

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

COMPUTER FIRM STOCK PROGRAM

**Graduation Project
COM400**

Student: Cevdet DİKİCİLER (20030572)

Supervisor: Assist. Prof. Dr. Elbrus Bashir IMANOV

Nicosia - 2008

ACKNOWLEDGMENTS

Firstly, I would like to thank to my supervisor Mr Elbrus IMANOV for his great advise and recommendation for finishing my project properly also, teaching and guiding me in other lectures.

I dont knowhow can I thank to my family for their endless support from my starting day in my educational life until today. I will never forget the things that my father Mr.Erden DİKİCİLER did for me during my educational life, also I want to say thanks to my mother Mrs. Şenay DİKİCİLER and my sister Cansu DİKİCİLER. I dedicate my project to them.

I thank all the staff of the faculty of engineering for giving facilities to practise, teaching and solving problem in my complete undergraduation program.

And finally I want to thank to my friends Esra OKUYAN,Murat EVEREKLI,Ali KARAHAN,Alper KÜRKÇÜ,Atakan AKAR for their help, they get tired with me, and they helped me everytime whenever I want.

I thank them with my all.

I promise to do my best in my life as an bachelor of engineer after finishing my undergraduate program.

ABSTRACT

The aim of this project is to keep the stock of the computer parts and to sell them to the customers. The program was prepared by using Delphi programming language and using Paradox Database. This project consist of so many forms and menus. The main form of the Project reaches to the other forms which are include information about the main computer parts, their stock status and their prices. This software can be used in Computer firms and they can control their stock status and they can keep reports of sales. These are the general information about the project. You can see the complete explanations in the next chapters.

TABLE OF CONTENTS

ACKNOWLEDGMENT	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	v

CHAPTER 1

1. BASIC CONCEPT OF DELPHI

1.1. Introduction to Delphi	1
1.2. What is Delphi?	1
1.2.1. Delphi Compilers	2
1.2.2. What kind of programming can you do with Delphi?	2
1.2.3. History Of Delphi	3
1.2.4. Advantages&Disadvantages of Delphi	5
1.3. Delphi 6 Editions	6
1.3.1. Delphi 6 Archite	7
1.3.2. Installation Delphi 6	7
1.4. A Tour Of The Environment	10
1.4.1. Running Delphi For The First Time	11
1.4.2. The Delphi IDE	11
1.4.3. The Menus & Toolbar	13
1.4.4. The Component Palette	13
1.4.5. The Code Editor	14
1.4.6. The Object Inspector	15
1.4.7. The Object TreeView	16
1.4.8. Class Completion	17
1.4.9. Debugging applications	18
1.4.10. Exploring databases	19
1.4.11. Templates and the Object Repository	20
1.5. Programming With Delphi	21
1.5.1. Starting a New Application	21
1.5.1.1. Setting Property Values	23
1.5.2. Adding objects to the form	24
1.5.3. Add a Table and a StatusBar to the form	24
1.5.4. Connecting to a Database	26

CHAPTER 2

2. DATA BASE SYSTEM

2.1. INTRODUCTION TO DATABASE	30
2.2. HISTORY	31
2.3. DATABASE MODELS	

2.3.1. Flat model	33
2.3.2. Hierarchical model	33
2.3.3. Network model	33
2.3.4. Relational model	34
2.3.4.1. Relational operations	35
2.3.5. Dimensional model	36
2.3.6. Object database models	37
2.4. DATABASE INTERNALS	37
2.4.1. Indexing	38
2.4.2. Transactions and concurrency	38
2.4.3. Replication	39
2.5. APPLICATIONS OF DATABASE	

CHAPTER 3

3. DESCRIPTION OF PROJECT	40
3.1. MAIN MENU	40
3.1.1. Add New Product	42
3.1.2. Delete Product	43
3.1.3. Search Product	44
3.2. STOCK CONTROL	45
3.3. SELL PRODUCT	46
3.4. CUSTOMER MENU	48

CHAPTER 4

4. CONCLUSION	50
REFERENCES	51
4.1. APPENDIX	52

INTRODUCTION

This project is keeps the stock information of a computer firm which uses PARADOX queries. This software was prepared by using Borland Delphi 7 and PARADOX Database.

The subjects chapter by chapter so let us go through the overview the chapters in breif:

In the First Chapter Borland Delphi 7 programming language is described, its properties, components and some examples, we used Borland Delphi 7 in our project, because Delphi is a visual programming language and it is easy to code. Borland Delphi 7 generally used for creating win32 applications, because it is easy to code than other programming languages to code win32 applications.

In the Second Chapter we described Database systems. We used PARADOX Database system in this software with Borland Delphi 7.

Third Chapter is about the project , how we create it, its forms and using the software.

Finally, the last chapter is the explanation of the program followed by the Appendices. So by developing and moderating of technology our program can be developed and updated. Also new properties could be added in to the program in the future.

CHAPTER 1

1.BASIC CONCEPT OF DELPHI

1.1.Introduction to Delphi

Although we are not the most experienced or knowledgeable person in the university. We thought it was time to write a good introductory article for Delphi.

1.2.What is Delphi?

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

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

For the purposes of this series we will be using Delphi 6. Delphi 6 provides all the tools you need to develop, test and deploy Windows applications, including a large number of so-called reusable components.

Borland Delphi, provides a cross platform solution when used with Borland Kylix - Borland's RAD tool for the Linux platform.

1.2.1. Delphi Compilers

There are two types compiler for Delphi

- **Turbo Delphi** : Free industrial strength Delphi RAD (Rapid Application Development) environment and compiler for Windows. It comes with 200+ components and its own Visual Component Framework.
- **Turbo Delphi for .NET**: Free industrial strength Delphi application development environment and compiler for the Microsoft .NET platform.

1.2.2. What kind of programming can you do with Delphi?

The simple answer is "more or less anything". Because the code is compiled, it runs quickly, and is therefore suitable for writing more or less any program that you would consider a candidate for the Windows operating system.

You probably won't be using it to write embedded systems for washing machines, toasters or fuel injection systems, but for more or less anything else, it can be used (and the chances are that probably someone somewhere has!)

Some projects to which Delphi is suited:

- Simple, single user database applications
- Intermediate multi-user database applications
- Large scale multi-tier, multi-user database applications
- Internet applications
- Graphics Applications
- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools
- Communications tools using the Internet, Telephone or LAN
- Web based applications

This is not intended to be an exhaustive list, more an indication of the depth and breadth of Delphi's applicability. Because it is possible to access any and all of the Windows API, and because if all else fails, Delphi will allow you to drop a few lines of assembler code directly into your ordinary Pascal instructions, it is possible to do more or less anything. Delphi can also be used to write Dynamically Linked Libraries (DLLs) and can call out to DLLs written in other programming languages without difficulty.

Because Delphi is based on the concept of self contained Components (elements of code that can be dropped directly on to a form in your application, and exist in object form, performing their function until they are no longer required), it is possible to build applications very rapidly. Because Delphi has been available for quite some time, the number of pre-written components has been increasing to the point that now there is a component to do more or less anything you can imagine. The job of the programmer has become one of gluing together appropriate components with code that operates them as required.

1.2.3. History Of Delphi

Delphi was one of the first of what came to be known as "RAD" tools, for Rapid Application Development, when released in 1995 for the 16-bit Windows 3.1 . Delphi 2, released a year later, supported 32-bit Windows environments, and a C++ variant, C++ Builder , followed a few years after.

The chief architect behind Delphi, and its predecessor Turbo Pascal , was Anders Hejlsberg until he was headhunted in 1996 by Microsoft , where he worked on Visual J++ and subsequently became the chief designer of C Sharp programming language|C# and a key participant in the creation of the Microsoft .NET Framework.

In 2001 a Linux version known as Kylix programming tool|Kylix became available. However, due to low quality and subsequent lack of interest, Kylix was abandoned after version 3.

Support for Linux and Windows cross platform development (through Kylix and the CLX component library) was added in 2002 with the release of Delphi 6.

Delphi 8, released December 2003, was a .NET -only release that allowed developers to compile Delphi Object Pascal code into .NET Microsoft Intermediate Language|MSIL . It was also significant in that it changed its IDE for the first time, from the multiple-floating-window-on-desktop style IDE to a look and feel similar to Microsoft's Visual Studio.NET.

Although Borland fulfilled one of the biggest requests from developers (.NET support), it was criticized both for making it available too late, when a lot of former Delphi developers had already moved to C#, and for focusing so much on backward compatibility that it was not very easy to write new code in Delphi. Delphi 8 also lacked significant high-level features of the c sharp|C# language, as well as many of the more appealing features of Microsoft's Visual Studio IDE. (There were also concerns about the future of Delphi Win32 development. Because Delphi 8 did not support Win32, Delphi 7.1 was included in the Delphi 8 package.)

The next version, Delphi 2005 (Delphi 9), included the Win32 and .NET development in a single IDE, reiterating Borland's commitment to Win32 developers. Delphi 2005 includes design-time manipulation of live data from a database. It also includes an improved IDE and added a "for ... in" statement (like C#'s foreach) to the language. However, it was criticized by some for its bugs; both Delphi 8 and Delphi 2005 had stability problems when shipped, which were only partially resolved in service packs.

In late 2005 , Delphi 2006 was released and federated development of C# and Delphi.NET, Delphi Win32 and C++ into a single IDE. It was much more stable than Delphi 8 or Delphi 2005 when shipped, and improved even more after the service packs and several hotfixes.

On February 8 , 2006 , Borland announced that it was looking for a buyer for its IDE and database line of products, which include Delphi, to concentrate on its Application Lifecycle Management|ALM line. The news met with voluble optimism from the remaining Delphi users.

On September 6 , 2006, The Developer Tools Group (the working name of the not yet spun off company) of Borland Software Corporation released single language versions

of Borland Developer Studio, bringing back the popular "Turbo" moniker. The Turbo product set includes Turbo Delphi for Win32, Turbo Delphi for .NET, Turbo C++, and Turbo C#. Each version is available in two editions: "Explorer"—a free downloadable version—and "Professional"—a relatively cheap (US\$399) version which opens access to thousands of third-party components. Unlike earlier "Personal" editions of Delphi, new "Explorer" editions can be used for commercial development.

On November 14, 2006, Borland announced the cancellation of the sale of its Development tools; instead of that it would spin them off into an independent company named "CodeGear"

1.2.4. Advantages & Disadvantages Delphi

Advantages

Delphi exhibits the following advantages:

- Rapid Application Development (RAD)
- Based on a well-designed language - high-level and strongly typed, with low-level escapes for experts
- A large community on Usenet and the World Wide Web (e.g. news://newsgroups.borland.com and Borland's web access to Delphi)
- Can compile to a single executable, simplifying distribution and reducing DLL versioning issues
- Many VCL and third-party components (usually available with full source code) and tools (documentation, debug tools, etc.)
- Quick optimizing compiler and ability to use assembler code
- Multiple platform native code from the same source code
- High level of source compatibility between versions
- Cross Kylix - a third-party toolkit which allows you to compile native Kylix/Linux applications from inside the Windows Delphi IDE, hence easily enabling dual-platform development and deployment

- Cross FBC - a sister project to CrossKylix, which enables you to cross-compile your Windows Delphi applications to multi-platform targets - supported by the Free Pascal compiler - without ever leaving the Delphi IDE
- Class helpers to bridge functionality available natively in the Delphi RTL, but not available in a new platform supported by Delphi
- The language's object orientation features only class- and interface-based Polymorphism in object-oriented programming|polymorphism

Disadvantages

- Limited cross-platform capability for Delphi itself. Compatibles provide more architecture/OS combinations
- Access to platform and third party libraries require header files to be translated to Pascal. This creates delays and introduces the possibilities of errors in translation.
- There are fewer published books on Delphi than on other popular programming languages such as C++ and C#
- A reluctance to break any code has lead to some convoluted language design choices, and orthogonality and predictability have suffered

1.3. Delphi 6 Editions

There are 3 editions in Delphi 6 :

- **Delphi Personal** - makes learning to develop non-commercial Windows applications fast and fun. Delphi 6 Personal makes learning Windows development easy with drag-and-drop visual programming.
- **Delphi Professional** - adds the tools necessary to create applications with the latest Windows® ME/2000 look-and-feel. Dramatically enhance functionality with minimal code using the power and flexibility of SOAP and XML to easily integrate Web Services into client-side applications.

- **Delphi Enterprise** - includes additional tools, extensive options for Internet. Delphi 6 makes next-generation e-business development with Web Services a snap.

This Program will concentrate on the Enterprise edition..

1.3.1. Delphi 6 Archite

Delphi 6 Architect is designed for professional enterprise developers who need to adapt quickly to changing business rules and manage sophisticated applications that synchronize with multiple database schemas. Delphi 2006 Architect includes an advanced ECO III framework that allows developers to rapidly deploy scalable external facing Web applications with executable state diagrams, object-relational mapping, and transparent persistence.

Delphi 6 Architect includes all of the capabilities of the Enterprise edition, and includes the complete ECO III framework, including new support for ECO State Machines powered by State Chart visual diagrams, and simultaneous persistence to multiple and mixed database servers.

- State Chart Diagrams
- Executable ECO State Machines
- Multi- and Mixed- ECO database support

1.3.2.Installation Delphi 6

To install Delphi 6 Enterprise, run INSTALL.EXE (default location C:\Program Files\Borland Delphi) and follow the installation instructions.

We are prompted to select a product to install, you only have one choice "Delphi 6":



Figure 1.1 The Select Page For Start Installation

While the setup runs, you'll need to enter your serial number and the authorization key (the two you got from inside a Cd rom driver).

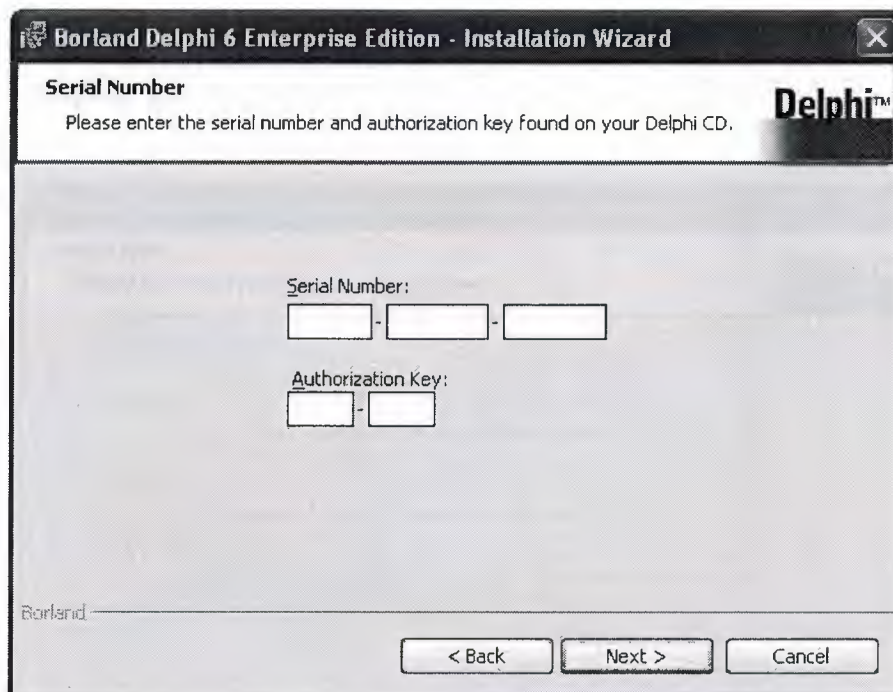


Figure 1.2 Serial Number And Authorization Screen

Later, the License Agreement screen will popup:

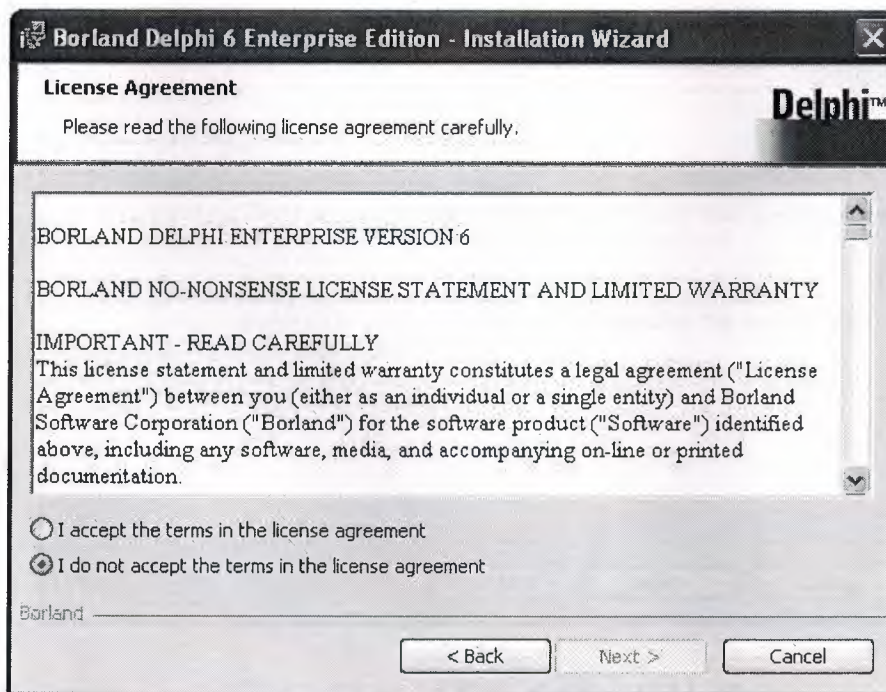


Figure 1.3 Lisanse Agreement Screen

After that, you have to pick the Setup Type, choose Typical. This way Delphi 6 Enterprise will be installed with the most common options. The next screen prompts you to choose the Destination folder.

