIPO BOOLEANOS.

Por ultimo Microsoft Access presenta un formato para los datos de tipo *booleanos*, es decir datos de tipo Si o No. Este formato por defecto puede ser Si o No, pero también existe la posibilidad de tener otros formatos como:

Verdadero/Falso

Activado/Desactivado

1.5 MANEJO DE CADENAS EN MICROSOFT ACCESS

Microsoft Access pone a disposición de los usuarios una gran variedad de herramientas para el manejo de cadenas con el propósito de tener un código de programación sin ambigüedades, tal es el caso del operador +, que en cierto momento puede prestarse para confusiones, puesto que Microsoft Access puede interpretar a este operador como si deseara realizar una suma de valores numéricos o como si deseara concatenar caracteres de tipo alfanuméricos, para evitar este tipo de ambigüedad, Microsoft Access optó por anteponer el símbolo *amperstand* (&) para concatenar cadenas de caracteres de tipo alfanuméricos.

Microsoft Access para facilitar a los usuarios su utilización pone a disposición de estos los siguientes comandos para el manejo de cadenas.

- El comando *Cbool* es utilizado para convertir una expresión a un tipo de datos bolean, es decir a un tipo de datos que sean evaluados como Verdadero o Falso.
- *Cbyte* es un comando que Microsoft Access provee al programador para convertir un tipo de datos (numérico) a datos de tipo *Byte*, es decir a un número que este comprendido entre 0 y 255.
- El comando *Ccur* es utilizado por Microsoft Access para convertir un tipo de datos numérico en datos de tipo *Currency* o datos de tipo moneda; con valores comprendidos entre:

  922.337.203.685.477,5808 a 922.337.203.685.477,5807.
- El comando *Cdate* es una expresión utilizada para convertir una cadena en datos de tipo *Date*, siempre y cuando se mantenga una relación y coherencia en la cadena a ser convertida.

- El comando *CDbl* es una expresión utilizada para convertir un tipo de datos numérico en un tipo de datos *Double* con expresiones numéricas comprendidas entre: $-1.8*10^{308}$ y $1.8*10^{308}$.
- *Cint* es un comando utilizado para convertir un valor de tipo numérico, cualesquiera sea este a datos de tipo entero que contengan valores comprendidos entre: -32.768 y 32.767; cave destacar que en este tipo de datos las fracciones se redondean.
- *CLng* es un comando utilizado para convertir un valor de tipo numérico, cualesquiera sea este a datos de tipo entero que contengan valores comprendidos entre: -2.147.483.648 y 2.147.783.647.
- El comando *CSng* es una expresión utilizada para convertir datos de tipo numérico con decimales a datos de tipo *Single*, Cave destacar que los datos de tipo *Single* acepta valores comprendidos entre $-3.4*10^{38}$ y $3.4*10^{38}$.
- *Cvar* es un comando que convierte una expresión a datos de tipo *Variant*. El mismo que contendrá intervalos iguales a los intervalos que los datos de tipo *Double* para valores numéricos. El mismo intervalo que los datos de tipo *String* para valores no numéricos.
- El comando *CStr* es una expresión que convierte datos de tipo numérico en datos de tipo *String* o cadena.

En general el código se puede documentar utilizando las funciones de conversión de tipos de datos para indicar que el resultado de alguna de las operaciones se debería expresar como un tipo de datos en particular, no como el tipo de datos predeterminado. Por ejemplo, utilice *CCur* para forzar la ejecución de operaciones con aritmética monetaria en los casos en los que se haría con precisión simple, doble precisión o aritmética entera.

Debe utilizar funciones de conversión de tipos en lugar de *Val* para proporcionar conversiones que reconozcan las variantes internacionales. Por ejemplo, cuando utiliza *CCur*, se reconocen diferentes separadores decimales, diferentes separadores de millares y varias opciones monetarias dependiendo de la configuración regional establecida en su equipo.

Cuando la parte fraccionaria es exactamente 0,5, *CInt y CLng* siempre redondean al número par más cercano. Por ejemplo, 0,5 redondea a 0, y 1,5 redondea a 2. *CInt y CLng* se diferencian de las funciones *Fix e Int* en que truncan, en lugar de redondear, la parte fraccionaria de un número. Además, *Fix e Int* siempre devuelven un valor del mismo tipo del que se le pasa.

Utilice la función *IsDate* para determinar si se puede convertir *Date* a una fecha o una hora. *CDate* reconoce literales de fecha y literales de hora además de números comprendidos dentro del intervalo de fechas aceptables. Al convertir un número a una fecha la parte numérica entera se convierte a una fecha. Cualquier parte fraccionaria del número se convierte a la hora del día, comenzando a medianoche.

CDate reconoce formatos de fecha que se ajusten a la configuración regional de su sistema. Es posible que no se determine el orden correcto del día, mes y año si se proporciona en un formato diferente del que reconoce la configuración de fecha. Además, no se puede reconocer un formato de fecha largo si contiene la cadena del día de la semana.

Se proporciona una función *CVDate* por compatibilidad con versiones anteriores de Visual Basic. La sintaxis de la función *CVDate* es idéntica a la de la función *CDate*; sin embargo, *CVDate* devuelve un *Variant* de subtipo *Date* en lugar del tipo *Date real*. Puesto que ahora hay

un tipo de dato intrínseco *Date*, no es necesario *CVDate*. Se puede observar el mismo efecto al convertir una expresión a *Date* y asignarla después a un *Variant*. Esta técnica es coherente con la conversión de todos los demás tipos intrínsecos a sus equivalentes subtipos *Variant*.

Nota La función *CDec* no devuelve un tipo de dato discreto; en su lugar, siempre devuelve un *Variant* con los valores convertidos a un subtipo Decimal.

1.6 MULTIMEDIA.

Lastimosamente para Microsoft Access no tiene incorporado de una forma directa los controles adecuados para el manejo de la multimedia, Access controla la multimedia, mediante controles que vienen incorporados en el Office97.

Si ha instalado Office desde un CD-ROM, puede instalar y utilizar Microsoft *Camcorder* para grabar acciones, procedimientos y sonidos realizados en el equipo. Puede guardar la grabación como una película para reproducirla en *Camcorder* o bien puede guardar la película como un programa independiente para enviarlo a otros usuarios con el fin de que lo reproduzcan.

1.7 OTRAS ESPECIFICACIONES DE MICROSOSFT ACCESS.

1.7.1 COSTOS DE MICROSOFT ACCESS.

Los costos que Microsoft Access no pueden ser excesivos, puesto que forma parte de la familia Microsoft Office, por tanto este ya viene incluido en el paquete, sin embargo este paquete también se lo puede conseguir de una forma independiente.

1.7.2 REQUERIMIENTOS DE HARDWARE.

Microsoft Access es un manipulador de Bases de Datos que básicamente no requiere de hardware de gran capacidad, y para sustento de lo dicho a continuación se detallan los requerimientos que esta herramienta necesita para su funcionamiento.

a) **Velocidad de procesamiento**.- Para que Microsoft Access pueda funcionar se hace necesario disponer de un procesador Pentium de 75 MHz o superior.

b) **Sistema Operativo.-** El sistema operativo bajo el cual se van a desenvolver las herramientas es parte esencial, y para que Microsoft Access pueda ejecutarse es necesario que se tenga instalado Windows 3.1 o superior, incluido Windows NT server o Windows NT WorkStation.

c) **Memoria.-** Microsoft Access para poder trabajar necesita que el ordenador disponga de al menos 8 Mega Bytes de memoria RAM.

d) **Espacio.-** El ordenador en el cual se va a instalar Microsoft Access debe tener un espacio disponible en el disco duro de 181 Megas para instalarlo completo.

CAPITULO 2

ANÁLISIS DE VISUAL BASIC

2.1 INTRODUCCIÓN.

Cuando la programación todavía era considerada como un paradigma, apareció un lenguaje de programación denominado BASIC, el mismo que fue diseñado con el propósito de explotar a las computadoras de la época, no solo en el campo matemático, sino también en el adelanto de las demás empresas que existían en el mercado.

Posteriormente el avance del Hardware obligó a que la empresa desarrolladora de Software MICROSOFT lance al mercado una nueva versión de BASIC. Cuando los primeros microprocesadores de la IBM fueron lanzados al mercado Microsoft se vio en la obligación de lanzar un nuevo producto, proponiendo al medio de la época el GWBASIC, con el pasar de los tiempos y acorde con la demanda; Microsoft lanzó su nuevo producto de nominado Quick Basic, el mismo que llego a convertirse en el software de mayor importancia y requerimiento de aquellos tiempos.

El sentido de programación vino a cambiar con el advenimiento del un nuevo sistema operativo denominado WINDOWS, puesto que la programación tomó un nuevo horizonte hacia la interfaz gráfica para el usuario. Por tal motivo la empresa desarrolladora de software MICROSOFT en el año de 1991 lanza su nuevo producto, denominado VISUAL BASIC, su portada es mostrada en la Fig. 11.

Visual Basic, convierte la complejidad de la programación para WINDOWS en una programación realmente sencilla y fácil de usar, con características gráficas que admiten la comodidad para el diseño.

Con el mismo Visual Basic, los programadores toman otro sentido de programación debido a que este lenguaje ya estuvo diseñado para poder soportar los nuevos paradigmas de la programación orientada a objetos.



Fig. #11: Portada del Visual Basic

2.2 CREACIÓN DE PROYECTOS.

Un proyecto es un archivo generalizado que contiene una lista de todos los archivos necesarios para ejecutar un programa de Visual Basic que esté creado. Visual Basic al iniciarse tiene cuatro entradas, la primera entrada es el nombre del archivo Form1.frm formulario como se puede observar en la Fig. 12; su etiqueta (Form1) indica que el archivo está asociado con la ventana del formulario denominada Form1. Si se guarda el formulario en un disco sin cambiarlo de nombre, por defecto Visual Basic le pondrá un nombre por omisión llamado Form1.frm. Una aplicación

puede estar compuesta por muchos formularios cada uno de los cuales se almacena en un archivo separado. Como los archivos se graban por separado, los formularios pueden ser utilizados por otras aplicaciones.

Las tres entradas adicionales que se presentan al iniciar Visual Basic son los nombres de los archivos CMDIALOG.VBX, GRID.VBX, MSOLE.VBX, la extensión VBX, indica que son archivos creados por Visual Basic.

En un proyecto Visual Basic muestra un formulario correspondiente a el mismo, en el momento que un archivo de proyecto es abierto. Esta vista permite al programador diseñar la interfaz de usuario para la aplicación, que no es otra cosa que la parte de la aplicación que el usuario observa, y es precisamente con la que interactúa, tal como se muestra en la Fig. 12.

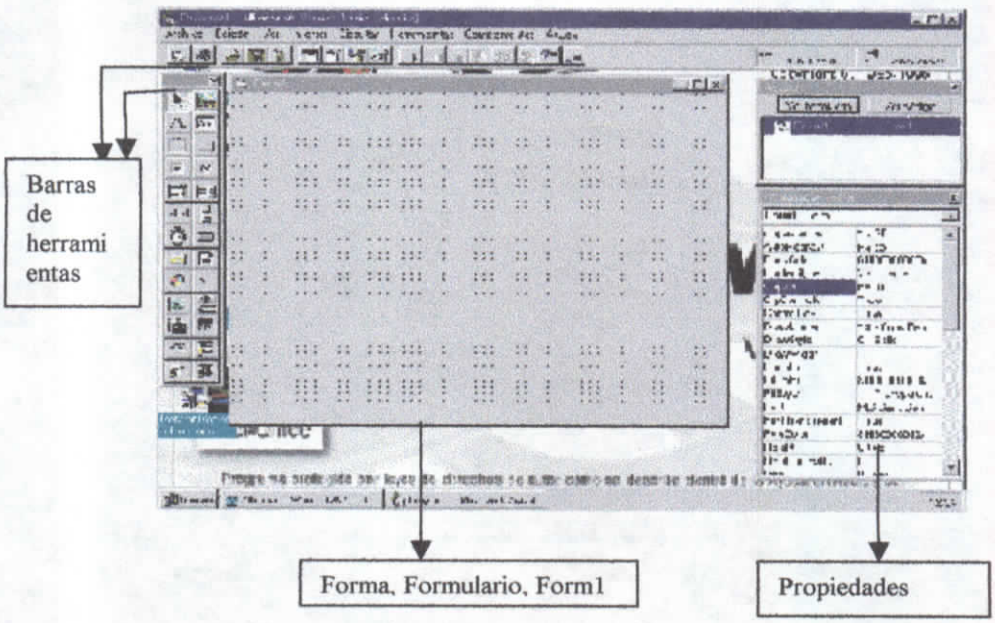


Fig. # 12 Emplazamiento de Visual Basic por medio de la cual se pueden crear proyectos

Cuando se crea un programa con Visual Basic el trabajo se divide en dos etapas, el diseño del formulario, y la codificación del programa o creación del código; que no es otra cosa que las líneas de instrucción destinadas a controlar el funcionamiento de dicho programa en todas sus instancias.

A medida que la aplicación sigue avanzando el formulario sirve como un lienzo, el mismo que contendrá diversas partes de la aplicación. Estas partes de la aplicación posteriormente se denominaran *objetos o controles*. De hecho Visual Basic considera al formulario como un objeto.

Visual Basic brinda dos facetas a sus programadores una en el momento del diseño, que es el momento en el cual esta creando la aplicación, y por ende el programador dispone de todas las facilidades que Visual Basic nos ofrece, por ejemplo: la caja de herramientas, como se muestra en la Fig. 12., es muy necesaria puesto que esta contiene todos los objetos necesarios para la construcción de la aplicación, tales como: botones de comando, cuadros de dibujo, cuadros de texto, botones de opción, cuadros de lista, barras de desplazamiento (Horizontal, Vertical), cuadro de lista de unidades, cuadro de lista de archivos, líneas, herramienta de datos, cuadriculas, punteros, etiquetas, marcos, cuadros de comprobación, cuadros combinados, herramientas de tiempo, cuadro de lista de directorios, cuadros de formas, cuadros de imágenes, cuadros de diálogos comunes, herramientas OLE, tal como se muestra en la Fig. 12.

Pero estas no son todas las herramientas que Visual Basic ofrece, también se puede proveer de herramientas desde las cajas de diálogos CMDIALOG.VBX, GRID.VBX, MSOLE.VBX. Visual

Basic, en la fase de diseño también permite escribir el programa. Pero en esta fase; ni los objetos estarán activos, ni se ejecutara el código.

Una vez terminada la aplicación está lista para pasar a la fase de ejecución, que es verdaderamente en donde se puede observar el verdadero potencial del programa realizado. Visual Basic retirara todo lo concerniente a ventanas de ayuda, y se fijara el diseño en la pantalla, en este momento los objetos en la pantalla pueden ser activados y los usuarios pueden interactuar con todos aquellos objetos que se hayan diseñado en el formulario. De la misma forma empezara a ejecutarse el código escrito en Visual Basic.

Los Proyectos de Visual Basic también contienen las propiedades de cada objeto, las propiedades son mecanismos normales que sirven para describir los atributos normales de un objeto, las propiedades de un objeto pueden ser: Color, nombre, tamaño, etc.

2.3 ESTUDIO DE LAS BASES DE DATOS EN VISUAL BASIC.

2.3.1 CARACTERISTICA DE LAS BASES DE DATOS EN VISUAL BASIC.

En general las Bases de Datos son estructuras capaces de almacenar información, dicho de otra forma, una Base de Datos es un conjunto de información relacionada que está constituida por medio de tablas, dotadas de la virtud de ir almacenando información de una forma bien estructurada y bien definida, y mas aun si hablamos de Visual Basic debido a la estrecha relación que mantiene con las Bases de Datos creadas en Microsoft Access.

Por ejemplo, en una tabla existen columnas, en las que se van a introducir datos; los mismos que contendrán información acerca de tal o cual cuestión en referencia y filas que describen todos los atributos de ese único sujeto o cuestión en referencia.

En una tabla de la Base de Datos, cada fila viene a constituirse en un registro, y cada registro, viene a describir a un único sujeto en cuestión.

Tomando en cuenta la terminología de Bases de Datos se puede decir que; una Base de Datos está constituida por un conjunto de tablas, y que a la ves una tabla esta constituida por un conjunto de registros que tienen la misma estructura.

En la actualidad recopilar y mantener información de este tipo y de esta forma se ha hecho tan habitual, que se han desarrollado aplicaciones conocidas como Sistemas de Gestión de Bases de Datos. Estos sistemas ayudan y simplifican la creación, mantenimiento, y visualización del contenido de las Bases de Datos.

Algunos de los Sistemas de Gestión de Bases de Datos que se conocen en nuestro medio son: Microsoft Access, Microsoft Fox Pro, Borland Paradox, en la actualidad la mayoría de las empresas, por no decir todas, están usando los Sistemas de Gestión de Bases de Datos, para organizar, actualizar sus registros, informaciones, etc.

Visual Basic se ha complementado de una mejor forma con el Micrososft Access, para poder organizar y manejar de una mejor forma todo en cuanto a Bases de Datos se refiere.

Visual Basic puede tener más de un *Control Data*, a mas de esto Visual Basic debe incluir un *Control Data* para cada grupo de datos seleccionado o definido. Cada vez que el *Control Data* accede; solo puede acceder a un registro, al cual se le da el nombre de registro actual. Para poder mostrar los datos contenidos en el registro actual, Visual Basic ofrece otros controles los mismos que también habrá que ligarlos a la aplicación.

Para poder acceder a los datos por medio del *Control Data* se hace indispensable y necesario añadir algunas propiedades, a estas propiedades se las puede adicionar en la fase de diseño o en la fase de ejecución, las propiedades son las siguientes:

a.1 Connect.- (Conexión) Visual Basic le entiende a esta propiedad, por que esta propiedad le entrega el formato que tiene los datos.

a.2 DatabaseName.- (Nombre de la Base de Datos). Esta propiedad identifica el archivo de Base de Datos, esta base de datos debe estar en el mismo directorio del programa de Visula Basic.

a.3 RecordSource.- (Fuente de registro) Esta propiedad le dice a Visual Basic, de donde obtener los datos, esta puede identificarse por medio de una tabla o por medio de una vista lógica, es decir lo que se conoce como un alias.

Visual Basic una vez que ya a logrado enlazar el control de datos también necesitara de otras herramientas para poder mostrar estos datos, como por ejemplo: Necesitara *Labels*, *Check Box*, *Picture Box*, *Image*, *Text Box*, en estos controles la propiedad *DataSource* se

conecta con un *Control Data*, y la propiedad *Data Field* identifica el nombre del campo de la tabla al que hay que ligar el control.

En el tiempo de diseño permite seleccionar un determinado registro utilizando el *Control Data*, pero Visual Basic brinda la facilidad de seleccionar un registro dentro del tiempo de ejecución, poniendo la propiedad *visible del Control Data a false*.

A continuación se describen algunos de los controles que tienen efecto sobre el *Control Data*.

a) **Refresh.-** (Refrescar). La Base de Datos se abre, es decir se pone a disponibilidad la base para ser utilizada, o se cierra si ya estuvo abierta.

b) **MoveFirst.-** (Mover al primero). El cursor se ubicara en el primer registro del conjunto de registros seleccionados, tomando este la denominación de registro actual.

c) **MoveNext.-** (Mover al Siguiente). Visual Basic con esta instrucción mueve el apuntador al siguiente registro, pasando a ser este el registro actual del conjunto de registros seleccionados.

d) **MovePrevious.-** (Mover al Anterior). El registro inmediato anterior pasa a ser el registro actual del conjunto de registros.

e) **MoveLast.-** (Mover al ultimo). Se establece que el registro actual es el último del conjunto de registros seleccionados.

f) **AddNew.-** (Adicionar un Nuevo Registro). Visual Basic incorpora un registro nuevo al conjunto de registros.

g) **UpDate.-** (Actualizar). Se escriben en la base de datos todos los campos editados.

h) **Delete.-** (Eliminar). Este comando permite eliminar el registro actual, es decir el registro en el cual el apuntador esta ubicado.

Visual Basic, permite intercambiar datos con los siguientes sistemas generadores de bases de datos:

Microsoft Access

Microsoft Visual Fox Pro

Borland Dbase

Borland Paradox

Cabe destacar que se podrá hacer el compartimiento de datos siempre y cuando el Visual Basic sea la versión 3.0 o superior.

Además de esto Visual Basic permite buscar registros de una forma aleatoria, es decir un registro especificado, gracias a las instrucciones siguientes:

FindFirst (Buscar primero)

FindNext (Buscar siguiente)

FindPrevious (Buscar anterior)

FindLast (Buscar último)

Estas instrucciones funcionan de manera similar a las instrucciones *Move* descritas anteriormente, con la diferencia que la instrucción *Find* funciona sobre el componente del conjunto de registros del *Control Data*, y que toma un parámetro de cadena que especifica el dato o criterio de búsqueda.

2.4 ESTUDIO DE LOS TIPOS DE DATOS EN VISUAL BASIC.

Para poder estudiar los tipos de datos se debe tener claro de cómo y en que unidad se puede medir el traspaso o representación de información, por consiguiente la descripción de un *bit* en este estudio es de suma importancia.

Un *bit* es la unidad fundamental que mide la cantidad de información que puede ser transmitida de un lugar a otro. Un *bit* es el bloque de información más pequeño del universo de la información, por ejemplo una respuesta a una pregunta en cuestión necesita de al menos un *bit* para poder ser representada, y si no hay *bits*, no hay respuestas.

Pero el uso de *bits* representan un conflicto para representar información, puesto que un *bit* en la realidad no representa mucha información, y como en la actualidad los ordenadores están regidos

por la nomenclatura de los códigos ASCII extendida, lo mínimo que se necesitan son 8 *bits*, para poder representar un caracter.

Por ende se puede hablar de que lo mínimo que necesita un caracter es 8 *bits*, y sirve a los programadores como unidad de medida básica, a esta unidad básica de medida de 8 *bits* se la denomina *byte*. Para representar datos, bloques de información, a menudo se hace necesaria la presencia de una mayor capacidad que la del *byte*, es por eso que se ha tomado en cuenta al *KiloByte*, *MegaByte y GigaByte*, y detrás de este podemos localizar al *TeraByte*, y luego al *PetaByte*.

En la actualidad se están manejando información de 32 y 64 *bits*, lo cual obliga a manejar números binarios de orden de miles de millones.

Para tratar de resolver en parte este problema, Visual Basic brinda a los usuarios la facilidad de poder especificar distintos tipos de datos, que sirven para poder especificar e introducir varios tipos de información. A continuación especificaremos las características de los tipos de datos que Visual Basic ofrece.

Tabla #2: Tipos de datos en Visual Basic.

Nombre de Tipo de Dato	**Descripción**	**Rango de Valores**	**Espacio requerido**
Integer (Entero)	Numérico Entero	De -32768 a 32767	12 bits (2 bytes)
Long (Largo)	Numérico Entero	De -2147483648 a 2147483647	32 bits (4 bytes)

String (Cadena)	Texto	De 0 a 65500 caracteres aproximadamente	8 bits (1 byte) por caracter
String (Longitud Variable)	Texto	De 0 hasta aproximadamente 2000 millones	10 bytes + long de cadena
Currency (Moneda)	Numérico Decimal Fijo	De -922337203685477,5807 a 922337203685477,5807	64 bits (8 byres)
Single (Simple)	Numérico Real	De +- 1,40 x 10^{45} a +- 3,40 x 10^{38}	32 bits (4 bytes)
Double (Doble)	Numérico Real	De +- 4.94 x 10^{324} a +- 1,79 x 10^{308}	64 bits (8bytes)
Variat (con números) Variable	Doble	Adaptable a cualquiera de los anteriores en el rango	16 Bytes
Variat (con caracteres) Variable		El mismo rango que para cadenas de long. Variable	22 Bytes + Long de cadena
Byte		De 0 a 255	1 Byte
Boolean		TRUE o FALSE	2 Bytes

Date		Del 01-01-100 al 31-12-9999	8 Bytes
Object		Cualquier referencia a objetos	4 Bytes
Type (Definido por el usuario)		El rango de cada elemento es el mismo # de elemento que el de su tio de datos requerido	

2.4.1 DATOS DE TIPO ENTERO.

Los datos de tipo *Integer* y tipo *Long* son datos que solo requieren de 16 y 32 *bits* respectivamente de espacio en memoria, y además de eso solo permiten ingresar información numérica, sin parte fraccionaria, los tipos de datos *integer* solo necesitan de 16 *bits* en memoria, pero de la misma forma que requieren menos espacio, este tiene un rango más restringido que los datos de tipo *long*.

Los valores *Single* y *Double*, son representados con números de coma flotante, o denominados números reales. Estos tipos de datos representan valores que poseen parte fraccionaria. El nombre de coma flotante aduce a que la coma puede aparecer en cualquier lugar del número, lo que contrasta con el tipo de datos *Currency*, que es un claro ejemplo de coma fija.

2.4.2 DATOS DE TIPO TEXTO.

Los datos de tipo *String* son representados por una cadena de caracteres, cada una de las cuales estará representada por su correspondiente código ASCII, a una secuencia que no contenga ningún caracter se le denomina cadena vacía.

2.4.3 DATOS DE TIPO MONEDA.

Visual Basic también proporciona el tipo de datos *Currency*, el mismo que fue diseñado primordialmente para representar datos de tipo moneda o valores monetarios, un valor monetario para su precisión se hace conveniente representarlo con cuatro decimales.

Visual Basic también permite representar la información de tipo fecha o de tipo hora por medio del tipo de datos *Date*.

Por último Visual Basic define un tipo de datos llamado *Variant*, que funciona como una especie de comodín, es decir este tipo de dato puede tomar cualquier valor de los antes pronunciados. Realmente este tipo de dato se lo puede tratar en dos partes, una parte que contiene el valor, el mismo que puede pertenecer a cualquier tipo de dato, y la otra parte contendrá un código que indique el código el tipo de valor, por ejemplo (*Currency, String*, etc).

2.5 MANEJO DE CADENAS EN VISUAL BASIC.

Una cadena en Visual Basic está tomada como una secuencia de caracteres en donde que cada caracter utiliza un *byte* de espacio en memoria, el mismo que estará representado por un sistema de codificación de caracteres ASCII.

Visual Basic brinda a los usuarios la facilidad de evaluar cualquier secuencia de caracteres contiguos, por ejemplo, Obtener la longitud de una cadena mediante la instrucción *string()*, diseñada específicamente para realizar dicha función en Visual Basic.

Visual Basic también incluye la posibilidad de concatenación de cadenas mediante el operador +, el mismo que puede traer confusión al sumar variables de tipo *entero, long, double*; tomando en cuenta estas consideraciones, Visual Basic diseñó otro tipo de operador para concatenar cadenas de tal forma que no conlleve a confusión, este tipo de operador es el &.

Visual Basic dentro del contexto de manejo de cadenas tiene incluido funciones, que facilita a los usuarios de este lenguaje la búsqueda de cadenas en el interior de otras cadenas, la devolución de un número específico de caracteres solicitados que se encuentren en el interior de otra cadena, devolución de caracteres de una parte de la cadena, indistintamente de cual sea su caracter de inicio, y el número de caracteres que se desea sea devuelto, reemplazar caracteres de una cadena con caracteres de otra cadena especificada.

Es decir Visual Basic brinda la facilidad de manejar y manipular cadenas de la forma que el usuario considere la mas correcta, de acuerdo a sus necesidades y la de el programa que esté desarrollando o desee realizar.

Podemos citar y explicar algunas de las funciones que Visual Basic proporciona a sus programadores:

- La función *Asc()* utilizada en Visual Basic con el único fin de devolver el código del caracter correspondiente a la primera letra de la cadena evaluada.
- La función *Cbool()*, convierte cualquier tipo de expresión valida, sea esta una expresión numérica o a una expresión de cadena a un tipo de datos *booleano*.
- *Cbyte()* es una función que convierte una expresión, valida a una expresión de tipo *byte*, siendo esta expresión de tipo numérico o de tipo cadena.
- La función *Ccur()*, convierte una expresión a datos de tipo *currency*, o datos de tipo moneda, siempre que el parámetro a ser convertido sea una expresión valida.
- La función *Cdate()*, convierte una expresión de fecha valida a una expresión de tipo *Date*.
- La función *IsDate()*, es utilizada para poder determinar si la expresión de fecha, es una expresión valida para poder transformar a un tipo de datos *Date* con la función *Cdate()*, anteriormente descrita.
- La función *CVDate()*, es incluida en Visual Basic por compatibilidad con otras versiones, puesto que esta función hace exactamente lo mismo que la función *Cdate()*, la diferencia, radica en el valor de retorno, la función *CVDate()*, retorna un valor de tipo *Variant*, en lugar de un valor de tipo *Date real*.

- *Chr()*, es una función que devuelve el caracter asociado con el código del carácter especificado.
- *Str()*, es una función que devuelve una representación de cadena a partir de un número.
- La función *Format()*, es importante para poder convertir valores de tipo numérico en valores con formato de tipo fecha.
- *CStr()*, esta función es utilizada para convertir valores numéricos en cadenas de caracteres.
- *Cint()*, Convierte una expresión a un tipo de datos *Integer*, siempre y cuando la expresión a convertir sea valida, y esté comprendida entre los valores que el tipo de datos *Integer* abarca.
- La función *Clong()*, es utilizada para convertir una expresión en datos de tipo Long, en momentos en que las operaciones aritméticas que se estén realizando sean de gran presición.

2.6 MULTIMEDIA.

En Visual Basic para crear animación o simulación se puede implementar el objeto denominado *PictureBox*, que por medio de la manipulación de sus propiedades y métodos gráficos en código, se pueden mostrar distintos tipos de gráficos.

Las propiedades y los eventos gráficos son útiles para operaciones de impresión en tiempo de ejecución, como modificar el formato de un formulario de pantalla para imprimirlo.

Un control *PictureBox* también puede actuar como vínculo de destino en una conversación DDE. *PictureBox* es el único control estándar de Visual Basic que se puede colocar en el área interna de un formulario MDI. Se puede emplearlo para agrupar controles en la parte superior o inferior del área interna para crear una barra de herramientas o de estado.

2.7 OTRAS ESPECIFICACIONES DE VISUAL BASIC.

2.7.1 COSTOS DE VISUAL BASIC.

Los costos que Visual Basic ofrece a sus usuarios son relativamente bajos, puesto que este tiene un precio para su instalación estándar y otro precio para el paquete completo.

