NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

BOOK STORE PROGRAM

Graduation Project
COM400

Student: MURAT KARAOĞUL (20020559)

Supervisor: Assist. Prof. Dr.Elburus IMANOV

Nicosia-2008

ACKNOWLEDGEMENTS

ABSTRACT

The aim of this project is to register book store program that contain registration, all applications and also customer and books application. The program was prepared by using Delphi programming and using database.

This project consist of so many forms and menues. The main form of the arrive the others forms . Which are include information about the books, customers.

First time i thing this program form my friend's books for help register. So this program is real life prepare to İrfan Book Store.

To show results show the efficiency of the program of book in program of the using in other chapters

TABLE OF CONTENTS

CHAPTER 3

CHAPTER 4

INTRODUCTION

This project is register book and company workers which uses PARADOX7 quarries. This program was prepared by using Borland Delphi 6 and PARADOX7.

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

In the first Borland Delphi 6 programming language is described, its properties, components and some examples, I used Borland Delphi 6 in my project, because I find it easy and I liked its coding system. Borland Delphi 6 for applications

In the Second Chapter I described Database system, I used PARADOX7 data base system in my program with Borland Delphi 6.

Third Chapter is About the project , how we create it, its forms and using the program

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

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 I 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 Compliers

There are two types complier 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

2

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).
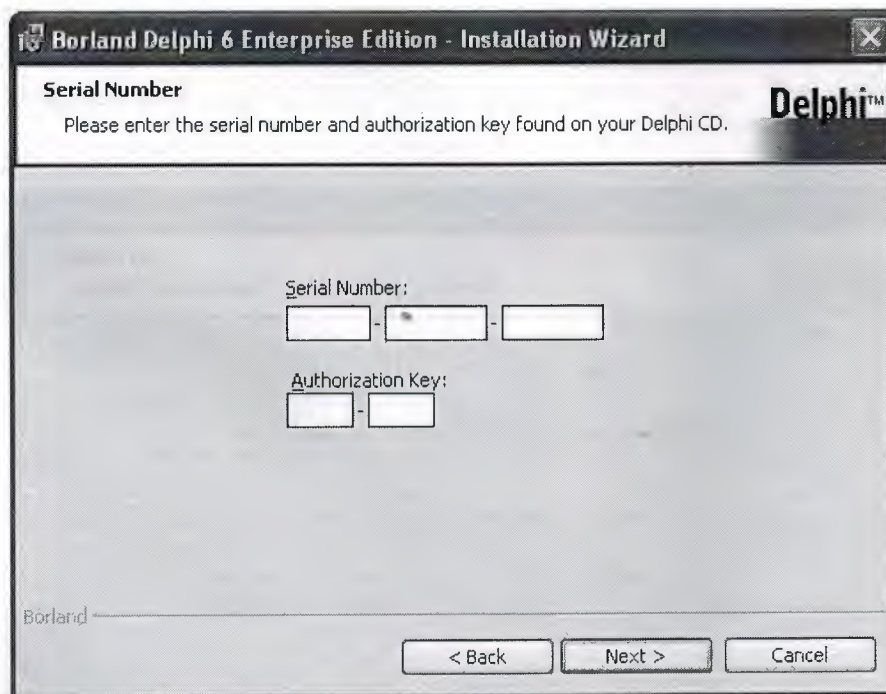
Figure 1.2 Serial Number And Authorization Screen

Later, the License Agreement screen will popup:
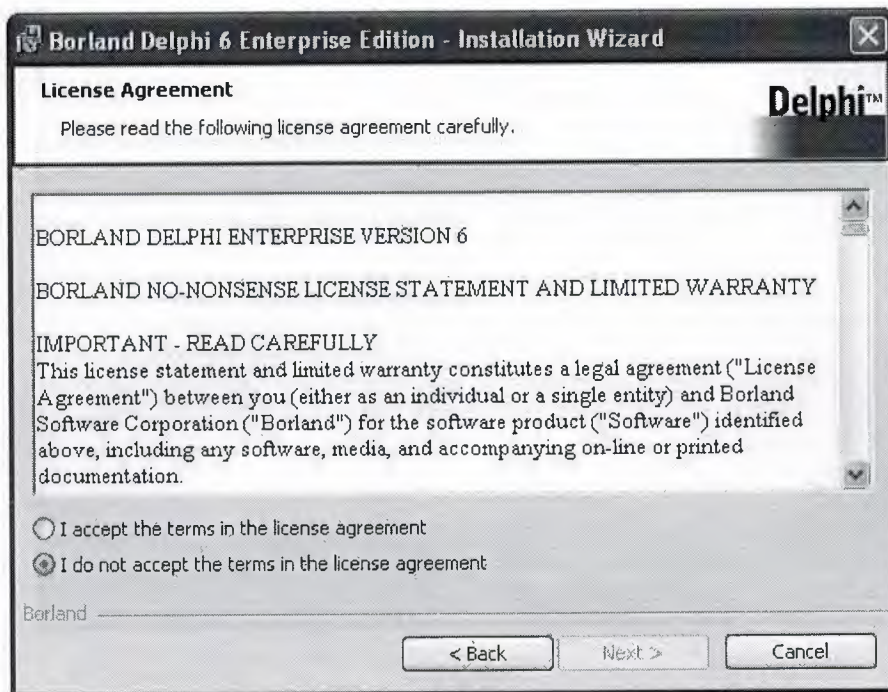
Figure 1.3 Lisanse Agreement Screen

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.
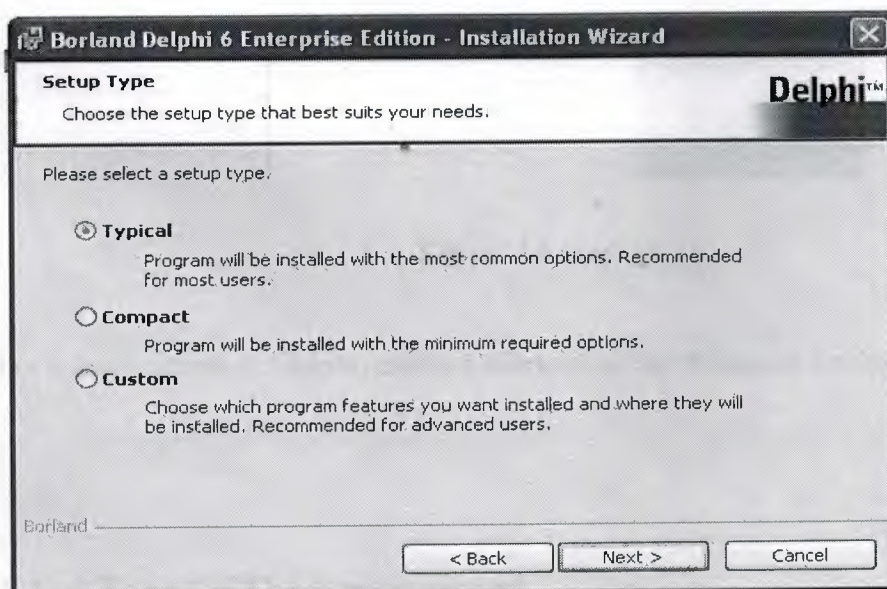
Figure 1.4. SetUp Type
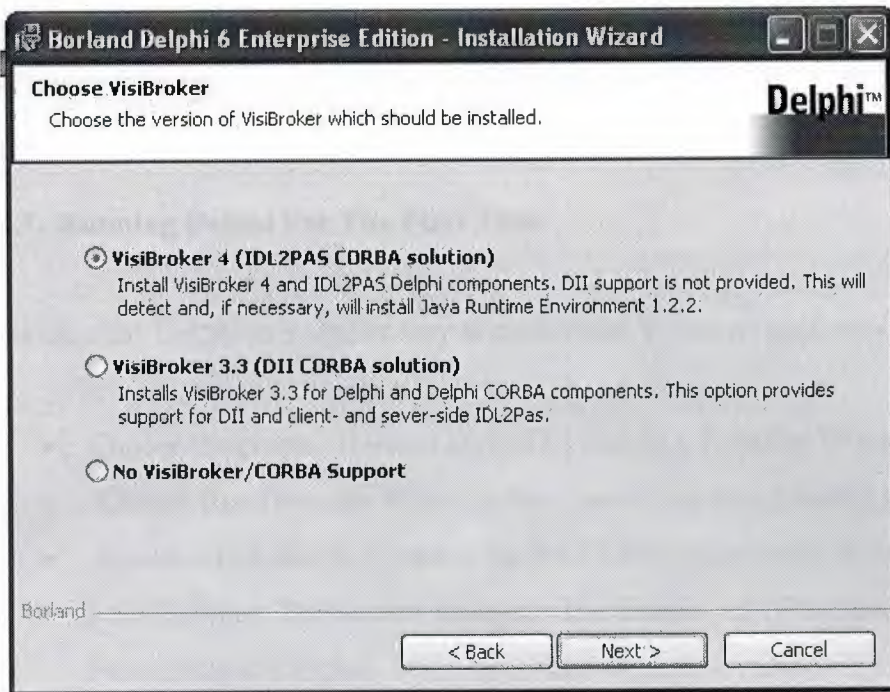
Figure 1.5 Destination Folder Screen

