



**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**BEKMAR PROGRAMMING  
SYSTEM WITH DELPHI**

**Graduation Project  
COM400**

**Student: Kadir BEKİROĞLU (20020565)**

**Supervisor: Mr. Elbrus IMANOV**

**Lefkoşa-2007**

*To my parents*

© Kadir BEKİROĞLU  
All Rights Reserved 2007

<b>TABLE OF CONTENT</b>	
<b>TABLE OF CONTENT</b>	<b>I</b>
<b>ACKNOWLEDGMENTS</b>	<b>IV</b>
<b>ABSTRACT</b>	<b>V</b>
<b>INTRODUCTION</b>	<b>VI</b>
<b>CHAPTER 1: DELPHI</b>	<b>1</b>
1.1 Introduction to Delphi	1
1.2 First Delphi Program	1
1.3 Delphi Style	4
1.4 Introduction To Design Patterns In Delphi	6
1.5 How Delphi Helps You Define Patterns	6
1.5.1 Delphi Examples of Design Patterns	7
1.5.2 Pattern: Singleton	8
1.5.2.1 Applications in Delphi	8
1.5.2.2 Implementation Example	8
1.5.3 Pattern: Adapter	13
1.5.3.1 Implementation Example	13
1.5.4 Pattern: Template Method	22
1.5.4.1 Applications in Delphi	22
1.5.5 Pattern: Builder	25
1.5.5.1 Applications in Delphi	25
1.5.5.2 Implementation Example	25
1.5.6 Pattern: Abstract Factory	29
1.5.6.1 Applications in Delphi	29
1.5.6.2 Implementation Example	29
1.5.7 Pattern: Factory Method	31
1.5.7.1 Applications in Delphi	31
1.5.7.2 Implementation Example	31
1.6 APPENDIX : Key Elements of Delphi Class Definitions	32
1.6.1 Unit Structure	32

<b>1.6.1 Class Interfaces</b>	<b>33</b>
<b>1.6.3 Properties</b>	<b>33</b>
<b>1.6.4 Inheritance</b>	<b>33</b>
<b>1.6.5 Abstract Methods</b>	<b>35</b>
<b>1.6.6 Messages</b>	<b>36</b>
<b>1.6.7 Events</b>	<b>36</b>
<b>1.6.8 Constructors and Destructors</b>	<b>36</b>
<b>1.7 Delphi Compilers</b>	<b>37</b>
<b>1.7.1 Free Delphi IDEs and Compilers</b>	<b>37</b>
<b>1.7.2 Free Delphi Compression Libraries</b>	<b>37</b>
<b>1.7.3 Free Delphi Script Engines</b>	<b>38</b>
<b>1.7.4 Free Database Components</b>	<b>38</b>
<b>1.7.5 Delphi UI Components</b>	<b>39</b>
<b>1.7.6 Printing and Reports</b>	<b>40</b>
<b>1.7.7 Free Delphi Unicode Libraries</b>	<b>40</b>
<b>1.7.8 Delphi Component Directories</b>	<b>41</b>
<b>1.7.9 Free Delphi Libraries, Components, Utilities</b>	<b>42</b>
<b>1.7.10 Graphics Libraries</b>	<b>43</b>
<b>1.7.11 3D Programming</b>	<b>44</b>
<b>1.7.12 Delphi Game Programming</b>	<b>45</b>
<b>1.7.13 Delphi Programs with Source</b>	<b>46</b>
<b>1.7.14 Internet and Communication Components</b>	<b>47</b>
<b>1.7.15 Core Delphi</b>	<b>47</b>
<b>1.7.16 Resource Editors</b>	<b>48</b>
<b>1.7.17 Delphi Tools</b>	<b>48</b>
<b>1.7.18 Delphi Magazines</b>	<b>48</b>
<b>1.7.19 Delphi Communities</b>	<b>48</b>
<b>1.7.20 Tutorials</b>	<b>49</b>
<b>1.7.21 Delphi Resources</b>	<b>49</b>
<b>1.8 Conclusion About Delphi</b>	<b>50</b>



<b>CHAPTER 2 DATABASE</b>	<b>53</b>
2.1 Introduction to Database	53
2.2 De-merits of Absence of Database	54
2.3 Merits of database	54
2.4 Introduction to Database Design	54
2.5 Database Models	55
2.5.1 Flat Model	56
2.5.2 Network Model	56
2.5.3 Relational Model	56
2.5.3.1 Why we use a Relational Database Design	57
2.6 Relationships between Tables	58
2.6.1 One-To-One Relationships	58
2.6.2 One-To-Many Relationships	58
2.7 Data Modeling	59
2.7.1 Database Normalization	59
2.7.2 Primary Key	60
2.7.3 Foreign Key	61
2.7.4 Compound Key	61
<b>CHAPTER 3 USER'S MANUAL</b>	<b>62</b>
<b>CONCLUSION</b>	<b>72</b>
<b>APPENDIX</b>	<b>73</b>

## ACKNOWLEDGMENT

*First of all I am happy to complete the task which I had given with blessing of God and also I am grateful to all the people in my life who have, supported me, advised me. Taught me and who have always encouraged me to follow my dreams and ambitions.*

*I wish to thank my supervisor, **Mr. ELBRUS IMANOV**, for intellectual support, encouragement, and enthusiasm, which made this project possible, and his patience for correcting both my stylistic and scientific errors.*

*And thank my dearest parents who encouraged me to continue beyond my undergraduate studies, to my father who proceeded before me and to my mother who encouraged me along the way.*

*To all my friends, especially **Ahmet Kayabaş, Fetullah Akatay and Hadi Turus** for sharing wonderful moments, advice, and for making me feel at home.*

*And above, I thank God for giving me stamina and courage to achieve my objectives.*

**KADİR BEKİROĞLU**

## **ABSTRACT**

Data, gathered around us as a collection of facts, is of no use unless it is organized and represented in some meaningful form. Data represented in some meaningful form like, tables, charts, or graphs become information, which can be easily processed. The collection of data, usually referred to as the database, contains information about one particular enterprise. These days databases are used by a variety of users and organizations, which are important tools in processing DBMS, which are designed to manage large amount of data.

This project has as its goal to develop software, processing information about activities of a computer-part sales company. Software developed in this project contains both employee information, and information associated with sales and purchase of computer parts. The project can be developed by improving the software for processing all activities of the company.

## INTRODUCTION

Nowadays the technology is developed a lot and started to use by anyone in the world no matter who he/she is. Because of the technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.

The automation is also became a part of our lives. The people operate with automation systems in everywhere. My project is Bekmar Program System. This Automation is used to keep the information about the receiving, coming and going products and sales and price.

Bekmar Program System is used in every super market, product, to storage databases and folders.

In my project the main point is making the user's job easy. It lets to the manager to documents, product, supplier and tax information easily. And we he/she can get the data report specific date, type who receiving documents.



# **CHAPTER 1**

## **DELPHI**

### **1.1 INTRODUCTION TO DELPHI**

Although I am not the most experienced or knowledgeable person on the forums I thought it was time to write a good introductory article for Delphi beginners.

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

For the purposes of this series I will be using Delphi 7. There are more recent versions available (2005 and 2006) however Delphi 7 should be available inexpensively compared to the new versions which will set you back a lot of money. Delphi 7 will more than likely be available in a magazine for free.

### **1.2 FIRST DELPHI PROGRAM**

First thing is first, fire up your copy of Delphi and open the Project > Options menu. To compile a console application you need to change a setting on the Linker tab called 'Generate console application', check the box and click OK. Now select File > Close All if anything is already loaded.

Then select File > New > Other > Console Application.

You should have something like this:

Delphi Code:

```
program Project1;
```

```
  {$APPTYPE CONSOLE}
```

```
uses
```

```
  SysUtils;
```

```
begin
```

```
  { TODO -oUser -cConsole Main : Insert code here }
```

```
end.
```

Notice the first line refers to the keyword program. You can rename this to HelloWorld.

You can also remove the commented portion enclosed in curly brackets.

The uses keyword allows you to list all units that you want to use in the program. At the moment just leave it as it is, SysUtils is all we need.

Your unit should now look like this:

Delphi Code:

```
program HelloWorld;
```

```
  {$APPTYPE CONSOLE}
```

```
uses
```

```
  SysUtils;
```

```
begin
```

```
end.
```

Now what we have just done is written a program, it currently doesn't do a thing however. Hit the run button and see the result. Now wasn't that completely worthless.

Luckily this isn't the end of the article so we'll actually have a worthwhile program at the end of it. All we need to do is insert some code in the main procedure we have just made.

Every good programmer's first program was 'Hello World' and you'll be no exception. All we need to do is use the WriteLn procedure to write 'Hello World!' to the console, simple.

Delphi Code:

```
program HelloWorld;
```

```
  {$APPTYPE CONSOLE}
```

```
  uses
```

```
    SysUtils;
```

```
  begin
```

```
    WriteLn('Hello World!')
```

```
  end.
```

Notice the semicolon at the end of the line, at the end of any statement you need to add a semicolon. Run the program and see the results...

Now I don't know about you but I saw hello world flash up and go away in a second, if you didn't write the program you wouldn't even know what it said. To solve this problem we need to tell the program to leave the console open until the user is ready to close it. We can use ReadLn for this which reads the users input from the console.

Delphi Code:

```
program HelloWorld;
```

```
{ $APPTYPE CONSOLE }
```

```
uses
```

```
  SysUtils;
```

```
begin
```

```
  WriteLn('Hello World!' + #13#10 + #13#10 +
```

```
    'Press RETURN to end...');
```

```
  ReadLn;
```

```
end.
```

I have added a few extra things into the 'Hello World' string so the user knows what to do to end the program as it could be a bit confusing. '#13#10' is to insert a carriage return as 13 and 10 are the ASCII codes for a carriage return followed by a new line feed. ASCII can be inserted in this way into strings.

### 1.3 DELPHI STYLE

You mean fashion? No, not which colour is this seasons black. I mean you coding style, the way you format your code and the way in which you present it on the page.

Q - At the end of the day who cares about my style, I can read it, and Delphi strips all the spaces out of it and doesn't care if I indent. Why waste my time?

A - Good question. Neatly present code which conforms to the accepted standards not only makes your code much easier for you to read and debug but also but any one else who might read your code to help you, or learn from you can do so with ease. After all which code is easier to follow, example 1 or 2?

Delphi Code:



// Example 1

```
procedure xyz();  
var  
x,y,z,a:integer;  
begin  
x:=1;y:=2;  
for z:=x to y do begin  
a:=power(z,y);  
showmessage(inttostr(a));  
end;  
end;
```

Delphi Code:

// Example 2

```
procedure XYZ();  
var  
X,Y,Z,A: Integer;  
begin  
X := 1;  
Y := 2;  
for Z := X to Y do  
begin  
A := Power(Z, Y);  
ShowMessage(IntToStr(A));  
end; // for end  
end; // procedure end
```

I think we'd both agree that although both examples do the same thing example two is set out in an easy to read manner where as example 1 is a mess.

To learn more about how to style your code take a look at BDN's Object Pascal Style Guide outlining the conventions set by the Delphi Team.

You see what power is - holding someone else's fear in your hand and showing it to them!

## **1.4 INTRODUCTION TO DESIGN PATTERNS IN DELPHI**

Design patterns are frequently recurring structures and relationships in object-oriented design. Getting to know them can help you design better, more reusable code and also help you learn to design more complex systems.

Much of the ground-breaking work on design patterns was presented in the book Design Patterns: Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides. You might also have heard of the authors referred to as "the Gang of Four". If you haven't read this book before and you're designing objects, it's an excellent primer to help structure your design. To get the most out of these examples, I recommend reading the book as well.

This paper takes some sample patterns from Design Patterns and discusses their implementation in Delphi.

Another good source of pattern concepts is the book Object Models: Strategies, Patterns and Applications by Peter Coad. Coad's examples are more business oriented and he emphasises learning strategies to identify patterns in your own work.

## **1.5 HOW DELPHI HELPS YOU DEFINE PATTERNS**

Delphi implements a fully object-oriented language with many practical refinements that simplify development.

A summary of Delphi class concepts can be found in the following [appendix](#).

The most important class attributes from a pattern perspective are the basic inheritance of classes; virtual and abstract methods; and use of protected and public scope. These give you the tools to create patterns that can be reused and extended, and let you isolate varying functionality from base attributes that are unchanging.

Delphi is a great example of an extensible application, through its component architecture, IDE interfaces and tool interfaces. These interfaces define many virtual and abstract constructors and operations.

### 1.5.1 Delphi Examples of Design Patterns

I should note from the outset, there may be alternative or better ways to implement these patterns and I welcome your suggestions on ways to improve the design. The following patterns from the book Design Patterns are discussed and illustrated in Delphi to give you a starting point for implementing your own Delphi patterns.

Pattern Name	Definition
Singleton	"Ensure a class has only one instance, and provide a global point of access to it."
Adapter	"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."
Template Method	"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."
Builder	"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

Abstract Factory	"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."
Factory Method	"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

Note: These definitions are taken from Design Patterns.

### **1.5.2 Pattern: Singleton**

Definition : "Ensure a class has only one instance, and provide a global point of access to it."

This is one of the easiest patterns to implement.

#### **1.5.2.1 Applications in Delphi**

There are several examples of this sort of class in the Delphi VCL, such as TApplication, TScreen or TClipboard. The pattern is useful whenever you want a single global object in your application. Other uses might include a global exception handler, application security, or a single point of interface to another application.

#### **1.5.2.2 Implementation Example**

To implement a class of this type, override the constructor and destructor of the class to refer to a global (interface) variable of the class.

Abort the constructor if the variable is assigned, otherwise create the instance and assign the variable.

In the destructor, clear the variable if it refers to the instance being destroyed.

Note: To make the creation and destruction of the single instance automatic, include its creation in the initialization section of the unit. To destroy the instance, include its destruction in an ExitProc (Delphi 1) or in the finalization section of the unit (Delphi 2).



The following Delphi 1 example illustrates two singleton classes, one derived from TComponent and another derived from TObject.

```
unit Singletn;
```

```
interface
```

```
uses
```

```
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
```

```
  Forms, Dialogs;
```

```
type
```

```
  TCSingleton = class(TComponent)
```

```
  public
```

```
    constructor Create(AOwner: TComponent); override;
```

```
    destructor Destroy; override;
```

```
  end;
```

```
  TOSingleton = class(TObject)
```

```
  public
```

```
    constructor Create;
```

```
    destructor Destroy; override;
```

```

end;

var
    Global_CSingleton: TCSingleton;

    Global_OSingleton: TOSingleton;

procedure Register;

implementation

procedure Register;

begin
    RegisterComponents('Design Patterns', [TCSingleton]);

end;

{ TCSingleton }

constructor TCSingleton.Create(AOwner: TComponent);

begin
    if Global_CSingleton <> nil then

        {NB could show a message or raise a different exception here}

        Abort

```

else begin

inherited Create(AOwner);

Global\_CSingleton := Self;

end;

end;

destructor TCSingleton.Destroy;

begin

if Global\_CSingleton = Self then

Global\_CSingleton := nil;

inherited Destroy;

end;

{ TOSingleton }

constructor TOSingleton.Create;

begin

if Global\_OSingleton <> nil then

{NB could show a message or raise a different exception here}

```

    Abort

    Courier

else

    Global_OSingleton := Self;

end;
destructor TOSingleton.Destroy;

begin

    if Global_OSingleton = Self then

        Global_OSingleton := nil;

        inherited Destroy;

    end;

procedure FreeGlobalObjects; far;

begin

    if Global_CSingleton <> nil then

        Global_CSingleton.Free;

    if Global_OSingleton <> nil then

        Global_OSingleton.Free;

    end;

```



begin

AddExitProc(FreeGlobalObjects);

end.

### 1.5.3 Pattern: Adapter

Definition : "Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

A typical example of this is the wrapper Delphi generates when you import a VBX or OCX. Delphi generates a new class which translates the interface of the external control into a Pascal compatible interface. Another typical case is when you want to build a single interface to old and new systems.

Note Delphi does not allow class adaption through multiple inheritance in the way described in Design Patterns. Instead, the adapter needs to refer to a specific instance of the old class.

#### 1.5.3.1 Implementation Example

The following example is a simple (read only) case of a new customer class, an adapter class and an old customer class. The adapter illustrates handling the year 2000 problem, translating an old customer record containing two digit years into a new date format. The client using this wrapper only knows about the new customer class. Translation between classes is handled by the use of virtual access methods for the properties. The old customer class and adapter class are hidden in the implementation of the unit.

unit Adapter;

interface

uses SysUtils, Classes;

type

{ The new class }

TNewCustomer = class

private

FCustomerID: Longint;

FFirstName: string;

FLastName: string;

FDOB: TDateTime;

protected

function GetCustomerID: Longint; virtual;

function GetFirstName: string; virtual;

function GetLastName: string; virtual;

function GetDOB: TDateTime; virtual;

public

constructor Create(CustID: Longint); virtual;

property CustomerID: Longint read GetCustomerID;

```

property FirstName: string read GetFirstName;

property LastName: string read GetLastName;

property DOB: TDateTime read GetDOB;

end;
{ An interface method }

{ Lets us hide details of TOldCustomer from the client }

function GetCustomer(CustomerID: Longint): TNewCustomer;

implementation

const

    Last_OldCustomer_At_Year_2000 = 15722;

    Last_OldCustomer_In_Database = 30000;

{ The new class }

constructor TNewCustomer.Create(CustID: Longint);

begin

    FCustomerID := CustID;

    FFirstName := 'A';

    FLastName := 'New_Customer';

```

```

    FDOB := Now;

end;

function TNewCustomer.GetCustomerID: Longint;

begin

    Result := FCustomerID;

end;

function TNewCustomer.GetFirstName: string;

begin

    Result := FFirstName;

end;

function TNewCustomer.GetLastName: string;

begin

    Result := FLastName;

end;

function TNewCustomer.GetDOB: TDateTime;

begin

    Result := FDOB;

```



end;

type

{ The old class }

TOldDOB = record

Day: 0..31;

Month: 1..12;

Year: 0..99;

end;

TOldCustomer = class

FCustomerID: Integer;

FName: string;

FDOB: TOldDOB;

public

constructor Create(CustID: Integer);

property CustomerID: Integer read FCustomerID;

property Name: string read FName;

property DOB: TOldDOB read FDOB;

```

function GetFirstName: string; override;

function GetLastName: string; override;

function GetDOB: TDateTime; override;

public

constructor Create(CustID: Longint); override;

destructor Destroy; override;

end;

{ The Adapter class }

constructor TAdaptedCustomer.Create(CustID: Longint);

begin

    inherited Create(CustID);

    FOldCustomer := TOldCustomer.Create(CustID);

end;

destructor TAdaptedCustomer.Destroy;

begin

    FOldCustomer.Free;

```

inherited Destroy;

end;

function TAdaptedCustomer.GetCustomerID: Longint;

begin

Result := FOldCustomer.CustomerID;

end;

function TAdaptedCustomer.GetFirstName: string;

var

SpacePos: integer;

begin

SpacePos := Pos(' ', FOldCustomer.Name);

if SpacePos = 0 then

Result := "

else

Result := Copy(FOldCustomer.Name,1,SpacePos-1);

end;

function TAdaptedCustomer.GetLastName: string;

var

SpacePos: integer;

begin

SpacePos := Pos(' ', FOldCustomer.Name);

if SpacePos = 0 then

Result := FOldCustomer.Name

else

Result := Copy(FOldCustomer.Name, SpacePos+1, 255);

end;

function TAdaptedCustomer.GetDOB: TDateTime;

var

FullYear: Word;

begin

if CustomerID > Last\_OldCustomer\_At\_Year\_2000 then

FullYear := 2000 + FOldCustomer.DOB.Year

else

FullYear := 1900 + FOldCustomer.DOB.Year;

```
Result      :=      EncodeDate(FullYear,      FOldCustomer.DOB.Month,  
FOldCustomer.DOB.Day);
```

```
end;
```

```
function GetCustomer(CustomerID: Longint): TNewCustomer;
```

```
begin
```

```
    if CustomerID > Last_OldCustomer_In_Database then
```

```
        Result := TNewCustomer.Create(CustomerID)
```

```
    else
```

```
        Result := TAdaptedCustomer.Create(CustomerID) as TNewCustomer;
```

```
end;
```

```
end.
```

#### **1.5.4 Pattern: Template Method**

Definition: "Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

This pattern is essentially an extension of abstract methods to more complex algorithms.