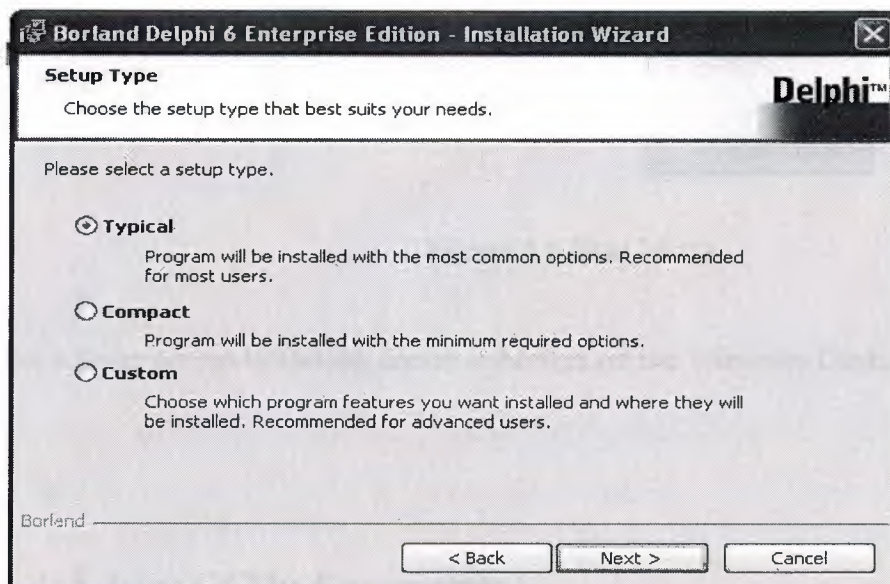


Figure 1.4.SetUp Type

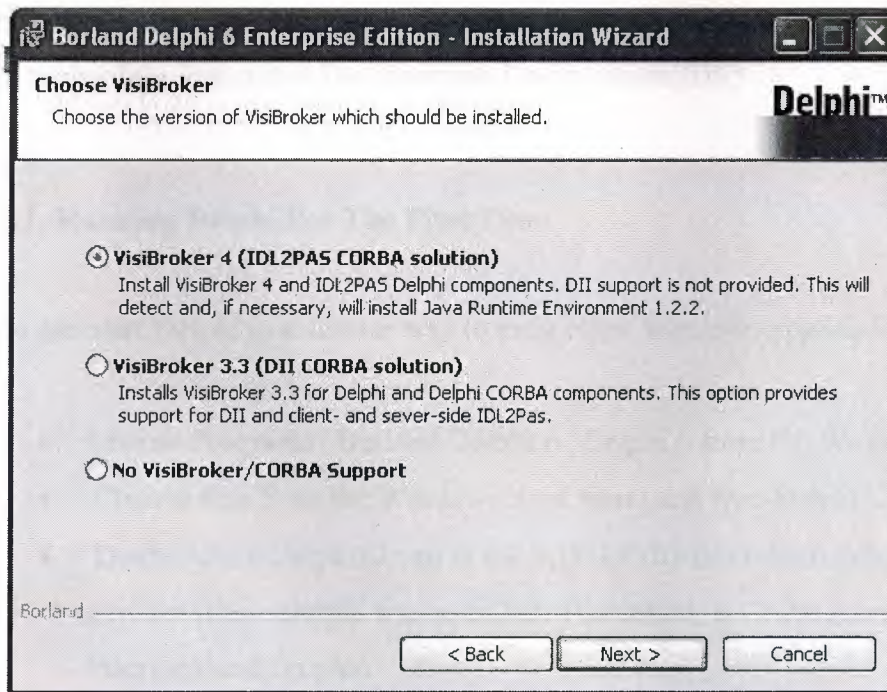


Figure 1.5 Destination Folder Screen

At the end of the installation process, the set-up program will create a sub menu in the Programs section of the Start menu, leading to the main Delphi 6 Enterprise program plus some additional tools.



1.5.Start Menu Screen

Figure 1.6.Start Menu

For a faster access to Delphi, create a shortcut on the Windows Desktop.

1.4. 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)

1.4.1. Running Delphi For The First Time

You can start Delphi in a similar way to most other Windows applications:

- Choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu
- Choose Run from the Windows Start menu and type Delphi32
- Double-click Delphi32.exe in the \$(DELPHI)\Bin folder. Where \$(DELPHI) is a folder where Delphi was installed. The default is C:\Program Files\Borland\Delphi6.
- Double-click the Delphi icon on the Desktop (if you've created a shortcut)

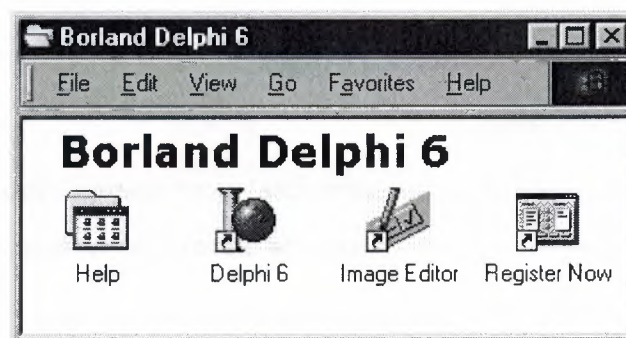


Figure 1.7.Borland Delphi 6 Folder

1.4.2. The Delphi IDE

As explained before, one of the ways to start Delphi is to choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu.

When Delphi starts (it could even take one full minute to start - depending on your hardware performance) you are presented with the IDE: the user interface where you can design, compile and debug your Delphi projects.

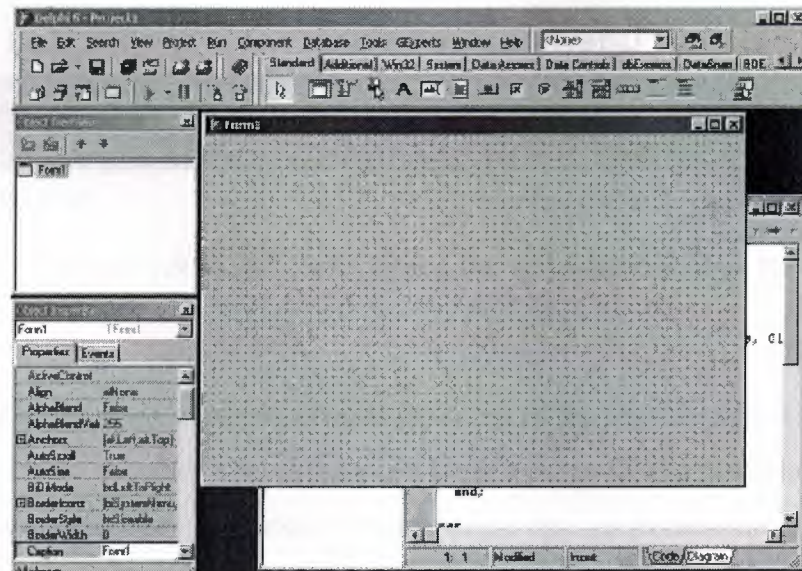


Figure 1.8.IDE

Like most other development tools (and unlike other Windows applications), Delphi IDE comprises a number of separate windows.

Some of the facilities that are included in the "Integrated Development Environment" (IDE) are listed below:

- A syntax sensitive program file editor
- A rapid optimising compiler
- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools
- Image/Icon/Cursor creation / editing tools
- Version Control CASE tools

1.4.3. The Menus & Toolbar

The main window, positioned on the top of the screen, contains the main menu, toolbar and Component palette.

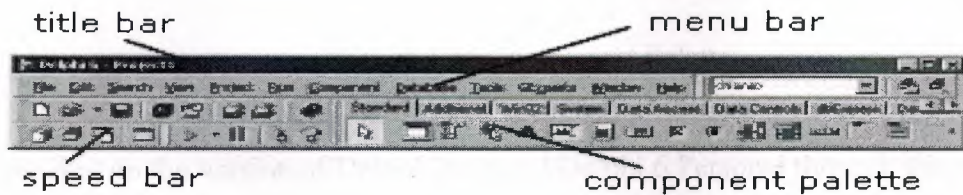


Figure 1.9.Menu ,Title , Speed Bar & Component Palette

The title bar of the main window contains the name of the current project (you'll see in some of the future chapters what exactly is a Delphi project). The menu bar includes a dozen drop-down menus - we'll explain many of the options in these menus later through this course. The toolbar provides a number of shortcuts to most frequently used operations and commands - such as running a project, or adding a new form to a project. To find out what particular button does, point your mouse "over" the button and wait for the tooltip. As you can see from the tooltip (for example, point to [Toggle Form/Unit]), many toolbuttons have keyboard shortcuts ([F12]).

The menus and toolbars are freely customizable. I suggest you to leave the default arrangement while working through the chapters of this course.

1.4.4. The Component Palette

You are probably familiar with the fact that any window in a standard Windows application contains a number of different (visible or not to the end user) objects, like: buttons, text boxes, radio buttons, check boxes etc. In Delphi programming terminology such objects are called controls (or components). Components are the building blocks of every Delphi application. To place a component on a window you drag it from the component palette. Each component has specific attributes that enable you to control your application at design and run time.

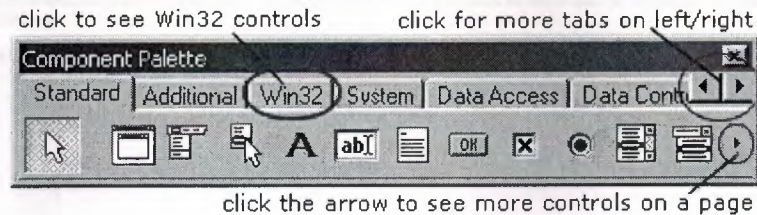


Figure 1.10.Component Palette

Depending on the version of Delphi (assumed Delphi 6 Personal through this course), you start with more than 85 components at your disposal - you can even add more components later (those that you create or from a third party component vendor).

The components on the Component Palette are grouped according to the function they perform. Each page tab in the Component palette displays a group of icons representing the components you can use to design your application interface. For example, the Standard and Additional pages include controls such as an edit box, a button or a scroll box.

To see all components on a particular page (for example on the Win32 page) you simply click the tab name on the top of the palette. If a component palette lists more components that can be displayed on a page an arrow will appear on a far right side of the page allowing you to click it to scroll right. If a component palette has more tabs (pages) that can be displayed, more tabs can be displayed by clicking on the arrow buttons on the right-hand side.

1.4.5. The Code Editor

Each time you start Delphi, a new project is created that consists of one *empty* window. A typical Delphi application, in most cases, will contain more than one window - those windows are referred to as forms.

In our case this form has a name, it is called Form1. This form can be renamed, resized and moved, it has a caption and the three standard minimize, maximize and close buttons. As you can see a Delphi form is a regular Windows window

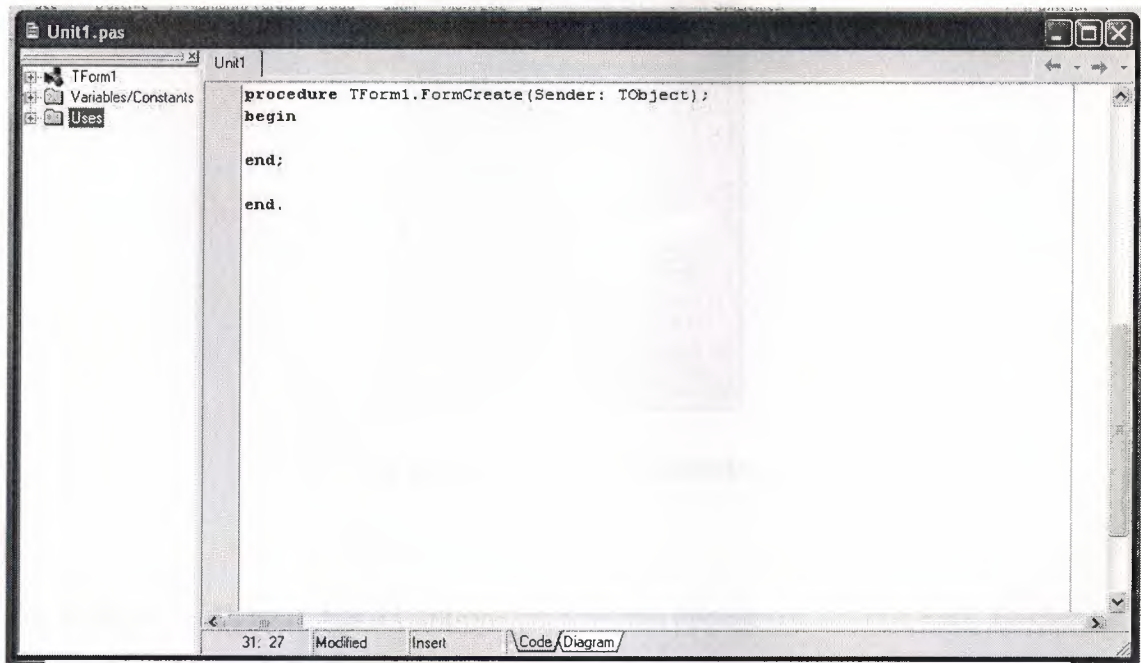


Fig.1.11.Code Editor Window

If the Form1 is the active window and you press [F12], the Code Editor window will be placed on top. As you design user interface of your application, Delphi automatically generates the underlying Object Pascal code. More lines will be added to this window as you add your own code that drives your application. This window displays code for the current form (Form1); the text is stored in a (so-called) unit - Unit1. You can open multiple files in the Code Editor. Each file opens on a new page of the Code editor, and each page is represented by a tab at the top of the window.

1.4.6. The Object Inspector

Each component and each form, has a set of properties – such as color, size, position, caption – that can be modified in the Delphi IDE or in your code, and a collection of events – such as a mouse click, keypress, or component activation – for which you can specify some additional behavior. The Object Inspector displays the properties and events (note the two tabs) for the selected component and allows you to change the property value or select the response to some event.