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) if 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)
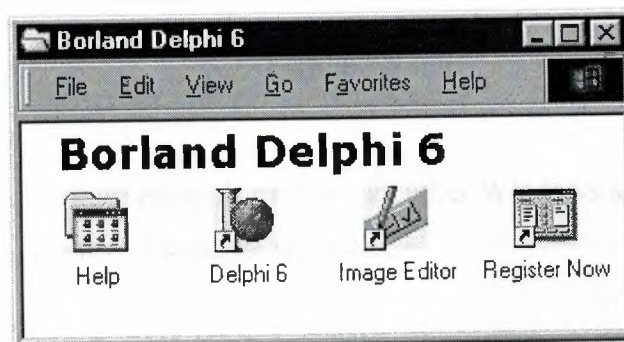
Figure 1.7. Borland Delphi 6 Folder

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.
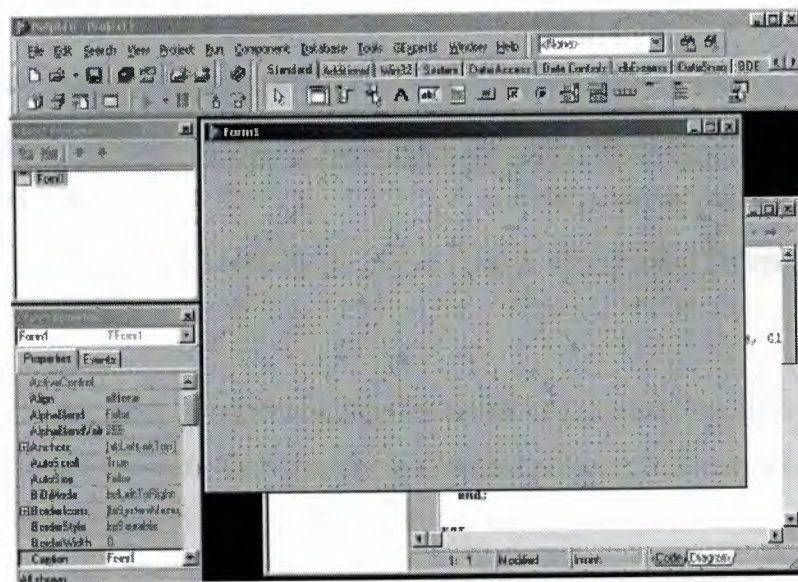
Figure 1.8.IDE

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

12

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.
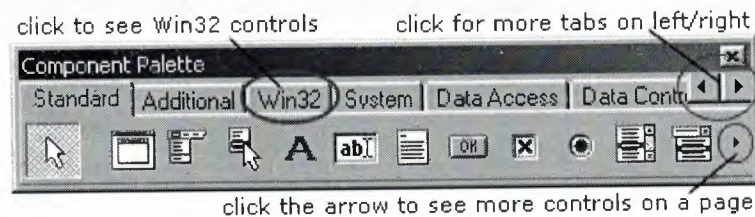
click to see Win32 controls   click for more tabs on left/right

Figure 1.10. Component Palatte

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

14
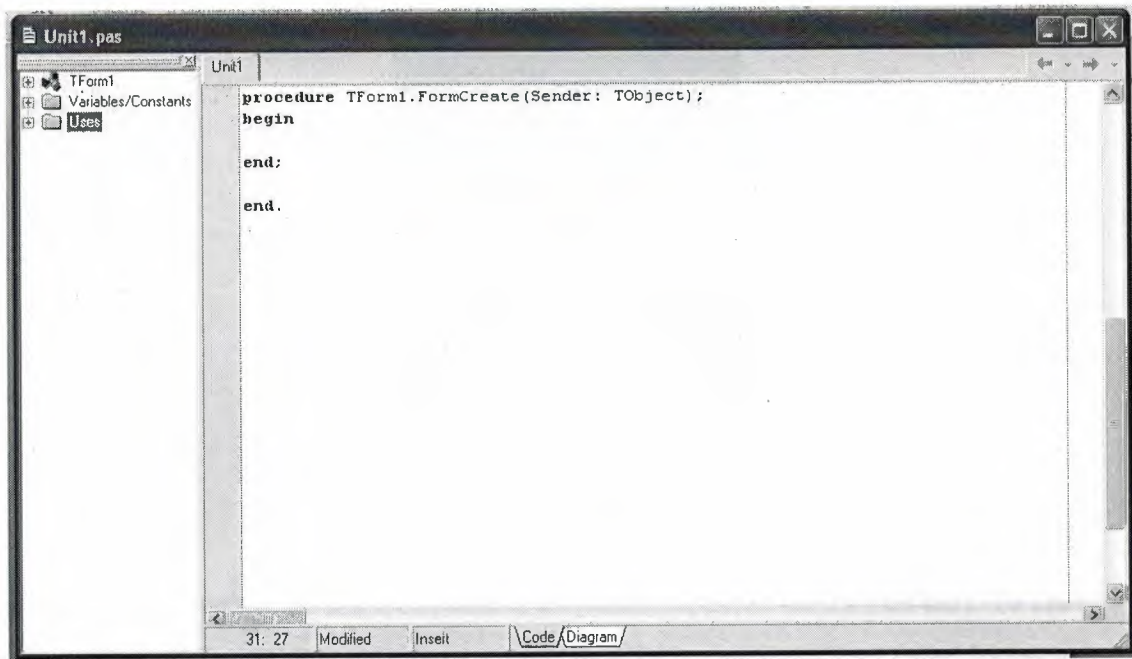
Fig.1.11. Code Editor Window

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.
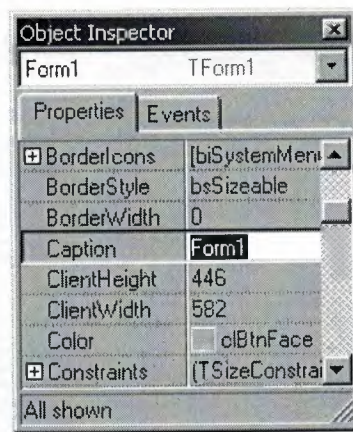
Figure 1.11. Object Inspector

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.

