NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

THE GOODS TRADİNG AUTOMATİON SYSTEM USİNG DELPHİ PROGRAMMİNG LANGUAGE

Graduation Project

COM- 400

Student: Fazıl KARGIN (20011115)

Supervisor: Assist. Prof. Dr. Firudin Muradov

Nicosia – 2006

ACKNOWLEDGMENT

First of all, I would like to express my thanks to my supervisor Mr. Firudin MURADOV for supervising my project. Under the guidance of him I successfully overcome many difficulties. He welcomed me whenever i wanted to discuss something with him without feeling shame or hesitation. Also I thank other instructors in Computer Engeenering department for their help and guideness.

special thanks to my family, especially my parents for being patientfull during my undergraduate degree study. I could never have completed my study without their encouragement and endless support.

Finally, I want to thank all my friends and specially Muhammed S. ABDULLAH , metin ULAŞ, who supported and helped me all the time.

ABSTRACT

Delphi is an object oriented , visual programming environment for rapid application development. With Delphi you can write Windows programs more quickly and more quickly and more easily than was ever possible before. Delphi Programming Language and Paradox7 Data base system in building this program. Delphi is one of the modern programming languages, it contains thousands of commands, components, tools…etc. it is easy to use too. The program stores every sale operation done by a user; it also stores income and outcome payments, lists the activities and duties of all the personals working in Market.

This project has as its goal to develop software, processing information about activities of Dress market. Many forms used in building this project, the most important form is Stock Detail form, using this form the program records the entrance of new items to the stock, also while selling something it automatically decrease the stock amount of that item Software developed in this project contains both employee information, and information associated with sales and purchase . The project can be developed by improving the software for processing all activities of the company.

TABLE OF CONTENTS

ACKNOWLEDGEMENT I
ABSTRACT II
TABLE OF CONTENTS III
INTRODUCTION 1
CHAPTER 1. .BASIC CONCEPT OF DELPHI 7 **2**
1.1 Introduction 2
1.2 What is Delphi? 3
1.2.1 Developer Support Services and Web Site 4
1.3 A Tour of The Environment 4
1.3.1 Starting Delphi 5
1.3.2 Delphi(IDE) 5
1.3.3 The object inspector 7
1.3.4 The Delphi Workspace 8
1.3.5 The Menus and Toolbars 8
1.3.6 The Component Palette and Form Designer 10
1.3.7 The Object Tree View 11
1.3.8 The Object Repository 12
1.3.9 The Code Editor 13
1.3.9.1 Code Insight 14
1.3.10 Class Completion 15
1.3.11 Code Browsing 16
1.3.12 The Diagram Page 17
1.3.13 Viewing Form Code 17
1.3.14 The Code Explorer 18
1.3.15 The Project Manager 19
1.3.16 The Project Browser 19
1.4 Programming With Delphi 20
1.4.1 Creating a Project 20
1.4.2 Adding Data Modules 20
1.4.3 Building the user interface 21
1.4.4 Placing components on a form 21
1.4.5 Setting the properties of the component 22
1.4.6 Writing Code 24
1.4.6.1 Using The Component Library 27
1.4.7 Compiling and Debugging Projects 28
1.4.7.1 Deploying Applications 30
1.4.7.2 Internationalizing Applications 31
1.4.8 Types of Projects 30
1.4.8.1Delphi (CLX Applications) 31
1.4.8.2 Delphi (Database Applications) 31
1.4.9 Administrator(BDE) 32

1.4.10 Database Explorer 32
1.4.11 Database Desktop 32
1.4.12 Data Dictionary 32
1.4.13 Components of custom 33
1.4.14 Dynamic-link libraries 33
1.4.15Delphi(COM and ActiveX) 33
1.4.16 Component Type Libraries 34
1.5 Work Area (İDE) 34
1.5.1 Arranging Menus and Toolbars 34
1.5.2 Tool Windows 35
1.5.3 Desktop Layouts 37
1.6The Component Palette 38
1.6.1 Creating Component Templates 39
CHAPTER 2. .DATABASE CONCEPT OF DELPHI 7 40
2.1 About Dbase And Paradox 40
2.1.1 Architecture of database 40
2.1.2 Relational database consept 40
2.1.3 Accessing data in other database 41
2.1.4 dBASE IV Table Specification 41
2.1.5 Dbase V Table Specification 42
2.1.6 dBASE Field Types 42
2.2 Paradox Standard Table Specifications 44
2.2.1 Paradox4 table structure 44
2.2.2 Paradox 5 Table Specifications 45
2.2.3 Paradox 7 and Above Table Specifications 46
2.2.4 Paradox Field Types 47
CHAPTER 3. .MAİN FORMS OF THE APLİCATİON PROGRAM 50
3.1 Database Design of The Program 50
3.2 Relationships between tables 51
3.3 Execution of the programs 51
CONCLUSION 68
REFERENCES 69
APPENDIX 1: Program Codes 70
APPENDIX 2: Database Tables 112

INTRODUCTION

The Goods Trader Automation System is an important program for all markets, all the operations done through this system. Delphi programming Language is used in this project. The project consist of three chapters

Chapter 1 describes Delphi programming basic consept, such as menus and toolbar,component palette, form designer, code editor, code explorer, project maneger and project browser.it is also present information about programing with Delphi and work area . same time this chapter explain Delphi's command and properties are available in menus,it has many usable and wonderful ready commands. In the same time it has some disadvantages, while writing Codes the program does not alert you when you write wrong codes and does not show you the exact fault.

Chapter 2 Describe about database and paradox . it's table arrangement's are easier than other database programs. Also it's Delphi's own program, while using delphi. Making changes in the tables are more easier than other database programs. Specialists prefer using this database than Microsoft Access.

In the thierd Chapter describe the forms and examples runs of the application program.There are tables such as login form,main form,product entry form,stock list..etc. my own program that i made it by myself, it is a Stock Program, used for all the market's not just Dress market, it is a useful program, do jobs of more than one person in a speedy way without mistake, also it can be developed in the future by adding other properties to the program.

Finally I promise myself to do a program much more better than this, because this was my first program, maybe I made mistakes during programing it, but in the future I'll made one better than this in appearance and properties.

CHAPTER 1

1. BASIC CONCEPT OF DELPHI 7

1.1. Introduction

The name "Delphi" was never a term with which either Olaf Helmer or Norman Dalkey (the founders of the method) were particular happy. Since many of the early Delphi studies focused on utilizing the technique to make forecasts of future occurrences, the name was first applied by some others at Rand as a joke. However, the name stuck. The resulting image of a priestess, sitting on a stool over a crack in the earth, inhaling sulfur fumes, and making vague and jumbled statements that could be interpreted in many different ways, did not exactly inspire confidence in the method.

The straightforward nature of utilizing an iterative survey to gather information "sounds" so easy to do that many people have done "one" Delphi, but never a second. Since the name gives no obvious insight into the method and since the number of unsuccessful Delphi studies probably exceeds the successful ones, there has been a long history of diverse definitions and opinions about the method. Some of these misconceptions are expressed in statements such as the following that one finds in the literature:

- It is a method for predicting future events.
- It is a method for generating a quick consensus by a group.
- It is the use of a survey to collect information.
- It is the use of anonymity on the part of the participants.
- It is the use of voting to reduce the need for long discussions.
- It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing

detailed critical examination and discussion, not at forcing a quick compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective.

The essence of Delphi is structuring of the group communication process. Given that there had been much earlier work on how to facilitate and structure face-to-face meetings, the other important distinction was that Delphi was commonly applied utilizing a paper and pencil communication process among groups in which the members were dispersed in space and time. Also, Delphis were commonly applied to groups of a size (30 to 100 individuals) that could not function well in a face-to-face environment, even if they could find a time when they all could get together.

The result, however, is not merely confusion due to different names to describe the same things; but a basic lack of knowledge by many people working in these areas as to what was learned in the studies of the Delphi Method about how to properly employ these techniques and their impact on the communication process. There seems to be a great deal of "rediscovery" and repeating of earlier misconceptions and difficulties.

Given this situation, the primary objective of this chapter is to review the specific properties and methods employed in the design and execution of Delphi Exercises and to examine how they may best be translated into a computer based environment.

1.2. What is Delphi?

Delphi is an object-oriented, visual programming environment for rapid application development (RAD). With Delphi, you can write Windows programs more quickly and more easily than was ever possible before. You can create Win32 console applications or Win32 graphical user interface (GUI) programs. When creating Win32 GUI applications with Delphi, you have all the power of a true compiled programming language (Object

Pascal) wrapped up in a RAD environment. What this means is that you can create the user interface to a program (the *user interface* means the menus, dialog boxes, main window, and so on) using drag-and-drop techniques for true rapid application development. You can also drop ActiveX controls on forms to create specialized programs such as Web browsers in a matter of minutes. Delphi gives you all this, and at virtually no cost: You don't sacrifice program execution speed because Delphi generates fast compiled code.

Delphi provides all the tools you need to develop, test, and deploy applications, including a large library of reusable components, a suite of design tools, application and form templates, and programming wizards.

Delphi does a good job of hiding some of the low-level details that make up the guts of a Windows program, but it cannot write programs for you. In the end, you must still be a programmer, and that means you have to learn programming. That can be a long, uphill journey some days. The good news is that Delphi can make your trek fairly painless and even fun. Yes, you can work and have fun doing it!

1.2.1Developer Support Services and Web Site

Borland offers a variety of support options to meet the needs of its diverse developer community. To find out about support, refer to http://www.borland.com/devsupport/. From the Web site, you can access many newsgroups where Delphi developers exchange information, tips, and techniques. From the Web site, you can access many newsgroups where Delphi developers exchange information, tips, and techniques. The site also includes a list of books about Delphi, additional Delphi technical documents, and Frequently Asked Questions (FAQs).

1.2. A Tour of The Environment

This chapter explains how to start Delphi and gives you a quick tour of the main parts and tools of the integrated development environment (IDE).

1.3.1. Starting Delphi

You can start Delphi in the following ways:

- Double-click the Delphi icon (if you've created a shortcut).
- Choose Programs|Borland Delphi 7|Delphi 7 from the Windows Start menu.
- Choose Run from the Windows Start menu, then enter Delphi32.
- Double-click Delphi32.exe in the Delphi\Bin directory.

1.3.2Delphi (IDE)

When you first start Delphi, you'll see some of the major tools in the IDE. In Delphi, the IDE includes the menus, toolbars, Component palette, Object Inspector, Object TreeView, Code editor, Code Explorer, Project Manager, and many other tools. The particular features and components available to you will depend on which edition of Delphi you've purchased.

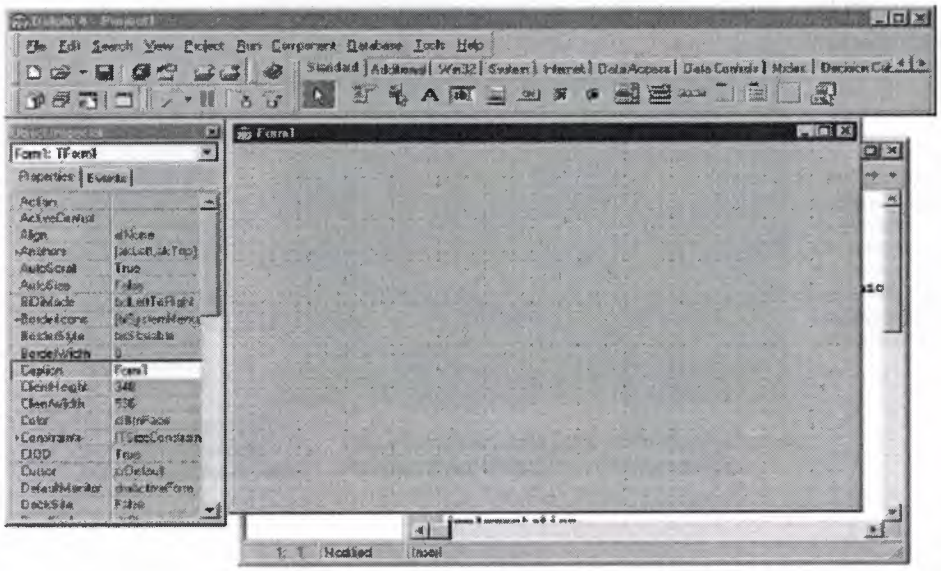


FIGURE 1.1.*The Delphi IDE and the initial blank form.*

The Delphi IDE is divided into three parts. The top window can be considered the main window. It contains the toolbars and the Component palette. The Delphi toolbars give you one-click access to tasks such as opening, saving, and compiling projects. The Component palette contains a wide array of components that you can drop onto your forms. (Components are text labels, edit controls, list boxes, buttons, and the like.) For convenience, the components are divided into groups. Did you notice the tabs along the top of the Component palette? Go ahead and click on the tabs to explore the different components available to you. To place a component on your form, you simply click the component's button in the Component palette and then click on your form where you want the component to appear. Don't worry about the fact that you don't yet know how to use components. You'll get to that in due time. When you are done exploring, click on the tab labeled Standard, because you'll need it in a moment.

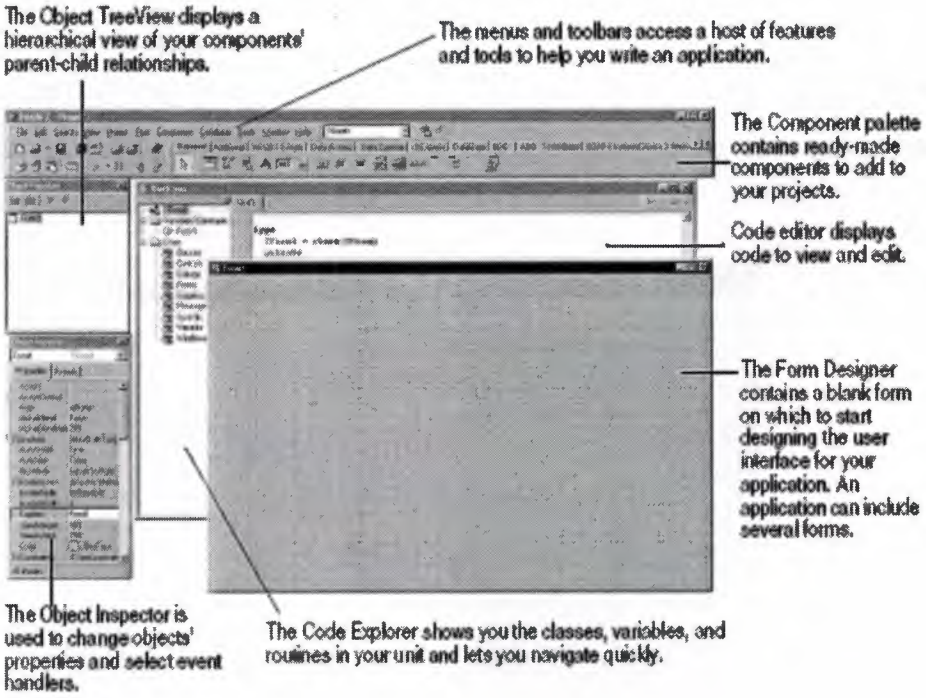


Fig. 1.2 IDE

Delphi's development model is based on two-way tools. This means that you can move back and forth between visual design tools and text-based code editing. For example, after using

the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the form file that contains the textual description of your form. You can also manually edit any code generated by Delphi without losing access to the visual programming environment.

From the IDE, all your programming tools are within easy reach. You can design graphical interfaces, browse through class libraries, write code, and compile, test, debug, and manage projects without leaving the IDE.

Delphi's development model is based on two-way tools. This means that you can move back and forth between visual design tools and text-based code editing. For example, after using the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the form file that contains the textual description of your form. You can also manually edit any code generated by Delphi without losing access to the visual programming environment.

From the IDE, all your programming tools are within easy reach. You can design graphical interfaces, browse through class libraries, write code, and compile, test, debug, and manage projects without leaving the IDE.

1.3.3 The Object Inspector

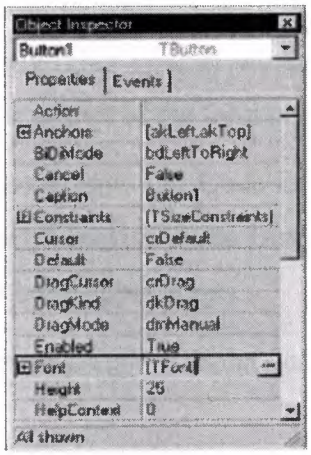


Fig. 1.3 object inspector

Below the main window and on the left side of the screen is the Object Inspector. It is through the Object Inspector that you modify a component's properties and events. You

will use the Object Inspector constantly as you work with Delphi. The Object Inspector has two tabs: the Properties tab and the Events tab. A component's *properties* control how the component operates. For example, changing the Color property of a component changes the background color of that component. The list of properties available varies from component to component, although components usually have several common elements (Width and Height properties, for instance).

The Events tab contains a list of events for a component. Events occur as the user interacts with a component. For example, when a component is clicked, an event is generated that tells you that the component was clicked. You can write code that responds to these events, performing specific actions when an event occurs. As with properties, the events that you can respond to vary from component to component.

1.3.4 The Delphi Workspace

The main part of the Delphi IDE is the workspace. The workspace initially displays the Form Designer. It should come as no surprise that the Form Designer enables you to create forms. In Delphi, a *form* represents a window in your program. The form might be the program's main window, a dialog box, or any other type of window. You use the Form Designer to place, move, and size components as part of the form creation process.

Hiding behind the Form Designer is the Code Editor. The Code Editor is where you type code when writing your programs. The Object Inspector, Form Designer, Code Editor, and Component palette work interactively as you build applications.

Now that you've had a look at what makes up the Delphi IDE, let's actually do something.

1.3.5 The Menus and Toolbars

The main window, which occupies the top of the screen, contains the main menu, toolbars, and Component palette.

Fig. 1.4 Menus and Toolbars

Delphi's toolbars provide quick access to frequently used operations and commands. Most toolbar operations are duplicated in the drop-down menus.

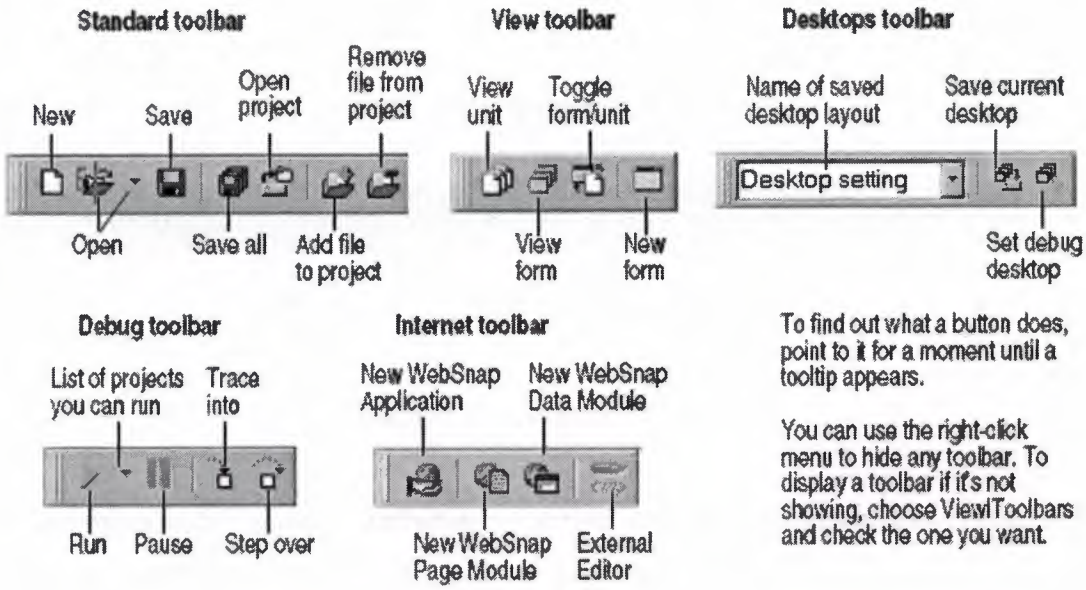


Fig. 1.5 Toolbars

Many operations have keyboard shortcuts as well as toolbar buttons. When a keyboard shortcut is available, it is always shown next to the command on the dropdown menu. You can right-click on many tools and icons to display a menu of commands appropriate to the object you are working with. These are called context menus.The toolbars are also customizable. You can add commands you want to them or move them to different locations.

1.3.2. The Component Palette and Form Designer

The Component palette, Form Designer, Object Inspector, and Object TreeView work together to help you build a user interface for your application. The Component palette includes tabbed pages with groups of icons representing visual or nonvisual components. The pages divide the components into various functional groups. For example, the Standard, Additional, and Win32 pages include windows controls such as an edit box and up/down button; the Dialogs page includes common dialog boxes to use for file operations such as opening and saving files.

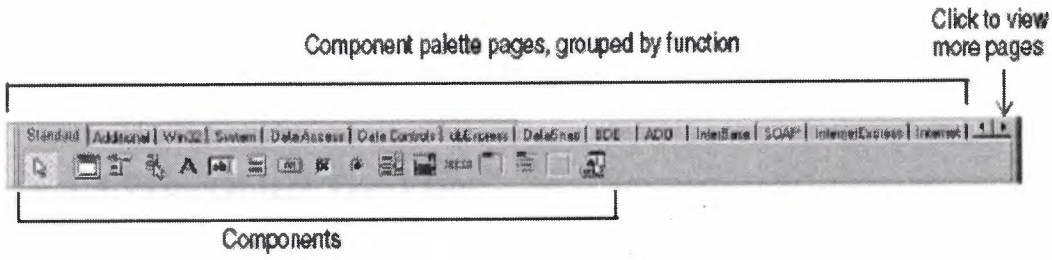


Fig. 1.6 Component Palatte

Each component has specific attributes properties, events, and methods that enable you to control your application. After you place components on the form, or Form Designer, you can arrange components the way they should look on your user interface. For the components you place on the form, use the Object Inspector to set design time properties, create event handlers, and filter visible properties and events, making the connection between your application's visual appearance and the code that makes your application run.

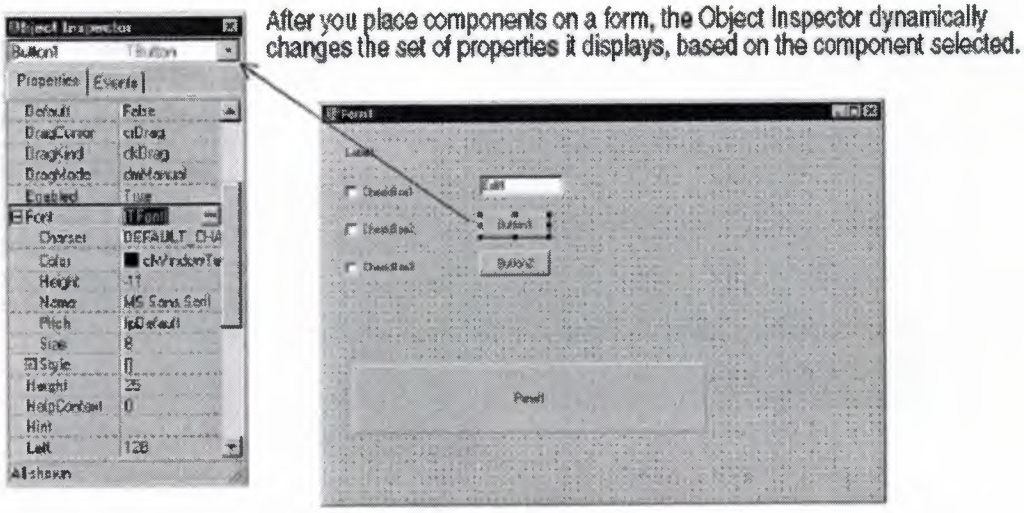


Fig. 1.7 Changing Set of Properties in Object Inspector

1.3.7The Object TreeView

The Object TreeView displays a component's sibling and parent-child relationships in a hierarchical, or tree diagram. The tree diagram is synchronized with the Object Inspector and the Form Designer so that when you change focus in the Object TreeView, both the Object Inspector and the form change focus.

You can use the Object TreeView to change related components' relationships to each other. For example, if you add a panel and check box component to your form, the two components are siblings. But in the Object TreeView, if you drag the check box on top of the panel icon, the check box becomes the child of the panel.

If an object's properties have not been completed, the Object TreeView displays a red question mark next to it. You can also double-click any object in the tree diagram to open the Code editor to a place where you can write an event handler. If the Object TreeView isn't displayed, choose View|Object TreeView.

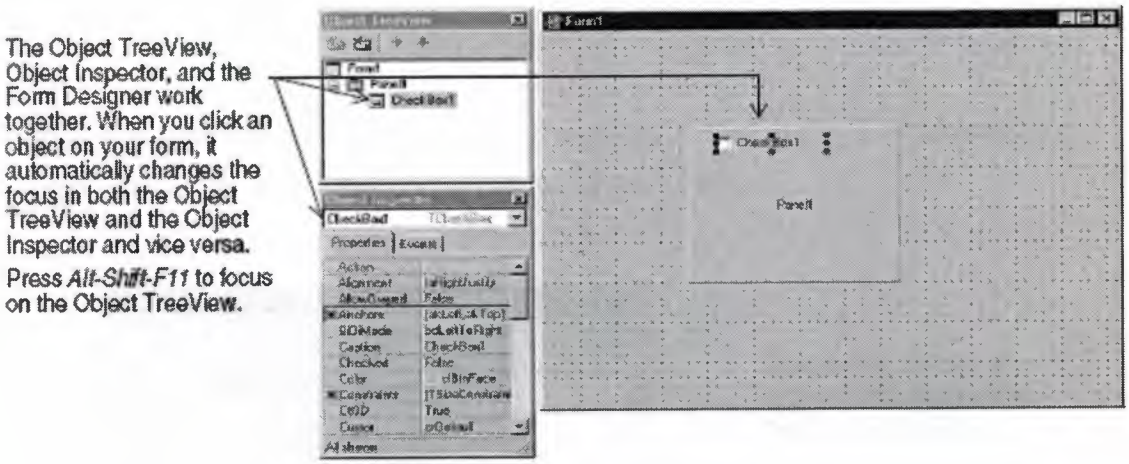


Fig. 1.8 Panel

The Object TreeView is especially useful for displaying the relationships between database objects.