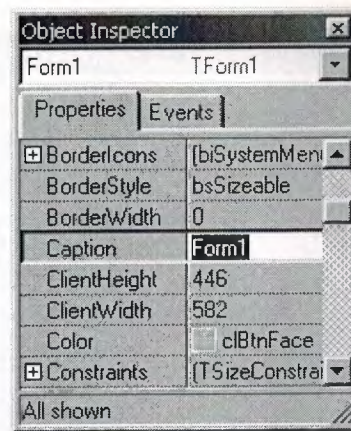


Figure 1.11.Object Inspector

For example, each form has a Caption (the text that appears on it's title bar). To change the caption of Form1 first activate the form by clicking on it. In the Object Inspector find the property Caption (in the left column), note that it has the 'Form1' value (in the right column). To change the caption of the form simply type the new text value, like 'My Form' (without the single quotes). When you press [Enter] the caption of the form will change to My Form.

Note that some properties can be changed more simply, the position of the form on the screen can be set by entering the value for the Left and Top properties - or the form can be simply dragged to the desired location.

1.4.7. The Object TreeView

Above the Object Inspector you should see the Object TreeView window. For the moment it's display is pretty simple. As you add components to the form, you'll see that it displays a component's parent-child relationships in a tree diagram. One of the great features of the Object TreeView is the ability to drag and drop components in order to change a component container without losing connections with other components.

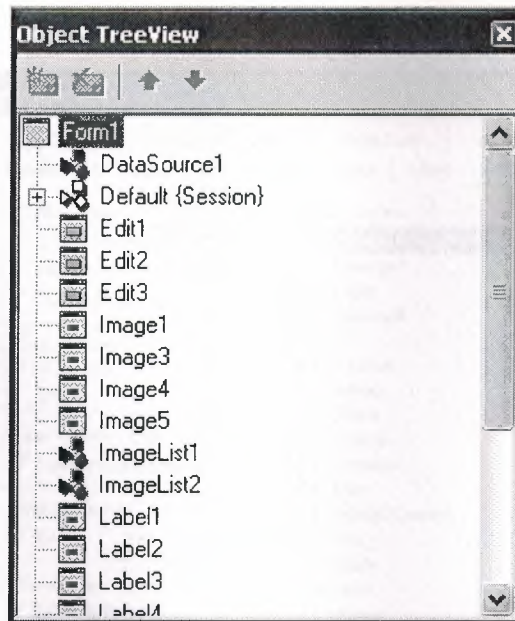


Figure 1.12.Object Tree View

The Object TreeView, Object Inspector and the Form Designer (the Form1 window) work cooperatively. If you have an object on a form (we have not placed any yet) and click it, its properties and events are displayed in the Object Inspector and the component becomes focussed in the Object TreeView.

1.4.8.Class Completion

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press **Ctrl+Shift+C**, or right-click and select **Complete Class** at Cursor. Delphi automatically adds **private read** and **write** specifiers to the declarations for any properties that require them, then creates skeleton 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 configure Class Completion, choose **Tools|Environment Options** and click the **Explorer** tab.

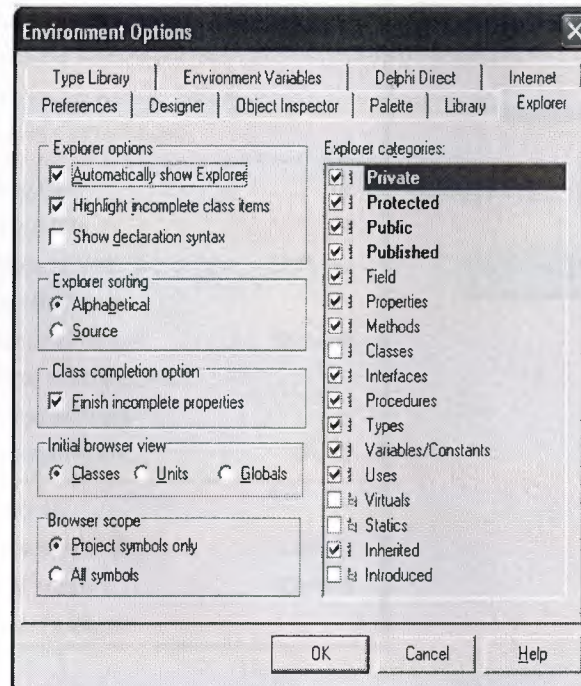
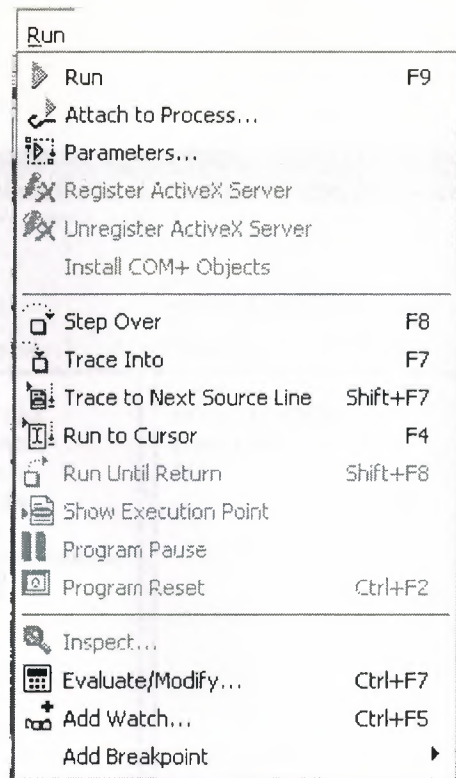


Fig.1.13.Class

1.4.9.Debugging applications

The IDE includes an integrated debugger that helps you locate and fix errors in your code. The debugger lets you control program execution, watch variables, and modify data values while your application is running. You can step through your code line by line, examining the state of the program at each breakpoint.



Choose any of the debugging commands from the Run menu.

Some commands are also available on the toolbar.



Figure 1.14. Run

To use the debugger, you must compile your program with debug information. Choose Project|Options, select the Compiler page, and check Debug Information. Then you can begin a debugging session by running the program from the IDE. To set debugger options, choose Tools|Debugger Options.

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. To learn how to combine debugging windows for more convenient use, see "Docking tool windows".

1.4.10. Exploring databases

The SQL Explorer (or Database Explorer in some editions of Delphi) lets you work directly with a remote database server during application development. For example, you can create, delete, or restructure tables, and you can import constraints while you are developing a database application.

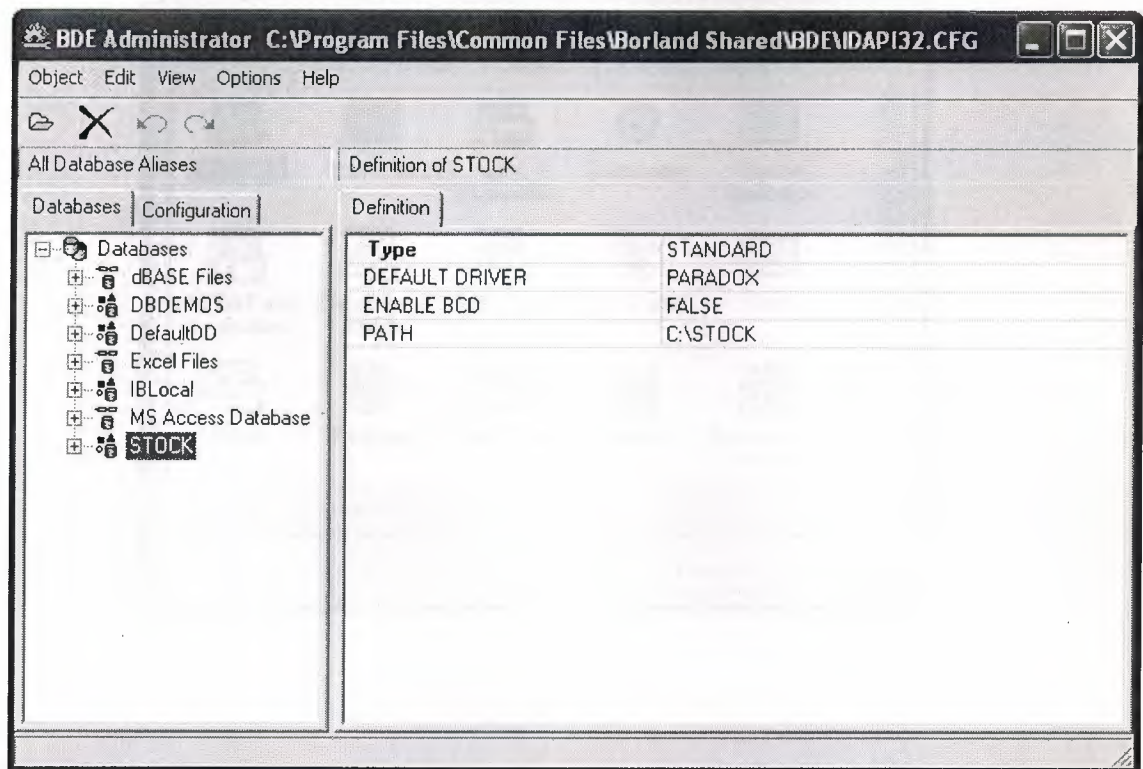


Figure 1.15.SQL Explorer

1.4.11. Templates and the 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 to display the New Items dialog when you begin a project. Check the Repository to see if it contains an object that resembles one you want to create.

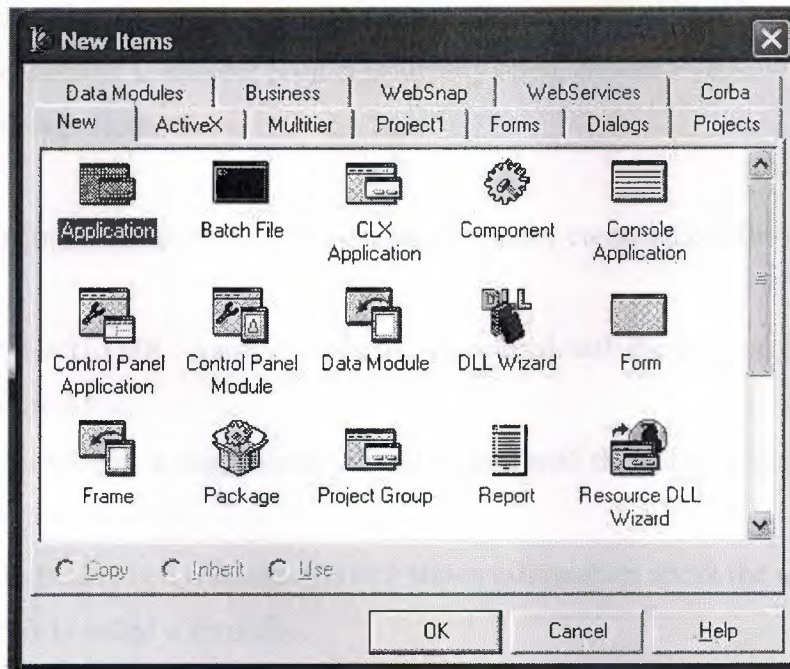


Figure 1.16.New Item

You can add your own objects to the Repository to facilitate reusing them and sharing them with other developers. Reusing objects lets you build families of applications with common user interfaces and functionality; building on an existing foundation also reduces development time and improves quality. The Object Repository provides a central location for tools that members of a development team can access over a network.

1.5.Programming With Delphi

The following section provide an overview of software development with Delphi.

1.5.1.Starting a New Application

Before beginning a new application, create a folder to hold the source files.

1. Create a folder called Seniha in the Projects directory off the main Delphi directory..
2. Open a new project.

Each application is represented by a project . When you start Delphi, it opens a blank project by default. If another project is already open, choose File|New Application to create a new project.

When you open a new project, Delphi automatically creates the following files.

- Project1.DPR : a source-code file associated with the project. This is called a project file.
- Unit1.PAS : a source-code file associated with the main project form. This is called a unit file.
- Unit1.DFM : a resource file that stores information about the main project form. This is called a form file.

Each form has its own unit and form files.

3. Choose File|Save All to save your files to disk. When the Save dialog appears, navigate to your Seniha folder and save each file using its default name.

Later on, you can save your work at any time by choosing File|Save All.

When you save your project, Delphi creates additional files in your project directory. You don't need to worry about them but don't delete them.

When you open a new project, Delphi displays the project's main form, named Form1 by default. You'll create the user interface and other parts of your application by placing components on this form.



Figure 1.17.Form Screen

The default form has maximize , minimize buttons and a close button , and a control menu

Next to the form, you'll see the Object Inspector, which you can use to set property values for the form and components you place on it.

The drop-down list at the top of the Object Inspector shows the current selected object.when an object is selected the Object Inspector show its properties.

1.5.1.1. Setting Property Values

When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called *design-time* settings.

For Example ; Set the background color of Form1 to Aqua.

Find the form's Color property in the Object Inspector and click the drop-down list displayed to the right of the property. Choose clAqua from the list.

1.5.2. Adding objects to the form

The Component palette represents components by icons grouped onto tabbed pages.

Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form.

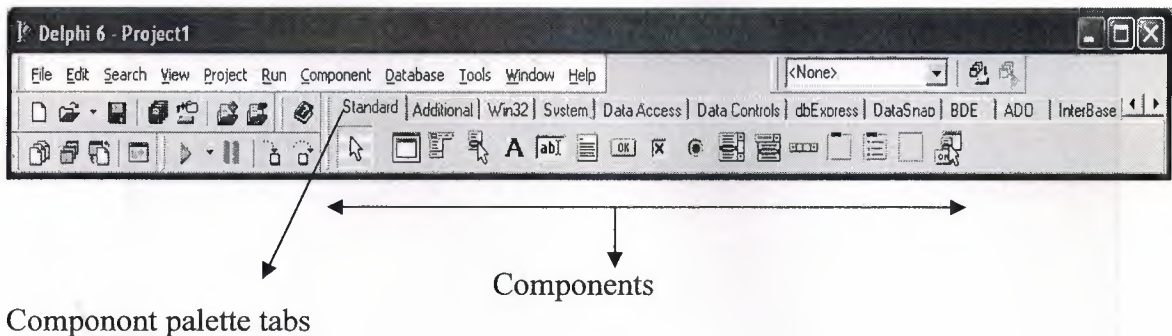


Figure 1.18.Standart Button

1.5.3.Add a Table and a StatusBar to the form:

Drop a Table component onto the form.

Click the BDE tab on the Component palette. To find the *Table* component, point at an icon on the palette for a moment; Delphi displays a Help hint showing the name of the component.

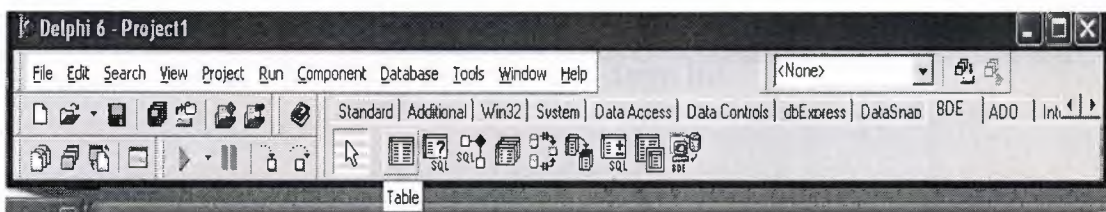


Fig.1.19.BDE Component palette

When you find the Table component, click it once to select it, then click on the form to place the component. The Table component is nonvisual, so it doesn't matter where you

put it. Delphi names the object Table1 by default. (When you point to the component on the form, Delphi displays its name--Table1--and the type of object it is--TTable.)