16
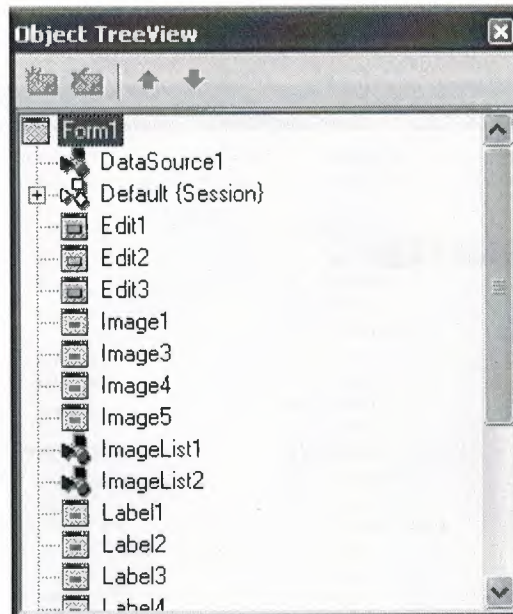
Figure 1.12. Object Tree View

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.

```
Run
  ▶  Run                           F9
  ⌇  Attach to Process...
  ▦  Parameters...
  ✗  Register ActiveX Server
  ✗  Unregister ActiveX Server
     Install COM+ Objects
  ─────────────────────────────────
  ⟶  Step Over                     F8
  ▯  Trace Into                    F7
  ▣  Trace to Next Source Line    Shift+F7
  ▦  Run to Cursor                 F4
  ▯  Run Until Return             Shift+F8
  ▤  Show Execution Point
  ‖  Program Pause
  ▣  Program Reset                Ctrl+F2
  ─────────────────────────────────
  🔍  Inspect...
  ▦  Evaluate/Modify...           Ctrl+F7
  ⬚  Add Watch...                 Ctrl+F5
     Add Breakpoint                  ▶
```
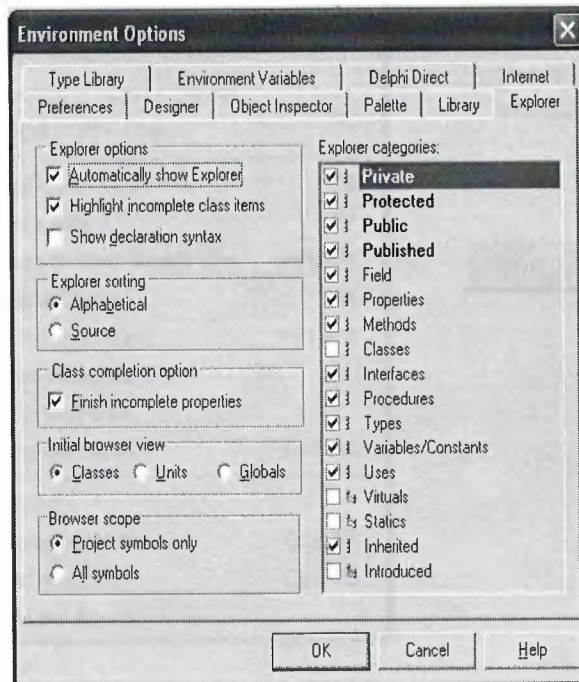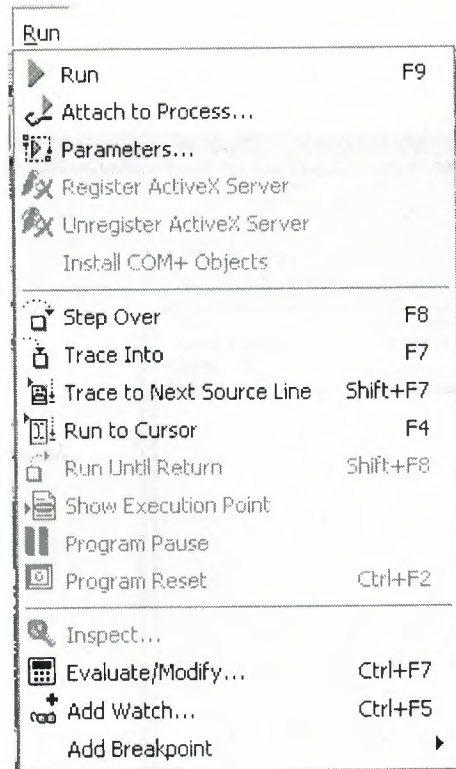
Choose any of the debugging commands from the Run menu.

Some commands are also available on the toolbar.

Figure1.14.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.
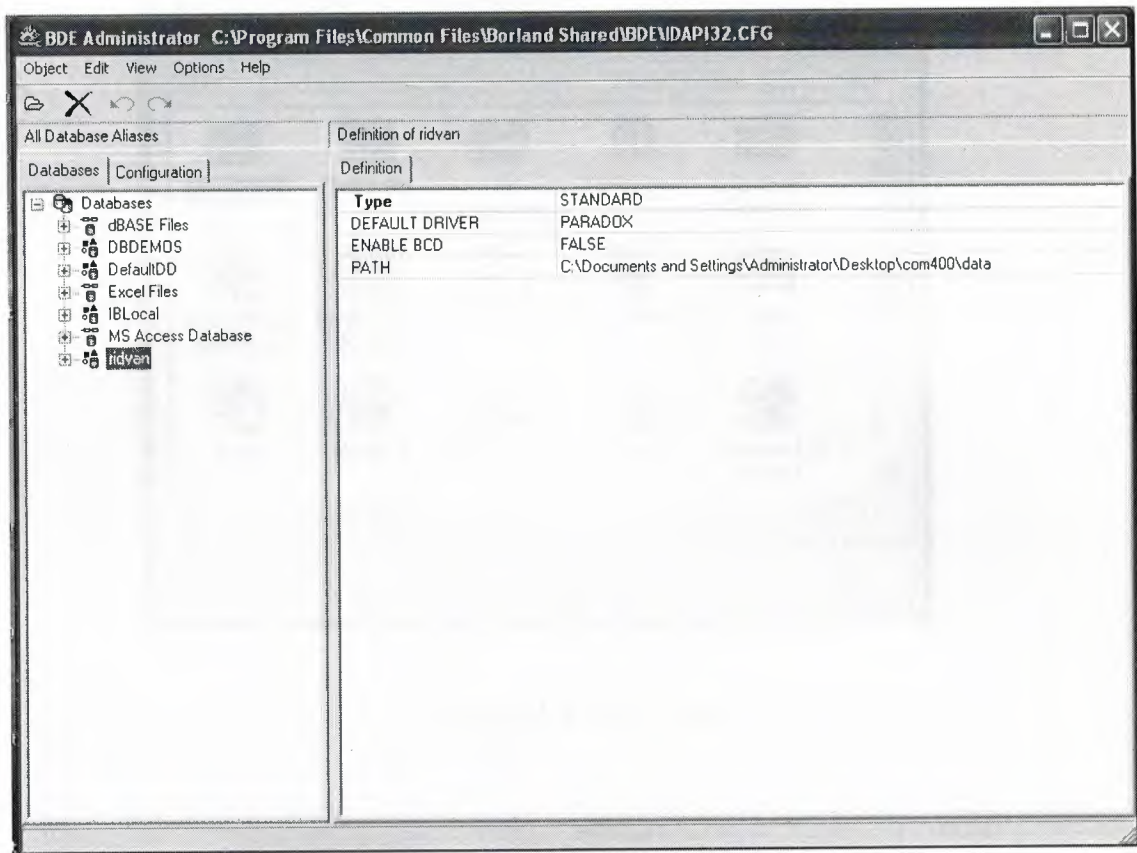
19