2.7.2 REQUERIMIENTOS DE HARDWARE.

Visual Basic como una herramienta de gestión básicamente no requiere de hardware de gran capacidad, y para sustento de lo dicho a continuación se detallan los requerimientos que esta herramienta necesita para su funcionamiento.

a) **Velocidad de procesamiento.-** Para que Visual Basic pueda funcionar se hace necesario disponer de un procesador 80286 o superior.

b) **Sistema Operativo.-** El sistema operativo bajo el cual se van a desenvolver las herramientas es parte esencial, y para que Visual Basic pueda ejecutarse es necesario que se tenga instalado Windows 3.1 o superior.

c) **Memoria.-** Visual Basic para poder trabajar necesita que el ordenador disponga de al menos 1 Mega Bytes de memoria RAM libres.

d) **Espacio.-** El ordenador en el cual se va a instalar Visual Basic debe tener un espacio disponible en el disco duro de 30 Megas para instalarlo en su mínima capacidad.

CAPITULO 3

ANÁLISIS DE MICROSOFT VISUAL FOXPRO

3.1 INTRODUCCIÓN.

El FoxPro fue creado por la empresa de software MICROSOFT, como una necesidad de gestionar bajo una mejor forma datos almacenados en las tablas, con una mayor eficiencia y rapidez. Los primeros gestionadores de datos del FoxPro fueron creados bajo una plataforma y un ambiente de programación de DOS (Sistema Operativo), el FoxPro con el pasar de los tiempos siguió perfeccionándose llegando a constituirse en uno de los más grandes y potentes gestionadores de Bases de Datos.

El FoxPro también se caracteriza por ser al mismo tiempo un lenguaje de tipo interactivo y compilable, potente, sencillo y compatible con dBASE III, PLUS, dBASE IV, la diferencia del FoxPro con estos gestionadores de bases de datos radica en que es mucho más rápido en cuanto a su tiempo de compilación y acceso a registros se refiere.

Con la implantación del sistema operativo WINDOWS. MICROSOFT también crea una nueva versión del FoxPro, como es el caso de FoxPro para WINDOWS. Con el afán competitivo del medio MICROSOFT destinado a la programación orientada a objetos crea el VISUAL FOXPRO.

El mismo que es introducido al mercado en 1995, presentando el cambio más grande en el desarrollo de Xbase, MICROSOFT, gastó mucho tiempo y esfuerzo con la creación del VISUAL

FOXPRO 3.0, y en la actualidad tenemos el VISUAL FOXPRO 5.0, y el VISUAL FOXPRO 6.0, los mismos que a pasos agigantados se están solidificando en el mercado, su portada se muestra en la Fig. 14.



Fig. #14 Portada del VISUAL FOXPRO 5.0

3.2 CREACIÓN DE PROYECTOS

Visual FoxPro es una de las herramientas más potentes que se han lanzado al mercado, por su potente sistema de gestión de bases de datos y su entorno de desarrollo de aplicaciones, el Visual FoxPro proporciona al usuario las herramientas necesarias para que los datos puedan ser manipulados directamente y accederlos por medio del ratón, todo esto es gracias a la potencia y versatilidad de Visual FoxPro.

Muchas de las tareas que sólo eran posible realizarlas sobre una supercomputadora, en la actualidad se lo puede realizar sobre su PC, gracias a las características antes mencionadas del Visual FoxPro.

En VISUAL FOXPRO las herramientas más utilizadas dentro de la construcción de proyectos son: las barras de herramientas visuales, los generadores de diseños; que permiten crear, modificar, eliminar, rápidamente y de forma gráfica elementos del proyecto como son los formularios, reportes, informes, bases de datos, programas, y, clases. Tal como se muestra en la Fig. 15.

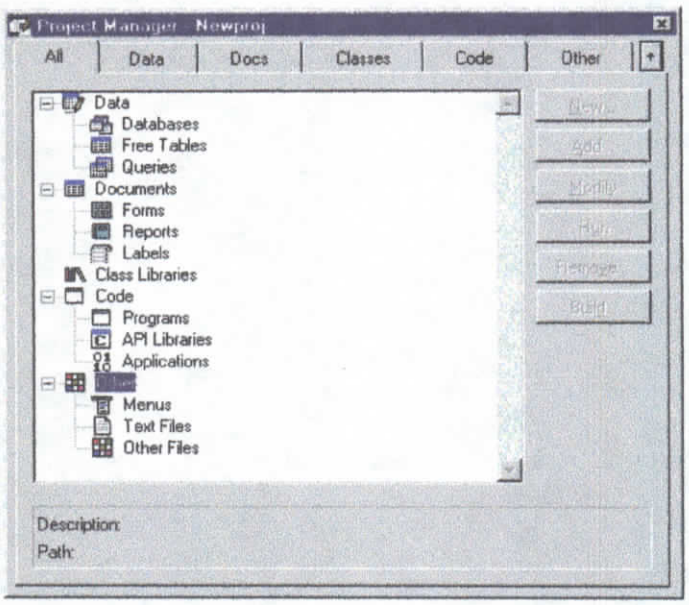


Fig. #15 Proyecto de VISUAL FOXPRO con sus respectivos elementos que forman el proyecto.

Las herramientas que Visual FoxPro pone a disposición de sus usuarios y programadores en general son: etiquetas, cuadros de texto, cuadros de edición, botones de comando, grupos de botones de comando, botones de opción, botones de alternar, cuadros combinados, cuadros de lista, mallas, cuadros de imagen, botones horario, páginas de selección, controles tipo OLE, líneas, figuras geométricas, contenedores, separadores, guías.

Los Asistentes de generadores, guían al usuario por medio de preguntas y respuestas fáciles durante la creación de formularios, reportes, informes, Bases de Datos; a las Bases de Datos se les puede proporcionar reglas de validación, con el fin de: Regular datos a nivel de tabla, gracias a la potencialidad que Visual FoxPro brinda a los usuarios.

En Visual FoxPro también se puede manipular el generador de clases, esto permite realizar nuevas y rápidas clases, permitiendo al programador que disminuya notablemente su tiempo de programación, puesto que los cambios realizados en las propiedades de la clase principal afectará a todas las clases dependientes de esta, del mismo modo si se hacen cambios en las subclases finales, solo afectara a esta.

Del mismo modo Visual FoxPro pone a disposición de los usuarios las herramientas de tipo Cliente/Servidor, el mismo que permite al usuario utilizar datos de un servidor remoto, o localización remota, y a datos almacenados en un formato diferente al Xbase (Formato .DBF). También se puede utilizar este tipo Cliente/Servidor para poder crear prototipos de aplicaciones en una máquina local y luego transportarla, sin tener la necesidad hacer cambios de programación de última hora.

Los sistemas multiusuarios en VISUAL FOXPRO no requieren de mucho código de programación adicional, mediante Visual FoxPro se puede establecer el método de almacenamiento en *buffer*, de registros, tablas y bloquearlos, para de este modo proteger los cambios que se realicen en los registros de un usuario con las modificaciones que se realicen desde otro usuario, permitiendo que los procesos de transacción fallidos sean revocados en su actualización.

Para que un proyecto realizado en Visual FoxPro tenga una consistencia reforzada se hace necesario comprender más a fondo los términos que permiten hacer consistentes a los proyectos como son:

a) **Abstracción.-** La abstracción modela un objeto del mundo real, debido a que el usuario solo necesita saber la función que cumple un objeto, que hace el objeto más no como este lo hace.

b) **Clases y Subclases.-** Son patrones o plantillas en donde los objetos son plasmados, los objetos son creados desde una clase en el momento que se ejecuta la aplicación. Un objeto es el resultado de convertir el diseño del patrón en realidad.

c) **Jerarquía de clases.-** Hace referencia a la relación de origen de Clases/Subclases/objetos.

d) **Encapsulamiento.-** Hace referencia a como esta constituido el objeto, queriendo decir que su funcionalidad ya esta establecida, y el usuario solo tiene que usarlo, es decir que sus características y propiedades ya están establecidas.

e) **Herencia.-** Hace referencia a la forma en la que los objetos toman las propiedades y métodos de la clase desde la que fueron creadas, en este caso las subclases heredan las propiedades de la clase base, los objetos heredan las propiedades de las subclases, y de sus clases de origen, si se generan cambios en las clases de origen, las subclases y objetos reflejaran automáticamente estos cambios.

f) **Instancia de una clase.-** Esta es otra forma de hacer referencia a un objeto.

g) **Método.-** Son procedimientos (programas), vinculados a objeto, y se ejecutan como respuesta a una orden o un evento.

h) **Objetos.-** Están formados por clases que combinan datos y propiedades, un formulario puede ser un objeto, pero también son los controles que aparecen dentro de este.

i) **Polimorfismo.-** Esto quiere decir que los objetos pueden llamar a métodos con el mismo nombre, pero el funcionamiento se basa en las funciones del objeto y no del método.

j) **Propiedades.-** Son las características, atributos que se les puede proporcionar a los objetos o clases.

Visual FoxPro permite la programación orientada a objetos, esto quiere decir que el usuario de este lenguaje de programación podrá reutilizar ó reducir código, y objetos; en general su programación será mucho más eficiente y rápida.

Los Objetos reutilizables son aquellos que una ves fueron creados y en posteriores ocasiones se las puede volver a utilizar, ya sea en el mismo programa o en otros, sin tener la necesidad de modificarla. De la misma forma se podrán combinar objetos para poder formar una super clase.

En cuento a que Visual FoxPro permite un desarrollo más rápido se puede hacer referencia a las clases y objetos reutilizables, así se ahorra tiempo al no programar largas y aburridas líneas de programación que ya están constituidas, al mismo tiempo obtenemos un menor código de programación, y por ende se obtiene un proyecto con menos posibilidades de error, estas características brindan la facilidad de un mantenimiento mucho más fácil, y la facilidad de modificar objetos.

Los Objetos no son otra cosa más que una clase basada en si mismo o una colección de clases que realizan una función especifica, un objeto siempre contiene dos tipos de información, las características, y su comportamiento, las características también son denominadas propiedades o atributos, y los comportamientos también son denominados métodos, si un objeto tiene propiedades y métodos en común, estos objetos deben ser creados a partir de una misma clase, como anteriormente se explico una clase define las propiedades y métodos de todos los objetos que a esta pertenecen, a esto se le denomina herencia.

Los eventos de los objetos son disparados o ejecutados al hacer click, doble click, mover el mouse, etc. y estos eventos ejecutan una acción, los mismos que provocan la ejecución del método.

Los métodos son controlados y regularizados por las líneas de programación o líneas de código, que conforman procedimientos o programas; necesariamente estos tienen que estar vinculados a un objeto, de una forma directa o indirecta.

En Visual FoxPro se utiliza el administrador de proyectos tal como se muestra en la Fig.15 para por medio de este poder gestionar sus archivos. Un proyecto es una colección de archivos, datos, documento, Bases de Datos y objetos de Visual Fox Pro, estos son guardados en una archivo con extensión .PJX, también brinda la oportunidad de crear archivos de proyecto con extensión .APP, o en su ves al proyecto hacerle un archivo ejecutable .EXE, esta opción solo esta disponible en la versión profesional, se pueden crear archivos ejecutables de 16 o 32 *bits*.

Visual FoxPro, también pone a disposición una opción que permite convertir proyectos realizados en versiones anteriores, o inclusive, proyectos construidos en FoxPro para windows o FoxPro con plataforma de DOS.

3.3 ESTUDIO DE LAS BASES DE DATOS EN VISUAL FOXPRO.

La característica del FoxPro bajo cualquier plataforma que esta se desarrolle, es la gestión de Bases de Datos, por tal motivo es una de las herramientas mas apetecidas en el mercado en cuanto a construcción de sistemas manipuladores de datos se refiere, y mucho más ahora que esta en el mercado el Visual FoxPro.

El visual FoxPro extiende su capacidad en cuanto al almacenamiento de datos se refiere. Una Base de Datos en Visual FoxPro está constituida por tablas como se puede observar en la Fig. 16, la misma que consta de registro y campos, el conjunto de campos constituyen un registro, y es precisamente este registro el que va a contener todos los datos referentes a un mismo caso en cuestión, y el conjunto de estos registros son los que conforman la tabla, que es el lugar físico del disco en el cual se almacenan todos los datos.

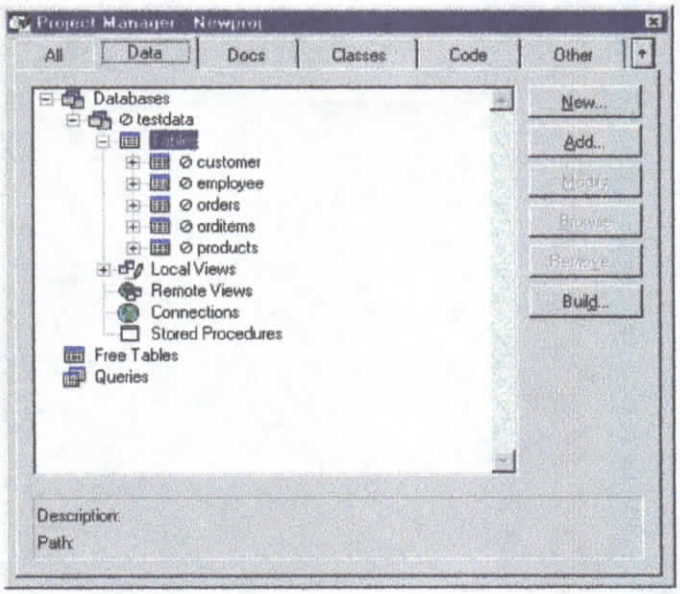


Fig. #16: Una Base de Datos Visual FoxPro con sus respectivas Tablas.

Visual FoxPro dispone de un administrador de Bases de Datos como se ve en la Fig. 16, el mismo que va a permitir al usuario *adicionar* tablas a una base de datos, esta tabla debe tener una extensión .DBF, y no es importante que sea diseñada en una versión de Visual FoxPro anterior, o incluso que haya sido diseñada en FoxPro con plataforma DOS, debido a que al ser importada dicha tabla esta cambiara su formato a la versión de Visual FoxPro actual, en el momento en que una tabla es adicionada esta pasa a formar parte del proyecto, y por ende de la Base de Datos perteneciente a este proyecto.

De la misma forma el administrador de Bases de Datos facilitara al usuario *crear* nuevas tablas, esta creación se la hará de dos formas; la primera forma que el administrador ofrece, es un ayudante, el mismo que guiará paso a paso durante toda la construcción de la Tabla con preguntas fáciles de seguir. Inclusive el ayudante facilita la creación de un nuevo archivo índice. Y la segunda forma es la construcción de la tabla manualmente tal como se observa en la Fig. 17, es decir que no existirá ninguna guía; Visual FoxPro, también brinda la facilidad de poder

modificar la estructura de las tablas, ya sean estas creadas bajo el asistente o creadas individualmente.

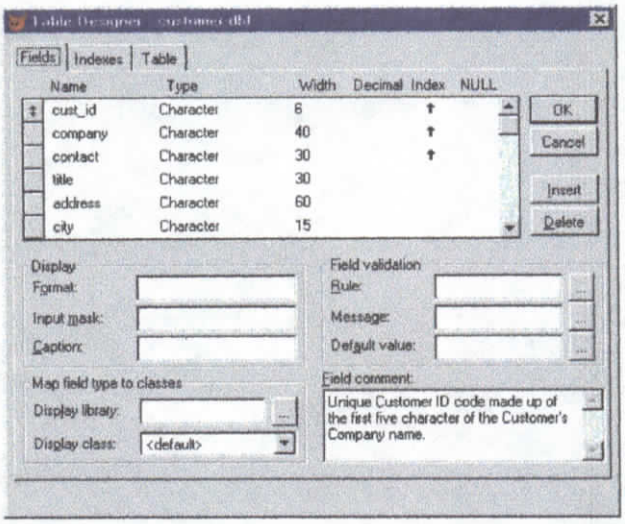


Fig. #17: Estructura de una tabla en Visual FoxPro con sus respectivos Indices y Campos.

Las tablas que se encuentran incluidas en el proyecto, y que son parte de la Base de Datos de este mismo proyecto, se las puede *eliminar*, *examinar* su contenido mediante una rejilla Fig. 19, en donde se podrá visualizar todos los datos que dicha tabla contenga.

3.3.1 BASES DE DATOS RELACIONALES.

Visual FoxPro brinda al usuario la facilidad de poder acceder a un mismo tipo de información, distribuidas en distintas tablas, mediante las relaciones como se puede observar por medio de la Fig. 18. Adicionalmente a esto en Visual FoxPro existen relaciones que a continuación se detallan:

3.3.1.1 Relación de uno a uno.

Consiste en tener dos tablas, la primera tabla debe ser la tabla principal, y la otra tabla debe ser la tabla relacional, y solo se puede hacer referencia o asociarse de un registro de la tabla principal a un registro de la tabla relacional.

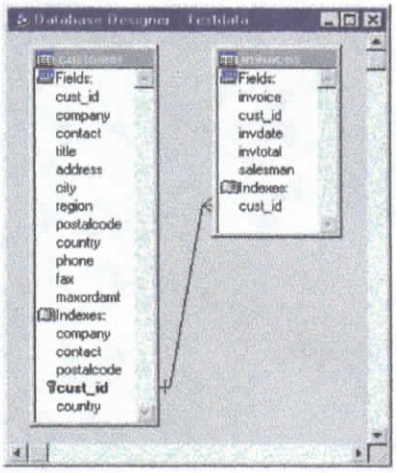


Fig. #18: Tablas con sus respectivas Relaciones en VISUAL FOXPRO

3.3.1.2 Relación de uno a varios.

Es una relación entre tablas, en la que debe existir una tabla maestra o primaria, y por lo menos una tabla secundaria o relacionada; en esta relación cada registro de la tabla primaria, debe tener asociados uno o más registros a la tabla relacionada, para esta relación la tabla maestra o primaria contendrá la clave principal, y en la tabla secundaria o relacionada el valor de esta clave principal puede repetirse un incontable numero de veces.

3.3.1.3 Relación de muchos a muchos.

Este tipo de relación es una relación entre tablas, en donde que los registros de cada tabla pueden tener varios registros coincidentes en la tabla relacional. Sin embargo este tipo de relación puede acarrear dificultades, tomando en cuenta la estructura de las tablas, por tal motivo se pueden dar ciertos cambios en el diseño de la Base de Datos. Para este tipo de relación, se hace también necesario revisar lo que es una tabla de uniones. Esta tabla de uniones es una tabla creada para poder separar una relación varios a varios creada entre dos tablas, la separación se la hace en dos relaciones; uno a varios y varios a uno, la tabla de uniones es la que contendrá las claves principales de las dos tablas con relación varios a varios.

En cuanto a la clave principal podemos decir que esta consiste en que los valores que contengan la columna o columnas, identificarán de manera exclusiva a una fila de la tabla, es decir que un valor que ya conste en esa tabla, y este dentro de la columna identificada como clave principal, no se podrá volver a repetir en ninguna otra fila de la tabla.

Visual FoxPro da al usuario una mayor facilidad y rapidez en cuanto al ordenamiento de sus registros y sus datos se refiere por medio de la construcción de archivos índices, estos archivos tendrán la extensión .IDX, CDX. Los archivos índices están compuestos por una serie de punteros ordenados lógicamente por los valores de una clave. Los archivos índices son los que ofrecen acceso rápido a datos y por medio de estos se puede exigir la integridad de los registros en las tablas. Los índices de Visual FoxPro se mantienen en un archivo de índices.

Los archivos índices con extensión .IDX son los archivos que no son de composición compuesta, es decir que solo tiene una entrada, este archivo de índices debe estar activo cuando se vaya a utilizar una tabla, y debe estar abierto explícitamente. En cambio que los archivos con extensión .CDX son los que tiene varias entradas de índices denominadas etiquetas, cada etiqueta esta definida por un nombre único.

Existen varios tipos de índices que Visual FoxPro ofrece a sus usuarios, estos son: índice primario, índice candidato, índice regular, índice único, clave principal, que a continuación se detallan.

a) **Índice Primario**.- Llamado también índice principal, debido a que este tipo de índice no permite que se ingresen valores duplicados para los campos especificados como principales de una misma tabla, este tipo de índices pueden ser usados en las tablas principales o referenciadas, que servirán para poder obtener una integridad referencial, o una relación persistente. Este índice principal solo puede ser creado una sola ves en una tabla.

   a.1 Integridad referencial.- La Integridad referencial, son las reglas que rigen la coherencia de los datos, especialmente con la relación que existe entre la clave principal, y los índices externos de tablas distintas, estas reglas se dan a nivel de campo y a nivel de registro, estas misma reglas son definidas por el usuario, y permitirá la eliminación de registros, modificación de registros en forma de cadena, es decir si se modifica o se elimina un campo de la tabla principal, se modificaran o se eliminaran los campos coincidentes de las tablas relacionadas.

a.2 Relación persistente.- Esta es una relación que durará mucho más allá que en el tiempo de ejecución, para que se de este tipo de relación las tablas deben estar en una misma base de datos.

b) Indice Candidato.- Este es un tipo de índice que en un determinado momento de la ejecución del programa, puede convertirse en clave principal, debido a que no permite valores nulos o duplicados, dentro de este tipo de índice se puede encontrar a un índice *alternativo*, que es aquel índice candidato que no se eligió como clave principal.

c) Índice Regular.- Este es un índice que no es único, principal o candidato, este tipo de índice es usado para poder ordenar los registros, para así poder tener un acceso más rápido en la búsqueda de registros, pero con este tipo de índices no se puede exigir la integridad de sus registros, o la unicidad de datos en las tablas.

d) Índice Único.- Este es un tipo de índice en el que dos registros no pueden tener el mismo valor de índice. Esta consideración es tomada como un requisito para poder construir una clave principal.

El Visual FoxPro permite tener varios archivos índices abiertos simultáneamente, pero solo puede tener un archivo índice de entrada único, o una etiqueta de un archivo índice compuesto, que son los que controlan el orden en que una tabla es visualizada, o en el orden en que se la podrá acceder a dicha tabla, por medio de los comandos especiales, que esta herramienta visual brinda.

Visual FoxPro brinda al usuario la facilidad de poder trabajar con un subconjunto de registros, tomados de la misma tabla sobre la cual se está trabajando; Esto se puede lograr gracias a los filtros que se realizan sobre este lenguaje visual de programación, estos filtros sirven también para poder ordenar los registros, tomando en cuenta algunos criterios indispensables es decir que no se los puede omitir.

Para poder ordenar los registros de una tabla, u ordenar los registros de un archivo índice bajo este criterio, se tomara en cuenta la expresión SET FILTER TO, y a continuación la expresión o la condición por la que se desea hacer el filtro.

Una forma de mostrar los datos contenidos en una base de datos es por medio de la cuadrícula como se observa en la Fig. 19, una cuadrícula es un objeto contenedor compuesto por filas y columnas, una columna tomada como un objeto puede contener otro objeto que es el objeto encabezado, y a su vez este objeto encabezado puede tener controles; y a su vez el conjunto de columnas encabezados y controles, disponen de su propio conjunto de propiedades. Estas cuadrículas pueden ser creadas automáticamente por el generador de cuadrículas que Visual FoxPro pone a disposición de sus usuarios, o también pueden ser generadas manualmente por el usuario, esto es sin ayuda del generador.

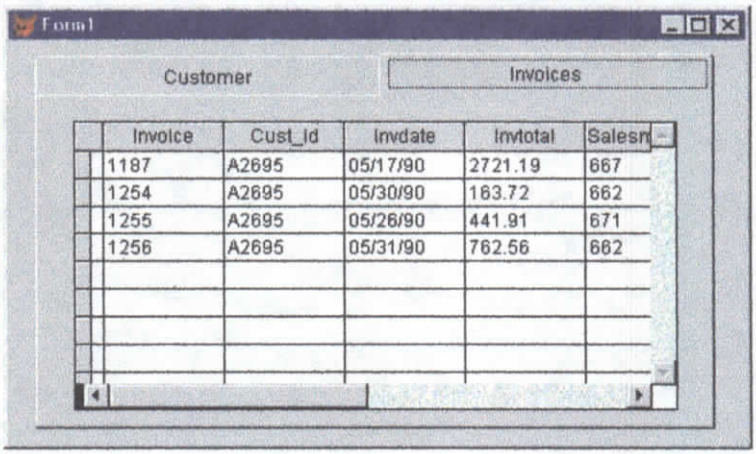


Fig. #19: Rejilla con datos de una Tabla de VISUAL FOXPRO

En una cuadrícula se puede agregar registros, eliminar registros, adicionar columnas y filas, ocultar y mostrar marcas de eliminación, una característica especial de la cuadrícula es que permite mostrar campos calculados, en un determinado campo de la cuadrícula.

Por esto y otras razones adicionales el Visual FoxPro pasa ser una de las herramientas de programación Visual mas poderosas en cuanto a gestión de bases de datos se refiere.

3.4 ESTUDIO DE LOS TIPOS DE DATOS EN VISUAL FOXPRO.

Visual FoxPro se ha preocupado de poner a disposición de sus usuarios una mayor cantidad de tipos de datos, y modificar y mejorar los ya existentes, en versiones anteriores, para poder llegar a ser una de las herramientas más poderosas, en cuanto a plataformas de soporte con gestión de bases de datos se refiere.

A continuación se detallan los tipos de datos que Visual FoxPro pone a disposición de sus usuarios.

Tipo de datos es el atributo de una variable o un campo que determina el tipo de datos que puede contener. Visual FoxPro tiene estos tipos de datos: carácter (Character), moneda (Currency), fecha (Date), fecha hora (DateTime), doble (Double), flotante (Float), lógico (Logical), numérico (Numeric), entero (Integer), general, memo y se detalla a continuación.

Tabla #3: TIPOS DE DATOS disponibles en Visual FoxPro

TIPO DE DATOS	DECRIPCIÓN	TAMAÑO	INTERVALO
CHARACTER	Cualquier texto	1 byte por carácter hasta 254	Cualquier Carácter
CURRENCY	Cantidades monetarias	8 bytes	-922337203685477,5808 a 922337203685477,5807
DATE	Datos cronologicos formados por mes,año y día	8 bytes	01/01/100 a 12/31/9999
DATETIME	Datos cronologicos formados por mes, año, día y hora	8 bytes	01/01/100 a 12/31/9999 más, 00:00:00 a.m. a 11:59:59 p.m.

LOGICAL	Valor booleano verdadero o falso	1 byte	Verdadero (.T.) Falso (.F.)
NUMERIC	Enteros o fracciones	8 bytes de la memoria; 1 a 20 bytes en una tabla	-.9999999999E+19 a .9999999999E+20
DOUBLE	Número de coma flotante de precisión doble	8 bytes	+/- 4.94065645841247-324 a +/- 8988465674315E307
FLOAT	Igual que el Numeric	8 bytes en la memoria; 1 a 20 bytes en una tabla	-.9999999999E+19 a .9999999999E+20
GENERAL	Referencia a un objeto OLE	4 bytes en una tabla	Limitado por la memoria dispponible
INTEGER	Valores enteros	4 bytes	-2147483647 a 2147483646
MEMO	Referencia a un bloque de datos	4 bytes en una tabla	Limitado por la memoria disponible
CHARACTER BINARIO	Datos de tipo carácter que quieren mantenerse sin modificación en la pág. de códigos	1 byte por carácter, hasta 254	

MEMO BINARIO	Datos de campo memo que quiere mantener sin modificación en pág. de códigos	4 bytes en una tabla	Limitado por la memoria disponible

3.4.1 TIPO DE DATOS CARACTER.

Este tipo de datos que Visual FoxPro pone a disposición de los usuarios son de gran utilidad, debido que el campo que disponga de este tipo de datos será capaz de contener; letras, números, espacios, símbolos y puntuación.

Los campos o variables que contenga este tipo de datos, podrán ser capaces de almacenar nombres, direcciones, o datos numéricos que no se vayan a utilizar en operaciones o cálculos matemáticos, también este tipo de datos es recomendable para utilizarlo como clave principal de una tabla, o como campo de un archivo índice normal.

3.4.2 TIPO DE DATOS MONEDA.

Llamado también *Currency*, este tipo de datos es utilizado especialmente cuando un campo requiere trabajar con datos monetarios, o cuando un valor necesita de más exactitud, o trabajar con más de cuatro dígitos decimales. Visual FoxPro tiene la capacidad de redondear la cantidad

especificada a cuatro dígitos decimales, en caso de que la cantidad de dígitos decimales sea mayor a cuatro.

Para que Visual FoxPro entienda que el dato numérico a introducir es de tipo moneda o currency se debe agregar un símbolo de dólares '$' antes de la cantidad, así:

Cantidad = $25,2365

En donde Cantidad es la variable que va a recibir el valor tipo moneda de 25,2365.

3.4.3 TIPO DE DATOS FECHA.

Este tipo de dato especifica valores de tipo fecha exclusivamente, es decir que no especifica el valor de la hora, dicho tipo de dato es almacenado en un formato de caracter 'aammdd', Visual FoxPro, consta en su interior de una de las características más importantes en cuanto a la fecha se refiere, puesto que acepta el año con cuatro dígitos, así, 'aaaammdd'.

Para que una variable reconozca y acepte que es un formato de fecha, esta fecha debe estar entre llaves como a continuación se describe {1998.09.30}, el formato de la fecha se la puede configurar de acuerdo a sus necesidades en una herramienta que el mismo Visual FoxPro presta a sus usuarios.

3.4.4 TIPO DE DATOS FECHA HORA.

Este tipo de datos sirven al usuario para poder almacenar datos que son de tipo fecha, hora o datos de tipo fecha hora, es decir los dos a la vez, este tipo de datos es almacenado en 8 *bytes*, 2 enteros de 4 *bytes*, los ocho *bytes* son distribuidos de la siguiente manera: los 4 primeros *bytes* sirven para almacenar la fecha y los cuatro restantes representan la hora.

En este tipo de datos se hace necesario especificar una fecha, puesto que si no se especifica la fecha, Visual FoxPro pondrá automáticamente una fecha que es el 30 de Diciembre de 1899, de la misma forma si no ingresa la hora Visual FoxPro tomará como hora predeterminada la media noche.

3.4.5 TIPO DE DATOS DOBLE.

Este tipo de datos viene a sustituir al tipo de datos numéricos, en el momento en el que el usuario desee agregar una mayor precisión en sus valores a almacenar, o cuando el usuario necesite almacenar en la tabla valores de coma flotante verdaderos.

3.4.6 TIPOS DE DATOS FLOTANTE.

Este tipo de datos son incluidos debido a la compatitividad de los datos, puesto que el tipo de datos *float*, es equivalente al tipo de datos numéricos.

3.4.7 TIPOS DE DATOS LOGICOS.

Este tipo de datos es conveniente usarlos cuando al usuario se le hace necesario almacenar solo dos tipos de valores, y estos valores solo pueden ser verdaderos .T. o falsos .F., este tipo de datos de datos no acepta otro tipo de valores.