1.3.8The Object Repository

The Object Repository contains forms, dialog boxes, data modules, wizards, DLLs, sample applications, and other items that can simplify development. Choose File| New|Other to display the New Items dialog box when you begin a project. The New Items dialog box is the same as the Object Repository. Check the Repository to see if it contains an object that resembles one you want to create.

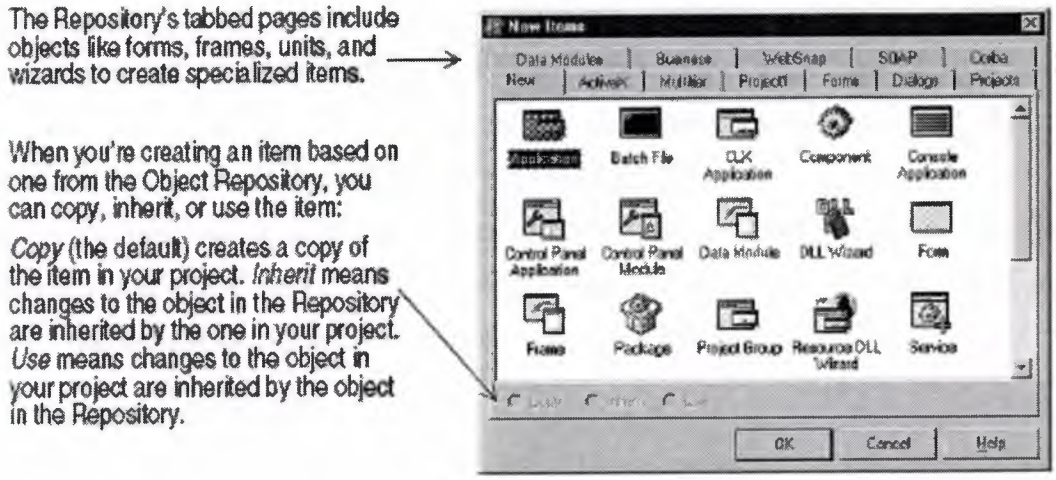


Fig. 1.9 Object Repository

To edit or remove objects from the Object Repository, either choose Tools|Repository or right-click in the New Items dialog box and choose Properties.

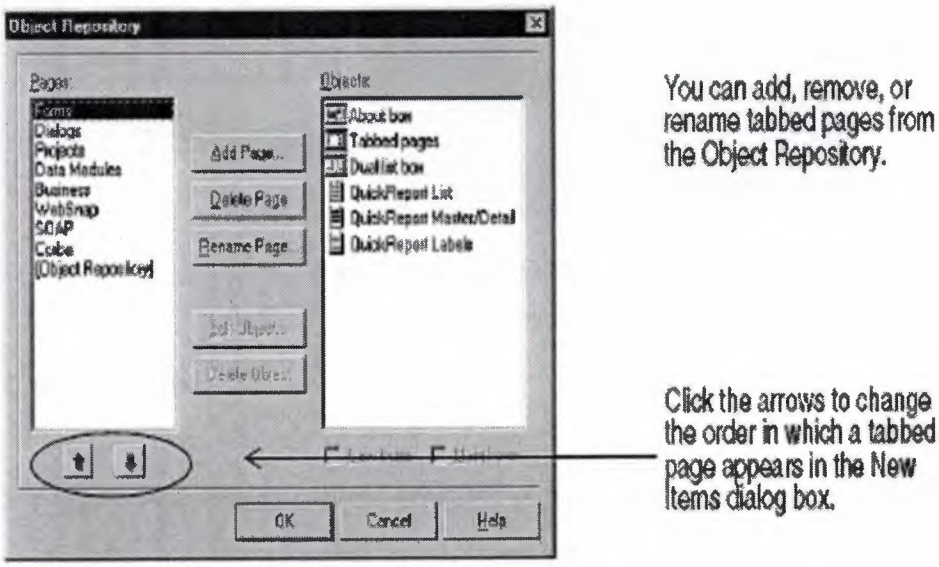


Fig. 1.10 Adding project and form templates to the Object Repository

1.3.9The Code Editor

As you design the user interface for your application, Delphi generates the underlying Delphi code. When you select and modify the properties of forms and objects, your changes are automatically reflected in the source files.

You can add code to your source files directly using the built-in Code editor, which is a full-featured ASCII editor. Delphi provides various aids to help you write code, including the Code Insight tools, class completion, and code browsing.

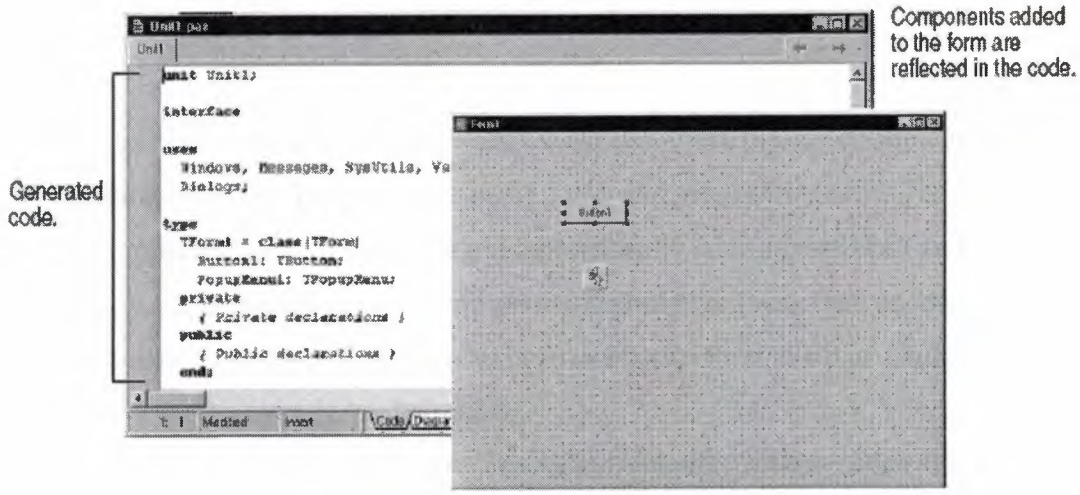


Fig. 1.11 Code Editor

1.3.9.1Code Insight

The Code Insight tools display context-sensitive pop-up windows.

Tool	How it works
Code completion	Type a class name followed by a dot (.) to display a list of properties, methods, and events appropriate to the class, select it, and press Enter. In the **interface** section of your code you can select more than one item. Type the beginning of an assignment statement and press Ctrl+space to display a list of valid values for the variable. Type a procedure, function, or method name to bring up a list of arguments.
Code parameters	Type a method name and an open parenthesis to display the syntax for the method's arguments.
Tooltip expression evaluation	While your program has paused during debugging, point to any variable to display its current value.
Tooltip symbol insight	While editing code, point to any identifier to display its declaration.
Code templates	Press Ctrl+Jto see a list of common programming statements that you can insert into your code. You can create your own templates in addition to the ones supplied with Delphi.

To turn these tools on or off, choose Tools|Editor Options and click the Code Insight tab.

Check or uncheck the tools in the Automatic features section.

1.3.10 Class Completion

Class completion generates skeleton code for classes. Place the cursor anywhere within a class declaration of the **interface** section of a unit and press Ctrl+Shift+C or right click and choose Complete Class at Cursor. Delphi automatically adds private **read** and **write** specifiers to the declarations for any properties that require them, then creates keleton code for all the class's methods. You can also use class completion to fill in class declarations for methods you've already implemented.

To turn on class completion, choose Tools|Environment Options, click the Explorer tab, and make sure Finish incomplete properties is checked.

1.3.11Code Browsing

While passing the mouse over the name of any class, variable, property, method, or other identifier, the pop-up menu called Tooltip Symbol Insight displays where the identifier is declared. Press Ctrl and the cursor turns into a hand, the identifier turns blue and is underlined, and you can click to jump to the definition of the identifier. The Code editor has forward and back buttons like the ones on Web browsers. As you jump to these definitions, the Code editor keeps track of where you've been in the code. You can click the drop-down arrows next to the Forward and Back buttons to ove forward and backward through a history of these references.

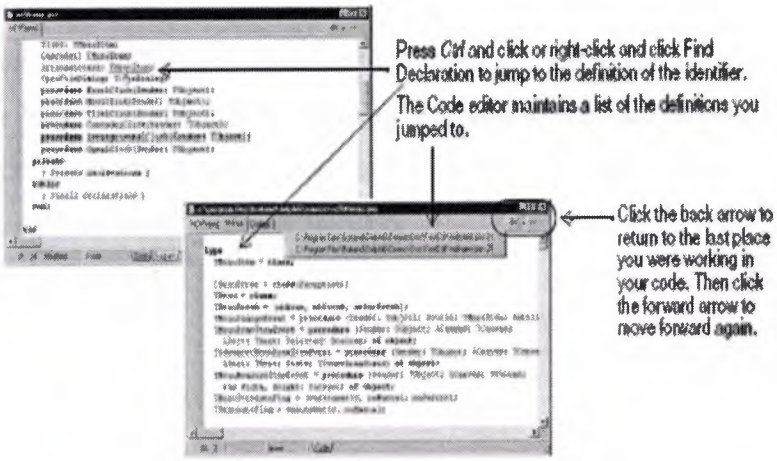


Fig. 1.12 Code Editor

You can also move between the declaration of a procedure and its implementation by pressing Ctrl+Shift+↑ or Ctrl+Shift+↓.
To customize your code editing environment, see "Customizing the Code Editor".

1.3.12 The Diagram Page

The bottom of the Code editor may contain one or more tabs, depending on which edition of Delphi you have. The Code page, where you write all your code, appears in the foreground by default. The Diagram page displays icons and connecting lines representing the relationships between the components you place on a form or data module. These relationships include siblings, parent to children, or components to properties.

To create a diagram, click the Diagram page. From the Object TreeView, simply drag one or multiple icons to the Diagram page to arrange them vertically. To arrange them horizontally, press Shift while dragging. When you drag icons with parentchildren or component-property dependencies onto the page, the lines, or connectors, that display the dependent relationships are automatically added. For example, if you add a dataset component to a data module and drag the dataset icon plus its property icons to the Diagram page, the property connector automatically connects the property icons to the dataset icon.

For components that don't have dependent relationships but where you want to show one, use the toolbar buttons at the top of the Diagram page to add one of four connector types, including allude, property, master/detail, and lookup. You can also add comment blocks that connect to each other or to a relevant icon.

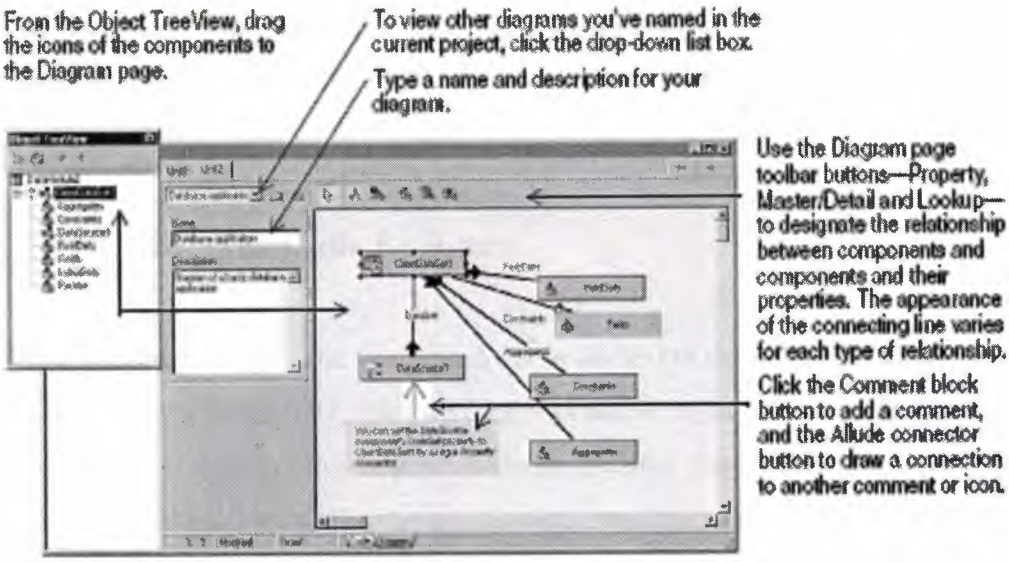


Fig. 1.13 Diagram Page Toolbar Button

You can type a name and description for your diagram, save the diagram, and print it when you are finished

.

1.3.13 Wiewing Form Code

Forms are a very visible part of most Delphi projects they are where you design the user interface of an application. Normally, you design forms using Delphi's visual tools, and Delphi stores the forms in form files. Form files (.dfm, or .xfm for a CLX application) describe each component in your form, including the values of all persistent properties. To view and edit a form file in the Code editor, right-click the form and select View as Text. To return to the graphic view of your form, right-click and choose View as Form.

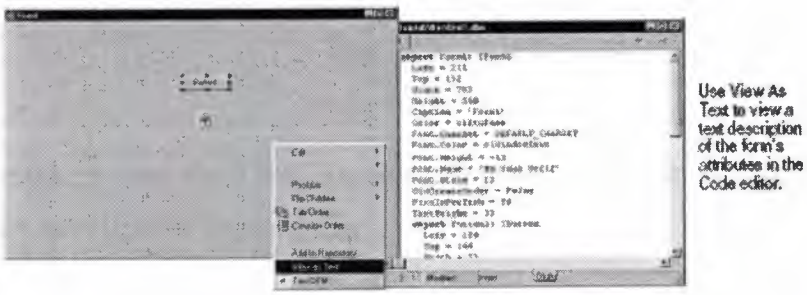


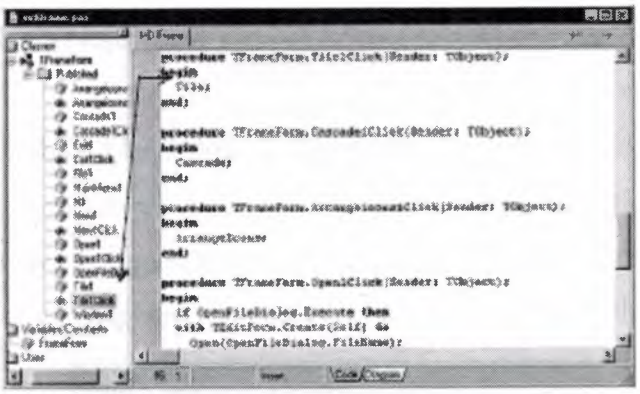
Fig. 1.14 View as Text Description of Form

You can save form files in either text (the default) or binary format. Choose Tools| Environment Options, click the Designer page, and check or uncheck the New forms as text check box to designate which format to use for newly created forms.

1.3.14 The Code Explorer

When you open Delphi, the Code Explorer is docked to the left of the Code editor window, depending on whether the Code Explorer is available in the edition of Delphi you have. The Code Explorer displays the table of contents as a tree diagram for the source code open in the Code editor, listing the types, classes, properties, methods, global variables, and routines defined in your unit. It also shows the other units listed in the **uses** clause.

You can use the Code Explorer to navigate in the Code editor. For example, if you double-click a method in the Code Explorer, a cursor jumps to the definition in the class declaration in the interface part of the unit in the Code editor.



Double-click an item in the Code Explorer and the cursor moves to that item's implementation in the Code editor. Press *Ctrl+Shift+E* to move the cursor back and forth between the last place you were in the Code Explorer and Code editor.

Each item in the Code Explorer has an icon that designates its type.

Fig. 1.15 Code Explorer

To configure how the Code Explorer displays its contents, choose Tools|Environment Options and click the Explorer tab.

1.3.15 The Project Manager

When you first start Delphi, it automatically opens a new project. A project includes several files that make up the application or DLL you are going to develop. You can view and organize these files such as form, unit, resource, object, and library files in a project management tool called the Project Manager. To display the Project Manager, choose View|Project Manager.

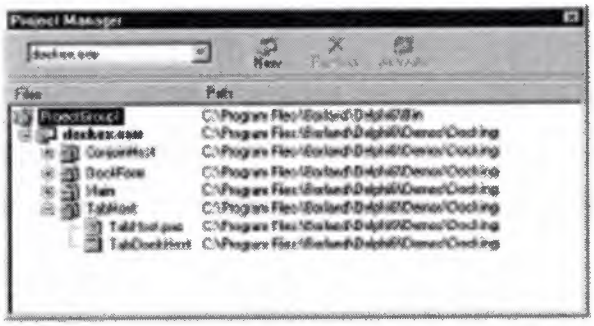
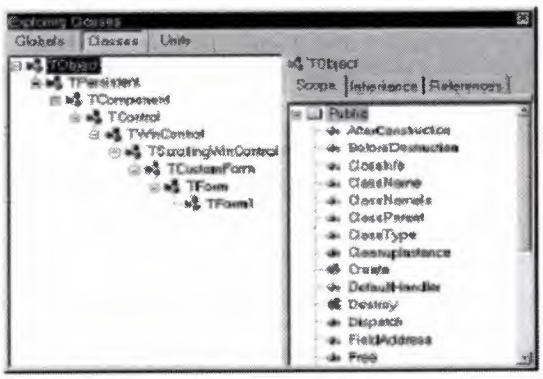


Fig. 1.16 Project Manager

You can use the Project Manager to combine and display information on related projects into a single project group. By organizing related projects into a group, such as multiple executables, you can compile them at the same time. To change project options, such as compiling a project, you can use Setting project options.

1.3.16 The Project Browser

The Project Browser examines a project in detail. The Browser displays classes, units, and global symbols (types, properties, methods, variables, and routines) your project declares or uses in a tree diagram. Choose View|Browser to display the Project Browser.



The Project Browser has two resizeable panes: the inspector pane (on the left) and the Details pane. The Inspector pane has three tabs for globals, classes, and units.

Globals displays classes, types, properties, methods, variables, and routines.

Classes displays classes in a hierarchical diagram.

Units displays units, identifiers declared in each unit, and the other units that use and are used by each unit.

Fig. 1.17 Project Browser

By default, the Project Browser displays the symbols from units in the current project only. You can change the scope to display all symbols available in Delphi. Choose Tools|Environment Options, and on the Explorer page, check All symbols.

1.4 Programming With Delphi

The following sections provide an overview of software development with Delphi, including creating a project, working with forms, writing code, and compiling, debugging, deploying, and internationalizing applications, and including the types of projects you can develop.

1.4.1 Creating a Project

A project is a collection of files that are either created at design time or generated when you compile the project source code. When you first start Delphi, a new project opens. It automatically generates a project file (Project1.dpr), unit file (Unit1.pas), and resource file (Unit1.dfm; Unit1.xfm for CLX applications), among others. If a project is already open but you want to open a new one, choose either File|New| Application or File|New|Other and double-click the Application icon. File|New| Other opens the Object Repository, which provides additional forms, modules, and frames as well as predesigned templates such as dialog boxes to add to your project. When you start a project, you have to know what you want to develop, such as an application or DLL.

1.4.2 Adding Data Modules

A data module is a type of form that contains nonvisual components only. Nonvisual components can be placed on ordinary forms alongside visual components. But if you plan on reusing groups of database and system objects, or if you want to isolate the parts of your

application that handle database connectivity and business rules, data modules provide a convenient organizational tool.To create a data module, choose File|New|Data Module. Delphi opens an empty data module, which displays an additional unit file for the module in the Code editor, and adds the module to the current project as a new unit. Add nonvisual components to a data module in the same way as you would to a form.

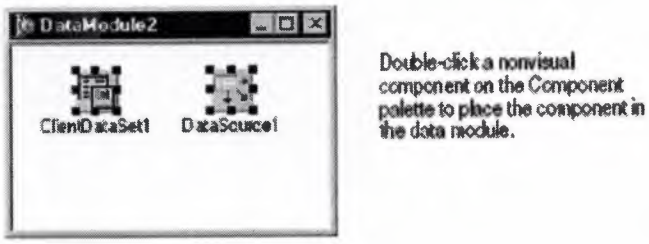


Fig. 1.18 Adding Data Modules

When you reopen an existing data module, Delphi displays its components.

1.4.3 Building the user interface

With Delphi, you first create a user interface (UI) by selecting components from the Component palette and placing them on the main form.

1.4.4 Placing components on a form

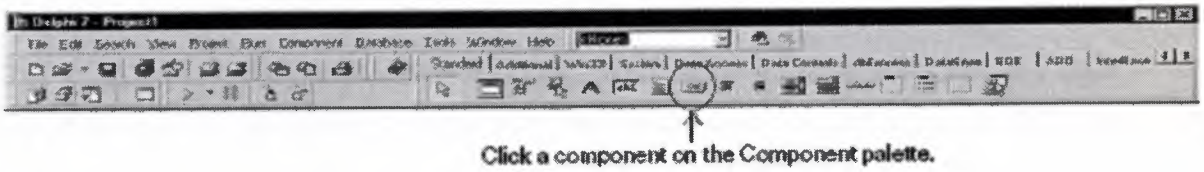


Fig. 1.19 Component Palette

Fig.1.20 Placing Component

To place a Button on the form, click once on the Button component on the toolbar. Then move the mouse cursor over to the Form and click on the Form where you want the Button to be. Repeat the same procedure with the Label component.

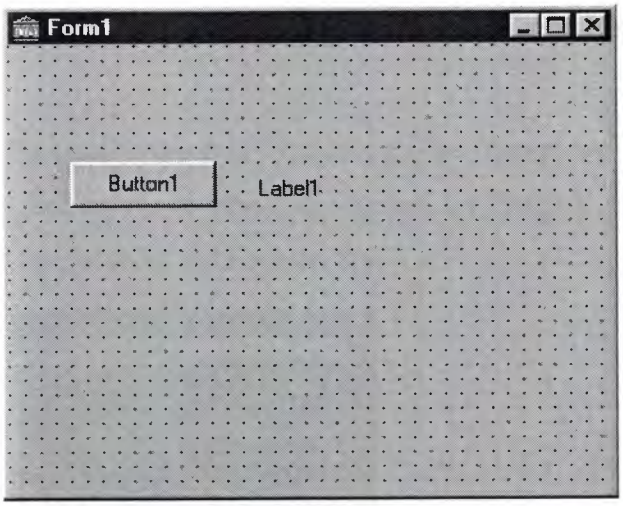


Fig.1.21 Buton and Label

It will look something like this. Actually you can run your application now. Simply press F9, click Ok to save your project and it's running.

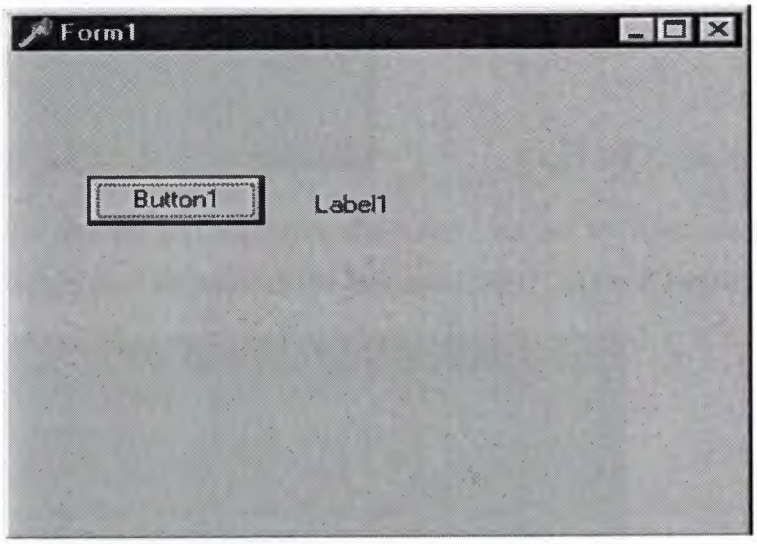


Fig.1.22. Run

Try to click on the button. Nothing happens? Well, since we have not added any code yet, there are no instructions for what will happen when you press the button. This will be done later. For now, exit your application (click on the ☒).

1.4.5 Setting the properties of the components

To change the text 'Button1' on the button, we change the value of its Caption property. Click on the Button once ('select' the Button), move to the Object Inspector and enter the new value on the Caption row.

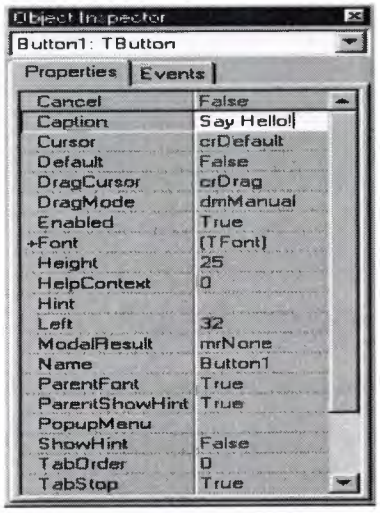


Fig.1.23. Properties Component

To remove the text from the Label, we set the Caption property to " (empty string). Select the Label and delete the text 'Label1' on the Caption row of the Object Inspector.