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.
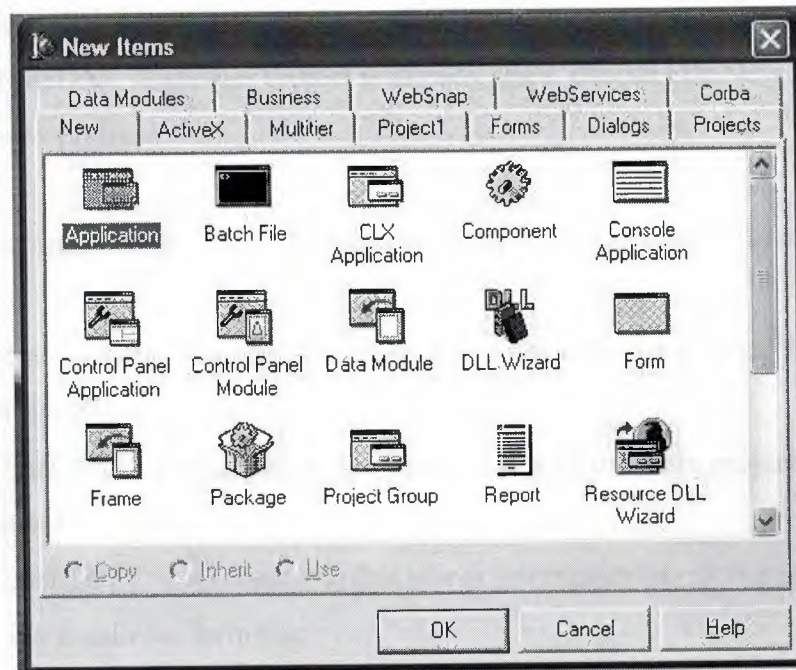
Figure 1.16.New Item

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 overwiew 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.

21

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.
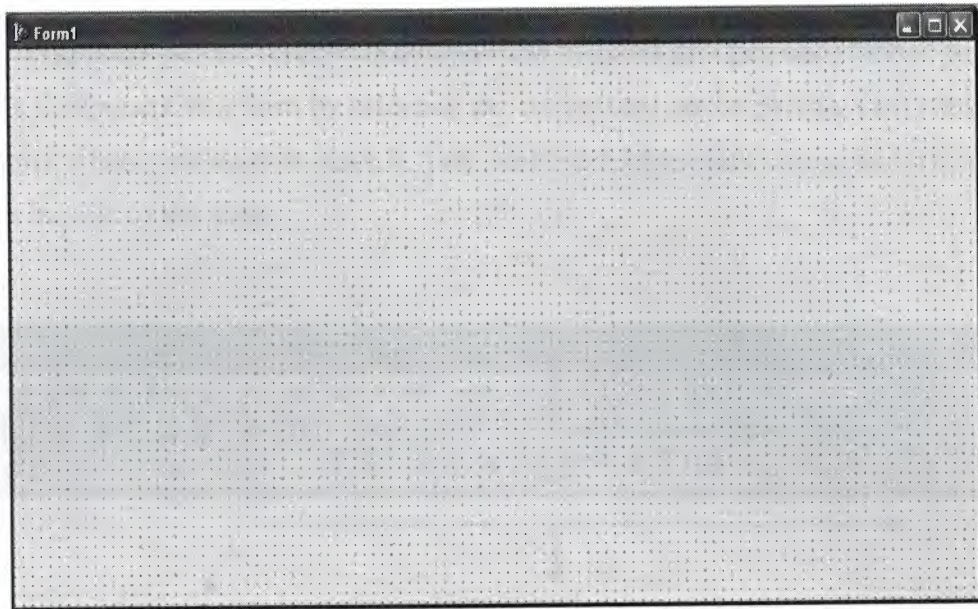
Figure 1.17. Form Screen

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 sellected 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.

Componont palette tabs   Components
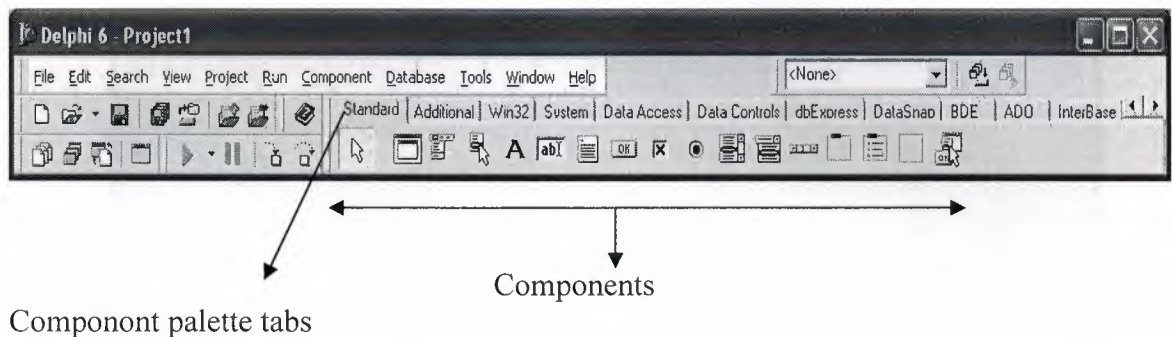
Figure 1.18.Standart Button

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.)
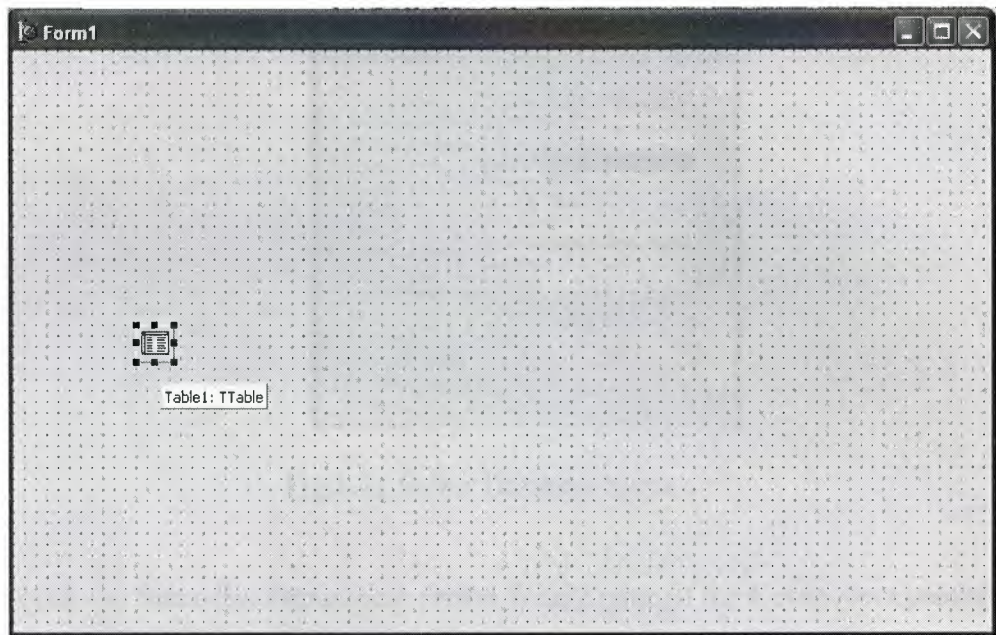
Figure 1.20. Table In The Form

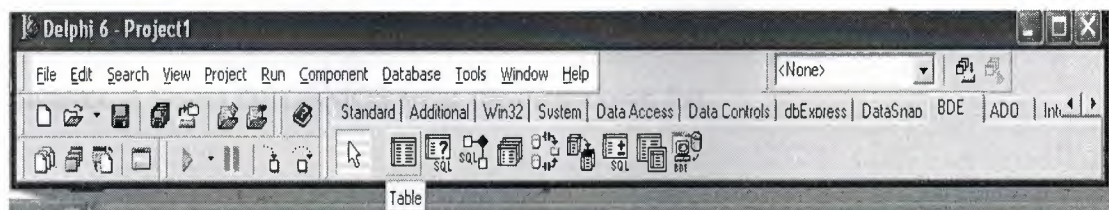Each Delphi component is a class; placing a component on a form creates an instance of that class. Once the component is on the form, Delphi generates the code necessary to construct an instance object when your application is running.