3.4.8 TIPOS DE DATOS NUMERICOS.

Los tipos de datos numéricos son usados para almacenar magnitudes, estos tipos de datos pueden almacenar valores que contengan dígitos entre 0 y 9, y opcionalmente se podrá agregar un signo o una coma para especificar los decimales.

En le tipo de datos numéricos, Visual FoxPro permite almacenar valores expresados en formato hexadecimal para datos numéricos.

3.4.9 TIPOS DE DATOS ENTERO.

A este tipo de datos se lo denomina también Integer, los datos de tipo entero son utilizados valores numéricos no decimales en los que las limitaciones de rendimiento y de almacenamiento de tabla son especialmente importantes.

En las tablas el tipo de campo entero se almacena como un valor binario de 4 *bytes*, por lo que requiere menos memoria que otros tipos de datos numéricos. Al ser un valor binario, no requiere conversión ASCII.

3.4.10 TIPOS DE DATOS GENERAL.

Este tipo de dato o más bien tipo de campo, es muy importante dentro de lo nuevo que Visual FoxPro propone, puesto que aquí se pueden almacenar objetos de tipo OLE, Este tipo de campo abarca una referencia de 10 *bytes* al contenido actual del campo. El campo de tipo general, podrá almacenar una hoja de cálculo, un texto del procesador de textos, o inclusive una imagen, esto no quiere decir que toda la hoja de cálculo, el texto, o la imagen estarán almacenados en este tipo de campo, si no sólo una referencia o un vínculo al objeto OLE que creo estos objetos.

El tamaño de estos campos en la tabla únicamente se puede dimensionar o establecer de acuerdo a la capacidad que presente el disco.

3.4.11 TIPO DE DATOS MEMO.

Este tipo de datos memo igual que el anterior, más bien es un tipo de campo, que sirve para almacenar bloques de datos en la tabla, el campo memo contiene una referencia de diez bytes al contenido del campo memo, sin embargo el tamaño de los memos, solo va a depender de la cantidad de datos que se introduzcan en dicho campo, y la capacidad de estos campos solo se los puede dimensionar de acuerdo con la capacidad del disco.

Los datos que son almacenados en este tipo de campos no se encuentran en la tabla, si no que se almacenan en un archivo distinto, que llevara el mismo nombre que el de la tabla pero con una extensión .FPT, y el dato que se encuentre en la tabla, será el dato de referencia a este archivo.

3.5 MANEJO DE CADENAS EN VISUAL FOXPRO.

En un sistema de estructura compleja la mayoría de los campos de una tabla serán de tipo cadena, o por lo general los campos que son tomados como claves principales, por tal motivo se hace indispensable analizar, lo que con una cadena de caracteres se pude hacer.

En Visual FoxPro, se pueden realizar las siguientes operaciones con las cadenas:

- El operador = es utilizado para compara datos que sean iguales, o que tengan el mismo valor, los datos de tipo carácter son comparados o igualados, caracter por caracter desde la izquierda, hasta que uno de estos caracteres sea distinto o la cadena culmine.

- El operador == es utilizado cuando necesite de una comparación exacta, de los datos de tipo caracter, cuando se utilice el operador == para compara cadenas, las cadenas que se encuentren a cada lado del operador, deberán ser exactamente iguales, deberán tener los mismos caracteres, inclusive los espacios en blanco, para que sean consideradas como iguales.

- El comando $ es un operador de tipo booleano, que es capaz de evaluar si una cadena, esta contenida dentro de otra cadena, y si dicha cadena esta contenida dentro de la otra, el resultado del $ será verdadero (.T.), caso contrario el resultado será (.F.), este operador es capaz de distinguir entre mayúsculas y minúsculas.

- El comando *DTOC( )*, evalúa una expresión de tipo fecha, fecha hora, y la transforma a carácter.

- El comando *DTOS( )*, devuelve una cadena de caracteres de tipo AAAAMMDD, a partir de una expresión de tipo fecha o fecha hora.

- El comando *VAL( )*, devuelve un valor numérico, a partir de una expresión de caracteres.

- El comando *STR( )*, devuelve un carácter equivalente, de una expresión numérica.

- El comando *CHR( )*, devuelve el carácter equivalente a un código ANSI numérico especificado.

- El comando *CHRSAW( )*, determina si un carácter esta o no presente en el buffer del teclado.

- El comando *CHRTRAN( )*, va reemplazando, los caracteres de una cadena, con los caracteres que corresponden a otra cadena.

- El comando *CHRTRANC( )*, Reemplaza cada carácter de una expresión de caracteres que concuerde con un carácter en una segunda expresión de caracteres con el carácter correspondiente de una tercera expresión de caracteres.

- El comando *ALLTRIM( )*, Elimina los espacios en blanco, que se pueden encontrar al principio o al final de una cadena de caracteres.

- El comando *LEFT( ), LEFTC( ),* Devuelven un número especificado de caracteres, a partir de una expresión de caracteres, tomado desde el primer carácter de la izquierda.

- El Comando *LEN( )*, devuelve el número de caracteres, que contendrá una expresión de caracteres.

- El Comando *LENC( ),* devuelve el número de caracteres, que contendrá una expresión de caracteres, o un campo memo.

- El Comando *LTRIM( ),* devuelve una cadena después de eliminar todos los espacios en blanco que se encuentren al inicio de esta.

- El Comando *RTRIM( )*, devuelve una cadena después de eliminar todos los espacios en blanco que se encuentren al final de esta.

Visual FoxPro, tiene un sin número de estos comandos que son utilizados para poder manipular cadenas de caracteres, sin embargo las que se acaban de describir son las mas utilizadas.

3.6 MULTIMEDIA.

Como toda Herramienta Visual, Visual Foxpro también incluye en su estructura y su programación herramientas y comandos para poder manipular los beneficios del hardware actual como es la multimedia.

En Visual Foxpro se pueden encontrar herramientas para manipular imagen, sonido, Vídeo, etc. por medio de la galería de componentes, de donde se pueden arrastrar las herramientas multimedia, hacia el diseñador de formularios, o en su ves hacia el administrador de proyectos.

Las herramientas que Visual Foxpro ofrece son:

Para la imagen, ofrece el *_ImageItem*, el mismo que agrega una clase imagen o establece una propiedad *picture*, que pueden ser archivos cuya extensión corresponda a una imagen, como por ejemplo *.bmp, .jpg, .gif, .ico, .cur, . ani*, etc. .

Para el sonido, ofrece *_SoundItem*, que agrega una clase multimedia, con archivos cuya extensión pueden ser *.wav, o .rmi.*

Para el vídeo, ofrece *_VideoItem*, que también agrega una clase multimedia, naturalmente con un archivo de extensión *.avi.*

En general Visual Foxpro dispone de una biblioteca de clases generalizada llamada *_multimedia.vcx*, que es la responsable de cargar, reporducir, implantar, archivos de sonido y vídeo, esta biblioteca de clases está representada por el objeto *Sound Player*.

3.7 OTRAS ESPECIFICACIONES DE VISUAL FOXPRO.

3.7.1 COSTOS DE VISUAL FOXPRO.

Los costos que Visual FoxPro ofrece al público son *relativamente* altos, puesto que en el paquete está incluido el costo de las bases de datos, debido a que Visual Foxpro es un gestionador de estas y por lo tanto no puede comercializarse en forma aislada.

3.7.2 REQUERIMIENTOS DE HARDWARE.

Visual FoxPro como una herramienta de gestión básicamente no requiere de hardware de gran capacidad, y para sustento de lo dicho a continuación se detallan los requerimientos que esta herramienta necesita para su funcionamiento.

a) **Velocidad de procesamiento.-** Para que Visual FoxPro pueda funcionar se hace necesario disponer de un procesador 80486 o superior.

b) **Sistema Operativo.-** El sistema operativo bajo el cual se van a desenvolver las herramientas es parte esencial, y para que Visual FoxPro pueda ejecutarse es necesario que se tenga instalado Windows 95 o superior y para el caso de Windows NT 3.51 o superior..

c) **Memoria.-** Visual FoxPro para poder trabajar necesita que el ordenador disponga de al menos 16 Mega Bytes de memoria RAM libres.

d) **Espacio.-** El ordenador en el cual se va a instalar Visual FoxPro debe tener un espacio disponible en el disco duro de 100 Megas para instalarlo en su mínima capacidad.

CAPITULO 4

ANÁLISIS DE BORLAND DELPHI

4.1 INTRODUCCION.

Delphi está diseñado por la empresa desarrolladora de software Borland, que originalmente desarrolló un lenguaje de programación llamado Pascal, el mismo que es una herramienta muy poderosa de programación para un entorno de desarrollo bajo DOS.

Borland en vista de que los sistemas operativos fueron mejorando, se sintió en la necesidad de mejorar sus aplicaciones, y es así como pone a disposición del mercado su nuevo lenguaje de programación llamado Delphi, tomando como base de su funcionalidad el sistema operativo denominado WINDOWS.

La primera versión fue el Delphi1, que es algo más que el sucesor del famoso Pascal 7.0 de Borland, este ya posee un compilador mucho mas rápido, y además de potentes herramientas para la creación de aplicaciones visuales, de completas herramientas para la creación y manejo de bases de datos, aplicaciones multimedia, creación de archivos .DLLs, etc.

Borland diseño a Delphi para trabajar con plataformas WINDOWS de 16 ó 32 bits, debido a que resulta un compilador idóneo para realizar potentes aplicaciones con las facilidades de uso y las herramientas que caracterizan a los entornos de programación modernos.

Posteriormente Borland desarrollo Delphi 2, 3, y 4, la portada de este último se la puede observar en la Fig. 20, estas versiones fueron y son considerados como *Two - Way - Tool*, es decir, una herramienta de dos direcciones, debido a que permite crear y desarrollar programas de dos formas, una visual en la pantalla, por medio de las funciones de *Drag & Drop* (Arrastrar y Soltar) y la otra a través de la programación convencional, esto es escribiendo el código. Delphi proporciona la facilidad de utilizar ambas técnicas en forma alternativa o simultanea acorde a las necesidades del programador.

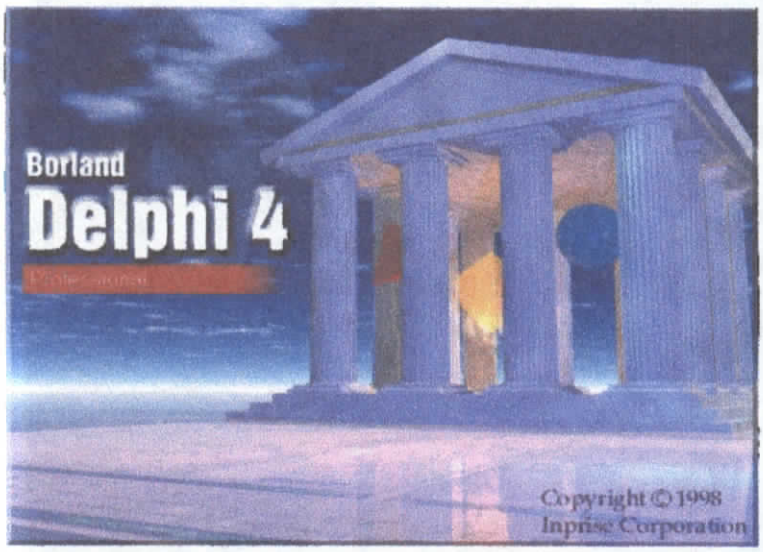


Fig. #20: Portada de Borland Delphi.

4.2 CREACIÓN DE PROYECTOS.

Esta herramienta de programación Visual, al igual que las otras herramientas estudiadas anteriormente, también genera un archivo de proyectos, y a continuación en la Fig. 21 se representa el emplazamiento de Borland Delphi para poder crear proyectos.

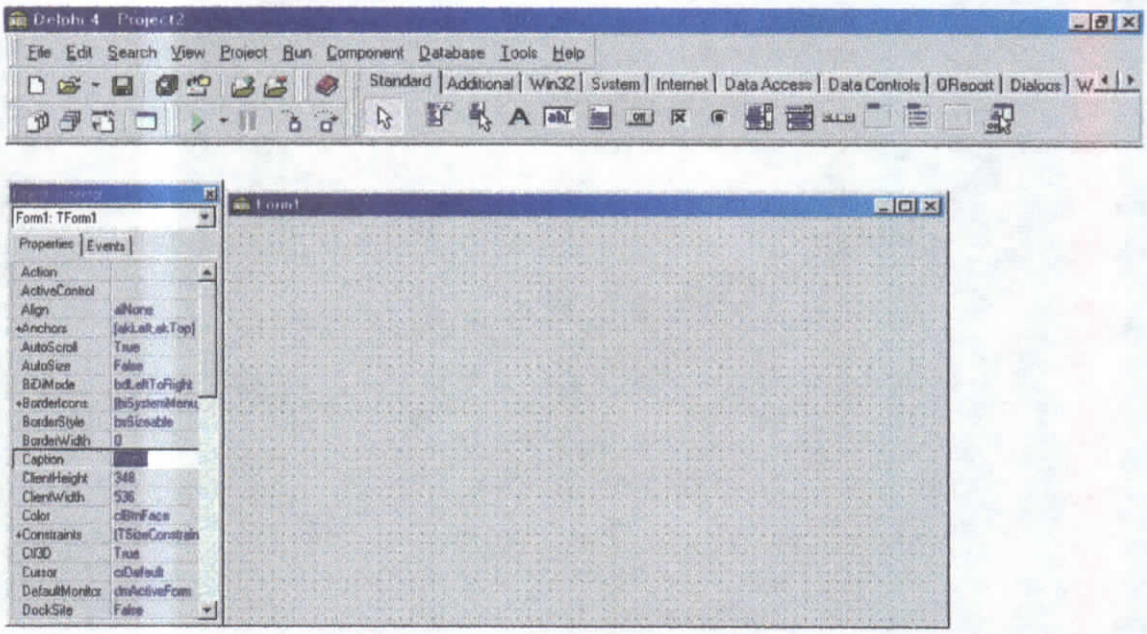


Fig. #21: Emplazamiento de Delphi para poder crear un PROYECTO.

En el instante que DELPHI crea un nuevo proyecto éste también genera un archivo de proyectos, este archivo de proyectos es guardado con una extensión .DPR, y es precisamente el que va a permanecer durante toda la ejecución de la aplicación, o mas bien durante toda la construcción de la aplicación, este archivo con extensión .DPR, solo puede existir o ser creado una sola vez en el proyecto.

En este archivo están contenidos todas las unit (unidades) y sus respectivos códigos que arán que el proyecto funcione.

Cada vez que al proyecto se le adicione una nueva forma esta se adicionará con su respectiva unit como se lo puede observar en la Fig. 22., por consiguiente abarcará su respectivo código de funcionamiento, en ese momento Delphi será capaz de adicionar esta modificación en la cláusula uses del archivo principal del proyecto.

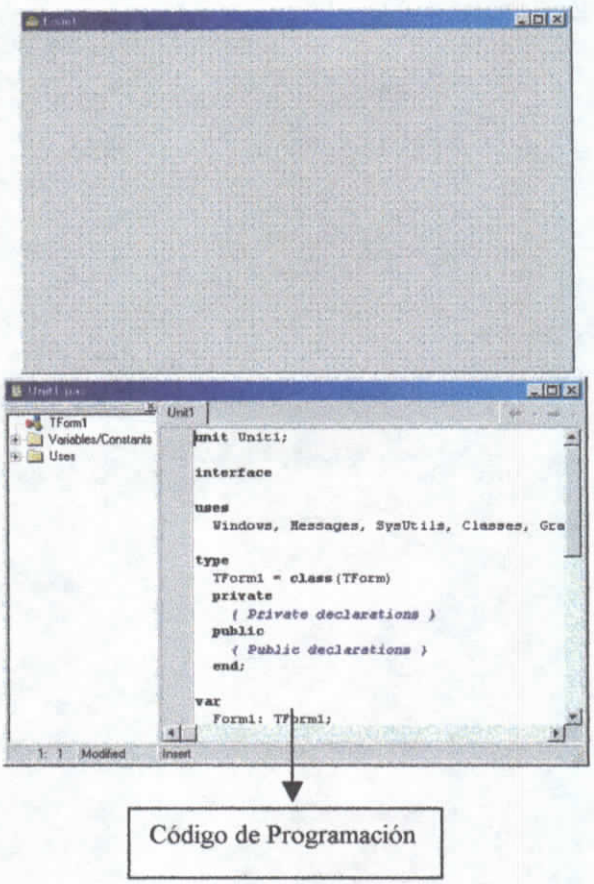


Fig. #22: Forma con su respectiva Unit. lugar en donde se implanta el código de programación para que su respectivo programa funcione.

Es conveniente que los usuarios eviten en lo posible hacer modificaciones manuales del código generado automáticamente, para el archivo principal del proyecto, puesto que Delphi puede ser que no reconozca algunas de las instrucciones usadas, o en su vez que las instrucciones estén mal usadas, pudiendo así causar errores de compilación o cualquier otro tipo de error o inconveniente durante la ejecución del programa.

Delphi dispone de un manipulador de proyectos que precisamente es la ventana que visualiza todas las unidades y formularios de las que el proyecto actual está compuesto.

La ventana manipuladora de los proyectos presentada en la Fig. 24 dispone de algunas características, como por ejemplo:

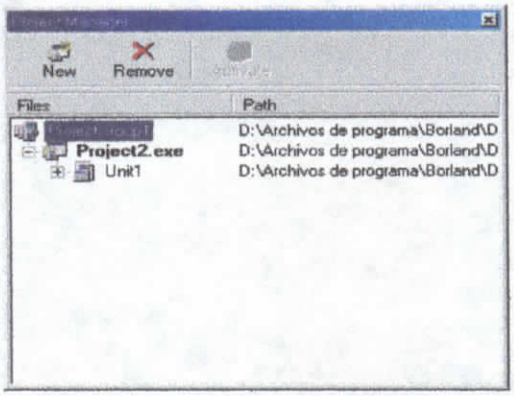


Fig. #23: Ventana por medio de la cual se hace posible manipulador un proyecto.

a) **File Status Line.-** Que muestra el nombre completo de la ruta que contendrá el archivo del proyecto. De la misma forma enlista el nombre de todas las unidades y formularios, que componen el proyecto.

b) **Column headers.-** Con esta opción es posible cambiar el tamaño de los encabezados de las divisiones Unit, Form y Path.

c) **Unit.-** En esta división se enlista el nombre de todas las unidades, y si esta unidad tiene su correspondiente forma, enlista el nombre en la línea correspondiente de enfrente.

d) **Form.-** Esta división enlista, el nombre de las formas, que corresponden al proyecto, el nombre de su respectiva unidad, también es enlistada en la misma línea.

e) **Path.-** En esta división se puede observar el nombre completo de la ruta de acceso a cada una de las unidades, si las unidades o formas se encuentran en el mismo directorio que el directorio del proyecto, esta división aparecerá en blanco.

Para un usuario, es sumamente conveniente utilizar la ventana manipuladora de proyectos (ver Fig. 23), puesto que reduce el tiempo de navegación entre los distintos formularios y unidades integradas en el proyecto. Debido a que este dispone de un sistema de botones de acceso directo, dirigidos a las funciones de uso mas frecuente, como son:

a) **Add unit.-** Adiciona una unidad al proyecto

.

b) **Remove unit.-** Elimina o remueve una unidad del proyecto, esto no quiere decir que esta unidad este eliminada físicamente.

c) **View unit.-** Centra y abre la unidad seleccionada.

d) **Project options.-** Abre una caja de dialogo con opciones que el proyecto dispone, las mismas que serán explicadas mas adelante.

e) **Update.-** Vuelve a sincronizar la información del proyecto, con el contenido de la información del archivo de origen .DPR, este botón solo esta disponible cuando se lo edite de forma manual al archivo .DPR.

4.2.1 OPCIONES DEL PROYECTO.

Delphi como parte de la empresa desarrolladora de software BORLAND, está constituida con las opciones que esta empresa creyó que son las mas necesarias para el desarrollo de proyectos, sin embargo a Delphi se le a incrementado información adicional acerca del proyecto en diversas paginas de dialogo del Project Options mostrada en la Fig. 24 que a continuación se la puede apreciar, y estas son: Forms, application, compiler, linker y directories/conditionals.

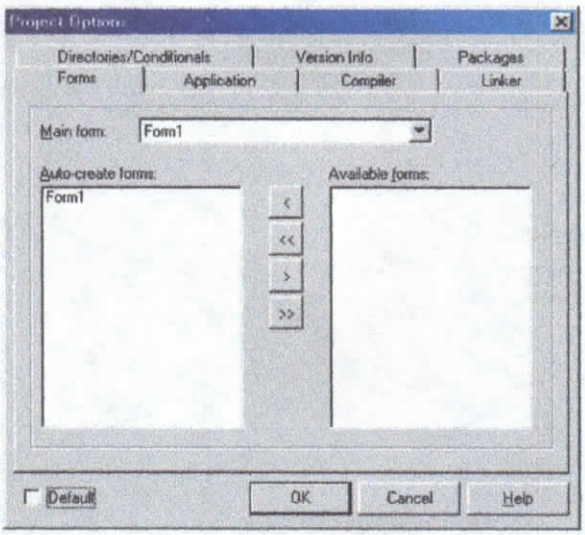


Fig. #24: Ventana que contiene las opciones de proyecto, que permite manipular los programas.

Estas opciones son las que controlan el funcionamiento del proyecto, es decir que formulario aparecerá al inicio del programa, los archivos de ayuda vinculados a este, y el icono de la aplicación, las opciones del vinculador y los directorios del proyecto.

4.2.1.1 Forms.

Esta pagina contiene opciones para manipular los formularios, por ejemplo: Para poder designar cual es el formulario principal del proyecto, en que orden se crearan los formularios, y cual es el que de manera automática se disparara al inicio del proyecto.

Sus parámetros son los siguientes.

a) **Main Form**.- Aquí se mostrara que formulario se visualizara como principal en tiempo de ejecución dentro del proyecto.

b) **Auto-Create Forms.-** Esta opción visualizara todos los formularios que se crearon automaticamente en el proyecto, en tiempo de ejecución.

c) **Available forms**.- Esta opción enlista los formularios del proyecto que no se crean automaticamente en tiempo de ejecución.

4.2.1.2 Application.

En esta pagina el usuario podrá especificar la ruta del archivo de ayuda, el titulo del proyecto y el icono de la aplicación.

Las opciones disponibles en esta pagina son:

a) **Title.-** Esta opción permite especificar el texto que aparecerá como titulo de la aplicación al minimizarla, si esta opción está en blanco, por defecto tomara el nombre del archivo .DPR.

b) **Help File.-** Esta opción le permite especificar que archivo de ayuda esta relacionado o vinculado con la aplicación.

c) **Browse.-** Esta opción presenta un cuadro de dialogo, por medio del que se podrá escoger el archivo de ayuda que se presentara en la aplicación.

d) **Icon.-** Aquí se presentara la figura del icono que se visualizara con la aplicación.

e) **Load Icon.-** Esta opción permitirá escoger el icono que ira relacionado con la aplicación, mediante un cuadro de dialogo que mostrará una variedad de iconos disponibles.

4.2.1.3 Compiler

En este cuadro de dialogo se encontrarán las directivas de compilación, además se mostraran las opciones de cómo manipulará el compilador al proyecto, estas opciones pueden ser omitidas si en el código fuente encuentra unas directivas de compilación.

Esta pagina de opciones está dividida en cinco grupos:

a) Code Generation (Generación de Códigos)

b) Runtime Errors (Errores en tiempos de ejecución)

c) Syntax (Sintaxis)

d) Debugging (Depurador)

e) Messages (Mensages)

f) Code Generation.- Controla el modo de cómo se generan el código de los objetos para el proyecto.

- *Optimization*.- Usa optimizaciones de código de 32 *Bits*.
- *Aligment record files*.- Alinea los campos de registro con los limites de las palabras al activarlas.
- *Stack frames*.- Al estar activa establece el marco para bloqueo antiguo, al estar desactivada permite optimizaciones con pasos de parámetros con estilo mas reciente.
- *Pentium safe FDIV*.- Al estar activa genera código que compensa de manera automática el defecto de división de punto flotante en algunos procesadores pentium.

g) Runtime errors.- Controla que determinadas condiciones generen excepciones en tiempo de ejecución.

- *Range checking*.- En el momento que está activa genera un error cuando los subíndices de disposición y cadena no están dentro de los limites.
- *Stack checking*.- Al activar esta opción se genera un error cuando ocurre un desbordamiento de pilas o un flujo insuficiente de estas.
- *I/O checking*.- Al estar activa esta opción genera un error al detectar fallas de entrada / salida.
- *Overflow checking*.- Cuando se activa esta opción genera un error si se desborda una operación de enteros, esto ocurre cuando una operación da como resultado un número mayor al que la variable resultante puede soportar.

h) Syntax.- Esta opción controlará si la sintaxis con la que esta escrita el código del programa es válida.

- *Strict var-strings*.- Si esta opción esta activa genera un error si no concuerdan los tipos de parámetros de cadena.
- *Complete Boolean eval*.- Cuando se activa esta opción las expresiones boolenas son evaluadas en su totalidad.
- *Extended syntax*.- Cuando se activa esta opción es posible llamar a funciones, como si se trataran de procedimientos, y descartar el resultado de la función, también permite descartar las disposiciones *Char* de base cero, como si fueran variables *Pchar*.
- *Typed @ operator*.- Si está activa esta opción el tipo de puntero que retornara el operador @ es el mismo que aquel al que se apunta; en caso contrario el operador @ retorna un puntero sin tipo.

- *Open parameters*.- Cuando se activa esta opción es posible pasar cadenas de tamaño distinto por el mismo parámetro.
- *Huhe strings*.- Activar esta opción permite usar cadenas de gran tamaño, cuya longitud esta limitada únicamente por la memoria disponible.
- *Assignable typed constants*.- Activar esta opción permite asignaciones a constantes impresas.

i) Debbuging.- Esra opción controla el tipo de información depurada que se incluirá en el archivo objeto .DCU

- *Debug informatio*.- Cuando se agrega esta opción cierto tipo de información de depuración es incluida en el archivo .DCU, como son: número de línea de código para cada mapeo de procedimientos a las direcciones de código de objeto.
- *Local symbol*.- Si esta opción esta activa, añade identificadores de mejoramiento, procedimiento al archivo .DCU, también debe ser verificada la información de depuración para que esta información se lleve a cabo.
- *Symbol info*.- Esta opción al ser activada añade información de referencia de símbolo, que incluye número de línea de todas las declaraciones y referencias a símbolos de la unidad. Para que esta opción funcione deben ser verificadas tanto: *Debug informatio*, como *symbol info*.

j) Messages.- En esta opción se pueden controlar que tipos de mensajes son los que Delphi va a desplegar después de la compilación.

- *Show Hints*.- Si esta opción esta activa presentara una lista de errores no fatales después de la compilación, como por ejemplo una lista de variables no usadas, estos mensajes serán desplegados en la ventana del *code editor*.
- *Show warnings*.- Si se activa esta opción Delphi presentará una lista de precauciones, que en el tiempo de ejecución se pueden convertir en errores.

4.2.1.4 Linker.-

Esta opción contiene información acerca del proyecto como por ejemplo, las relativas que contendrá el archivo de mapa, si se compila o no con la información del archivo. TDW, y si deben o no ser optimizados los archivos .EXE o los archivos .DCU, también se incluirá los tamaños de las memorias para la pila y pila local.

El grupo de opciones *map files* controla si el archivo de mapa se genera al elaborar el proyecto, y que información contendrá.

a) **Off.-** Seleccionada; evita la creación del archivo de mapas.

b) **Segments.-** Seleccionada; genera el archivo con una lista de segmentos, dirección de inicio del programa, y mensajes de vinculación.

c) **Publics.-** Seleccionada; genera un archivo que contenga segments, incluidos una lista de orden alfabético de símbolos públicos.

d) **Detailed.-** Seleccionada; genera un archivo *segments, publics*, adicionado un mapa detallado de segmentos (dirección, bytes, nombre, grupo e información de módulo).

Dentro de esta opción general, también encontramos el grupo de opciones LINKER OUTPUT, que especifica el tipo de archivo que envía al vinculador de disco.

a) **Generate DCUs.-** Seleccionada, el vinculador de Delphi da salida a los archivos .DCU.

b) **Generate Object Files.-** Seleccionada, el vinculador, genera archivos .OBJ.

c) **El grupo de opciones .EXE y .DLL.-** controla si el proyecto es una aplicación de tipo CRT, y si los archivos .EXE ó .DLL, contendrán código de depuración.

d) **Generate console aplication.-** indica si la aplicación se genera como una aplicación de tipo CRT.

e) **Include TDW debug info.-** Esta opción incluye la información de depuración apropiada en el archivo .EXE, y si usted esta depurando con el *debuger* para WINDOWS, esta opción puede ser habilitada.

También Delphi pone a disposición el grupo de opciones MEMORY SIZE, este grupo de opciones, contiene información acerca del tamaño de la pila, y de la memoria de procesamiento.

a) **Min Stack Size.-** Especifica el mínimo del área de memoria donde las variables locales son guardadas, así como las llamadas de función, durante la ejecución del programa.

b) **Max Stack Size.-** Especifica el máximo del área de memoria donde las variables locales son guardadas, así como las llamadas de función durante la ejecución del programa.

c) **Image Base.-** Especifica la dirección de memoria base donde se carga el programa ejecutable, esta dirección es cambiada solo cuando se compila un archivo .DLL.

El grupo de opciones descripción.

a) **EXE Description.-** Especifica, el texto de entrada en la descripción del archivo ejecutable ó .DLL

4.2.1.5 Directores/Conditional.-

Esta página contiene opciones relativas a la localización de archivos del proyecto, direcciones de los archivos de entrada salida del proyecto.

a) **Output directory.-** esta opción permite especificar, la ruta para ciertos archivos del proyecto, como son los archivos .MAP, .DCU y .EXE.

b) **Search Path.-** Aquí se pondrán los caminos, en donde encontrar los archivos fuentes del proyecto, incluyendo los archivos .DPR (código fuente) o los archivos .DFM (formularios), y .RES (recursos).

El grupo de opciones *conditionals* contiene el cuadro combinado condicional define; esta opción permite usar directivas del compilador condicionado, se puede delimitar una lista de condiciones solo por medio del punto y coma.

El grupo de opciones ALIASES.

a) Unit aliases.- Esta opción le permite definir alias para las unidades del proyecto, la lista de los alias está delimitada por un punto y coma.

4.3 ESTUDIO DE LAS BASES DE DATOS EN DELPHI.

El término base de datos es algo muy común en la actualidad, y por lo general se tiene una idea básica de lo que es una base de datos, que no es más que un conjunto de ficheros con características propias que los hacen especiales, más aun con la facilidad y potencialidad que el Borland Delphi ofrece.