##### **1.5.4.1 Applications in Delphi**

Abstraction is implemented in Delphi by abstract virtual methods. Abstract methods differ from virtual methods by the base class not providing any implementation. The



descendant class is completely responsible for implementing an abstract method. Calling an abstract method that has not been overridden will result in a runtime error.

A typical example of abstraction is the TGraphic class.

TGraphic is an abstract class used to implement TBitmap, TIcon and TMetafile. Other developers have frequently used TGraphic as the basis for other graphics objects such as PCX, GIF, JPG representations. TGraphic defines abstract methods such as Draw, LoadFromFile and SaveToFile which are then overridden in the concrete classes. Other objects that use TGraphic, such as a TCanvas only know about the abstract Draw method, yet are used with the concrete class at runtime.

Many classes that use complex algorithms are likely to benefit from abstraction using the template method approach. Typical examples include data compression, encryption and advanced graphics processing.

#### Implementation Example

To implement template methods you need an abstract class and concrete classes for each alternate implementation. Define a public interface to an algorithm in an abstract base class. In that public method, implement the steps of the algorithm in calls to protected abstract methods of the class. In concrete classes derived from the base class, override each step of the algorithm with a concrete implementation specific to that class.

This example shows some very simple algorithm steps, but illustrates the principle of deferring implementation to a subclass.

```
unit Tpl_meth;
```

```
interface
```

```
type
```

```
TAbstractTemplateClass = class(TObject)
```

protected

function Algorithm\_StepA: Integer; virtual; abstract;

function Algorithm\_StepB: Integer; virtual; abstract;

function Algorithm\_StepC: Integer; virtual; abstract;

public

function Algorithm: Integer;

end;

TConcreteClassA = class(TAbstractTemplateClass)

protected

function Algorithm\_StepA: Integer; override;

function Algorithm\_StepB: Integer; override;

function Algorithm\_StepC: Integer; override;

end;

TConcreteClassB = class(TAbstractTemplateClass)

protected

function Algorithm\_StepA: Integer; override;

function Algorithm\_StepB: Integer; override;

```
function Algorithm_StepC: Integer; override;  
  
end;
```

### **1.5.5 Pattern: Builder**

Definition : "Separate the construction of a complex object from its representation so that the same construction process can create different representations."

A Builder seems similar in concept to the Abstract Factory. The difference as I see it is the Builder refers to single complex objects of different concrete classes but containing multiple parts, whereas the abstract factory lets you create whole families of concrete classes. For example, a builder might construct a house, cottage or office. You might employ a different builder for a brick house or a timber house, though you would give them both similar instructions about the size and shape of the house. On the other hand the factory generates parts and not the whole. It might produce a range of windows for buildings, or it might produce a quite different range of windows for cars.

#### **1.5.5.1 Applications in Delphi**

*The functionality used in Delphi's VCL to create forms and components is similar in concept to the builder. Delphi creates forms using a common interface, through Application.CreateForm and through the TForm class constructor. TForm implements a common constructor using the resource information (DFM file) to instantiate the components owned by the form. Many descendant classes reuse this same construction process to create different representations. Delphi also makes developer extensions easy. TForm's OnCreate event also adds a hook into the builder process to make the functionality easy to extend.*

#### **1.5.5.2 Implementation Example**

The following example includes a class TAbstractFormBuilder and two concrete classes TRedFormBuilder and TBlueFormBuilder. For ease of development some common

functionality of the concrete classes has been moved into the shared TAbstractFormBuilder class.

type

```
TAbstractFormBuilder = class
```

```
private
```

```
  FForm: TForm;
```

```
  procedure BuilderFormClose(Sender: TObject; var Action: TCloseAction);
```

```
protected
```

```
  function GetForm: TForm; virtual;
```

```
public
```

```
  procedure CreateForm(AOwner: TComponent); virtual;
```

```
  procedure CreateSpeedButton; virtual; abstract;
```

```
  procedure CreateEdit; virtual; abstract;
```

```
  procedure CreateLabel; virtual; abstract;
```

```
  property Form: TForm read GetForm;
```

```
end;
```

type

```
TRedFormBuilder = class(TAbstractFormBuilder)
```



private

FNextLeft, FNextTop: Integer;

public

procedure CreateForm(AOwner: TComponent); override;

procedure CreateSpeedButton; override;

procedure CreateEdit; override;

procedure CreateLabel; override;

end;

type

TBlueFormBuilder = class(TAbstractFormBuilder)

private

FNextLeft, FNextTop: Integer;

public

procedure CreateForm(AOwner: TComponent); override;

procedure CreateSpeedButton; override;

procedure CreateEdit; override;

procedure CreateLabel; override;

end;

At runtime the client application instructs one of the concrete classes to create parts using the public part creation procedures. The concrete builder instance is passed to the following procedure:

```
procedure TForm1.Create3ComponentFormUsingBuilder(ABuilder:
TAbstractFormBuilder);
var
    NewForm: TForm;

begin
    with ABuilder do begin
        CreateForm(Application);

        CreateEdit;

        CreateSpeedButton;

        CreateLabel;

        NewForm := Form;

        if NewForm <> nil then NewForm.Show;
    end;
end;
```

### 1.5.6 Pattern: Abstract Factory

Definition: "Provide an interface for creating families of related or dependant objects without specifying their concrete classes."

The Factory Method pattern below is commonly used in this pattern.

#### 1.5.6.1 Applications in Delphi

This pattern is ideal where you want to isolate your application from the implementation of the concrete classes. For example if you wanted to overlay Delphi's VCL with a common VCL layer for both 16 and 32 bit applications, you might start with the abstract factory as a base.

#### 1.5.6.2 Implementation Example

The following example uses an abstract factory and two concrete factory classes to implement different styles of user interface components. TOAbstractFactory is a singleton class, since we usually want one factory to be used for the whole application.

```
TOAbstractFactory = class(TObject)
```

```
public
```

```
    constructor Create;
```

```
    destructor Destroy; override;
```

```
    { abstract widget constructors }
```

```
    function CreateSpeedButton(AOwner: TComponent): TSpeedButton; virtual;  
    abstract;
```

```
    function CreateEdit(AOwner: TComponent): TEdit; virtual; abstract;
```

```
function CreateLabel(AOwner: TComponent): TLabel; virtual; abstract;
```

```
end;
```

TORedFactory and TOBlueFactory override the abstract interface to support different widget styles.

```
TORedFactory = class(TOAbstractFactory)
```

```
public
```

```
{ concrete widget constructors }
```

```
function CreateSpeedButton(AOwner: TComponent): TSpeedButton; override;
```

```
function CreateEdit(AOwner: TComponent): TEdit; override;
```

```
function CreateLabel(AOwner: TComponent): TLabel; override;
```

```
end;
```

```
TOBlueFactory = class(TOAbstractFactory)
```

```
public
```

```
{ concrete widget constructors }
```

```
function CreateSpeedButton(AOwner: TComponent): TSpeedButton; override;
```

```
function CreateEdit(AOwner: TComponent): TEdit; override;
```



```
function CreateLabel(AOwner: TComponent): TLabel; override;
```

```
end;
```

At runtime, our client application instantiates the abstract factory with a concrete class and then uses the abstract interface. Parts of the client application that use the factory don't need to know which concrete class is actually in use.

### **1.5.7 Pattern: Factory Method**

Definition : "Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

The Abstract Factory pattern can be viewed as a collection of Factory Methods.

#### **1.5.7.1 Applications in Delphi**

This pattern is useful when you want to encapsulate the construction of a class and isolate knowledge of the concrete class from the client application through an abstract interface.

One example of this might arise if you had an object oriented business application potentially interfacing to multiple target DBMS. The client application only wants to know about the business classes, not about their implementation-specific storage and retrieval.

#### **1.5.7.2 Implementation Example**

In the Abstract Factory example, each of the virtual widget constructor functions is a Factory Method. In their implementation we define a specific widget class to return.

```
TRedSpeedButton = class(TSpeedButton)
```

```
public
```

```

    constructor Create(AOwner: TComponent); override;

end;

constructor TRedSpeedButton.Create(AOwner: TComponent);

begin

    inherited Create(AOwner);

    Font.Color := clRed;

end;

function TRedFactory.CreateSpeedButton(AOwner: TComponent): TSpeedButton;

begin

    Result := TRedSpeedButton.Create(AOwner);

end;

```

## **1.6 APPENDIX : Key Elements of Delphi Class Definitions**

### **1.6.1 Unit Structure**

Delphi units (.PAS files) allow declaration of interface and implementation sections. The interface defines the part that is visible to other units using that unit. The keyword `uses` can be added to a unit's interface or implementation section to list the other units that your unit uses. This indicates to the compiler that your unit refers to parts of the used unit's interface. Parts of a unit declared in the implementation section are all private to that unit, i.e. never visible to any other unit. Types, functions and procedures

declared in the interface of a unit must have a corresponding implementation, or be declared as external (e.g. a call to a function in a DLL).

### **1.6.2 Class Interfaces**

Classes are defined as types in Delphi and may contain fields of standard data types or other objects, methods declared as functions or procedures, and properties. The type declaration of a class defines its interface and the scope of access to fields, methods and properties of the class. Class interfaces are usually defined in the interface of a unit to make them accessible to other modules using that unit. However they don't need to be. Sometimes a type declaration of a class may be used only within the implementation part of a unit.

### **1.6.3 Properties**

Properties are a specialised interface to a field of a defined type, allowing access control through read and write methods. Properties are not virtual, you can replace a property with another property of the same name, but the parent class doesn't know about the new property. It is however possible to make the access methods of a property virtual.

### **1.6.4 Inheritance**

Delphi's inheritance model is based on a single hierarchy. Every class inherits from TObject and can have only one parent.

A descendant class inherits all of the interface and functionality of its parent class, subject to the scope described below.

Multiple inheritance from more than one parent is not allowed directly. It can be implemented by using a container class to create instances one or more other classes and selectively expose parts of the contained classes.

Private, Protected, Public and Published Scope



Scope refers to the visibility of methods and data defined in the interface of a class, i.e. what parts of the class are accessible to the rest of the application or to descendant classes.

The default scope is public, for instance the component instances you add to a form at design time. Public says "come and get me"; it makes the data or method visible to everything at runtime.

Published parts of a class are a specialized form of Public scope. They indicate special behaviour for classes derived from `TPersistent`. A persistent class can save and restore its published properties to persistent storage using Delphi's standard streaming methods. Published properties also interact with Delphi Object Inspector in the IDE. A class must descend from `TPersistent` in order to use Published. There's also not much point in publishing methods, since you can't store them, although Delphi's compiler doesn't stop you. Published also lets another application access details of the class through Delphi's runtime type information. This would be rarely used, except in Delphi's design time interaction with its VCL.

Encapsulation or information hiding is essential to object orientation, so Protected and Private scope let you narrow the access to parts of a class.

Protected parts are visible only to descendant classes, or to other classes defined in the same unit.

Private parts are visible only to the defining class, or to other classes defined in the same unit.

It's important to note that once something is given public or published scope, it cannot be hidden in descendant classes.

#### Static, Virtual and Dynamic Methods; Override and Inherited

Methods declared as virtual or dynamic let you change their behaviour using override in a descendant class. You're unlikely to see a virtual method in the private part of a class, since it could only be overridden in the same unit, although Delphi's compiler doesn't stop you from doing this.

Override indicates that your new method replaces the method of the same name from the parent class. The override must be declared with the same name and parameters as the original method.

When a method is overridden, a call to the parent class's method actually executes the override method in the real class of the object.

Static methods on the other hand have no virtual or override declaration. You can replace a method of a class in a descendant class by redeclaring another method, however this is not object oriented. If you reference your descendant class as the parent type and try to call the replaced method, the static method of the parent class is executed. So in most cases, it's a bad idea to replace a static method.

Virtual and dynamic methods can be used interchangeably. They differ only in their treatment by the compiler and runtime library. Delphi's help explains that dynamic methods have their implementation resolved at compile time and run slightly faster, whereas virtual methods are resolved at runtime, resulting in slightly slower access but a smaller compiled program. Virtual is usually the preferred declaration. Delphi's help suggests using dynamic when you have a base class with many descendants that may not override the method.

The inherited directive lets you refer back to a property or method as it was declared in the parent class. This is most often used in the implementation of an override method, to call the inherited method of the parent class and then supplement its behaviour.

### **1.6.5 Abstract Methods**

Abstract is used in base classes to declare a method in the interface and defer its implementation to a descendant class. I.e. it defines an interface, but not the underlying operation. Abstract must be used with the virtual or dynamic directive. Abstract methods are never implemented in the base class and must be implemented in descendant classes to be used. A runtime error occurs if you try to execute an abstract method that is not overridden. Calling inherited within the override implementation of an abstract method will also result in a runtime error, since there is no inherited behaviour.



### **1.6.6 Messages**

Delphi's handling of Windows messages is a special case of virtual methods. Message handlers are implemented in classes that descend from TControl. I.e classes that have a handle and can receive messages. Message handlers are always virtual and can be declared in the private part of a class interface, yet still allow the inherited method to be called. Inherited in a message handler just uses the keyword inherited, there is no need to supply the name of the method to call.

### **1.6.7 Events**

Events are also an important characteristic of Delphi, since they let you delegate extensible behaviour to instances of a class. Events are properties that refer to a method of another object. Events are not inherited in Delphi 1; Delphi 2 extends this behaviour to let you use inherited in an event. . Inherited in an event handler just uses the keyword inherited, there is no need to supply the name of the method to call.

Events are particularly important to component developers, since they provide a hook for the user of the component to modify its behaviour in a way that may not be foreseen at the time the component is written.

### **1.6.8 Constructors and Destructors**

The constructor and destructor are two special types of methods. The constructor initializes a class instance (allocates memory initialized to 0) and returns a reference (pointer) to the object. The destructor deallocates memory used by the object (but not the memory of other objects created by the object).

Classes descended from TObject have a static constructor, Create, and a virtual destructor Destroy.

TComponent introduces a new public property, the Owner of the component and this must be initialized in the constructor. TComponent's constructor is declared virtual, i.e. it can be overridden in descendant classes.

It is essential when you override a virtual constructor or destructor in a TComponent descendant to include a call to the inherited method. `is one of the best programming tools to create software for Windows (and the .NET framework).`

## **1.7 Delphi Compilers**

With Delphi you can without much effort create small yet powerful Windows applications, which do not need to be installed, do not depend on Active X controls, or special dlls. This means that you will have far fewer users complaining about installation problems than if you created your software using Java, Visual Basic or Visual C++. Besides, many quality Delphi freeware, shareware and open source components can be found on the Web. See also below.

Delphi allows fast and high-level/abstract programming, like Java and Visual Basic, but you can also use it to code at a lower/more fundamental level, like you can do with most C++ environments. Delphi is based on the Pascal programming language making it ideal for educational purposes as well.

### **1.7.1 Free Delphi IDEs and Compilers**

Turbo Delphi: Free industrial strength Delphi RAD (Rapid Application Development) environment and compiler for Windows. It comes with 200+ components and its own Visual Component Framework. Note: this is the free edition of Borland's Delphi. The only limitation of this free edition is that you can not install additional components.

Turbo Delphi for .NET: Free industrial strength Delphi application development environment and compiler for the Microsoft .NET platform.

### **1.7.2 Free Delphi Compression Libraries**

Abbrevia: Freeware open-source compression toolkit for Borland Delphi, C++Builder, & Kylix. It supports PKZIP 4, Microsoft CAB, TAR, & gzip formats & the creation of

self-extracting archives. It includes visual components that simplify the manipulation of ZIP files.

DelphiZip: Freeware Delphi zip component including source code.

DIUcl: Delphi port of the fast and memory-efficient UCL C++-compression Library.

### **1.7.3 Free Delphi Script Engines**

Innerfuse Pascal Script: Freeware script engine written in Delphi. IFPS allows you to use most of Object Pascal language within your projects at runtime. It's a set of units that can be compiled into your exe file so there is no need to distribute any external files.

### **1.7.4 Free Database Components**

B-Tree Filer: Fast file-based database-system created with Borland Delphi, and which is not dependent on other tools, such as the Borland Database Engine or third-party dll's. Freeware, open-source.

SQL Parser for Delphi: A string parser that is capable to parse SQL statements into tokens, allows changing these tokens and rebuilding (modified) SQL statement.

Direct SQL: Cross-platform (Windows and Linux) Delphi/Kylix native components for directly accessing MySQL servers (without using any external dll's).

FlashFiler: Freeware, open-source client/server database created using Delphi. FlashFiler features a component-based architecture and can be compiled into your applications. Easy to configure, fast, and includes SQL.

kbmMemTable: Freeware. A full TDataset compatible in-memory table.



Open source dbExpress drivers: DbExpress driver for ODBC. Supports Delphi 7 & 6.02, BCB 6, and Kylix 2 & 3. Tested against many Databases including: Microsoft SqlServer, Oracle, IBM DB2, Centura SqlBase, MySql, Microsoft Access, and

Real Isam: A database library (DLL) that uses the ISAM method (Indexed Sequential Access Method) to manage access keys and variable length data records. For C++, Delphi, Visual Basic, etc. Freeware for Windows

tDBF: A native data access component for Delphi, BCB, Kylix, FreePascal. It allows you to create very compact database programs which don't need any special installer programs. The DB engine code is compiled right into your executable.

TJanSQL: TjanSQL is a single user SQL Database engine using plain text files with semi-colon separated data for data storage. In-memory handling of tables and recordsets; semi-compiled expressions. Freeware, open-source.

MidWare: Middleware the genral term for an application Layer put in the "middle" of a multi-tiered architecture software, allowing the various layers or "Tiers" to talk to each other. Middleware is the "glue" used to build efficient and scalable N-Tier Client/Server programs. The component set 'MidWare' consists of a set of 'Middleware' components and units which allow you to create an Application Server and related thin Client application, in just a few minutes. All without worrying about Client connections, data formatting and other details which make a multi-tiered Client/Server program so difficult to write. MidWare also includes CGI, ISAPI and ASP modules for web applications.

### **1.7.5 Delphi UI Components**

Human Interface Device controller suite: The HID controller (Human Interface Devices) is a component suite, which gives complete access to HID devices. Handles all devices which are in the HID subclass of USB

LS Speller: A non-visual Delphi component designed to add spell check capability to any application

Soft Gems Color Picker: Free advanced color picker component for Delphi and BCB.