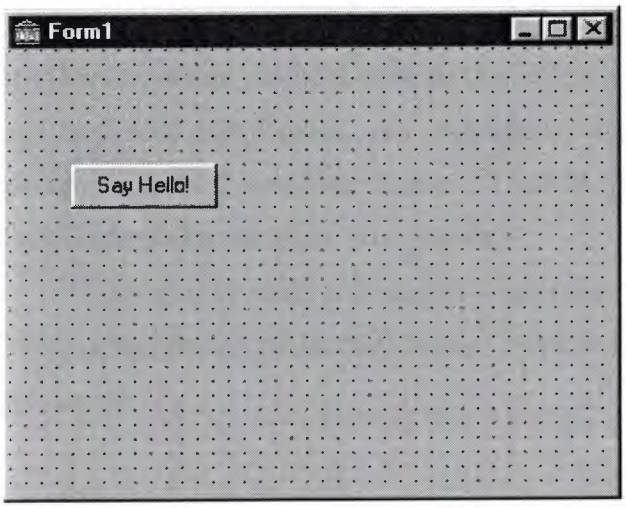


Fig.1.24. Button (Say Hello)

If we run our application now, we see that we have made some progress since the first step. Press F9 to run.

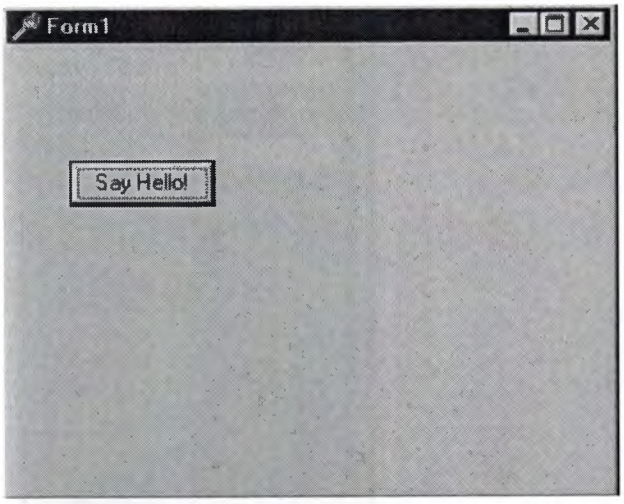


Fig.1.25. Run form

The Button and the Label now show the messages we want them to do. However, it still doesn't happen anything if we press the Button. We will deal with this now.

1.4.6 Writing Code

An integral part of any application is the code behind each component. While Delphi's RAD environment provides most of the building blocks for you, such as preinstalled visual and nonvisual components, you will usually need to write event handlers, methods, and perhaps some of your own classes. To help you with this task, you can choose from thousands of objects in the class library.

To specify what will happen if we press the button, we enter code for the OnClick event of the Button. Select the button, move over to the Object Inspector and click on the Events tab.

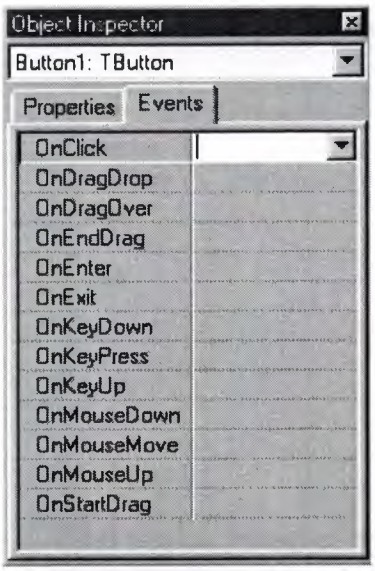


Fig.1.26. OnClick

Double click on the OnClick row. The Code Editor will popup and a procedure for the OnClick event will be created.

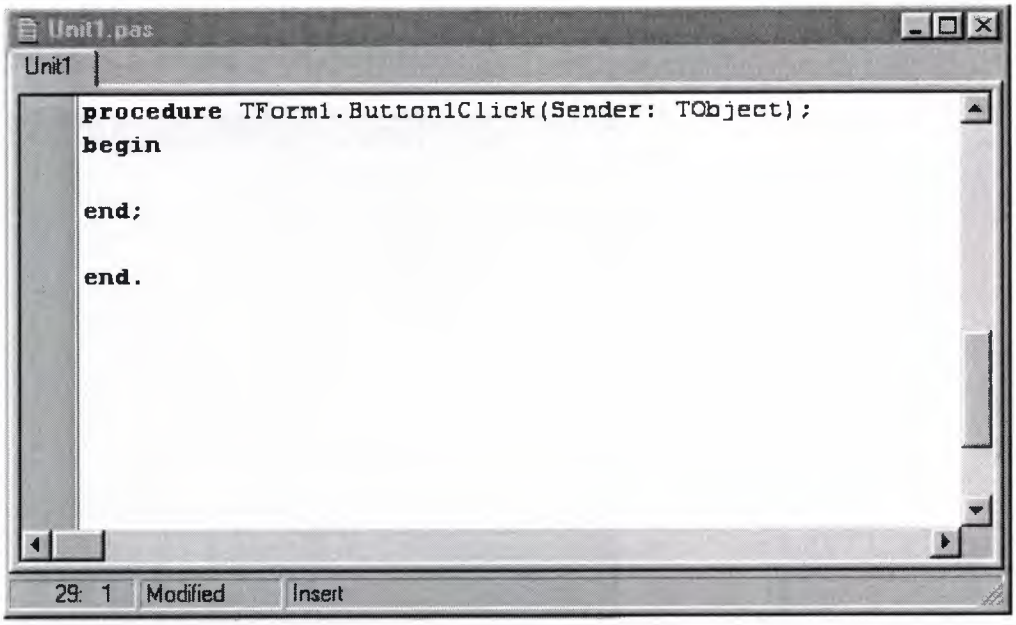


Fig.1.27. Unit.pas page

You can fill this empty procedure with code that is going to be executed when you press the Button. The task of this application was to let the Label show the 'Hello World' message when we click on the button, remember?

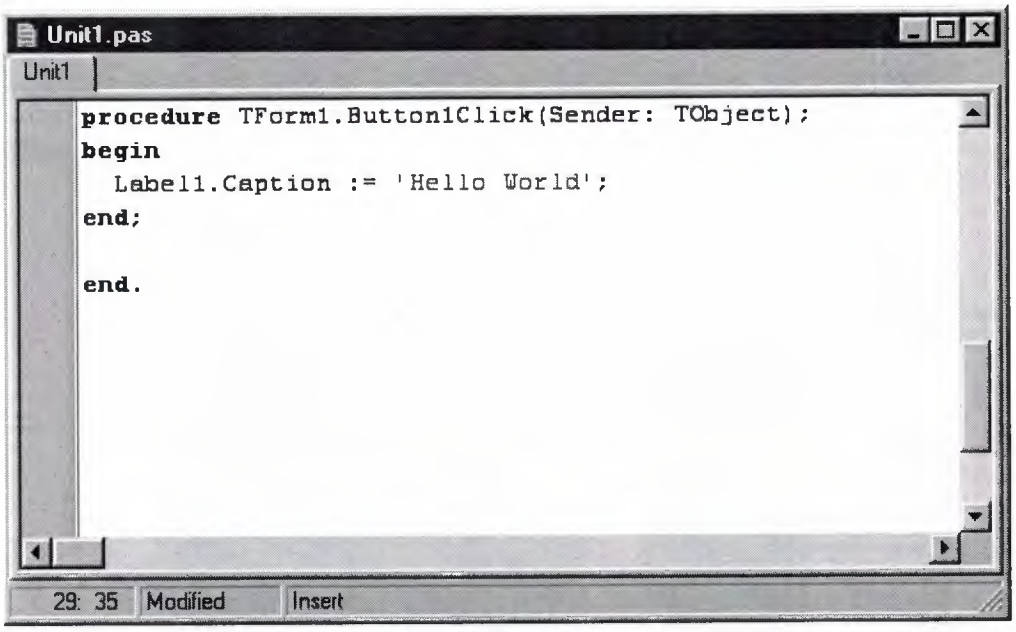


Fig.1.28. On Cick Operation

This code should do the trick. It will be explained in the next lessons. Just type it in, run the application and press the Button.

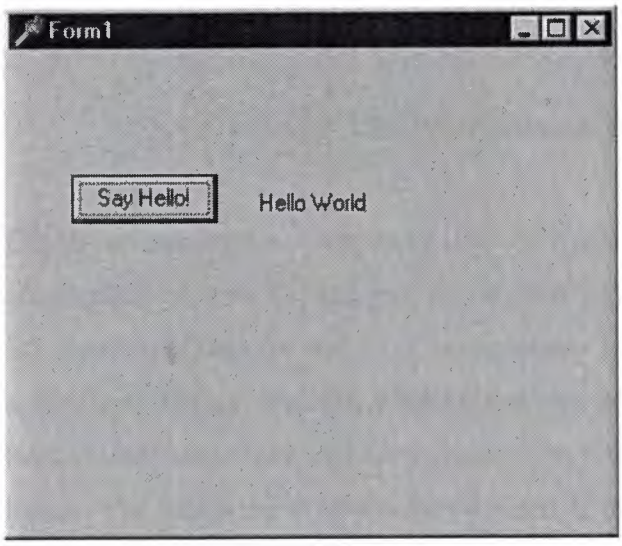


Fig.1.28. Run Aplication (say hello)

Here we go! Hello World!

As you have noticed by now, this is not a very useful application. However, by creating it you have learnt the three essential steps of creating any application in Borland Delphi. Creating applications in other 'visual' programming languages like Visual Basic or Borland C++ Builder is done in the same way, it is sort of a standard. But once you have

learnt some more Delphi, you will probably not change to another programming language unless you have to. Delphi has all the features you will ever need.

Select the component and drag it to wherever you want on the form.

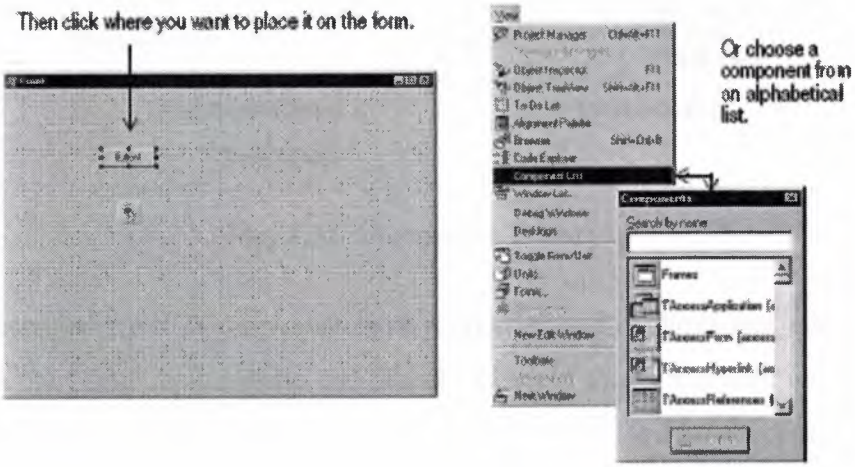


Fig. 1.30 Component List

1.4.6.1Using The Component Library

Delphi comes with a component library made up of objects, some of which are also components or controls, that you use when writing code. You can use VCL components for Windows applications and CLX components for Windows and Linux applications. The component library includes objects that are visible at Runtime such as edit controls, buttons, and other user interface elements as well-as non visual controls like datasets and timers. The following diagram shows some of the principal classes that make up the VCL hierarchy. The CLX hierarchy is similar.

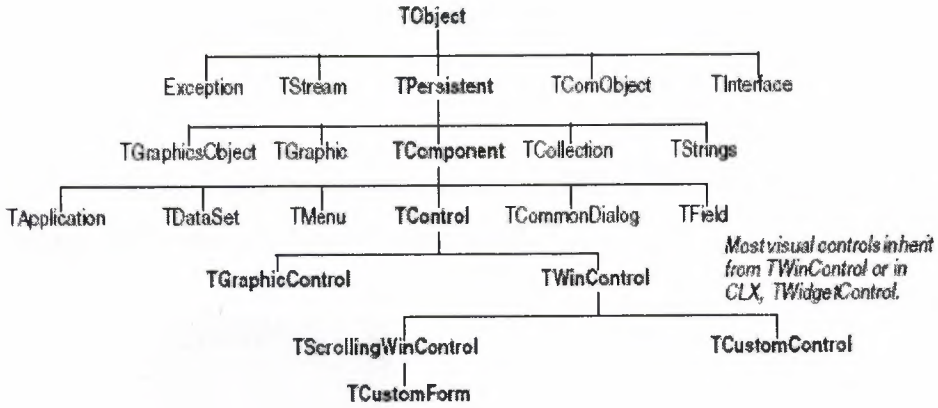


Fig. 1.31 Component Library

Objects descended from TComponent have properties and methods that allow them to be installed on the Component palette and added to Delphi forms and data modules Because the components are hooked into the IDE, you can use tools like the Form Designer to develop applications quickly.Components are highly encapsulated. For example, buttons are preprogrammed to respond to mouse clicks by firing OnClick events. If you use a button control, you don't have to write code to handle generated events when the button is clicked; you are responsible only for the application logic that executes in response to the click itself. Most editions of Delphi come with the component library source code and examples of Delphi programming techniques.

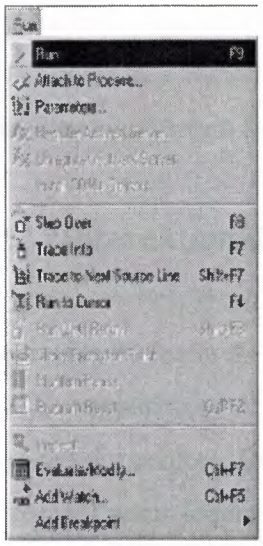
1.4.7Compiling and Debugging Projects

After you have written your code, you will need to compile and debug your project. With Delphi, you can either compile your project first and then separately debug it, or you can compile and debug in one step using the integrated debugger. To compile your program with debug information, choose Project|Options, click the Compiler page, and make sure Debug information is checked.

Delphi uses an integrated debugger so that you can control program execution, watch variables, and modify data values. You can step through your code line by line, examining the state of the program at each breakpoint. To use the integrated debugger, choose

Tools|Debugger Options, click the General page, and make sure Integrated debugging is checked.

You can begin a debugging session in the IDE by clicking the Run button on the Debug toolbar, choosing Run|Run, or pressing F9.



Choose any of the debugging commands from the Run menu. Some commands are also available on the toolbar.

Run button

Fig. 1.32 Compiling and Debugging

With the integrated debugger, many debugging windows are available, including Breakpoints, Call Stack, Watches, Local Variables, Threads, Modules, CPU, and Event Log. Display them by choosing View|Debug Windows. Not all debugger views are available in all editions of Delphi.

Once you set up your desktop as you like it for debugging, you can save the settings as the debugging or runtime desktop. This desktop layout will be used whenever you are debugging any application.

1.4.7.1 Deploying Applications

You can make your application available for others to install and run by deploying it When you deploy an application, you will need all the required and supporting files, such as the executables, DLLs, package files, and helper applications. Delphi comes bundled with a setup toolkit called InstallShield Express that helps you create an installation program with these files. To install InstallShield Express, from the Delphi setup screen, choose InstallShield Express Custom Edition for Delphi.

1.4.7.2Internationalizing Applications

Delphi offers several features for internationalizing and localizing applications. The IDE and the VCL support input method editors (IMEs) and extended character sets to internationalize your project. Delphi includes a translation suite, not available in all editions of Delphi, for software localization and simultaneous development for different locales. With the translation suite, you can manage multiple localized versions of an application as part of a single project.

<u>The translation suite includes three integrated tools:</u>

- Resource DLL wizard, a DLL wizard that generates and manage resource DLLs.
- Translation Manager, a table for viewing and editing translated resources.
- Translation Repository, a shared database to store translations.

To open the Resource DLL wizard, choose File|New|Other and double-click the Resource DLL Wizard icon. To configure the translation tools, choose Tools| Translation Tools Options.

1.4.8Types of Projects

All editions of Delphi support general-purpose 32-bit Windows programming, DLLs, packages, custom components, multithreading, COM (Component Object Model) and automation controllers, and multiprocess debugging. Some editions support server

applications such as Web server applications, database applications, COM servers, multi-tiered applications, CORBA, and decision-support systems.

1.4.8.1Delphi (CLX Applications)

You can use Delphi, to develop cross-platform 32-bit applications that run on both the Windows and Linux operating systems. To develop a CLX application, choose File| New|CLX Application. The IDE is similar to that of a regular Delphi application, except that only the components and items you can use in a CLX application appear on the Component palette and in the Object Repository. Windows-specific features supported on Delphi will not port directly to Linux environments.

1.4.8.2 Delphi (Database Applications)

Delphi offers a variety of database and connectivity tools to simplify the development of database applications. To create a database application, first design your interface on a form using the Data Controls page components. Second, add a data source to a data module using the Data Access page. Third, to connect to various database servers, add a dataset and data connection component to the data module from the previous or corresponding pages of the following connectivity tools:

• dbExpress is a collection of database drivers for cross-platform applications that provide fast access to SQL database servers, including DB2, Informix, InterBase, MSSQL, MySQL, and Oracle. With a dbExpress driver, you can access databases using unidirectional datasets.

• The Borland Database Engine (BDE) is a collection of drivers that support many popular database formats, including dBASE, Paradox, FoxPro, Microsoft Access, and any ODBC data source. ActiveX Data Objects (ADO) is Microsoft's high-level interface to any data source, including relational and nonrelational databases, e-mail and file systems, text and graphics, and custom business objects.

• InterBase Express (IBX) components are based on the custom data access Delphi component architectures. IBX applications provide access to advanced InterBase features and offer the highest performance component interface for InterBase 5.5 and later. IBX is compatible with Delphi's library of data-aware components. Certain database connectivity tools are not available in all editions of Delphi.

1.4.9Administrator (BDE)

Use the BDE Administrator (BDEAdmin.exe) to configure BDE drivers and set up the aliases used by data-aware VCL controls to connect to databases.

1.4.10 Database Explorer

The SQL Explorer (DBExplor.exe) lets you browse and edit databases. You can use it to create database aliases, view schema information, execute SQL queries, and maintain data dictionaries and attribute sets.

1.4.11Database Desktop

The Database Desktop (DBD32.exe) lets you create, view, and edit Paradox and dBase database tables in a variety of formats.

1.4.12Data Dictionary

When you use the BDE, the Data Dictionary provides a customizable storage area, independent of your applications, where you can create extended field attribute sets that describe the content and appearance of data. The Data Dictionary can reside on a remote server to share additional information.

1.4.13 Components of custom

The components that come with Delphi are preinstalled on the Component palette and offer a range of functionality that should be sufficient for most of your development needs. You could program with Delphi for years without installing a new component, but you may sometimes want to solve special problems or display particular kinds of behavior that require custom components. Custom components promote code reuse and consistency across applications. You can either install custom components from third-party vendors or create your own. To create a new component, choose Component|New Component to display the New Component wizard.

1.4.14 Dynamic-link libraries

Dynamic-link libraries (DLLs) are compiled modules containing routines that can be called by applications and by other DLLs. A DLL contains code or resources typically used by more than one application.

1.4.15 Delphi (COM and ActiveX)

Delphi supports Microsoft's COM standard and provides wizards for creating ActiveX controls. Choose File|New|Other and click the ActiveX tab to access the wizards. Sample ActiveX controls are installed on the ActiveX page of the Component palette. Numerous COM server components are provided on the Servers tab of the Component palette. You can use these components as if they were VCL components. For example, you can place one of the Microsoft Word components onto a form to bring up an instance of Microsoft Word within an application interface.

1.4.16Cmponent Type Libraries

Type libraries are files that include information about data types, interfaces, member functions, and object classes exposed by an ActiveX control or server. By including a type library with your COM application or ActiveX library, you make information about these entities available to other applications and programming tools. Delphi provides a Type Library editor for creating and maintaining type libraries.

1.5Work Area (İDE)

The IDE provides many tools to support development, so you'll want to reorganize your work area for maximum convenience. You can rearrange menus and toolbars, combine tool windows, and save your new desktop layout.

1.5.1Arranging Menus and Toolbars

In the main window, you can reorganize the menu, toolbars, and Component palette by clicking the grabber on the left-hand side of each one and dragging it to another location.

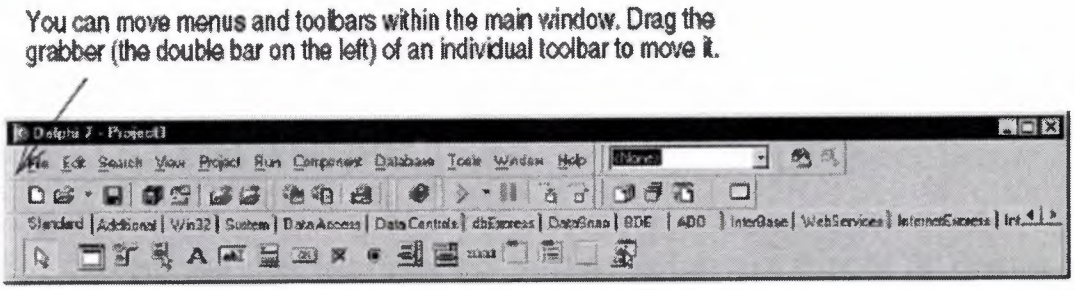


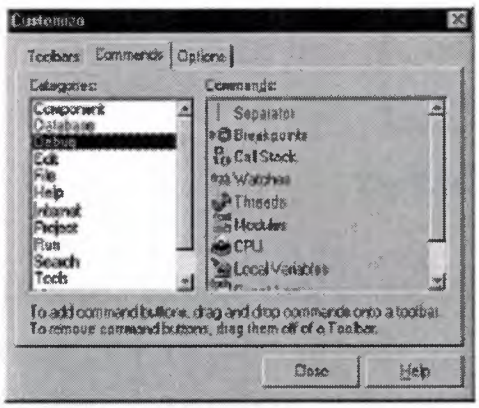
Fig. 1.33 Arranging Menus and Toolbars

You can separate parts from the main window and place them elsewhere on the screen or remove them from the desktop altogether. This is useful if you have a dual monitor setup.

Main window organized differently.

Fig. 1.34 Main Window

You can add or delete tools from the toolbars by choosing View|Toolbars|Customize. Click the Commands page, select a category, select a command, and drag it to the toolbar where you want to place it.



On the Commands page, select any command and drag it onto any toolbar.

On the Options page, click Show tooltips to make sure the hints for components and toolbar icons appear.

Fig. 1.35 Customize Command

1.5.2 Tool Windows

You can open and close individual tool windows and arrange them on the desktop as you wish. Many windows can also be docked to one another for easy management. Docking which means attaching windows to each other so that they move Together helps you use screen space efficiently while maintaining fast access to tools. From the View menu, you can bring up any tool window and then dock it directly to another. For example, when you first open Delphi in its default configuration, the Code Explorer is docked to the left of the

Code editor. You can add the Project Manager to the first two to create three docked windows.

Here the Project Manager and Code Explorer are docked to the Code editor. You can combine, or "dock" windows with either grabbers, as on the right, or tabs, as on page 5-4.

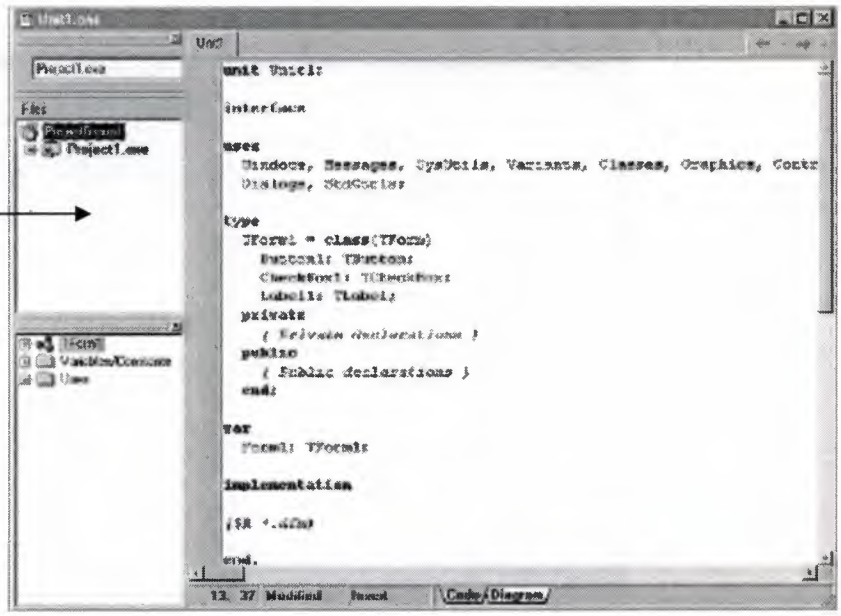


Fig. 1.36 Docking Tool Windows

To dock a window, click its title bar and drag it over the other window. When the drag outline narrows into a rectangle and it snaps into a corner, release the mouse. The two windows snap together.

To get docked windows with grabbers, release the mouse when the drag outline snaps to the window's corner.

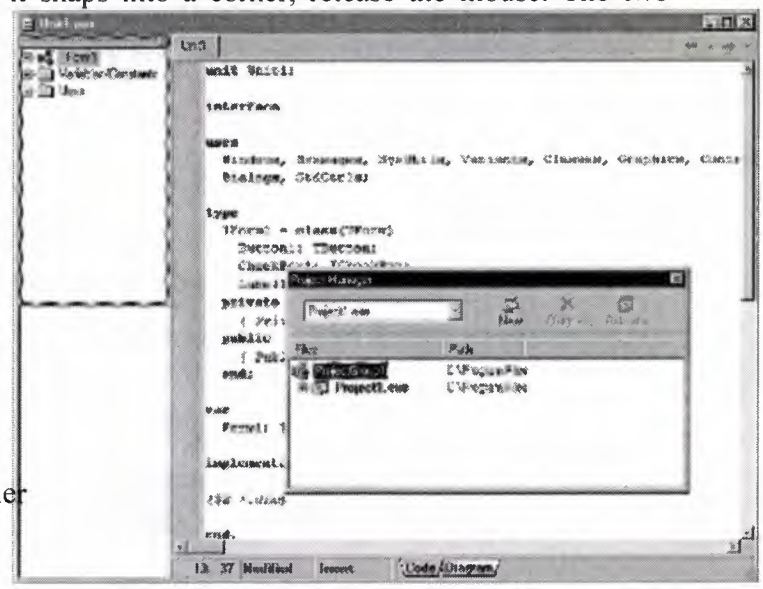


Fig. 1.37 Two Windows Snap Together

You can also dock tools to form tabbed windows.