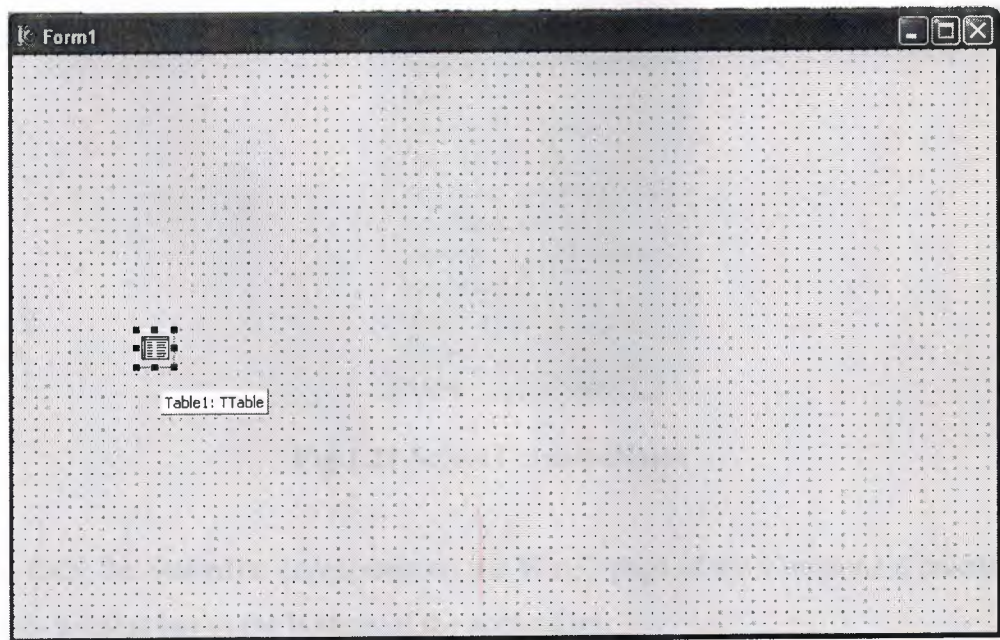


Figure 1.20.Table In The Form

Each Delphi component is a class; placing a component on a form creates an instance of that class. Once the component is on the form, Delphi generates the code necessary to construct an instance object when your application is running.

Set the DatabaseName property of Table1 to DBDEMOS. (DBDEMOS is an alias to the sample database that you're going to use.)

Select Table1 on the form, then choose the DatabaseName property in the Object Inspector. Select DBDEMOS from the drop-down list.

Active	True
AutoCalcFields	True
AutoRefresh	False
CachedUpdate	False
Constraints	(TCheckConstra
DatabaseName	STOCK
DefaultIndex	True
Exclusive	False
FieldDefs	(TFieldDefs)
Filter	
Filtered	False
FilterOptions	()
IndexDefs	(TIndexDefs)
IndexFieldName	
IndexFiles	(TIndexFiles)
IndexName	
MasterFields	
MasterSource	
Name	Table1

Fig.1.21.Select DatabaseName

Double-click the StatusBar component on the Win32 page of the Component palette. This adds a status bar to the bottom of the application.

Set the AutoHint property of the status bar to True. The easiest way to do this is to double-click on False next to AutoHint in the Object Inspector. (Setting AutoHint to True allows Help hints to appear in the status bar at runtime.)

1.5.4. Connecting to a Database

The next step is to add database controls and a DataSource to your form.

1. From the Data Access page of the Component palette, drop a DataSource component onto the form. The DataSource component is nonvisual, so it doesn't matter where you put it on the form. Set its DataSet property to Table1.
2. From the Data Controls page, choose the DBGrid component and drop it onto your form. Position it in the lower left corner of the form above the status bar, then expand it by dragging its upper right corner.

If necessary, you can enlarge the form by dragging its lower right corner. Your form should now resemble the following figure :

The Data Control page on Component palette holds components that let you view database tables.

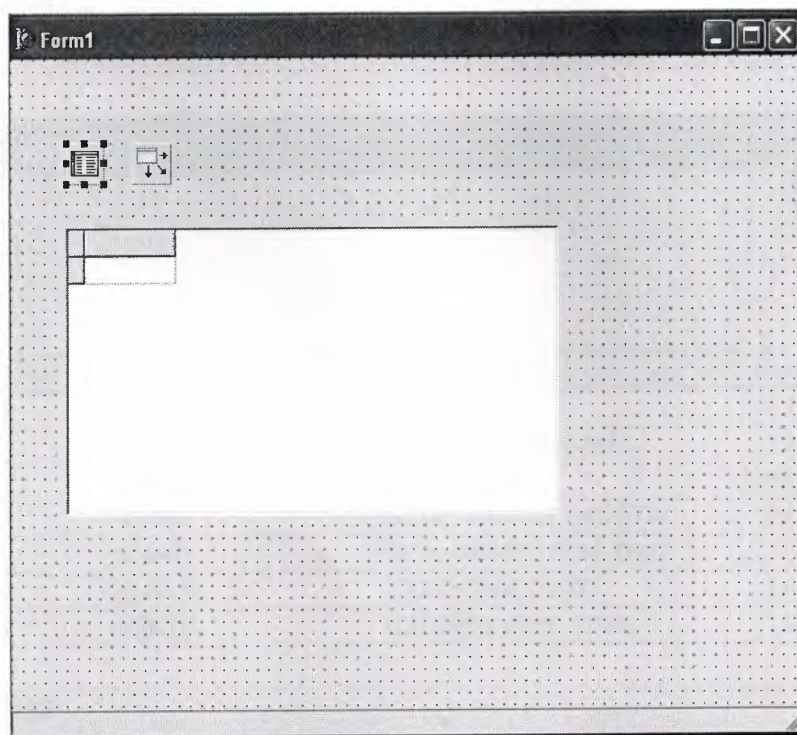


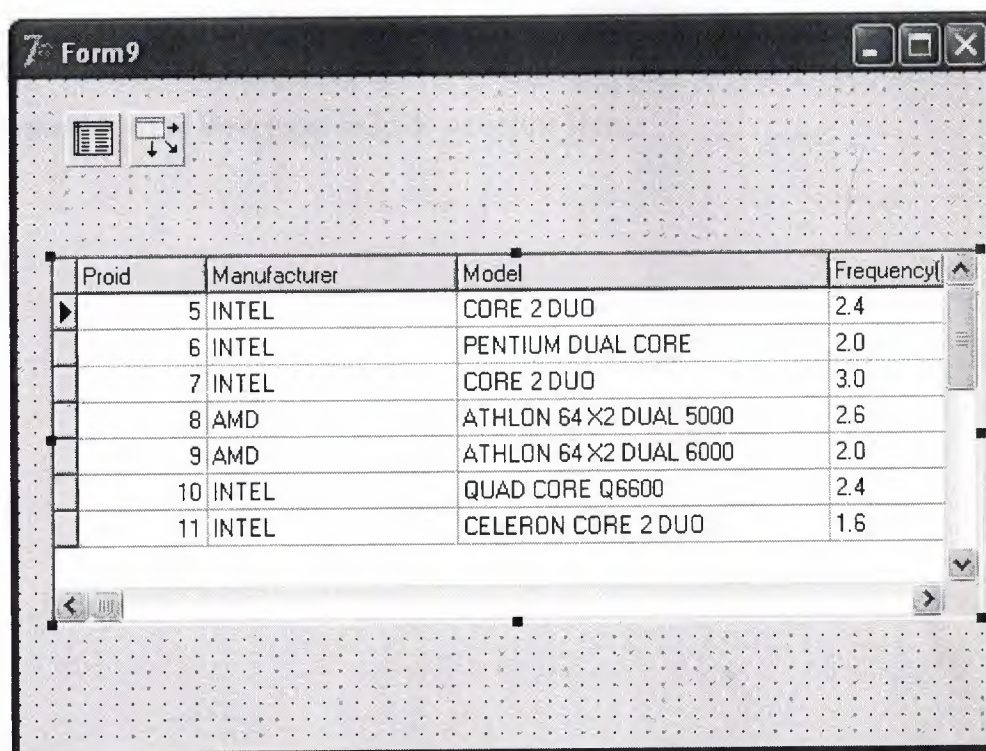
Figure 1.22.DBGrid In The Form

3. Set DBGrid properties to align the grid with the form. Double-click Anchors in the Object Inspector to display `akLeft`, `akTop`, `akRight`, and `akBottom`; set them all to True.
4. Set the `DataSource` property of DBGrid to `DataSource1` (the default name of the `DataSource` component you just added to the form).

Now you can finish setting up the *Table1* object you placed on the form earlier.

5. Select the Table1 object on the form, then set its TableName property to BIOLIFE.DB. (Name is still Table1.) Next, set the Active property to True.

When you set Active to True, the grid fills with data from the BIOLIFE.DB database table. If the grid doesn't display data, make sure you've correctly set the properties of all the objects on the form, as explained in the instructions above. (Also verify that you copied the sample database files into your ...\Borland Shared\Data directory when you installed Delphi.)



Proid	Manufacturer	Model	Frequency
5	INTEL	CORE 2 DUO	2.4
6	INTEL	PENTIUM DUAL CORE	2.0
7	INTEL	CORE 2 DUO	3.0
8	AMD	ATHLON 64 X2 DUAL 5000	2.6
9	AMD	ATHLON 64 X2 DUAL 6000	2.0
10	INTEL	QUAD CORE Q6600	2.4
11	INTEL	CELERON CORE 2 DUO	1.6

Figure 1.23.Show Table

The DBGrid control displays data at design time, while you are working in the IDE. This allows you to verify that you've connected to the database correctly. You cannot, however, edit the data at design time; to edit the data in the table, you'll have to run the application.

6. Press F9 to compile and run the project. (You can also run the project by clicking the Run button on the Debug toolbar, or by choosing Run from the Run menu.)
7. In connecting our application to a database, we've used three components and several levels of indirection. A data-aware control (in this case, a DBGrid)

points to a DataSource object, which in turn points to a dataset object (in this case, a Table). Finally, the dataset (Table1) points to an actual database table (BIOLIFE), which is accessed through the BDE alias DBDEMOS. (BDE aliases are configured through the BDE Administrator.)



This architecture may seem complicated at first, but in the long run it simplifies development and maintenance. For more information, see "Developing database applications" in the Developer's Guide or online Help.

CHAPTER 2

2.1 INTRODUCTION TO DATABASE

A database is an organized collection of data. The term originated within the computer industry, but its meaning has been broadened by popular use to the extent that the European Database Directive includes non-electronic databases within its definition. This article is confined to a more technical use of the term; though even amongst computing professionals some attach a much wider meaning to the word than others.

One possible definition is that a database is a collection of records stored in a computer in a systematic way, so that a computer program can consult it to answer questions. For better retrieval and sorting, each record is usually organized as a set of data elements. The items retrieved in answer to queries become information that can be used to make decisions. The computer program used to manage and query a database is known as a database management system (DBMS). The properties and design of database system are included in the study of information science.

The central concept of a database is that of a collection of records, or pieces of knowledge. Typically, for a given database, there is a structural description of the type of facts held in that database: this description is known as a schema. The schema describes the objects that are represented in the database, and the relationships among them. There are a number of different ways of organizing a schema, that is, of modeling the database structure: these are known as database models (or data models). The model in most common use today is the relational model, which in layman's terms represents all information in the form of multiple related tables each consisting of rows and columns (the true definition uses mathematical terminology). This model represents relationships by the use of values common to more than one table. Other models such as the hierarchical model and the network model use a more explicit representation of relationships.

The term database refers to the collection of related records, and the software should be referred to as the database management system or DBMS. When the context is unambiguous, however, many database administrators and programmers use the term database to cover both meanings.

Many professionals would consider a collection of data to constitute a database only if it has certain properties: for example, if the data is managed to ensure its integrity and quality, if it allows shared access by a community of users, if it has a schema, or if it supports a query language. However, there is no agreed definition of these properties.

Database management systems are usually categorized according to the data model that they support: relational, object-relational, network, and so on. The data model will tend to determine the query languages that are available to access the database. A great deal of the internal engineering of a DBMS, however, is independent of the data model, and is concerned with managing factors such as performance, concurrency, integrity, and recovery from hardware failures. In these areas there are large differences between products.

2.2 HISTORY

The earliest known use of the term 'data base' was in June 1963, when the System Development Corporation sponsored a symposium under the title Development and Management of a Computer-centered Data Base. Database as a single word became common in Europe in the early 1970s and by the end of the decade it was being used in major American newspapers. (Databank, a comparable term, had been used in the Washington Post newspaper as early as 1966.)

The first database management systems were developed in the 1960s. A pioneer in the field was Charles Bachman. Bachman's early papers show that his aim was to make more effective use of the new direct access storage devices becoming available: until then, data processing had been based on punched cards and magnetic tape, so that serial processing was the dominant activity. Two key data models arose at this time: CODASYL developed the network model based on Bachman's ideas, and (apparently independently) the hierarchical model was used in a system developed by North American Rockwell, later adopted by IBM as the cornerstone of their IMS product.

The relational model was proposed by E. F. Codd in 1970. He criticized existing models for confusing the abstract description of information structure with descriptions of physical access mechanisms. For a long while, however, the relational model remained of academic interest only. While CODASYL systems and IMS were conceived as practical engineering solutions taking account of the technology as it existed at the time, the relational model took a

much more theoretical perspective, arguing (correctly) that hardware and software technology would catch up in time. Among the first implementations were Michael Stonebraker's Ingres at Berkeley, and the System R project at IBM. Both of these were research prototypes, announced during 1976. The first commercial products, Oracle and DB2, did not appear until around 1980. The first successful database product for microcomputers was dBASE for the CP/M and PC-DOS/MS-DOS operating systems.

During the 1980s, research activity focused on distributed database systems and database machines, but these developments had little effect on the market. Another important theoretical idea was the Functional Data Model, but apart from some specialized applications in genetics, molecular biology, and fraud investigation, the world took little notice.

In the 1990s, attention shifted to object-oriented databases. These had some success in fields where it was necessary to handle more complex data than relational systems could easily cope with, such as spatial databases, engineering data (including software engineering repositories), and multimedia data. Some of these ideas were adopted by the relational vendors, who integrated new features into their products as a result.

The 2000s, the fashionable area for innovation is the XML database. As with object databases, this has spawned a new collection of startup companies, but at the same time the key ideas are being integrated into the established relational products. XML databases aim to remove the traditional divide between documents and data, allowing all of an organization's information resources to be held in one place, whether they are highly structured or not.

2.3 DATABASE MODELS

Various techniques are used to model data structure. Most database systems are built around one particular data model, although it is increasingly common for products to offer support for more than one model. For any one logical model various physical implementations may be possible, and most products will offer the user some level of control in tuning the physical implementation, since the choices that are made have a significant effect on performance. An example of this is the relational model: all serious implementations of the relational model allow the creation of indexes which provide fast access to rows in a table if the values of certain columns are known.

A data model is not just a way of structuring data: it also defines a set of operations that can be performed on the data. The relational model, for example, defines operations such as select, project, and join. Although these operations may not be explicit in a particular query language, they provide the foundation on which a query language is built.

2.3.1 Flat model

This may not strictly qualify as a data model, as defined above. The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password that might be used as a part of a system security database. Each row would have the specific password associated with an individual user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is, incidentally, a basis of the spreadsheet.

2.3.2 Hierarchical model

In a hierarchical model, data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list. Hierarchical structures were widely used in the early mainframe database management systems, such as the Information Management System (IMS) by IBM, and now describe the structure of XML documents. This structure allows one 1:N relationship between two types of data. This structure is very efficient to describe many relationships in the real world; recipes, table of contents, ordering of paragraphs/verses, any nested and sorted information. However, the hierarchical structure is inefficient for certain database operations when a full path (as opposed to upward link and sort field) is not also included for each record.

2.3.3 Network model

The network model (defined by the CODASYL specification) organizes data using two fundamental constructs, called *records* and *sets*. Records contain fields (which may be organized hierarchically, as in the programming language COBOL). Sets (not to be confused with mathematical sets) define one-to-many relationships between records: one owner, many members. A record may be an owner in any number of sets, and a member in any number of sets.

The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.

Although it is not an essential feature of the model, network databases generally implement the set relationships by means of pointers that directly address the location of a record on disk. This gives excellent retrieval performance, at the expense of operations such as database loading and reorganization.

2.3.4 Relational model

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

The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model defined by Codd. The data structures in these products are tables, rather than relations: the main differences being that tables can contain duplicate rows, and that the rows (and columns) can be treated as being ordered. The same criticism applies to the SQL language which is the primary interface to these products. There has been considerable controversy, mainly due to Codd himself, as to whether it is correct to describe SQL implementations as "relational": but the fact is that the world does so, and the following description uses the term in its popular sense.