Delphi es conocido en el medio de la programación, por su potencialidad en el manejo de las bases de datos, el manejo de las bases de datos en Delphi es responsabilidad exclusiva del Delphi, es decir que no necesita de terceros, o de otros lenguajes de gestión de Bases de Datos

para manejarlas, por este motivo Delphi es el motor de las bases de datos en la Borland, y en la Fig. 25 se puede observara la creación de una tabla en Borlan Delphi.

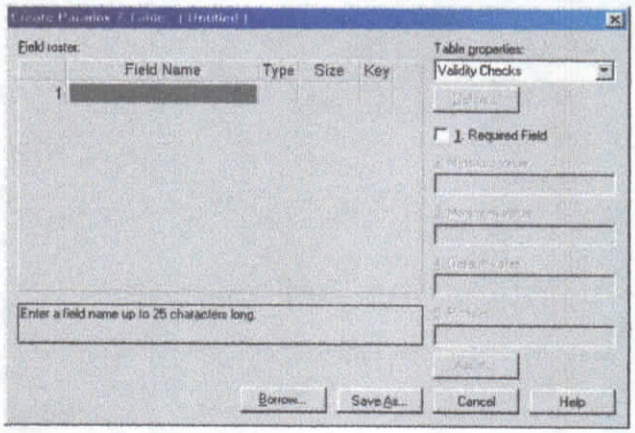


Fig #25: creación de una Tabla en Borland Delphi.

Delphi por defecto gestiona tres tipos de bases de datos, estas bases de datos son de la Borland, pero además de esto Delphi proporciona otro tipo de controladores, para poder manipular o gestionar otros tipos de Bases de Datos.

Los tipos de Bases de Datos que Delphi controla de la Borland son:

- DBASE.
- PARADOX.
- INTERBASE.

Los otros tipos de bases de datos que Delphi controla son:

- MICROSOFT ACCESS

- FOXPRO

Dentro del estudio de Delphi y en particular de las bases de datos, se hace necesario estudiar los ALIAS, los alias, no son otra cosa que la simplificación de un camino, el direccionamiento de archivos, los alias son una de las características que más llamo la atención dentro de las Bases de Datos de Delphi, de tal forma que si los archivos del programa son trasladados a otra dirección solo bastaría con modificar el *path* del *alias* en el administrados de Bases de Datos.

Las versiones mejoradas de Delphi proporcionan también la facilidad de comunicarse con servidores de datos, como son los SQL server.

Como una de las características principales de Delphi es el manejo de las bases de datos, Delphi está en la capacidad de poder manejar bases de datos locales, y bases de datos remotas.

4.3.1 BASES DE DATOS LOCALES.

Las bases de datos locales residen en el disco local, o en alguna área de la red local, estas bases locales tienen propiedades APIs, para permitir a sus usuarios acceder a sus datos. Este tipo de Bases de Datos es dedicado para la construcción y utilización de sistemas simples, cuando estas Bases de Datos sean utilizadas por distintos usuarios, se hace necesario implementar, algunos mecanismos de bloque para la utilización de dichas bases.

Este tipo de Bases de Datos pueden ser más rápidas que las Bases de Datos ubicadas en un servidor de bases de datos remotas, a pesar que estas residen en algunas aplicaciones de sistemas de Bases de Datos locales, las Bases de Datos locales tiene más limitaciones que las bases ubicadas en los servidores de bases remotas.

Algunos ejemplos de bases de datos locales son: Paradox, Dbase, FoxPro, Access

4.3.2 SERVIDOR DE BASES DE DATOS REMOTAS.

Usualmente un servidor de bases de datos remotas reside en una máquina remota, está estructurada por un lenguaje Query (SQL), dispone de datos de acceso para los clientes debido a que estos clientes hacen algunas llamadas a los servidores SQL.

Este tipo de bases de datos también son conocidas con el nombre de RDBMS (Remote Database Managment system), sistema manipulador de bases de datos remotas.

Este tipo de bases de datos no residen en una máquina simple, pero sin embargo es distribuida sobre algunos servidores.

Las aplicaciones que utilizan Bases de Datos remotas son llamadas también son llamadas de doble tarea, o aplicaciones multitarea, debido a que la aplicación y la base de datos operan independientemente del sistema local.

Para que un componente tabla pueda hacer referencia a una tabla específica de una base de datos, es necesario que a dicho componente se le fijen algunas propiedades, tales como: *DatabaseName* y el *TableName*,

.

En la propiedad *DatabaseName* va el nombre de la bases de datos a la cual el componente *table* va a acceder.

La propiedad *TableName*, es el nombre de la tabla en la Base de Datos especificada en la propiedad *DatabaseName*.

Para poder acceder a los datos encapsulados en las tablas ya especificadas, es necesario agregar el componente *DataSource* que Delphi provee, en dicho componente existe la propiedad *DataSet*, a la misma que se le debe dar el nombre de la tabla que contienen los datos a acceder.

La tabla estará lista para poder brindar su contenido solo si la propiedad *Active* de la tabla esta dispuesta a *True*.

4.3.9 ARCHIVOS ÍNDICES EN DELPHI.

Un Índice es un archivo que determina el orden con que se accede a la base de datos, y por ende a los registros de la tabla. Las tablas en Paradox, Dbase, y SQL, utilizan índices para organizar los registros, pero los índices en estas bases de datos, trabajan de una forma diferente.

Delphi tiene índices primarios o secundarios, pero en las tablas de Paradox, se puede llamar también claves de índice primario, Delphi organiza los registros de las tablas acorde con la clave de índice primario, por defecto todos los registros se organizan y se acceden en forma ascendente.

Cuando se define una clave compuesta la Base de Datos crea un archivo índice compuesto, y este archivo índice se organizará por el primer campo de la clave compuesta, posteriormente por el próximo, y así sucesivamente.

En las tablas de Paradox un índice secundario define una vista con un orden alternante produciendo cambio en los registros al momento de desplegar su contenido, cabe destacar que físicamente los registros en la tabla no son alterados ni modificados en su estructura y ubicación, los índices secundarios, también son usados por los SQLs, teniendo de esta forma una velocidad excesiva en cuanto a su acceso se refiere.

Las tablas en Dbase usan un índice organizado por los registros existentes en las tablas acorde con uno o más de sus campos.

Las tablas SQL usan índices únicos y no únicos, pero no usan las claves primarias que utilizan las tablas del paradox, En Delphi se pueden crear múltiples índices con las tablas SQL, por cada índice se debe especificar si son índices únicos o no únicos.

Se puede utilizar el *Data Base Desktop* para poder crear o modificar los índices, pero aquí no se puede especificar cual índice va a ser usado.

4.4 ESTUDIO DE LOS TIPOS DE DATOS EN BORLAN DELPHI.

Borland Delphi a introducido al mercado una nueva e innovadora forma de manejo de tipos de datos, incluso a mejorado los ya existentes convirtiéndolos en datos de 32 bits, tanto así que en la actualidad los datos de tipo cadena han alcanzado una longitud practicamente ilimitada.

Delphi acoplándose a las ventajas que WINDOWS 95 en adelante ofrece, los direccionamientos de memoria de 32 bits, tanto así que una aplicación realizada bajo la plataforma de programación visual Borland Delphi puede acceder hasta 2GB (giga bites) de memoria virtual.

Los tipos de datos nuevos y redefinidos que Borlan Delphi incluye a los tipos de datos que se detallan a continuación: *integer, cardenal, currency, character, string, variant.*

Tabla #4: Tipos de datos en Borland Delphi

TIPO DE DATOS	DESCRIPCIÓN	TAMAÑO	INTERVALO
INTEGER	Tipos de datos enteros.	Datos de 16 bits y 32 bits.	Con 16 bits: 32767 a -32768. Con 32 bits: 2147483647 a -2147483648
CARDINAL	Datos de tipo entero.	Datos de 16 y 32 bits.	De 0 a 65535 con 16 bits. Con 32 bits de 0 a 2147483647.

LongInt	Tipo de dato entero largo	De 32 bits con signo.	De 2147483647 a -2147483648
SmallInt	Tipo de dato entero corto	De 16 bits con signo.	De 32767 a -32768
Word	De tipo numérico	De 16 bits sin signo.	De 0 a 65535
ShortInt	De tipo numérico	De 16 bits con signo.	De 127 a -128
Byte	Dato de tipo numérico	De 8 bits sin signo.	De 0 a 255
CURRENCY	Dato de tipo monetario	De 64 bits.	Extención de datos de punto flotante
CHARACTER	Datos de tipo carácter	Datos de 16 bits.	
STRING	Datos de tipo cadena	Compuesta por caracteres char (16 bits).	Disponibilidad de memoria.
VARIANT	Cualquier tipo de datos	Datos de 16 bits.	Disponibilidad de memoria.

4.4.1 TIPOS DE DATOS INTEGER, CARDINAL.

Los tipos de datos Integer y Cardinal son definidos en la actualidad por las personas que programan con Borland Delphi como tipos de datos genéricos, debido a que manejan variables de 16 bits para aplicaciones de 16 bits y variables de 32 bits para aplicaciones de 32 bits.

Los datos de tipo *integer* para aplicaciones de 16 bits tienen un límite de 32767 a -32768 y de 2147483647 a -2147483648 para aplicaciones realizadas a 32 bits .

Los tipos de datos *cardinal*, están en el límite de 0 a 65535 para aplicaciones de 16 bits, pero para aplicaciones de 32 bits entran en el rango de 0 a 2147483647.

Además de estos tipos genéricos existen cinco tipos fundamentales de enteros cuyo significado está definido de manera estática, estos son:

a) **LongInt**: Numérico de 32 bits con signo, entra en el límite de -2147483648 a 2147483647.

b) **SmallInt:** Numérico de 16 bits con signo, está en el límite de -32768 a 32767.

c) **Word:** Numérico de 16 bits con signo, en el límite de 0 a 65535.

d) **ShortInt:** Numérico de 8 bits con signo, en el límite de -128 a 127.

e) Byte: Numérico de 8 bits sin signo, en el límite de 0 a 255.

4.4.2 TIPOS DE DATOS CURRENCY.

Borland Delphi definió un nuevo tipo de dato llamado *currency* o monetario, con el único propósito de incrementar la precisión en los cálculos financieros, dependiendo de las matemáticas del tipo decimal codificado a binario. Este es compatible con los tipos de punto flotante, aunque de echo se almacena como entero de 64 bits cuyos últimos cuatro dígitos representan la parte decimal del valor.

4.4.3 TIPOS DE DATOS CHARACTER.

Borland Delphi adopto nuevos datos de tipo carácter, con la finalidad de manejar *Unicode*, es la norma que define como se codifican los caracteres internacionales, requiriendo representaciones de 16 *bits*, en lugar de la anterior representación de 8 *bits*.

ANSIChar: Es una representación de caracteres de 8 *bits* (1 byte); su apuntador es PANSIChar. Este tipo se usa para el manejo de caracteres ANSI. Este es el mismo tipo que el ya existente Char de 8 bits.

WideChar: Es una representación de carácter de 16 *bits* (2 bytes); su apuntador es el PwideChar. Este tipo se puede utilizar para el manejo de caracteres Unicode de 16 *bits*, empleando ambos *bytes*, pero también es posible usarlo para manejar caracteres ANSI de 8 *bits*, asignandolo cero al *byte* de orden superior.

Delphi tomando en cuenta que las variables de tipo carácter pueden ser de 8 a16 *bits*, pone a disposición de los usuarios la función SizeOf para determinar el tamaño del carácter en cuestión, para evitar suposiciones con respecto al tamaño de su código.

4.4.4 TIPOS DE DATOS STRING.

Delphi ha cambiado el significado de la palabra reservada *String* (cadena), ahora este tipo de dato está condicionado por el estado de la directiva ($H) del compilador. Este estado de compilación en su forma predeterminada ($H+), representa una cadena larga con terminación nula, cuya longitud solo está limitada por la memoria disponible. Si se configura como ($H-), representa a una cadena corta de hasta 255 caracteres.

ShortString: Una cadena corta compuesta por caracteres *char*, cuyo bite cero contiene la longitud de la cadena, y puede ser de máximo 255 caracteres.

AnsiString: Una cadena larga de terminación nula compuesta por caracteres AnsiChar, que pueden ser de longitud ilimitada, En Delphi este es el tipo de datos predeterminado que emplean todos los componentes VCL.

4.4.5 TIPOS DE DATOS VARIANT.

Borland Delphi con el fin de permitir a los desarrolladores de software escribir código capaz de manejar parámetros de tipo desconocido, como resultado de operaciones de automatización OLE, se implemento el nuevo tipo de datos *Variant*..

Este tipo de datos asigna una estructura de 16 *bits*, la cual es capaz de representar cualquier tipo de datos Pascal, con la única excepción del tipo de dato de registro. Aunque es posible asignar con variantes, si está activa la verificación de límite se creará una excepción cuando el valor resultante esté fuera de alcance.

Debido a que el tipo real de la variante se desconoce antes de la ejecución, las variables de este tipo tienen la capacidad integrada de promover el valor que contienen, a un tipo de dato adecuado, cuando se incluyan en las líneas de código las asignaciones u operadores.

En Borland Delphi con el nuevo tipo de datos variant es posible promover un valor de tipo *Integer* a una variable de tipo *Float* un valor de tipo *flotante* a una cadena con el fin de terminar la operación asignada. Si el tipo variante no puede promover el valor a un tipo compatible con el resultado de la asignación u operación, se generara una excepción.

4.5 MANEJO DE CADENAS.

Las cadenas en su estructura presentan muy diversas formas, por lo cual se convierte y es vital saberlas manejar adecuadamente, para poder tener una correcta escritura de código en las aplicaciones.

Es necesario comprender que las cadenas están compuestas por un grupo de caracteres almacenados con un arreglo específico, las cadenas por su estructura en un programa pueden representar: Texto, encabezados, títulos y nombres; es decir a través de estas se podrán

representar todos los mensajes convenientes para poder tener una mejor comunicación con el usuario.

En Borland Delphi existen cadenas de terminación nulas, en donde la longitud de la cadena está determinada por la posición donde aparece el primer caracter nulo en el arreglo, el caracter nulo representado de la siguiente manera, (#0).

Borland Delphi en su estructura también presenta las cadenas de nombre contadas, en donde el byte cero del arreglo de caracteres llamado también byte de longitud, contiene el número que representa la longitud de la cadena.

El manejo de cadenas en una aplicación es importante de tal forma que el estudio se enfocara en el manejo de cadenas cortas, largas, y conversión de cadenas.

En Delphi una cadena larga es aquella que tiene una terminación nula, y está representada por medio de un arreglo de caracteres, sin embargo la memoria de cadenas largas es asignada de una forma dinámica, cabe destacar que el arreglo contienen los caracteres que forman la cadena, indicando al final de este arreglo el carácter nulo (#0) que representa la terminación de esta cadena.

En Delphi las cadenas largas pueden ser de longitud prácticamente ilimitada, este tipo de cadena funciona mientras este activa la directiva predeterminada de manejo de cadenas del compilador, ($H+), en Delphi existen dos tipos de cadenas largas el ANSIString y el WideString.

El ANSIString es el tipo de cadenas predeterminadas que utilizan los VCL, y está compuesta por caracteres ANSIChar, los cuales tiene 1 *byte* de longitud por cada uno, en Delphi al declarar una cadena de este tipo no se asigna memoria, si no hasta que se especifiquen los caracteres de la cadena. Luego de esto el usuario puede acceder a los caracteres de la cadena como si se tratara de un arreglo en ANSIChar, es decir carácter por carácter.

El WideString es una cadena compuesta por caracteres WideChar, los mismos que tiene dos bytes de longitud, y su terminación esta dispuesta por con dos bytes de longitud nula (#0), este tipo de cadena tiene una asignación dinámica, al igual que el tipo ANSIString descrito anteriormente.

En el manejo de cadenas largas, Delphi brinda la facilidad de poder utilizar las funciones *SetLength* y *SetString*, las cuales asignan memoria, debido a que el único límite para la longitud de las cadenas largas es la disponibilidad de memoria.

- La función *SetLength* permite especificar la longitud de una cadena larga y reasignar memoria. Si la cadena es mas larga que la longitud asignada, esta se truncará, pero si la longitud es mayor a la cadena asignada el espacio extra contendrá lo que en ese momento hubiera en memoria, es decir caracteres nulos.

- La función *SetString* permite copiar caracteres de una cadena *PChar* a una cadena larga, si la longitud especificada es mayor a la de la cadena que apunta *Pchar*, el espacio adicional se completara con caracteres nulos.

En Delphi a las cadenas cortas se les conoce también como cadenas contadas o de longitud de *byte*, y se define como un arreglo de caracteres Char con un máximo de 256 bytes asignados de manera estática.

En Delphi este tipo de cadenas el *byte* cero contiene un carácter cuyo equivalente decimal ASCII corresponde a la longitud de la cadena, por tanto los caracteres que comprenden la cadena serán de máximo 255 caracteres.

El tipo de datos que corresponde a esta clase de cadenas es *ShortString*, con caracteres de tipo *char*. Si la directiva de manejo de cadenas del compilador es (SH-), la palabra reservada *String*, también indica una *ShortString*.

En delphi podemos encontrar funciones que van a facilitar el manipuleo de cadenas, tales funciones pueden ser:

- *Length* función por medio de la cual retorna el número de caracteres de una cadena, o el número de elementos de un arreglo.

- *Concat* por medio de esta función se pueden concatenar dos o mas cadenas en una.

- La función + en Delphi puede concatenar cadenas o caracteres en una sola variable, y si la capacidad de la variable es menor que la cadena concatenada el resultado será truncado.

- De la misma forma las funciones =, >, <, <>, >=, <=, son utilizadas para comparar caracteres de un arreglo o cadenas completas.

En muchas ocasiones será necesario convertir un arreglo de caracteres en números o viceversa, para lo cual delphi proporciona a los usuarios funciones que se detallan a continuación.

- *IntToStr* Función que toma un número entero y lo convierte a una cadena que contenga la representación decimal del número entero.

- *FloatToStr* Toma un número de punto flotante y lo convierte en una cadena que contenga la representación decimal del número de punto flotante.

- *StrToInt* Esta función toma una cadena y la convierte en un entero, pero si la cadena a convertir no tuviera una representación decimal, la conversión fallaría y se generaría un error.

- *StrToFloat* Toma una cadena y a sus caracteres los convierte en un número de punto flotante, si la cadena no tuviera una representación decimal de un valor de punto flotante la conversión fallaría y se generaría un error.

4.6 MULTIMEDIA.

Delphi es una herramienta de programación visual muy versátil en cuanto a la manipulación de multimedia se refiere.

Borland Delphi para facilidad de sus usuarios presenta el icono denominado *MediaPlayer*, el mismo que va a contener los controles como: inicio pausa, adelante, atrás, stop. Que son los controles que administran la *Media Control Interface*.

Otro de los controles que Delphi proporciona es el *PaintBox* el mismo que proporciona a los usuarios una superficie de dibujo rectangular.

Otra utilidad que también es ofrecida por esta herramienta de programación es el icono *Image*, que presenta un mapa de bits, un icono o un meta archivo cuyo tamaño puede ser modificado de acuerdo a las necesidades del programador.

Estos controles para manipular la multimedia son fáciles de comprender y programar puesto que Borlan Delphi brinda propiedades para todos y cada uno de sus componentes.

4.7 OTRAS ESPECIFICACIONES DE BORLAND DELPHI.

4.7.1 COSTOS DE BORLAND DELPHI.

Los costos de Borland Delphi son dependientes de que si se adquiere el paquete de forma independiente o con inclusión del motor de bases de datos.

4.7.2 REQUERIMIENTOS DE HARDWARE.

Borland Delphi como una gran herramienta de gestión básicamente y como motor de Bases de Datos de la Borland requiere de hardware de gran capacidad, y para sustento de lo dicho a continuación se detallan los requerimientos que esta herramienta necesita para su funcionamiento.

a) **Velocidad de procesamiento**.- Para que Borlan Delphi pueda funcionar se hace necesario disponer de un procesador pentium de 95 MHz. o superior.

b) **Sistema Operativo**.- El sistema operativo bajo el cual se van a desenvolver las herramientas es parte esencial, y para que Borlan Delphi pueda ejecutarse es necesario que se tenga instalado Windows 95 o superior y para el caso de Windows NT 3.51 o superior.

c) **Memoria.-** Borlan Delphi para poder trabajar necesita que el ordenador disponga de al menos 16 Mega Bytes de memoria RAM libres.

d) **Espacio**.- El ordenador en el cual se va a instalar Borland Delphi debe tener un espacio disponible en el disco duro de 101 Megas para instalarlo en su mínima capacidad.

CAPITULO 5

INGENIERIA DE BASES DE DATOS

5.1 TEMA I

5.1.1 ASPECTOS INTRODUCTORIOS

Contenido:

5.1.1.1 Surgimiento histórico de las bases de datos integradas.

5.1.1.2 Objetivos de los SBD.

5.1.1.3 Arquitectura de un sistema de base de datos.

5.1.1.1 Surgimiento histórico de las bases de datos integradas.

Al estudiar el desarrollo del procesamiento automatizado de datos, en lo que se refiere al aseguramiento técnico, se habla de diferentes generaciones.

Desde el punto de vista del aseguramiento matemático y en particular, el aseguramiento de programas, algunos autores reconocen 3 generaciones:

a) Solución de tareas aisladas.

b) Integración de tareas aisladas en sistemas particulares.

c) Integración de sistemas particulares en sistemas automatizados de dirección.

Este proceso de integración ocurre paralelamente, aunque no simultáneamente, en dos esferas:

a) Integración de los programas.

b) Integración de los datos.

a) Ha estado facilitada por el uso de lenguajes de programación cada vez más sofisticados y de redactores que permiten el acoplamiento de módulos escritos en lenguajes diferentes.

b) En la integración de los datos se han producido tres categorías de técnicas para su manipulación:

b.1 Sistemas orientados a los dispositivos.- Programas y ficheros son diseñados y empleados de acuerdo a las características físicas de la unidad central y los periféricos. Cada programa está altamente interconectado con sus ficheros, por lo que la integración de datos de diferentes sistemas es imposible prácticamente.

b.2 Sistemas orientados a los ficheros.- La lógica de los programas depende de las técnicas de organización de los ficheros (secuencial, directo, etc.). Cada usuario organiza su fichero de acuerdo con sus necesidades y las relaciones entre los elementos se establecen a través de los programas de aplicación.

Esta forma de trabajo implica redundancia de datos que trae aparejada mayor gasto de memoria y complica las operaciones de actualización (modificar un dato donde

quiera que aparezca). Esto aumenta el tiempo de tratamiento y atenta contra la integridad de la información; entendiéndose por integridad a que en todo momento los datos almacenados estén correctos en correspondencia con la realidad.

Además, en la vida real se establecen relaciones entre los objetos que son muy difíciles de representar u obtener a partir de sistemas tradicionales de ficheros. Por ejemplo, si se tiene información sobre trabajadores y estudiantes de una facultad, las aplicaciones requeridas van a definir la manera de organizar y estructurar los ficheros.

Si se desea obtener datos como:

- Calificaciones promedio de cada estudiante.
- Listado de estudiantes por grupos.
- Categoría científica y docente de cada profesor.
- Salario de cada profesor.

Resulta adecuado establecer dos ficheros: uno de profesores y uno de estudiantes.

Qué ocurre si se quiere establecer vínculos entre los profesores y estudiantes? Por ejemplo, si se desea obtener:

- Los estudiantes de un profesor.
- Los profesores de un estudiante.

Se estructuraría un fichero de profesores y estudiantes que resolvería algunas demandas, pero sería ineficiente para otras.

Entonces, es posible representar de manera eficiente utilizando los medios de cómputo, los fenómenos o procesos de la realidad objetiva, aunque sea por supuesto de forma esquemática, pero en la que se establezcan determinados vínculos entre los elementos u objetos que forman parte de esos procesos o fenómenos?

Veremos que esto es posible hacerlo a través de la utilización de bases de datos (BD) y de los sistemas de gestión de bases de datos (SGBD) que dirigen su manipulación.

b.3 Sistemas orientados a BD.- En los que hay una débil interdependencia entre los programas de aplicación y la organización física de los datos. Para poder explicar de una forma clara lo concerniente a sistemas orientados a Bases de Datos, es necesario explicar que Base de Datos es un conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una BD puede considerarse una colección de datos variables en el tiempo.

El software que permite la utilización y/o la actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista y a la vez, se denomina *sistema de gestión de bases de datos* (SGBD).

Es importante diferenciar los términos BD *Bases de Datos* y SGBD *Sistema de Gestión de Bases de Datos.*

El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manipular en términos abstractos los datos, o sea, de forma que no le sea necesario conocer el modo de almacenamiento de los datos en la computadora, ni el método de acceso empleado.

Los programas de aplicación operan sobre los datos almacenados en la base utilizando las facilidades que brindan los SGBD, los que, en la mayoría de los casos, poseen lenguajes especiales de manipulación de la información que facilitan el trabajo de los usuarios.

5.1.1.2 Objetivos de los SBD

Existen muchas formas de organizar las bases de datos, pero hay un conjunto de objetivos generales que deben cumplir todas los SGBD, de modo que faciliten el proceso de diseño de aplicaciones y que los tratamientos sean más eficientes y rápidos, dando la mayor flexibilidad posible a los usuarios.

Los objetivos fundamentales de los SBD son:

a) **Independencia de los datos y los programas de aplicación.-** Ya vimos que con ficheros tradicionales la lógica de la aplicación contempla la organización de los ficheros y el método de acceso. Por ejemplo, si por razones de eficiencia se utiliza un fichero secuencial indexado, el programa de aplicaciones debe considerar la

existencia de los índices y la secuencia del fichero. Entonces es imposible modificar la estructura de almacenamiento o la estrategia de acceso sin afectar el programa de aplicación. Naturalmente, lo que se afecta en el programa son las partes de éste que tratan los ficheros, lo que es ajeno al problema real que el programa de aplicación necesita resolver. En un SBD sería indeseable la existencia de aplicaciones y datos dependientes entre sí, por dos razones fundamentales:

a.1 Diferentes aplicaciones necesitarán diferentes aspectos de los mismos datos (decimal o binario).

a.2 Se debe poder modificar la estructura de almacenamiento o el método de acceso según los cambios en el fenómeno o proceso de la realidad sin necesidad de modificar los programas de aplicación (también para buscar mayor eficiencia).

Independencia de los datos: inmunidad de las aplicaciones a los cambios en la estructura de almacenamiento y en la estrategia de acceso y constituye el objetivo fundamental de los SBD.

b) **Minimización de la redundancia.-** Habíamos visto cómo, con los ficheros tradicionales, se produce redundancia de la información. Uno de los objetivos de los SBD es minimizar la redundancia de los datos. Se dice disminuir la redundancia, no eliminarla, pues aunque se definen las BD como no redundantes, en realidad existe redundancia en un grado no significativo para disminuir el tiempo de acceso a los

datos o para simplificar el método de direccionado. Lo que se trata de lograr es la eliminación de la redundancia superflua.

c) **Integración y sincronización de las bases de datos.-** La integración consiste en garantizar una respuesta a los requerimientos de diferentes aspectos de los mismos datos por diferentes usuarios, de forma que, aunque el sistema almacene la información con cierta estructura y cierto tipo de representación, debe garantizar entregar al programa de aplicación datos que solicita y en la forma en que lo solicita.

Está vinculada a la sincronización, que consiste en la necesidad de garantizar el acceso múltiple y simultáneo a la BD, de modo que los datos puedan ser compartidos por diferentes usuarios a la vez. Están relacionadas, ya que lo usual es que diferentes usuarios trabajen con diferentes enfoques y requieran los mismos datos, pero desde diferentes puntos de vista.

d) **Integridad de los datos.-** Consiste en garantizar la no contradicción entre los datos almacenados de modo que, en cualquier momento del tiempo, los datos almacenados sean correctos, es decir, que no se detecte inconsistencia entre los datos. Está relacionada con la minimización de redundancia, ya que es más fácil garantizar la integridad si se elimina la redundancia.

e) **Seguridad y protección de los datos.-** La protección es garantizar el acceso autorizado a los datos, de forma tal que interrumpa cualquier intento de acceso no autorizado, ya sea por error del usuario o por mala intención, en cuanto que

seguridad es donde que el sistema de bases de datos disponga de métodos que garanticen la restauración de las BD al producirse alguna falla técnica, interrupción de la energía eléctrica, etc.

f) **Facilidad de manipulación de la información.-** Los usuarios de una BD pueden referirse a ella con las solicitudes para resolver muchos problemas diferentes. El SBD debe contar con la capacidad de una búsqueda rápida por diferentes criterios, permitir que los usuarios planteen sus demandas de una forma simple, aislándolo de las complejidades del tratamiento de los ficheros y del direccionamiento de los datos.

Los SBD actuales brindan lenguajes de alto nivel con diferentes grados de facilidad para el usuario no programador que garantizan este objetivo, los llamados sublenguajes de datos.

g) **Control centralizado.-** Uno de los objetivos más importantes de los SBD es garantizar el control centralizado de la información. Permite controlar de manera sistemática y única los datos que se almacenan en la BD, así como el acceso a ella.

Lo anterior implica que debe existir una persona o conjunto de personas que tenga la responsabilidad de los datos operacionales: el administrador de la BD, que puede considerarse parte integrante del SBD. Entre las tareas del administrador de la BD está :

- decidir el contenido informativo de la BD

- decidir la estructura de almacenamiento y la estrategia de acceso
- garantizar el enlace con los usuarios
- definir los chequeos de autorización y procedimientos de validación
- definir la estrategia de reorganización de las BD para aumentar la eficiencia del sistema

Existen otros objetivos que deben cumplir los SBD que en muchos casos dependen de las condiciones o requerimientos específicos de utilización del sistema.

5.1.1.3 Arquitectura de un SBD

Presentaremos a continuación la arquitectura de un SBD, aunque no podemos asegurar que cualquier SBD se corresponda exactamente con ella. Sin embargo, esta arquitectura se corresponde suficientemente bien con un gran número se sistemas. Además, está de acuerdo con la arquitectura propuesta por el grupo ANSI/SPARC.

La arquitectura se divide en tres niveles generales: interno, lógico global y externo.