To get docked windows that are tabbed, release the mouse *before* the drag outline snaps to the other window's corner.

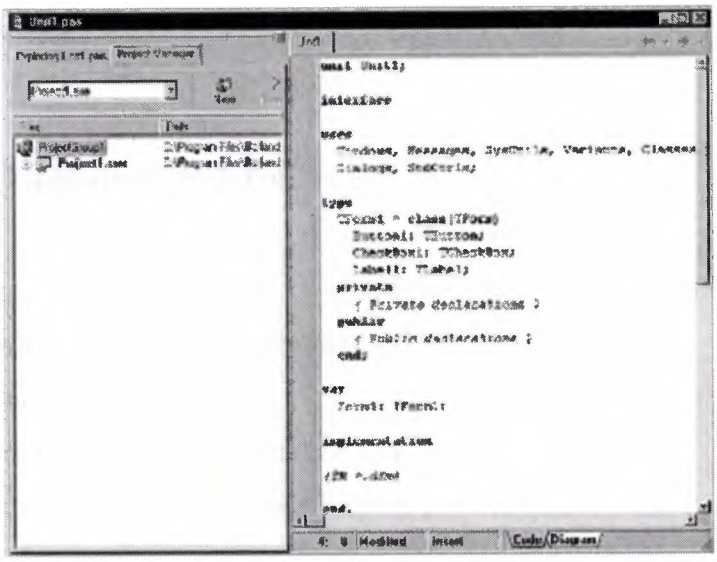


Fig. 1.38 Docking Tools to Form

To undock a window, double click its grabber or tab, or click and drag the tab outside of the docking area. To turn off automatic docking, either press the Ctrl key while moving windows around the screen, or choose Tools|Environment Options, click the references page, and uncheck the Auto drag docking check box.

1.5.3 Desktop Layouts

You can customize and save your desktop layout. The Desktops toolbar in the IDE includes a pick list of the available desktop layouts and two icons to make it easy to customize the desktop.

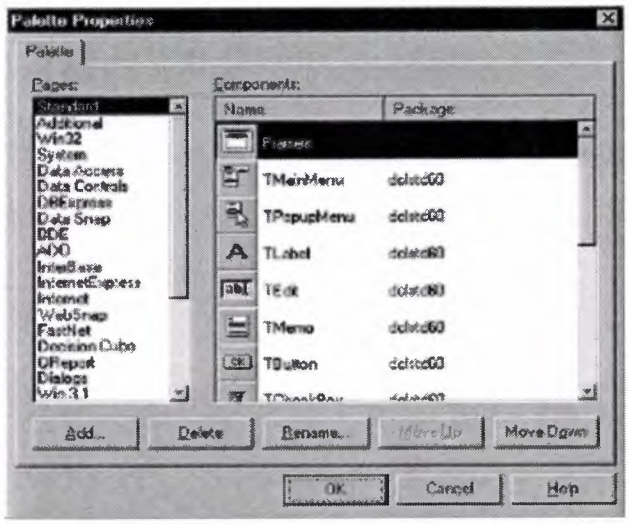
Fig. 1.39 Saving Desktop Layouts

Arrange the desktop as you want, including displaying, sizing, and docking particular windows. On the Desktops toolbar, click the Save current desktop icon or choose View|Desktops|Save Desktop, and enter a name for your new layout.

1.4.5 The Component Palette

To add, delete, rearrange, or rename pages, or to hide or rearrange components, use the Palette Properties dialog box. You can open this dialog box in several ways:

- Choose Component|Configure Palette.
- Choose Tools|Environment Options and click the Palette tab.
- Right-click the Component palette and choose Properties.



You can rearrange the palette and add new pages.

Fig. 1.40 Palette Properties Dialog Box

1.4.6 Creating Component Templates

Component templates are groups of components that you add to a form in a single operation. Templates allow you to configure components on one form, then save their arrangement, default properties, and event handlers on the Component palette to reuse on other forms.

To create a component template, simply arrange one or more components on a form and set their properties in the Object Inspector, and select all of the components by dragging the mouse over them. Then choose Component|Create Component Template. When the Component Template Information dialog box opens, select a name for the template, the palette page on which you want it to appear, and an icon to represent the template on the palette.

After placing a template on a form, you can reposition the components independently, reset their properties, and create or modify event handlers for them just as if you had placed each component in a separate operation.

CHAPTER 2

DATABASE CONCEPT OF DELPHI 7

2.1 About Dbase And Paradox

2.1.1 Architecture of database

- Relational database concepts
- The pieces of a database system
- How the pieces fit together
- Multi-tier computing architecture
- Using multiple databases
- About dbase
-

2.1.2 Relational database concepts

A relational database-management system (RDBMS) is a system for storing and retrieving data, in which the data is organized into interrelated tables.

SQL Anywhere Studio provides two relational database systems. Adaptive Server Anywhere is the primary, full featured RDBMS, with a multitude of uses, from a network database server hosting many clients to a compact embedded database. UltraLite is a small-footprint relational database. The UltraLite deployment technology allows you to use Adaptive Server Anywhere features on even the smallest of devices.

- Database tables
- Relations between tables
- Other database objects

2.1.3Accessing data in other databases

You can access databases on multiple database servers, or even on the same server, using the Adaptive Server Anywhere Remote Data Access features. The application is still connected to a single database as in the architecture diagrams above, but by defining remote servers, you can use proxy tables that exist on the remote database as if they were in the database to which you are connected.

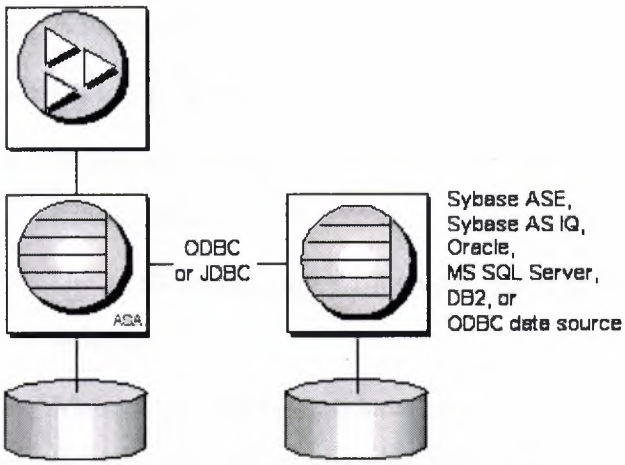


Fig. 2.1 Relation Diagram

2.1.4dBASE IV Table Specification

The dBASE IV table format was introduced in dBASE IV for DOS. Following are the specifications for dBASE IV tables:

- 2GB file size.
- Two billion records per file.
- A maximum of 255 fields per record.

Maintained indexes can have up to 47 indexes per file. Each index can be created using field expressions of virtually any combination, including conditional expressions of up to

255 characters per expression that result in an index of up to 100 bytes. Unlimited nonmaintained indexes can be stored on disk. You can use up to 47 of them simultaneously.

2.1.5 dBase V Table Specifications

The dBASE V table format was introduced in dBASE V for Windows. Following are the specifications for dBASEA V tables.

- Up to one billion records per file.
- A maximum of 1,024 fields per record.
- Up to 32,767 bytes per record.
- Unlimited nonmaintained indexes can be stored on disk. You can use up to 47 of them simultaneously.
- Up to 10 master index files open per database. Each master index can have up to 47 indexes.
- Maintained indexes can have up to 47 indexes per file. Each index can be created using field expressions of virtually any combination, including conditional expressions of up to 255 characters per expression that result in an index of up to 100 bytes.

2.1.6 dBASE Field Types

Character (C)

dBASE III+, IV, and V field type that can contain up to 254 characters (including blank spaces). This field is similar to the Paradox Alpha field type.

Date (D)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

Float (F)dBASE IV, and V floating-point numeric field type provides up to 20 significant digits.

Logical (L)

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

Memo (M)

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

OLE (O)

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

Number (N)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

dBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format. Use number fields when you need to perform precise calculations on the field data. Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

Short (S)

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal)

2.2Paradox Standard Table Specifications-

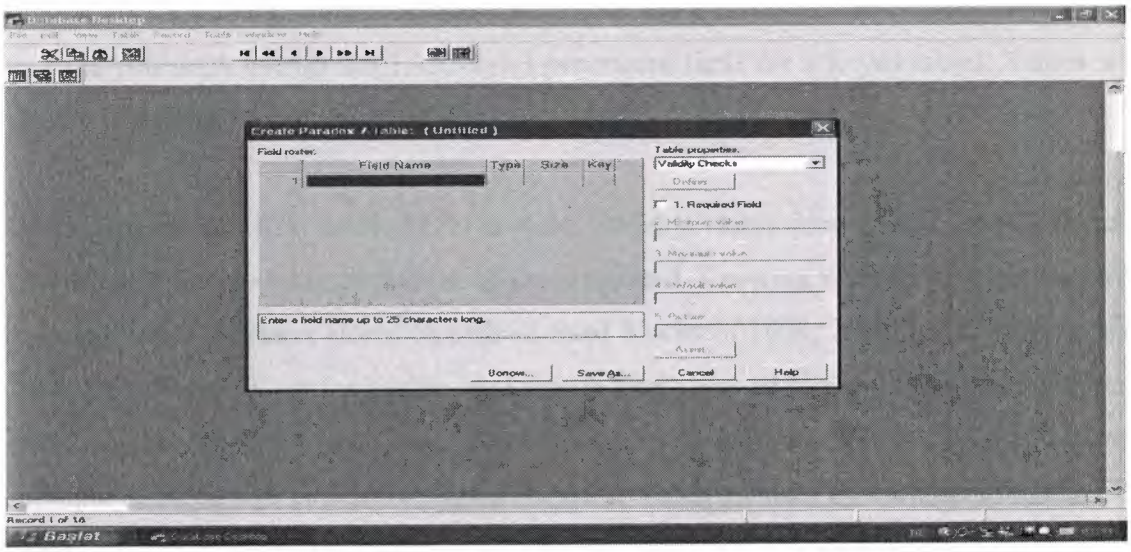


Fig. 2.2 Paradox Standart Table

2.2.1 Paradox 4 table structure.

The Paradox standard table format was introduced in Paradox for DOS version 4. Other products that use the standard format include Paradox for DOS version 4.5, ObjectVision 2.1, and Paradox for Windows versions 1.0 and 4.5.

Earlier versions of the Paradox table type are referred to as the Compatible table type. In the BDE Configuration Utility, the level option for the Paradox driver dictates what default table type is created by Paradox for Windows. Use 3 for Compatible tables, 4 for Standard tables (the default). Following are the specifications for standard Paradox tables:

- 256MB file size limit if the table is in Paradox format and using a 4K block size.
- Up to 255 fields per record.
- Up to 64 validity checks per table.
- A primary index can have up to 16 fields.
- Tables can have up to 127 secondary indexes.

- Up to two billion records per file.

Because of the 256MB file size limit and other factors such as block size, however, the limit is much smaller. Tables of 190,000 records are easily achievable (and you can have more if you don't use up the 1,350-bytes-per-record limit for a keyed table). Tables with close to a million records are common.

Block size can be 1024, 2048, 3072, or 4096. Paradox stores data in fixed records. Even if part or all of the record is empty, the space is claimed. Knowing the interworkings can save you disk space. Paradox stores records in fixed blocks of 1024, 2048, 3072, 4096 in size. After a block size is set for a table, that size is fixed, and all blocks in the table will be of that size. To conserve disk space, you want to try to get your record size as close to a multiple of block size as possible (minus 6 bytes, which are used by Paradox to manage the table).

Record size. 1,350 for keyed tables and 4,000 for unkeyed tables. When figuring out the size (the number of bytes or characters) of a table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.
Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.

2.2.2Paradox 5 Table Specifications

The Paradox 5 table format was introduced in Paradox for Windows version 5.
Following are the specifications for Paradox 5 tables:

- Up to two billion records per file.
- File size is limited to two gigabytes.
- Up to 255 fields per record.

Record size: Up to 10,800 bytes per record for indexed tables and 32,750 bytes per record for nonindexed tables. When figuring out the size (the number of bytes or characters) of a table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.

Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.

Up to 64 validity checks per table for Paradox for Windows tables.

A primary index can have up to 16 fields.

Tables can have up to 127 secondary indexes.

Block size can be from 1K to 32K in steps of 1K. For example, 1024, 2048, 3072, 4096, 5120...32768.

2.2.3Paradox 7 and Above Table Specifications

The Paradox 7 table format was introduced in Paradox version 7 for Windows 95/NT. The Paradox 7 table format has all the same specifications as the Paradox 5 table format with two additions. Following are the specification additions for the Paradox 7 table format:

- Added descending secondary indexes.
- Added unique secondary indexes

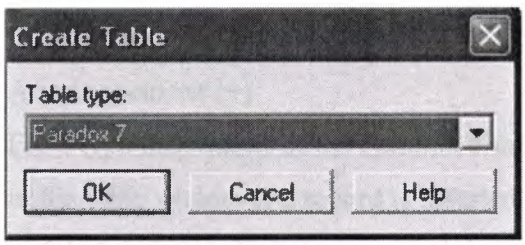


Fig. 2.3 Paradox Create Table

2.2.2.1Paradox Field Types

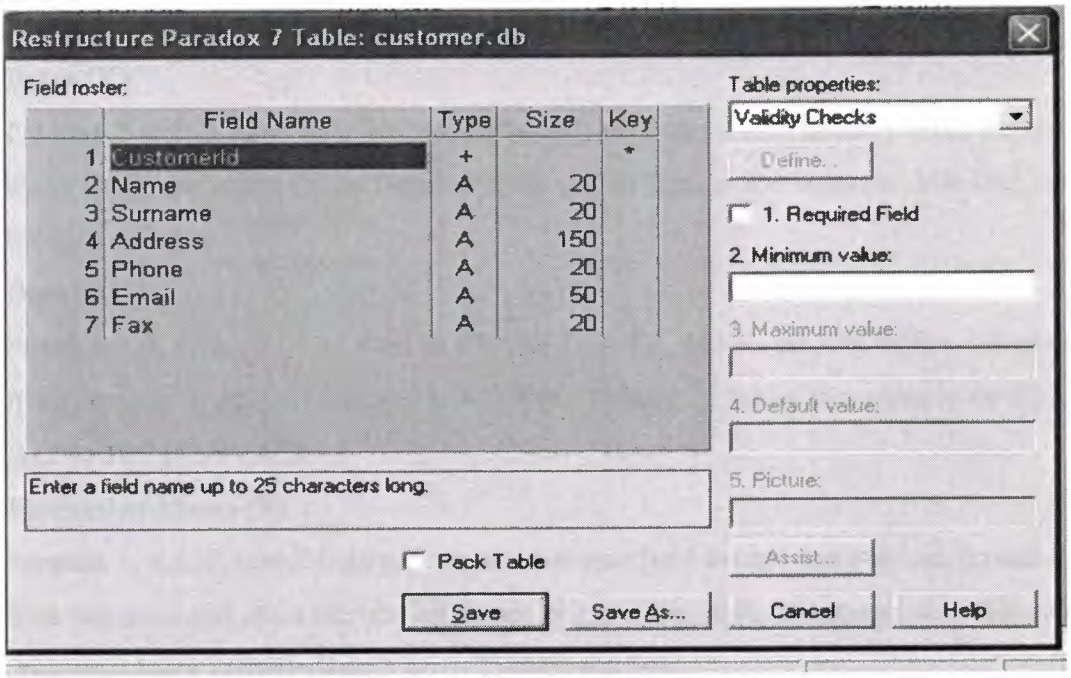


Fig. 2.4 Paradox fields type

Alpha (A)

Paradox 3.5, 4, 5, and 7 field type that can contain up to 255 letters and numbers. This field type was called Alphanumeric in versions of Paradox before version 5. It is similar to the Character field type in dBASE.

Autoincrement (+)

Field type introduced in the Paradox 5 table format that adds one to the highest number in the table whenever a record is inserted. The starting range can from -2,147,483,647 to 2,147,483,647. Deleting a record does not change the field values of other records.

BCD (#)

Paradox 5 and 7 field type which is provided only for compatibility with other applications that use BCD data. Paradox correctly interprets BCD data from other applications that use the BCD type. When Paradox performs calculations on BCD data, it converts the data to

the numeric float type, then converts the result back to BCD. When this field type is fully supported, it will support up to 32 significant digits.

Binary (B)

Paradox 1, 5, and 7 field type that can store binary data up to 256MB per field.

Bytes (Y)

Paradox 5 and 7 field type for storing binary data up to 255 bytes. Unlike binary fields, bytes fields are stored in the Paradox table (rather than in the separate .MB file), allowing for faster access.

Date (D)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

Formatted Memo (F)

Paradox 1, 4.5, 5, and 7 field type is like a memo field except that you can format the text. You can alter and store the text attributes of typeface, style, color, and size. This rich text document has a variable-length up to 256MB per field.

Graphic (G)

Paradox 1, 5, and 7 field type can contain pictures in .BMP (up to 24 bit), .TIF (up to 256 color), .GIF (up to 256 color), .PCX, and .EPS file formats. Not all graphic variations are available. For example, currently you cannot store a 24-bit .TIF graphic. When you paste a graphic into a graphic field, Paradox converts the graphic into the .BMP format.

Logical (L)

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

Memo (M)

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

For Paradox tables, the file is divided into blocks of 512 characters. Each block is referenced by a sequential number, beginning at zero. Block 0 begins with a 4-byte number in hexadecimal format, in which the least significant byte comes first. This number

specifies the number of the next available block. It is, in effect, a pointer to the end of the memo file. The remainder of Block 0 isn't used.

Money ($)

Paradox 3.5, 4, 5, and 7 field type, like number fields, can contain only numbers. They can hold positive or negative values. Paradox recognizes up to six decimal places when performing internal calculations on money fields. This field type was called Currency in previous versions of Paradox.

OLE (O)

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

Number (N)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

dBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format. Use number fields when you need to perform precise calculations on the field data. Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

Short (S)

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal).

Time (T)

Paradox 5 and 7 field type that can contain time times of day, stored in milliseconds since midnight and limited to 24 hours.This field type does not store duration which is the difference between two times. For example, if you need to store the duration of a song, use an Alpha field. Whenever you need to store time, make a distinction between clock time and duration. The Time field type is perfect for clock time. Duration can be stored in an Alpha field and manipulated with code.

TimeStamp (@)

Paradox 5 field type comprised of both date and time values. Rules for this field type are the same as those for date fields and time fields.

CHAPTER 3

MAİN FORMS OF THE APLICATION PROGRAM

3.1 Database Design of The Program

The stock database consists of seven tables those are stock list table, company table, customer table, personel table, order table, sale table, login table.

Stock List table contains nine fields :

- Stock Id , Stock Code , Stock Name , Purchase Price , Selling Price
- Company , Size , Color , Quantity , Date , Guarantee

Company table contains six fields :

- Company Id , Name , Phone
- Fax , Address , Web

Customer table contains seven fields :

- Customer Id , Name , Surname , Fax
- Phone , E-mail , Address

Customer table contains ten fields :

- Personel No , First Name , Last Name , Duty , Department
- Salary , Address , Phone , E-mail , Hire Date

Sale table contains six fields :

- Sale Id , Customer Id , Product Id
- Date , Price , Quantity

Order table contains eight fields :

- Order Id , Order Name , Properties , Quantity ,
- Company , Price Arrival Date , Order Date

Login table contains threefields :

- User Id , User Name , User Password

3.2 The relationships between tables will as follows:

In Company Table Company Id field is a primary key.

In Order Table Order Id is a primary key.

In Stock List Table Stock Id is a primary key.

In Customer Table Customer Id is a primary key.

In Sale Table Sale Id is a primary key.

In Login Table User Id is a primary key.

In Personel No is a primary key.

3.3Exequtions of the program

When you execute the program Login Form opens, then it will ask you username and password. You can see it in Fig 3.1.

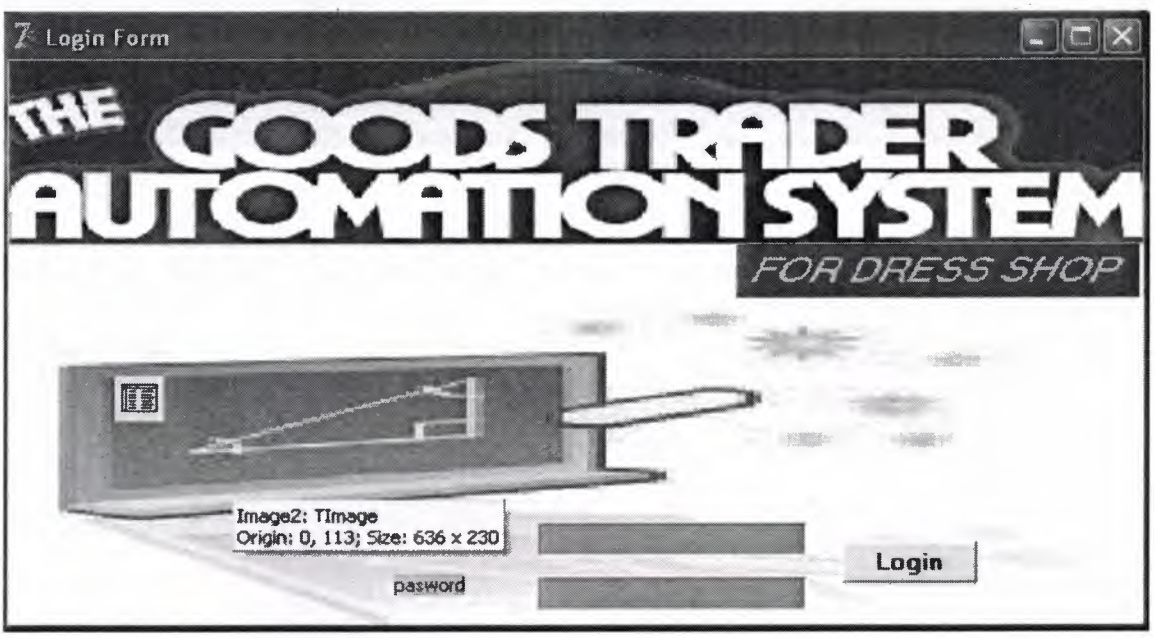


Fig 3.1 Login Form

If you do not know username and password you can not login this program. In fig3.1 write username and password then Click Login Button or Press Enter Key to login this program.

In this program there is Product , Customer, Worker, Order , Firm , Report , Users , About and Exit menus (shown in Fig 3.2)

Fig 3.2 Main Form

In Stock Menu there are two submenus. These are Stock Entry Submenu and product List Submenu.

Fig 3.3 Product Entry Submenu

When you select Product Entry Submenu then Stock Entry form is appeared.

Fig 3.4 Product Entry Form

In this form we can add new product to the database then it is shown in the Fig 3.4 on the form. And also we can delete the product which was added before after select on the table. Moreover we can delete the product which we have added before. And when we press the clean button it clears all the texts. Also we can go next and prior by pressing next and previous button of the navigator. If the product exists in the stock, then we can select "Product in the stock" radio button and select in the table as well, then we can update its details as well.

When you select Stock List Submenu then Stock List form is appeared.

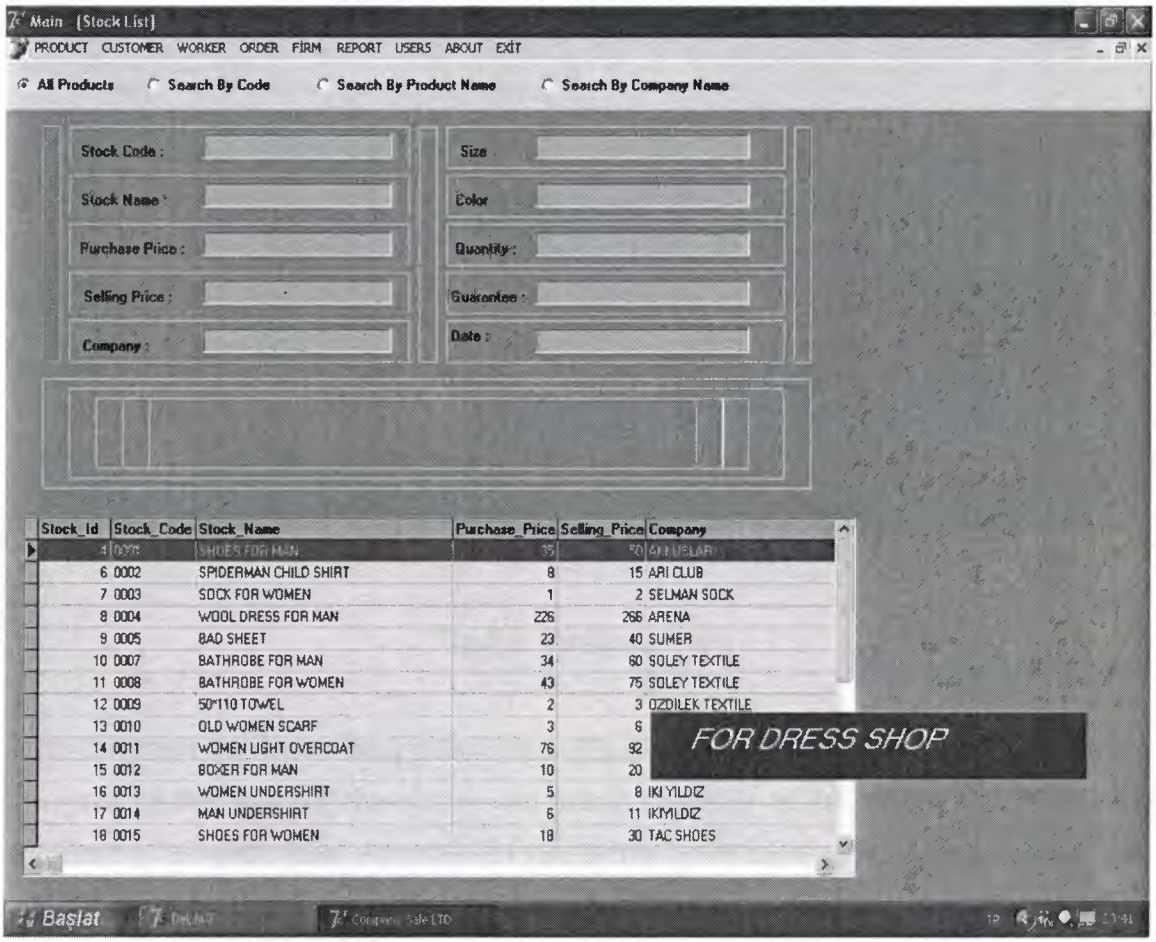


Fig 3.5 Stock List

In this form we can see all the product details by Stock_Id , Stock_Code , Stock_Name , Purchase_Price , Selling_Price , Company ,Size,Color,Quantity, Guarantee and Date . And we can search the product by Company Name , search by Product Name and search by Product Code as well.