SynEdit: An advanced multi-line cross-platform edit control, for Borland Delphi, Kylix and C++Builder. It supports Syntax Highlighting and code completion, it includes exporters for html, txt and rtf.

Virtual Treeview: Very powerful freeware open-source Delphi treeview component.

### **1.7.6 Printing and Reports**

Print Preview Suite: Excellent freeware Delphi print & preview component including source code.

FreeReport: Free report engine for Delphi and C++ Builder with report designer and previewer. Comparable to QuickReport3 and ReportBuilder 3.52. Freeware, full source code, royalty-free

Report Manager: Both a print scheme designer (report) and a high level printing (reporting) engine. Also a TCP Report Server and a Web Report Server (PDF on the fly), supports Windows and Linux. Connectivity to almost all databases is provided.

### **1.7.7 Free Delphi Unicode Libraries**

TntWare Delphi Unicode Controls: Delphi controls which allow you to develop applications that take advantage of the Unicode capabilities of Windows NT/2000/XP/2003.

SoftGems UniCodeEditor: UCE, the UniCodeEditor is an edit control for Delphi and Borland C++ Builder with syntax highlighting and WideString/Unicode support. This



edit control comes with syntax highlighter classes for Delphi, C/C++, HTML, SQL and DCG (the Delphi Compiler Generator).

Delphi Fundamentals Unicode Library: Delphi units which provide common functions necessary to utilize Unicode strings in Delphi applications.

SoftGems Unicode Library: A Delphi Unicode support library to use WideStrings/Unicode strings in your application. It includes more than 100 functions and classes for handling Unicode widestrings, as well as a unicode-enabled search engine and a unicode enabled regular expression search engine.

### **1.7.8 Delphi Component Directories**

Delphi Pages: Delphi components, tips, articles, forums, resumes, etc.

VCL Components: Components and libraries for Delphi, C++, Basic, Assembler, etc

Delphi Super Page: A large listing of components and sources.

Delphi32.com: Lots of Delphi-related information and downloads ordered by category.

DelphiSource: The latest news, libraries, components and utilities.

Torry's Delphi Page: Numerous components ordered by category, but a bit outdated here and there

DelphiABC: Delphi components, tools, applications, samples, tips and articles.

ComponentSource: Very commercial components site for a variety of compilers including Delphi and .NET.

### 1.7.9 Free Delphi Libraries, Components, Utilities

Delphi Fundamentals: Comprehensive collection of Delphi code units. Includes libraries for Unicode, Strings, Data Structures, Socket components and Mathematics

Delphi Hotkey: Create a system wide hotkey. The way this works is that your program will received an event whenever the specified hotkey is pressed regardless of which application has focus.

delphi2cpp: A free utility which converts Delphi/Pascal units into C++ code.

FastMM: Fast replacement memory manager for Borland Delphi Win32 applications that scales well in multi-threaded applications, is not prone to memory fragmentation, and supports shared memory without the use of external .DLL files.

GpHugeFile: Encapsulation of Windows file-handling routines that allows work with files larger than 2GB.

Jedi: A large collection of freeware/open-source components.

MYTHcode.org: A collection of object pascal libraries for parsing text strings and macro languages HTML, XHTML, XML, CSS and others.

PSV Delphi Components Library: A set of Open Source Delphi components containing: Windows Dialogs Library, php4Delphi, ISAPI and CGI support for Indy, RichEdit Syntax Highlighter .

Standard Interface Library: A framework library which aims to independize we all, the delphi developers, from the different glitches and jerks of the CLX/VCL, and attempts to provide code that is independent from the operating system. It is based on \*heavy\* use of interfaces.

TfisFileNotification: Creates a thread and uses it to monitor the contents of a directory or directory tree. Many different file changes can be monitored such as file size, last write, creation & deletion etc

TurboCASH Accounting: Entry level Delphi, Windows Accounting package for single users, small networks and distributed networks. Accomodates developer scripts, local plugins and multi language translation. Ideal for SME market.

TurboPower Essentials: 13 native VCL controls including drop-down calendars & calculators, roll-up dialogs, 3-D labels, tiled backgrounds, scrolling messages, menu buttons, and more. Freeware, open-source for Delphi and C++ Builder.

TurboPower Orpheus: Freeware, open-source UI toolkit for Borland Delphi and C++Builder. It contains over 120 components covering everything from data entry to calendars and clocks. Other noteworthy components include an Object Inspector, LookOut bar, and report views.

TurboPower ShellShock: A set of freeware, open-source components that let you customize applications with the functionality available in the Windows Shell and Windows Explorer, all without writing code. The components are written in native VCL for Borland Delphi and C++Builder

TurboPower SysTools: Freeware, open-source library of utility routines and classes for Borland Delphi, C++Builder, and other environments that support COM. It includes 1-D and 2-D bar codes, sorting, money routines, logging, high-precision math, a run-time math expression analyzer, etc.

Unofficial VisualCLX patches: Unofficial VisualCLX patches

### **1.7.10 Graphics Libraries**

Crispy Plotter: Plots mathematical functions at high speed.



Filters: An image processing library: sobel, convolution, morphology, vectorization, segmentation, blob, blur, histogram, susan, threshold, texture, contrast, standard deviation, canny, distance map, douglas-peuker, sklansky-gonzales, contour, edge, etc.

Genesis Device: Open Source 3D engine project to create a complete FPS game engine, including tools to create and view a virtual-scene.

Graph Package: Two components TGraph & TGraph3D which can help users to easily and quickly create 2D/3D graphics applications.

GraphicEx: An addendum to Delphi's Graphics.pas to enable your application to load many additional image formats.

Graphics32: A library designed for fast 32-bit graphics handling on Delphi and Kylix. Optimized for 32-bit pixel formats, it provides fast operations with pixels and graphic primitives, and in most cases Graphics32 outperforms the standard TCanvas classes. It is almost a hundred times faster in per-pixel access and about 2 - 5 times faster in drawing lines.

Real-Time Oscilloscope DLL Library: Freeware real-time Windows Oscilloscope DLL with an API for C++ , Delphi, MathWorks Matlab and Simulink.

TEffects: Free component for creating effects similar to what can be achieved by using commercial photo editing programs.

### **1.7.11 3D Programming**

3D Engines: Large list of 3D engines on various platforms and for various programming languages.

3D Studio Import Library: A collection of classes and structures to allow loading 3d Studio mli (material) and prj (project) files into your Delphi and Borland C++ Builder application.

Delphi OpenGL community: German OpenGL site for Delphi programmers.

Delphi3d.net: Site devoted to rapid OpenGL development.

GLScene: OpenGL package written in Delphi; freeware, open-source.

Open GL projects including source: [www.sulaco.co.za/opengl.htm](http://www.sulaco.co.za/opengl.htm).

OpenGL with Delphi: Borland community article on OpenGL. Get acquainted with OpenGL 3D graphics programming with this cookbook approach.

OpenGL.org: General home page of OpenGL.

OpenGL\_SG: An interface unit for using OpenGL with Delphi. It contains the translations of the gl.h, glu.h, glx.h and glext.h header files as well as a number additional support functions, and an interface with for most OpenGL extensions.

TOpenGL: This TOpenGL component for Borland C++ Builder allows you to build great two or three dimensional images inside your program. You can even animate these images.

Codehead's Bitmap Font Generator: A free bitmap font creation tool for OpenGL or DirectX applications.

### **1.7.12 Delphi Game Programming**

3dstate: 3D Engines and sets of tools for writing 3D games. Freeware for non-commercial use.

About.com Delphi game programming: Real-time 2D particle systems, VCL sprite engine, 3D engines list, basic game programming, DirectX.



Chessboard: Freeware chessboard component for Delphi and C++ Builder. It provides a 2-Dimensional and customizable chessboard with a drag and drop interface and event handlers (OnLegalMove, OnIllegalMove, OnCapture, OnCheck, OnMate, OnStaleMate, OnDraw etc.) A simple chess engine is included but of course you can use your own chess engine as well.

### **1.7.13 Delphi Programs with Source**

Apophysis: A windows application made in delphi for creating and editing fractal flames. Fractal flames are a extension on the ifs fractal.

Free Delphi Programs with Source: Free Delphi programs with source code for Internet, Database, HTML generation, RSR232 communication, terminal emulation, graphics, data conversion, help file making and more.

Extreme Performance Hospital IS: Freeware, open-source application for hospitals, containing a large number of modules which keep data of Patient Image, Symptoms, Physical Condition, Investigation, Diagnosis, Treatment including Procedure / Medication,etc. There are 50 hospitals in Thailand using this program.

Monex: Personal finance manager based on double entry bookkeeping principles. Supports download of financial data (exchange rates, stock quotes ...). Download of online available data is intended for a Slovenian audience, but can be customized.

MultiPro - FTP Client: Windows FTP program.

OpenSeeIT: An opensource image viewer program for Windows written in Delphi.

Phoenix Mail: An open source email client for Windows (and Linux?). Developed using Delphi.

VSpeech: Application enabling users to control their computer by their voice.

### 1.7.14 Internet and Communication Components

Internet Components: Native components implementing FTP, Mail, etc. This page also lists other components and various useful resources.

Indy Project: (1) Internet Direct (INDY) is an open source internet component suite comprised of popular internet protocols based on blocking sockets. (2) IndySoap is a Open Source Library for implementing Web services using Borland Pascal Compilers.

TurboPower Async Professional: A comprehensive communications toolkit for Borland Delphi, C++Builder, & ActiveX environments. It provides direct access to serial ports, TAPI, and the Microsoft Speech API. It supports faxing, terminal emulation, VOIP, & more. Freeware, open-source.

TurboPower Internet Professional: A set of freeware/open-source VCL components providing Internet connectivity. It includes POP3, SMTP, NNTP, FTP, HTTP, Instant Messaging, HTML viewer components, as well as components for low-level socket access.

Kylix WebProvider: Open Source CGI WebBroker replacement. It allows you to develop CGI applications in Apache environment using Kylix. It has a very small size compared to WebBroker and works fine with the Indy library and IndySOAP

### 1.7.15 Core Delphi

Programming without: Example of a Delphi 'Hello World' application without using the Borland application framework, but by calling the Windows API directly. The advantage is a fast and very small program (15 Kb) but it is more work.

### **1.7.16 Resource Editors**

XN Resource Editor: Freeware open-source resource editor and PE module explorer for Windows. It works with all resource files (.RES) and PE modules (.EXE, .DLL, etc.) and it has special knowledge of modules written in Delphi. It can display all the modules that comprise a Delphi program, and let you edit the properties of the components used on Delphi forms

Resource Hacker: A freeware resource editor for 32bit Windows executables and resource files (\*.res).

### **1.7.17 Delphi Tools**

FreeVCS: Freeware, open-source version control system having a Client/Server architecture and written in Delphi. It has syntax highlighting for Delphi, Pascal, C++, VB, Perl, Java, JavaScript, HTML, CSS, PHP, SQL, etc. and can be integrated with the Delphi IDE.

### **1.7.18 Delphi Magazines**

Dr. Bob's Delphi Clinic: Indepth discussion of Delphi-related issues by Dr. Bob: links, news, facts, tricks, articles, etc.

DelphiZine: Articles on Delphi and related matters.

### **1.7.19 Delphi Communities**

ADUG: ADUG is an organisation dedicated to providing a forum for activities and information that promote and improve the professional use of Delphi and related products and services in the Australian developer community.

Borland Newsgroups: The official Borland Delphi discussion forums. Discuss your questions, ideas and problems with other programmers.



BUG: UK Borland user group.

Delphi Newsgroups: Useful information regarding Delphi newsgroups.

Delphi User groups: A large list of local/country-based user groups.

Delphi-Talk: This mailing list is an open forum for discussing anything related to Borland Delphi.

### **1.7.20 Tutorials**

A Beginner's Guide to Delphi Database Programming: This free online course for database beginners and those who want a broad overview of the art of database programming with Delphi. Learn how to design, develop and test a database application using ADO.

Delphi Basics: Help and reference for the fundamentals of the Delphi language. It is an introduction to the Delphi Object Oriented Language for newcomers, and provides a ready reference for experienced programmers.

Delphi Land: Tutorials for beginners and intermediate level programmers, crash Course Delphi, projects with fully commented source code, Tips and hints, book reviews, Questions and Answers Forum, etc.

Free Online Programming Tutorials: Free online programming tutorials for beginning Delphi programmers.

### **1.7.21 Delphi Resources**

The Delphi-Box: Delphi portal; 1000+ links and tips for Developers.

HABit Delphi links: Very large collection of well-maintained links to Delphi sites, user-groups, downloads, tutorials, FAQs and tips.

DelphiSeek: Directory and search engine for Delphi components, resources, etc.

The Delphi compendium: Delphi-related books and links.

Delphi32.com: Components, downloads, articles and news.

EFG's Computer Lab: A page containing lots of very interesting projects. E.g. in the area of image processing, color, graphics, encryption, mathematics, fractals and chaos, science and engineering.

Project Jedi: The main goal of Project Jedi is to translate Windows API C++ library calls to native pascal units which can be used in Delphi. Another goal is to be a "portal" Website through which the whole community can share support, reusable code and components.

About Delphi programming: Useful site containing articles on Delphi (programming, backgrounds, news, etc.). An extensive list containing many freeware, shareware and commercial components can be found here.

## **1.8 CONCLUSION ABOUT DELPHI**

Delphi, as a tool, has reached a stage of maturity in that it is used fairly extensively in organizational settings in either the paper and pencil mode or in combination with face-to-face meetings and Nominal Group Techniques. Since most of these exercises are proprietary in nature there is not much of this activity reported in the open literature. The one exception to this is the applications in the medical field which are in fact actively reported and documented (Fink, Kosecoff, Chassin, and Brook, 1984). This clearly is a result of the growing need to formulate collaborative judgements about



complex issues that are associated with the production of guidelines on medical practice and decisions.

Computer Mediated Communications has also seen some very significant applications in the medical field with respect to the formulation of collaborative judgements. One of the most significant to be reported in the literature was the use of leading researchers in Viral Hepatitis to review the research literature and update guidelines for practitioners (Siegel, 1980). While this was not run in an anonymous mode, it had all the other aspects of structure necessary for a dozen experts to deal with some five thousand documents and reach complete consensus on the resulting guidelines.

Another CMC application that had Delphi like structuring with Anonymity was a Group Therapy process to aid individuals in the cessation of smoking (Schneider, 1986; Schneider and Tooley, 1986). A general review of CMC applications in the medical field can be found in Lerch (1988).

However, there is yet to be a true merger of Delphi with Computer Mediated Communications. It is only now that the technology is becoming generally available to support the high degree of tailoring necessary to dynamically structure communications within a single conferencing system (Turoff, 1991). Most conference systems, to date, have only represented single design structures with very little control available to facilitators and moderators of discussions. Also, the general lack of graphics has placed a considerable limitation on just what Delphi techniques could be adapted to the computer environment. The merger of Delphi and Computer Mediated Communications potentially offers far more than the sum of the two methods.

Long before the concept of Expert Systems it was known that statistical factor models (Dalkey, 1977) applied to a large sample of expert judgements could produce performance that was consistently in the upper quarter of the performance distribution curve. Such models did not suffer from "regression to the mean" and could result in matching the best decisions by the best experts in the group. Expert Systems is really the emergence of tools to allow this to be done on a fairly wide scale. However, the results of Expert System approaches, as currently practiced, are never going to do better than the best experts.

The merger of the Delphi Method, Computer Mediated Communications and the tools that we have discussed opens the possibility for performance of human groups that exceeds the composite performance curve. We have termed this phenomenon "collective intelligence" (Hiltz and Turoff, 1978). This is the ability of a group to produce a result that is of better quality than any single individual in the group could achieve acting alone. This rarely occurs in face-to-face groups.

A recent experiment in utilizing human judgement in conjunction with the types of models that are used in Expert Systems confirms that this is in fact possible (Blattberg and Hoch, 1990). There has been too much attention in recent years to utilizing computer technology to replace humans and far too little effort devoted to the potential for directly improving the performance of human groups. This can be achieved through integration of computer based methods and the concept of structured communications at the heart of the Delphi Method.



## CHAPTER 2

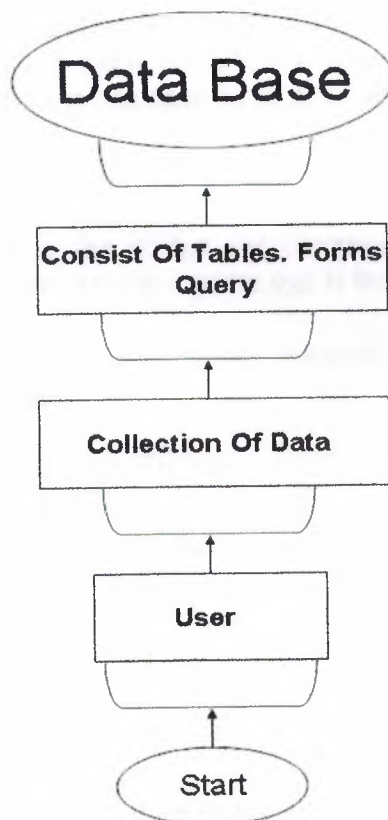
### DATABASE

#### 2.1 INTRODUCTION TO DATABASE

Every thing around us has a particular identity. To identify anything system, actor or person in words we need a data or information. So this information is valuable and in this advanced era we can store it in database and access this data by the blink of eye.

For an instant if we go through the definitions of database we may find following definitions.

- A **database** is a collection of related information.
- A **database** is an organized body of related information.
- Each database has **objects** such as a table, query, form, or macro



Pictorial definition of the Database



## **2.2 De-merits of Absence of Database**

A glance on the past will may help us to reveal the drawbacks in case of absence of database.

- In the past when there wasn't proper system of database, Much paper work was need to do and to handle great deal of written paper documentation was giant among the problems itself.
- In the huge networks to deal with equally bulky data, more workers are needed which affidavit cost much labor expanses.
- The old criteria for saving data and making identification was much time consuming such as if we want to search the particular data of a person.
- Before the Development of Computer database it was a great problem to search for some thing. Efforts to avoid the headache of search often results in new establishments of data.
- Before the development of database it seemed very unsafe to keep the worthy information. In Some situation some big organization had to employee the special persons in order to secure the data.
- Before the implementation of database any firm had to face the plenty of difficulties in order to maintain their Management. To hold the check on the expenses of the firm, the manager faced difficulties.

## **2.3 Merits of database**

The modern era is known as the golden age computer sciences and technology. In a simple phrase we can express that the modern age is built on the foundation of database.

If we carefully watch our daily life we can examine that some how our daily life is being connected with database.

- There are several benefits of database developments.
- Now with the help of computerized database we can access data in a second.
- By the development of the database we can make data more secure.
- By the development of database we can reduce the cost.

## **2.4 Introduction to Database Design**

The design of a database has to do with the way data is stored and how that data is related. The design process is performed after you determine exactly what information needs to be stored and how it is to be retrieved.

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMS ranging from small systems



that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

- Computerized library systems
- Automated teller machines
- Flight reservation systems
- Computerized parts inventory systems

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information. Requests for information from a database are made in the form of a query.

Database design is a complex subject. A properly designed database is a model of a business, Country Database or some other in the real world. Like their physical model counterparts, data models enable you to get answers about the facts that make up the objects being modeled. It's the questions that need answers that determine which facts need to be stored in the data model.