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.
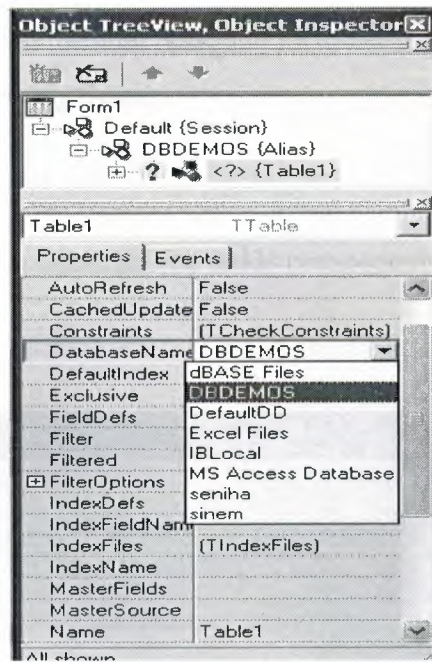
Fig.1.21.Select DatabaseName

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.

26

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
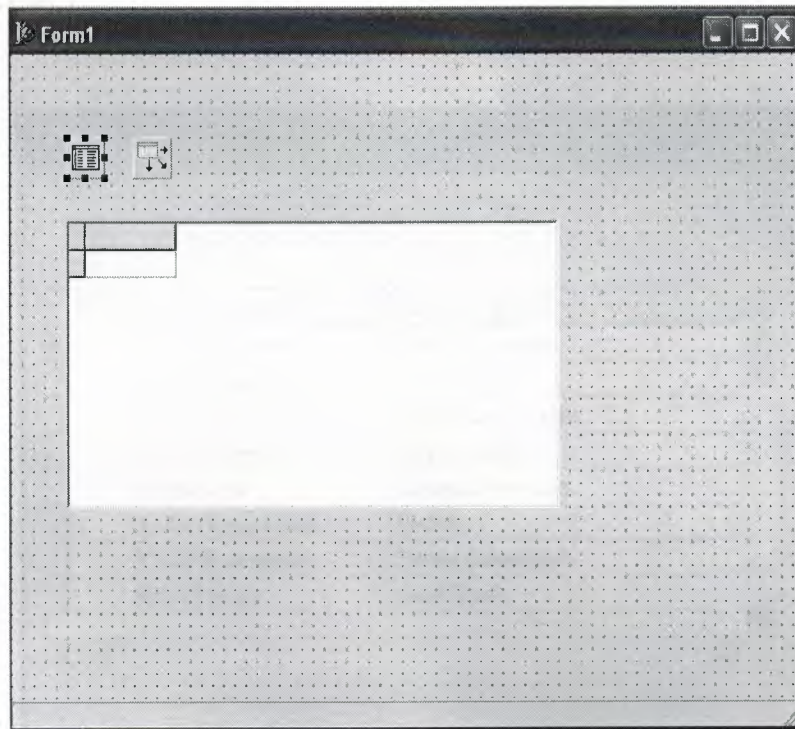
3. Set DBGrid properties to align the grid with the form. Double-click Anchors in the Object Inspector to display akLeft, akTop, akRight, and akBottom; set them all to True.

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.)
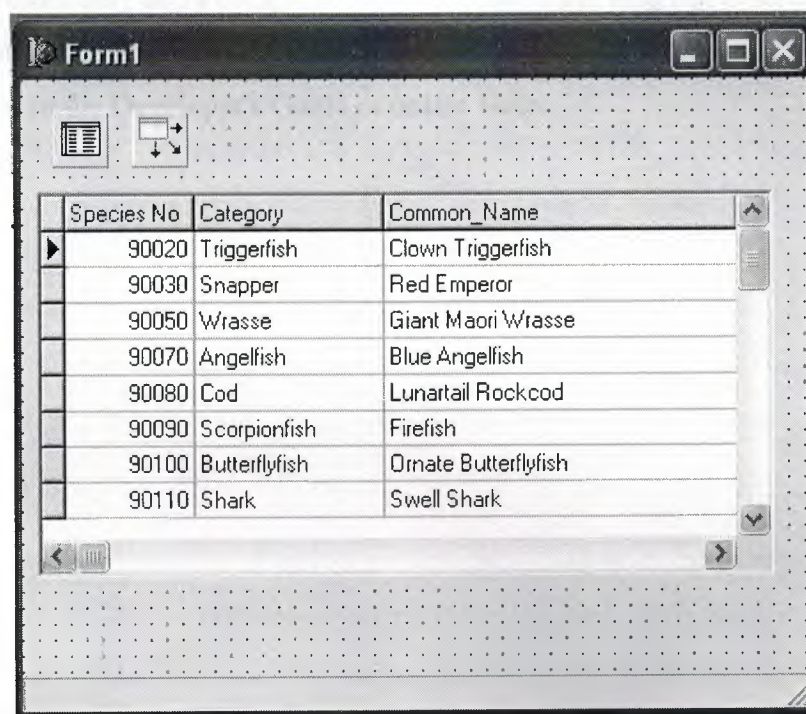
Figure 1.23. Show Table

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)

28

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.)

data-aware control (Grid) $\longrightarrow$ DataSource $\longrightarrow$ dataset (Table) $\longrightarrow$ BDE $\longrightarrow$ database
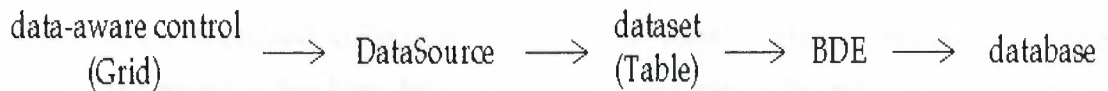
This architecture may seem complicated at first, but in the long run it simplifies development and maintenance. For more information, see "Developing database applications" in the Developer's Guide or online Help.

CHAPTER 2

2.1 INTRODUCTION TO DATABASE

A database is an organized collection of data. The term originated within thecomputer industry, but it s 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 collaction 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 lan_guage, 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