Fig 3.6 Customer Form

Here Customer details are held in this form. We can add customer, delete customer, update customer in this form. Moreover we can search customer by Id and by First Name and Surname in the same form. Also there is sale button in the same form which are Retail Sale and Whole Sale. When you select one customer then click one of the sale button the sale form appears. To sell retail product you should click the "Retail Sale" button and to sell

wholesale product should click the "Wholesale" button. After that there will appear sale form to do sale process.

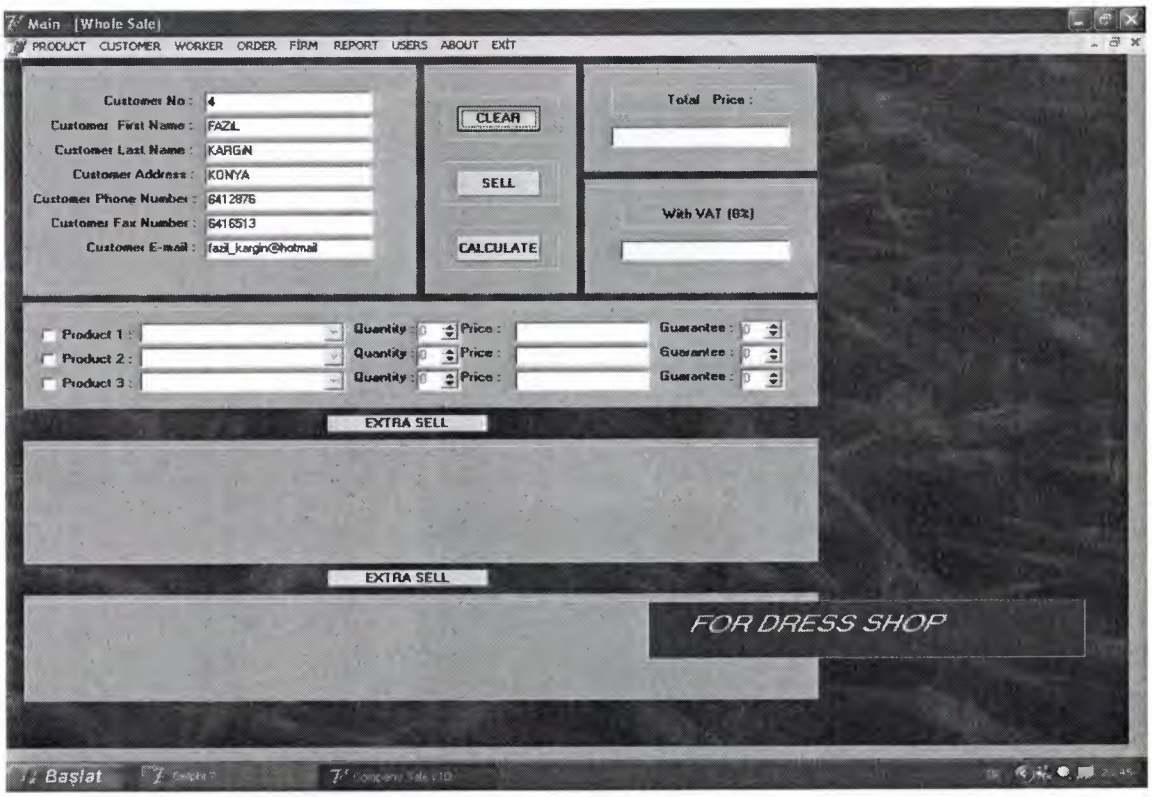


Fig 3.7 Whole Sale Form

In Wholesale form at the first clicked checkboxes then Comboboxes, Quantities, Prices, Guarantees will be activated. Then we can select the products in the Comboboxes and determine the quantity. Then when it is clicked the "CALCULATE" button "Total Price" and "With VAT (8%)" are calculated and displayed on the form. When you press the "SELL" button you will see such as a "Please Confirm Sale" message:

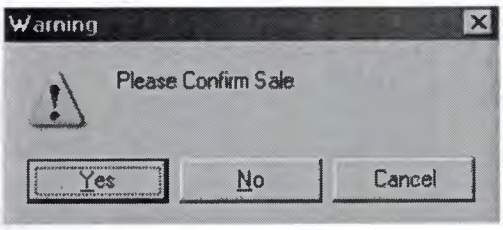


Fig 3.8 Confirm Sale Message

If you press "No" button the sale is cancelled and, If you press "Yes" button you will get such as a "Product Sold Successfully" message. Then the sale process is added to the sale database table.

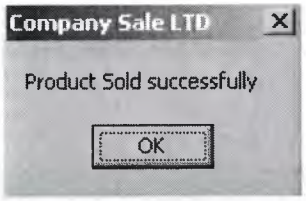


Fig 3.9 Sale is Successful Message

If there is not enough quantity in the stock you will get "Not Enough Product Quantity" message.

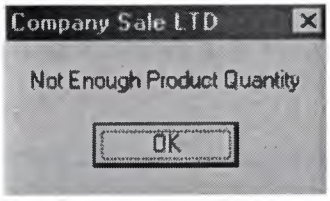


Fig 3.10 Quantity Message

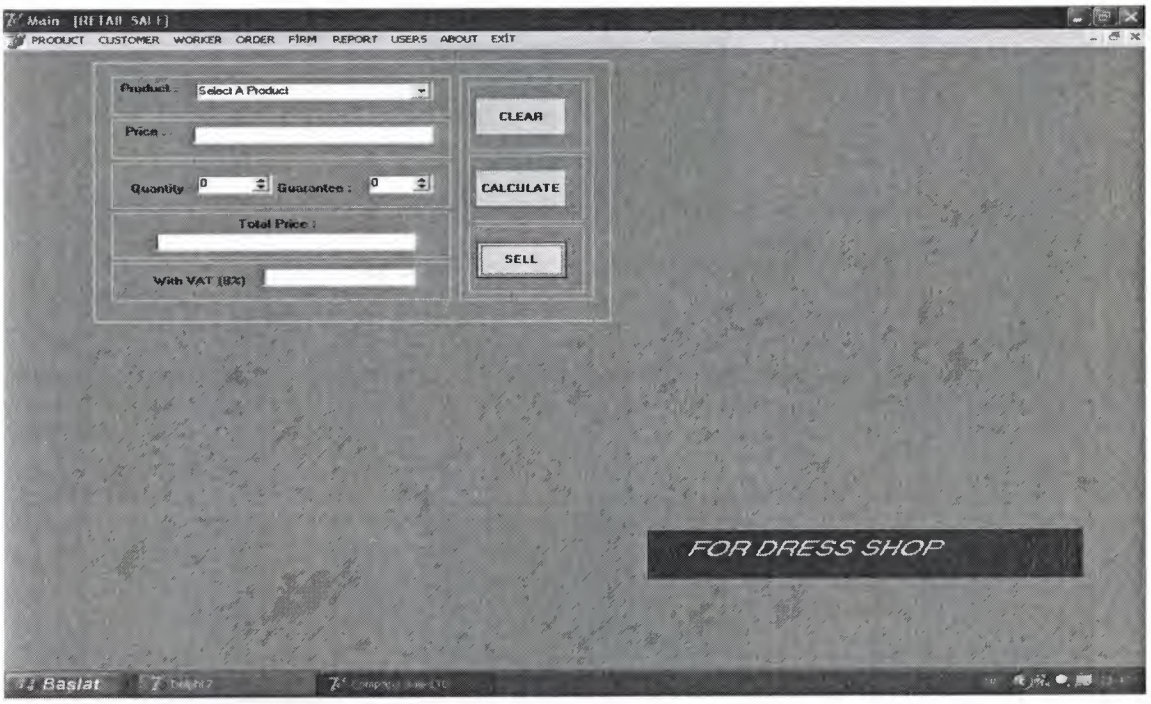


Fig 3.11 Retail Form

In retail sale form the customer select one of the products which is shown in ComboBox, then customer determines the quantity. After that the price and guarantee are appeared automatically. Then when it is clicked the "Calculate" button then "Total Price" and "With VAT (8%)" are calculated and displayed on the form. When you press the "SELL" button you will see such as a "Please Confirm Sale" message:

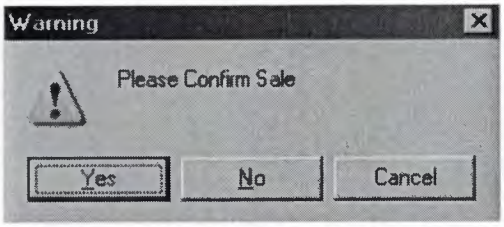


Fig 3.12 Confirm Sale Message

If you press "No" or "Cancel" button the sale is cancelled and, If you press "Yes" button you will get such as a "Product Sold Successfully" message. And then sale process is added to the sale database table.

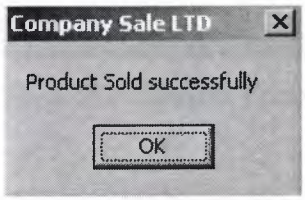


Fig 3.13 Sale is Successful Message

If there is not enough quantity in the stock you will get message "Not Enough Product Quantity" .

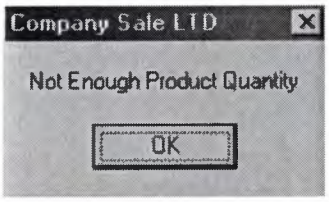


Fig 3.14 Warning About Quantity

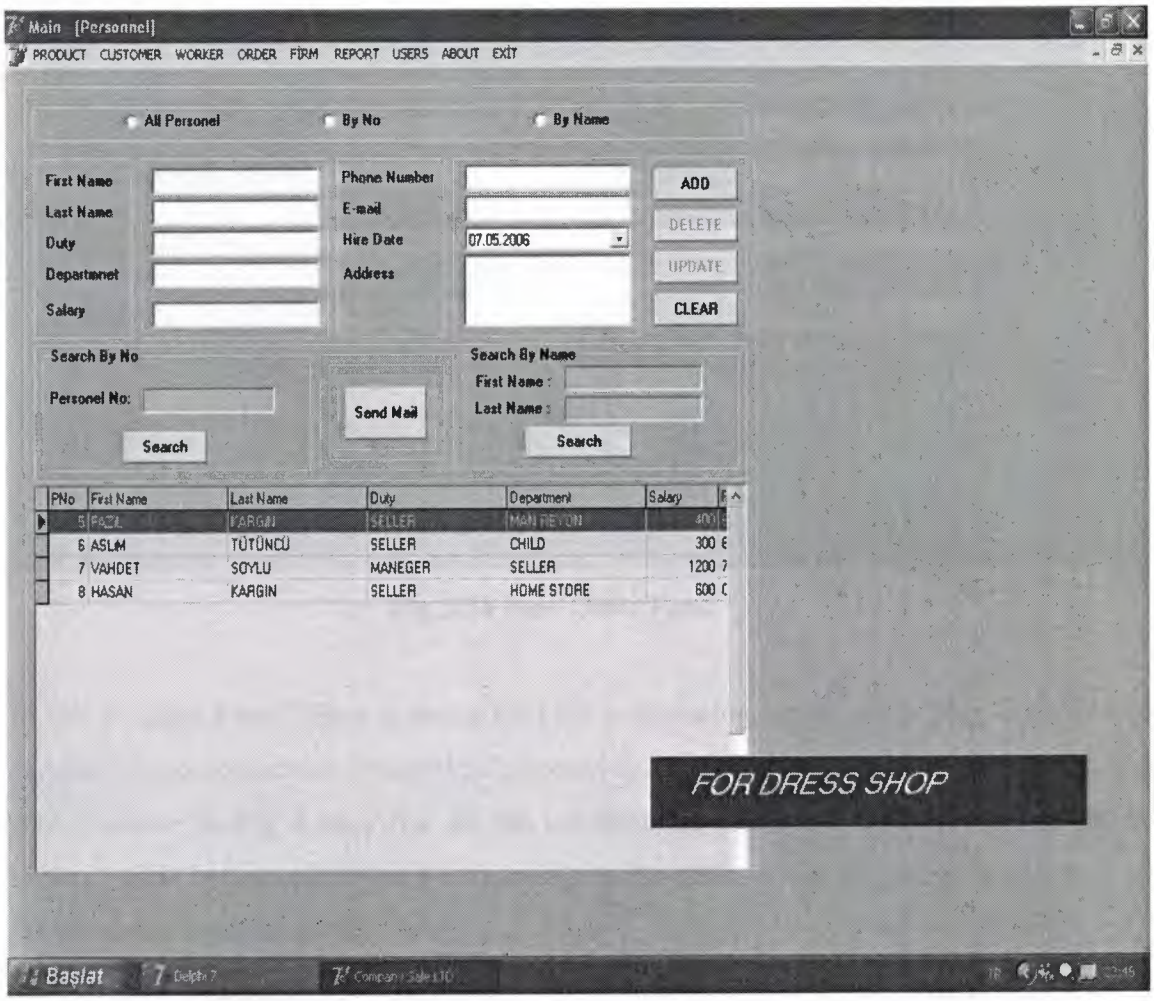


Fig 3.15 Personnel Form

Here Personnel details are held in this form. We can add personel, delete personel, update personel in this form. Moreover we can search personel by No and by First Name and Surname in the same form in Fig 3.7. And the personnel can send e-mail to each other by selecting a person in the table.

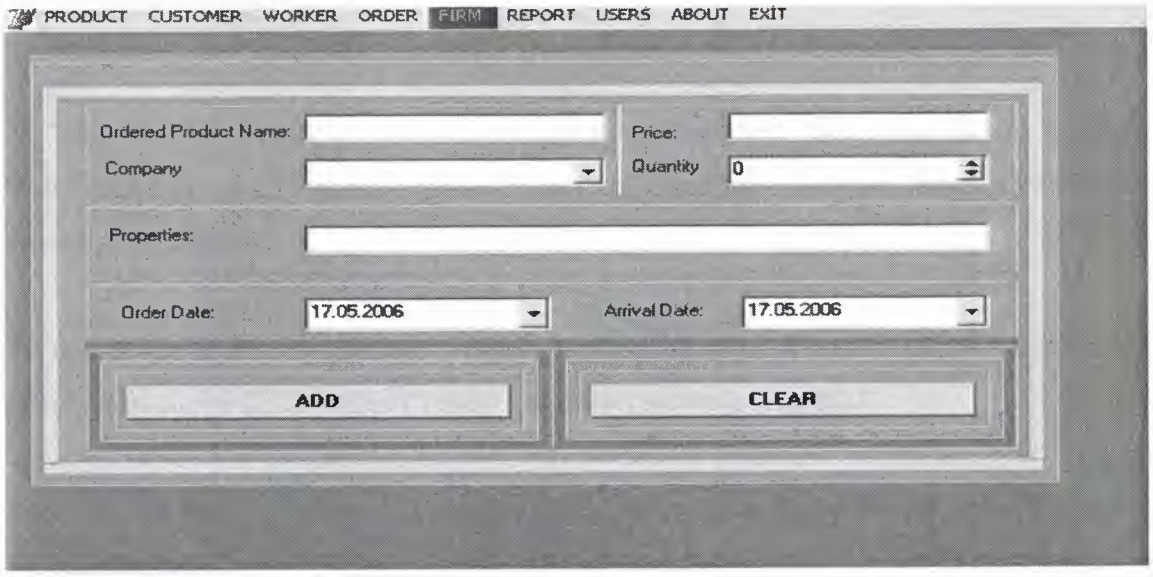


Fig 3.16 New Order Form

In "New Order Form" there is being kept the information about order. They are Ordered Product Name, Properties, Quantity, Company, Price, Order Date and Arrival Date as it is shown above **in Fig 3.14.** After all the informations filled in the form then should be pressed ADD button to add these information to the table. When you press clear button all the boxes are cleared as well.

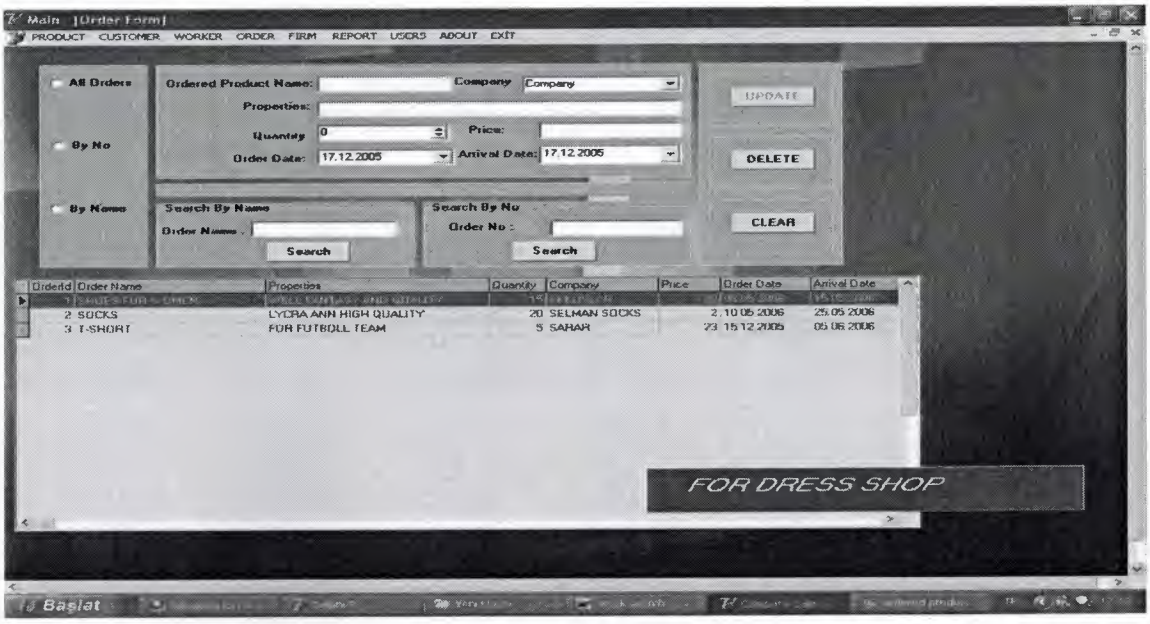


Fig 3.17 List of Order Form

In "List of Order Form" we can see all the ordered product information in the database table. Order Id, Order Name, Product Properties, Quantity, Company, Price, Order Date, Arrival Date fidelds.

To update the order information, we have to press the update button after we correct the information in the EditBox and ComboBox shown above **in Fig. 3.17.**

To delete the order, we have to press the delete button after we select the order product in the databae table.

To Search a product. We can search by Order No and search by Order Name after it is written Order No or Order Name.

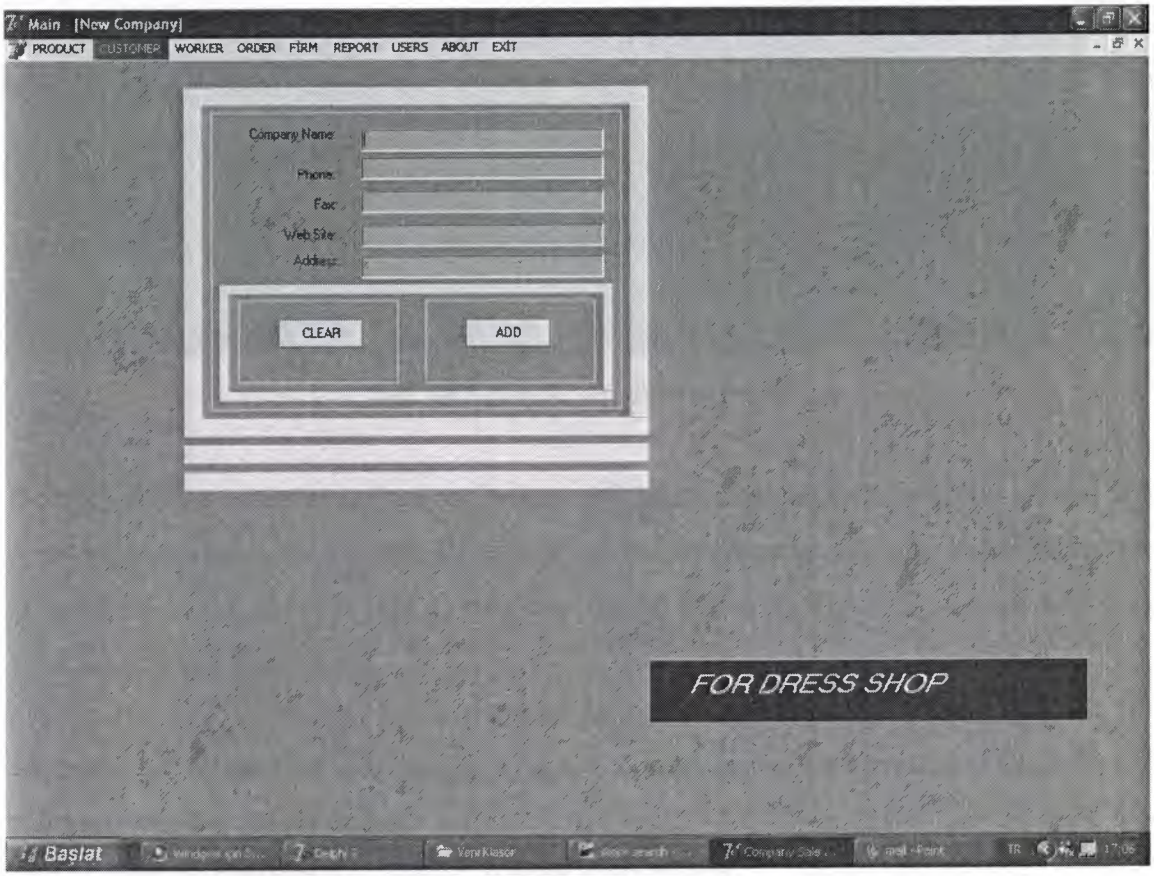


Fig 3.18 New Company Form

In "New Company Form " we can add new company to the database by pressing add button, after all the information are filled, from which company we are buying the product. And to clear the EditBox it is needed to press clear button.

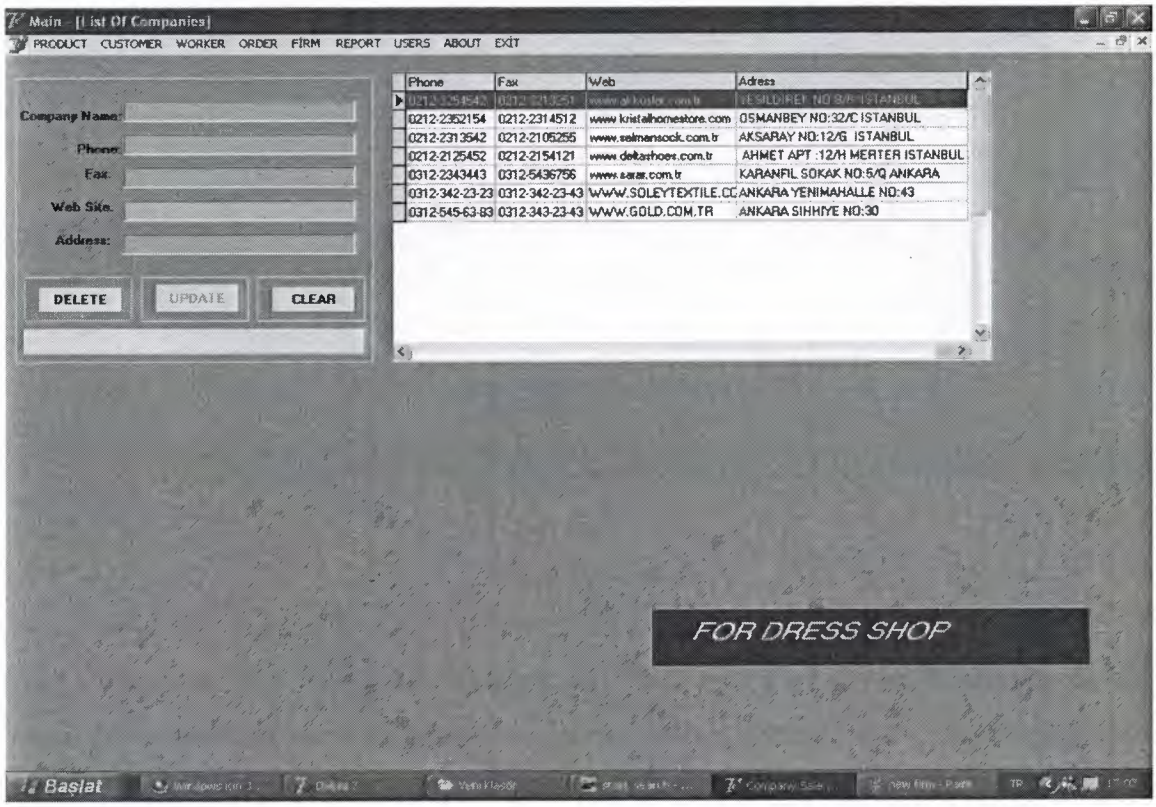


Fig 3.19 List of Companies Form

In "List of Companies Form " we can update and delete the companies which is needed. To update the company information, first we select the company on the table then correct the information in the EditBox and ComboBox shown above **in Fig 3.19.** above, then it is needed to press update button.