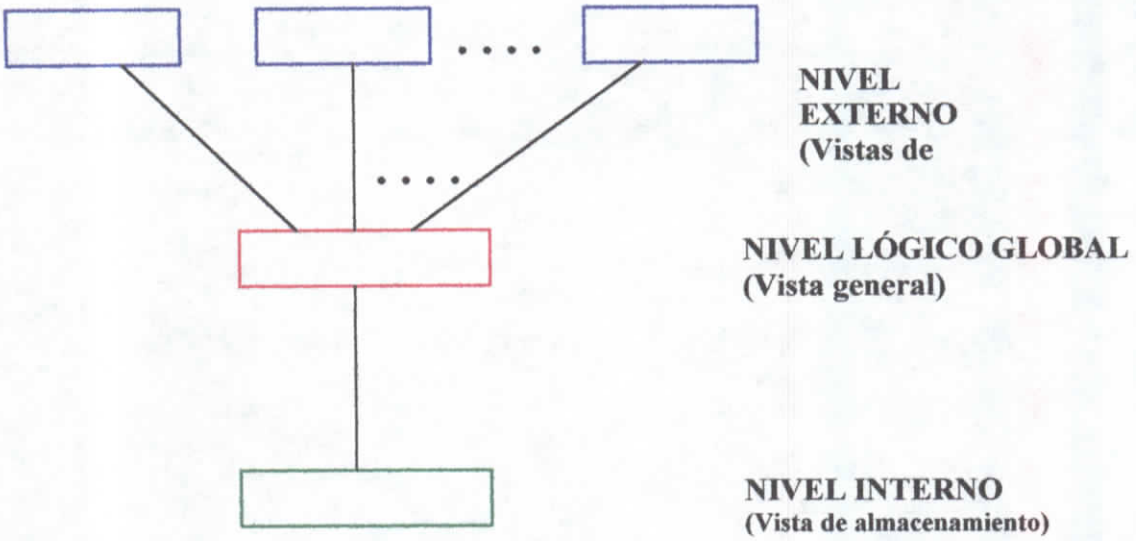


Fig. #26: Figura jerárquica representativa de la división de la arquitectura de las bases de datos

El nivel interno es el más cercano al almacenamiento físico, es decir es el relacionado con la forma en que los datos están realmente almacenados.

El nivel externo es el más cercano a los usuarios, es decir es el relacionado con la forma en que los datos son vistos por cada usuario individualmente.

Al *nivel lógico global* es considerado como un nivel intermedio entre los *niveles interno* y *externo* respectivamente.

Existirán varias *vistas externas* diferentes, siendo cada una de estas una representación más o menos abstracta de alguna porción de la BD total y existirá únicamente una *vista general*, consistente en una representación también abstracta de la BD en su totalidad. Igualmente, existirá una única *vista interna* que representa a la BD completa, tal y como está realmente almacenada.

A continuación estudiaremos con mayor detalle cada uno de los niveles de la arquitectura vista anteriormente y la forma en que ellos interactúan.

a) **El nivel externo.-** Es el nivel del usuario individual, donde un usuario puede ser bien un programador de aplicación o un usuario final con cualquier grado de sofisticación. Cada usuario tiene un lenguaje a su disposición:

   Para el programador, ese lenguaje será bien un lenguaje de programación convencional, tal como Pascal o C, ó bien un lenguaje de programación específico de un sistema, tal como el FoxPro.

   Para el usuario final, el lenguaje será bien un lenguaje de consulta (interrogaciones, query) o un lenguaje de propósito especial, quizás basado en sistemas de menús y construido para satisfacer los requerimientos de un usuario, encontrándose soportado por algún programa de aplicación en línea.

   Es importante señalar que todo lenguaje incluirá un sublenguaje de datos, tal que un subconjunto del lenguaje que trata específicamente con los objetos de la Base de Datos y sus operaciones. Se dice que el sublenguaje de datos (DSL) está embebido dentro del correspondiente lenguaje huésped. Este lenguaje huésped es el encargado de asegurar otras facilidades ajenas a la base de datos, tales como variables locales, operaciones de cálculo, lógica *if-then-else*, etc. Un sistema dado, puede soportar múltiples lenguajes huésped y múltiples sublenguajes de datos.

En principio, cualquier sublenguaje de datos es realmente una combinación de, al menos, dos lenguajes subordinados: un lenguaje de definición de datos (DDL), el cual garantiza la definición o descripción de los objetos de la base de datos, y un lenguaje de manipulación de datos (DML), el que garantiza la manipulación o procesamiento de esos objetos.

Ya se ha indicado que un usuario individual estará generalmente interesado sólo en cierta porción de la BD completa. Aún más, la vista de esa porción será generalmente abstracta cuando se compara con la forma en que los datos están físicamente almacenados. El término definido por el comité ANSI/SPARC para una vista de un usuario es vista externa, la cual es el contenido de la BD como es vista por un usuario en particular. O sea, para ese usuario, la vista externa es la BD.

En general una vista externa consiste en múltiples ocurrencias de múltiples tipos de artículos externos. Un articulo externo no es necesariamente igual a un articulo almacenado.

El sublenguaje de datos del usuario se define en términos de artículos externos; por ejemplo, una operación del DML que sea recuperar artículos, recuperará una ocurrencia de artículos externos y no una ocurrencia de artículos almacenados.

Cada vista externa se define mediante un esquema externo, consistente, básicamente en definiciones de cada uno de los diferentes tipos de artículos externos en esa vista. El esquema externo se escribe usando la porción del DDL del sublenguaje de datos

del usuario, además tiene que existir una definición de la correspondencia entre el esquema externo y el esquema lógico global.

b) **El nivel lógico global.-** La vista lógica es una representación del contenido informativo total de la BD. Es una forma abstracta en comparación con la forma en que los datos están almacenados físicamente. Esta vista puede ser bien diferente de la forma en la que los datos son vistos por un usuario en particular. La vista lógica pretende ser una vista de los datos tal como son, en lugar de cómo los usuarios están forzados a verlos por las restricciones, digamos, de un lenguaje particular o de un determinado hardware que utilicen.

La vista lógica consiste en múltiples ocurrencias de múltiples tipos de artículos lógicos. Por ejemplo, puede ser una colección de ocurrencias de artículos de departamentos, más una colección de ocurrencia de artículos de empleados, etc. Un artículo lógico no es necesariamente igual a un artículo externo ni a un artículo almacenado.

La vista lógica se define mediante el esquema lógico que incluye las definiciones de cada uno de los diferentes tipos de artículos lógicos. El esquema lógico se describe usando otro lenguaje de definición de datos: el DDL lógico. Si se desea lograr la independencia de los datos, entonces las definiciones del DDL lógico no deben comprender ninguna consideración sobre la estructura de almacenamiento ni la estrategia de acceso. Ellas tienen que ser definiciones sólo referentes al contenido informativo.

Si el esquema lógico logra verdaderamente la independencia de los datos, entonces los esquemas externos que se definen sobre el esquema lógico lograrán también, necesariamente, la independencia de los datos.

La vista lógica es entonces una vista del contenido total de la BD y el esquema lógico es una definición de esa vista. Sin embargo, el esquema lógico no es simplemente un conjunto de definiciones como las que se encuentran, por ejemplo, en un programa Pascal. Las definiciones en el esquema lógico deben incluir una gran cantidad de aspectos adicionales, tales como los chequeos de protección y los chequeos de integridad.

En la mayoría de los sistemas actuales, el esquema lógico es realmente sólo un poco más que la simple unión de todos los esquemas externos individuales, posiblemente con la adición de algunos chequeos simples de protección e integridad. Sin embargo, está claro que los sistemas del futuro soportarán un nivel lógico mucho más sofisticado, que permita también describir la forma en que se usan los datos, cómo fluyen de un punto a otro, para qué se usan en cada punto, a qué controles son sometidos, etc.

c) **El nivel interno.-** La vista interna es una representación de bajo nivel de la BD completa, que consiste en múltiples ocurrencias de múltiples tipos de artículos internos.

Artículo interno es el término definido por ANSI/SPARC para la construcción que hasta ahora hemos llamado artículo almacenado. La vista interna está entonces aún a un paso del nivel físico, ya que ella no opera en término de artículos físicos (también llamados páginas o bloques) ni con consideraciones específicas de los equipos, tales como tamaños de sectores o pistas. Básicamente, la vista interna asume un espacio de dirección lineal infinita. Los detalles de cómo se hace corresponder ese espacio con el almacenamiento físico son muy específicos de un sistema y deliberadamente se omitieron de la arquitectura.

La vista interna se describe mediante el esquema interno, el cual no sólo define los diferentes tipos de artículos almacenados, sino que también especifica los índices que existen, la representación de las campos almacenados, la secuencia física en que están los artículos almacenados, etc. El esquema interno se describe usando otro lenguaje de definición de datos: el DDL interno.

En el esquema presentado de la arquitectura de un SBD, se observan los niveles de correspondencias , una entre los niveles externo y lógico global y otra entre los niveles lógico global e interno.

La correspondencia lógica/interna especifica la forma en que los artículos y campos lógicos se representan en el nivel interno. Si se cambia la estructura de la vista interna, o sea, si se hace un cambio en el esquema interno, entonces la correspondencia lógica/interna tiene también que cambiar en consecuencia, de modo que el esquema lógico permanezca invariable. En otras palabras, los efectos de estos

cambios deben ser aislados por debajo del nivel lógico para que se mantenga la independencia datos.

Existe también una correspondencia externo/lógica entre cada vista externa particular y la vista lógica.

Las diferencias que pueden existir entre estos dos niveles son similares a las que pueden existir entre las vistas lógica y la interna. Por ejemplo, los campos pueden tener diferentes tipos de datos, se pueden cambiar los nombres de artículos y campos, múltiples campos lógicos pueden ser combinados en un único campo externo, etc. Puede existir al mismo tiempo cualquier cantidad de vistas externas; cualquier cantidad de usuarios puede compartir una vista externa dada; las diferentes vistas externas se pueden solapar. Algunos sistemas permiten la definición de una vista externa a partir de otra (mediante una correspondencia externa/externa); esta característica es útil cuando varias vista externas están estrechamente relacionadas entre si.

Es importante señalar que en la arquitectura de un sistema de bases de datos también es usual que se considere, como parte de ella, al administrador de la base de datos, quien es la persona o grupo de personas responsable del control total de todo el sistema.

El SGBD interactúa con cada uno de los niveles y las correspondencias entre ellos.

5.2 TEMA II

5.2.1 MODELACION CONCEPTUAL

Contenido:

5.2.1.1 Representación de la información

5.2.1.2 Características del modelo conceptual.

5.2.1.3 Modelo Entidad-Relación.

5.2.1.1 Representación de la información

En el proceso y construcción de todo sistema informativo automatizado, el diseño de la BD ocupa un lugar importante, a tal punto que ésta puede verse como un proceso relativamente independiente dentro del diseño del sistema y compuesto por una serie de etapas. Es por ello que resulta de interés el estudio de los problemas relacionados con el diseño de las bases de datos y la modelación de la información.

Cuando se habla de información se hace referencia de forma general, a tres diferentes niveles, tendiéndose a saltar de uno a otro sin establecer una advertencia previa. El primero de estos niveles es el del *MUNDO REAL*, en el que existen *entidades u objetos*, que no son más que cosas o elementos que existen y están bien diferenciados entre si, que poseen propiedades y entre los cuales se establecen relaciones. Por ejemplo, una silla es una entidad u objeto, un automóvil, un empleado, un profesor, un estudiante, que son cosas concretas; pero también puede ser algo no tangible, como un suceso cualquiera, una cuenta de ahorro, o un concepto abstracto.

Entre las propiedades que caracterizan a una entidad u objeto pudieran encontrarse el color, el valor monetario, el nombre, etc.

De las relaciones entre las entidades u objetos hablaremos más adelante.

La determinación de cierta entidad u objeto correspondiente a un fenómeno o proceso, está muy relacionada con el nivel de abstracción en que se esté realizando el análisis. Así, por ejemplo, si se estudia el comportamiento de un insecto especifico en determinadas condiciones climáticas, las propiedades y relaciones que interesan son de un cierto tipo; sin embargo, si se estuviera realizando un estudio de las diferentes especies de insectos, entonces serían otros los objetos a definir, así como las propiedades que los caracterizarían y las relaciones que se establecerían. Si se estuviera analizando todo el reino animal, serían también otros los objetos a definir, con sus características y propiedades.

El segundo de los niveles es el *DOMINIO DE LAS IDEAS* y es en el que se decide la información que debe existir en la BD sobre un fenómeno o proceso del mundo real, o sea, qué información debe almacenarse. En este nivel es donde realmente se define el contenido informativo que representará al fenómeno, proceso o ente de la realidad objetiva que se está analizando. De modo que en este nivel se definen cuáles objetos y qué propiedades de éstos son representativos y sobre los cuales es necesario almacenar información.

En este nivel es donde se trabaja con los conceptos más importantes del modelo de datos, que establecen la relación entre el mundo real y la información almacenada físicamente en la Base de Datos:

necesario distinguir los trabajadores de una empresa de acuerdo a su ocupación como obreros, dirigentes y administrativos. Esto no puede ser representado en el modelo anterior y sólo a través del hecho de que el conjunto entidad (entidad) "obrero", por ejemplo, es siempre (o sea, en todo momento) un subconjunto del conjunto entidad (entidad) "trabajador", podemos deducir cierta clase de dependencia entre los dos tipos. Usando la generalización podemos obtener un nuevo diagrama como se muestra parcialmente en la figura siguiente Fig. #31.

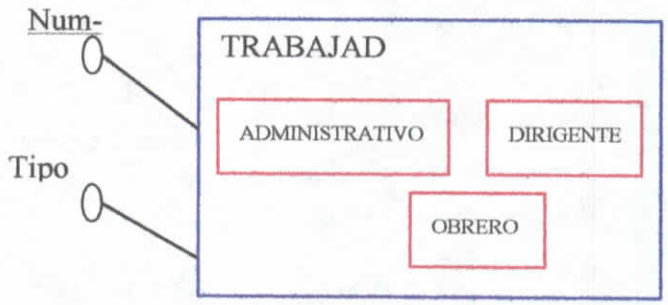


Fig. #31: Generalización del conjunto entidad Trabajador a partir de subentidades

Nótese que hemos introducido un nuevo par adicional atributo-valor (atributo) para el conjunto trabajador. Este atributo nos permite distinguir entre los miembros de diferentes clases de trabajadores. Si tenemos un conjunto entidad (entidad) TRABAJADOR y queremos usar la operación de Especialización como inversa a la generalización, tenemos que especificar "roles" en el modelo, o sea, reglas que definan cuándo una entidad (ocurrencia) TRABAJADOR pertenece a uno u otro componente del conjunto entidad (entidad). Entonces la representación de esta operación en el DER se generaliza como se muestra en la figura número 32.

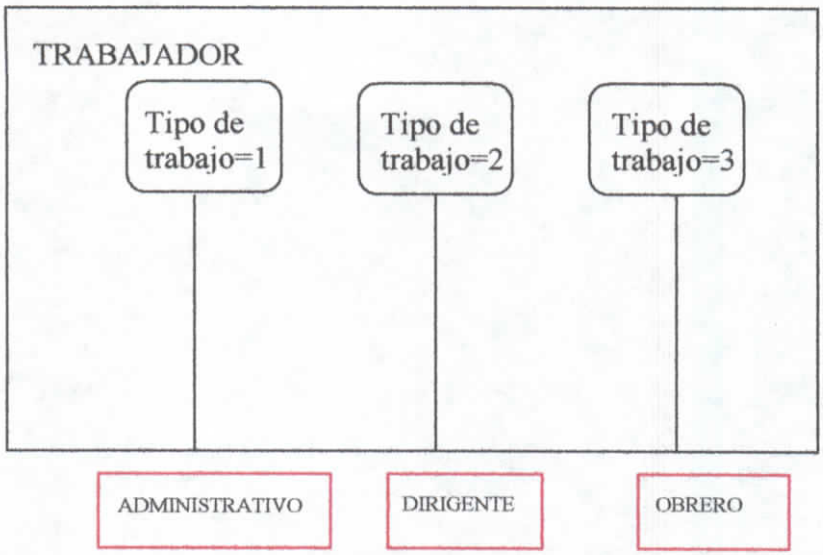


Fig. #32: Especificación de roles en un conjunto entidad "Trabajador" determinando la especialización

Si por cada entidad Trabajador nosotros podemos siempre deducir a cuál conjunto entidad componente pertenece usando alguna propiedad ya representada, entonces no es necesario introducir un nuevo par atributo-valor (atributo) Tipo de Trabajo.

Las reglas que definen la especialización de un conjunto entidad (entidad) se denominan "caracterizaciones". Por ejemplo, Tipo de Trabajo = 1 es la caracterización del conjunto entidad (entidad) Administrativo dentro del conjunto entidad (entidad) Trabajador.

En una Generalización / Especialización, los atributos y relaciones del conjunto entidad "generalizado" (entidad generalizada) son heredados por los conjuntos entidad componentes (entidades especializadas). Además, se pueden definir nuevos atributos y relaciones para cada

conjunto entidad especializada. Por ejemplo, la relación Obrero-Máquina se define ahora sólo para la entidad especializada Obrero, componente de la entidad generalizada Trabajador:

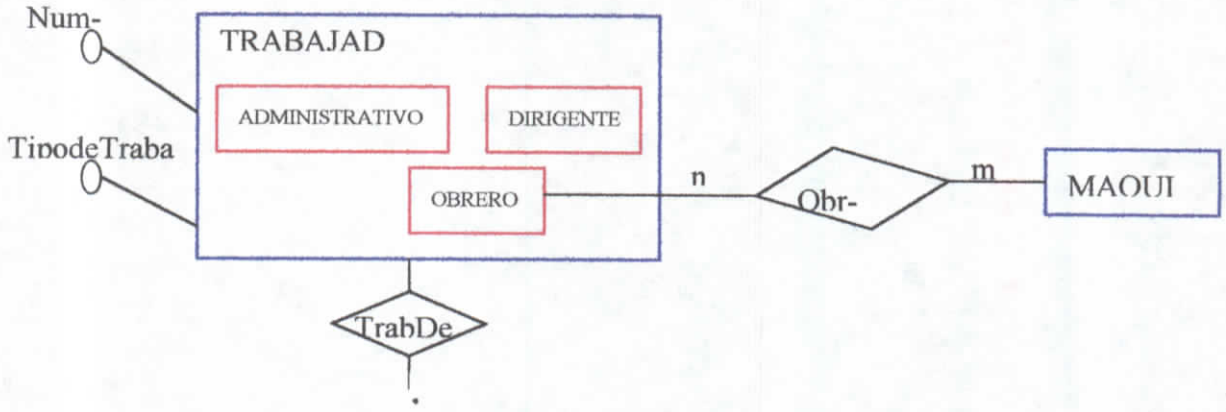


Fig. #33: Figura en donde se puede distinguir las entidades especializadas y generalizadas con sus respectivos atributos y relaciones

b) Agregación.- Obsérvese en el ejemplo que representa la situación de la producción en las empresas, que la relación ternaria Trab-Máq-Pieza representa la idea de que una actividad en la empresa se describe en términos de "un obrero en alguna máquina produce una pieza dada en alguna cantidad específica". Sin embargo, la misma situación puede ser vista de forma algo diferente. En la empresa las máquinas pueden estar asignadas a los obreros y estos "equipos", producir piezas en cierta cantidad. En el MER original esta situación no hubiera podido ser modelada correctamente, ya que una relación no puede relacionarse con otra relación o entidad. Con la operación de Agregación esta situación se resuelve fácilmente, tal y como se muestra en la figura número 34 representada a continuación:

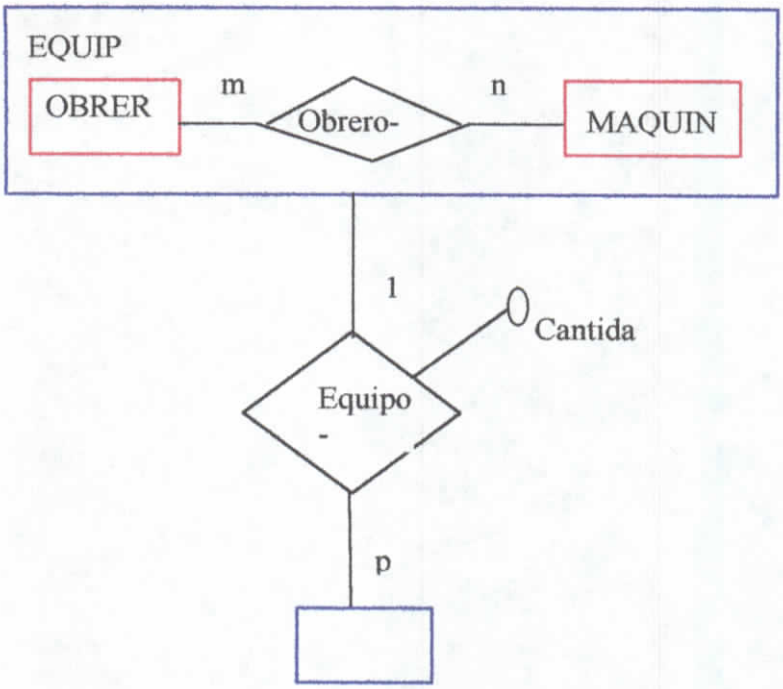


Fig. #34: Representación ternaria de entidades con una relación a una entidad especializada

La agregación se define de la siguiente forma:

Si T1, T2, ...Tn son conjuntos entidad (entidad), la operación define un nuevo conjunto entidad (entidad) T con el siguiente significado.:

$$T = \{t \mid \exists \quad t1, t2, ...tn\ (t1 \in T1 \wedge t2 \in T2 \ldots \wedge tn \in Tn \wedge <t1,t2,..tn> = t)\}$$

O sea, las nuevas entidades (ocurrencias) se forman como tuplas de entidades (ocurrencias) a partir de los conjuntos entidad (entidades) componentes. Para que la operación tenga sentido, los conjuntos entidad (entidades) T1,T2,...Tn tienen que formar parte en alguna relación común y esa relación siempre será incluida en la representación del conjunto entidad (entidad) generado.

Al nuevo conjunto entidad (entidad) se le pueden asignar atributos. También puede tomar parte en cualquier relación. Otro ejemplo de Agregación se muestra a continuación en la figura 35:

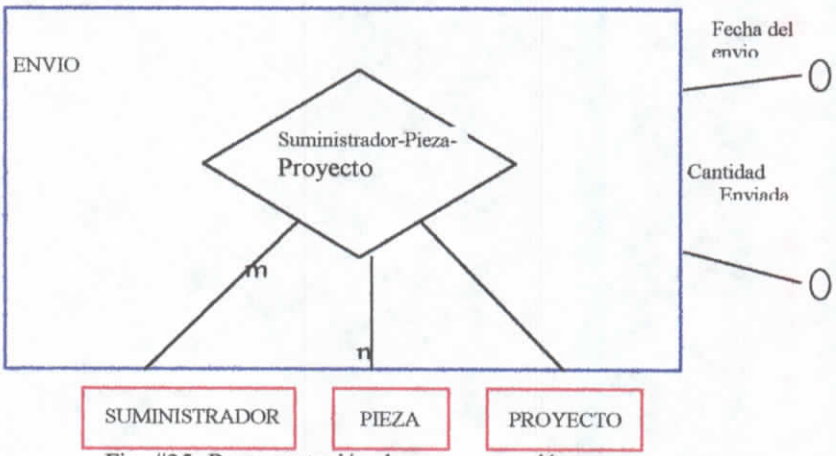


Fig. #35: Representación de una agregación

El nuevo conjunto entidad (entidad) ENVÍO se define como una agregación de tres conjuntos entidad (entidades): Suministrador, Pieza y Proyecto con los nuevos atributos Fecha de envío y Cantidad enviada. Hay una diferencia importante entre estos dos atributos: Esta claro que la Fecha de envío no puede pertenecer a ninguno de los conjuntos entidad (entidad) componentes, sin embargo, la Cantidad enviada se refiere claramente a las piezas. Diremos entonces, que la Cantidad enviada es una "caracterización" del conjunto entidad (entidad) PIEZA con respecto al ENVÍO.

c) **Agrupamiento.-** Si T designa a algún conjunto entidad (entidad) y T1, T2 ...Tn son bien conjuntos valor (dominios) asociados con T o conjuntos entidad (entidades) relacionados con T vía alguna relación, entonces el operador de Agrupamiento construye un nuevo conjunto entidad agrupado (entidad agrupada o agrupamiento)

Tg donde cada elemento es un conjunto de entidades (ocurrencias) de T, tales que, dentro de cada uno de tales conjuntos, todas las entidades (ocurrencias) tienen los mismos valores y entidades relacionadas desde los conjuntos entidad (entidad) T1, T2,..Tn asociados. Los tipos T1, T2,...Tn se llamarán tipos índice y T se llamará base.

Por ejemplo, podemos usar el par Salario-Valor Monetario del DER anterior para formar un conjunto entidad (entidad) agrupado a partir del conjunto entidad Trabajador. Cada entidad (ocurrencia) Trabajador dentro de un grupo, que representa a una entidad (ocurrencia) en Trabajadores de igual salario, tiene el mismo valor del salario.



Fig. #36: Representación de un agrupamiento

Para el MER, incluyendo las tres operaciones estudiadas pueden plantearse una serie de restricciones de integridad:

a) Al aplicar la generalización/especialización, una entidad puede pertenecer a una jerarquía de diferentes conjuntos entidad (entidades). Por ejemplo, los conjuntos

Una relación que satisface este último punto se denomina "Normalizada".

La teoría de la normalización se basa en la necesidad de encontrar una representación del conjunto de relaciones que en el proceso de actualización sea más adecuada. Llevar una relación no normalizada a normalizada es muy simple. Existen diferentes niveles de normalización que se llaman formas normales que veremos más adelante, para aclarar estos conceptos a continuación se detalla un ejemplo:

Veamos cómo el ejemplo de suministrador y producto se puede representar fácil y claramente mediante el modelo relacional expresado en la figura número 37 expuesta a continuación.

Suministrador: Número, Nombre, Tipo y Municipio donde radica.

Producto: Número, Nombre, Precio unitario y Peso.

Se conoce la cantidad de un determinado producto que suministra un suministrador dado.

SUMINISTRADOR

SNUM	**SNOM**	**TIPO**	**MUN**
S1	PÉREZ	30	CERRO
S2	RAMOS	10	PLAZA
S3	ARENAS	20	CERRO
S4	VALLE	20	PLAYA
S5	LÓPEZ	15	PLAYA

PRODUCTO

PNUM	**PNOM**	**PRECIO**	**PESO**
P1	CLAVO	0.10	12
P2	TUERCA	0.15	17
P3	MARTILO	3.50	80
P4	TORNILLO	0.20	10
P5	ALICATE	2.00	50
P6	SERRUCHO	4.00	90

SP

S	**P**	**CANT**
S1	P1	3
S1	P2	2
S1	P3	4
S1	P4	2
S1	P5	1
S1	P6	1
S2	P1	3
S2	P2	4
S3	P3	4
S3	P5	2
S4	P2	2
S4	P4	3
S4	P5	4

Fig. # 38: Representación de un Modelo Relacional en tablas con datos; visualizando sus relaciones

La representación en el modelo relacional es más simple que con el modelo jerárquico y el modelo reticular, ya que con 3 tablas se tiene todo el modelo representado.

En el modelo relacional el resultado de una demanda es también una relación.

Demandas simétricas ====> Operaciones simétricas

Sólo tiene que indicar qué relación desea recuperar. Las diversas formas de hacer las recuperaciones dan lugar a los lenguajes relacionales cuyas formas más representativas son:

- Álgebra relacional (basado en las operaciones del álgebra de relaciones).
- Cálculo relacional (basado en el cálculo de predicados)

- Ventajas: Una de la principales ventajas es su simplicidad, pues el usuario formula sus demandas en términos del contenido informativo de la BD sin tener que atender a las complejidades de la realización del sistema, lo que implica gran independencia de los datos.

  - La información se maneja en forma de tablas, lo que constituye una manera familiar de representarla.
  - Al igual que en el modelo reticular, si se tienen relaciones normalizadas, no surgen dificultades grandes en la actualización.

Veamos en el modelo del SUMINISTRADOR-PRODUCTO visto anteriormente en la figura 38 un ejemplo de cada tipo de operación de actualización:

- Creación: Añadir un producto P7. Se agrega la nueva ocurrencia en la tabla PRODUCTO. Es posible hacerlo aunque ningún suministrador lo suministre.

- Supresión: Se puede eliminar el suministrador S1 sin perder el producto P6, a pesar de que es el único suministrador que lo suministra.

- Modificación: Se puede cambiar el precio del producto P2 sin necesidad de búsquedas adicionales ni posibilidad de inconsistencias.

Veremos que el proceso de normalización no es suficiente hasta el punto aquí visto, también en este punto es necesario dar a conocer ó poner en conocimiento las desventajas que presenta este modelo.

- Desventajas: Se dice que la fundamental consiste en la dificultad de lograr productividad adecuada de los sistemas, ya que no se fabrican los medios técnicos idóneos, tales como las memorias asociativas, siendo necesario simular este proceso, pero, en realidad, la eficiencia y productividad de los sistemas actuales resultan realmente satisfactorias.

A continuación se describen ejemplos de SGBD (Sistemas Gestionador de Bases de Datos) relacionales: Entre otros tenemos los Query (QBE)(IBM); dBase II, III, IV, DataEase, FoxPro, Informix (micros), Microsft Access.

5.3.1.2 Normalización

La teoría de la normalización se ha desarrollado para obtener estructuras de datos eficientes que eviten las anomalías de actualización. El concepto de normalización fue introducido por E.D.

Codd y fue pensado para aplicarse a sistemas relacionales. Sin embargo tiene aplicaciones más amplias.

La normalización es la expresión formal del modo de realizar un buen diseño. Provee los medios necesarios para describir la estructura lógica de los datos en un sistema de información.

- Ventajas:
  - Evita anomalías en la actualización.
  - Mejora la independencia de los datos permitiendo realizar extensiones de la BD, afectando muy poco o nada, a los programas de aplicación existentes que accesan la base de datos.

La normalización involucra varias fases que se realizan en orden. La realización de la 2^{da} fase supone que se ha concluido la 1^{ra} y así sucesivamente. Tras completar cada fase se dice que la relación está en:

a) Primera Forma Normal (1FN)
b) Segunda Forma Normal (2FN)
c) Tercera Forma Normal (3FN)
d) Forma Normal de Boyce-Codd (FNBC)
e) Existen, además, 4FN y 5FN

Las relaciones en 1FN son un subconjunto del universo de todas las relaciones posibles. Las relaciones en 2FN son un subconjunto de las que están en 1FN y así sucesivamente, como se muestra en la figura número 35.

5.3.1.3 Primera Forma Normal

Se puede decir que toda relación normalizada, o sea, con valores atómicos de los atributos, se encuentra en la 1FN, por ende la relación no incluye ningún grupo repetitivo. Formalmente hablando una relación está en (1FN) si cumple la propiedad de que sus dominios no tienen elementos que, a su vez, sean conjuntos.