A relational database contains multiple tables, each similar to the one in the "flat" database model. Relationships between tables are not defined explicitly; instead, *keys* are used to match up rows of data in different tables. A key is a collection of one or more columns in one table whose values match corresponding columns in other tables: for example, an *Employee* table may contain a column named *Location* which contains a value that matches the key of a *Location* table. Any column can be a key, or multiple columns can be grouped together into a single key. It is not necessary to define all the keys in advance; a column can be used as a key even if it was not originally intended to be one.

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

A key that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number) is sometimes called a "natural" key. If no natural key is suitable (think of the many people named *Brown*), an arbitrary key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that cannot break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

2.3.4.1 Relational operations

Users (or programs) request data from a relational database by sending it a query that is written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it is much more common for SQL queries to be embedded into software that provides an easier user interface. Many web sites, perform SQL queries when generating pages.

In response to a query, the database returns a result set, which is just a list of rows containing the answers. The simplest query is just to return all the rows from a table, but more often, the rows are filtered in some way to return just the answer wanted.

Often, data from multiple tables are combined into one, by doing a join. Conceptually, this is done by taking all possible combinations of rows (the Cartesian product), and then filtering out everything except the answer. In practice, relational database management systems rewrite ("optimize") queries to perform faster, using a variety of techniques.

There are a number of relational operations in addition to join. These include project (the process of eliminating some of the columns), restrict (the process of eliminating some of the rows), union (a way of combining two tables with similar structures), difference (which lists the rows in one table that are not found in the other), intersect (which lists the rows found in

both tables), and product (mentioned above, which combines each row of one table with each row of the other). Depending on which other sources you consult, there are a number of other operators - many of which can be defined in terms of those listed above. These include semi-join, outer operators such as outer join and outer union, and various forms of division. Then there are operators to rename columns, and summarizing or aggregating operators, and if you permit relation values as attributes (RVA - relation-valued attribute), then operators such as group and ungroup. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for decades. This has made the idea and implementation of relational databases very popular with businesses.

2.3.5 Dimensional model

The dimensional model is a specialized adaptation of the relational model used to represent data in data warehouses in a way that data can be easily summarized using OLAP queries. In the dimensional model, a database consists of a single large table of facts that are described using dimensions and measures. A dimension provides the context of a fact (such as who participated, when and where it happened, and its type) and is used in queries to group related facts together. Dimensions tend to be discrete and are often hierarchical; for example, the location might include the building, state, and country. A measure is a quantity describing the fact, such as revenue. It's important that measures can be meaningfully aggregated - for example, the revenue from different locations can be added together.

In an OLAP query, dimensions are chosen and the facts are grouped and added together to create a summary.

The dimensional model is often implemented on top of the relational model using a star schema, consisting of one table containing the facts and surrounding tables containing the dimensions. Particularly complicated dimensions might be represented using multiple tables, resulting in a snowflake schema.

A data warehouse can contain multiple star schemas that share dimension tables, allowing them to be used together. Coming up with a standard set of dimensions is an important part of dimensional modeling.

2.3.6 Object database models

In recent years, the object-oriented paradigm has been applied to database technology, creating a new programming model known as object databases. These databases attempt to bring the database world and the application programming world closer together, in particular by ensuring that the database uses the same type system as the application program. This aims to avoid the overhead (sometimes referred to as the *impedance mismatch*) of converting information between its representation in the database (for example as rows in tables) and its representation in the application program (typically as objects). At the same time object databases attempt to introduce the key ideas of object programming, such as encapsulation and polymorphism, into the world of databases.

A variety of these ways have been tried for storing objects in a database. Some products have approached the problem from the application programming end, by making the objects manipulated by the program persistent. This also typically requires the addition of some kind of query language, since conventional programming languages do not have the ability to find objects based on their information content. Others have attacked the problem from the database end, by defining an object-oriented data model for the database, and defining a database programming language that allows full programming capabilities as well as traditional query facilities.

Object databases suffered because of a lack of standardization: although standards were defined by ODMG, they were never implemented well enough to ensure interoperability between products. Nevertheless, object databases have been used successfully in many applications: usually specialized applications such as engineering databases or molecular biology databases rather than mainstream commercial data processing. However, object database ideas were picked up by the relational vendors and influenced extensions made to these products and indeed to the SQL language.

2.4 DATABASE INTERNALS

2.4.1 Indexing

All of these kinds of database can take advantage of indexing to increase their speed, and this technology has advanced tremendously since its early uses in the 1960s and 1970s. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Various methods of indexing are commonly used; B-trees, hashes, and linked lists are all common indexing techniques.

Relational DBMSs have the advantage that indexes can be created or dropped without changing existing applications making use of it. The database chooses between many different strategies based on which one it estimates will run the fastest. In other words, indexes are transparent to the application or end user querying the database; while they affect performance, any SQL command will run with or without indexes existing in the database.

Relational DBMSs utilize many different algorithms to compute the result of an SQL statement. The RDBMS will produce a plan of how to execute the query, which is generated by analyzing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are Nested Loops Join, Sort-Merge Join and Hash Join. Which of these is chosen depends on whether an index exists, what type it is, and its cardinality.

2.4.2 Transactions and concurrency

In addition to their data model, most practical databases ("transactional databases") attempt to enforce a database transaction model that has desirable data integrity properties. Ideally, the database software should enforce the ACID rules, summarized here:

Atomicity: Either all the tasks in a transaction must be done, or none of them. The transaction must be completed, or else it must be undone (rolled back).

Consistency: Every transaction must preserve the integrity constraints — the declared consistency rules — of the database. It cannot place the data in a contradictory state.

Isolation: Two simultaneous transactions cannot interfere with one another. Intermediate results within a transaction are not visible to other transactions.

Durability: Completed transactions cannot be aborted later or their results discarded. They must persist through (for instance) restarts of the DBMS after crashes

In practice, many DBMS's allow most of these rules to be selectively relaxed for better performance.

Concurrency control is a method used to ensure that transactions are executed in a safe manner and follow the ACID rules. The DBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions.

2.4.3 Replication

Replication of databases is closely related to transactions. If a database can log its individual actions, it is possible to create a duplicate of the data in real time. The duplicate can be used to improve performance or availability of the whole database system. Common replication concepts include:

Master/Slave Replication: All write requests are performed on the master and then replicated to the slaves

Quorum: The result of Read and Write requests is calculated by querying a "majority" of replicas.

Multimaster: Two or more replicas sync each other via a transaction identifier.

2.5 APPLICATIONS OF DATABASES

Databases are used in many applications, spanning virtually the entire range of computer software. Databases are the preferred method of storage for large multi user applications, where coordination between many users is needed. Even individual users find them convenient, though, and many electronic mail programs and personal organizers are based on standard database technology. Software database drivers are available for most database platforms so that application software can use a common application programming interface (API) to retrieve the information stored in a database. Two commonly used database APIs are JDBC and ODBC. A database is also a place where you can store data and then arrange that data easily and efficiently.

CHAPTER 3

3.Description of Project

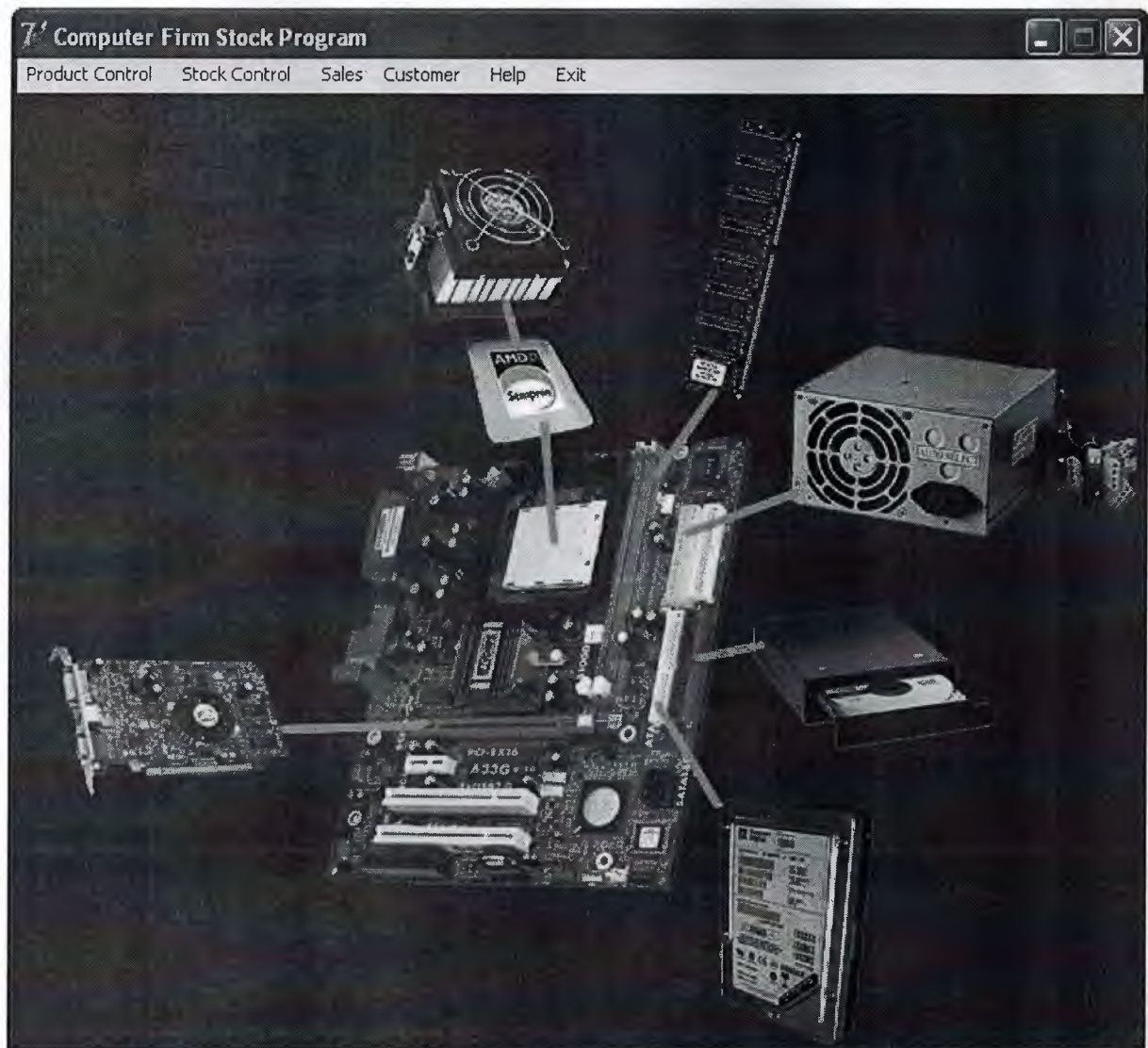


Figure 3.1.Main Menu

3.1.Main Menu

Main form contains the main menu which include your requirement. It has product control,stock control,sales,customer,help and exit buttons. This main menu has submenus.You can reach every form of this application with the main menu and submenus.

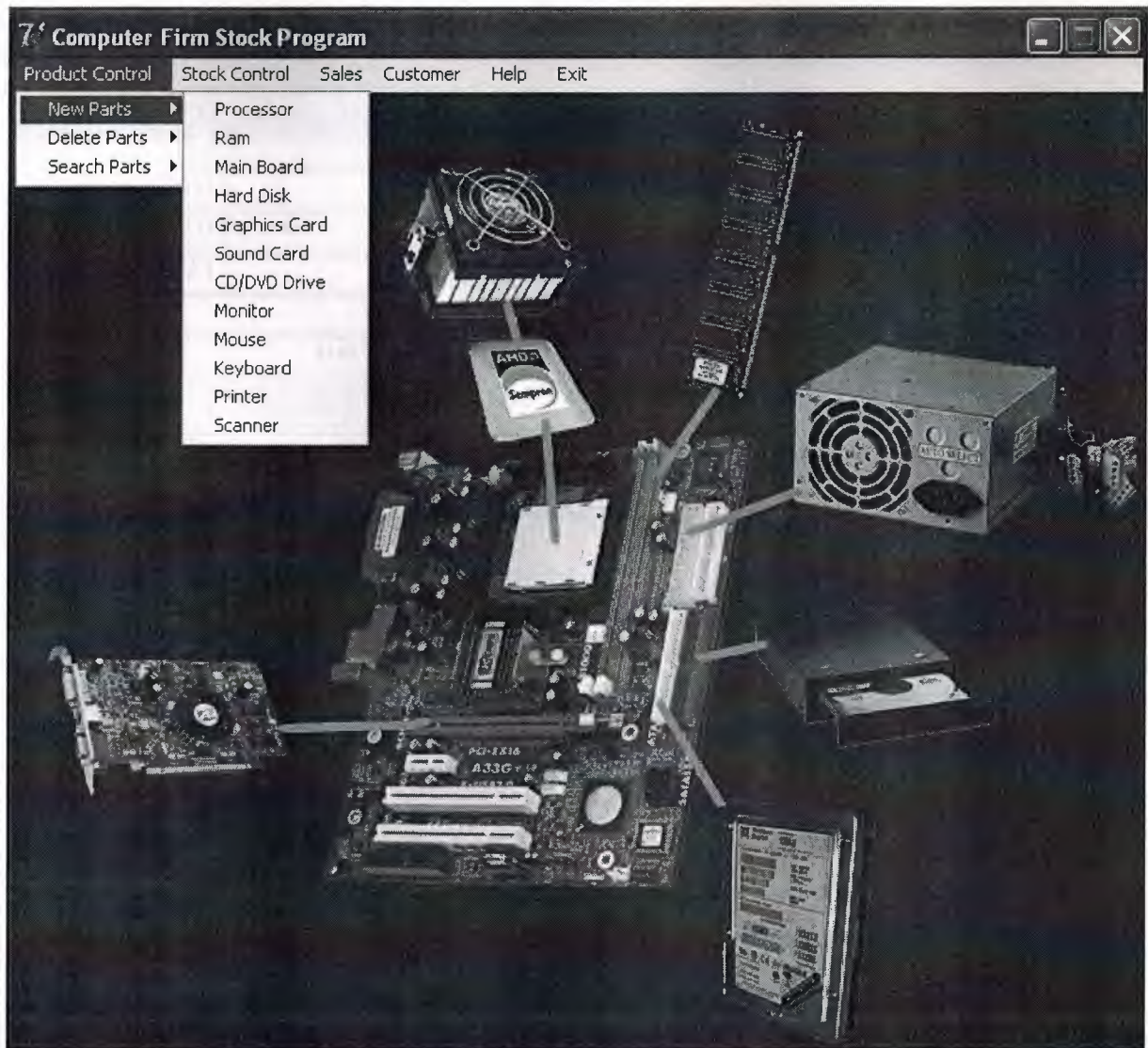


Figure 3.2. Submenuss of Product Control

If you click new parts and after any product from the main menu, you can reach the form of adding any product.