35

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 rmodel, 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
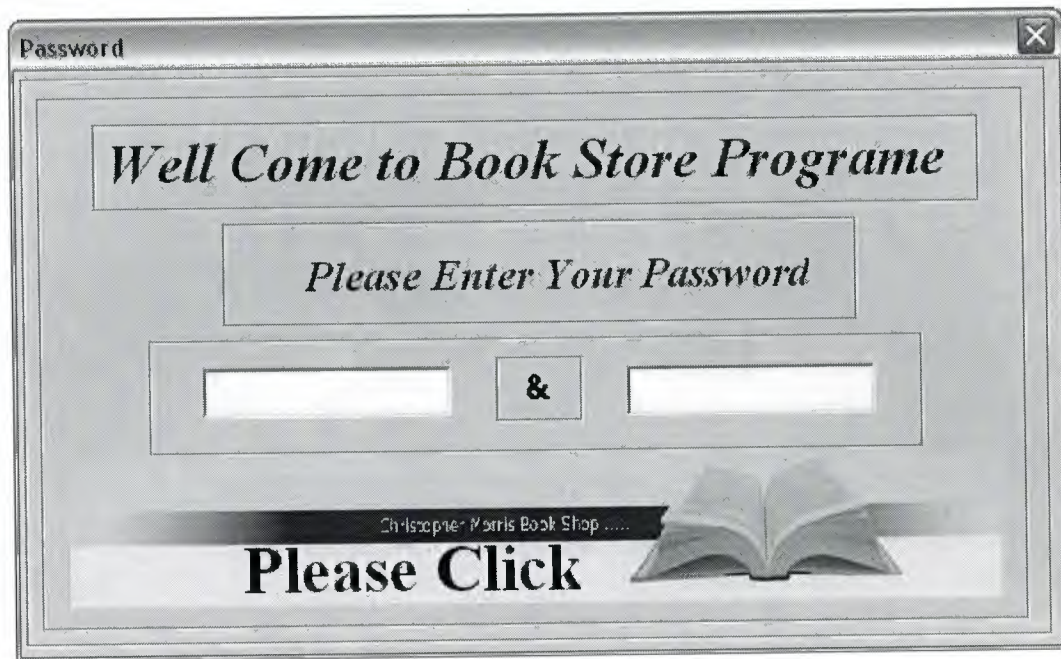
Description About Project

Figure 3.1(password menu)

3.1 Password Menu

When the execute my programme we see this enterance page. There are two box in the menu. First box is name and second box is number for your password. This password for fisrt box murat and second box 20020559 This is my name and school number.
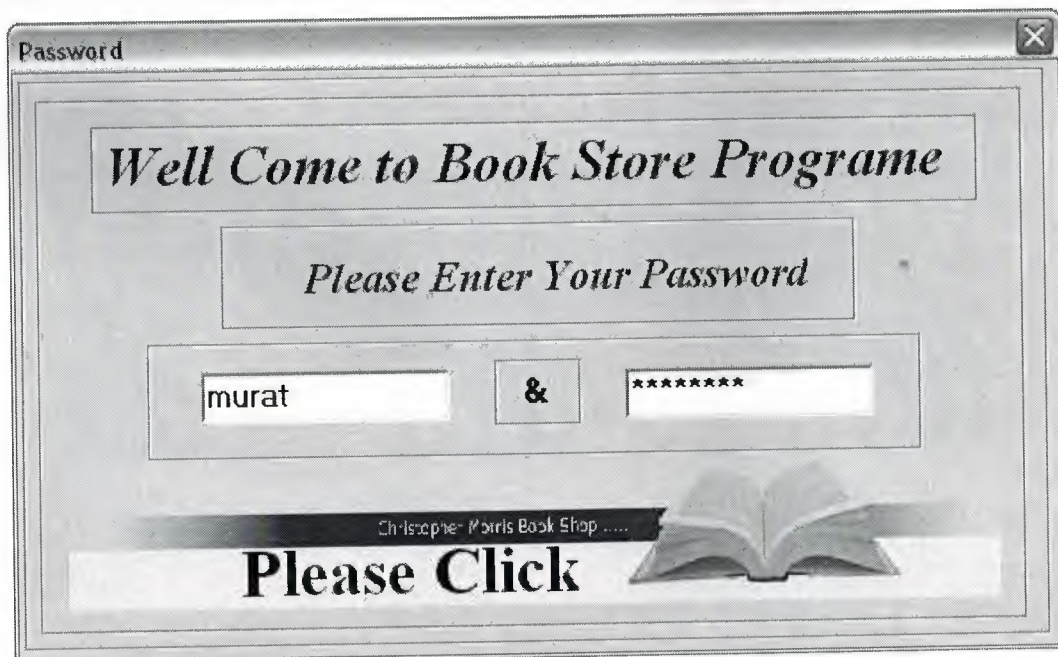

Figure 3.2(password menu)

3.2 Password Menu

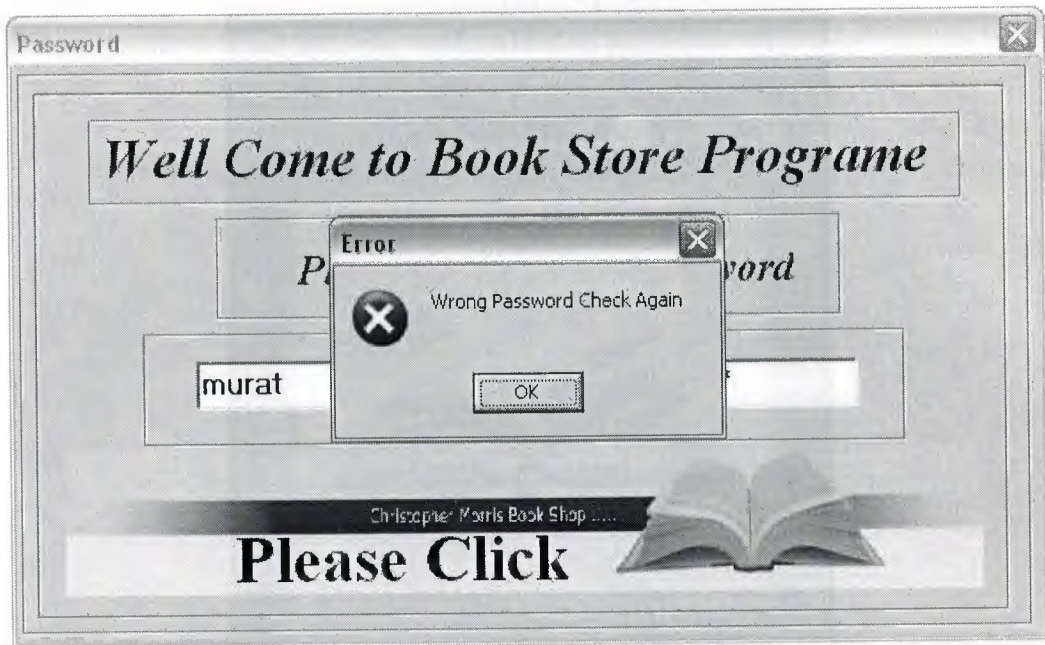If you write my code my programe will open.

Figure 3.3(password menu)

3.3 Password Menu

If you write another name or number you will see a messagebox (Wrong Password Check Again) You must click ok. Bok1 and Box2 will be clean. Please enterance name and number again. If you write correct name and number programe will open.
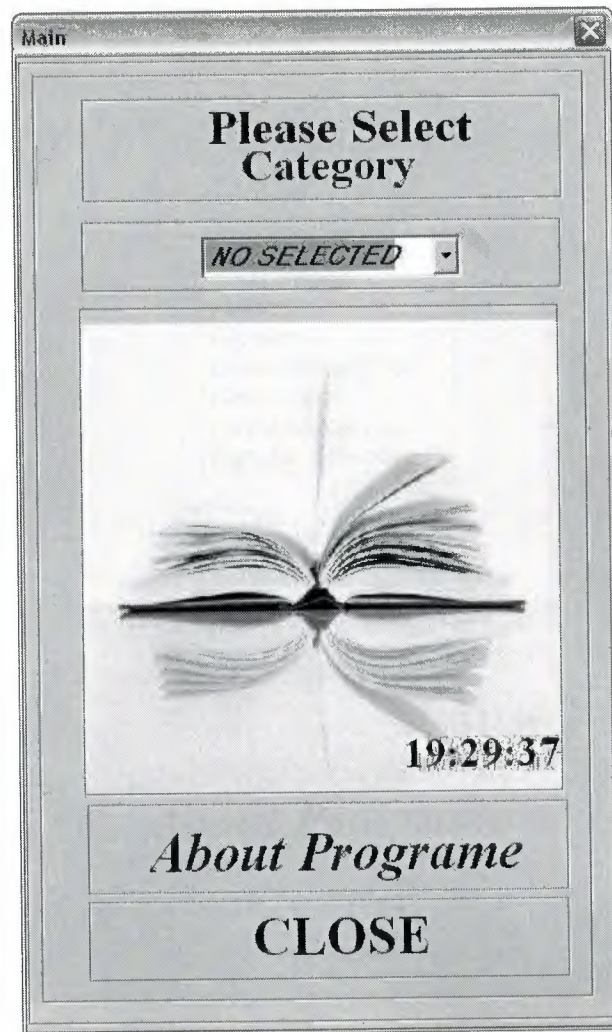
Figure 3.4(main menu)