In the relational model, data is organized in tables that have the following characteristics: every record has the same number of facts, every field contains the same type of facts (Data) in each record, and there is only one entry for each fact. No two records are exactly the same.

The more carefully you design, the better the physical database meets users' needs. In the process of designing a complete system, you must consider user needs from a variety of viewpoints.

## **2.5 Database Models**

Various techniques are used to model data structures. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementation may be possible. An example of this is the relational model: in larger systems the physical implementation often has indexes which point to the data; this is similar to some aspects of common implementations of the network model. But in small relational database the data is often stored in a set of files, one per table, in a flat, un-indexed structure. There is some confusion below and elsewhere in this article as to logical data model vs. its physical implementation.

### **2.5.1 Flat Model**

The flat (or table) model consists of a single, two dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password might be used as a part of a system security database. Each row would have the specific password associated with a specific user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is the basis of the spreadsheet.

### **2.5.2 Network Model**

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.

### **2.5.3 Relational Model**

The relational data model was introduced in an academic paper by E.F. Cod in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allow tables to be defined that allow duplicate rows an



extension (or violation) of the relational model. In common English usage, a DBMS is called relational if it supports relational operational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, not-technical explanation of how “relational” database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the “flat” database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it’s not necessary to define all the keys in advance; a column can be used as a key even if it wasn’t originally intended to be one.

A key that can be used to uniquely identify a row in a table is called a unique key. Typically one of the unique keys is the preferred way to refer to row; this is defined as the table’s primary key.

When a key consists of data that has an external, real-world meaning (such as a person’s name, a book’s ISBN, or a car’s serial number), it’s called a “natural” key. If no nature key is suitable, an arbitrary key can be assigned (such as by given employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that can’t break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed).

### **2.5.3.1 Why we use a Relational Database Design**

Maintaining a simple, so-called flat database consisting of a single table doesn’t require much knowledge of database theory. On the other hand, most database worth maintaining are quite a bit more complicated than that. Real life databases often have

hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full-fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database.

## **2.6 Relationships between Tables**

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many and many-to-many relationships.

### **2.6.1 One-To-One Relationships**

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their trucks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support in a table requires security, placing them in a separate table lets your application restrict to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to these fields.

### **2.6.2 One-To-Many Relationships**



A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a **Many-To-Many** relationship as well.

## **2.7 Data Modeling**

In information system design, data modeling is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is “Data Analysis” the activity actually has more in common with the ideas and methods of synthesis (putting things together), than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships. In the early phases of a software development project, emphasis will be on the design of a conceptual data model. This can be detailed into a logical data model sometimes called functional data model. In later stages, this model may be translated into physical data model.

### **2.7.1 Database Normalization**

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMS lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case de-normalizations are sometimes used to improve performance, at the cost of reduced consistency.

### 2.7.2 Primary Key

In database design, a **primary key** is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions) and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a **primary key** is a candidate key chosen as the main method of uniquely identifying a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System Numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a relation. A **surrogate key** can be used as the **primary key**. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others. In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The **primary key** should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The **primary key** should be immutable, meaning its value should not be changed during the course of normal operations of the database. (Recall that a **primary key** is the means of uniquely identifying a tuple, and that identity by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the **primary key** is immutable, this can never happen.



### 2.7.3 Foreign Key

A **foreign key (FK)** is a field in a database record under one primary key that points to a key field of another database record in another table where the **foreign key** of one table refers to the **primary key** of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail needs not to include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book, or even the book itself. The ISBN is the **primary key** of the book, and it is used as a **foreign key** in the e-mail.

Note that using a **foreign key** often assumes its existence as a **primary key** somewhere else. Improper **foreign key/primary key** relationships are the source of many database problems.

### 2.7.4 Compound Key

In database design, a compound key (also called a composite key) is a key that consists on 2 or more attributes.

No restriction is applied to the attribute regarding their (initial) ownership within the data model. This means that any one, none or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute value

## CHAPTER 3

### USER'S MANUEL

When you run the program the program will meet with entry page.(Figure 3.1)



Figure 3.1

This page verify the user name and password after the entering the user name and password correctly, its going to main menu (Figure 3.2)





Figure 3.2

By the way on the first page there is five administrator level. If you are an manager, all of the menus are enable. If you are only system analyst, the password settings are disable

This page verify the add new product, delete product and update product. This part we will to form product Id, product name and supplier Id. Besides we will change setting product (Figure 3.3)

Add New Product	Delete A Product	Update A Product
* Product ID	<input type="text"/>	<input type="button" value="Search"/>
* Product Name	<input type="text"/>	
* Supplier ID	<input type="text"/>	
Quantity Per Unit	<input type="text"/>	
Unit Price	<input type="text"/>	
Reorder Level	<input type="text"/>	
* Category Name	<input type="text"/>	
Description	<input type="text"/>	
		<input type="button" value="Ignore"/> <input type="button" value="Add"/>
		<input type="button" value="Main Menu"/>




Figure 3.3

If a user want ,the user can change the password settings, entering information.  
(Figure 3.4)

Change Password Form	
Enter your Nick	<input type="text" value="kadir"/>
Enter your Password	<input type="text"/>
Verify your Password	<input type="text"/>
<input type="button" value="Ignore"/> <input type="button" value="Update"/>	
<input type="button" value="Main Menu"/>	

Figure 3.4

When you click the customer form is add new customer, delete customer and update a customer.This customer as below

Add New Customer	Delete A Customer	Update A Customer
Customer ID		 <b>Search</b>
Company Name		
Contact Name		
Contact Title		
Address		
City		
Phone1		
Phone2		
Phone3		
Fax No		
Customer Credit		 <b>Ignore</b>  <b>Add</b>


 **Main Menu**

Figure 3.5

This page is save to database the new customer information, call the this information from the database, delete this information and update this information and when you click the show button it shows the original document.(Figure 3.5)

When you click the order form is loaded and seems as below



The screenshot shows a software interface with two main sections: 'General Information' and 'Product(s) Information'. The 'General Information' section includes fields for 'Bill ID', 'Customer ID', 'Employee ID', 'Order Date' (set to 07.01.2007), and 'Required Date' (set to 07.01.2007). Each ID field has a '? Learn' button. The 'Product(s) Information' section includes fields for 'Product ID', 'Unit Price', and 'Quantity', each with a '? Learn' button. Below these fields is a green button labeled 'Add More Product'. At the bottom right, there are two buttons: a green 'OK' button and a blue 'Main Menu' button with a circular arrow icon.

General Information		Product(s) Information	
* Bill ID	<input type="text"/>	* Product ID	<input type="text"/>
* Customer ID	<input type="text"/>	* Unit Price	<input type="text"/>
* Employee ID	<input type="text"/>	* Quantity	<input type="text"/>
Order Date	07.01.2007	<a href="#">Add More Product</a>	
Required Date	07.01.2007		

Shipping Information	
Shipper ID	<input type="text"/>
Shipped Date	07.01.2007
Ship Name	<input type="text"/>
Ship Address	<input type="text"/>
Ship City	<input type="text"/>

[OK](#) [Main Menu](#)

Figure 3.6

This page is search to database the scanned files information, call the this information from the database. We will entering Bill Id and Customer Id and Employee Id after that we will seem product information and when you click the show button it shows the information.(Figure 3.6)

When you click the report on the main menu the report form is loaded and seems like below.





Figure 3.7

The report from is mainly includes ten parts which are coming report screen .if you want select to any report the that you can see solution report page.

This page is save to database the add a new supplier information, call the this information from the database, delete this information and update this information and when you click the show button it shows the original document. (Figure 3.8)





Add New Supplier	Delete A Supplier	Update A Supplier
* SupplierID	<input type="text"/>	 Search
* Company Name	<input type="text"/>	
Contact Name	<input type="text"/>	 Ignore  Add
Contact Title	<input type="text"/>	
Address	<input type="text"/>	
City	<input type="text"/>	
Phone1	<input type="text"/>	
Phone2	<input type="text"/>	
Phone3	<input type="text"/>	
Fax No	<input type="text"/>	
Home Page	<input type="text"/>	 Main Menu

Figure 3.8

When it is clicked you can delete, add and update supplier according to its Product.(Figure 3.8)

This page is save to database the add a new shipper information, call the this information from the database, delete this information and update this information and when you click the show button it shows the original document. (Figure 3.9)





Add New Shipper	Delete A Shipper	Update A Shipper	
* Shipper ID	<input type="text"/>	 <b>Search</b>	
* Company Name	<input type="text"/>		
Phone 1	<input type="text"/>		
Phone 2	<input type="text"/>		
Phone 3	<input type="text"/>		
Fax No	<input type="text"/>		
		 <b>Ignore</b>	 <b>Add</b>
			 <b>Main Menu</b>

Figure 3.9

This page is save to database the add a new bill information, call the this information from the database, delete this information and update this information and when you click the show button it shows the original document. (Figure 3.10)


Add New Bill	Delete A Bill	Update A Bill	
* Bill To	<input type="text"/>	 <b>Search</b>	
* Tax ID	<input type="text"/>		
Bill Amount	<input type="text"/>		
VAT	<input type="text"/>		
		 <b>Ignore</b>	 <b>Add</b>

Figure 3.10



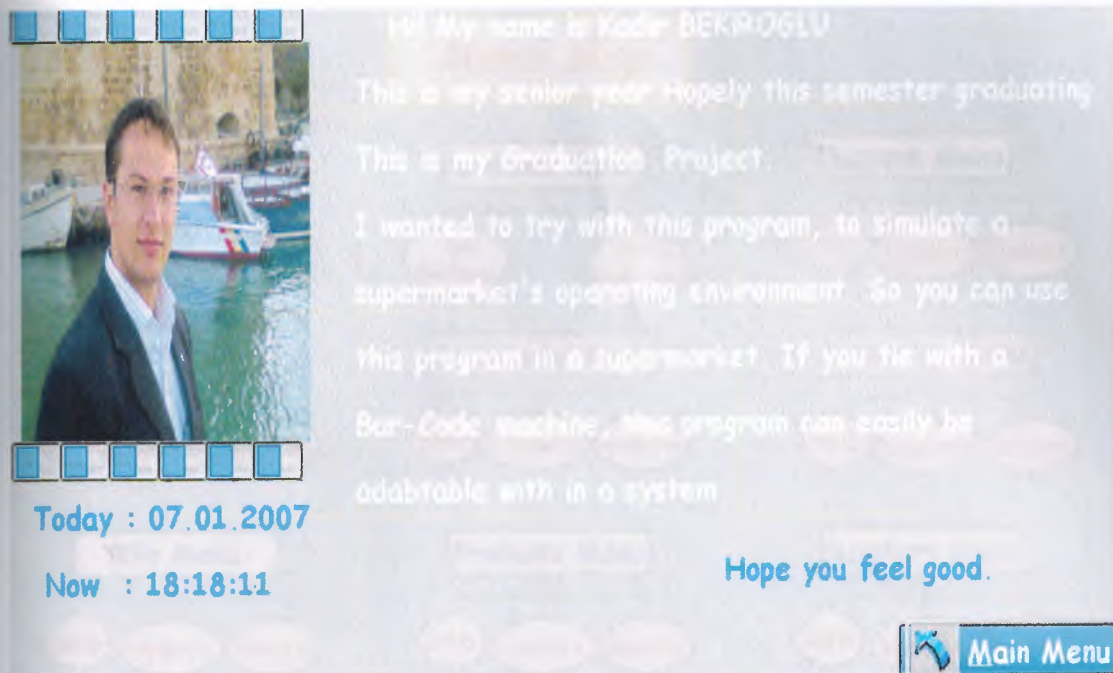


Figure 3.11

I introduce my self in "Who am I" form names.(Figure 3.11)

The other form is showed by menu map when you enter the form you will see below. In this form you can see all forms,it helps you to see about the program.To enter any menu click on it. (Figure 3.12)



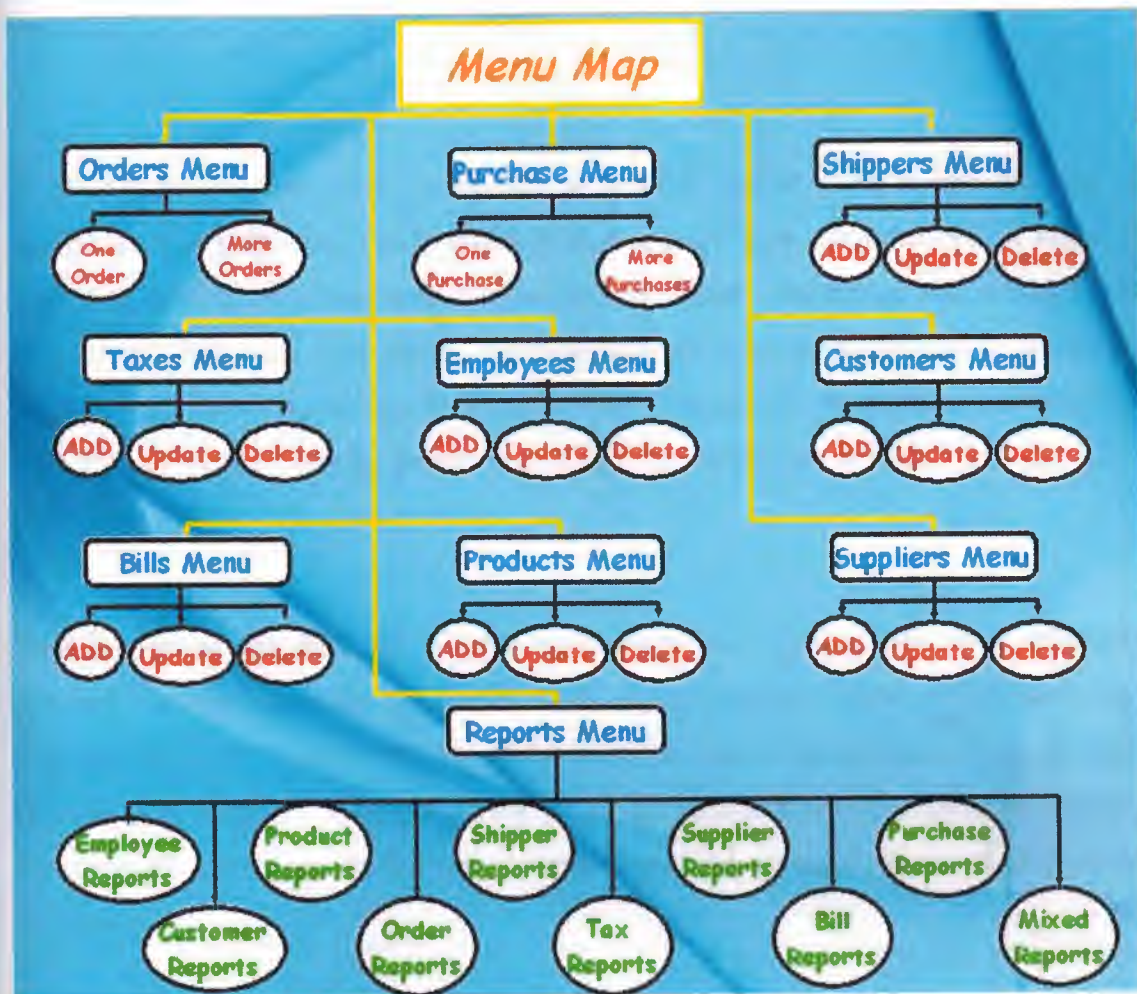


Figure 3.12

## CONCLUSION

A Delphi survey has been conducted to provide expert opinion on the life of components in buildings. Thirty different components were surveyed with a range of materials, coatings, environments and failure considered. The survey was conducted in two stages. After the first stage, approximately 80% of questions had a consistent answer from the survey group. In Stage 2, 10% of questions were further investigated, with 75% of these remaining questions then having a consistent answer.

Examination of the data for internal consistency and comparisons with externally available data indicates that the Delphi study appears reliable. However, the study was difficult to carry out owing to difficulties in obtaining answers from possible respondents. Thus, if a larger survey is to be undertaken to include all building components, it is recommended that committed respondents be obtained before devising the survey.

## APPENDIX

### PROGRAM CODE

#### FORM 1

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  ReorderLevel,w : integer;
begin
  query1.Close;
  query1.SQL.Clear;
  query1.SQL.Add('Select * From db\Employees Where Nick="'+edit1.text+'"');
  query1.ExecSQL;
  query1.Open;
  if (Datasource2.DataSet.RecordCount > 0) then
    begin
      if (Datasource2.DataSet.fieldValues ['Password'] = edit2.text) then
        begin
          {1} if Datasource2.DataSet.fieldValues ['Title'] ='Manager' then
            begin

              query10.Close;
              query10.SQL.Clear;
              query10.SQL.Add('Select ProductId,ReorderLevel, UnitsInStock-
UnitsOnOrder As Summation From db\Products Order By ProductId,ReorderLevel');
              query10.ExecSQL;
              query10.Open;

              if (Datasource10.DataSet.RecordCount > 0) then
                begin
                  while not (DataSource10.dataset.Eof) do
                    begin
                      ReorderLevel := Datasource10.DataSet.FieldValues['ReorderLevel'];
                      if ((Datasource10.DataSet.FieldValues['Summation'] = Null) OR
(Datasource10.DataSet.FieldValues['Summation'] < ReorderLevel)) then
                        begin
                          w:= datasource10.DataSet.FieldValues['ProductID'];

                          showmessage('Product ID ('+inttostr(w)+') is under Reorder Level
('+inttostr(ReorderLevel)+') .Please check product levels. ');
                          end;
                          DataSource10.dataset.Next;
                        end;
                      end;
                    form1.hide;
                    form20.BitBtn9.Visible:= True;
                    form2.Show;
                    Password;
```



```

        end
    {1} else
        {2} if (Datasource2.DataSet.fieldValues ['Title'] ='Sales Person') then
            begin
                form1.hide;
                form2.BitBtn3.Enabled := False;
                form2.BitBtn4.Enabled := False;
                form2.SpeedButton1.Enabled := False;
                form2.SpeedButton6.Enabled := False;
                form2.SpeedButton8.Enabled := False;
                form2.SpeedButton9.Enabled := False;
                form2.Show;
                Password;
            end
        {2} else
            {3} if (Datasource2.DataSet.fieldValues ['Title'] ='Accountant') then
                begin
                    form1.hide;
                    form2.Show;
                    Password;
                end
            {3} else
                {4} if (Datasource2.DataSet.fieldValues ['Title'] ='User') then
                    begin
                        form1.hide;
                        form2.SpeedButton5.Enabled := False;
                        form2.SpeedButton6.Enabled := False;
                        form2.SpeedButton8.Enabled := False;
                        form2.SpeedButton9.Enabled := False;
                        form2.BitBtn3.Enabled := False;
                        form2.BitBtn4.Enabled := False;
                        form2.BitBtn5.Enabled := False;
                        form2.Show;
                        Password;
                    end
                {4} else
                    begin
                        form1.hide;
                        form2.Show;
                        Password;
                    end;
            end
        else
            begin
                showmessage('You entered wrong password');
                edit2.SetFocus;
            end
        end
    else
        begin

```