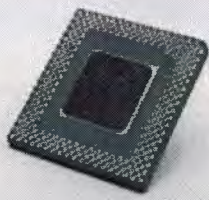
3.1.1.Add New Product

Processor

Exit

Manufacturer: INTEL
Model: CORE 2 DUO
Frequency: 2.4
Amount: 10
Price: \$140.00

New Add Product Clear

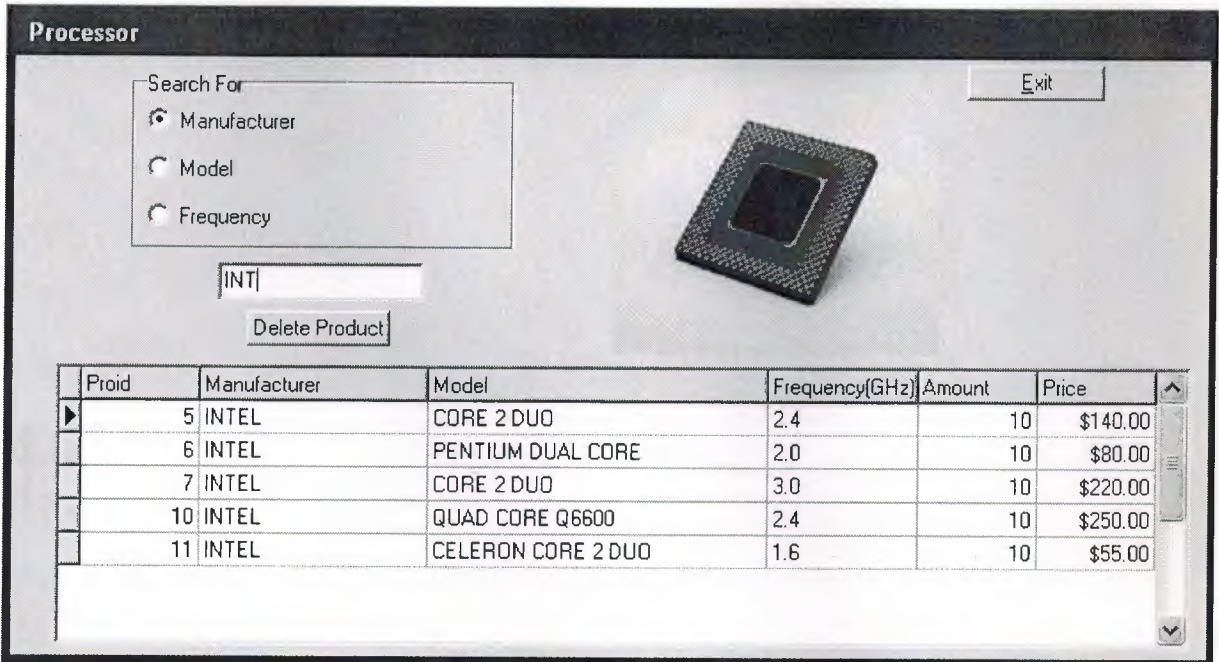


Proid	Manufacturer	Model	Frequency(GHz)	Amount	Price
5	INTEL	CORE 2 DUO	2.4	10	\$140.00
6	INTEL	PENTIUM DUAL CORE	2.0	10	\$80.00
7	INTEL	CORE 2 DUO	3.0	10	\$220.00
8	AMD	ATHLON 64 X2 DUAL 5000	2.6	10	\$110.00
9	AMD	ATHLON 64 X2 DUAL 6000	2.0	10	\$200.00
10	INTEL	QUAD CORE Q6600	2.4	10	\$250.00
11	INTEL	CELERON CORE 2 DUO	1.6	10	\$55.00

Figure 3.1.1.Add New Product

When you click the processor from the New Parts you see the form which you can add new processor to the database. When you click New button on the form database creates a new row for the table. If you enter the Manufacturer, Model, Frequency, Amount and the Price of the Processor and click the Add Product button, your processor is posted to the database. If you want to clear the edit boxes you can click the Clear button. But if you want to add a new processor you have to click New button again. Because you can't post any new processor if you don't insert any row.

3.1.2.Delete Product



The screenshot shows a window titled "Processor". It contains a "Search For" section with three radio buttons: "Manufacturer" (selected), "Model", and "Frequency". Below these is a text input field containing "INT" and a "Delete Product" button. To the right is an "Exit" button and an image of a processor. At the bottom is a table with the following data:

Proid	Manufacturer	Model	Frequency(GHz)	Amount	Price
5	INTEL	CORE 2 DUO	2.4	10	\$140.00
6	INTEL	PENTIUM DUAL CORE	2.0	10	\$80.00
7	INTEL	CORE 2 DUO	3.0	10	\$220.00
10	INTEL	QUAD CORE Q6600	2.4	10	\$250.00
11	INTEL	CELERON CORE 2 DUO	1.6	10	\$55.00

Figure 3.1.2.Delete Product

If you want to delete any processor you have to make a search to find the processor which you want to delete. In this form we have a Search For radio group component. This component allows you to select the search criteria. You can search by Manufacturer, Model, Frequency of the product. After the selection of the search criteria we write the key words of the product. When you enter the characters of the word which you want to search, the search engine starts to search. You don't have to write all the characters of the word or you don't have to click any button. After these you select the product which you want to delete and click Delete Product button.

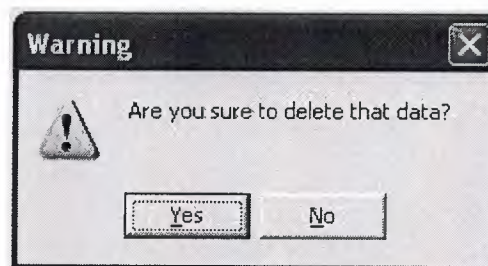


Figure 3.1.3. Warning Message

When you click to the Delete Product button the program asks you 'Are you sure to delete that data?'. If you want to delete you click Yes button, if you don't, you click the No button.

3.1.3.Search Product

Processor

Search For

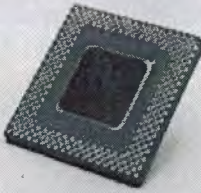
Manufacturer

Model

Frequency

AT

Exit



	Proid	Manufacturer	Model	Frequency(GHz)	Amount	Price
▶	8	AMD	ATHLON 64 X2 DUAL 5000	2.6	10	\$110.00
	9	AMD	ATHLON 64 X2 DUAL 6000	2.0	10	\$200.00

Figure 3.1.4.Search Product

This form is the same with the Delete Product form(Figure 3.1.2).But you can not delete a product.You can only search in this form.

3.2. Stock Control

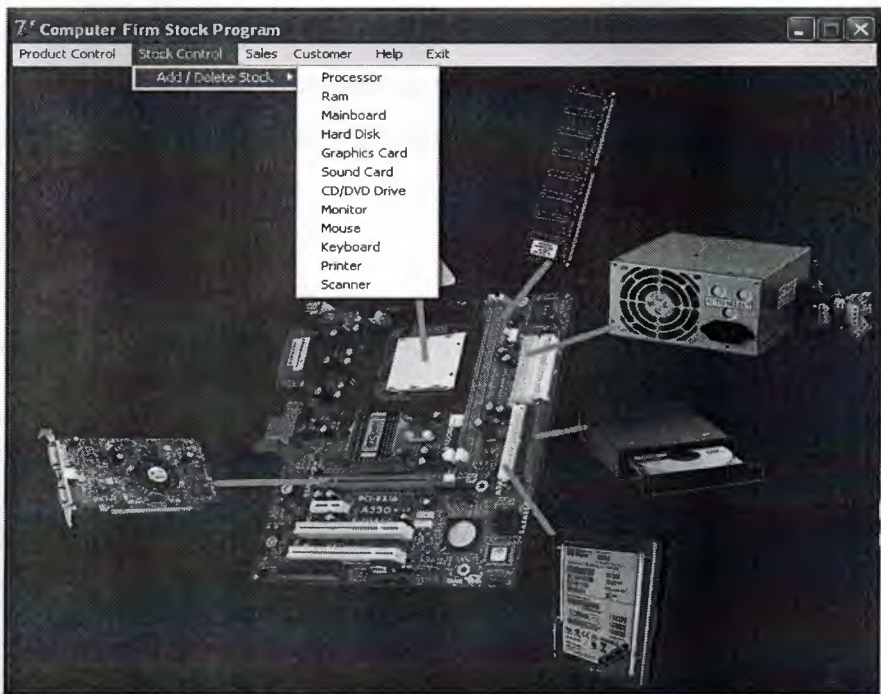


Figure 3.2.1. Stock Control Menu

From this menu you can reach the Stock Control Form and you can only change the amount of any product. You can increase the or decrease the amount of the product but you can't change any other data of the product. This menu allows you to increase or decrease of the products amounts without any sales.

Processor_Stock_Control

Search for:

- ☒ Manufacturer
- ☐ Model
- ☐ Frequency

Search:

Amount:

Proid	Manufacturer	Model	Frequency(GHz)	Amount	Price
5	INTEL	CORE 2 DUO	2.4	12	\$
6	INTEL	PENTIUM DUAL CORE	2.0	10	\$
7	INTEL	CORE 2 DUO	3.0	10	\$
10	INTEL	QUAD CORE Q6600	2.4	10	\$

Figure 3.2.2. Stock Control Form

In Stock Control Form we have a radio group for search. First of all we select the search criteria. You can search by Manufacturer, Model or Frequency. After that we are entering the key words which we are searching to the search edit box. After we find the product we enter the amount value which we want to increase or decrease from the stock. If we click Add Stock button, the value which we entered is adding to the products amount, or if we click Delete From Stock button, the value which we entered is subtracting from the products amount.

3.3. Sell Product

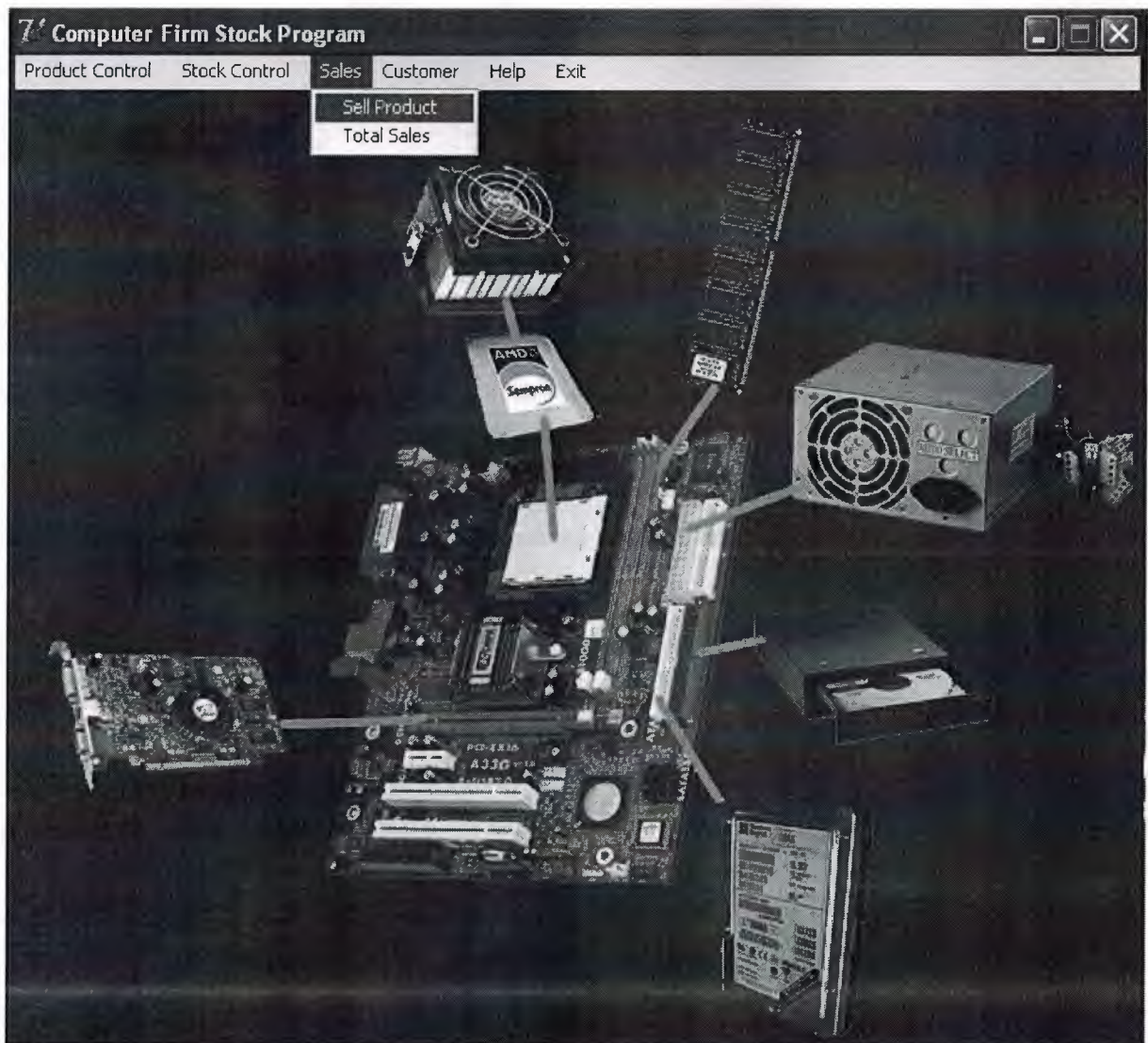


Figure 3.3.1. Sales Menu

This Menu allows to reach the Sell and Total Sales Forms.

Exit

Surname: DIKICILER
Name: CEVDET
Cust ID: 1
Product Type: Ram

Ramid	Manufacturer	Model	Size(GB)	Frequency(MHz)	Amount	Price
3	KINGSTON	DDR2	2	667	7	\$50.0
4	KINGSTON	DDR2	1	667	10	\$25.0
5	KINGSTON	DDR2	1	533	10	\$23.0

Amount: 3 Add To List

Sell_ID	Cust_ID	Name	Surname	Product	Amount	Price	Total_Price	Date
16	1	CEVDET	DIKICILER	KINGSTON DDR2 2 GB 2 MHz	3	50	150	6/8/2008

Delete Selected Part Finish Selling Show Report

Figure 3.3.2.Sell Product Form

In Sell Product Form we have 4 combobox component. First combobox shows us the customers surnames. After selecting the surname of the customer in the second combobox shows us the name or names of the customers whose surname matches with in first combobox. The name and surname can be same with another person. To cover this problem we have to look at the customer ID of the person. Because this ID is unique. After these we are selecting the product type which we want to sell to the customer. For example if we select processor the first table shows us the processor table, if we select another one this table shows this table. After that we are selecting to the product which we want to sell, from the table. We have a textbox middle of the two tables. This is for arranging the amount of the product to sell. When you click to the Add To List button the product which we selected is copied to the customers buying table. In the first table, the amount of the product which we select will be decrease. In the second table the amount and the price of the product is multiplied and the total price will be found. In the last column of the second table we have the Sale Date. If we made a wrong sale we can cancel this sale by clicking the Delete Selected Part. If we click Finish Selling this sales will be record to the total sales table which we can't see now.