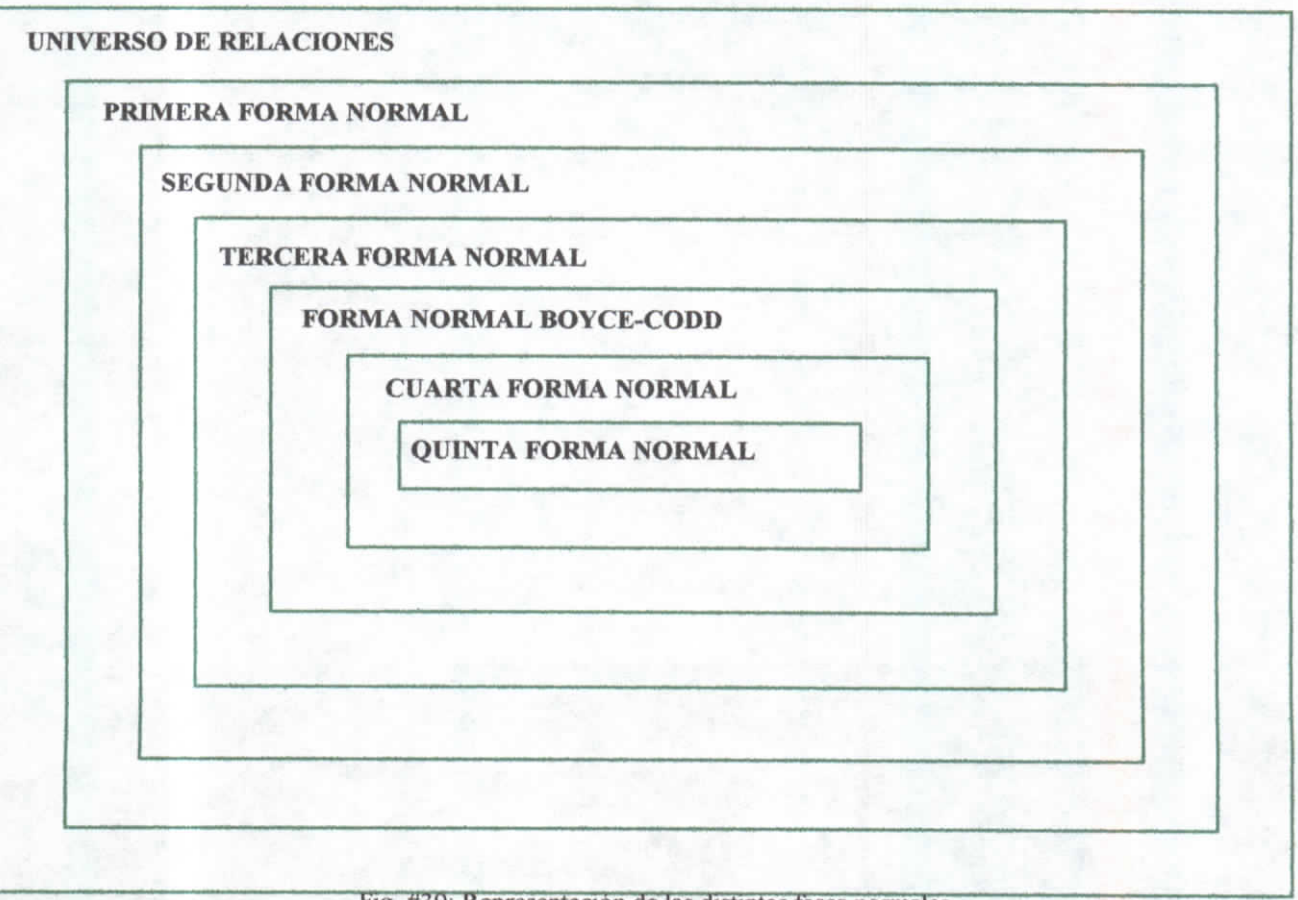


Fig. #39: Representación de las distintas fases normales.

A continuación de desarrolla un ejemplo para ilustrar las fases de 1FN hasta 3FN que consiste en el diseño de la base de datos para la automatización del control de los pedidos de productos:

FECHA: 25/4/86

PEDIDO NO. : 123456
PROVEEDOR NO. : 75621
NOMBRE PROVEEDOR: J. PÉREZ
DIRECCION PROVEEDOR: CERRO

DESEAMOS ENVÍEN:

NÚMERO PRODUCTO	DESCRIPCIÓ	PRECIO UNITARIO	CANTIDAD	TOTAL
969715	TELEVISOR	600	1	600
439124	ANTENA	20	10	200
439126	ENCHUFE	10	10	100
			IMPORTE TOTAL:	900

Fig. #40: Representación de la primera fase normal

El análisis de este pedido muestra que los siguientes datos son de interés. # pedido mostrado en la figura número 40 es único y se utiliza para referirse a un pedido, por tanto, se usará como clave (llave).

NPED

FECHA

NUPROV En forma de relación se escribiría (atributos)

NOPROV

DIREC

NUPROD PEDIDO (NUPED, Fecha, NUPROV, NOPROV, DIREC,

DESC NUPROD, DESC, PRUN, CANT, PRPROD, PRPED)

PRUN

CANT

PRPROD

PRPED

Este pedido contiene 5 grupos repetitivos: NUPROD, DESC, PRUN, CANT, PRPROD, ya que puede contener más de una línea de pedido.

Hay que eliminar esos grupos. Para ello se crea:

a) Una relación para los campos que sean únicos.

b) Una relación para los grupos repetitivos.

a1) Pedido (NUPED, FECHA, NUPROV, NOPROV, DIREC, PRPED).

b1) PED-PROD (NUPED, NUPROD, DESC, PRUN, CANT, PRPROD).

Ambos tienen como llave o parte de la llave, a NUPED. Pero en PED-PROD es necesario la llave compuesta para identificar los productos individuales.

Esta 1FN tiene problemas de actualización:

- Creación: Información sobre un nuevo producto no se puede insertar si no hay un pedido que lo incluya.

- Supresión: Eliminar una línea de pedido que sea la única que pida un producto implica perder la información del producto.

- Modificación: Cada línea de pedido en la que se solicite determinado artículo repite la información sobre éste. Si cambia algún atributo del artículo, entonces es necesario hacer muchas actualizaciones.

5.3.1.4 Segunda Forma Normal

Antes de pasar a describir la 2FN, es necesario abordar algunas definiciones previas como:

- Dependencia Funcional (DF) dada una relación R se dice que el atributo Y de R es funcionalmente dependiente del atributo X de R, si y sólo si, cada valor X en R tiene asociado a él, precisamente, un valor de Y en R en cualquier momento del tiempo.

  X ----> Y

  Ejemplo en la relación SUMINISTRADOR:

  SNOM, TIPO y NUM son funcionalmente dependientes cada uno de SNUM, ya que para un valor de SNUM existe un valor correspondiente de SNOM, TIPO y MUN.

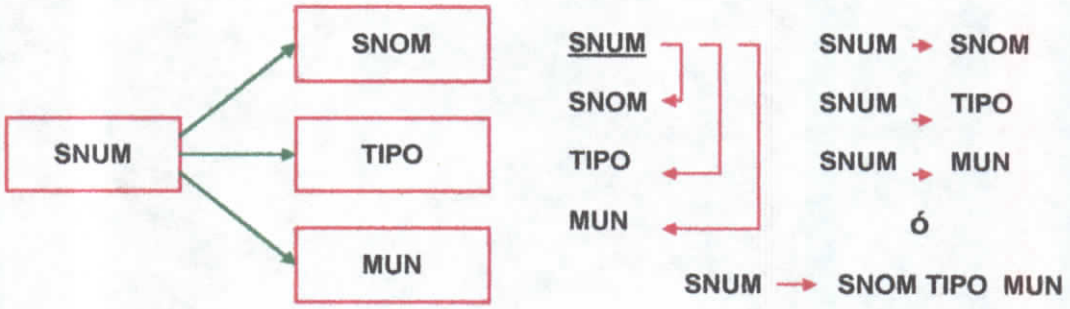


Fig. #41: Representación de tres dependencias funcionales

El reconocimiento de las dependencias funcionales es una parte esencial de la comprensión de la semántica, del significado del dato. El hecho de que MUN dependa funcionalmente de SNUM significa que cada suministrador esta situado en un municipio.

La noción de dependencia funcional puede ser extendida hasta cubrir el caso en que X, Y o ambos atributos sean compuestos.

Ejemplo en la relación SP:

El atributo CANT es funcionalmente dependiente del par de atributos (S,P)



Fig. #42 Representación de una dependencia funcional completa

- Dependencia funcional completa se produce cuando el atributo Y es funcionalmente dependiente y completamente del atributo X, si es funcionalmente dependiente de X y no es funcionalmente dependiente de algún subconjunto de X.

  Se representa: X--->Y

  Ejemplo en la relación SUMINISTRADOR:

  MUN es funcionalmente dependiente de (SNUM, SNOM), pero no es funcionalmente dependiente y completamente de (SNUM, SNOM), ya que también es funcionalmente dependiente de SNUM.

  Ejemplo en la relación SP:

  CANT es funcional y completamente dependiente de (S,P)

- Determinante se produce cuando cualquier atributo del cual depende funcional y completamente cualquier otro atributo. O sea, la parte izquierda de la implicación cuando la dependencia funcional es completa.

  Ejemplo en la relación SUMINISTRADOR:

  SNUM es un determinante.

Al hablar de entidad en el MER se asumió que existe una *llave*, un conjunto de atributos que definen de forma única una entidad. Un concepto análogo se define para las relaciones.

- Llave se define si R es una relación con atributos A1, A2, ...An y X es un subconjunto de A1, A2, ...An, X es una llave de R si:

  a) X---> A1, A2, ...An

  o sea, todos los atributos de la relación dependen funcionalmente de X.

  b) $\nexists Y \subset X \mid Y \rightarrow A1, A2, ...An$

  (esta condición de minimalidad no se requería para el concepto de llave en el MER).

Una relación puede tener varias llaves. Una de ellas se nombra "llave primaria" (la que se escoja para trabajar) y las restantes se nombran "llaves candidatas".

Una superllave será cualquier superconjunto de una llave; entonces, una llave es un caso especial de superllave, para poder hablar de llave es necesario considerar los siguientes procedimiento:

Se tiene una relación R(A,B,C,D,E) con dependencias funcionales:

a) 1. A-->B 2. BC-->D 3. AB->E

Para comenzar se parte de que no existen más llaves que dependencias funcionales, pues el concepto de llave incluye la existencia de dependencia funcional. Se analiza, por tanto, cada una de las DF presentes en la relación, añadiendo los atributos que sean imprescindibles en la parte izquierda para lograr determinar a todos los atributos de la relación. El conjunto de atributos así formado debe ser mínimo.

Luego se analiza cada uno de esos conjuntos mínimos, de forma que, si alguno es un superconjunto de otro, ya no es llave, sino superllave. Pueden resultar varias llaves candidatas. Ejemplo:

a) A-->B AC-->A B E C D ------> AC es llave

b) BC-->D BCA-->B C D A E ------> BCA es superllave

c) AB->E ABC-->A B E C D ------> ABC es superllave

La única llave es AC. No hay ninguna otra llave candidata, pues en las otras DF se obtiene el mismo conjunto (ABC) y contiene a AC.

Ejemplo: Sea la relación R (ciudad, calle, código postal)

C A P

donde se tiene que:

CA--->P (una calle en una ciudad tiene un código postal).

P---->C (el código postal tiene una estructura tal que su valor determina la ciudad, pero en una ciudad, varias calles pueden tener el mismo código, por lo que no se cumple P--->A.

Entonces, CA es llave, ya que determina a todos los atributos de P (determina a P y obviamente a si mismo), CA--->CAP. P determina a C y no a A, pero haciendo: PA--->CA, es obvio que PA--->PA también, entonces PA--->CAP, por tanto, PA también es llave.

PA y CA son llaves candidatas. En ninguno de los casos un subconjunto de ellas es también llave.

Un atributo Ai de R es un atributo llave si él es (o es miembro de) una llave (candidata o primaria).

Un atributo Aj de R es un atributo no llave si él no es miembro de ninguna llave.

En el ejemplo C, A y P son todos atributos llaves.

Una vez descritas y ejemplarizadas las definiciones más importantes previo el estudio y descripción de la segunda forma normal se puede pasar a dar una definición de lo que es la 2FN (segunda forma normal).

Una relación R se dice que está en 2FN si está en 1FN y si, y sólo si, los atributos no llaves (ni primarias, ni candidatas) de R, si los hubiese, son funcional y completamente dependientes de la llave primaria de R.

Entonces, este segundo paso se aplica sólo con relación a llaves compuestas.

Continuando con el ejemplo de los pedidos de productos figura 40, habíamos visto que en la relación PED-PROD subsistían problemas de actualización. Analicemos las DF que existen en dicha relación:

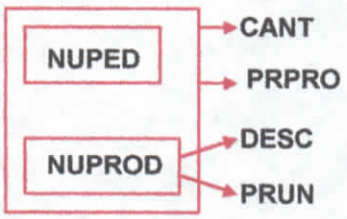


Fig.#43: Representación de una relación que no forma parte de la segunda forma normal

Esta relación no está en 2FN, pues DESC y PRUN no dependen funcional y completamente de la llave (NUPED, NUPROD).

La 2FN se hace:

a) Creando una relación para todos los atributos que dependen funcional y completamente de la llave (y los atributos que no se analizan por ser atributos llaves, pertenecientes a claves candidatas).

b) Y una relación para los atributos que dependan de parte de la llave.

PED-PROD (NUPED,NUPROD, CANT, PRPROD)

PRODUCTO (NUPROD, DESC, PRUN)

Los problemas planteados en la 1FN se resuelven con la 2FN.

- Creación: Se puede insertar la información sobre un producto.

- Supresión: Se puede eliminar una línea de pedido y no se pierde la información sobre el producto, aunque sea el único pedido que pide ese producto.

- Modificación: Si cambia un atributo del producto, sólo hay que cambiarlo en un lugar. Se elimina redundancia.

Pero aún se tienen problemas en este caso que son similares a las vistos, pero con la relación PEDIDO y, específicamente, cuando se trata de insertar, eliminar o modificar la información de PROVEEDORES:

- Creación: No podemos insertar información de proveedores a menos que haya un pedido para él.

- Supresión: Se perderá la información sobre un proveedor al borrar un pedido que es el único que se le hacía a él.

- Modificación: Para cambiar información sobre un proveedor, hay que recorrer todos los pedidos de ese proveedor.

5.3.1.5 Tercera Forma Normal

Se puede decir que una relación R está en 3FN si está en 2FN y si, y sólo si, los atributos no llaves son independientes de cualquier otro atributo no llave primaria.

Esto es lo mismo que decir que se deben eliminar las dependencias transitivas de atributos no llaves respecto a la llave primaria, estando ya la relación en 2FN.

Para entender de una mejor forma lo acabado de explicar es necesario dar a conocer la definición de *dependencia transitiva*, y para esto se toman los conjuntos A, B y C como conjuntos de atributos de una relación R. Si B es dependiente funcionalmente de A y C lo es de B, entonces C depende transitivamente de A, como se lo puede observar en la figura siguiente.

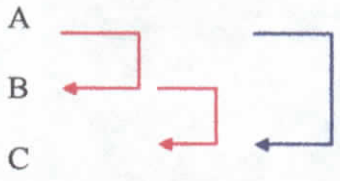


Fig. #:44 Representación gráfica de una dependencia transitiva.

Este paso se ejecuta examinando todas las relaciones para ver si hay atributos no llaves que dependan unos de otros. Si se encuentran, se forma una nueva relación para ellos.



Fig. #45: Forma de cómo crear una nueva relación a partir de atributos no llaves que dependan unos de otros.

PEDIDO (NUPED, FECHA, NUPROV, PRPED)

PROV (NUMPROV, NOPROV, DIREC)

(Analizar las otras relaciones. No hay dependencia entre atributos no llaves)

La 3FN ha eliminado los problemas asociados con la información sobre el proveedor en la 2FN, es por eso que en esta etapa se puede optimizar la 3FN. Puede haber relaciones degeneradas que contengan sólo la clave, por lo que se pueden eliminar. Puede que otras relaciones tengan la misma clave, por lo que se pueden combinar en una sola, siempre que el resultado sea lógico y tenga sentido.

Los analistas y diseñadores con experiencia producen relaciones en 3FN casi sin saber o preocuparse de esto y es que utilizan el sentido común y la experiencia para escribir relaciones normalizadas. No obstante, no siempre la intuición es suficiente y la metodología para normalizar las bases de datos se convierte en una herramienta imprescindible, que garantiza un diseño idóneo de los datos.

Aún en la 3FN existen problemas y habrán de ser resueltos con las siguientes formas normales:

5.3.1.6 Forma Normal de Boyce-Codd (FNBC)

La definición de la 3FN puede resultar inadecuada en el caso de una relación donde ocurre lo siguiente:

a) La relación tiene varias llaves candidatas, donde

b) esas llaves candidatas son compuestas y

c) esas llaves candidatas se solapan (o sea, tienen al menos un atributo común).

Es decir, para una relación donde no tengan lugar las tres condiciones anteriores, la FNBC es idéntica a la 3FN. Esas tres condiciones son necesarias, pero no suficientes, para que la relación no esté en FNBC.

A continuación se detalla la definición de la Forma Normal de Boyce-Codd(FNBC):

Una relación R está en FNBC si, y sólo si, cada determinante es una llave (candidata o primaria), obsérvese que se habla en términos de llaves candidatas y no sólo de la llave primaria, ya que una llave es un caso especial de superllave y la llave puede ser candidata o primaria, adicionalmente a esto se dice que la definición de FNBC es conceptualmente más simple, aunque es una FN más "fuerte". Una relación que esté en FNBC está también en 1FN, 2FN y 3FN, como se detalla en el siguiente ejemplo, la relación:

PRODUCTO (PNUM, PNOM, PRECIO, PESO)

con las DF PNUM--->PNOM, PRECIO, PESO.

PNOM--->PNUM, PRECIO, PESO.

suponiendo que PNUM y PNOM sean llaves candidatas, está en FNBC, ya que, en todas las DF que existen, los determinantes son llaves candidatas y, por tanto, superllaves de PRODUCTO.

Analicemos otro ejemplo:

Sea la relación EAP (Estudio, Asignatura, Profesor) donde una tupla significa que un estudiante E recibe la asignatura A impartida por el profesor P y en la cual se cumple:

- para cada asignatura, cada estudiante tiene un solo profesor.

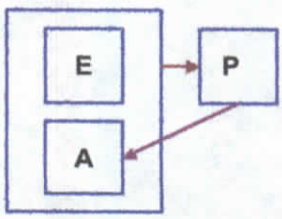
- cada profesor imparte sólo una asignatura.

- cada asignatura es impartida por varios profesores.

O sea que, EA--->P

P--->A

y no existe DF de P con respecto a A



pero si, P--->A, entonces EP--->A también, por lo que EP es una llave candidata de la relación, ya que determina a todos los atributos (E,P,A). Ambas llaves (EA y EP) son compuestas y se solapan, por lo que debe analizarse la FNBC

Esta relación está en 3FN, pero no en FNBC, ya que el determinante P no es llave de la relación.

Por ejemplo:

EAP	E	A	P
	Pérez	Matem	prof. Blanco
	Pérez	Físic	prof. Valdés
	Rdguez	Matem	prof. Blanco
	Rdguez	Físic	prof. Hdez

Esta relación presenta anomalías de actualización, por ejemplo, si queremos eliminar la información de que Rdguez estudia Física, perdemos la información de que el profesor Hdez imparte Física.

Estos problemas pueden eliminarse sustituyendo la relación original EAP por dos relaciones:

EP (E,P) y PA (P,A)

Es decir que se separa la DF problemática (P--->A) en una relación independiente, donde el determinante será la llave (P), y se forma otra relación, donde dicho determinante P participará

como atributo, pero en la cual no puede participar el atributo determinado en la dependencia funcional conflictiva (A).

De la misma forma que en anteriores ocasiones a continuación se detalla otro ejemplo: Sea la relación R1 (C, A, P) C- ciudad

A- calle

P- código postal

donde CA--->P

y P--->C, con llaves candidatas CA y PA.

Esta relación no está en FNBC, ya que existe un determinante (P) que no es superllave.

La solución será dividir R1 en

R2 (P, C) y R3 (A,P)

Para aclarar este concepto citamos otro ejemplo:

Sea la relación EXAM (E, A, L) donde:

E- estudiante

A- asignatura

L- lugar en el escalafón alcanzado por un estudiante en una asignatura y en la que se supone que 2 estudiantes no pueden alcanzar el mismo lugar en una asignatura.

Entonces:

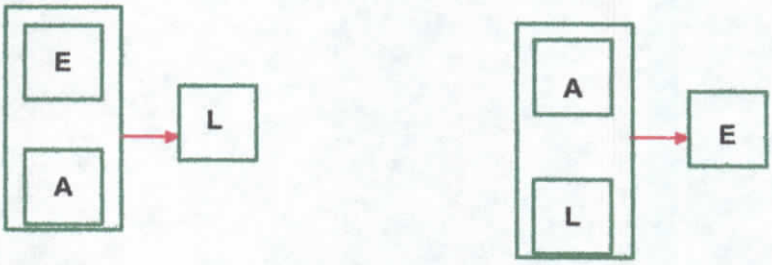


Fig. #46: forma de representar dos llaves candidatas.

Como se ve, existen dos llaves candidatas, EA y AL, las cuales se solapan, pero no existen otros determinantes que no sean esas llaves candidatas (que son casos especiales de superllaves), por lo que la relación EXAM está en FNBC.

Por último, es necesario destacar que la descomposición de una relación para obtener la FNBC puede implicar que se pierdan dependencias funcionales. Por ejemplo en la relación EAP donde EA-->P y P-->A al descomponerse en

EP (E,P) y PA (P,A)

se pierde la DF EA-->P ya que esta DF no puede ser deducida a partir de las que existen en EP y PA (que sólo es P-->A), lo cual implica que ambas relaciones EP y PA no pueden ser actualizadas "independientemente", sino que una actualización en un lugar conlleva un chequeo

de actualización en el otro lugar (para añadir un profesor a un estudiante hay que chequear la materia que el profesor imparte y hay que chequear si el estudiante ya tiene un profesor de esa materia y en ese caso no se puede añadir).

Entonces, para el proceso de normalización se deben dar los siguientes pasos:

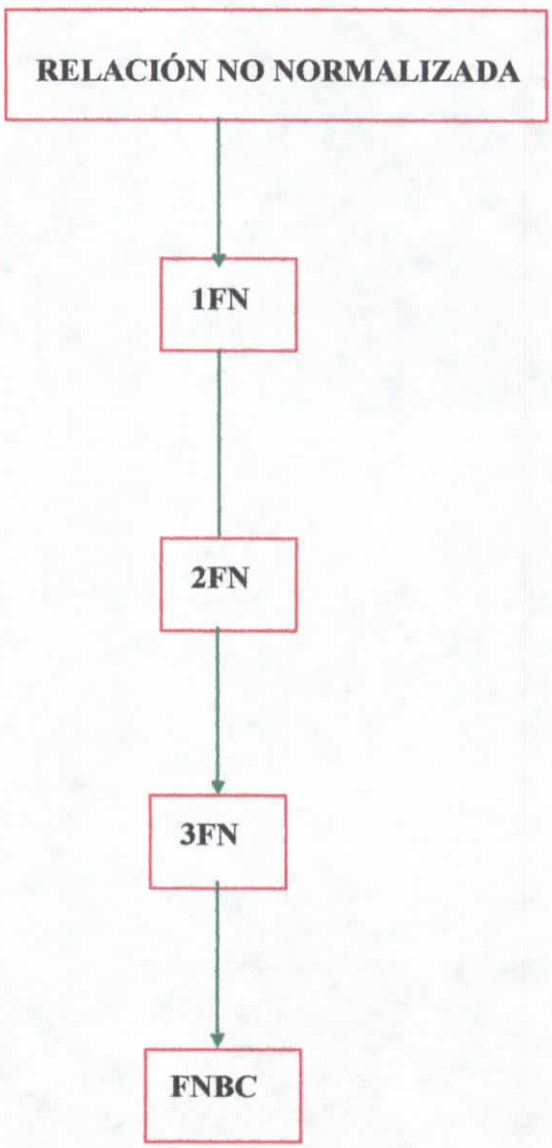


Fig. #47: Pasos de un proceso de normalización.

Reducir a valores elementales de los atributos.

- Eliminar las dependencias funcionales incompletas de los atributos no llaves respecto a la llave primaria.

Eliminar dependencias transitivas

- Eliminar las dependencias en las que el determinante no sea superllave.

A continuación se desarrollará un ejemplo de normalización hasta llegar a la fase de la Forma Normal Boyce Codd FNBC

Se dice que se desea diseñar una BD para controlar la disponibilidad de materiales de construcción: De cada proveedor de materiales se conoce su código (cprov), que lo identifica, su nombre (nomprov) y el municipio en que radica (mun). De cada material se sabe su código (cmat), que lo identifica, su descripción (desc), la unidad de medida que se aplica al material (um) y el precio por unidad de medida (precio).

Para guardar estos materiales hasta su posterior distribución existen diversos almacenes. De cada almacén se conoce su código (calm), que lo identifica, su dirección (diralm) y la capacidad de almacenaje (capac).

Un proveedor puede suministrar varios materiales y un material puede ser suministrado por diferentes proveedores. Se sabe que un material suministrado por un proveedor está en un solo

almacén y además, se sabe qué cantidad de un material suministrado por un proveedor se encuentra en el almacén (cantmat). En un almacén sólo se guarda un tipo de material, aunque puede proceder de distintos proveedores y pueden existir varios almacenes donde se guarde un mismo material.

Se detallan las dependencias funcionales que se derivan de esta situación:

pues cprov identifica al proveedor.

a) cprov --> nomprov mun es el determinante de nomprov y mun

b) cmat --> desc um precio análogo al anterior para materiales

c) calm --> diralm capac cmat desc um precio

1 2 3 4 5 6

Esta última dependencia funcional tiene la siguiente explicación:

Los atributos 1 y 2 dependen funcionalmente de calm pues calm identifica al almacén. Pero se sabe que en un almacén sólo se guarda un material, por lo que conocido calm se conoce cmat y, por lo tanto, el atributo 3 depende funcionalmente de calm, pero como 4, 5 y 6 dependen funcionalmente de 3, ellos están incluidos también en la dependencia funcional expresada anteriormente 3).

1 2 3 4 5

d) cprov cmat --> nomprov mun desc um precio

6 7 8 9

calm diralm capac cantmat

Se incluyen en esta dependencia funcional los atributos:

1 y 2: por lo explicado para **a**)

3,4 y 5: por lo explicado para **b**)

6: porque dado un proveedor y un material se conoce en qué almacén se guarda ese material suministrado por ese proveedor.

7 y 8: porque 6 se incluye y entonces vale lo explicado para **c**)

9: porque dado un proveedor y un material se conoce también en qué cantidad se guarda en el almacén correspondiente.

1 2 3 4 5 6

e) cprov calm --> nomprov mun diralm capac cmat desc

7 8 9

um precio cantmat

Se incluyen en esta dependencia funcional los atributos:

1 y 2: por lo explicado para **a**)

3, 4, 5, 6, 7 y 8: por lo explicado para **c**)

9: porque se incluyen en los atributos implicados el atributo **e** y cprov y **e** determinan 9, según lo explicado para el atributo 9 en **d**)

Entonces:

cprov cmat son ambas llaves candidatas ya que todos los atributos dependen funcionalmente de ellas y cprov calm no existe ningún subconjunto propio del cual todos los atributos dependan.

Escojamos como llave primaria cprov cmat

R (cprov, cmat, nomprov, mun, desc, um, precio, calm, diralm, capac, cantmat)

a) 1FN

La relación original está en 1FN, pues no existen grupos repetitivos.

b) 2FN

No está en 2FN pues existen dependencias funcionales incompletas de atributos no llaves respecto a la llave primaria (según se aprecia en las dependencias funcionales 1) y 2)) por lo que se separan las relaciones siguientes:

- proveedor (cprov, nomprov, mun) relación (I)
- material (cmat, desc, um, precio) relación (II)

c) 3FN

Las relaciones anteriores (I y II) están en 3FN, pero la relación restante de la original, no, ya que existe dependencia transitiva respecto a la llave primaria pues:

cprov cmat --> calm y

calm --> diralm capac

por lo que se separa la siguiente relación:

- almacén (calm, diralm, capac) relación (III)

d) FNBC

Luego de haberse obtenido, es decir, separado, las relaciones I, II y III al analizarse la 2FN y la 3FN, resta la relación:

- suministro (cprov, cmat, calm, cantmat) relación (1)

que está en 3FN. Analicemos las dependencias funcionales que existen en esta relación:

- cprov cmat --> calm cantmat problemática (a)
- calm --> cmat problemática (b)

Luego no está en FNBC, pues existe un determinante (calm) que no es superllave.

Para llevarla a FNBC hay que separar la dependencia funcional problemática (b.), obteniéndose de la relación suministro las siguientes:

- suministro (cprov, calm, cantmat) puede aparecer ya cmat relación (2)
- distribución (calm, cmat) relación (3)

Ahora bien, debe señalarse que al descomponerse la relación suministro (1) de esta manera (suministro (2) y distribución) deja de poder expresarse el hecho de que el calm depende del conjunto cprov, cmat.