To delete the order, first it is needed to select a company in the database table then it is necessary to press **delete** button.
And to clear the EditBox it is needed to press **clear** button.

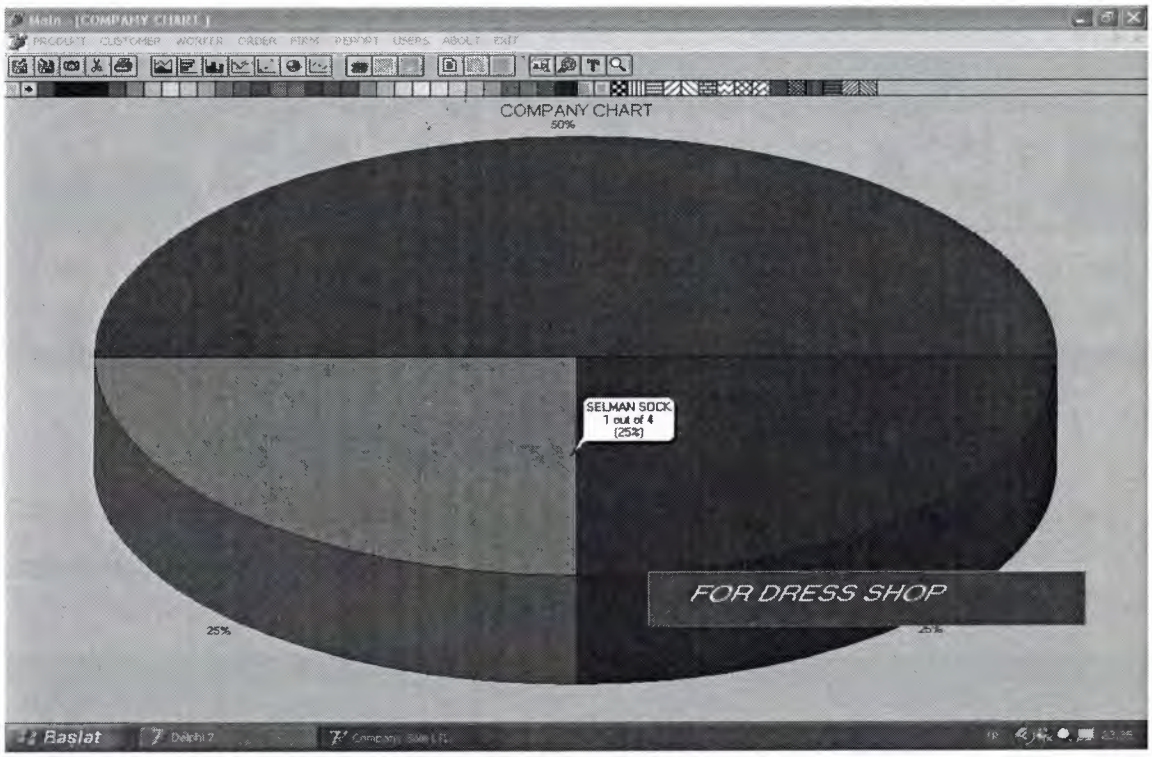


Fig 3.20 Chart

Company chart includes number of work done with the companies in a percentage value. The company names are colored of chartfx control. Above there are toolbox, palette bar, pattern bar which give user to customize the chart by changing the color of each slices of the chart which is in pie type chart control.

And also the chart type, 3d vision, show/hide list can be changed from the toolbar. Furthermore toolbar allows users to save, import, export or print the chart by clicking the proper buttons on the toolbar.

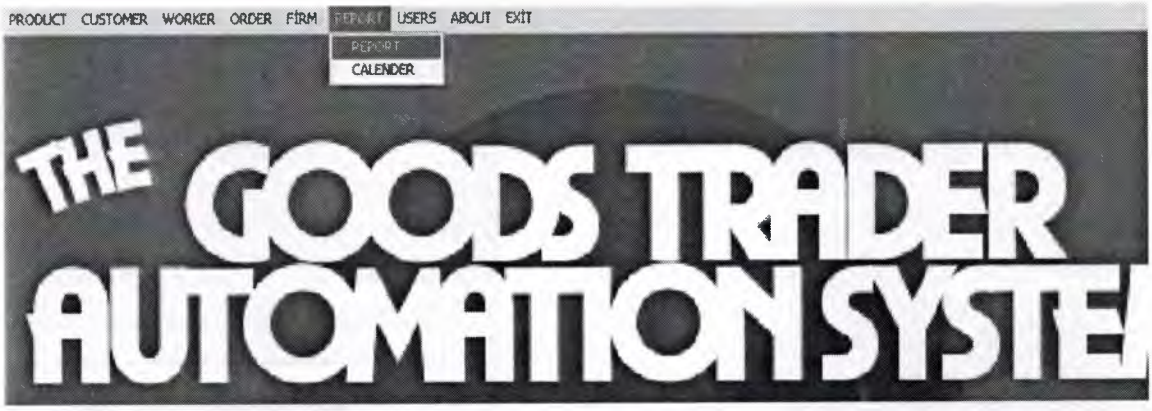


Fig 3.21 Select Report Form

When we press **report** button, then displays product name, date, price, quantity information as it is shown fig. below.

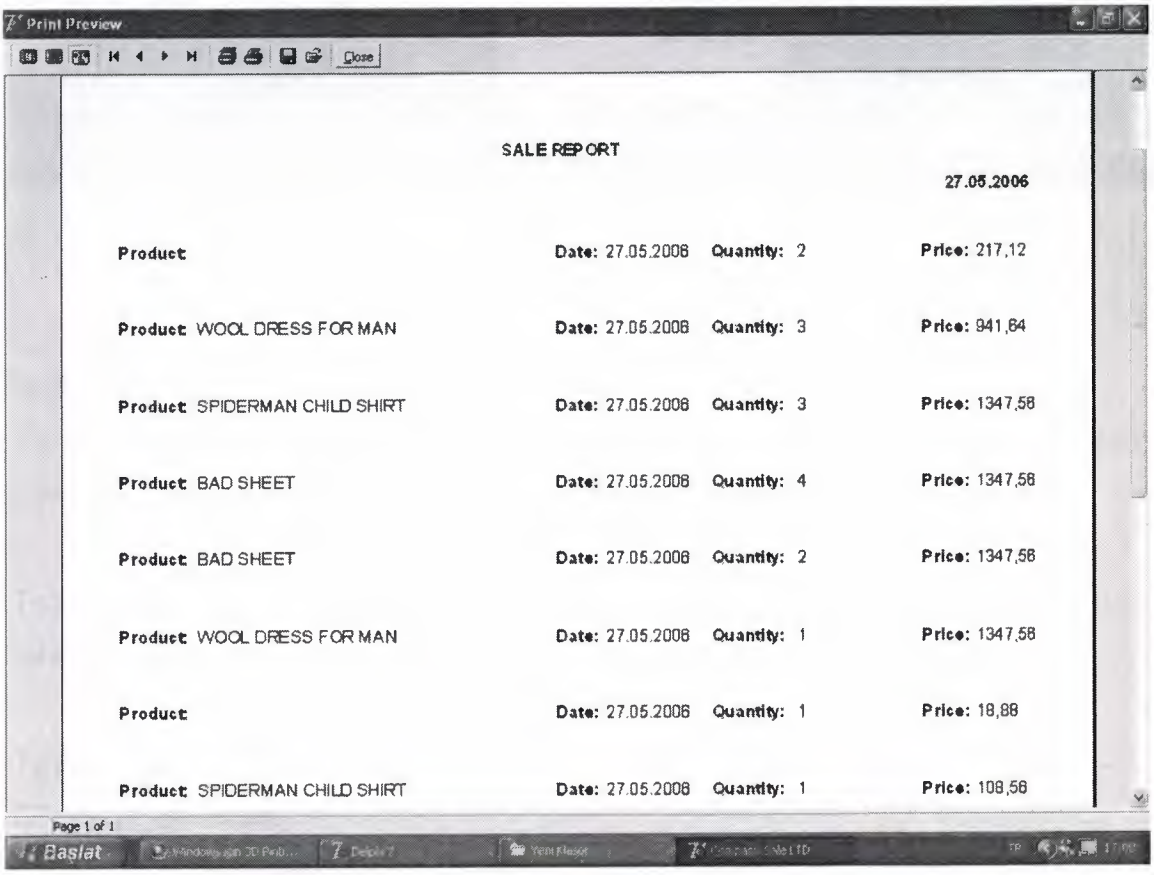


Fig 3. 22 Report

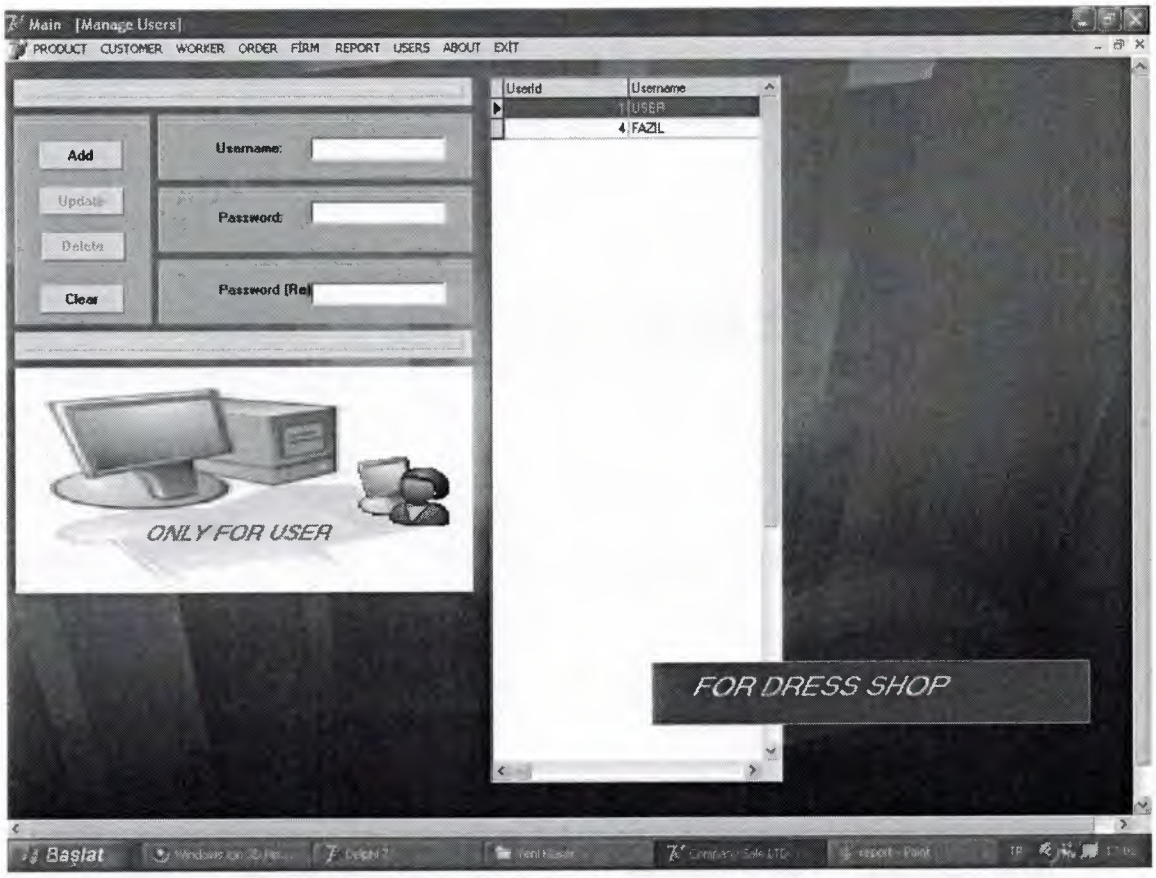


Fig 3.23 Manage User Form

In "Manage User Form" we can add user, update user and delete user.

To add the user to the database table, first it is needed to filled the EditBoxes those are Username, Password, Password(Re), then it is necessary to press add button.

To update the user, first we select the user in the table, then correct the information in the EditBox shown above in Fig. above, then it is needed to press update button.

To delete user, first it is needed to select a user in the database table, then it is necessary to press delete button.

And to clear the EditBoxes it is needed to press clear button.

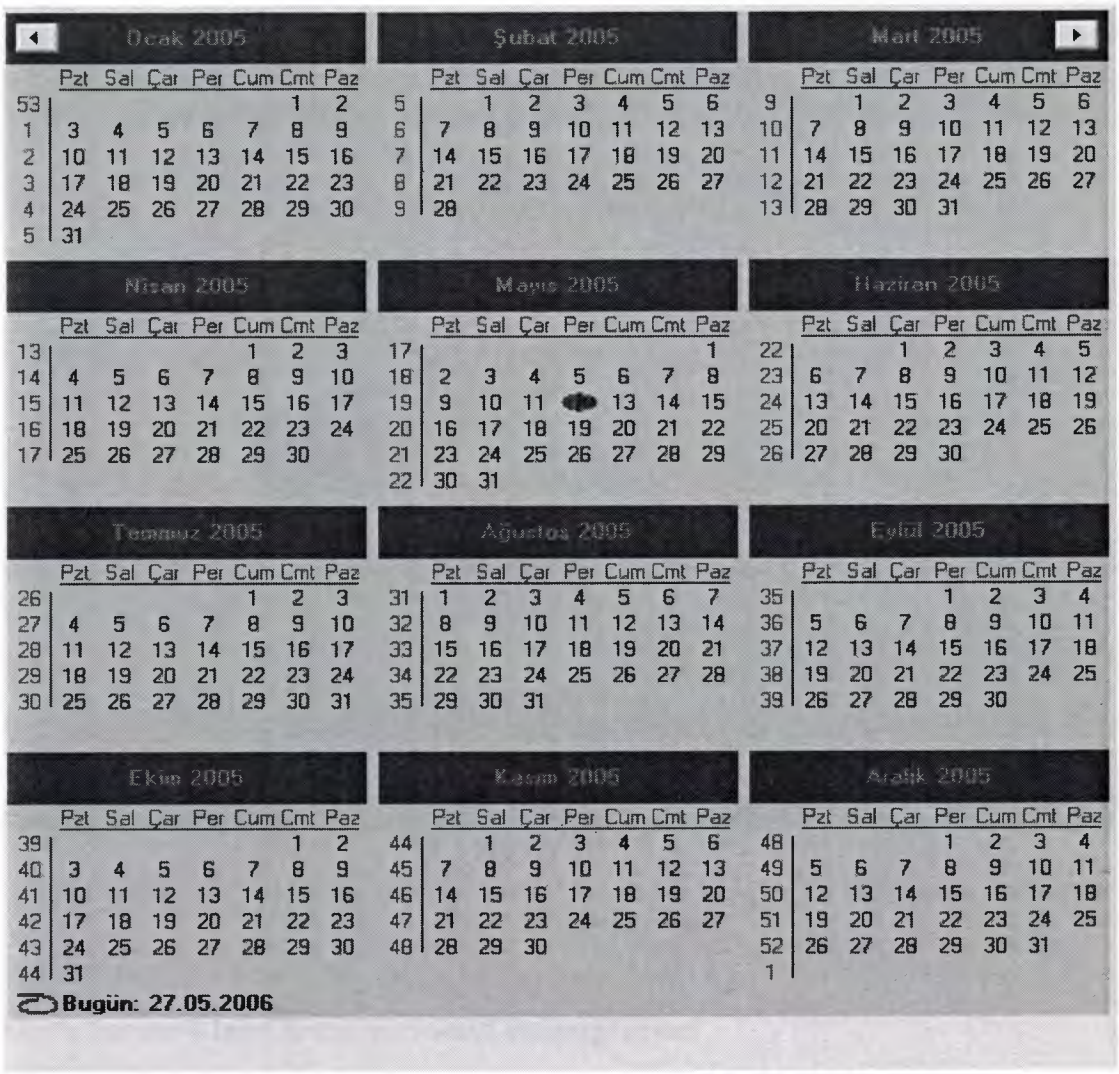


Fig 3.24 Calendar Form

In Calendar Form there is shown the Calendar as we see above Fig 3.19. We can see the date or specific date which we want to learn.

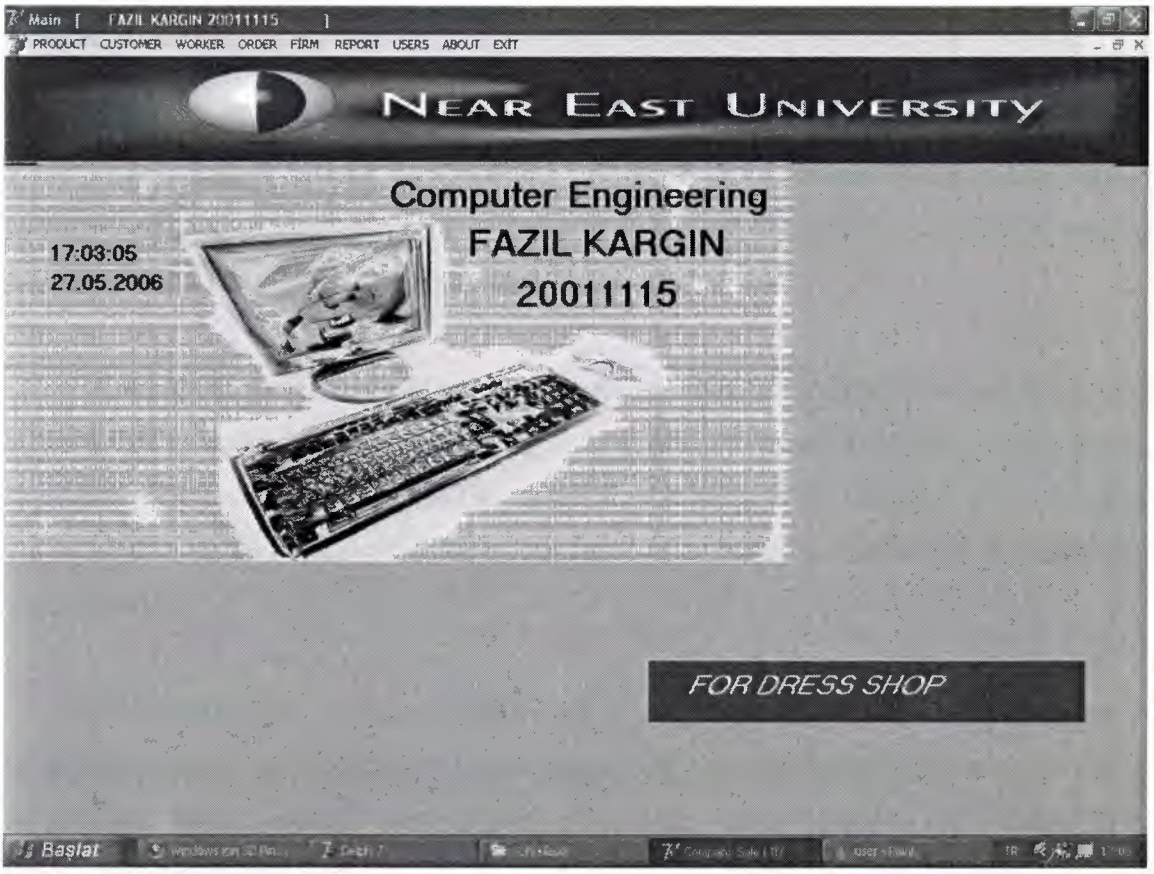


Fig 3.25 About Form

Gives the user a brief description about the programmer.

Conclusion

After making so many researches about Delphi Programming language and investigating through internet to make this project, I learned many things about Delphi, because I obliged to finish my project and everything had to be done by myself alone. So making practical things is much better than learning it literary.

During the project I faced so many problems they were difficult for me because it was my first program, later after practicing and learning from books it became easier by the time and I used to know how to use Delphi and how to write codes. So the first 2 or 3 forms where hard to organize and write codes, but later other forms become easier in design and writing codes.

In the future other options could be add to the program, it can be updated according to the need, also it can be connected to the internet, at that time sales could be done on net, for instance when someone wants to buy something, he/she looks to the internet first, investigates about that item, its image, size, color and price, also the payment facilities could be shown to the customer, so if the customer likes what he/she wants to buy, he orders through internet and the workplace provides that item for him in the limited time.

REFERENCES

Reference to Book:

[1] Turkmen Kitabevi, Borland Delphi 7

[2] Quick Star Publishing Borland Delphi 7

Reference to Electronic-Book:

[1] Delphi 7 Delphi Handle software .PDF

[2] Delphi 7 Paradox star Delphi .PDF

Reference to Electronic Source- Online source from Web:

[1]J..Micky DOFNARD ''http://www.vegasoftware.com''

[2]Jimmy,T.F ''http://www. borland.com''

APPENDIX 1: PROGRAM CODES

LOGIN FORM

```
unit Unit21;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls, DB, DBTables;
type
  TLoginF = class(TForm)
    Uname: TEdit;
    Upass: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    LOGIN: TTable;
    Panel1: TPanel;
    Label3: TLabel;
    Label4: TLabel;
    Image2: TImage;
    Image1: TImage;
    Label5: TLabel;
    Label6: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure UpassKeyPress(Sender: TObject; var Key: Char);
    procedure UnameKeyPress(Sender: TObject; var Key: Char);
    procedure FormShow(Sender: TObject);
    procedure Image1Click(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;
var
  LoginF: TLoginF;
implementation
uses Unit1;
{$R *.dfm}
procedure TLoginF.Button1Click(Sender: TObject);
begin
//open login table
login.Open;
//find user by username and password
login.Filter:='Uname='+quotedstr(Uname.text);
login.Filtered:=true;
login.Filter:='UPass='+quotedstr(Upass.Text);
login.Filtered:=true;
//if no user found give message
if login.RecordCount =0 then
 showmessage('Wrong Username Or Password')
 else
  begin
//show form1
 form1.AlphaBlend :=false;;
 LoginF.AlphaBlend :=true;
 form1.SetFocus;
 end;
end;
procedure TLoginF.FormClose(Sender: TObject; var Action: TCloseAction);
begin
```

```
//close application
application.Terminate;
end;
procedure TLoginF.UpassKeyPress(Sender: TObject; var Key: Char);
begin
//ENTER key is pressed call button1 click which is login
if key=#13 then
 button1.Click;
end;
procedure TLoginF.UnameKeyPress(Sender: TObject; var Key: Char);
begin
//ENTER key is pressed call button1 click which is login
if key=#13 then
 button1.click;
end;
procedure TLoginF.FormShow(Sender: TObject);
begin
loginf.setfocus;
end;
procedure TLoginF.Image1Click(Sender: TObject);
begin
uname.SetFocus;
end;
end.
```