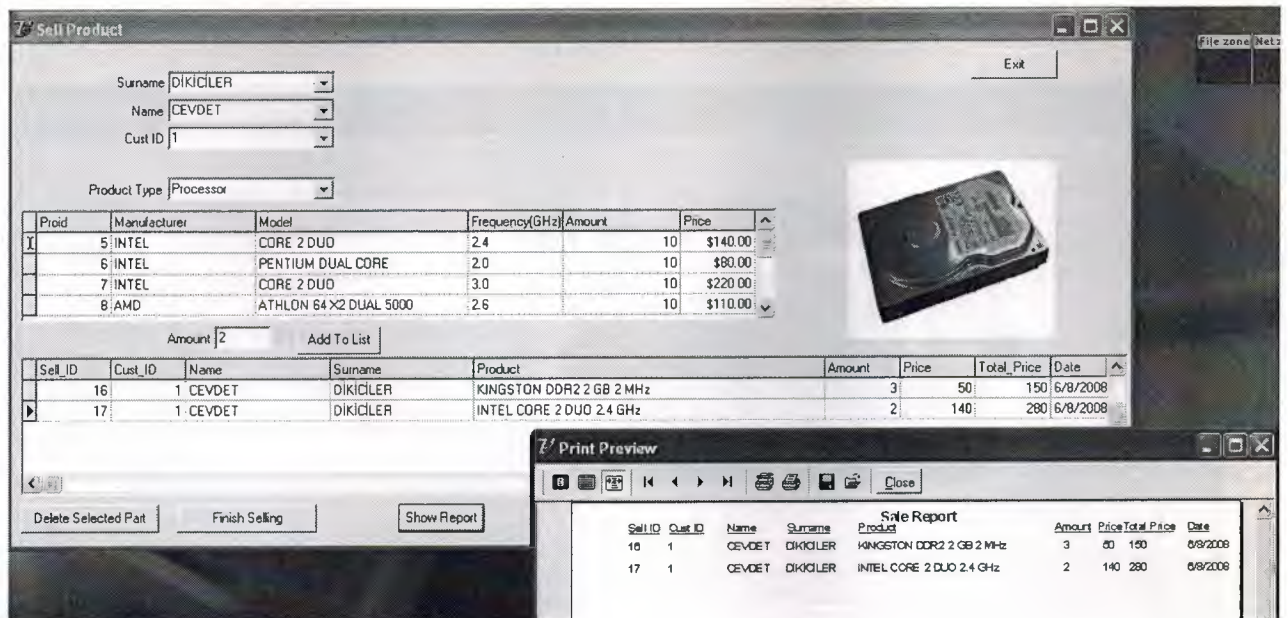


Figure 3.3.3.Show Report of Sale

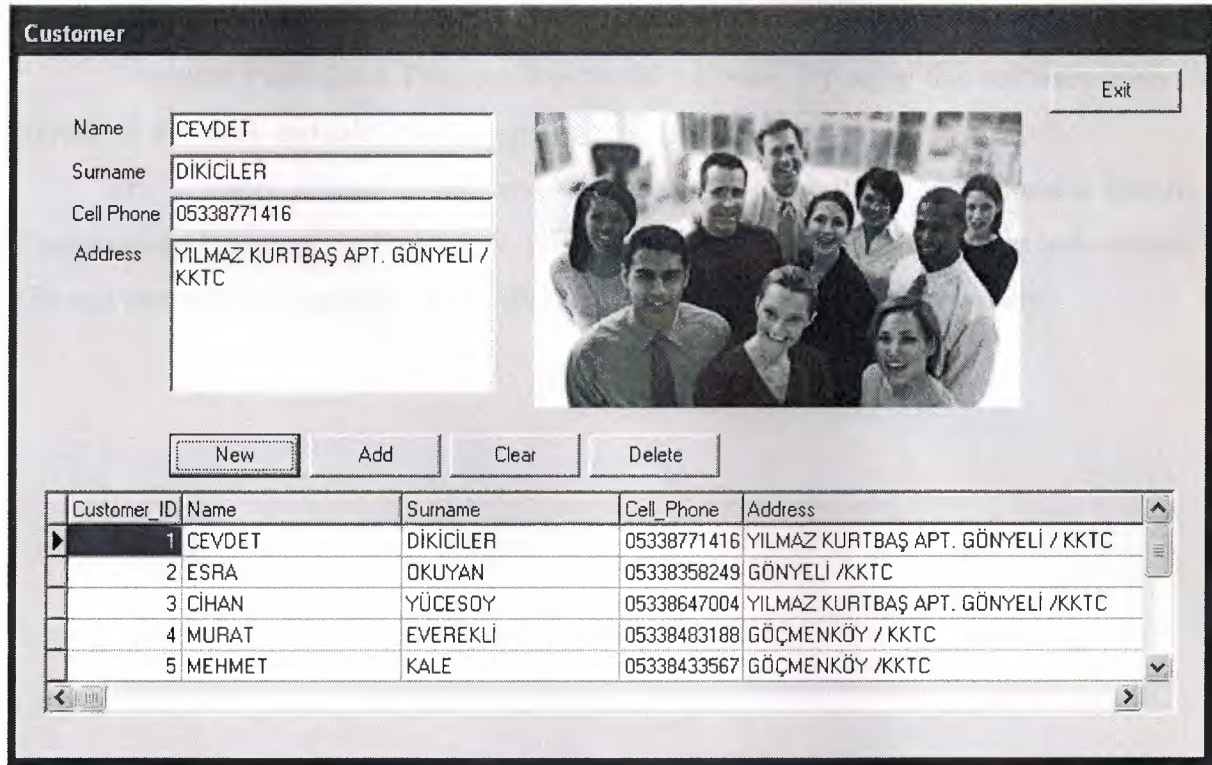
If you want to print or preview this sale you click the Show Report button and you will see a new form which has the sale information. Now, from this form you can print or you can save this report if you want.

3.4. Customer Menu



Figure 3.4.1.Customer Menu

You can reach the Customer Add / Delete / Search form from this menu.



Customer_ID	Name	Surname	Cell_Phone	Address
1	CEVDET	DİKİCİLER	05338771416	YILMAZ KURTBAŞ APT. GÖNYELİ / KKTC
2	ESRA	OKUYAN	05338358249	GÖNYELİ /KKTC
3	CIHAN	YÜCESOY	05338647004	YILMAZ KURTBAŞ APT. GÖNYELİ /KKTC
4	MURAT	EVEREKLİ	05338483188	GÖÇMENKÖY / KKTC
5	MEHMET	KALE	05338433567	GÖÇMENKÖY /KKTC

Figure 3.4.2.Customer Form

In this form we are adding, deleting and searching the customer informations. Before adding this information we have to click New button. Because we need a new empty row for adding these informations. After all the needed areas filled we click to the Add button to save the customer informations to the database. If we make a mistake we can delete the selected row by clicking the Delete button. If we want to clear the information areas we click to the clear button.

CONCLUSION

Computer Parts Stock Program is a useful software. By using this software users can control all the stock and sale informations.

The software is easy in use, and everything is in detail, we used borland Delphi 7 Programming Language in building it, also PARADOX Database for storing information's. The software can have mistakes.If we find,we will cover all the mistakes rapidly.

REFERENCES

- [1] Memik YANIK, Borland Delphi6, Beta A.Ş., Istanbul, 2002.
- [2] Guide for searching reference softwares
“<http://www.google.com>”.
- [3] Guide for some source codes
“<http://www.delphiturkiye.com>”.
- [4] Guide for “What is Delphi”
“<http://www.wikipedia.org>”.
- [5] Guide for some codes
“<http://www.delphiturk.com>”.
- [6] Guide for some codes
“<http://www.programlama.com>”.

APPENDIX 1

Program Codes

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Product_Control_Processor},
  Unit3 in 'Unit3.pas' {Processor_Stock_Control},
  Unit4 in 'Unit4.pas' {Product_Control_Ram},
  Unit5 in 'Unit5.pas' {Sell},
  Unit6 in 'Unit6.pas' {Customer},
  Unit7 in 'Unit7.pas' {Total_Sales},
  Unit8 in 'Unit8.pas' {Sale_Report};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TProduct_Control_Processor, Product_Control_Processor);
  Application.CreateForm(TProcessor_Stock_Control, Processor_Stock_Control);
  Application.CreateForm(TProduct_Control_Ram, Product_Control_Ram);
  Application.CreateForm(TSell, Sell);
  Application.CreateForm(TCustomer, Customer);
  Application.CreateForm(TTotal_Sales, Total_Sales);
  Application.CreateForm(TSale_Report, Sale_Report);
  Application.Run;
end.
```

```
unit Unit1;
```

```
interface
```

```
uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, jpeg, ExtCtrls;
```

```
type
```

```
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    St1: TMenuItem;
    DeleteStock1: TMenuItem;
    Ram1: TMenuItem;
```


Processor2: TMenuItem;
Ram2: TMenuItem;
MainBoard2: TMenuItem;
HardDisk1: TMenuItem;
Image1: TImage;
DeleteMain1: TMenuItem;
Processor1: TMenuItem;
SearchParts1: TMenuItem;
Processor3: TMenuItem;
AddStock1: TMenuItem;
Processor5: TMenuItem;
Help1: TMenuItem;
Exit1: TMenuItem;
Sell1: TMenuItem;
Processor7: TMenuItem;
Ram3: TMenuItem;
Ram4: TMenuItem;
Mainboard1: TMenuItem;
HardDisk2: TMenuItem;
Ram5: TMenuItem;
Mainboard1: TMenuItem;
HardDisk3: TMenuItem;
Ram6: TMenuItem;
Mainboard3: TMenuItem;
HardDik1: TMenuItem;
Customer1: TMenuItem;
Add1: TMenuItem;
GraphicsCard1: TMenuItem;
SoundCard1: TMenuItem;
CDDVDDrives1: TMenuItem;
Monitor1: TMenuItem;
Mouse1: TMenuItem;
Keyboard1: TMenuItem;
Printer1: TMenuItem;
Scanner1: TMenuItem;
GraphicsCard2: TMenuItem;
SoundCard2: TMenuItem;
CDDVDDrive1: TMenuItem;
Monitor2: TMenuItem;
Mouse2: TMenuItem;
Keyboard2: TMenuItem;
Printer2: TMenuItem;
Scanner2: TMenuItem;
GraphicsCard3: TMenuItem;
SoundCard3: TMenuItem;
CDDVDDrive2: TMenuItem;
Mpnitor1: TMenuItem;
Mouse3: TMenuItem;
Keyboard3: TMenuItem;
Printer3: TMenuItem;

```

Scanner3: TMenuItem;
GraphicsCard4: TMenuItem;
SoundCard4: TMenuItem;
CDDVDDrive3: TMenuItem;
Monitor3: TMenuItem;
Mouse4: TMenuItem;
Keyboard4: TMenuItem;
Printer4: TMenuItem;
Scanner4: TMenuItem;
About1: TMenuItem;
procedure Processor2Click(Sender: TObject);
procedure Processor1Click(Sender: TObject);
procedure Processor5Click(Sender: TObject);
procedure Processor3Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
procedure Ram2Click(Sender: TObject);
procedure Ram4Click(Sender: TObject);
procedure Ram5Click(Sender: TObject);
procedure Add1Click(Sender: TObject);
procedure Processor7Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4, Unit6, Unit5;

{$R *.dfm}

procedure TForm1.Processor2Click(Sender: TObject);
begin
  Form1.Hide;
  Product_Control_Processor.show;
  Product_Control_Processor.Label1.Visible:=true;
  Product_Control_Processor.Label2.Visible:=true;
  Product_Control_Processor.Label3.Visible:=true;
  Product_Control_Processor.Label4.Visible:=true;
  Product_Control_Processor.Label5.Visible:=true;

  Product_Control_Processor.DBEdit1.Visible:=true;
  Product_Control_Processor.DBEdit2.Visible:=true;
  Product_Control_Processor.DBEdit3.Visible:=true;
  Product_Control_Processor.DBEdit4.Visible:=true;
  Product_Control_Processor.DBEdit5.Visible:=true;

```

```
Product_Control_Processor.Button3.Visible:=true;
Product_Control_Processor.Button1.Visible:=true;
Product_Control_Processor.Button2.Visible:=true;

Product_Control_Processor.Button5.Visible:=false;
Product_Control_Processor.RadioGroup1.Visible:=false;
Product_Control_Processor.Edit1.Visible:=false;
Product_Control_Processor.Query1.Insert;
```

```
end;
```

```
procedure TForm1.Processor1Click(Sender: TObject);
begin
```

```
Form1.Hide;
Product_Control_Processor.show;
Product_Control_Processor.Label1.Visible:=False;
Product_Control_Processor.Label2.Visible:=False;
Product_Control_Processor.Label3.Visible:=False;
Product_Control_Processor.Label4.Visible:=False;
Product_Control_Processor.Label5.Visible:=False;
```

```
Product_Control_Processor.DBEdit1.Visible:=False;
Product_Control_Processor.DBEdit2.Visible:=False;
Product_Control_Processor.DBEdit3.Visible:=False;
Product_Control_Processor.DBEdit4.Visible:=False;
Product_Control_Processor.DBEdit5.Visible:=False;
```

```
Product_Control_Processor.Button3.Visible:=False;
Product_Control_Processor.Button1.Visible:=False;
Product_Control_Processor.Button2.Visible:=False;
```

```
Product_Control_Processor.Button5.Visible:=True;
Product_Control_Processor.RadioGroup1.Visible:=True;
Product_Control_Processor.Edit1.Visible:=True;
```

```
end;
```

```
procedure TForm1.Processor5Click(Sender: TObject);
begin
```

```
Processor_Stock_Control.show;
form1.Hide;
end;
```

```
procedure TForm1.Processor3Click(Sender: TObject);
begin
```



```

Form1.Hide;
Product_Control_Processor.show;
Product_Control_Processor.Label1.Visible:=False;
Product_Control_Processor.Label2.Visible:=False;
Product_Control_Processor.Label3.Visible:=False;
Product_Control_Processor.Label4.Visible:=False;
Product_Control_Processor.Label5.Visible:=False;

Product_Control_Processor.DBEdit1.Visible:=False;
Product_Control_Processor.DBEdit2.Visible:=False;
Product_Control_Processor.DBEdit3.Visible:=False;
Product_Control_Processor.DBEdit4.Visible:=False;
Product_Control_Processor.DBEdit5.Visible:=False;

Product_Control_Processor.Button3.Visible:=False;
Product_Control_Processor.Button1.Visible:=False;
Product_Control_Processor.Button2.Visible:=False;

Product_Control_Processor.Button5.Visible:=False;
Product_Control_Processor.RadioGroup1.Visible:=True;
Product_Control_Processor.Edit1.Visible:=True;
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
form1.Close;
end;

procedure TForm1.Ram2Click(Sender: TObject);
begin
Form1.Hide;
Product_Control_Ram.Show;
Product_Control_Ram.Label1.Visible:=True;
Product_Control_Ram.Label2.Visible:=True;
Product_Control_Ram.Label3.Visible:=True;
Product_Control_Ram.Label4.Visible:=True;
Product_Control_Ram.Label5.Visible:=True;
Product_Control_Ram.DBEdit1.Visible:=True;
Product_Control_Ram.DBEdit2.Visible:=True;
Product_Control_Ram.DBEdit3.Visible:=True;
Product_Control_Ram.DBEdit4.Visible:=True;
Product_Control_Ram.DBEdit5.Visible:=True;
Product_Control_Ram.RadioGroup1.Visible:=False;
Product_Control_Ram.Edit1.Visible:=False;
Product_Control_Ram.Button1.Visible:=True;
Product_Control_Ram.Button2.Visible:=True;
Product_Control_Ram.Button3.Visible:=True;
Product_Control_Ram.Button4.Visible:=False;
Product_Control_Ram.Button5.Visible:=True;
Product_Control_Ram.DBGrid1.Visible:=True;

```

end;

procedure TForm1.Ram4Click(Sender: TObject);
begin

Form1.Hide;

Product_Control_Ram.Show;

Product_Control_Ram.Label1.Visible:=False;

Product_Control_Ram.Label2.Visible:=False;

Product_Control_Ram.Label3.Visible:=False;

Product_Control_Ram.Label4.Visible:=False;

Product_Control_Ram.Label5.Visible:=False;

Product_Control_Ram.Label6.Visible:=False;

Product_Control_Ram.DBEdit1.Visible:=False;

Product_Control_Ram.DBEdit2.Visible:=False;

Product_Control_Ram.DBEdit3.Visible:=False;

Product_Control_Ram.DBEdit4.Visible:=False;

Product_Control_Ram.DBEdit5.Visible:=False;

Product_Control_Ram.DBEdit6.Visible:=False;

Product_Control_Ram.RadioGroup1.Visible:=True;

Product_Control_Ram.Edit1.Visible:=True;

Product_Control_Ram.Button1.Visible:=False;

Product_Control_Ram.Button2.Visible:=False;

Product_Control_Ram.Button3.Visible:=False;

Product_Control_Ram.Button4.Visible:=True;

Product_Control_Ram.Button5.Visible:=True;

Product_Control_Ram.DBGrid1.Visible:=True;

end;

procedure TForm1.Ram5Click(Sender: TObject);
begin

Form1.Hide;

Product_Control_Ram.Show;

Product_Control_Ram.Label1.Visible:=False;

Product_Control_Ram.Label2.Visible:=False;

Product_Control_Ram.Label3.Visible:=False;

Product_Control_Ram.Label4.Visible:=False;

Product_Control_Ram.Label5.Visible:=False;

Product_Control_Ram.DBEdit1.Visible:=False;

Product_Control_Ram.DBEdit2.Visible:=False;

Product_Control_Ram.DBEdit3.Visible:=False;