Finalmente; analizando la relación distribución (3) y la relación almacén (III) puede apreciarse que ambas pueden unirse: tienen la misma llave y tiene sentido la unión de ambas. Entonces las relaciones obtenidas son:

proveedor (cprov, nomprov, mun)

material (cmat, desc, um, precio)

almacén (calm, diralm, capac, cmat)

suministro (cprov, calm, cantmat)

5.3.1.7 Obtención del modelo relacional a partir del DER

Para obtener el modelo lógico global de los datos según el enfoque relacional a partir del DER, se sigue un procedimiento que iremos describiendo paso a paso y aplicándolo, así mismo, al ejemplo siguiente:

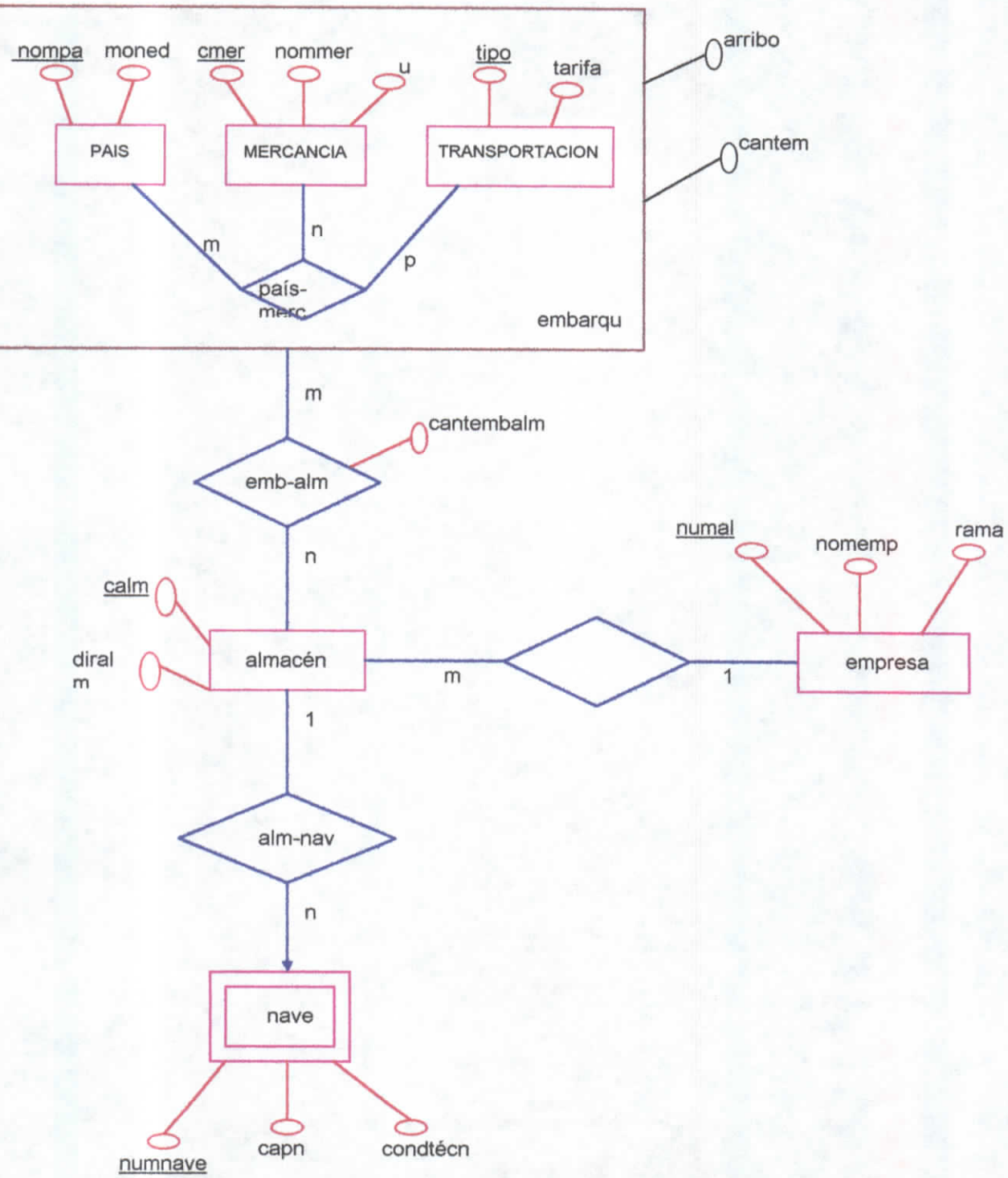


Fig. #48: Obtención del modelo relacional a partir del DER

En el DER anterior se representa el fenómeno del movimiento mercantil de un organismo. En el organismo existen mercancías de las que se conoce su código, nombre y unidad de medida. Las mercancías proceden de diferentes países de los que se sabe nombre y área de moneda. Para la transportación de las mercancías existen diversas formas, cada una de las cuales se caracteriza por su tipo (barco, avión, tren, etc.) y tarifa. De un país se reciben muchas mercancías y una mercancía puede venir de muchos países. Para cada mercancía de un país existen diferentes formas de transportación y una forma de transportación puede serlo de diferentes mercancías de diferentes países. Una mercancía procedente de un país transportada de una forma dada constituye un embarque y para éste se conoce su fecha de arribo y cantidad.

Un embarque se distribuye entre diferentes almacenes y en un almacén se tienen diferentes embarques, cada uno en cierta cantidad. De cada almacén se tiene su código y dirección. Un almacén envía sus productos a una sola empresa y cada empresa recibe productos de diferentes almacenes. Una empresa se caracteriza por su número, nombre y rama económica.

Cada almacén tiene distintas naves subordinadas. De cada nave se conoce su número (que se puede repetir en diferentes almacenes), capacidad y condiciones técnicas.

5.3.1.7.1 PASO # 1

Representar cada entidad regular en una tabla relacional.

a) país (nompa, moneda)

b) mercancía (cmerc, nommerc, um)

c) transportación (tipo, tarifa)

d) almacén (calm, diralm)

e) empresa (numemp, nomemp, rama)

5.3.1.7.2 PASO # 2

Representar en una tabla relacional cada entidad agregada con sus correspondientes atributos (entre ellos un identificador si fue definido) y, las llaves de las entidades que forman la agregación

embarque (nompa, cmerc, tipo, arribo, cantemb)

Nótese que la llave estaría formada por las llaves de las 3 entidades regulares que intervienen en la agregación.

Pero podía haberse definido un identificador para la entidad embarque (Añádase en el DER lo que está oscuro: idemb). Entonces se añadiría como atributo llave en la agregación y los 3 atributos nompa, cmerc y tipo permanecerían en la relación pero no como llaves:

embarque (idemb, nompa, cmerc, tipo, arribo, cantemb)

5.3.1.7.3 PASO # 3

Representar cada entidad generalizada en una tabla que contendrá sus atributos (sólo los de la generalizada) y, entre ellos, la llave.

Representar cada entidad especializada en una tabla que contendrá la llave de la generalización y los atributos propios sólo de la especialización.

5.3.1.7.4 PASO # 4

Representar en una tabla relacional cada relación de m:m, incluyendo las llaves de las entidades relacionadas y los atributos de la relación si los hubiese.

emb-alm (nompa, cmerc, tipo, calm, cantembalm)

Esta relación sería de esta forma sin considerar la existencia del idemb. Caso de que éste estuviera definido quedaría:

emb-alm (idemb, calm, cantembalm)

ya que entonces la llave de la entidad embarque sería idemb.

5.3.1.7.5 PASO # 5

Para cada relación de 1:m, añadir la llave de la entidad del extremo "1" como un nuevo atributo a la entidad del extremo "m" y los atributos de la relación si existe n.

En nuestro ejemplo, se da la relación

almacén <<----------> empresa

por lo que la llave de la entidad empresa (numemp) debe añadirse como atributo en la tabla que representa la entidad almacén. Esto hace que la relación d, obtenida en el paso # 1 quede modificada de la siguiente manera:

almacén (calm, diralm, numemp)

Al agregarse el atributo numemp en la relación almacén se sabe entonces a qué empresa abastece el almacén. Está claro que muchas ocurrencias de almacén pueden tener el mismo valor en el atributo numemp, pues una empresa recibe mercancías de varios almacenes, en general.

5.3.1.7.6 PASO # 6

Representar cada entidad débil en una tabla relacional que contendrá la llave de la entidad regular determinante y el identificador de la entidad débil con sus atributos.

nave (calm, numnav, capnav, condtécn)

Estas son las tablas relacionales que se derivan del DER del ejemplo. Ahora es preciso analizar cada una de las relaciones para asegurarse que están normalizadas hasta FNBC.

En este caso, puede verse que todas las relaciones cumplen con la FNBC, por lo que no es preciso hacer nada más.

En caso de que no estuvieran debidamente normalizadas, era preciso aplicar el proceso de normalización tal y como lo vimos anteriormente.

5.3.1.8 Metodología para el diseño de bases de datos.

. Determinación de entidades y atributos

. Normalización de entidades

. Determinación de relaciones (DER)

. Obtención del modelo lógico global de los datos

. Diseño físico de la BD

Cuando se va a realizar el diseño de la base de datos para un sistema determinado, es necesario determinar los datos que es necesario tomar en cuenta y las dependencias funcionales existentes entre ellos.

Rigurosamente, esto se obtiene luego de realizada la etapa de análisis del sistema y partiendo de lo obtenido en ésta, pero trataremos de dar indicaciones que permitan lograr un buen diseño de la BD a partir de lo que conocemos.

La situación a partir de la cual se discutirá la metodología para diseñar la base de datos es la siguiente:

En un hospital Clínico Quirúrgico se desea automatizar el control de la actividad quirúrgica. En el hospital se tienen varios quirófanos y se realizan intervenciones quirúrgicas en diferentes especialidades. Las intervenciones quirúrgicas pueden ser electivas (planificadas) o urgentes (no planificadas). Se tiene la planificación de las operaciones quirúrgicas a realizar cada día. Las operaciones planificadas pueden ser suspendidas por distintas causas. Para cada operación realizada (haya sido planificada o no), se confecciona un informe operatorio. En cada operación realizada se emplean materiales cuyo gasto se controla.

Con el sistema automatizado se pretende obtener las sgtes. salidas:

Paciente	**Hist Clín.**	**Edad**	**Sexo**	**Diagnóstico Pre-Oper.**	**Intervenc. Indicada**	**Hora**	**Salón**	**Cirujano**
. .				. .				. .

OPERACIONES SUSPENDIDAS		**En el día: de**	**de 19**
Paciente	**Historia Clínica**	**Edad**	**Causa**
. .			. .

INFORME OPERATORIO	**Fecha de Operación:**	
	Hora Comienzo	**Hora Terminación**

Paciente	**Historia Clínica**	**Edad**

Diagnóstico Clínico:
Operación Realizada: _____
Diagnóstico Operatorio: _____

_____

Situación Final del Paciente:

Satisfactoria	**No Satisfactoria**	**Fallecido**
☐	☐	☐

Especialidad	**Tipo de Operación Realizada**	
	☐ **Electiva**	☐ **Urgente**

Cirujano: _____

RESUMEN DE ACTIVIDADES QUIRURGICAS	**Mes:**

Cantidad de Operaciones Electivas: _____
Cantidad de Operaciones Urgentes: _____

Especialidad	**Cantidad de Operaciones**

GASTO DE MATERIALES DE OPERACION					En el día: de			
Paciente	**Hist. Clín.**	**Edad**	**Hora Com.**	**Patología**	**Cód. Mat.**	**Descr. Mater.**	**Cantidad**	**Costo**

Para comenzar a estudiar la metodología de diseño de la base de datos, partiremos de una premisa: lo que se obtiene con un sistema automatizado son sus salidas, es decir, el resultado que se desea obtener, lo que llega al usuario, son justamente las salidas del sistema y para ello se desarrolla; por lo tanto, lo que no sea necesario para obtener las salidas del sistema, no tiene por qué estar almacenado en la base de datos. En fin, lo que debe almacenarse en la base de datos es lo que es imprescindible para obtener las salidas.

Es por esta razón que para el diseño de la base de datos, se parte del análisis de las salidas que se desea obtener con el sistema.

a) Determinar las entidades y los atributos.-

Para cada salida:

. Consultar su formato

. Determinar datos que se calculen u obtengan a partir de otros e ir sustituyéndolos hasta llegar a los primarios

- Determinar existencia de ficheros con información normativa y/o de consulta
- Cada salida y cada fichero, tomarlos como entidad y relacionar sus atributos

En el ejemplo, representando cada una de las salidas por una relación, se tienen las sgtes. relaciones o entidades iniciales:

1.

PQ (Fecha, NomPac, HC, Ed, Sexo, DiagPre, Interv, Hora, Salón, Cirujano)

2.

OS (Fecha, NomPac, HC, Ed, Causa)

3.

IO (Fecha, HoraCom, HoraTerm, NomPac, HC, Ed, DiagClín, Operac, DiagOper, SitPac, Espec, TipoOpReal, CirReal)

4.

RAQ (Mes, Elect, Urg, Espec, Cant)

5.

GMO (Fecha, NomPac, HC, Ed, HoraCom, Patología, CódMat, DescMat, Cant, Costo)

Analizando cada dato se ve que:

En 1.

Hora se calcula según el tiempo promedio de demora para cada operac., por lo que es necesario un fichero normativo:

TPO (Operación, Tiempo Prom)

Salón se decide teniendo en cuenta la distribución de los salones, que en este caso, es por día de la semana:

DS (Salón, DíaSem, Espec)

En 2.

Todos son datos primarios

En 3.

Todos son datos primarios

En 4.

Elect, Urg y Cant son datos que se calculan a partir de contar en IO (relación 3), por lo que la relación 4 se puede eliminar completamente para el análisis, pues al ser calculables todos los datos contenidos en ella, no aporta información nueva a almacenar en la base de datos.

En 5.

Costo = cant x precio, por lo que se sustituye costo por precio (ya cant está en el modelo)

necesidad de existencia de un fichero de consulta MATERIAL (puede salir en entrevistas a usuarios)

b) Normalizar las entidades.-

. Determinar las DF

. Normalizar cada entidad

. Fusionar, de ser lógico, las entidades normalizadas que tengan la misma llave

1.

PQ (Fecha, NomPac, HC, Ed, Sexo, DiagPre, Interv, Hora, Salón, Cirujano)

Normalizando:

a. Paciente (HC, NomPac, Ed, Sexo)

b. PQ (Fecha, HC, DiagPre, Interv, Hora, Salón, Cirujano)

2.

OS (Fecha, NomPac, HC, Ed, Causa)

Normalizando:

c. Paciente1 (HC, NomPac, Ed)

d. OS (Fecha, HC, Causa)

3.

IO (Fecha, HoraCom, HoraTerm, NomPac, HC, Ed, DiagClín, Operac, DiagOper, SitPac, Espec, TipoOpReal, CirReal)

Normalizando:

2FN:

Paciente2 (HC, NomPac, Ed)

PlanOperac (Fecha, HC, DiagClín, Operac, Espec)

IO (Fecha, HoraCom, HC, HoraTerm, DiagOper, SitPac, TipoOpReal, CirReal, OperReal)

3FN:

e. Paciente2 (HC, NomPac, Ed)

f. PlanOperac (Fecha, HC, DiagClín, Operac)

g. ClasifOperac (Operac, Espec)

h. IO (Fecha, HoraCom, HC, HoraTerm, DiagOper, SitPac, TipoOpReal, CirReal, OperReal)

4. (Se elimina para este análisis)

5.

GMO (Fecha, NomPac, HC, Ed, HoraCom, Patología, CódMat, DescMat, Cant, Precio) (Precio en lugar de Costo)

i. Paciente3 (HC, NomPac, Ed)

j. Diagóstico (Fecha, HC, Patología)

k. Material (CódMat, DescMat, Precio)

l. GMO (Fecha, HC, HoraCom, CódMat, Cant)

Además, se tienen los dos ficheros normativos o de consulta:

m. TPO (Operación, Tiempo Prom)

n. DS (Salón, DíaSem, Espec)

Fusionando las entidades normalizadas con igual llave, siempre que sea lógico, se tiene:

. De a, c, e y i ==>Paciente (HC, NomPac, Ed, Sexo)

. De b, d, f y j ==>f está contenida en b y j está contenida en b también

(DiagPre = DiagClín = Patología) Son alias diferentes para indicar un mismo atributo, que tiene el mismo dominio

Pero d, a pesar de tener = llave, está representando otra entidad, la Operación Suspendida; semánticamente, no representa lo mismo que una Operación Planificada y, por lo tanto, no tiene sentido unirlas. Es más, si se hiciera, todas las operaciones tendrían el atributo Causa, y en la mayoría de las ocurrencias, este atributo estaría vacío, pues sólo son algunas operaciones las que se suspenden.

PQ (Fecha, HC, DiagPre, Interv, Hora, Salón, Cirujano)

OS (Fecha, HC, Causa)

. De m y g ==>Intervención (Operación, TiempoProm, Espec)

. De n ==>DS (Salón, DíaSem, Espec)

. De h ==>IO (Fecha, HoraCom, HC, HoraTerm, DiagOper, SitPac, TipoOpReal, CirReal, OperReal)

. De k ==>Material (CódMat, DescMat, Precio)

. De l ==>GMO (Fecha, HC, HoraCom, CódMat, Cant)

c) Determinar las relaciones entre las entidades (DER).-

. Si hay alguna entidad cuya llave es combinación de llaves de otras entidades, generalmente, es porque representa una relación entre ellas (m:m)

. Analizar cada entidad con las restantes para determinar si existe relación, determinando tipo de proyección.

. Ir confeccionando el DER

. Analizar entidades agregadas, entidades débiles y sus relaciones con otras

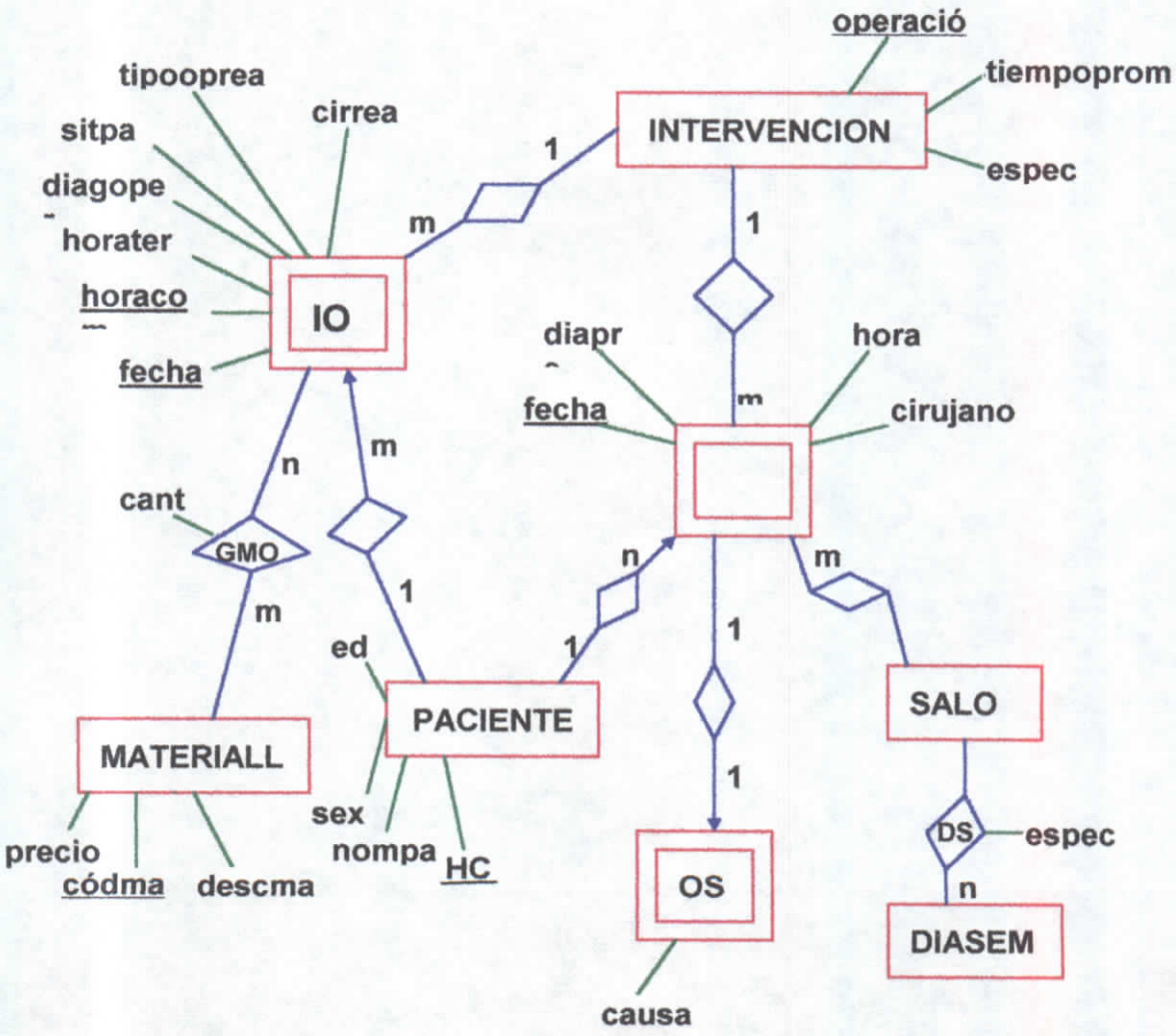


Fig.# 49: DER del ejemplo de la automatización de la clínica.

En este paso se puede comprobar si lo que se ha realizado es correcto o si surge alguna contradicción, se puede completar el modelo con nuevos elementos que se hayan podido obtener en intercambios con los usuarios, ya que con el DER se tiene un modelo menos abstracto, más cercano al fenómeno. El analista puede proponer nuevas posibilidades a los usuarios y agregarlas en caso de acordarse esto, etc.

d) Obtener el modelo lógico global de los datos

Aquí se deben seguir los pasos para llevar de DER al modelo lógico global de los datos.

Si en el paso anterior no se modifica el DER obtenido, entonces las tablas que se obtienen en este paso, en general, son las mismas que las obtenidas en el paso II. Supongamos que en nuestro ejemplo, este es el caso. Entonces:

Diseño lógico:

Paciente (HC, NomPac, Ed, Sexo)

Intervención (operación, TiempoProm, Espec)

Material (CódMat, DescMat, Precio)

PQ (Fecha, HC, DiagPre, Interv, Hora, Salón, Cirujano)

IO (Fecha, HoraCom, HC, HoraTerm, DiagOper, SitPac, TipoOpReal, CirReal, OperReal)

OS (Fecha, HC, Causa)

DS (Salón, DíaSem, Espec)

GMO (Fecha, HC, HoraCom, CódMat, Cant)

e) Diseño físico de la BD

En este paso hay que tener en cuenta:

. Aplicaciones a realizar sobre los datos

. Características particulares del SGBD

Ejm.: - ficheros índices

Ficheros para reportes

considerar la inclusión de campos que sean secundarios debido a la cantidad de veces que habría que calcularlos en una determinada aplicación.

separar un fichero en varios debido a la cantidad de campos y lo que se accesa cada vez.

CAPITULO 6

RESULTADO DEL ANÁLISIS

6.1 FACILIDAD DE MANEJO.

Luego de haber culminado el estudio y análisis de las herramientas de programación Visual, tales como:

1. *Microsoft Access.*
2. *Microsoft Visual Basic.*
3. *Microsoft Visual FoxPro.*
4. *Borland Delphi.*

Podemos brindar un criterio analítico referente a las bondades y potencialidades que ofrecen cada una de esta herramientas visuales de programación con respecto al ámbito financiero, en los siguientes aspectos:

- Creación de Proyectos.
- Manejo y creación de Bases de datos.
- Tipos de datos.
- Manejo de cadenas.
- Multimedia.
- Requerimientos de Hardware y Costos.

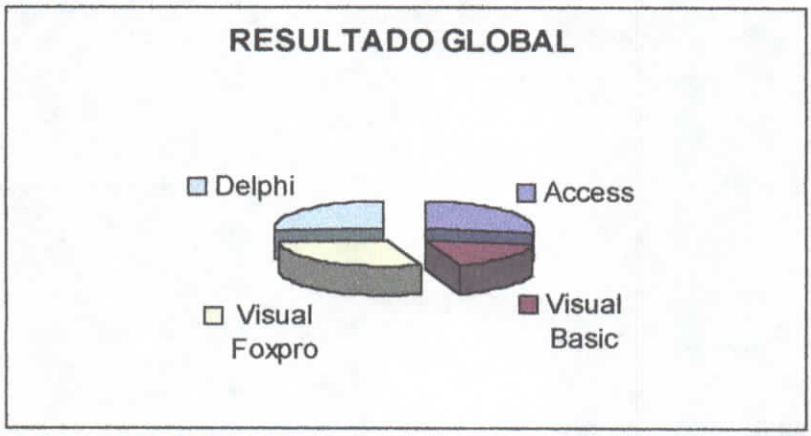
6.1.1 CREACIÓN DE PROYECTOS

Según el análisis realizado de los puntos que a continuación se detallan: contención de programas en el archivo de proyectos, facilidad de navegación entre los archivos inmiscuidos en el proyecto, herramientas disponibles para la creación del proyecto, compatibilidad con versiones anteriores, disposición de generadores, soporte a la programación orientada a objetos. Estas son las características tomadas en cuenta en cada una de las herramientas estudiadas, y de acuerdo a este análisis se puede aducir el siguiente cuadro.

	Access	Visual Basic	Visual FoxPro	Delphi
Contención de archivos	5	5	5	5
Navegación en el proyecto	5	4	5	5
Herramientas de creación	3	4	3	5
Referencia a proyectos	4	4	5	5
Compatibilidad con versio.	4	3	5	4
Generadores	4	3	5	4
Programación O. a Objetos	4	4	5	5

Estos datos son evaluados en un porcentaje del 100% en una escala del 0 al 5, vertiendo un criterio en cuanto a potencialidad en el manejo de bases de datos se refiere.

	Access	Visual Basic	Visual Foxpro	Delphi
TOTAL	28=>93.3%	18=>60%	30=>100%	27=>90%



De acuerdo al análisis realizado se puede deducir que todas las herramientas estudiadas están en capacidad de brindar seguridad y estabilidad de información, en el siguiente orden.

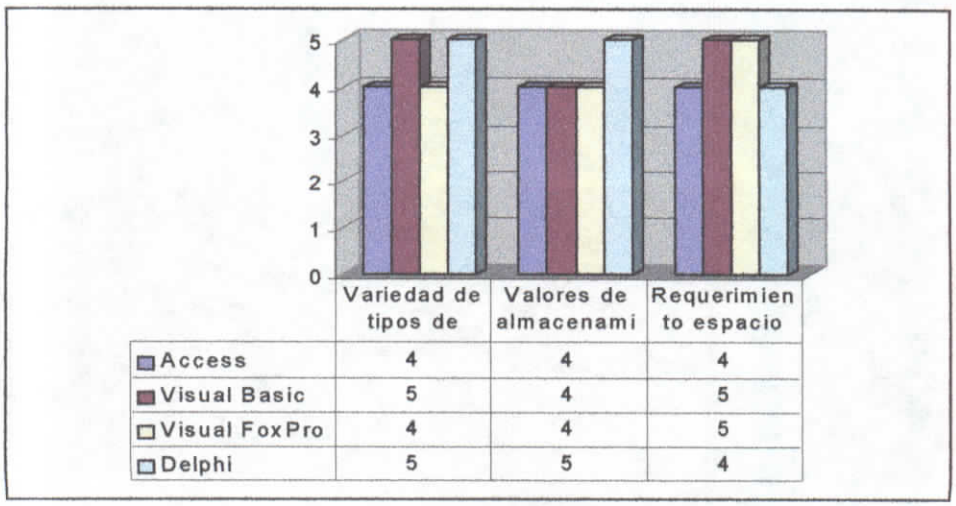
- Microsoft Visual FoxPro.
- Microsoft Access.
- Borland Delphi.
- Microsoft Visual Basic.

6.1.3 TIPOS DE DATOS.

Dentro de la construcción de sistemas, los tipos de datos que disponen las herramientas es parte primordial, debido a que estos dan la versatilidad en el ingreso de información, y puesto que los sistemas informáticos financieros necesitan de una gran variedad de estos por cuanto necesitan de gran exactitud para sus cálculos.

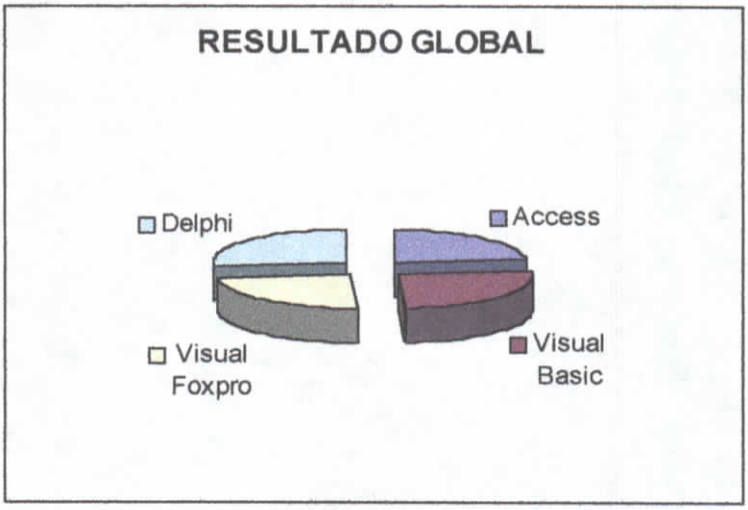
En las herramientas analizadas se han estudiado los tipos de datos con sus respectivas capacidades, con los límites máximos y mínimos en cuanto a rango de valores admitidos se refiere, y de este análisis se puede desprender el punto de vista que a continuación se detalla.

	Access	Visual Basic	Visual FoxPro	Delphi
Variedad de tipos de datos	4	5	4	5
Valores de almacenamiento	4	4	4	5
Requerimiento espacio	4	5	5	4



De la misma forma que los análisis anteriores estos valores son dados en una escala de 0 a 5 y valorados al 100%.

	Access	Visual Basic	Visual Foxpro	Delphi
TOTAL	12=>85.71%	14=>100%	13=>92.86%	14=>100%



Pudiendo brindar un criterio acerca de cual de las herramientas estudiadas podría ser la recomendada en cuanto a la versatilidad, facilidad y capacidad de datos se refiere, llegando a la siguiente conclusión.

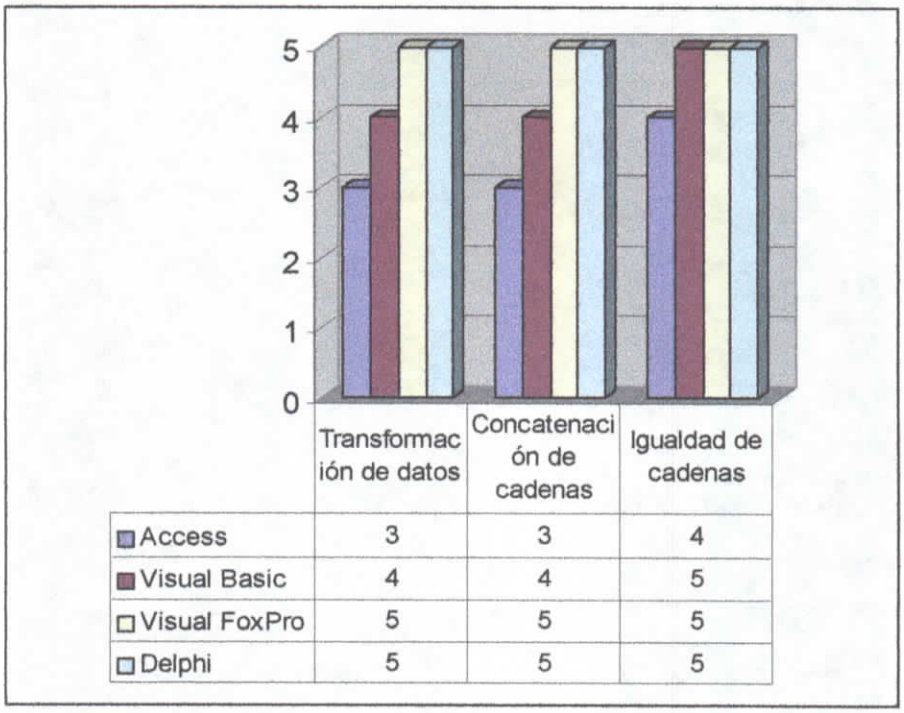
- Borland Delphi.
- Microsoft Visual Basic.
- Microsoft Visual Foxpro.
- Microsoft Access.