FORM 2. PRODUCT ENTRY FORM

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```

```
  Dialogs, StdCtrls, Mask, DBCtrls, Spin, ComCtrls, ExtCtrls, Grids,
  DBGrids, DB, DBTables, jpeg;
type
  TForm2 = class(TForm)
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Table1: TTable;
    compT: TTable;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    GroupBox2: TGroupBox;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    GroupBox3: TGroupBox;
    code: TEdit;
    Sname: TEdit;
    price: TEdit;
    Sprice: TEdit;
    company: TEdit;
    GroupBox4: TGroupBox;
    quantity: TSpinEdit;
    guarantee: TSpinEdit;
    size: TEdit;
    color: TEdit;
```

```
date1: TDateTimePicker;
GroupBox5: TGroupBox;
Add: TButton;
Clean: TButton;
Delete: TButton;
Update: TButton;
GroupBox6: TGroupBox;
Label13: TLabel;
Label14: TLabel;
ComboBox1: TComboBox;
comboBox2: TComboBox;
GroupBox7: TGroupBox;
RadioButton1: TRadioButton;
RadioButton2: TRadioButton;
GroupBox12: TGroupBox;
DBNavigator1: TDBNavigator;
Image1: TImage;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormActivate(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure RadioButton1Click(Sender: TObject);
procedure AddClick(Sender: TObject);
procedure DeleteClick(Sender: TObject);
procedure CleanClick(Sender: TObject);
procedure quantityChange(Sender: TObject);
procedure guaranteeChange(Sender: TObject);
procedure ComboBox2Change(Sender: TObject);
procedure DBGrid1CellClick(Column: TColumn);
procedure UpdateClick(Sender: TObject);
procedure ComboBox1Change(Sender: TObject);
procedure FormCreate(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form2: TForm2;
implementation
uses unit3;
{$R *.dfm}
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
//Close Table1 on form close
  table1.Close;
  action:=caFree;
end;
procedure TForm2.FormActivate(Sender: TObject);
begin
//On form active make the window maximize
WindowState := wsMaximized;
end;
procedure TForm2.RadioButton2Click(Sender: TObject);
var     k:integer;
 i:integer;
begin
//Clear all fields on form
  clean.Click;
  Combobox1.Clear;
  Combobox2.Clear;
  Combobox1.Enabled := True;
  Combobox2.Enabled := True;
```

```
  //List All Products in Combobox1 and Combobox2
  table1.First;
  for i:=0 to Table1.RecordCount-1 do
  begin
  combobox1.Items.Add(table1.FieldValues['Stock_Name']);
  combobox2.Items.Add(table1.FieldValues['Stock_Id']);
  table1.Next;
  end;
end;
procedure TForm2.RadioButton1Click(Sender: TObject);
begin
//combobox1 and combobox2 will be disabled
 Combobox1.Enabled := False;
 Combobox2.Enabled := False;
 //Clear all fields on form
 clean.Click;
end;
procedure TForm2.AddClick(Sender: TObject);
begin
// Add if New Product is selected
if RadioButton1.Checked=true then
 begin
//insert mode to add new record to table
   Table1.insert;
   Table1.FieldValues['Stock_Code']:=code.Text;
   Table1.FieldValues['Stock_Name']:=Sname.Text;
   Table1.FieldValues['Purchase_Price']:=strtofloat(price.Text);
   Table1.FieldValues['Selling_Price']:=strtofloat(Sprice.Text);
   Table1.FieldValues['Company']:=Company.Text;
     Table1.FieldValues['size']:=size.Text;
       Table1.FieldValues['color']:=Color.Text;
```

```
  Table1.FieldValues['Quantity']:=strtoint(Quantity.Text);
  Table1.FieldValues['Guarantee']:=strtoint(Guarantee.Text);
  Table1.FieldValues['Date']:=DateToStr(date1.Date);
  Table1.Post;
  //Select Product in stock to list all products again
   radiobutton2.OnClick(sender);
   clean.Click;
   table1.close;
   table1.Open;
 end;
end;
procedure TForm2.DeleteClick(Sender: TObject);
begin
//Make user confirm deletion process
if Messagedlg('Please Confirm Deletion',mtwarning, [mbYes, mbNo,mbCancel],0)=6 then
  begin
  //if accepts delete record
  Table1.Delete;
  //list and refresh the product comboboxes
  radiobutton2.OnClick(sender);
  table1.Close;
  table1.Open;
  clean.Click;
  Showmessage('Deletion is successfull');
  end
  else
   Showmessage('Deletion is Cancelled');
end;
procedure TForm2.CleanClick(Sender: TObject);
begin
//Clear all fields on form
```

```
code.Text:='';
Sname.Text:='';
price.Text:='';
Sprice.Text:='';
Company.Text:='';
size.Text:='';
Color.Text:='';
quantity.Text:='0';
guarantee.Text:='0';
ADD.Enabled :=true;
Update.Enabled :=false;
end;
procedure TForm2.quantityChange(Sender: TObject);
begin
//if quantity is less then 0 make it 0
  if Quantity.Value<0 then
  Quantity.Value:=0;
end;
procedure TForm2.guaranteeChange(Sender: TObject);
begin
//if Guarantee is less then 0 make it 0
    if Guarantee.Value<0 then
    Guarantee.Value:=0;
end;
procedure TForm2.ComboBox2Change(Sender: TObject);
var     k,i:integer;
begin
//assign text of combobox2 to variable k
  k:=strtoint(Combobox2.Text);
  //set combobox1 index from the  variable k
  Combobox1.ItemIndex:=k-1;
```

```
 Table1.open;
  //Find Record which has k id
 table1.Locate('Stock_code',k,[loPartialKey]);
 //bring values to the fields on form
Sname.Text:=Table1.FieldValues['Stock_Name'];
Code.Text:=Table1.FieldValues['Stock_Code'];
Price.Text:=Table1.FieldValues['Purchase_Price'];
Sprice.Text:=Table1.FieldValues['Selling_Price'];
Company.Text:=Table1.FieldValues['Company'];
size.Text:=Table1.FieldValues['size'];
Color.Text:=Table1.FieldValues['Color'];
Guarantee.Text:=Table1.FieldValues['Guarantee'];
Quantity.Text:='0';
Date1.Date:=Table1.FieldValues['Date'];
end;
procedure TForm2.DBGrid1CellClick(Column: TColumn);
begin
if dbgrid1.Fields[dbgrid1.SelectedIndex].AsString<>'' then
begin
//bring values to the fields on form
Sname.Text := table1.FieldValues['Stock_Name'];
code.Text := table1.FieldValues['Stock_Code'];
Price.Text := table1.FieldValues['Purchase_Price'];
SPrice.Text := table1.FieldValues['Selling_Price'];
Company.Text := table1.FieldValues['Company'];
size.Text := table1.FieldValues['size'];
Color.Text := table1.FieldValues['color'];
Guarantee.Text := table1.FieldValues['Guarantee'];
Date1.date := table1.FieldValues['Date'];
Quantity.Text := table1.FieldValues['Quantity'];
//disable add button and enable update button
```

```
Add.Enabled :=false;
Update.Enabled :=true;
end;
end;
procedure TForm2.UpdateClick(Sender: TObject);
begin
//Edit mode to update records
table1.Edit;
Table1.FieldValues['Stock_Code'] :=code.Text ;
table1.FieldValues['Stock_Name']:= Sname.Text;
 table1.FieldValues['Purchase_Price'] :=Price.Text;
 table1.FieldValues['Selling_Price'] :=SPrice.Text;
table1.FieldValues['Company']:=Company.Text ;
table1.FieldValues['size']:=size.Text ;
table1.FieldValues['Color']:=Color.Text ;
table1.FieldValues['Guarantee'] :=Guarantee.Text;
table1.FieldValues['Date']:=Date1.date ;
table1.FieldValues['Quantity']:=Quantity.Text ;
// send values to table
table1.Post;
//close table
table1.Close;
table1.Open;
end;
procedure TForm2.ComboBox1Change(Sender: TObject);
begin
//Find the selected product and bring values to the fields on form
  table1.Locate('Stock_name',combobox1.Text,[loPartialKey]);
Sname.Text:=Table1.FieldValues['Stock_Name'];
Code.Text:=Table1.FieldValues['Stock_Code'];
Price.Text:=Table1.FieldValues['Purchase_Price'];
```

```
Sprice.Text:=Table1.FieldValues['Selling_Price'];
Company.Text:=Table1.FieldValues['Company'];
size.Text:=Table1.FieldValues['size'];
Color.Text:=Table1.FieldValues['color'];
Guarantee.Text:=Table1.FieldValues['Guarantee'];
Quantity.Text:='0';
Date1.Date:=Table1.FieldValues['Date'];
combobox2.ItemIndex := combobox2.Items.IndexOf(Table1.FieldValues['Stock_Id'] );
end;
end.
```

FORM 3. NEW ORDER FORM

```
unit Unit3;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Spin, ComCtrls, DB, DBTables, ExtCtrls;
type
  TForm3 = class(TForm)
    order: TTable;
    comp: TTable;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Price: TEdit;
    Quantity: TSpinEdit;
    Label6: TLabel;
    Label2: TLabel;
    GroupBox3: TGroupBox;
    PName: TEdit;
    Company: TComboBox;
```

```
  Label1: TLabel;
  Label4: TLabel;
  GroupBox4: TGroupBox;
  Properties: TEdit;
  Label3: TLabel;
  GroupBox5: TGroupBox;
  adate: TDateTimePicker;
  Label7: TLabel;
  odate: TDateTimePicker;
  Label5: TLabel;
  GroupBox6: TGroupBox;
  GroupBox7: TGroupBox;
  GroupBox8: TGroupBox;
  ADDB: TButton;
  GroupBox9: TGroupBox;
  GroupBox10: TGroupBox;
  GroupBox11: TGroupBox;
  CLEARB: TButton;
  Panel1: TPanel;
  Panel2: TPanel;
  Panel3: TPanel;
  Panel4: TPanel;
  procedure ADDBClick(Sender: TObject);
  procedure CLEARBClick(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure QuantityChange(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
```

```
 end;
var
 Form3: TForm3;
implementation
{$R *.dfm}
procedure TForm3.ADDBClick(Sender: TObject);
begin
//order name , properties , quantity, comapny , price can not be empty
if (Pname.Text<>'') and (Properties.Text<>'') and (quantity.value>0) and
(Company.Text<>'') and (Price.Text<>'') then
begin
Order.Open;
//Insert mode for add new record to table
 Order.Insert;
 order.FieldValues['OName']:=Pname.Text;
 order.FieldValues['Properties']:=Properties.Text;
order.FieldValues['Quantity']:=Quantity.Value ;
order.FieldValues['Company']:=Company.Text  ;
order.FieldValues['Price']:=Price.Text;
order.FieldValues['Odate']:=Odate.Date;
order.FieldValues['Adate']:=Adate.Date ;
//send values to the table
Order.Post;
order.close;
//show message to user
showmessage('Order is Added Successfully');
end
else
begin
 Showmessage('Do Not Leave Empty Spaces');
end;
```

```
end;
procedure TForm3.CLEARBClick(Sender: TObject);
begin
//Clear All fields on form
Pname.Clear;
Properties.clear;
Quantity.Value :=0;
Company.clear;
price.Clear;
Odate.Date:=now;
Adate.Date:=now;
end;
procedure TForm3.FormCreate(Sender: TObject);
var i:integer;
begin
//List Companies into the company combobox
Comp.Open;
Comp.First;
for i:=0 to comp.RecordCount-1 do
begin
 company.Items.Add(comp.FieldValues['Name']);
 comp.Next;
end;
end;
procedure TForm3.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  action:=caFree; end;
procedure TForm3.QuantityChange(Sender: TObject);
begin
if quantity.Value <0 then
quantity.Value:=0;
```

```
end;  end.
```

FORM 4 PRODUCT LİST

```
unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Spin, ComCtrls, StdCtrls, Mask, DBCtrls, DB, DBTables, Grids,
  DBGrids, ExtCtrls;
type
  TForm4 = class(TForm)
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    Table1: TTable;
    Panel1: TPanel;
    All: TRadioButton;
    Company: TRadioButton;
    Byname: TRadioButton;
    ByCode: TRadioButton;
    compT: TTable;
    GroupBox1: TGroupBox;
    Label8: TLabel;
    date1: TEdit;
    GroupBox2: TGroupBox;
    Label7: TLabel;
    guarantee: TEdit;
    GroupBox3: TGroupBox;
    Label6: TLabel;
    quantity: TEdit;
    GroupBox4: TGroupBox;
```

```
Label13: TLabel;
color: TEdit;
GroupBox5: TGroupBox;
Label12: TLabel;
size: TEdit;
GroupBox6: TGroupBox;
Label1: TLabel;
Scode: TEdit;
GroupBox7: TGroupBox;
Label2: TLabel;
Sname: TEdit;
GroupBox8: TGroupBox;
Label3: TLabel;
Pprice: TEdit;
GroupBox9: TGroupBox;
Label4: TLabel;
Sprice: TEdit;
GroupBox10: TGroupBox;
Label5: TLabel;
company1: TEdit;
GroupBox14: TGroupBox;
GroupBox15: TGroupBox;
GroupBox16: TGroupBox;
GroupBox17: TGroupBox;
GroupBox18: TGroupBox;
GroupBox19: TGroupBox;
ByCodeP: TPanel;
Label11: TLabel;
Code: TEdit;
SearchByCode: TButton;
ByNameP: TPanel;
```

```
  Label9: TLabel;
  PName: TEdit;
  SearchByNameB: TButton;
  ByComp: TPanel;
  Label10: TLabel;
  CompName: TEdit;
  SearchByComp: TButton;
  GroupBox12: TGroupBox;
  GroupBox13: TGroupBox;
  GroupBox11: TGroupBox;
  GroupBox20: TGroupBox;
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormActivate(Sender: TObject);

  procedure AllClick(Sender: TObject);
  procedure ByCodeClick(Sender: TObject);
  procedure CompanyClick(Sender: TObject);
  procedure BynameClick(Sender: TObject);
  procedure SearchByNameBClick(Sender: TObject);
  procedure SearchByCompClick(Sender: TObject);
  procedure SearchByCodeClick(Sender: TObject);
  procedure DBGrid1CellClick(Column: TColumn);
  procedure PNameKeyPress(Sender: TObject; var Key: Char);
  procedure CompNameKeyPress(Sender: TObject; var Key: Char);
  procedure CodeKeyPress(Sender: TObject; var Key: Char);
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

```
var
  Form4: TForm4;
implementation
{$R *.dfm}
procedure Clear(sender:Tobject);
begin
//Clear All Fields on form4
with form4 do
begin
Scode.clear;
Sname.Clear;
Pprice.Clear;
Sprice.Clear;
company1.clear;
size.clear;
color.Clear;
quantity.Clear;
guarantee.Clear;
date1.Clear;
end;
end;
procedure TForm4.FormClose(Sender: TObject; var Action: TCloseAction);
begin
//close table on form4 when form closes
  table1.Close;
    action:=caFree;
end;
procedure TForm4.FormActivate(Sender: TObject);
begin
WindowState := wsMaximized;
end;
```

```
procedure TForm4.AllClick(Sender: TObject);
begin
// set visible false for all panels and list all products
BycodeP.Visible:=false;
Bycomp.Visible := false;
Bynamep.Visible:=false;
table1.Filtered:=false;
end;
procedure TForm4.ByCodeClick(Sender: TObject);
begin
//When Bycode is selected show Bycode panel and setfocus to the code textbox
if bycode.Checked =true then
begin
bycodeP.Visible:=true;
Bycomp.Visible:=false;
bynamep.Visible :=false;
code.setfocus;
clear(sender);
end
else
bycodeP.Visible:=false;
end;
procedure TForm4.CompanyClick(Sender: TObject);
begin
//When Company is selected show Company panel and setfocus to the compname textbox
if company.Checked =true then
begin
Bycomp.Visible:=true;
bynamep.Visible :=false;
bycodep.Visible :=false;
Compname.SetFocus ;
```

```
Clear(sender);
end
else
bycomp.Visible:=false;
end;
procedure TForm4.BynameClick(Sender: TObject);
begin
//When Byname is selected show Byname panel and setfocus to the Pname textbox
if Byname.Checked =true then
begin
bynameP.Visible:=true;
Bycomp.Visible:=false;
bycodep.Visible :=false;
Pname.SetFocus ;
clear(sender);
end
else
bynameP.Visible:=false;
end;
//UserDefined procedure to bring values of a selected record to the fields on form
procedure Getinfo(sender:Tobject);
begin
form4.scode.Text:= form4.table1.FieldValues['Stock_code'];
form4.sname.Text:= form4.table1.FieldValues['Stock_Name'];
form4.Pprice.Text:= form4.table1.FieldValues['Purchase_Price'];
form4.Sprice.Text:= form4.table1.FieldValues['Selling_Price'];
form4.company1.Text:= form4.table1.FieldValues['Company'];
form4.size.Text:= form4.table1.FieldValues['size'];
form4.color.Text:= form4.table1.FieldValues['color'];
form4.guarantee.Text:= form4.table1.FieldValues['Guarantee'];
form4.date1.Text:= form4.table1.FieldValues['Date'];
```

```
form4.quantity.Text:= form4.table1.FieldValues['Quantity'];
end;
procedure TForm4.SearchByNameBClick(Sender: TObject);
begin
// search in table by stock name
table1.filter:='Stock_Name='+ quotedstr(Pname.Text );
table1.Filtered:=true;
//if no record found
if table1.RecordCount =0 then
//show user message
 showmessage('No Record Found')
 else
 //bring values by calling the userdefined procedure
 getinfo(form4);
 Pname.Clear;
 Pname.SetFocus;
end;
procedure TForm4.SearchByCompClick(Sender: TObject);
begin
// search in table by company
table1.filter:='Company='+ quotedstr(CompName.Text );
table1.Filtered:=true;
if table1.RecordCount =0 then
 showmessage('No Record Found')
 else
 //bring values by calling the userdefined procedure
 getinfo(form4);
 compname.Clear;
 compname.SetFocus; end;
procedure TForm4.SearchByCodeClick(Sender: TObject);
begin
```

```
// search in table by code
table1.filter:='Stock_Code='+ quotedstr(Code.Text );
table1.Filtered:=true;
if table1.RecordCount =0 then
 showmessage('No Record Found')
 else
  //bring values by calling the userdefined procedure
 getinfo(form4);
 code.Clear;
 code.SetFocus;
end;
//bring values of selected in dbgrid record onto form
procedure TForm4.DBGrid1CellClick(Column: TColumn);
begin
if dbgrid1.Fields[dbgrid1.SelectedIndex].AsString<>'' then
begin
getinfo(form4);
end;
end;
procedure TForm4.PNameKeyPress(Sender: TObject; var Key: Char);
begin
if key=#13 then
SearchByNameB.Click;
end;
procedure TForm4.CompNameKeyPress(Sender: TObject; var Key: Char);
begin
if key=#13 then
SearchByComp.Click;
end;

procedure TForm4.CodeKeyPress(Sender: TObject; var Key: Char);
```

```
begin
if key=#13 then
SearchByCode.Click;
end;
procedure TForm4.FormCreate(Sender: TObject);
begin
end;
end.
```

FORM 6 CUSTOMER İNFORMATİON

```
unit Unit6;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, DBCtrls, Grids, DBGrids, DB, DBTables, StdCtrls, Mask;
type
  TForm6 = class(TForm)
    DataSource1: TDataSource;
    customer: TTable;
    DBGrid1: TDBGrid;
    GroupBox1: TGroupBox;
    Label11: TLabel;
    SearchBynoB: TButton;
    No: TEdit;
    GroupBox2: TGroupBox;
    Label9: TLabel;
    Label10: TLabel;
    Fname: TEdit;
    Lname: TEdit;
    searchbyname: TButton;
```

```
GroupBox3: TGroupBox;
Label2: TLabel;
Label3: TLabel;
Cname: TEdit;
CLname: TEdit;
GroupBox4: TGroupBox;
Label6: TLabel;
Cfax: TEdit;
GroupBox5: TGroupBox;
CADD: TButton;
Button2: TButton;
Update: TButton;
Clear: TButton;
GroupBox6: TGroupBox;
GroupBox7: TGroupBox;
GroupBox8: TGroupBox;
GroupBox9: TGroupBox;
DBNavigator1: TDBNavigator;
Retail: TButton;
Button1: TButton;
GroupBox10: TGroupBox;
GroupBox11: TGroupBox;
GroupBox12: TGroupBox;
ALL: TRadioButton;
byno: TRadioButton;
Byname: TRadioButton;
Caddress: TEdit;
Label4: TLabel;
email: TEdit;
CPhone: TEdit;
Label7: TLabel;
```

```
    Label5: TLabel;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure CADDClick(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure SearchBynoBClick(Sender: TObject);
    procedure searchbynameClick(Sender: TObject);
    procedure ALLClick(Sender: TObject);
    procedure ClearClick(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure bynoClick(Sender: TObject);
    procedure BynameClick(Sender: TObject);
    procedure RetailClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure NoKeyPress(Sender: TObject; var Key: Char);
    procedure LnameKeyPress(Sender: TObject; var Key: Char);
    procedure UpdateClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form6: TForm6;
implementation
uses Unit5, Unit8;
{$R *.dfm}


procedure TForm6.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
  Customer.Close;
    action:=caFree;
end;
procedure TForm6.FormActivate(Sender: TObject);
begin
WindowState := wsMaximized;
end;
procedure TForm6.FormCreate(Sender: TObject);
begin
  customer.Open;
end;
procedure clear1(Sender:Tobject);
begin
//Clear all fields on the form
form6.Cname.clear;
form6.CLname.clear;
form6.Caddress.Clear;
form6.CPhone.clear;
form6.Cfax.Clear;
form6.email.Clear;
form6.No.Clear;
form6.Fname.clear;
form6.Lname.clear;
form6.customer.Filtered:=false;
Form6.Cadd.Enabled :=true;
form6.update.Enabled:=false;
form6.Button2.Enabled :=false;
form6.Cname.SetFocus ;
end;
procedure TForm6.CADDClick(Sender: TObject);
```

```
begin
//Customer name, last name, address, fax, phone , email cant be empty
if (Cname.Text<>'') and (Clname.Text<>'') and (caddress.Text<>'') and (cfax.Text<>'') and
(cphone.Text<>'') and (email.Text<>'') then
begin
//Insert mode to add new record
customer.Insert;
customer.FieldValues['Name']:=Cname.Text;
customer.FieldValues['Surname']:=CLname.Text;
customer.FieldValues['Address']:=Caddress.Text;
customer.FieldValues['Phone']:=Cphone.Text;
customer.FieldValues['Email']:=Email.Text;
customer.FieldValues['Fax']:=Cfax.Text;
//send values to table customer
customer.Post;
//Close table customer
customer.Close;
customer.Open;
//Call the clear procedure
clear1(form6);
end
else
showmessage('Do not leave Fields Empty');
end;
procedure TForm6.Button2Click(Sender: TObject);
begin
//Make user confirm deletion
if MessageDLG('Confirm Deletion',mtwarning,[mbYes,mbNo,mbCancel],0)=6 then
begin
//delete customer
customer.Delete;
```

```
//clear all fields
clear1(form6);
end
else
//Show message to uesr
 ShowMessage('Deletion is Canceled');
end;
procedure TForm6.SearchBynoBClick(Sender: TObject);
var code:integer;
  a:integer;
begin
val(no.Text,a,code);
//if number entered in the textfield
if code=0 then
begin
//search customer by no
customer.Filter:= 'CustomerId='+quotedstr(No.Text);
customer.Filtered:=true;
if customer.RecordCount =0 then
begin
//Show message to user
showmessage('No Record Found');
// Cancel Filter on customer table
customer.Filtered:=false;
No.Clear;
end
else
begin
//Clear No editbox
No.Clear;
No.SetFocus;
```

```
end;
end;
end;

procedure TForm6.searchbynameClick(Sender: TObject);
begin
//search customer by name and surname
customer.Filter:= 'Name='+quotedstr(Fname.Text);
customer.Filtered:=true;
customer.Filter:='Surname='+quotedstr(Lname.Text);
customer.Filtered:=true;
if customer.RecordCount =0 then
begin
customer.Filtered:=false;
Fname.Clear;
Lname.Clear;
//Show message to user
showmessage('No Customer Found');
end
else
begin
Fname.Clear;
Lname.Clear;
Fname.SetFocus;
end;
end;
procedure TForm6.ALLClick(Sender: TObject);
begin
//Show all customers cancel filter
customer.Filtered:=false;   end;
procedure TForm6.ClearClick(Sender: TObject);
```

```
begin
//Call clear1 procedure
clear1(sender);
end;
procedure TForm6.DBGrid1CellClick(Column: TColumn);
begin
if dbgrid1.Fields[dbgrid1.SelectedIndex].AsString<>'' then
begin
//bring the values of the selected record onto fields on form
Cname.Text:=customer.FieldValues['Name'];
CLname.Text:=customer.FieldValues['Surname'];
Caddress.Text:=customer.FieldValues['Address'];
Cphone.Text:=customer.FieldValues['Phone'];
Email.Text:=customer.FieldValues['Email'];
Cfax.Text:=customer.FieldValues['Fax'];
Cadd.Enabled :=false;
Update.Enabled :=true;
button2.Enabled:=true;
end;
end;
procedure TForm6.bynoClick(Sender: TObject);
begin
no.SetFocus;
end;
procedure TForm6.BynameClick(Sender: TObject);
begin
fname.SetFocus;
end;
procedure TForm6.RetailClick(Sender: TObject);
begin
Form5:=TForm5.Create(self);
```

```
Form5.show;
form5.Cno:=customer.FieldValues['CustomerID'];
end;
procedure TForm6.Button1Click(Sender: TObject);
begin
Form8:=TForm8.create(self);
form8.Edit1.Text := customer.FieldValues['CustomerId'];
form8.FormStyle := fsMDIChild;
Form8.show;
end;
procedure TForm6.NoKeyPress(Sender: TObject; var Key: Char);
begin
//if ENTER key is pressed on No editbox click searchbyno button
if key=#13 then
  begin
  SearchBynoB.Click;
  end;
end;
procedure TForm6.LnameKeyPress(Sender: TObject; var Key: Char);
begin
//if ENTER key is pressed on Lastname editbox click searchbyname button
 if key=#13 then
  begin
  searchbyname.Click;
  end;
end;
procedure TForm6.UpdateClick(Sender: TObject);
begin
if (Cname.Text<>'') and (Clname.Text<>'') and (caddress.Text<>'') and (cfax.Text<>'') and
(cphone.Text<>'') and (email.Text<>'') then
  begin
```

```
  Customer.Edit;
  customer.FieldValues['Name']:=Cname.Text;
  customer.FieldValues['Surname']:=CLname.Text;
  customer.FieldValues['Address']:=Caddress.Text;
  customer.FieldValues['Phone']:=Cphone.Text;
  customer.FieldValues['Email']:=Email.Text;
  customer.FieldValues['Fax']:=Cfax.Text;
  //send values to table customer
  customer.Post;
  //Close table customer
  customer.Close;
  customer.Open;
  clear1(sender);
  ShowMessage ('Update Successful');
  end;
end;
end.
```

FORM 7 PERSONEL İNFORMATİON

```
unit Unit7;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables, Mask, DBCtrls, ComCtrls;
type
  TForm7 = class(TForm)
    DataSource1: TDataSource;
    Personel: TTable;
    GroupBox1: TGroupBox;
    DBGrid1: TDBGrid;
```

```
GroupBox3: TGroupBox;
AddB: TButton;
UpdateB: TButton;
Clear: TButton;
DeleteB: TButton;
GroupBox2: TGroupBox;
GroupBox4: TGroupBox;
Label1: TLabel;
Label11: TLabel;
Sname: TEdit;
Ssurname: TEdit;
searchbyname: TButton;
GroupBox5: TGroupBox;
GroupBox6: TGroupBox;
GroupBox7: TGroupBox;
Button1: TButton;
GroupBox8: TGroupBox;
Label12: TLabel;
SearchBynoB: TButton;
No: TEdit;
GroupBox9: TGroupBox;
Phone: TEdit;
Email: TEdit;
Date1: TDateTimePicker;
Address: TEdit;
GroupBox10: TGroupBox;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
GroupBox11: TGroupBox;
```