```

    showmessage('You entered wrong User Name. Please enter correct one!');
    edit1.SetFocus;
end;

end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
    application.Terminate;
end;
procedure TForm1.Password;
var
    x,y : string;
begin
    x := edit2.Text;
    y := edit1.Text;
    Form14.Edit2.Text := x;
    Form12.Edit1.Text := y;
    //Form12.Edit2.Text := x;
    Form12.Edit4.Text := y;
    Form12.Edit5.Text := x;
end;
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    application.Terminate;
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
begin
    winexec('C:\windows\system32\osk.exe',9);
end;

end.

```

## FORM 2

```

procedure TForm2.SpeedButton1Click(Sender: TObject);
begin
    form2.Hide;
    form3.show;
end;

procedure TForm2.SpeedButton2Click(Sender: TObject);
begin
    form2.Hide;
    form4.show;
end;

procedure TForm2.SpeedButton7Click(Sender: TObject);
begin

```

```
form2.Hide;  
form5.show;  
end;
```

```
procedure TForm2.SpeedButton4Click(Sender: TObject);  
begin  
form2.Hide;  
form6.show;  
end;
```

```
procedure TForm2.SpeedButton3Click(Sender: TObject);  
begin  
  
form2.Hide;  
form7.show;  
end;
```

```
procedure TForm2.SpeedButton5Click(Sender: TObject);  
begin  
form2.Hide;  
form8.show;  
end;
```

```
procedure TForm2.SpeedButton6Click(Sender: TObject);  
begin  
form2.Hide;  
form9.show;  
end;
```

```
procedure TForm2.SpeedButton8Click(Sender: TObject);  
begin  
form2.Hide;  
form10.show;  
end;
```

```
procedure TForm2.SpeedButton9Click(Sender: TObject);  
begin  
form2.Hide;  
form11.show;  
end;
```

```
procedure TForm2.BitBtn2Click(Sender: TObject);  
begin  
Application.Terminate;  
end;
```

```
procedure TForm2.BitBtn3Click(Sender: TObject);  
begin  
form2.Hide;  
form14.show;
```

end;

```
procedure TForm2.BitBtn4Click(Sender: TObject);  
begin
```

```
form2.Hide;  
form13.show;  
end;
```

```
procedure TForm2.BitBtn5Click(Sender: TObject);  
begin  
Form2.Hide;  
Form15.Show;  
end;
```

```
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);  
begin  
Application.Terminate;  
end;
```

```
procedure TForm2.BitBtn1Click(Sender: TObject);  
begin  
Form2.Hide;  
Form26.Show;  
end;
```

end.

### **FORM 3**

```
procedure TForm3.OpenUpdate ;  
begin  
edit24.Enabled := True;  
edit25.Enabled := True;  
edit26.Enabled := True;  
edit27.Enabled := True;  
edit28.Enabled := True;  
edit29.Enabled := True;  
edit30.Enabled := True;  
edit31.Enabled := True;  
edit32.Enabled := True;  
edit33.Enabled := True;  
DateTimePicker5.Enabled := True;  
DateTimePicker6.Enabled := True;  
DbImage3.Enabled := True;  
bitbtn5.Enabled := True;  
bitbtn6.Enabled := True;  
bitbtn12.Enabled := True;  
  
end;
```

```

procedure TForm3.CloseUpdate;
begin
edit24.Enabled := False;
edit25.Enabled := False;
edit26.Enabled := False;
edit27.Enabled := False;
edit28.Enabled := False;
edit29.Enabled := False;
edit30.Enabled := False;
edit31.Enabled := False;
edit32.Enabled := False;
edit33.Enabled := False;
DateTimePicker5.Enabled := False;
DateTimePicker6.Enabled := False;
DbImage3.Enabled := False;
bitbtn5.Enabled := False;
bitbtn6.Enabled := False;
bitbtn12.Enabled := False;

```

```

end;

```

```

procedure TForm3.OpenAdd ;
begin
edit2.Enabled := True;
edit3.Enabled := True;
edit4.Enabled := True;
edit5.Enabled := True;
edit6.Enabled := True;
edit7.Enabled := True;
edit8.Enabled := True;
edit9.Enabled := True;
edit10.Enabled := True;
edit11.Enabled := True;
DateTimePicker1.Enabled := True;
DateTimePicker2.Enabled := True;
DbImage1.Enabled := True;
bitbtn3.Enabled := True;
bitbtn8.Enabled := True;
bitbtn11.Enabled := True;

```

```

end;

```

```

procedure TForm3.CloseAdd;
begin
edit2.Enabled := False;
edit3.Enabled := False;
edit4.Enabled := False;
edit5.Enabled := False;

```



```

edit6.Enabled := False;
edit7.Enabled := False;
edit8.Enabled := False;
edit9.Enabled := False;
edit10.Enabled := False;
edit11.Enabled := False;
DateTimePicker1.Enabled := False;
DateTimePicker2.Enabled := False;
DbImage1.Enabled := False;
bitbtn3.Enabled := False;
bitbtn8.Enabled := False;
bitbtn11.Enabled := False;

```

```

end;

```

```

procedure TForm3.OpenDelete;
begin
edit13.Enabled := True;
edit14.Enabled := True;
edit15.Enabled := True;
edit16.Enabled := True;
edit17.Enabled := True;
edit18.Enabled := True;
edit19.Enabled := True;
edit20.Enabled := True;
edit21.Enabled := True;
edit22.Enabled := True;
DateTimePicker3.Enabled := True;
DateTimePicker4.Enabled := True;
DbImage2.Enabled := True;
bitbtn4.Enabled := True;
bitbtn7.Enabled := True;
end;

```

```

procedure TForm3.CloseDelete;
begin
edit13.Enabled := False;
edit14.Enabled := False;
edit15.Enabled := False;
edit16.Enabled := False;
edit17.Enabled := False;
edit18.Enabled := False;
edit19.Enabled := False;
edit20.Enabled := False;
edit21.Enabled := False;
edit22.Enabled := False;
DateTimePicker3.Enabled := False;
DateTimePicker4.Enabled := False;
DbImage2.Enabled := False;
bitbtn4.Enabled := False;

```

```
    bitbtn7.Enabled := False;  
end;
```

```
procedure TForm3.ClearAdd;  
begin  
    edit1.Clear;  
    edit2.Clear;  
    edit3.Clear;  
    edit5.Clear;  
    edit6.Clear;  
    edit7.Clear;  
    edit8.Clear;  
    edit9.Clear;  
    edit10.Clear;  
    edit11.Clear;  
    DateTimePicker1.CleanupInstance;  
    DateTimePicker2.CleanupInstance;  
    DbImage1.CleanupInstance;
```

```
end;
```

```
procedure TForm3.ClearDelete;  
begin  
    edit12.Clear;  
    edit13.Clear;  
    edit14.Clear;  
    edit15.Clear;  
    edit16.Clear;  
    edit17.Clear;  
    edit18.Clear;  
    edit19.Clear;  
    edit20.Clear;  
    edit21.Clear;  
    edit22.Clear;  
    DateTimePicker3.CleanupInstance;  
    DateTimePicker4.CleanupInstance;  
    DbImage2.CleanupInstance;
```

```
end;
```

```
procedure TForm3.ClearUpdate;  
begin  
    edit23.Clear;  
    edit24.Clear;  
    edit25.Clear;  
    edit27.Clear;  
    edit28.Clear;  
    edit29.Clear;  
    edit30.Clear;  
    edit31.Clear;  
    edit32.Clear;
```

```

edit33.Clear;
DateTimePicker5.CleanupInstance;
DateTimePicker6.CleanupInstance;
DbImage3.CleanupInstance;
end;

procedure TForm3.BitBtn2Click(Sender: TObject);
begin
form3.Close;
form2.show;
end;

procedure TForm3.BitBtn3Click(Sender: TObject);
begin

if (edit2.Text = "") then
begin
showmessage('You have to enter First Name.');
```

edit2.SetFocus;

end

```

else if (edit3.Text = "") then
begin
showmessage('You have to enter Last Name.');
```

edit3.SetFocus;

end

```

else if (edit9.Text = "") then
begin
showmessage('You have to enter a Salary amount.');
```

edit9.SetFocus;

end

```

else if (edit10.Text = "") then
begin
showmessage('You have to enter User Name.');
```

edit10.SetFocus;

end

```

else if (edit11.Text = "") then
begin
showmessage('You have to enter a Password.');
```

edit11.SetFocus;

end

```

else if (edit1.Text <> "") then
begin
datasource1.Edit;
datasource1.DataSet.Append;
datasource1.DataSet.FieldValues['EmployeeID']:=edit1.Text;
```

if (edit2.Text <> "") then

datasource1.DataSet.FieldValues['FirstName']:=edit2.Text;

if (edit3.Text <> "") then

```

datasource1.DataSet.FieldValues['LastName']:=edit3.Text;

if (edit4.Text <> "") then
datasource1.DataSet.FieldValues['Title']:=edit4.Text;

datasource1.DataSet.FieldValues['BirthDate']:=
DateTimePicker1.Date;

datasource1.DataSet.FieldValues['HireDate']:=DateTimePicker2.Date;

if (edit5.Text <> "") then
datasource1.DataSet.FieldValues['Address']:=edit5.Text;

if (edit6.Text <> "") then
datasource1.DataSet.FieldValues['City']:=edit6.Text;

if (edit7.Text <> "") then
datasource1.DataSet.FieldValues['HomePhone']:=edit7.Text;

if (edit8.Text <> "") then
datasource1.DataSet.FieldValues['Extension']:=edit8.Text;

if (edit9.Text <> "") then
datasource1.DataSet.FieldValues['Salary']:=edit9.Text;

if (edit10.Text <> "") then
datasource1.DataSet.FieldValues['Nick']:=edit10.Text;

if (edit11.Text <> "") then
datasource1.DataSet.FieldValues['Password']:=edit11.Text;

//if (DbImage1.Picture <> Null) then
//datasource1.DataSet.FieldValues['Photo']:=DbImage1.Picture;

datasource1.DataSet.Post;

showmessage('Record is Added Successfully. ');
ClearAdd;
end // if
else
begin
showmessage('You have to enter Employee ID');
edit1.SetFocus;
end;

end;

procedure TForm3.BitBtn9Click(Sender: TObject);
begin
closeupdate;

```



```

openupdate;
if (edit23.Text <> "") then
begin

query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * From db\Employees Where EmployeeId='+edit23.Text+");
query1.ExecSQL;
query1.Open;

if (Datasource2.DataSet.RecordCount > 0) then
begin
openupdate;

if (Datasource2.DataSet.fieldValues ['FirstName'] <> Null) then
edit24.text := Datasource2.DataSet.fieldValues ['FirstName'];

if (Datasource2.DataSet.fieldValues ['LastName'] <> Null) then
edit25.text := Datasource2.DataSet.fieldValues ['LastName'];

if (Datasource2.DataSet.fieldValues ['BirthDate'] <> Null) then
DateTimePicker5.Date:= datasource2.DataSet.FieldValues['BirthDate'];

if (Datasource2.DataSet.fieldValues ['HireDate'] <> Null) then
DateTimePicker6.Date:= datasource2.DataSet.FieldValues['HireDate'];

if (Datasource2.DataSet.fieldValues ['Address'] <> Null) then
edit27.text := Datasource2.DataSet.fieldValues ['Address'];

if (Datasource2.DataSet.fieldValues ['City'] <> Null) then
edit28.text := Datasource2.DataSet.fieldValues ['City'];

if (Datasource2.DataSet.fieldValues ['HomePhone'] <> Null) then
edit29.text := Datasource2.DataSet.fieldValues ['HomePhone'];

if (Datasource2.DataSet.fieldValues ['Extension'] <> Null) then
edit30.text := Datasource2.DataSet.fieldValues ['Extension'];

if (Datasource2.DataSet.fieldValues ['Salary'] <> Null) then
edit31.text := Datasource2.DataSet.fieldValues ['Salary'];

if (Datasource2.DataSet.fieldValues ['Nick'] <> Null) then
edit32.text := Datasource2.DataSet.fieldValues ['Nick'];

if (Datasource2.DataSet.fieldValues ['Password'] <> Null) then
edit33.text := Datasource2.DataSet.fieldValues ['Password'];

// if (Datasource2.DataSet.fieldValues ['Photo'] <> Null) then
// DbImage3.Field := datasource2.DataSet.FieldValues['Photo'];

```

```

if (edit24.Text <> "") then
datasource1.DataSet.FieldValues['FirstName']:=edit24.Text;

if (edit25.Text <> "") then
datasource1.DataSet.FieldValues['LastName']:=edit25.Text;

if (edit26.Text <> "") then
datasource1.DataSet.FieldValues['Title']:=edit26.Text;

if (DateTimePicker5.Date <> Null) then
datasource1.DataSet.FieldValues['BirthDate']:=
DateTimePicker5.Date;

if (DateTimePicker6.Date <> Null) then
datasource1.DataSet.FieldValues['HireDate']:=DateTimePicker6.Date;

if (edit27.Text <> "") then
datasource1.DataSet.FieldValues['Address']:=edit27.Text;

if (edit28.Text <> "") then
datasource1.DataSet.FieldValues['City']:=edit28.Text;

if (edit29.Text <> "") then
datasource1.DataSet.FieldValues['HomePhone']:=edit29.Text;

if (edit30.Text <> "") then
datasource1.DataSet.FieldValues['Extension']:=edit30.Text;

if (edit31.Text <> "") then
datasource1.DataSet.FieldValues['Salary']:=edit31.Text;

if (edit32.Text <> "") then
datasource1.DataSet.FieldValues['Nick']:=edit32.Text;

if (edit33.Text <> "") then
datasource1.DataSet.FieldValues['Password']:=edit33.Text;

//if (DbImage3.Picture <> Null) then
//datasource1.DataSet.FieldValues['Photo']:=DbImage3.Picture;

datasource1.DataSet.Post;

showmessage('Record is Updated Successfully. ');
ClearUpdate;
CloseUpdate;
end
else
begin
showmessage('You have to enter Employee ID');

```

```

        edit23.SetFocus;
    end;

end;

procedure TForm3.PageControl1Change(Sender: TObject);
begin
    closeAdd;
    closedelete;
    closeupdate;
end;

procedure TForm3.Edit1Exit(Sender: TObject);
begin
    if (edit1.Text <> "") then
    begin
        query2.Close;
        query2.SQL.Clear;
        query2.SQL.Add('Select EmployeeID From db\Employees Where
EmployeeId='+edit1.Text+");
        query2.ExecSQL;
        query2.Open;

        if (Datasource3.DataSet.RecordCount > 0) then
        begin
            showmessage ('This EmployeeID added before, Please Enter another number. ');
            edit1.clear;
            edit1.SetFocus;
        end
        else
        begin
            OpenAdd;
        end;
    end
    else
    begin
        //showmessage('You have to enter EmployeeID. ');
        // edit1.Text := "";
    end;
end;

procedure TForm3.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9'])or (Key = #8)) then
    begin
        OpenAdd;
    end
    else
    begin

```



```

        showmessage('EmployeeID must be numeric value.');
```

Key := #0;

```

        edit1.SetFocus;
    end;
end;
```

```

procedure TForm3.BitBtn6Click(Sender: TObject);
var
up : word;
begin
    up := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
    if (up=MrYes) then
        ClearUpdate;
    end;
end;
```

```

procedure TForm3.BitBtn8Click(Sender: TObject);
var
ad : word;
begin
    ad := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
    if (ad=MrYes) then
        ClearAdd;
    end;
end;
```

```

procedure TForm3.BitBtn7Click(Sender: TObject);
var
de : word;
begin
    de := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
    if (de=MrYes) then
        ClearDelete;
    end;
end;
```

```

procedure TForm3.BitBtn10Click(Sender: TObject);
begin
    if (edit12.Text <> "") then
        begin
            query1.Close;
            query1.SQL.Clear;
            query1.SQL.Add('Select * From db\Employees Where EmployeeId='+edit12.Text+");
            query1.ExecSQL;
            query1.Open;

            if (Datasource2.DataSet.RecordCount > 0) then
                begin
                    opendelete;

                    if (Datasource2.DataSet.fieldValues ['FirstName'] <> Null) then
                        edit13.text := Datasource2.DataSet.fieldValues ['FirstName'];

```



```

if (Datasource2.DataSet.fieldValues ['LastName'] <> Null) then
edit14.text := Datasource2.DataSet.fieldValues ['LastName'];

if (Datasource2.DataSet.fieldValues ['Title'] <> Null) then
edit15.text := Datasource2.DataSet.fieldValues ['Title'];

if (Datasource2.DataSet.fieldValues ['BirthDate'] <> Null) then
DateTimePicker3.Date:= datasource2.DataSet.FieldValues['BirthDate'];

if (Datasource2.DataSet.fieldValues ['HireDate'] <> Null) then
DateTimePicker4.Date:= datasource2.DataSet.FieldValues['HireDate'];

if (Datasource2.DataSet.fieldValues ['Address'] <> Null) then
edit16.text := Datasource2.DataSet.fieldValues ['Address'];

if (Datasource2.DataSet.fieldValues ['City'] <> Null) then
edit17.text := Datasource2.DataSet.fieldValues ['City'];

if (Datasource2.DataSet.fieldValues ['HomePhone'] <> Null) then
edit18.text := Datasource2.DataSet.fieldValues ['HomePhone'];

if (Datasource2.DataSet.fieldValues ['Extension'] <> Null) then
edit19.text := Datasource2.DataSet.fieldValues ['Extension'];

if (Datasource2.DataSet.fieldValues ['Salary'] <> Null) then
edit20.text := Datasource2.DataSet.fieldValues ['Salary'];

if (Datasource2.DataSet.fieldValues ['Nick'] <> Null) then
edit21.text := Datasource2.DataSet.fieldValues ['Nick'];

if (Datasource2.DataSet.fieldValues ['Password'] <> Null) then
edit22.text := Datasource2.DataSet.fieldValues ['Password'];

//   if (Datasource2.DataSet.fieldValues ['Photo'] <> Null) then
//     DbImage2.Picture := datasource2.DataSet.FieldValues['Photo'];

    closedelete;
  end
else
  begin
    showmessage('There are no such record. ');
    edit12.SetFocus;
  end;
end
else
  begin
    showmessage('You have to enter EmployeeID');
    edit12.SetFocus;
  end;
end;

```

```

end;

procedure TForm3.BitBtn4Click(Sender: TObject);
begin
  query3.Close;
  query3.SQL.Clear;
  query3.SQL.Add('Delete From db\Employees Where EmployeeId=(Select
EmployeeID From db\Employees Where EmployeeId='+edit12.Text+')');
  query3.ExecSQL;
  query3.Open;
  showmessage('Record is Deleted Successfully. ');
  cleardelete;
  edit12.SetFocus;
end;

procedure TForm3.Edit23KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9']) or (Key = #8)) then
  begin
    OpenUpdate;
  end
  else
  begin
    showmessage('EmployeeID must be numeric value. ');
    Key := #0;
    edit23.SetFocus;
  end;
end;

procedure TForm3.Edit12KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9']) or (Key = #8)) then
  begin
    OpenDelete;
  end
  else
  begin
    showmessage('EmployeeID must be numeric value. ');
    Key := #0;
    edit12.SetFocus;
  end;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  closeAdd;
  closedelete;

```