```

Product_Control_Ram.DBEdit4.Visible:=False;
Product_Control_Ram.DBEdit5.Visible:=False;
Product_Control_Ram.RadioGroup1.Visible:=True;
Product_Control_Ram.Edit1.Visible:=True;
Product_Control_Ram.Button1.Visible:=False;
Product_Control_Ram.Button2.Visible:=False;
Product_Control_Ram.Button3.Visible:=False;
Product_Control_Ram.Button4.Visible:=False;
Product_Control_Ram.Button5.Visible:=True;
Product_Control_Ram.DBGrid1.Visible:=True;

```

```
end;
```

```

procedure TForm1.Add1Click(Sender: TObject);
begin
  Customer.Show;
  Form1.Hide;
end;

```

```

procedure TForm1.Processor7Click(Sender: TObject);
begin
  Sell.show;
end;

```

```
end.
```

```
unit Unit2;
```

```
interface
```

```
uses
```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, DBTables, DB, Grids, DBGrids, ExtCtrls,
  jpeg;

```

```
type
```

```

TProduct_Control_Processor = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Button1: TButton;
  Label4: TLabel;
  Label5: TLabel;

```



```

DBGrid1: TDBGrid;
Button2: TButton;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
Button3: TButton;
Button4: TButton;
Button5: TButton;
Edit1: TEdit;
RadioGroup1: TRadioGroup;
Query1: TQuery;
DataSource1: TDataSource;
Image1: TImage;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Button5Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Product_Control_Processor: TProduct_Control_Processor;

implementation

uses Unit1;

{$R *.dfm}

procedure TProduct_Control_Processor.Button1Click(Sender: TObject);
begin
  QUERY1.Insert;
  query1.Post;
end;

procedure TProduct_Control_Processor.Button2Click(Sender: TObject);
begin
  dbedit1.text:="";
  dbedit2.text:="";
  dbedit3.text:="";
  dbedit4.text:="";
  dbedit5.text:="";
  dbedit1.SetFocus;

```

end;

```
procedure TProduct_Control_Processor.Button3Click(Sender: TObject);
begin
  Query1.Insert;
end;
```

```
procedure TProduct_Control_Processor.Button4Click(Sender: TObject);
begin
  Product_Control_Processor.Close;
  Form1.show;
end;
```

```
procedure TProduct_Control_Processor.Edit1Change(Sender: TObject);
begin
  if(radiogroup1.ItemIndex=0) then
  begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select * from TB_Processor where Manufacturer like
'+#39+(edit1.text)+'%'+#39);
    query1.Open;
  end;
  if(radiogroup1.ItemIndex=1) then
  begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select * from TB_Processor where Model like
'+#39+(edit1.text)+'%'+#39);
    query1.Open;
  end;
  if(radiogroup1.ItemIndex=2) then
  begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select * from TB_Processor where Frequency(GHz) like
'+#39+(edit1.text)+'%'+#39);
    query1.Open;
  end;
end;
```

end;

```
procedure TProduct_Control_Processor.Button5Click(Sender: TObject);
begin
```

```
  If query1.Eof = True then
    ShowMessage ('There is no data to delete !!!')
  else
    query1.delete;
```

```

end;

end.

unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DBTables, StdCtrls, DB, Grids, DBGrids, ExtCtrls;

type
  TProcessor_Stock_Control = class(TForm)
    RadioGroup1: TRadioGroup;
    Edit1: TEdit;
    Label1: TLabel;
    Edit2: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    DataSource2: TDataSource;
    Query1: TQuery;
    DBGrid2: TDBGrid;
    Button3: TButton;
    procedure Edit1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Processor_Stock_Control: TProcessor_Stock_Control;

implementation

uses Unit1;

{$R *.dfm}

procedure TProcessor_Stock_Control.Edit1Change(Sender: TObject);
begin
  if(radiogroup1.ItemIndex=0) then
  begin
    query1.Close;
  end;
end;

```



```

query1.SQL.Clear;
query1.SQL.Add('Select * from TB_Processor where Manufacturer like
'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=1) then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * from TB_Processor where Model like
'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=2) then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * from TB_Processor where Frequency like
'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
end;
end;

```

```

procedure TProcessor_Stock_Control.Button1Click(Sender: TObject);
var
a:integer;
begin
if edit2.Text="" then begin
ShowMessage('Enter Amount');
end
else begin
a:=0;
a:=query1.Fields[4].AsInteger;
query1.Edit;
query1.Fields[4].AsInteger:=strtoint(edit2.Text)+ a;
query1.Post;
end;
end;

```

```

procedure TProcessor_Stock_Control.Button2Click(Sender: TObject);
var
a:integer;
begin
if edit2.Text="" then begin
ShowMessage('Enter Amount');
end
else begin
a:=0;
a:=query1.Fields[4].AsInteger;

```

```

query1.Edit;
query1.Fields[4].AsInteger:=a-strtoint(edit2.Text);
query1.Post;
end;
end;

```

```

procedure TProcessor_Stock_Control.Button3Click(Sender: TObject);
begin
Processor_Stock_Control.Close;
Form1.show;
end;

end.

```

```

unit Unit4;

```

```

interface

```

```

uses

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, StdCtrls, ExtCtrls, Grids, DBGrids, Mask, DBCtrls,
  jpeg;

```

```

type

```

```

  TProduct_Control_Ram = class(TForm)

```

```

    Label1: TLabel;

```

```

    Label2: TLabel;

```

```

    Label3: TLabel;

```

```

    Label4: TLabel;

```

```

    DBEdit1: TDBEdit;

```

```

    DBEdit2: TDBEdit;

```

```

    DBEdit3: TDBEdit;

```

```

    DBEdit4: TDBEdit;

```

```

    DBEdit5: TDBEdit;

```

```

    Label5: TLabel;

```

```

    DBGrid1: TDBGrid;

```

```

    Button1: TButton;

```

```

    Button2: TButton;

```

```

    Button3: TButton;

```

```

    Button4: TButton;

```

```

    RadioGroup1: TRadioGroup;

```

```

    Edit1: TEdit;

```

```

    Button5: TButton;

```

```

    DataSource1: TDataSource;

```

```

    Query1: TQuery;

```

```

Label6: TLabel;
DBEdit6: TDBEdit;
Image1: TImage;
procedure Button1Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Product_Control_Ram: TProduct_Control_Ram;

implementation

uses Unit1;

{$R *.dfm}

procedure TProduct_Control_Ram.Button1Click(Sender: TObject);
begin
  Query1.Insert;

end;

procedure TProduct_Control_Ram.Edit1Change(Sender: TObject);
begin
  if(radiogroup1.ItemIndex=0) then
  begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select * from TB_Processor where Manufacturer like
'+#39+(edit1.text)+'%'+#39);
    query1.Open;
  end;
  if(radiogroup1.ItemIndex=1) then
  begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('Select * from TB_Processor where Model like
'+#39+(edit1.text)+'%'+#39);
    query1.Open;
  end;
  if(radiogroup1.ItemIndex=2) then
  begin

```



```

query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * from TB_Processor where Size like '+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=3) then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * from TB_Processor where Frequency(MHz) like
'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
end;

```

```

procedure TProduct_Control_Ram.Button2Click(Sender: TObject);
begin
query1.Post;

end;

```

```

procedure TProduct_Control_Ram.Button3Click(Sender: TObject);
begin
dbedit1.text:="";
dbedit2.text:="";
dbedit3.text:="";
dbedit4.text:="";
dbedit5.text:="";
dbedit6.Text:="";
dbedit1.SetFocus;
end;

```

```

procedure TProduct_Control_Ram.Button4Click(Sender: TObject);
begin
query1.delete;

end;

```

```

procedure TProduct_Control_Ram.Button5Click(Sender: TObject);
begin

Product_Control_Ram.Close;
form1.show;
end;

end.

```

```

var
  Sell: TSell;

implementation

uses Unit1, Unit8;

{$R *.dfm}

procedure TSell.FormActivate(Sender: TObject);
begin
  combobox1.Items.Clear;
  query1.First;
  while not(query1.Eof) do begin
    Sell.ComboBox1.Items.Add(query1.Fields[2].AsString);
    query1.Next;
  end;

end;
procedure TSell.ComboBox1Change(Sender: TObject);
begin
  Combobox2.Items.Clear;
  combobox4.Items.Clear;
  query1.First;
  while not(query1.Eof) do begin
    if (query1.Fields[2].AsString=combobox1.Text) then begin
      Sell.combobox2.Items.Add(query1.Fields[1].AsString);
      Sell.combobox4.Items.Add(query1.Fields[0].AsString);
    end;
    query1.Next;
  end;
end;

procedure TSell.Button1Click(Sender: TObject);
begin
  Form1.Show;
  Sell.Close;

end;

procedure TSell.ComboBox3Change(Sender: TObject);
begin
  if combobox3.ItemIndex=0 then
  begin
    Sell.Image1.Visible:=True;
    query2.Close;
    query2.SQL.Clear;
    query2.SQL.Add('Select * from TB_Processor');
  end;
end;

```

```
query2.Open;
end;
```

```
if combobox3.ItemIndex=1 then
begin
Sell.Image2.Visible:=True;
query2.Close;
query2.SQL.Clear;
query2.SQL.Add('Select * from TB_Ram');
query2.Open;
end;
```

```
if combobox3.ItemIndex=2 then
begin
Sell.Image3.Visible:=True;
query2.Close;
query2.SQL.Clear;
query2.SQL.Add('Select * from TB_HDD');
query2.Open;
end;
```

```
if combobox3.ItemIndex=3 then
begin

Sell.Image4.Visible:=True;
query2.Close;
query2.SQL.Clear;
query2.SQL.Add('Select * from TB_Mainboard');
query2.Open;
end;
```

```
end;
```

```
procedure TSell.Button2Click(Sender: TObject);
var
a:integer;
amount:integer;
price:integer;
```

```
begin
```

```
if combobox3.ItemIndex=0 then
begin
if edit1.Text="" then begin
ShowMessage('Enter Amount');
end
```



```

else begin
a:=0;
a:=query2.Fields[4].AsInteger;
query2.Edit;
query2.Fields[4].AsInteger:=a-strtoint(edit1.Text);
end;
query3.edit;
query3.Insert;
dbgrid2.Columns.Grid.Fields[3].AsString:=combobox1.Text;
dbgrid2.Columns.Grid.Fields[2].AsString:=combobox2.Text;
dbgrid2.Columns.Grid.Fields[1].AsString:=combobox4.Text;

dbgrid2.Columns.Grid.Fields[4].AsString:=dbgrid1.columns.grid.fields[1].AsString
+ '' + dbgrid1.columns.grid.fields[2].AsString
+ '' + dbgrid1.columns.grid.fields[3].AsString
+ ' GHz';
dbgrid2.Columns.Grid.Fields[5].AsString:=edit1.Text;

dbgrid2.Columns.Grid.Fields[6].AsString:=dbgrid1.columns.grid.fields[5].AsString;
amount:=0;
price:=0;
amount:=query3.fields[5].asinteger;
price:=query3.fields[6].asinteger;

dbgrid2.Fields[7].AsString:=inttostr(amount*price);
Sell.Query3.Fields[8].AsString:=datetostr(date);

query3.Open;
query3.Post;
end;

if combobox3.ItemIndex=1 then
begin
if edit1.Text="" then begin
ShowMessage('Enter Amount');
end
else begin
a:=0;
a:=query2.Fields[5].AsInteger;
query2.Edit;
query2.Fields[5].AsInteger:=a-strtoint(edit1.Text);
end;
query3.Edit;
query3.Insert;
dbgrid2.Columns.Grid.Fields[3].AsString:=combobox1.Text;
dbgrid2.Columns.Grid.Fields[2].AsString:=combobox2.Text;
dbgrid2.Columns.Grid.Fields[1].AsString:=combobox4.Text;

dbgrid2.Columns.Grid.Fields[4].AsString:=dbgrid1.columns.grid.fields[1].AsString
+ '' + dbgrid1.columns.grid.fields[2].AsString

```

```

+ '' + dbgrid1.columns.grid.fields[3].AsString
+ ' GB '
+ dbgrid1.columns.grid.fields[3].AsString
+ ' MHz';

```

```
dbgrid2.Columns.Grid.Fields[5].AsString:=edit1.Text;
```

```

dbgrid2.Columns.Grid.Fields[6].AsString:=dbgrid1.columns.grid.fields[6].AsString;
amount:=0;
price:=0;
amount:=query3.fields[5].asinteger;
price:=query3.fields[6].asinteger;

```

```
dbgrid2.Fields[7].AsString:=inttostr(amount*price);
```

```
Sell.Query3.Fields[8].AsString:=datetostr(date);
```

```

query3.Open;
query3.Post;
end;
end;

```

```

procedure TSell.Button3Click(Sender: TObject);
begin
If query3.Eof = True then
ShowMessage ('There is no data to delete !!!')
else
if Sell.combobox3.ItemIndex=0 then
begin
query2.Edit;
query2.Fields[4].AsInteger:=query2.Fields[4].AsInteger+strtoint(edit1.Text);
query3.delete;
end;

```

```

if Sell.combobox3.ItemIndex=1 then
begin
query2.Edit;
query2.Fields[5].AsInteger:=query2.Fields[5].AsInteger+strtoint(edit1.Text);
query3.delete;
end;

```

```
end;
```

```

procedure TSell.Button5Click(Sender: TObject);
begin
Sale_Report.QuickRep1.Preview;

```

end;

end.

unit Unit6;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, DB, DBTables, StdCtrls, Grids, DBGrids, DBCtrls, Mask, jpeg,
ExtCtrls;

type

TCustomer = class(TForm)

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

DBEdit1: TDBEdit;

DBEdit2: TDBEdit;

DBEdit3: TDBEdit;

DBMemo1: TDBMemo;

DBGrid1: TDBGrid;

Button1: TButton;

Button2: TButton;

Button3: TButton;

Button4: TButton;

DataSource1: TDataSource;

Query1: TQuery;

Button5: TButton;

Image1: TImage;

procedure Button4Click(Sender: TObject);

procedure Button5Click(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Customer: TCustomer;

implementation


```

uses Unit1;

{$R *.dfm}

procedure TCustomer.Button4Click(Sender: TObject);
begin
  Customer.Close;
  Form1.Show;
end;

procedure TCustomer.Button5Click(Sender: TObject);
begin

  query1.Insert;
end;

procedure TCustomer.Button1Click(Sender: TObject);
begin
  query1.Post;
  query1.CommitUpdates;
  Query1.Refresh;
end;

procedure TCustomer.Button2Click(Sender: TObject);
begin
  dbedit1.text:="";
  dbedit2.text:="";
  dbedit3.text:="";
  dbmemo1.Text:="";
  dbedit1.SetFocus;
end;

procedure TCustomer.Button3Click(Sender: TObject);
begin

  If query1.Eof = True then
    ShowMessage ('There is no data to delete !!!')
  else
    query1.delete;
    query1.CommitUpdates;
  end;

end.

```

unit Unit7;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Grids, DBGrids, DB, DBTables;

type

TTotal_Sales = class(TForm)

DataSource1: TDataSource;

Query1: TQuery;

DBGrid1: TDBGrid;

private

{ Private declarations }

public

{ Public declarations }

end;

var

Total_Sales: TTotal_Sales;

implementation

{ \$R *.dfm }

end.

unit Unit8;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls, QuickRpt, DB, QRCtrl, DBTables;

type

TSale_Report = class(TForm)

Table1: TTable;

QuickRep1: TQuickRep;

QRLabel1: TQRLabel;

DetailBand1: TQRBand;

QRDBText1: TQRDBText;

QRDBText2: TQRDBText;

QRDBText3: TQRDBText;

QRDBText4: TQRDBText;

QRDBText5: TQRDBText;

```

QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
QRDBText8: TQRDBText;
QRDBText9: TQRDBText;
QRLabel2: TQRLabel;
QRLabel3: TQRLabel;
QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRLabel8: TQRLabel;
QRLabel9: TQRLabel;
QRLabel10: TQRLabel;
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Sale_Report: TSale_Report;

implementation

{$R *.dfm}

end.

```