3.4 Main Menu

You are seeing my main menu. There are three button and time this area . If you select (no selected) you can see department. And choose a class. If you click (about Programe) you can see my profile and select (close) programme will close.
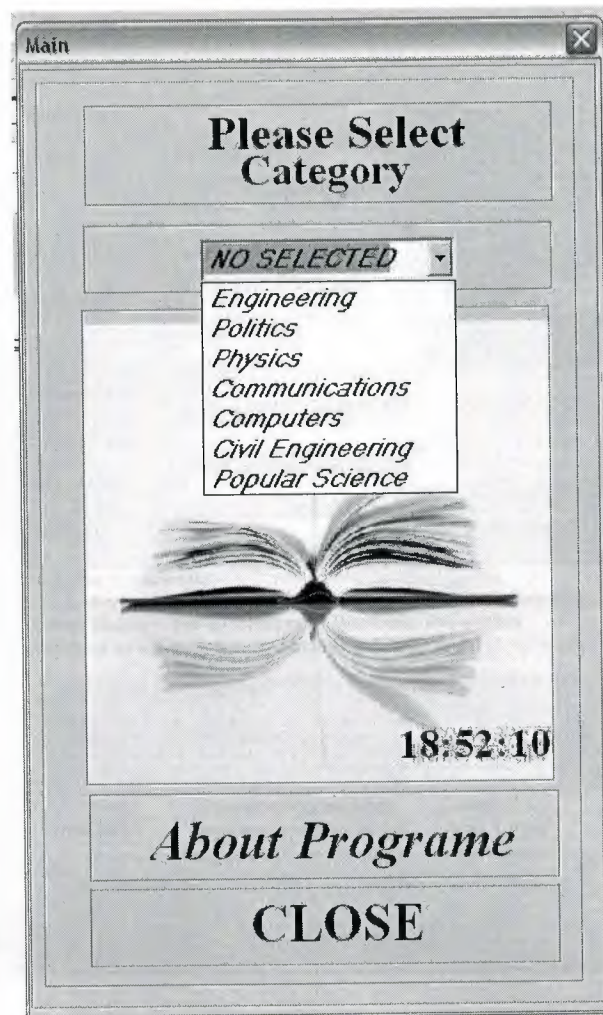
Figure 3.5(password menu)

3.5 Main Menu

You see when the click (No Selected) section. There are seven main area for this programe. If you select by select you can see all department. My main menu has got very simple and useful design for users.Now i will see all department one bye one.
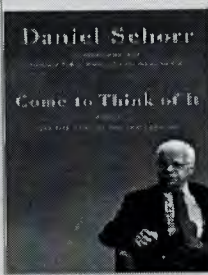
Figure 3.6(Engineering menu)

3.6 Engineering Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only engineering books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area.

Figure 3.7(Politicks menu)

3.7 Politicks Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only politicks books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area

Figure 3.8(Physics menu)

3.8 Physics Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only physics books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area

Figure 3.9(Communications menu)

3.9 Communications Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only commuunications books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area

Figure 3.10(Computers menu)

3.10 Computers Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only computers books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area

Figure 3.11(Civil Engineerings menu)

3.11 Civil Engineering Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only civil engineering books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area

Figure 3.12(Popular Sciences menu)

3.12 Popular Sciences Menu

You can enter the book information this department. Every book has got a ISBN number, book name, writer name but we needed buying date, stock, selling date, buying cost, selling cost, earning and introduction of bokks. Separately you can see database, time, date, select picture and customer registrations in this menu. You can select only popular siences books in this area. Because every books datebase is not same. Other books deparment execute look like this department. If you want to see another registration department close this department and you will come back main menu, And you can select another department this area

Figure 3.13(Customer Registration menu)

3.13 Customer Registration Menu

You can customer registration in this area. Programe has got every information for customer registration and we can see datebase of we can find old registration from name and register code. We can see cost and selling date.

Figure 3.14(About Programme menu)

3.14 Introduction Menu

You can see to me picture, department, name, surname and school number
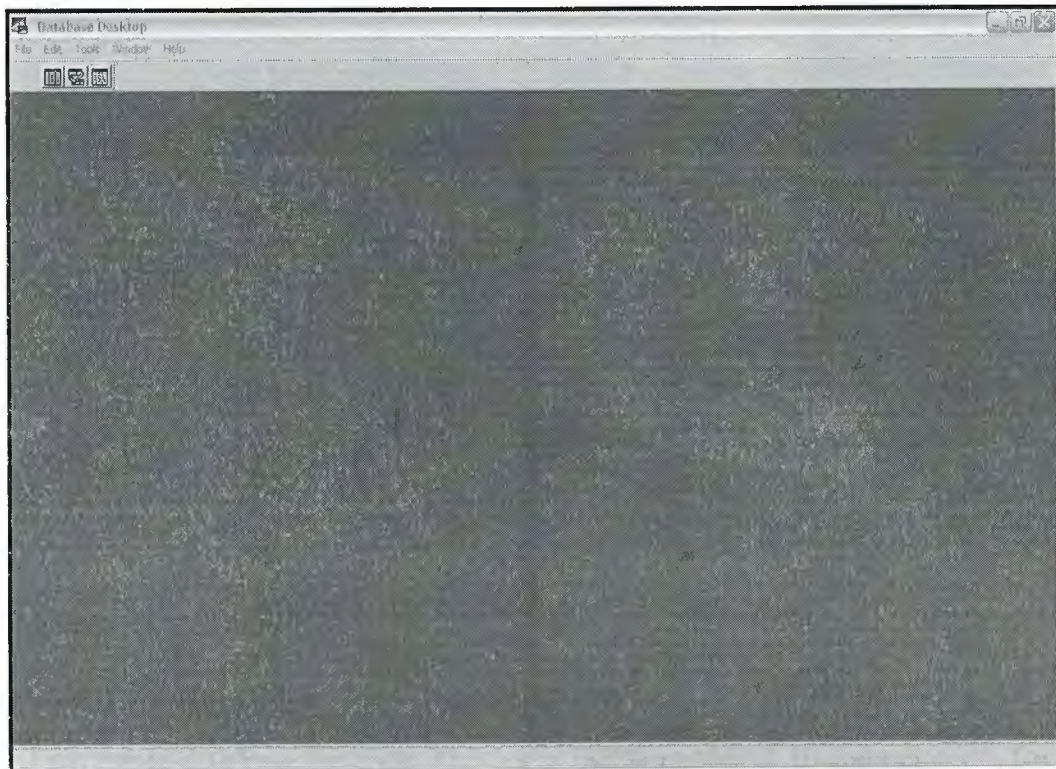
Figure 3.15(Paradox7 menu)