```

closeupdate;
Datetimepicker1.Date :=(Date);
Datetimepicker2.Date :=(Date);
Datetimepicker3.Date :=(Date);
Datetimepicker4.Date :=(Date);
Datetimepicker5.Date :=(Date);
Datetimepicker6.Date :=(Date);
end;

procedure TForm3.BitBtn11Click(Sender: TObject);
begin
    if op1.Execute then
        Dbimage1.Picture.LoadFromFile(op1.FileName);
end;

procedure TForm3.Edit31KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9'])or (Key = #8)) then
        begin
            end
        else
            begin
                showmessage('Salary must be numeric value. ');
                Key := #0;
                edit31.SetFocus;
            end;
end;

procedure TForm3.Edit9KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9'])or (Key = #8)) then
        begin
            end
        else
            begin
                showmessage('Salary must be numeric value. ');
                Key := #0;
                edit9.SetFocus;
            end;
end;

procedure TForm3.BitBtn13Click(Sender: TObject);
begin
    Form16.BitBtn2.Click;
    Form16.BitBtn8.Visible:=True;
    Form16.BitBtn2.Enabled:=False;
    Form16.BitBtn3.Enabled:=False;
    Form16.BitBtn4.Enabled:=False;
    Form16.BitBtn5.Enabled:=False;
    Form16.BitBtn6.Enabled:=False;

```

```
Form16.BitBtn7.Enabled:=False;  
Form16.ShowModal;  
end;  
  
end.
```

#### **FORM 4**

```
procedure TForm4.OpenUpdate ;  
begin  
edit24.Enabled := True;  
edit25.Enabled := True;  
edit26.Enabled := True;  
edit27.Enabled := True;  
edit28.Enabled := True;  
edit29.Enabled := True;  
edit30.Enabled := True;  
edit31.Enabled := True;  
edit32.Enabled := True;  
edit33.Enabled := True;  
bitbtn5.Enabled := True;  
bitbtn6.Enabled := True;  
bitbtn9.Enabled := True;  
end;
```

```
procedure TForm4.CloseUpdate;  
begin  
edit24.Enabled := False;  
edit25.Enabled := False;  
edit26.Enabled := False;  
edit27.Enabled := False;  
edit28.Enabled := False;  
edit29.Enabled := False;  
edit30.Enabled := False;  
edit31.Enabled := False;  
edit32.Enabled := False;  
edit33.Enabled := False;  
bitbtn5.Enabled := False;  
bitbtn6.Enabled := False;  
bitbtn9.Enabled := False;  
end;
```

```
procedure TForm4.OpenAdd ;  
begin  
edit2.Enabled := True;  
edit3.Enabled := True;  
edit4.Enabled := True;  
edit5.Enabled := True;  
edit6.Enabled := True;  
edit7.Enabled := True;
```



```
edit8.Enabled := True;  
edit9.Enabled := True;  
edit10.Enabled := True;  
edit11.Enabled := True;  
bitbtn3.Enabled := True;  
bitbtn8.Enabled := True;  
end;
```

```
procedure TForm4.CloseAdd;  
begin  
edit2.Enabled := False;  
edit3.Enabled := False;  
edit4.Enabled := False;  
edit5.Enabled := False;  
edit6.Enabled := False;  
edit7.Enabled := False;  
edit8.Enabled := False;  
edit9.Enabled := False;  
edit10.Enabled := False;  
edit11.Enabled := False;  
bitbtn3.Enabled := False;  
bitbtn8.Enabled := False;  
end;
```

```
procedure TForm4.OpenDelete;  
begin  
edit13.Enabled := True;  
edit14.Enabled := True;  
edit15.Enabled := True;  
edit16.Enabled := True;  
edit17.Enabled := True;  
edit18.Enabled := True;  
edit19.Enabled := True;  
edit20.Enabled := True;  
edit21.Enabled := True;  
edit22.Enabled := True;  
bitbtn4.Enabled := True;  
bitbtn7.Enabled := True;  
bitbtn10.Enabled := True;  
end;
```

```
procedure TForm4.CloseDelete;  
begin  
edit13.Enabled := False;  
edit14.Enabled := False;  
edit15.Enabled := False;  
edit16.Enabled := False;  
edit17.Enabled := False;  
edit18.Enabled := False;
```

```
edit19.Enabled := False;  
edit20.Enabled := False;  
edit21.Enabled := False;  
edit22.Enabled := False;  
bitbtn4.Enabled := False;  
bitbtn7.Enabled := False;  
bitbtn10.Enabled := False;  
end;
```

```
procedure TForm4.ClearAdd;  
begin  
edit1.Clear;  
edit2.Clear;  
edit3.Clear;  
edit4.Clear;  
edit5.Clear;  
edit6.Clear;  
edit7.Clear;  
edit8.Clear;  
edit9.Clear;  
edit10.Clear;  
edit11.Clear;  
end;
```

```
procedure TForm4.ClearDelete;  
begin  
edit12.Clear;  
edit13.Clear;  
edit14.Clear;  
edit15.Clear;  
edit16.Clear;  
edit17.Clear;  
edit18.Clear;  
edit19.Clear;  
edit20.Clear;  
edit21.Clear;  
edit22.Clear;  
end;
```

```
procedure TForm4.ClearUpdate;  
begin  
edit23.Clear;  
edit24.Clear;  
edit25.Clear;  
edit26.Clear;  
edit27.Clear;  
edit28.Clear;  
edit29.Clear;  
edit30.Clear;  
edit31.Clear;
```

```
edit32.Clear;  
edit33.Clear;  
end;
```

```
procedure TForm4.BitBtn2Click(Sender: TObject);  
begin  
form4.Close;  
form2.show;  
end;
```

```
procedure TForm4.BitBtn3Click(Sender: TObject);  
begin
```

```
if (edit2.Text = "") then
```

```
begin
```

```
showmessage('You have to enter Company Name.');
```

```
edit2.SetFocus;
```

```
end
```

```
else if (edit1.Text <> "") then
```

```
begin
```

```
datasource1.Edit;
```

```
datasource1.DataSet.Append;
```

```
datasource1.DataSet.FieldValues['SupplierID']:=edit1.Text;
```

```
if (edit2.Text <> "") then
```

```
datasource1.DataSet.FieldValues['CompanyName']:=edit2.Text;
```

```
if (edit3.Text <> "") then
```

```
datasource1.DataSet.FieldValues['ContactName']:=edit3.Text;
```

```
if (edit4.Text <> "") then
```

```
datasource1.DataSet.FieldValues['ContactTitle']:=edit4.Text;
```

```
if (edit5.Text <> "") then
```

```
datasource1.DataSet.FieldValues['Address']:=edit5.Text;
```

```
if (edit6.Text <> "") then
```

```
datasource1.DataSet.FieldValues['City']:=edit6.Text;
```

```
if (edit7.Text <> "") then
```

```
datasource1.DataSet.FieldValues['Phone1']:=edit7.Text;
```

```
if (edit8.Text <> "") then
```

```
datasource1.DataSet.FieldValues['Phone2']:=edit8.Text;
```

```
if (edit9.Text <> "") then
```

```
datasource1.DataSet.FieldValues['Phone3']:=edit9.Text;
```

```
if (edit10.Text <> "") then
```

```
datasource1.DataSet.FieldValues['Fax']:=edit10.Text;
```

```

if (edit11.Text <> "") then
datasource1.DataSet.FieldValues['HomePage']:=edit11.Text;
datasource1.DataSet.Post;

showmessage('Record is Added Successfully. ');
ClearAdd;
end // if
else
begin
showmessage('You have to enter Supplier ID');
clearadd;
edit1.SetFocus;
end;
end;

procedure TForm4.BitBtn9Click(Sender: TObject);
begin
if (edit23.Text <> "") then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * From db\suppliers Where SupplierId='+edit23.Text+");
query1.ExecSQL;
query1.Open;

if (Datasource2.DataSet.RecordCount > 0) then
begin
openupdate;
if (Datasource2.DataSet.fieldValues ['CompanyName'] <> Null) then
edit24.text := Datasource2.DataSet.fieldValues ['CompanyName'];

if (Datasource2.DataSet.fieldValues ['ContactName'] <> Null) then
edit25.text := Datasource2.DataSet.fieldValues ['ContactName'];

if (Datasource2.DataSet.fieldValues ['ContactTitle'] <> Null) then
edit26.text := Datasource2.DataSet.fieldValues ['ContactTitle'];

if (Datasource2.DataSet.fieldValues ['Address'] <> Null) then
edit27.text := Datasource2.DataSet.fieldValues ['Address'];

if (Datasource2.DataSet.fieldValues ['City'] <> Null) then
edit28.text := Datasource2.DataSet.fieldValues ['City'];

if (Datasource2.DataSet.fieldValues ['Phone1'] <> Null) then
edit29.text := Datasource2.DataSet.fieldValues ['Phone1'];

if (Datasource2.DataSet.fieldValues ['Phone2'] <> Null) then
edit30.text := Datasource2.DataSet.fieldValues ['Phone2'];

```



```

if (Datasource2.DataSet.fieldValues ['Phone3'] <> Null) then
edit31.text := Datasource2.DataSet.fieldValues ['Phone3'];

if (Datasource2.DataSet.fieldValues ['Fax'] <> Null) then
edit32.text := Datasource2.DataSet.fieldValues ['Fax'];

if (Datasource2.DataSet.fieldValues ['HomePage'] <> Null) then
edit33.text := Datasource2.DataSet.fieldValues ['HomePage'];

```

```

end
else
begin
showmessage('There are no such record. ');
edit23.SetFocus;
end;

```

```

end
else
begin
showmessage('You have to enter SupplierID');
edit23.SetFocus;
end;

```

```

end;

```

```

procedure TForm4.BitBtn5Click(Sender: TObject);
begin

```

```

if (edit24.Text = "") then
begin
showmessage('You have to enter Company Name. ');
edit24.SetFocus;
end
else if (edit23.Text <> "") then
begin
datasource1.Edit;
if (edit24.Text <> "") then
datasource1.DataSet.FieldValues['CompanyName']:=edit24.Text;

if (edit25.Text <> "") then
datasource1.DataSet.FieldValues['ContactName']:=edit25.Text;

if (edit26.Text <> "") then
datasource1.DataSet.FieldValues['ContactTitle']:=edit26.Text;

if (edit27.Text <> "") then
datasource1.DataSet.FieldValues['Address']:=edit27.Text;

if (edit28.Text <> "") then
datasource1.DataSet.FieldValues['City']:=edit28.Text;

```

```

if (edit29.Text <> "") then
datasource1.DataSet.FieldValues['Phone1']:=edit29.Text;

if (edit30.Text <> "") then
datasource1.DataSet.FieldValues['Phone2']:=edit30.Text;

if (edit31.Text <> "") then
datasource1.DataSet.FieldValues['Phone3']:=edit31.Text;

if (edit32.Text <> "") then
datasource1.DataSet.FieldValues['Fax']:=edit32.Text;

if (edit33.Text <> "") then
datasource1.DataSet.FieldValues['HomePage']:=edit33.Text;

datasource1.DataSet.Post;

showmessage('Record is Updated Successfully.');
```

ClearUpdate;

CloseUpdate;

end

else

begin

showmessage('You have to enter supplier ID.');

edit23.SetFocus;

end;

end;

```

procedure TForm4.PageControl1Change(Sender: TObject);
begin
closeAdd;
closedelete;
closeupdate;
end;

procedure TForm4.Edit1Exit(Sender: TObject);
begin

if (edit1.Text <> "") then
begin
query2.Close;
query2.SQL.Clear;
query2.SQL.Add('Select SupplierID From db\suppliers Where
SupplierId='+edit1.Text+');
query2.ExecSQL;
query2.Open;

if (Datasource3.DataSet.RecordCount > 0) then
```

```

begin
  showmessage ('This SupplierID added before, Please Enter another number.');
```

-2004,02

```

  edit1.clear;
  edit1.SetFocus;
end
else
  begin
    OpenAdd;
  end;
end
else
  begin
    //showmessage('You have to enter SupplierID.');
```

-2007,01

```

    // edit1.Text := "";
  end;
end;

procedure TForm4.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['1'..'9'])or (Key = #8)) then
    begin
      OpenAdd;
    end
  else
    begin
      showmessage('SupplierID must be numeric value.');
```

-2007,01

```

      Key := #0;
      edit1.SetFocus;
    end;
end;

procedure TForm4.BitBtn6Click(Sender: TObject);
var
  up : word;
begin
  up := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbytes,mbno],0);
  if (up=MrYes) then
    ClearUpdate;
end;

procedure TForm4.BitBtn8Click(Sender: TObject);
var
  ad : word;
begin
  ad := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbytes,mbno],0);
  if (ad=MrYes) then
    ClearAdd;
end;
```



```

procedure TForm4.BitBtn7Click(Sender: TObject);
var
de : word;
begin
de := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbytes,mbno],0);
if (de=MrYes) then
ClearDelete;
end;

```

```

procedure TForm4.BitBtn10Click(Sender: TObject);
begin

```

```

if (edit12.Text <> "") then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * From db\suppliers Where SupplierId='+edit12.Text+");
query1.ExecSQL;
query1.Open;

```

```

if (Datasource2.DataSet.RecordCount > 0) then
begin

```

```

// opendeleate;
if (Datasource2.DataSet.fieldValues ['CompanyName'] <> Null) then
edit13.text := Datasource2.DataSet.fieldValues ['CompanyName'];

```

```

if (Datasource2.DataSet.fieldValues ['ContactName'] <> Null) then
edit14.text := Datasource2.DataSet.fieldValues ['ContactName'];

```

```

if (Datasource2.DataSet.fieldValues ['ContactTitle'] <> Null) then
edit15.text := Datasource2.DataSet.fieldValues ['ContactTitle'];

```

```

if (Datasource2.DataSet.fieldValues ['Address'] <> Null) then
edit16.text := Datasource2.DataSet.fieldValues ['Address'];

```

```

if (Datasource2.DataSet.fieldValues ['City'] <> Null) then
edit17.text := Datasource2.DataSet.fieldValues ['City'];

```

```

if (Datasource2.DataSet.fieldValues ['Phone1'] <> Null) then
edit18.text := Datasource2.DataSet.fieldValues ['Phone1'];

```

```

if (Datasource2.DataSet.fieldValues ['Phone2'] <> Null) then
edit19.text := Datasource2.DataSet.fieldValues ['Phone2'];

```

```

if (Datasource2.DataSet.fieldValues ['Phone3'] <> Null) then
edit20.text := Datasource2.DataSet.fieldValues ['Phone3'];

```

```

if (Datasource2.DataSet.fieldValues ['Fax'] <> Null) then
edit21.text := Datasource2.DataSet.fieldValues ['Fax'];

```



```

    if (Datasource2.DataSet.fieldValues ['HomePage'] <> Null) then
        edit22.text := Datasource2.DataSet.fieldValues ['HomePage'];
        closedelete;
        bitbtn4.Enabled := True;
        bitbtn7.Enabled := True;
    end
else
    begin
        showmessage('There are no such record. ');
        edit12.SetFocus;
    end;
end
else
begin
    showmessage('You have to enter SupplierID');
    edit12.SetFocus;
end;

end;

procedure TForm4.BitBtn4Click(Sender: TObject);
begin

if (edit12.Text <> "") then
begin
table1.Open;
table1.Edit;
while not (datasource1.dataset.Eof) do
begin
if (datasource1.DataSet.FieldByName('supplierid').AsString=edit12.Text) then
begin
table1.Delete;
end;
datasource1.dataset.Next;
end;

table1.Close;
table1.Open;
showmessage('Record is Deleted Successfully. ');
end
else
begin
showmessage('You have to enter Supplier ID. ');
edit12.SetFocus;
end;
cleardelete;
edit12.SetFocus;
end;

```

```

procedure TForm4.Edit23KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['1'..'9'])or (Key = #8)) then
        begin
            OpenUpdate;
        end
    else
        begin

            showmessage('SupplierID must be numeric value. ');
            Key := #0;
            closeupdate;
            edit23.SetFocus;
        end;
    end;
end;

```

```

procedure TForm4.Edit12KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['1'..'9'])or (Key = #8)) then
        begin
            OpenDelete;
        end
    else
        begin

            showmessage('SupplierID must be numeric value. ');
            Key := #0;
            edit12.SetFocus;
        end;
    end;
end;

```

```

procedure TForm4.FormCreate(Sender: TObject);
begin
    closeAdd;
    closedelete;
    closeupdate;
end;

```

```

procedure TForm4.BitBtn11Click(Sender: TObject);
begin
    Form19.BitBtn2.Click;
    Form19.BitBtn8.Visible:=True;
    Form19.BitBtn2.Enabled:=False;
    Form19.BitBtn3.Enabled:=False;
    Form19.BitBtn4.Enabled:=False;
    Form19.BitBtn5.Enabled:=False;
    Form19.BitBtn6.Enabled:=False;
    Form19.BitBtn7.Enabled:=False;
    Form19.ShowModal;
end;

```

end.

## FORM 5

```
procedure TForm5.OpenUpdate ;
begin
edit24.Enabled := True;
edit25.Enabled := True;
edit26.Enabled := True;
edit27.Enabled := True;
edit28.Enabled := True;
edit29.Enabled := True;
edit30.Enabled := True;
edit31.Enabled := True;
edit32.Enabled := True;
edit33.Enabled := True;
bitbtn5.Enabled := True;
bitbtn6.Enabled := True;
bitbtn9.Enabled := True;
end;
```

```
procedure TForm5.CloseUpdate;
begin
edit24.Enabled := False;
edit25.Enabled := False;
edit26.Enabled := False;
edit27.Enabled := False;
edit28.Enabled := False;
edit29.Enabled := False;
edit30.Enabled := False;
edit31.Enabled := False;
edit32.Enabled := False;
edit33.Enabled := False;
bitbtn5.Enabled := False;
bitbtn6.Enabled := False;
bitbtn9.Enabled := False;
end;
```

```
procedure TForm5.OpenAdd ;
begin
edit2.Enabled := True;
edit3.Enabled := True;
edit4.Enabled := True;
edit5.Enabled := True;
edit6.Enabled := True;
edit7.Enabled := True;
edit8.Enabled := True;
edit9.Enabled := True;
edit10.Enabled := True;
```

```
edit11.Enabled := True;  
bitbtn3.Enabled := True;  
bitbtn8.Enabled := True;  
end;
```

```
procedure TForm5.CloseAdd;  
begin  
edit2.Enabled := False;  
edit3.Enabled := False;  
edit4.Enabled := False;  
edit5.Enabled := False;  
edit6.Enabled := False;  
edit7.Enabled := False;  
edit8.Enabled := False;  
edit9.Enabled := False;  
edit10.Enabled := False;  
edit11.Enabled := False;  
bitbtn3.Enabled := False;  
bitbtn8.Enabled := False;  
end;
```

```
procedure TForm5.OpenDelete;  
begin  
edit13.Enabled := True;  
edit14.Enabled := True;  
edit15.Enabled := True;  
edit16.Enabled := True;  
edit17.Enabled := True;  
edit18.Enabled := True;  
edit19.Enabled := True;  
edit20.Enabled := True;  
edit21.Enabled := True;  
edit22.Enabled := True;  
bitbtn4.Enabled := True;  
bitbtn7.Enabled := True;  
bitbtn10.Enabled := True;  
end;
```

```
procedure TForm5.CloseDelete;  
begin  
edit13.Enabled := False;  
edit14.Enabled := False;  
edit15.Enabled := False;  
edit16.Enabled := False;  
edit17.Enabled := False;  
edit18.Enabled := False;  
edit19.Enabled := False;  
edit20.Enabled := False;  
edit21.Enabled := False;
```