6.1.4 MANEJO DE CADENAS.

Este aspecto, también fue tomado en cuenta para el análisis realizado puesto que es una de las características importantes, dentro de la construcción de sistemas y/o programas. El estudio de las herramientas de programación visual nos ha conllevado a poder valorar y por ende dar un criterio de las mismas.

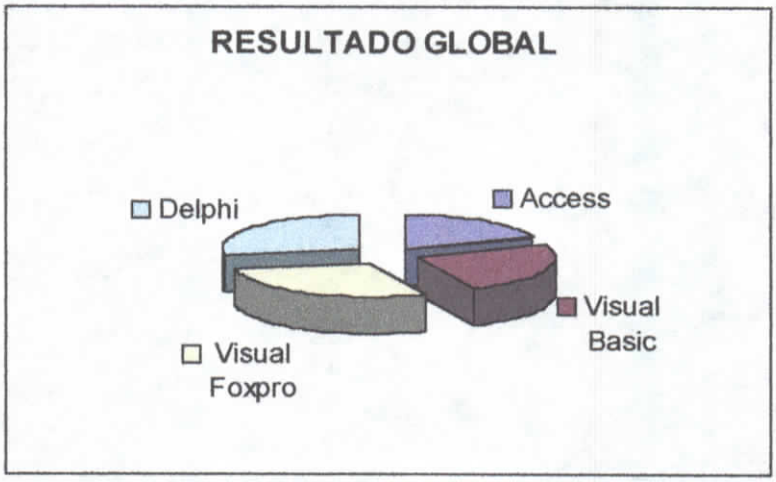
En general los aspectos estudiados y tratados sobre este punto en cuestión son: transformación de tipos de datos, concatenación de cadenas, comparar cadenas de caracteres, y dentro de esto podemos realizar el siguiente cuadro analítico.

	Access	**Visual Basic**	**Visual FoxPro**	**Delphi**
Transformación de datos	3	4	5	5
Concatenación de cadenas	3	4	5	5
Igualdad de cadenas	4	5	5	5



Este análisis también es tomado en una proporción del 100%, con una escala de 0 a 5, según el diagnóstico del cuadro se podría aducir que las herramienta óptima en el manejo de cadenas es:

	Access	Visual Basic	Visual Foxpro	Delphi
TOTAL	10=>66.67%	13=>86.6%	15=>100%	15=>100%



De acuerdo con el análisis y los resultados reflejados en los cuadros analíticos se puede recomendar la utilización de las herramientas de programación visual en el orden siguiente.

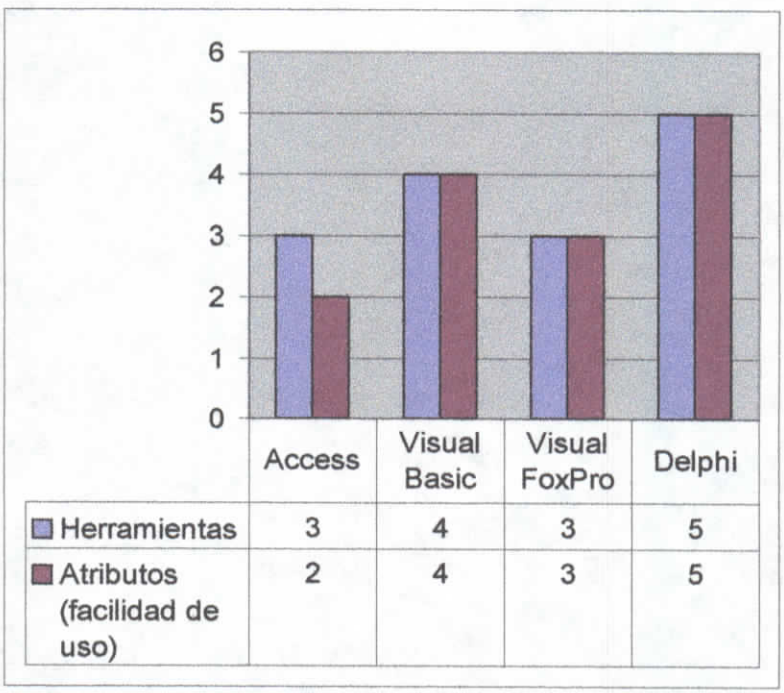
- Borland Delphi.
- Microsoft Visual FoxPro.
- Microsoft Visual Basic.
- Microsoft Access.

6.1.5 MULTIMEDIA.

En la actualidad el uso de la multimedia es muy habitual, pero dado el caso que nuestro análisis es orientado para el campo financiero, no se ha profundizado, por cuanto no es muy útil en este ámbito.

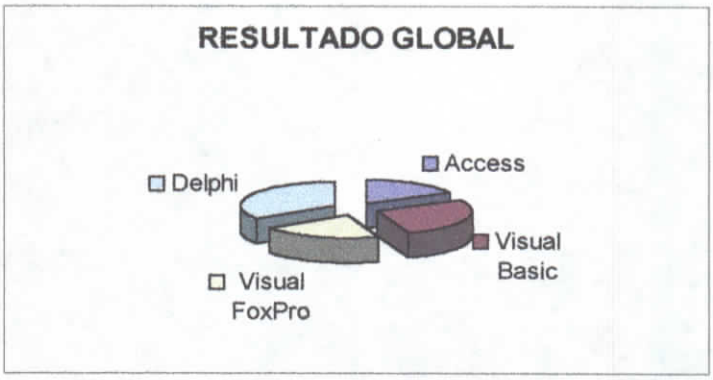
Pero sin embargo nos atreveríamos a hacer un pequeño análisis para poder recomendar la herramienta visual a utilizar en caso de que se programe con multimedia, analizando sus herramientas y sus atributos asiendo referencia a la facilidad de uso que brindan estas herramientas.

	Access	**Visual Basic**	**Visual FoxPro**	**Delphi**
Herramientas	3	4	3	5
Atributos (facilidad de uso)	2	4	3	5



Este análisis también es tomado en una proporción del 100%, con una escala de 0 a 5, según el diagnóstico del cuadro se podría aducir que las herramienta óptima en el manejo de cadenas es:

	Access	Visual Basic	Visual Foxpro	Delphi
TOTAL	5=>50%	8=>80%	6=>60%	10=>100%



De acuerdo con el estudio realizado se puede recomendar las siguientes herramientas de programación visual en el orden que a continuación se detalla.

- Borland Delphi.
- Microsoft Visual Basic.
- Microsoft Visual FoxPro.
- Microsoft Access.

CONCLUSIONES

Al término del análisis y estudio de las herramientas de programación visual se concluye que:

1. En Base a los resultados mostrados en el capítulo 6 la herramienta más adecuada para desarrollar software de apoyo financiero es el Borland Delphi, debido a que esta herramienta brinda gran facilidad en cuanto a Manejo y Creación de Proyectos Informáticos, se refiere; facilidad en el uso y desenvolvimiento con cadenas y tipos de datos, como se detalla en el capítulo 4.
2. El programador podrá tomar una decisión en cuanto al software de base que deba utilizar para el desarrollo de sus aplicaciones, en función de lo señalado en el capítulo 6 en el cual se despliegan listados en orden jerárquico de cuál de las herramientas analizadas es la óptima en cada uno de los aspectos tratados.
3. Para el Manejo y Creación de Proyectos Informáticos la mejor herramienta es el Borland Delphi, debido a la facilidad que brinda para la manipulación de archivos que están presentes en éste; su manipulación es realmente sencilla, así se lo explica en el capítulo 4.
4. La herramienta que brinda mayor facilidad para el manejo de Bases de Datos es el Microsoft Visual FoxPro por su facilidad de utilización, rapidez en la extracción de datos, versatilidad en el mantenimiento de tablas e índices.
5. La herramienta con más virtudes para el manejo de cadenas es el Borland Delphi debido a su capacidad para optimizar memoria en el uso de cadenas de extensión prácticamente ilimitadas y su gran variedad de comandos que facilitan la manipulación de dichas cadenas.

6. Por la transformación a 32 bit que Borland Delphi dio a sus tipos de datos ya existentes, además de la variedad y versatilidad de éstos, dicha herramienta es la que se considera como óptima para el manejo de los tipos de datos.
7. El software desarrollado en la SOCIEDAD FINANCIERA LEASINGCORP S.A. es una muestra clara de las bondades que presenta el Borland Delphi en cuanto a la creación de software de apoyo financiero se refiere.
8. Es posible tener un conocimiento más amplio en cuanto a la Programación Orientada a Objetos, las estructuras, implementación, desarrollo y creación de Bases de Datos.
9. Microsoft Access es una herramienta netamente de Gestión de Bases de Datos que requiere de una herramienta adicional para su total acoplamiento con la programación de Software.
10. Microsoft Access brinda gran facilidad para el uso y creación de tablas y por ende de Bases de Datos, sin embargo la estructura de sus índices tienden a tornarse lentos si las tablas tienen gran cantidad de información.
11. Microsoft Visual Basic es un lenguaje orientado netamente a la programación pero que lastimosamente no es totalmente Orientado a Objetos.
12. Microsoft Visual Basic se acopla de una forma estupenda con Microsoft Access por tanto los programadores pueden hacer uso de estas dos herramientas para así poder desarrollar software realmente poderoso.
13. Microsoft Visual Basic dispone de una gran cantidad de herramientas y tipos de datos para la creación de proyectos informáticos.
14. Microsoft Visual FoxPro es una herramienta destinada a la programación y manipulación de Bases de Datos y netamente Orientadas a Objetos.
15. Microsoft Visual FoxPro manipula datos contenidos en las tablas con una velocidad sorprendente puesto que sus índices y tablas son de fácil y rápida actualización.

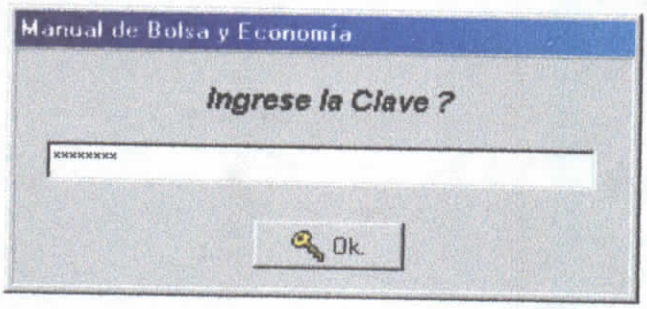
- Kroenke David, PROCESAMIENTO DE BASES DE DATOS, Editorial Prentice Hall, quinta edición.
- J.M. Rosenberg, DICCIONARIO DE ADMINISTRACIÓN Y FINANZAS, Grupo editorial Océano.
- Anderson David, Sweeney Dennis, Williams Thomas, INTRODUCCIÓN A LOS MODELOS CUANTITATIVOS PARA ADMINISTRACIÓN, Grupo editorial Iberoamerica, México 1993.
- Brealey Richard, Myres Steward, MANUAL DE FINANZAS CORPORATI, Editorial McGraw Hill, Tomo 3, Cuarta Edición, Colombia 1995.
- www.microsoft.com
- www.borland.com
- www.imprice.com

ANEXO 1

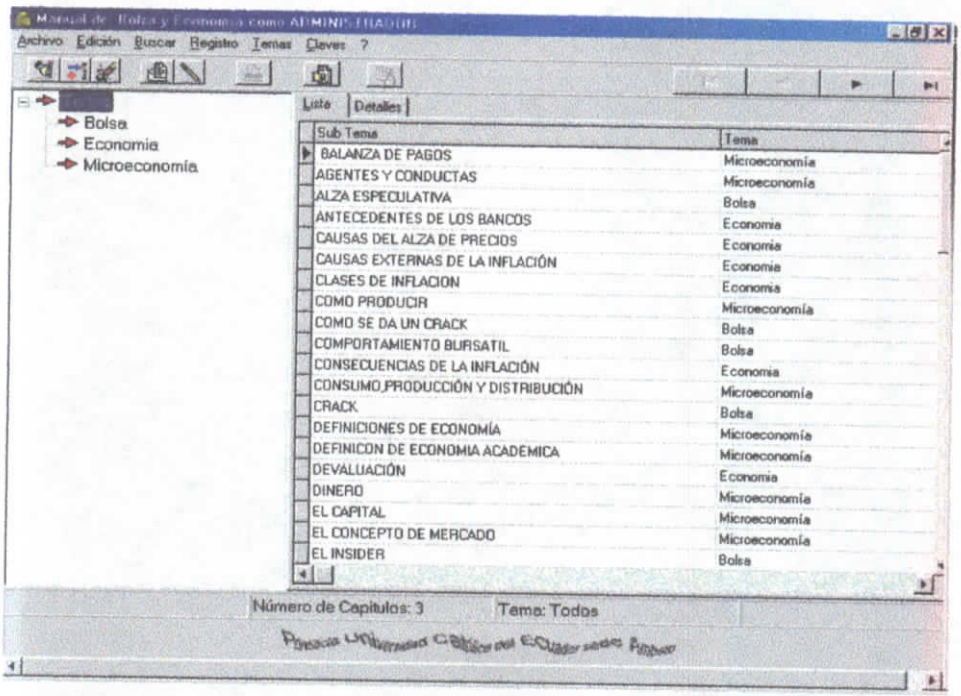
CONTENIDO DE LA APLICACIÓN

1. Instalación de la aplicación.
2. Pantalla de Inicialización.
3. Pantalla de Registro de Usuario.
4. Aplicación desplegada.
5. Pantallas de navegación por su contenido.
    - Impresiones.
    - Copiar.
    - Cambiar tipos de letras.
    - Búsqueda de registros.
    - Navegación por registros.
    - Insertar registros.
    - Editar registros.
    - Borrar registros.
    - Insertar temas.
    - Borrar Temas.
    - Ayuda.
    - Acerca de.

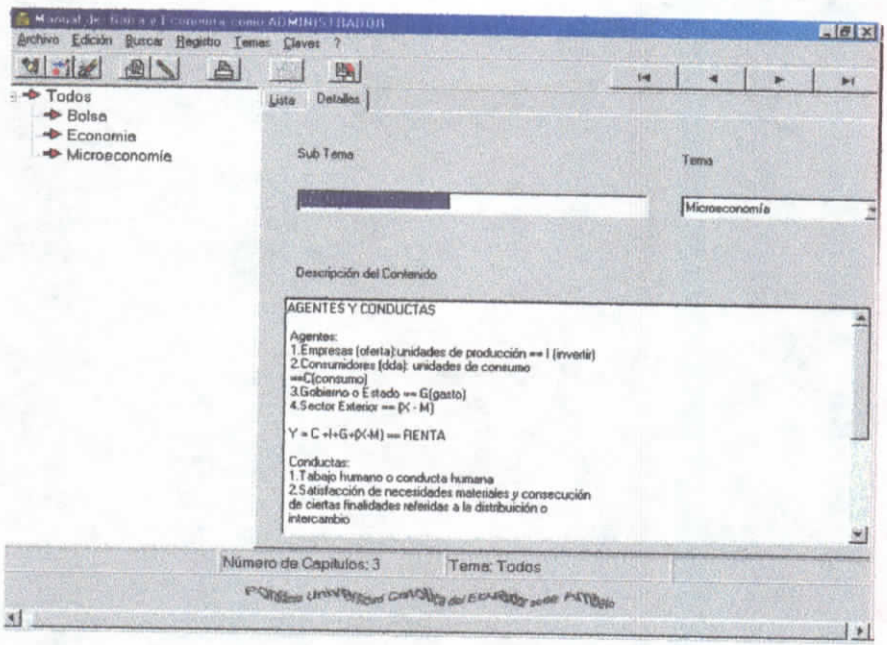
PANTALLA DE INGRESO



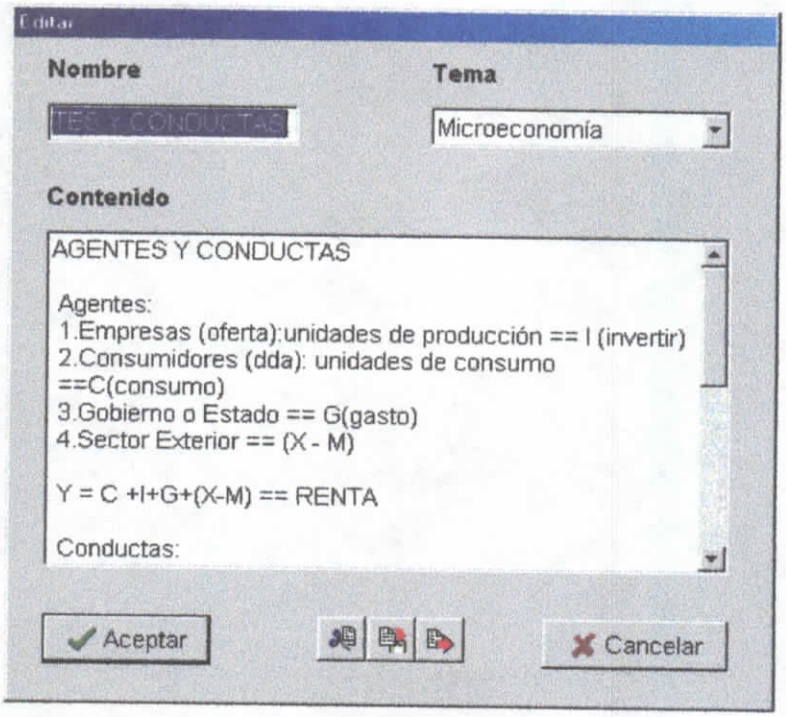
MANUAL DE BOLSA Y ECONOMÍA



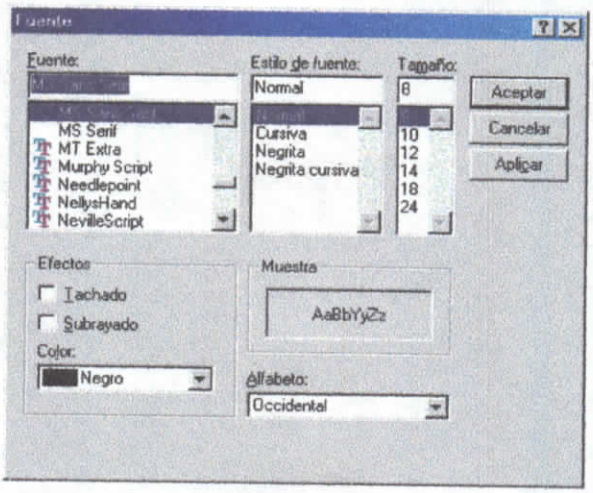
PANTALLA QUE MUESTRA EL CONTENIDO DEL SUBTEMA



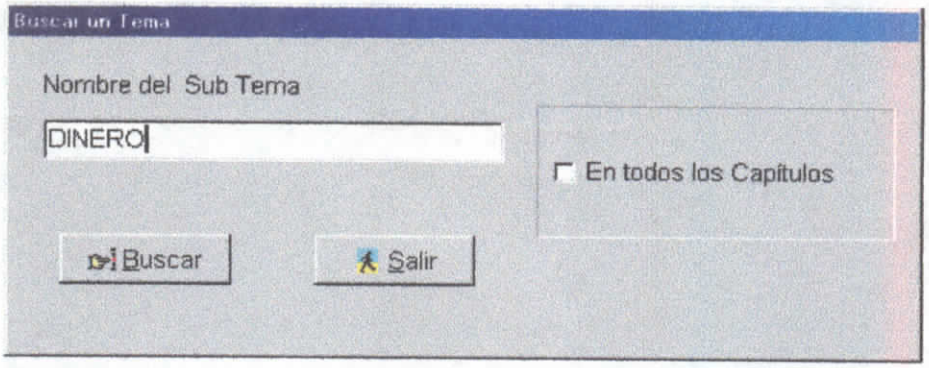
FORMA PARA EDITAR SUBTEMAS (Cortar, Copiar y Pegar).



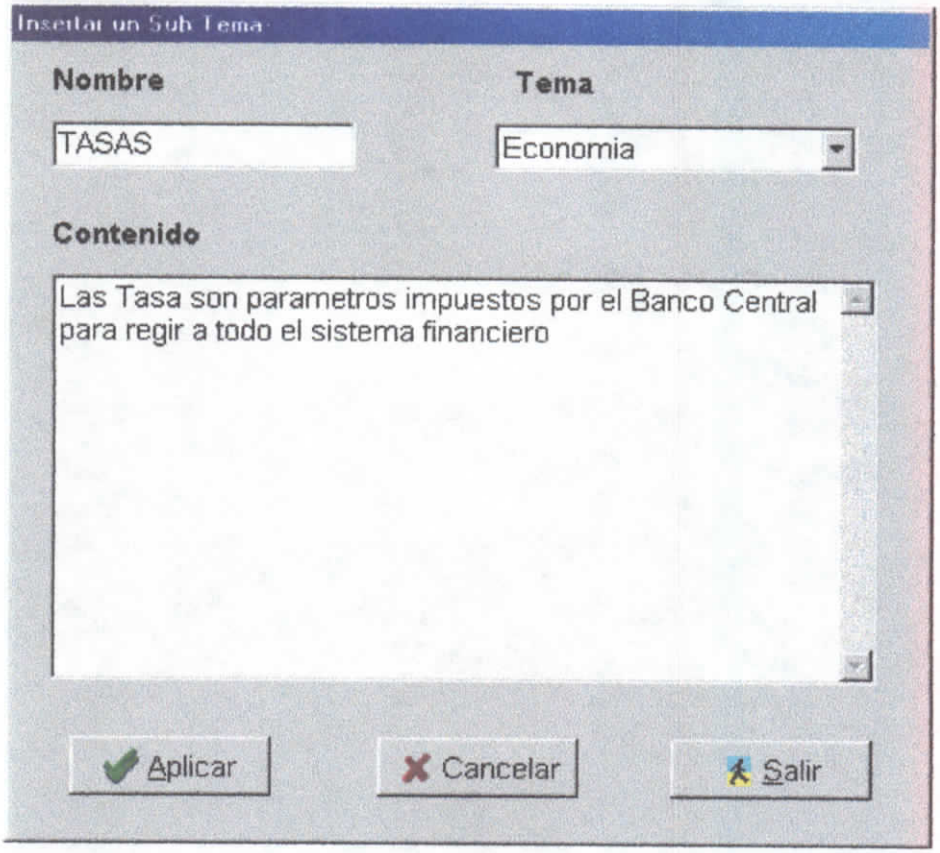
PANTALLA PARA CAMBIAR LAS FUENTES



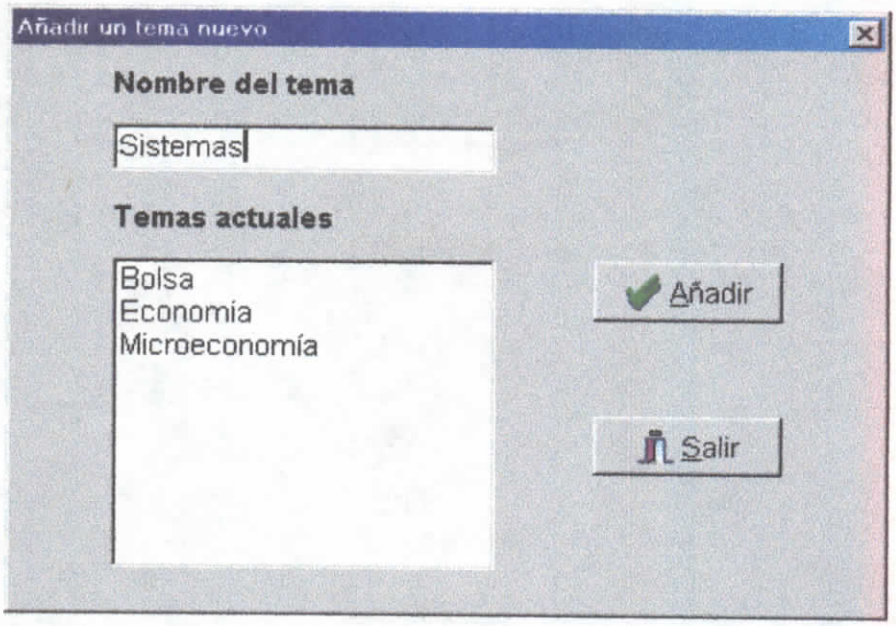
FORMA PARA BUSCAR SUB TEMAS



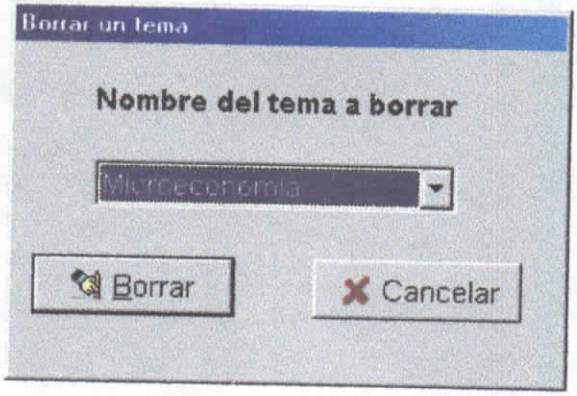
FORMA PARA INSERTAR SUB TEMA



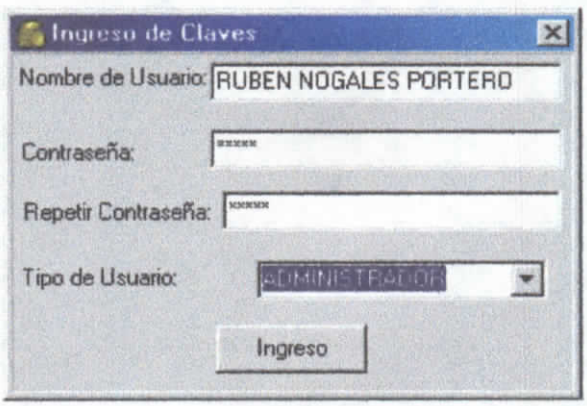
FORMA PARA INSERTAR UN NUEVO TEMA

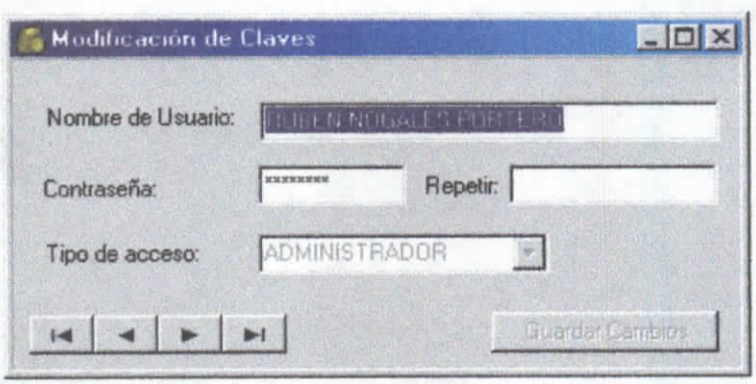


FORMA PARA BORRAR UN TEMA

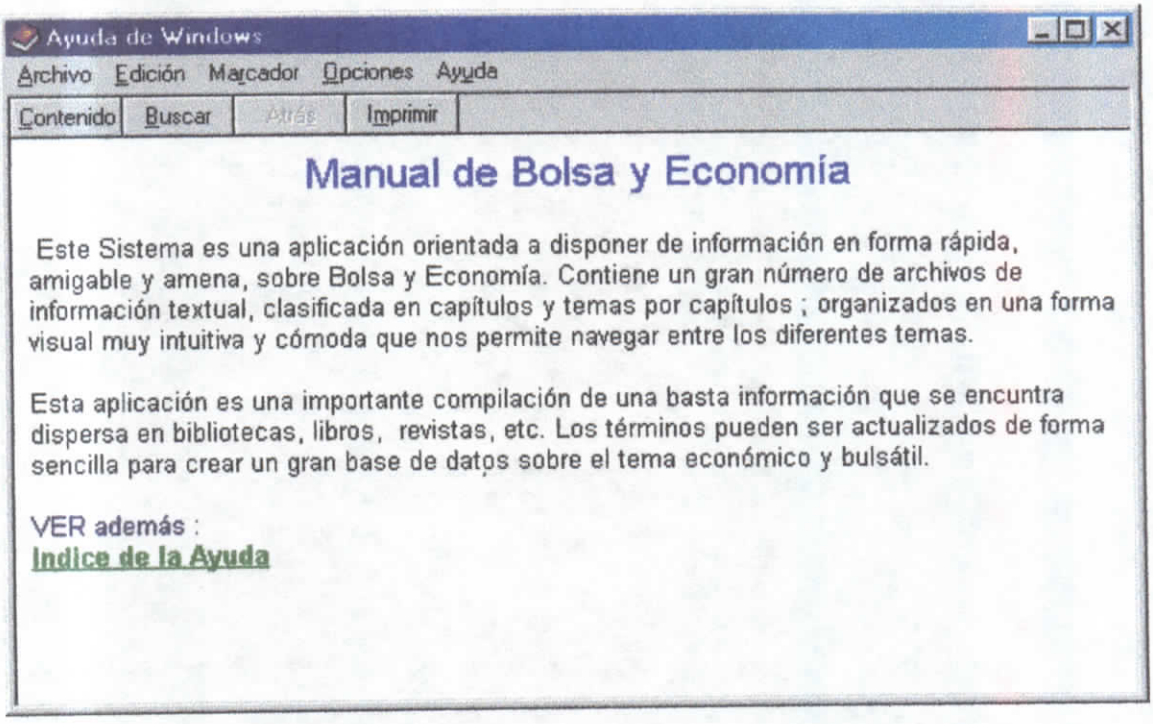


FORMAS PARA INGRESAR Y MODIFICAR CLAVES COMO ADMINISTRADOR O COMO CLIENTE RESPECTIVAMENTE





AYUDA DEL MANUAL DE BOLSA Y ECONOMÍA



PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
PROGRAMAS ACADÉMICOS AMBATO
BIBLIOTECA