```
Fname: TEdit;
Lname: TEdit;
duty: TEdit;
dept: TEdit;
salary: TEdit;
GroupBox12: TGroupBox;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
GroupBox13: TGroupBox;
Byname: TRadioButton;
byno: TRadioButton;
ALL: TRadioButton;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormActivate(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure AddBClick(Sender: TObject);
procedure DBGrid1CellClick(Column: TColumn);
procedure ClearClick(Sender: TObject);
procedure DeleteBClick(Sender: TObject);
procedure UpdateBClick(Sender: TObject);
procedure ALLClick(Sender: TObject);
procedure bynoClick(Sender: TObject);
procedure BynameClick(Sender: TObject);
procedure SearchBynoBClick(Sender: TObject);
procedure searchbynameClick(Sender: TObject);
procedure NoKeyPress(Sender: TObject; var Key: Char);
procedure SsurnameKeyPress(Sender: TObject; var Key: Char);
procedure Button1Click(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form7: TForm7;
implementation
uses Unit9;
{$R *.dfm}
procedure TForm7.FormClose(Sender: TObject; var Action: TCloseAction);
begin
     //close table personel on form close
  Personel.Close;
    action:=caFree;
end;
procedure TForm7.FormActivate(Sender: TObject);
begin
WindowState := wsMaximized;
end;
procedure TForm7.FormCreate(Sender: TObject);
begin
//open personel table
personel.Open;
end;
Procedure Clear1(Sender:Tobject);
begin
//Clear all fields on form
//by using with ... do statement no need to write the name of the form for every
//statements.
with form7 do
```

```
begin
Fname.clear;
Lname.Clear;
Duty.Clear;
Dept.Clear;
Salary.Clear;
Address.Clear;
Phone.Clear;
Email.Clear;
Date1.Date:=now;
addb.Enabled :=true;
updateB.Enabled :=false;
deleteb.Enabled :=false;
end;
end;
procedure TForm7.AddBClick(Sender: TObject);
begin
//First , last name , duty , department, salary , address cant be empty
if (Fname.Text<>'')  and (Lname.Text <>'') and (duty.Text<>'') and (dept.Text <>'') and
(salary.Text<>'') and (address.Text <>'') then
begin
//Insert mode to enter new record
personel.Insert;
personel.FieldValues['Fname']:=Fname.Text;
personel.FieldValues['Lname']:=Lname.Text;
personel.FieldValues['Duty']:=Duty.Text;
personel.FieldValues['Department']:=Dept.Text;
personel.FieldValues['Salary']:=strtofloat(Salary.Text);
personel.FieldValues['Address']:=Address.Text;
personel.FieldValues['Phone']:=Phone.Text;
personel.FieldValues['Email']:=Email.Text;
```

```
personel.FieldValues['HireDate']:=Date1.date;
//Send values to table
personel.Post;
//Close personel table
personel.Close;
personel.Open;
//clear all fields on form
Clear1(form7);
 showmessage('Personel is added successfully');
end
else
//show user message
 showmessage('Do not Leave Fields Empty');
end;
procedure TForm7.DBGrid1CellClick(Column: TColumn);
begin
if dbgrid1.Fields[dbgrid1.SelectedIndex].AsString<>'' then
begin
//Bring values to the fields on form
Fname.text:= personel.FieldValues['Fname'];
Lname.text:= personel.FieldValues['Lname'];
Duty.text:= personel.FieldValues['Duty'];
Dept.text:= personel.FieldValues['Department'];
Salary.text:= personel.FieldValues['Salary'];
Address.text:= personel.FieldValues['Address'];
Phone.text:= personel.FieldValues['Phone'];
Email.text:= personel.FieldValues['Email'];
Date1.Date:= personel.FieldValues['HireDate'];
updateB.Enabled:=true;
addB.Enabled :=false;
deleteB.Enabled :=true;
```

```
end;
end;
procedure TForm7.ClearClick(Sender: TObject);
begin
//Call userdefined procedure to clear all fields
clear1(form7);
end;
procedure TForm7.DeleteBClick(Sender: TObject);
begin
// Make user confirm the deletion
if Messagedlg('Please Confirm Deletion',mtwarning, [mbYes, mbNo,mbCancel],0)=6 then
  begin
  //Delete personel
  personel.Delete;
  //Show message
  showmessage('Personel is Deleted');
  end;
end;
procedure TForm7.UpdateBClick(Sender: TObject);
begin
//First , last name , duty , department, salary , address cant be empty
if (Fname.Text<>'') and (Lname.Text<>'') and (Duty.Text<>'') and (Dept.Text<>'')  and
(Salary.Text<>'') then
begin
//Edit mode to update personel
personel.Edit;
personel.FieldValues['Fname']:=Fname.Text;
personel.FieldValues['Lname']:=Lname.Text;
personel.FieldValues['Duty']:=Duty.Text;
personel.FieldValues['Department']:=Dept.Text;
personel.FieldValues['Salary']:=Salary.Text;
```

```
personel.FieldValues['Address']:=Address.Text;
personel.FieldValues['Phone']:=Phone.Text;
personel.FieldValues['Email']:=Email.Text;
personel.FieldValues['HireDate']:=Date1.date;
//Send values to table
personel.Post;
//close table
personel.Close;
personel.Open;
 showmessage('Personel is updated successfully');
end
else
showmessage('Please Do not Leave Empty Spaces');
end;
procedure TForm7.ALLClick(Sender: TObject);
begin
//Cancel Filter on table Personel
PERSONel.Filtered :=false;
end;
procedure TForm7.bynoClick(Sender: TObject);
begin
no.SetFocus;
end;
procedure TForm7.BynameClick(Sender: TObject);
begin
Sname.SetFocus;
end;
procedure TForm7.SearchBynoBClick(Sender: TObject);
begin
//Find Personel  by No
personel.Filter:='PNo='+quotedstr(No.Text);
```

```
personel.Filtered :=true;
if personel.RecordCount =0 then
  begin
  showmessage('No Record Found');
  //if no record found cancel filter show all personels
  personel.Filtered:=false;
  No.Clear;
  No.SetFocus;
  end
  else
  begin
   No.Clear;
  No.SetFocus;
  end;
end;
procedure TForm7.searchbynameClick(Sender: TObject);
begin
//Find Personel  by Name
personel.Filter:='Fname='+quotedstr(Sname.Text);
personel.Filtered :=true;
personel.Filter:='Lname='+quotedstr(Ssurname.Text);
personel.Filtered :=true;
if personel.RecordCount =0 then
  begin
  showmessage('No Record Found');
    //if no record found cancel filter show all personels
  personel.Filtered:=false;
  Sname.Clear;
  Ssurname.Clear;
Sname.SetFocus;
  end
```

```
  else
  begin
    Sname.Clear;
  Ssurname.Clear;
Sname.SetFocus;
  end;
end;
procedure TForm7.NoKeyPress(Sender: TObject; var Key: Char);
begin
//if ENTER key is pressed in No EditBox SearchByno button's click event is called
if key=#13 then
  SearchBynoB.click;
end;
procedure TForm7.SsurnameKeyPress(Sender: TObject; var Key: Char);
begin
//if ENTER key is pressed in Ssurname EditBox, SearchByname button's click event is
called
if key=#13 then
  searchbyname.click;
end;
procedure TForm7.Button1Click(Sender: TObject);
begin
//Show Mail Form
mail:=Tmail.create(self);
mail.show;
mail.txtTo.Text :=personel.FieldValues['email'];
end;

end.
```

APPENDIX 2: DATABASE TABLES

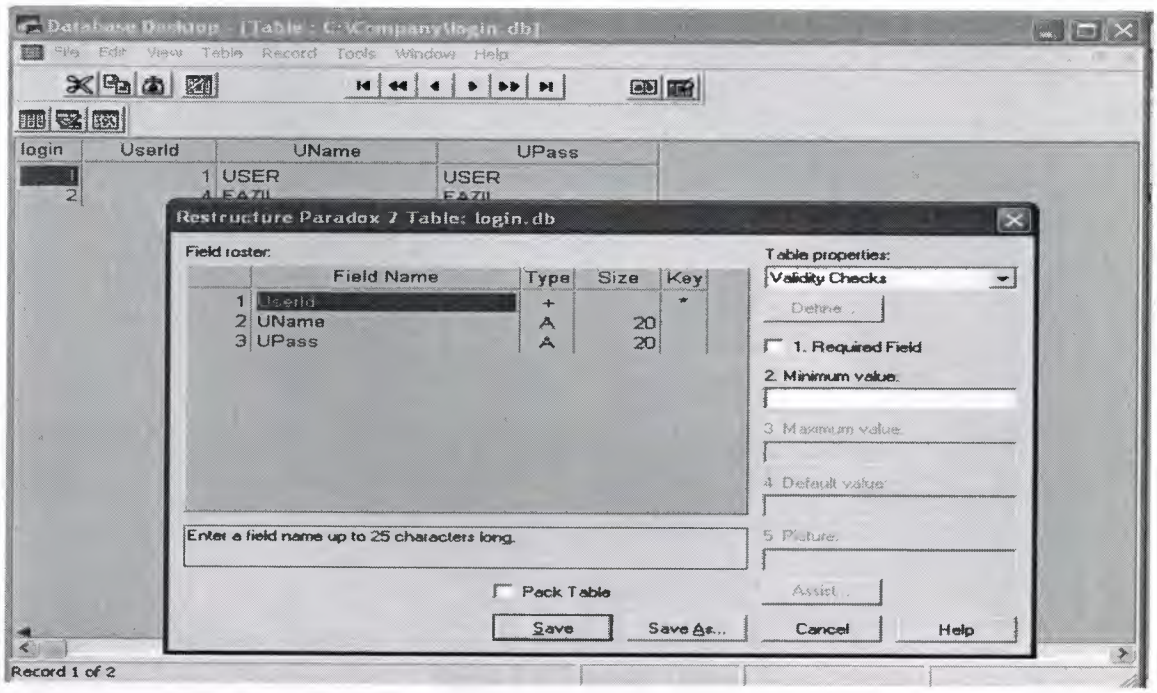


Fig. 4.1 Login Table

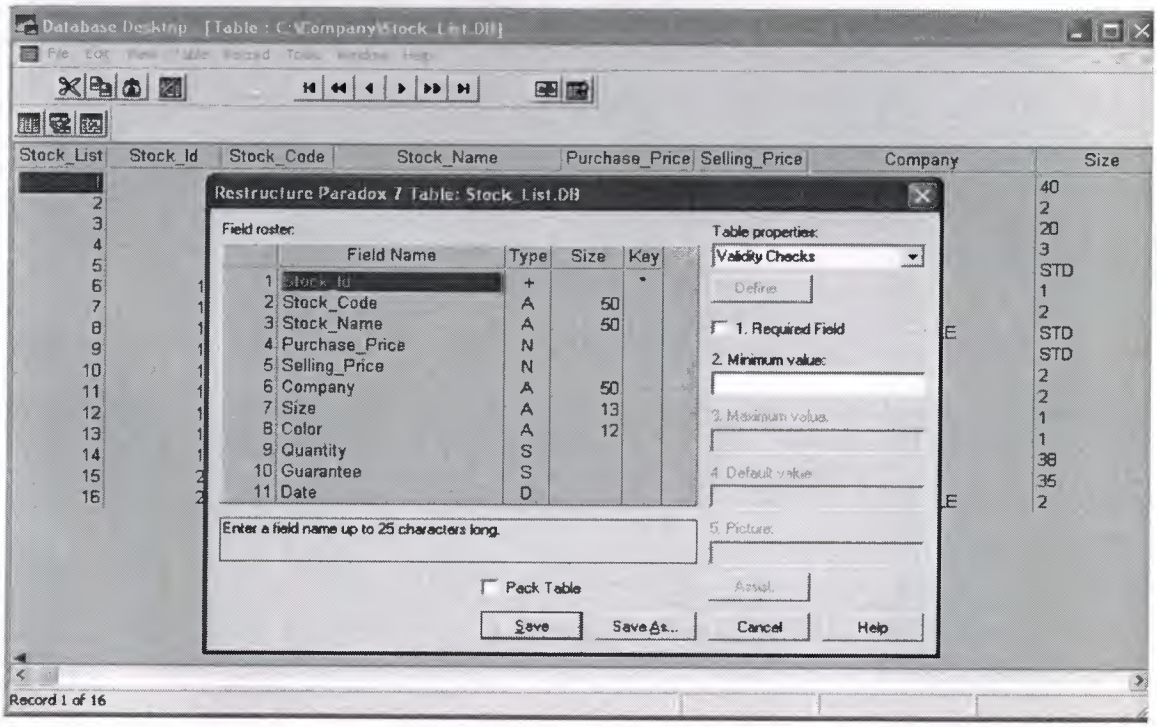


Fig. 4.2 Stock List Table

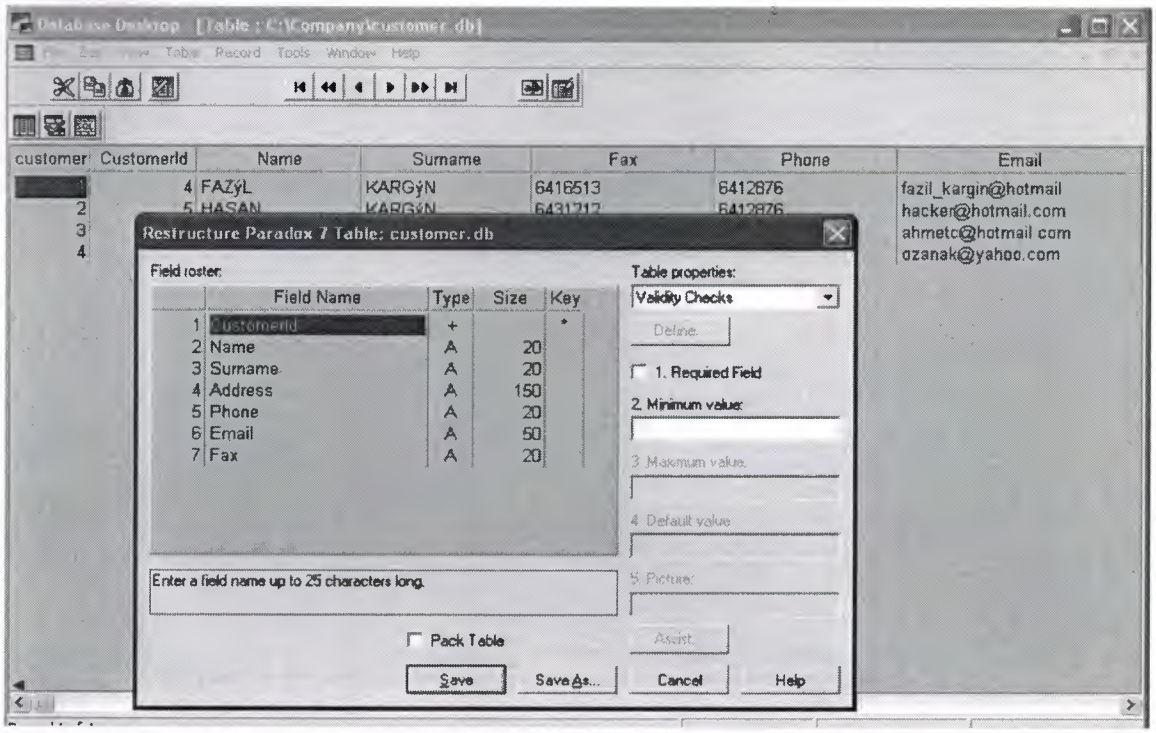


Fig. 4.3 Customer Table

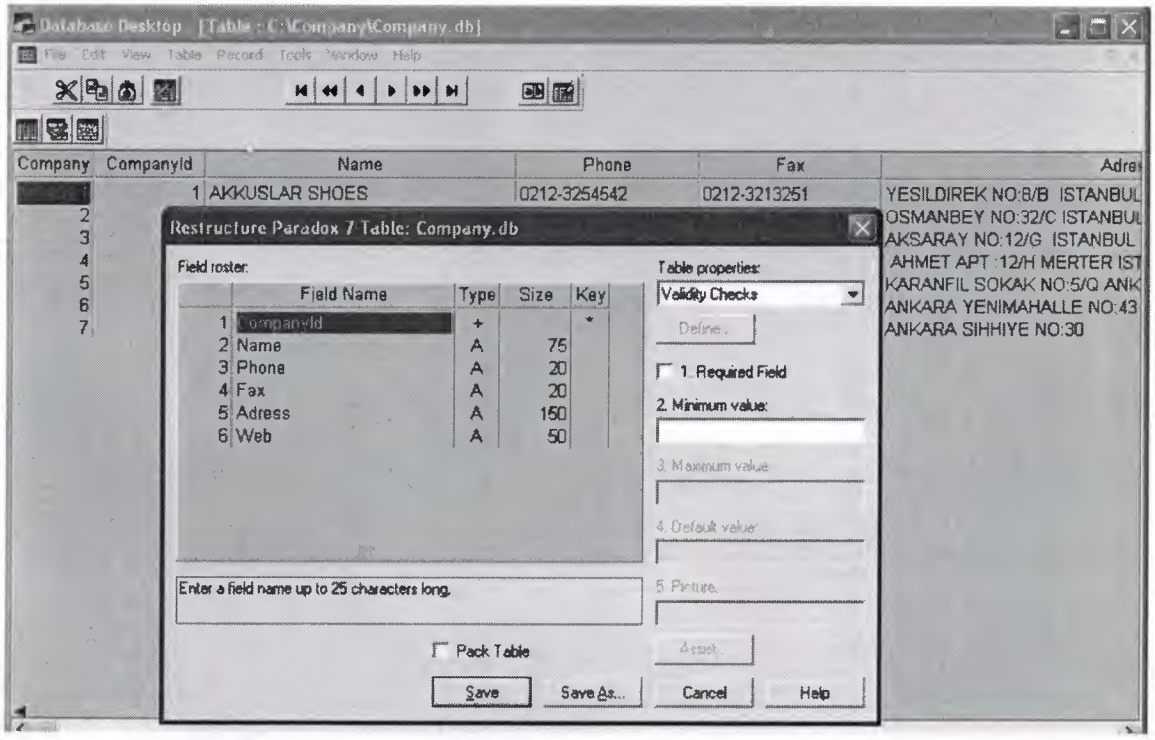


Fig. 4.4 Company Table

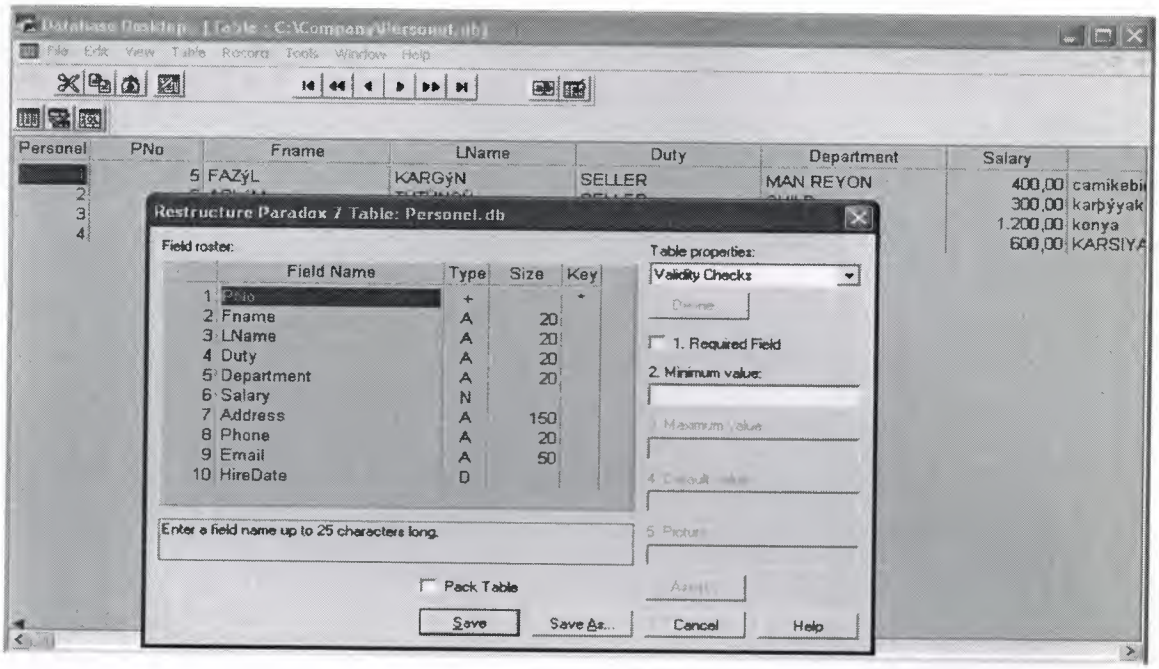


Fig. 4.5 personelTable

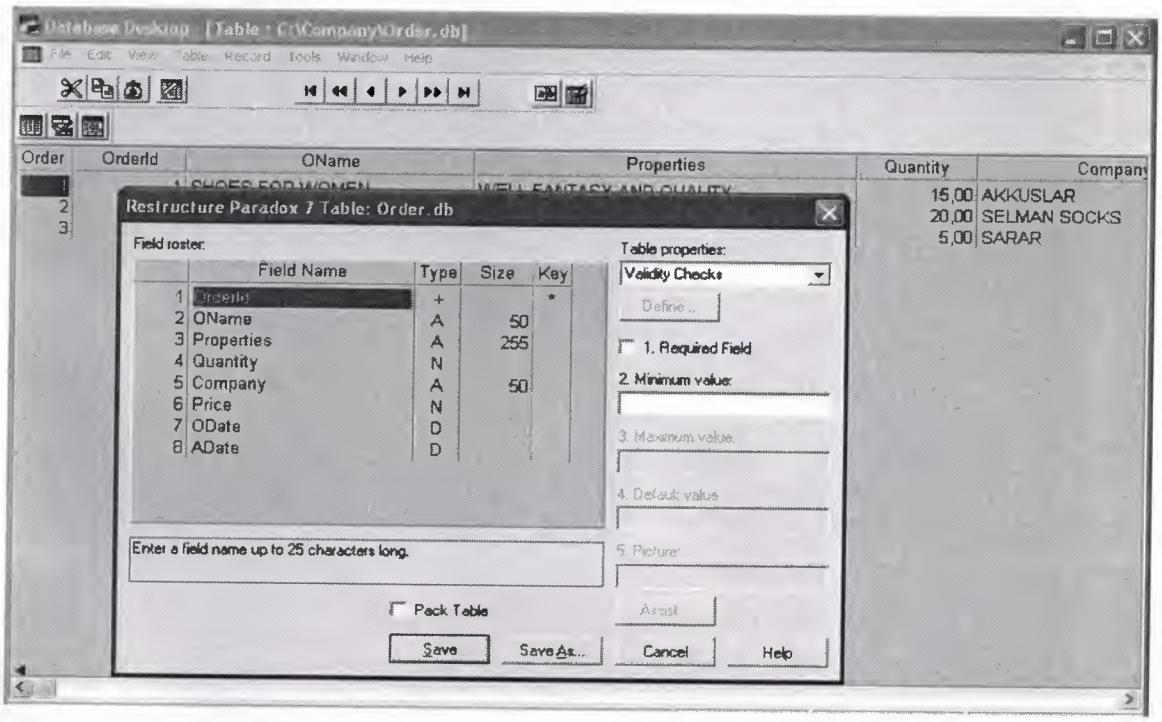


Fig. 4.6 order Table

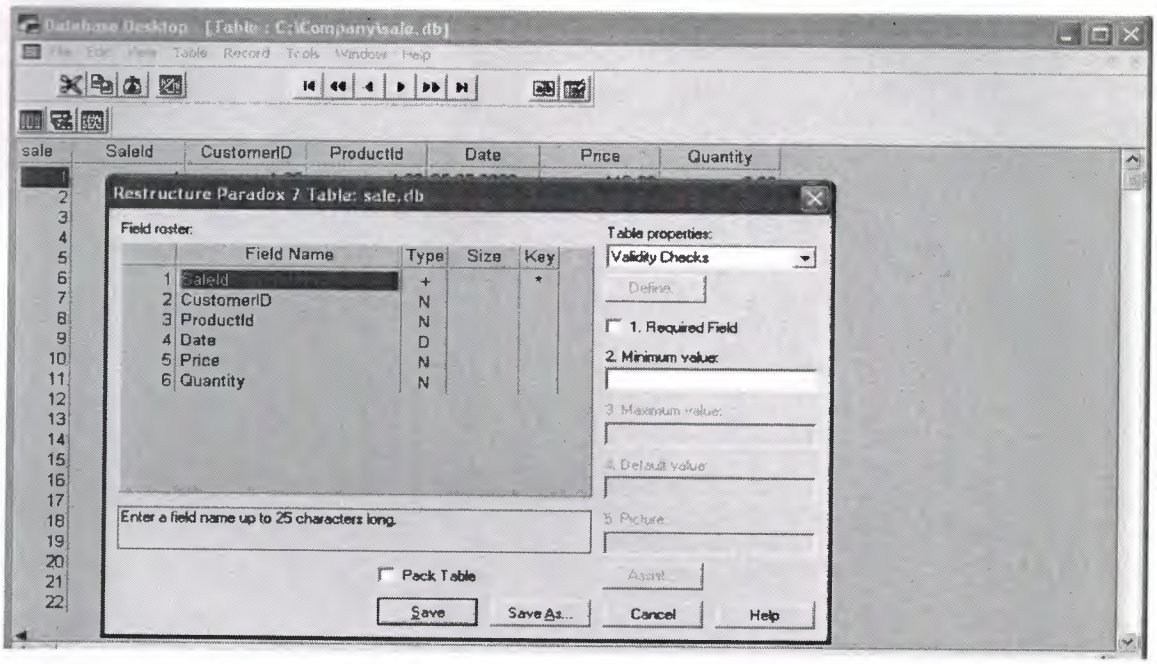


Fig. 4.7 Sale Table