```
edit22.Enabled := False;  
bitbtn4.Enabled := False;  
bitbtn7.Enabled := False;  
bitbtn10.Enabled := False;  
end;
```

```
procedure TForm5.ClearAdd;  
begin  
edit1.Clear;  
edit2.Clear;  
edit3.Clear;  
edit4.Clear;  
edit5.Clear;  
edit6.Clear;  
edit7.Clear;  
edit8.Clear;  
edit9.Clear;  
edit10.Clear;  
edit11.Clear;  
end;
```

```
procedure TForm5.ClearDelete;  
begin  
edit12.Clear;  
edit13.Clear;  
edit14.Clear;  
edit15.Clear;  
edit16.Clear;  
edit17.Clear;  
edit18.Clear;  
edit19.Clear;  
edit20.Clear;  
edit21.Clear;  
edit22.Clear;  
end;
```

```
procedure TForm5.ClearUpdate;  
begin  
edit23.Clear;  
edit24.Clear;  
edit25.Clear;  
edit26.Clear;  
edit27.Clear;  
edit28.Clear;  
edit29.Clear;  
edit30.Clear;  
edit31.Clear;  
edit32.Clear;  
edit33.Clear;  
end;
```

```

procedure TForm5.BitBtn2Click(Sender: TObject);
begin
form5.Close;
form2.show;
end;
procedure TForm5.BitBtn3Click(Sender: TObject);
begin

if (edit1.Text <> "") then
begin
datasource1.Edit;
datasource1.DataSet.Append;
datasource1.DataSet.FieldValues['CustomerID']:=edit1.Text;

if (edit2.Text <> "") then
datasource1.DataSet.FieldValues['CompanyName']:=edit2.Text;

if (edit3.Text <> "") then
datasource1.DataSet.FieldValues['ContactName']:=edit3.Text;

if (edit4.Text <> "") then
datasource1.DataSet.FieldValues['ContactTitle']:=edit4.Text;

if (edit5.Text <> "") then
datasource1.DataSet.FieldValues['Address']:=edit5.Text;

if (edit6.Text <> "") then
datasource1.DataSet.FieldValues['City']:=edit6.Text;

if (edit7.Text <> "") then
datasource1.DataSet.FieldValues['Phone1']:=edit7.Text;

if (edit8.Text <> "") then
datasource1.DataSet.FieldValues['Phone2']:=edit8.Text;

if (edit9.Text <> "") then
datasource1.DataSet.FieldValues['Phone3']:=edit9.Text;

if (edit10.Text <> "") then
datasource1.DataSet.FieldValues['Fax']:=edit10.Text;

if (edit11.Text <> "") then
datasource1.DataSet.FieldValues['CustomerCredit']:=edit11.Text;
datasource1.DataSet.Post;

showmessage('Record is Added Successfully.');
```

ClearAdd;

```

end // if
else

```

```

begin
    showmessage('You have to enter Customer ID');
    clearadd;
    edit1.SetFocus;
end;

end;

procedure TForm5.BitBtn9Click(Sender: TObject);
begin
    if (edit23.Text <> "") then
    begin
        query1.Close;
        query1.SQL.Clear;
        query1.SQL.Add('Select * From db\Customers Where CustomerId='+edit23.Text+");
        query1.ExecSQL;
        query1.Open;

        if (Datasource2.DataSet.RecordCount > 0) then
        begin
            openupdate;
            if (Datasource2.DataSet.fieldValues ['CompanyName'] <> Null) then
                edit24.text := Datasource2.DataSet.fieldValues ['CompanyName'];

            if (Datasource2.DataSet.fieldValues ['ContactName'] <> Null) then
                edit25.text := Datasource2.DataSet.fieldValues ['ContactName'];

            if (Datasource2.DataSet.fieldValues ['ContactTitle'] <> Null) then
                edit26.text := Datasource2.DataSet.fieldValues ['ContactTitle'];

            if (Datasource2.DataSet.fieldValues ['Address'] <> Null) then
                edit27.text := Datasource2.DataSet.fieldValues ['Address'];

            if (Datasource2.DataSet.fieldValues ['City'] <> Null) then
                edit28.text := Datasource2.DataSet.fieldValues ['City'];

            if (Datasource2.DataSet.fieldValues ['Phone1'] <> Null) then
                edit29.text := Datasource2.DataSet.fieldValues ['Phone1'];

            if (Datasource2.DataSet.fieldValues ['Phone2'] <> Null) then
                edit30.text := Datasource2.DataSet.fieldValues ['Phone2'];

            if (Datasource2.DataSet.fieldValues ['Phone3'] <> Null) then
                edit31.text := Datasource2.DataSet.fieldValues ['Phone3'];

            if (Datasource2.DataSet.fieldValues ['Fax'] <> Null) then
                edit32.text := Datasource2.DataSet.fieldValues ['Fax'];

            if (Datasource2.DataSet.fieldValues ['CustomerCredit'] <> Null) then
                edit33.text := Datasource2.DataSet.fieldValues ['CustomerCredit'];
        end;
    end;
end;

```

```

    end
else
    begin
        showmessage('There are no such record.');
```

Figure 10-27. The Key: Check

```

        edit23.SetFocus;
    end;
end
else
begin
    showmessage('You have to enter CustomerID');
    edit23.SetFocus;
end;

end;

procedure TForm5.BitBtn5Click(Sender: TObject);
begin
    datasource1.Edit;

    if (edit23.Text <> "") then
    begin

        if (edit24.Text <> "") then
            datasource1.DataSet.FieldValues['CompanyName']:=edit24.Text;

        if (edit25.Text <> "") then
            datasource1.DataSet.FieldValues['ContactName']:=edit25.Text;

        if (edit26.Text <> "") then
            datasource1.DataSet.FieldValues['ContactTitle']:=edit26.Text;

        if (edit27.Text <> "") then
            datasource1.DataSet.FieldValues['Address']:=edit27.Text;

        if (edit28.Text <> "") then
            datasource1.DataSet.FieldValues['City']:=edit28.Text;

        if (edit29.Text <> "") then
            datasource1.DataSet.FieldValues['Phone1']:=edit29.Text;

        if (edit30.Text <> "") then
            datasource1.DataSet.FieldValues['Phone2']:=edit30.Text;

        if (edit31.Text <> "") then
            datasource1.DataSet.FieldValues['Phone3']:=edit31.Text;

        if (edit32.Text <> "") then
            datasource1.DataSet.FieldValues['Fax']:=edit32.Text;

```



```

else
begin
  //showmessage('You have to enter CustomerID.');
```

```

  // edit1.Text := '';
end;
end;

procedure TForm5.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9'])or (Key = #8)) then
  begin
    OpenAdd;
  end
  else
  begin
    showmessage('CustomerID must be numeric value.');
```

```

    Key := #0;
    edit1.SetFocus;
  end;
end;

procedure TForm5.BitBtn6Click(Sender: TObject);
var
  up : word;
begin
  up := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
  if (up=MrYes) then
    ClearUpdate;
end;

procedure TForm5.BitBtn8Click(Sender: TObject);
var
  ad : word;
begin
  ad := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
  if (ad=MrYes) then
    ClearAdd;
end;

procedure TForm5.BitBtn7Click(Sender: TObject);
var
  de : word;
begin
  de := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
  if (de=MrYes) then
    ClearDelete;
end;

procedure TForm5.BitBtn10Click(Sender: TObject);
```

begin

if (edit12.Text <> "") then

begin

query1.Close;

query1.SQL.Clear;

query1.SQL.Add('Select \* From db\Customers Where CustomerId='+edit12.Text+');

query1.ExecSQL;

query1.Open;

if (Datasource2.DataSet.RecordCount > 0) then

begin

openupdate;

if (Datasource2.DataSet.fieldValues ['CompanyName'] <> Null) then

edit13.text := Datasource2.DataSet.fieldValues ['CompanyName'];

if (Datasource2.DataSet.fieldValues ['ContactName'] <> Null) then

edit14.text := Datasource2.DataSet.fieldValues ['ContactName'];

if (Datasource2.DataSet.fieldValues ['ContactTitle'] <> Null) then

edit15.text := Datasource2.DataSet.fieldValues ['ContactTitle'];

if (Datasource2.DataSet.fieldValues ['Address'] <> Null) then

edit16.text := Datasource2.DataSet.fieldValues ['Address'];

if (Datasource2.DataSet.fieldValues ['City'] <> Null) then

edit17.text := Datasource2.DataSet.fieldValues ['City'];

if (Datasource2.DataSet.fieldValues ['Phone1'] <> Null) then

edit18.text := Datasource2.DataSet.fieldValues ['Phone1'];

if (Datasource2.DataSet.fieldValues ['Phone2'] <> Null) then

edit19.text := Datasource2.DataSet.fieldValues ['Phone2'];

if (Datasource2.DataSet.fieldValues ['Phone3'] <> Null) then

edit20.text := Datasource2.DataSet.fieldValues ['Phone3'];

if (Datasource2.DataSet.fieldValues ['Fax'] <> Null) then

edit21.text := Datasource2.DataSet.fieldValues ['Fax'];

if (Datasource2.DataSet.fieldValues ['CustomerCredit'] <> Null) then

edit22.text := Datasource2.DataSet.fieldValues ['CustomerCredit'];

closedelete;

bitbtn4.Enabled := True;

bitbtn7.Enabled := True;

end

else

begin

showmessage('There are no such record.');

edit12.SetFocus;

```

        end;
    end
    else
    begin
        showmessage('You have to enter CustomerID');
        edit12.SetFocus;
    end;

end;

procedure TForm5.BitBtn4Click(Sender: TObject);
begin

if (edit12.Text <> '') then
begin
table1.Open;
table1.Edit;
while not (datasource1.dataset.Eof) do
begin
if (datasource1.DataSet.FieldName('Customerid').AsString=edit12.Text) then
begin
table1.Delete;
end;
datasource1.dataset.Next;
end;

table1.Close;
table1.Open;
showmessage('Record is Deleted Successfully. ');
end
else
begin
showmessage('You have to enter Customer ID. ');
edit12.SetFocus;
end;
cleardelete;
edit12.SetFocus;
end;

procedure TForm5.Edit23KeyPress(Sender: TObject; var Key: Char);
begin
if ((Key in ['0'..'9'])or (Key = #8)) then
begin
OpenUpdate;
end
else
begin

showmessage('CustomerID must be numeric value. ');

```

```

        Key := #0;
        closeupdate;
        edit23.SetFocus;
    end;
end;

procedure TForm5.Edit12KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9']) or (Key = #8)) then
    begin
        OpenDelete;
    end
    else
    begin
        showmessage('CustomerID must be numeric value. ');
        Key := #0;
        edit12.SetFocus;
    end;
end;

procedure TForm5.FormCreate(Sender: TObject);
begin
    closeAdd;
    closedelete;
    closeupdate;
end;

procedure TForm5.BitBtn11Click(Sender: TObject);
begin
    Form17.BitBtn2.Click;
    Form17.BitBtn8.Visible:=True;
    Form17.BitBtn2.Enabled:=False;
    Form17.BitBtn3.Enabled:=False;
    Form17.BitBtn4.Enabled:=False;
    Form17.BitBtn5.Enabled:=False;
    Form17.BitBtn6.Enabled:=False;
    Form17.BitBtn7.Enabled:=False;
    Form17.ShowModal;
end;

procedure TForm5.Edit11KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9']) or (Key = #8)) then
    begin
        OpenAdd;
    end
    else
    begin

```



```

        showmessage('Customer Credit must be numeric value.');
```

Key := #0;

```

        edit11.SetFocus;
    end;
end;

procedure TForm5.Edit33KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9'])or (Key = #8)) then
        begin
            OpenAdd;
        end
    else
        begin

            showmessage('Customer Credit must be numeric value.');
```

Key := #0;

```

            edit33.SetFocus;
        end;
    end;

end.
```

## FORM 9

```

procedure TForm9.ClearAll;
begin
    edit1.Clear;
    edit2.Clear;
    edit3.Clear;
    edit4.Clear;
    edit5.Clear;
    edit6.Clear;
    edit8.Clear;
    edit9.Clear;
    edit10.Clear;
    edit11.Clear;
    edit12.Clear;
    edit13.Clear;
    edit14.Clear;

end;

procedure TForm9.BitBtn2Click(Sender: TObject);
begin
    ClearAll;
    form9.Close;
    form2.show;
end;
```

```

procedure TForm9.Edit1Exit(Sender: TObject);
begin

    if (edit1.Text <> "") then
    begin
        query1.Close;
        query1.SQL.Clear;
        query1.SQL.Add('Select * From db\Products Where ProductId='+edit1.Text+");
        query1.ExecSQL;
        query1.Open;

        if (Datasource1.DataSet.RecordCount > 0) then
        begin
            GroupBox1.Visible := True;

            edit3.Text := Datasource1.DataSet.FieldValues['UnitPrice'];
            x:= strtofloat(edit3.Text);
            edit9.Text := Datasource1.DataSet.FieldValues['ProductName'];
            edit10.Text := Datasource1.DataSet.FieldValues['CategoryName'];
            // DbImage1.Picture := Datasource10.DataSet.FieldValues['Picture'];
            // Dbimage1.Picture.LoadFromFile(Datasource10.DataSet.FieldValues['Picture']);

            end
        else
        begin
            showmessage('This is not valid Product ID. Please enter correct one.');
```

edit1.clear;

```

            end;
        end;

    end;

procedure TForm9.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9'])or (Key = #8)) then
    begin
        end
    else
    begin
        showmessage('ProductID must be numeric value.');
```

Key := #0;

```

        edit1.SetFocus;
        end;
    end;

procedure TForm9.Edit2Exit(Sender: TObject);
begin

    if (edit2.Text <> "") then
    begin
```

```

query2.Close;
query2.SQL.Clear;
query2.SQL.Add('Select * From db\Suppliers Where SupplierID='+edit2.Text+");
query2.ExecSQL;
query2.Open;

if (Datasource20.DataSet.RecordCount > 0) then
begin
    GroupBox2.Visible := True;

    edit11.Text := Datasource20.DataSet.FieldValues['ContactName'];
    edit12.Text := Datasource20.DataSet.FieldValues['City'];
    edit13.Text := Datasource20.DataSet.FieldValues['Phone1'];
    edit14.Text := Datasource20.DataSet.FieldValues['HomePage'];
end
else
begin
    showmessage('This is not valid Supplier ID. Please enter correct one. ');
    edit2.clear;
end;
end
else
begin
end;
end;

procedure TForm9.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0'..'9']) or (Key = #8)) then
    begin
    end
    else
    begin
        showmessage('SupplierID must be numeric value. ');
        Key := #0;
        edit2.SetFocus;
    end;
end;

procedure TForm9.Edit5Exit(Sender: TObject);
begin

if (edit5.Text <> '') then
begin
    query3.Close;
    query3.SQL.Clear;
    query3.SQL.Add('Select ShipperID From db\Shippers Where ShipperId='+edit5.Text+");
    query3.ExecSQL;
    query3.Open;

```

```

if (Datasource30.DataSet.RecordCount > 0) then
  begin
    end
  else
    begin
    end;
  end
else
  begin
  end;
end;

```

```

procedure TForm9.Edit5KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9'])or (Key = #8)) then
    begin
    end
  else
    begin
      showmessage('Shipper ID must be numeric value. ');
      Key := #0;
      edit5.SetFocus;
    end;
  end;
end;

```

```

procedure TForm9.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0','1','2','3','4','5','6','7','8','9','.'])or (Key = #8)) then
    begin
    end
  else
    begin
      showmessage('Unit Price must be numeric value. ');
      Key := #0;
      edit3.SetFocus;
    end;
  end;
end;

```

```

procedure TForm9.Edit4KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9'])or (Key = #8)) then
    begin
    end
  else
    begin
      showmessage('Quantity must be numeric value. ');
      Key := #0;
      edit4.SetFocus;
    end;
  end;
end;

```



```

end;

procedure TForm9.Edit6KeyPress(Sender: TObject; var Key: Char);
begin
    if ((Key in ['0','1','2','3','4','5','6','7','8','9','.']) or (Key = #8)) then
        begin
            end
        else
            begin
                showmessage('Purchase Freight must be numeric value.');
```

Key := #0;

edit6.SetFocus;

end;

end;

```

procedure TForm9.FormCreate(Sender: TObject);
begin
    GroupBox1.Visible := False;
    GroupBox2.Visible := False;
    edit9.Enabled := False;
    edit10.Enabled := False;
    edit11.Enabled := False;
    edit12.Enabled := False;
    edit13.Enabled := False;
    edit14.Enabled := False;
    DbImage1.Enabled := False;
    Datetimepicker1.Date := (Date);
end;

procedure TForm9.BitBtn1Click(Sender: TObject);
var
    q,y,z : real;
begin
    if (edit2.Text = "") then
        begin
            showmessage('You have to enter Supplier ID.');
```

edit2.setfocus;

end

else if (edit3.Text = "") then

begin

showmessage('You have to enter Unit Price.');

edit3.setfocus;

end

else if (edit4.Text = "") then

begin

showmessage('You have to enter Quantity.');

edit4.setfocus;

end

else if (edit1.Text <> "") then

begin

```

table6.Open;
datasource6.Edit;
datasource6.DataSet.Append;

datasource6.DataSet.FieldValues['ProductID']:= strtoint(edit1.Text);

if (edit2.Text <> "") then
datasource6.DataSet.FieldValues['SupplierID']:=edit2.Text;

if (edit3.Text <> "") then
datasource6.DataSet.FieldValues['BuyingUnitPrice']:=edit3.Text;

if (edit4.Text <> "") then
datasource6.DataSet.FieldValues['Quantity']:=edit4.Text;

if (edit5.Text <> "") then
datasource6.DataSet.FieldValues['ShipperID']:=edit5.Text;

if (edit6.Text <> "") then
datasource6.DataSet.FieldValues['PurchaseFreight']:=edit6.Text;

if (edit8.Text <> "") then
datasource6.DataSet.FieldValues['EmployeeID']:=edit8.Text;

datasource6.DataSet.FieldValues['BuyingDate']:=DateTimePicker1.Date;

datasource6.DataSet.Post;
table6.Close;

qqqqq.Open;
qqqqq.Edit;

while not (DataSourceqqqqq.dataset.Eof) do
begin
if
(DataSourceqqqqq.DataSet.FieldByName('ProductId').AsString=edit1.Text) then
begin
y := strtoint(edit4.Text);

if (datasourceqqqqq.DataSet.FieldValues['UnitsInStock']<> Null) then
begin
q := datasourceqqqqq.DataSet.FieldValues['UnitsInStock'];
end
else
begin
q := 0;
end;
z := q+y;

```

```

        datasourceqqqqq.DataSet.Edit ;

        datasourceqqqqq.DataSet.FieldValues['UnitsInStock']:= z;
        end;
        DataSourceqqqqq.dataset.Next;
    end;
        showmessage('Purchase is Added Successfully. ');
        ClearAll;
    end
else
    begin
        showmessage('You have to enter Product ID. ');
        edit1.SetFocus;
    end;
end;

end;

procedure TForm9.Edit3Exit(Sender: TObject);
var
    y : real;
begin

    if (edit1.Text <> "") then
        begin

            if (DataSource1.DataSet.RecordCount > 0) then
                begin
                    y := strtofloat(edit3.Text);

                    if (y < x) then
                        begin
                            showmessage ('Unit Price must be greater than or equal to default Unit Price. ');
                            edit3.SetFocus;
                        end
                    else
                        begin
                            end;
                        end
                    end
                else
                    begin
                        end;
                    end
                begin
                    end;
                end
            else
                begin
                    end;
                end
            begin
                end;
            end;
        end;

procedure TForm9.BitBtn3Click(Sender: TObject);
begin
    Form20.BitBtn2.Click;
    Form20.BitBtn8.Visible:=True;

```

```
Form20.BitBtn2.Enabled:=False;  
Form20.BitBtn3.Enabled:=False;  
Form20.BitBtn4.Enabled:=False;  
Form20.BitBtn5.Enabled:=False;  
Form20.BitBtn6.Enabled:=False;  
Form20.BitBtn7.Enabled:=False;  
Form20.ShowModal;  
end;
```

```
procedure TForm9.BitBtn6Click(Sender: TObject);  
begin  
Form16.BitBtn2.Click;  
Form16.BitBtn8.Visible:=True;  
Form16.BitBtn2.Enabled:=False;  
Form16.BitBtn3.Enabled:=False;  
Form16.BitBtn4.Enabled:=False;  
Form16.BitBtn5.Enabled:=False;  
Form16.BitBtn6.Enabled:=False;  
Form16.BitBtn7.Enabled:=False;  
Form16.ShowModal;  
end;
```

```
procedure TForm9.BitBtn5Click(Sender: TObject);  
begin  
Form18.BitBtn2.Click;  
Form18.BitBtn8.Visible:=True;  
Form18.BitBtn2.Enabled:=False;  
Form18.BitBtn3.Enabled:=False;  
Form18.BitBtn4.Enabled:=False;  
Form18.BitBtn5.Enabled:=False;  
Form18.BitBtn6.Enabled:=False;  
Form18.BitBtn7.Enabled:=False;  
Form18.ShowModal;  
end;
```

```
procedure TForm9.BitBtn4Click(Sender: TObject);  
begin  
Form19.BitBtn2.Click;  
Form19.BitBtn8.Visible:=True;  
Form19.BitBtn2.Enabled:=False;  
Form19.BitBtn3.Enabled:=False;  
Form19.BitBtn4.Enabled:=False;  
Form19.BitBtn5.Enabled:=False;  
Form19.BitBtn6.Enabled:=False;  
Form19.BitBtn7.Enabled:=False;  
Form19.ShowModal;  
end;
```

```
procedure TForm9.Edit8KeyPress(Sender: TObject; var Key: Char);  
begin
```



```

if ((Key in ['0'..'9']) or (Key = #8)) then
begin
end
else
begin
showmessage('SupplierID must be numeric value. ');
Key := #0;
edit8.SetFocus;
end;
end;

end.

```

## FORM 12

```

procedure TForm12.BitBtn2Click(Sender: TObject);
begin
form12.Close;
form2.show;
end;

```

```

procedure TForm12.BitBtn3Click(Sender: TObject);
var
up : word;
begin
up := messagedlg('Are you really want to ignore ?',mtconfirmation, [mbyes,mbno],0);
if (up=MrYes) then
edit1.Clear;
edit2.Clear;
edit3.Clear;
end;

```

```

procedure TForm12.BitBtn4Click(Sender: TObject);
begin
if (edit1.Text = "") then
begin
showmessage('You have to enter a User Name');
edit1.SetFocus;
end
else if (edit2.Text = "") then
begin
showmessage('You have to enter a Password');
edit2.SetFocus;
end
else if (edit2.Text <> edit3.Text) then
begin
showmessage('Enter your password regularly. ');
edit2.Clear;
edit3.Clear;

```

```

        edit2.SetFocus;
    end
else if (edit2.Text = edit3.Text) then
    begin
        if ((edit1.Text <> "") AND (edit2.Text <> "")) then
            begin

                query1.Close;
                query1.SQL.Clear;
                query1.SQL.Add('Select * From db\Employees Where Nick
= '"+edit4.Text+"'");
                query1.ExecSQL;
                query1.Open;

                if (Datasource2.DataSet.RecordCount > 0) then
                    edit6.Text := Datasource2.DataSet.FieldValues ['EmployeeID']
                else
                    showmessage ('Yanlis');

                query2.Close;
                query2.SQL.Clear;
                query2.SQL.Add('Select * From db\Employees Where EmployeeID
= '+edit6.Text+'");
                query2.ExecSQL;
                query2.Open;

                datasource1.Edit;

                if (edit1.Text <> "") then
                    datasource1.DataSet.FieldValues['Nick']:=edit1.Text;

                if (edit2.Text <> "") then
                    datasource1.DataSet.FieldValues['Password']:=edit2.Text;

                datasource1.DataSet.Post;

                showmessage('Information(s) changed Successfully. ');
                edit1.Clear;
                edit2.Clear;
                edit3.Clear;
                Form12.Close;
                Form2.Show;
            end
        else
            begin
                showmessage('You have to enter your User Name and Password. ');
                edit1.SetFocus;
            end;
        end
    end
else

```

```

begin
    showmessage('Enter your password regularly. ');
    edit2.Clear;
    edit3.Clear;
    edit2.SetFocus;
end;

end;

end.

```

## FORM 15

```

procedure TForm15.SpeedButton2Click(Sender: TObject);
begin
    Form15.Hide;
    Form16.BitBtn8.Visible:=False;
    Form16.BitBtn2.Enabled:=True;
    Form16.BitBtn3.Enabled:=True;
    Form16.BitBtn4.Enabled:=True;
    Form16.BitBtn5.Enabled:=True;
    Form16.BitBtn6.Enabled:=True;
    Form16.BitBtn7.Enabled:=True;
    Form16.Show;
end;

procedure TForm15.SpeedButton3Click(Sender: TObject);
begin
    Form15.Hide;
    Form17.BitBtn8.Visible:=False;
    Form17.BitBtn2.Enabled:=True;
    Form17.BitBtn3.Enabled:=True;
    Form17.BitBtn4.Enabled:=True;
    Form17.BitBtn5.Enabled:=True;
    Form17.BitBtn6.Enabled:=True;
    Form17.BitBtn7.Enabled:=True;
    Form17.Show;
end;

procedure TForm15.SpeedButton4Click(Sender: TObject);
begin
    Form15.Hide;
    Form18.BitBtn8.Visible:=False;
    Form18.BitBtn2.Enabled:=True;
    Form18.BitBtn3.Enabled:=True;
    Form18.BitBtn4.Enabled:=True;
    Form18.BitBtn5.Enabled:=True;
    Form18.BitBtn6.Enabled:=True;
    Form18.BitBtn7.Enabled:=True;
    Form18.Show;
end;

```

```

procedure TForm15.SpeedButton5Click(Sender: TObject);
begin
Form15.Hide;
Form19.BitBtn8.Visible:=False;
Form19.BitBtn2.Enabled:=True;
Form19.BitBtn3.Enabled:=True;
Form19.BitBtn4.Enabled:=True;
Form19.BitBtn5.Enabled:=True;
Form19.BitBtn6.Enabled:=True;
Form19.BitBtn7.Enabled:=True;
Form19.Show;
end;

```

```

procedure TForm15.SpeedButton6Click(Sender: TObject);
begin
Form15.Hide;
Form20.BitBtn8.Visible:=False;
Form20.BitBtn2.Enabled:=True;
Form20.BitBtn3.Enabled:=True;
Form20.BitBtn4.Enabled:=True;
Form20.BitBtn5.Enabled:=True;
Form20.BitBtn6.Enabled:=True;
Form20.BitBtn7.Enabled:=True;
Form20.Show;

end;

```

```

procedure TForm15.SpeedButton7Click(Sender: TObject);
begin
Form15.Hide;
Form21.BitBtn8.Visible:=False;
Form21.BitBtn2.Enabled:=True;
Form21.BitBtn3.Enabled:=True;
Form21.BitBtn4.Enabled:=True;
Form21.BitBtn5.Enabled:=True;
Form21.BitBtn6.Enabled:=True;
Form21.BitBtn7.Enabled:=True;
Form21.Show;
end;

```

```

procedure TForm15.SpeedButton8Click(Sender: TObject);
begin
Form15.Hide;
Form22.BitBtn8.Visible:=False;
Form22.BitBtn2.Enabled:=True;
Form22.BitBtn3.Enabled:=True;
Form22.BitBtn4.Enabled:=True;
Form22.BitBtn5.Enabled:=True;
Form22.BitBtn6.Enabled:=True;

```



```

Form22.Show;
end;

procedure TForm15.SpeedButton9Click(Sender: TObject);
begin
Form15.Hide;
Form23.Show;
end;

procedure TForm15.SpeedButton10Click(Sender: TObject);
begin
Form15.Hide;
Form24.Show;
end;

procedure TForm15.SpeedButton11Click(Sender: TObject);
begin
Form15.Hide;
Form25.Show;
end;

procedure TForm15.BitBtn1Click(Sender: TObject);
begin
Form15.Close;
Form2.Show;
end;

end.

```

## FORM 16

```

procedure TForm16.BitBtn1Click(Sender: TObject);
begin
Form16.Close;
Form15.Show;
end;

procedure TForm16.BitBtn2Click(Sender: TObject);
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select EmployeeID,FirstName,LastName,Title From db\Employees
');
query1.ExecSQL;
query1.Open;
end;

procedure TForm16.BitBtn3Click(Sender: TObject);
begin

```

```

query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select FirstName,LastName,Title,BirthDate From db\Employees
Order by BirthDate DESC');
query1.ExecSQL;
query1.Open;
end;

```

```

procedure TForm16.BitBtn4Click(Sender: TObject);
begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select FirstName,LastName,Nick From db\Employees');
    query1.ExecSQL;
    query1.Open;
end;

```

```

procedure TForm16.BitBtn5Click(Sender: TObject);
begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select City,count(EmployeeID)As Counter From db\Employees
Group By City');
    query1.ExecSQL;
    query1.Open;
end;

```

```

procedure TForm16.BitBtn6Click(Sender: TObject);
begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select E.FirstName,E.LastName,count(P.EmployeeID) As Counter
From db\Employees E,db\Purchase P Where E.EmployeeID=P.EmployeeID Group By
E.FirstName,E.LastName');
    query1.ExecSQL;
    query1.Open;
end;

```

```

procedure TForm16.BitBtn7Click(Sender: TObject);
begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select FirstName,LastName,Salary From db\Employees Order By
Salary DESC');
    query1.ExecSQL;
    query1.Open;
end;

```

```

procedure TForm16.BitBtn8Click(Sender: TObject);
begin

```

```
Form16.Close;  
end;
```

```
end.
```

## **FORM 26**

```
procedure TForm26.Label1Click(Sender: TObject);  
begin  
Form26.Hide;  
Form8.Show;  
end;
```

```
procedure TForm26.Label20Click(Sender: TObject);  
begin  
Form26.Hide;  
Form2.Show;  
end;
```

```
procedure TForm26.Label21Click(Sender: TObject);  
begin  
Form26.Hide;  
Form9.Show;  
end;
```

```
procedure TForm26.Label27Click(Sender: TObject);  
begin  
Form26.Hide;  
Form6.Show;  
end;
```

```
procedure TForm26.Label31Click(Sender: TObject);  
begin  
Form26.Hide;  
Form6.TabSheet2.Enabled := True;  
Form6.Show;  
end;
```

```
procedure TForm26.Label29Click(Sender: TObject);  
begin  
Form26.Hide;  
Form6.TabSheet3.Enabled := True;  
Form6.Show;  
end;
```

```
procedure TForm26.Label46Click(Sender: TObject);  
begin  
Form26.Hide;  
Form6.TabSheet1.Enabled := True;
```

```

Form6.Show;
end;

procedure TForm26.Label26Click(Sender: TObject);
begin
Form26.Hide;
Form5.Show;
end;

procedure TForm26.Label45Click(Sender: TObject);
begin
Form26.Hide;
Form5.TabSheet1.Enabled := True;
Form5.Show;
end;

procedure TForm26.Label28Click(Sender: TObject);
begin
Form26.Hide;
Form5.TabSheet3.Enabled := True;
Form5.Show;
end;

procedure TForm26.Label34Click(Sender: TObject);
begin
Form26.Hide;
Form5.TabSheet2.Enabled := True;
Form5.Show;
end;

procedure TForm26.Label25Click(Sender: TObject);
begin
Form26.Hide;
Form4.Show;
end;

procedure TForm26.Label44Click(Sender: TObject);
begin
Form26.Hide;
Form4.TabSheet1.Enabled := True;
Form4.Show;
end;

procedure TForm26.Label18Click(Sender: TObject);
begin
Form26.Hide;
Form4.TabSheet3.Enabled := True;
Form4.Show;
end;

```



```
procedure TForm26.Label35Click(Sender: TObject);
begin
Form26.Hide;
Form4.TabSheet2.Enabled := True;
Form4.Show;
end;
```

```
procedure TForm26.Label6Click(Sender: TObject);
begin
Form26.Hide;
Form3.Show;
end;
```

```
procedure TForm26.Label43Click(Sender: TObject);
begin
Form26.Hide;
Form3.TabSheet1.Enabled := True;
Form3.Show;
end;
```

```
procedure TForm26.Label17Click(Sender: TObject);
begin
Form26.Hide;
Form3.TabSheet3.Enabled := True;
Form3.Show;
end;
```

```
procedure TForm26.Label32Click(Sender: TObject);
begin
Form26.Hide;
Form3.TabSheet2.Enabled := True;
Form3.Show;
end;
```

```
procedure TForm26.Label5Click(Sender: TObject);
begin
Form26.Hide;
Form11.Show;
end;
```

```
procedure TForm26.Label40Click(Sender: TObject);
begin
Form26.Hide;
Form11.TabSheet1.Enabled := True;
Form11.Show;
end;
```

```
procedure TForm26.Label15Click(Sender: TObject);
begin
Form26.Hide;
```

```
Form11.TabSheet3.Enabled := True;  
Form11.Show;  
end;
```

```
procedure TForm26.Label30Click(Sender: TObject);  
begin  
Form26.Hide;  
Form11.TabSheet2.Enabled := True;  
Form11.Show;  
end;
```

```
procedure TForm26.Label22Click(Sender: TObject);  
begin  
Form26.Hide;  
Form10.Show;  
end;
```

```
procedure TForm26.Label41Click(Sender: TObject);  
begin  
Form26.Hide;  
Form10.TabSheet1.Enabled := True;  
Form10.Show;  
end;
```

```
procedure TForm26.Label13Click(Sender: TObject);  
begin  
Form26.Hide;  
Form10.TabSheet3.Enabled := True;  
Form10.Show;  
end;
```

```
procedure TForm26.Label14Click(Sender: TObject);  
begin  
Form26.Hide;  
Form10.TabSheet2.Enabled := True;  
Form10.Show;  
end;
```

```
procedure TForm26.Label23Click(Sender: TObject);  
begin  
Form26.Hide;  
Form7.Show;  
end;
```

```
procedure TForm26.Label42Click(Sender: TObject);  
begin  
Form26.Hide;  
Form7.TabSheet1.Enabled := True;  
Form7.Show;  
end;
```

```

procedure TForm26.Label16Click(Sender: TObject);
begin
Form26.Hide;
Form7.TabSheet3.Enabled := True;
Form7.Show;
end;

```

```

procedure TForm26.Label33Click(Sender: TObject);
begin
Form26.Hide;
Form7.TabSheet2.Enabled := True;
Form7.Show;
end;

```

```

procedure TForm26.Label24Click(Sender: TObject);
begin
Form26.Hide;
Form15.Show;
end;

```

```

procedure TForm26.Label19Click(Sender: TObject);
begin
Form26.Hide;
Form16.BitBtn8.Visible:=False;
Form16.BitBtn2.Enabled:=True;
Form16.BitBtn3.Enabled:=True;
Form16.BitBtn4.Enabled:=True;
Form16.BitBtn5.Enabled:=True;
Form16.BitBtn6.Enabled:=True;
Form16.BitBtn7.Enabled:=True;
Form16.Show;
end;

```

```

procedure TForm26.Label2Click(Sender: TObject);
begin
Form26.Hide;
Form17.BitBtn8.Visible:=False;
Form17.BitBtn2.Enabled:=True;
Form17.BitBtn3.Enabled:=True;
Form17.BitBtn4.Enabled:=True;
Form17.BitBtn5.Enabled:=True;
Form17.BitBtn6.Enabled:=True;
Form17.BitBtn7.Enabled:=True;
Form17.Show;
end;

```

```

procedure TForm26.Label3Click(Sender: TObject);
begin
Form26.Hide;

```

```
Form20.BitBtn8.Visible:=False;  
Form20.BitBtn2.Enabled:=True;  
Form20.BitBtn3.Enabled:=True;  
Form20.BitBtn4.Enabled:=True;  
Form20.BitBtn5.Enabled:=True;  
Form20.BitBtn6.Enabled:=True;  
Form20.BitBtn7.Enabled:=True;  
Form20.Show;  
end;
```

```
procedure TForm26.Label4Click(Sender: TObject);  
begin  
Form26.Hide;  
Form23.Show;  
end;
```

```
procedure TForm26.Label7Click(Sender: TObject);  
begin  
Form26.Hide;  
Form18.BitBtn8.Visible:=False;  
Form18.BitBtn2.Enabled:=True;  
Form18.BitBtn3.Enabled:=True;  
Form18.BitBtn4.Enabled:=True;  
Form18.BitBtn5.Enabled:=True;  
Form18.BitBtn6.Enabled:=True;  
Form18.BitBtn7.Enabled:=True;  
Form18.Show;  
end;
```

```
procedure TForm26.Label9Click(Sender: TObject);  
begin  
Form26.Hide;  
Form22.BitBtn8.Visible:=False;  
Form22.BitBtn2.Enabled:=True;  
Form22.BitBtn3.Enabled:=True;  
Form22.BitBtn4.Enabled:=True;  
Form22.BitBtn5.Enabled:=True;  
Form22.BitBtn6.Enabled:=True;  
Form22.Show;  
end;
```

```
procedure TForm26.Label8Click(Sender: TObject);  
begin  
Form26.Hide;  
Form19.BitBtn8.Visible:=False;  
Form19.BitBtn2.Enabled:=True;  
Form19.BitBtn3.Enabled:=True;  
Form19.BitBtn4.Enabled:=True;  
Form19.BitBtn5.Enabled:=True;  
Form19.BitBtn6.Enabled:=True;
```



```
Form19.BitBtn7.Enabled:=True;  
Form19.Show;  
end;
```

```
procedure TForm26.Label10Click(Sender: TObject);  
begin  
Form26.Hide;  
Form21.BitBtn8.Visible:=False;  
Form21.BitBtn2.Enabled:=True;  
Form21.BitBtn3.Enabled:=True;  
Form21.BitBtn4.Enabled:=True;  
Form21.BitBtn5.Enabled:=True;  
Form21.BitBtn6.Enabled:=True;  
Form21.BitBtn7.Enabled:=True;  
Form21.Show;  
end;
```

```
procedure TForm26.Label12Click(Sender: TObject);  
begin  
Form26.Hide;  
Form24.Show;  
end;
```

```
procedure TForm26.Label11Click(Sender: TObject);  
begin  
Form26.Hide;  
Form25.Show;  
end;
```

```
end.
```