3.15 Paradox7 Database Menu

We can create and orders in date base.
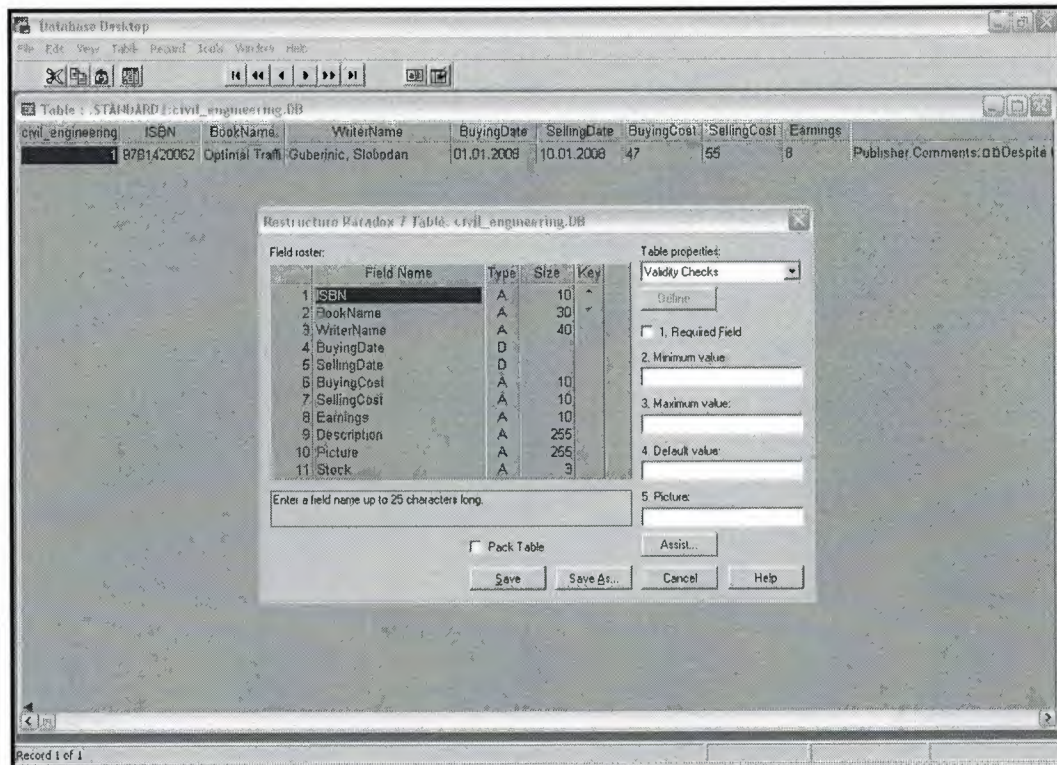
Figure 3.16(Paradox7 Books Design menu)

3.16 Paradox7 Database (For Books) Menu

I am create date base for book. This are is ISBN number A 10 and pointer *, Book name A 30 pointer *, WriterName A 40, BuyingDate D, SellingDate D, BuyingCost A 10, SellingCost A 10, Earnings A 10, Description A 255, Picture A 255, Stock A 3.
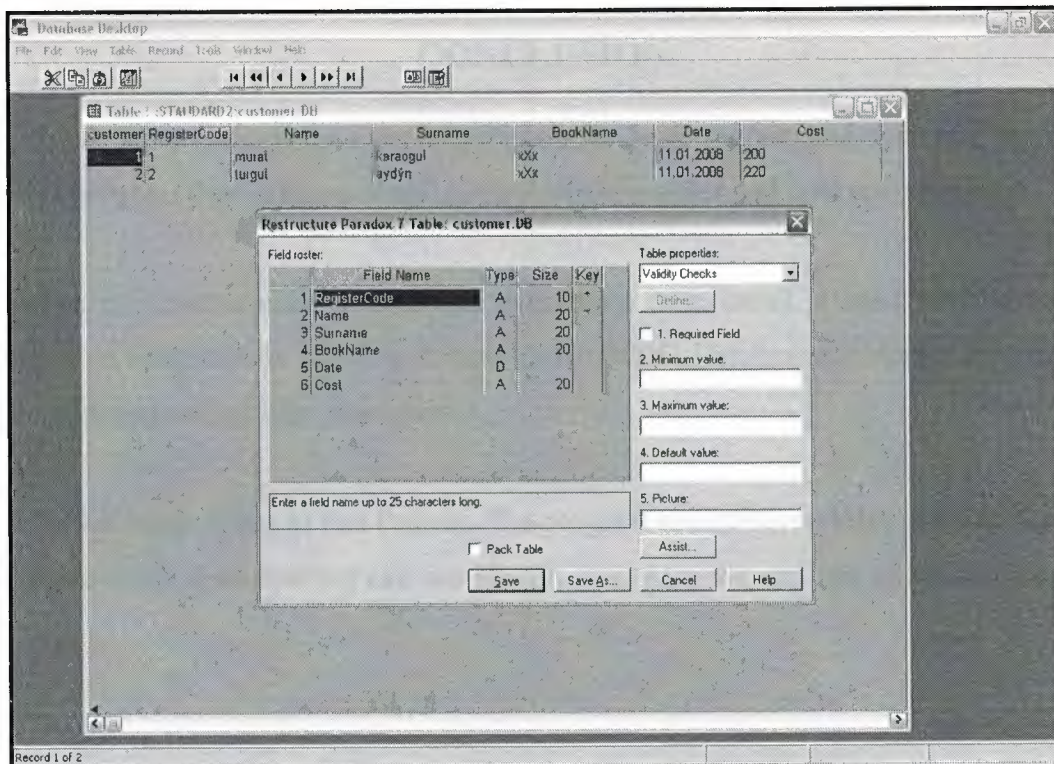
Figure 3.17(Paradox7 Customer Design menu)

3.17 Paradox7 Database (For Customers) Menu

I am create date base for customer. This are is RegisterCode A 10 and pointer *, name A 20 pointer *, Surname A 20, BookName A 20, Date D, Cost A 20

CONCLUSION

Book Store Program is a useful program for book register and customer register. By using this program they can record and control books register and customers register.

The program is easy in use, and everything is in detail, I used borland Delphi 6 Programming Language in building it, also PARADOX7 Database for storing information's. The program records register operation.

I used many forms in this Project. The program records everything, we can see who is work in our book store and we can see abouth this. Also we can see all information about book.

REFERENCES

[1] Yüksel İnan - Nihat Demirli Delphi 7 Learning Book

[2] Ihsan Karagülle Delphi 7 Edition Book

[3] Memik Yanık Borland Delphi- Sistem Yayıncılık

[4] http://www.google.com

[5] http://www.wikipedia.org

[6]Ezel Balkan Borland Delphi

[7] http://www.1keydata.com/sql

APPENDIX 1
Program Code

```pascal
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;
type
TForm1 = class(TForm)
  GroupBox5: TGroupBox;
  GroupBox6: TGroupBox;
      GroupBox1: TGroupBox;
  Label1: TLabel;
  GroupBox2: TGroupBox;
  Label2: TLabel;
  GroupBox3: TGroupBox;
  Edit1: TEdit;
  Edit2: TEdit;
  GroupBox7: TGroupBox;
  Label3: TLabel;
  Image1: TImage;
  Label4: TLabel;
  procedure FormCreate(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure Image1Click(Sender: TObject);
  procedure Label4Click(Sender: TObject);
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure Edit2KeyPress(Sender: TObject; var Key: Char);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation
  uses unit2;
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
edit1.text:='';
edit2.text:='';
```

```
end;

procedure TForm1.Button1Click(Sender: TObject);
var
c:word;
begin
if (edit1.text='1') and (edit2.text='2')  then
begin
form2.Show;

end
else
begin
c:=messagedlg('Wrong Password Check Again',mterror,[mbok],0);
edit1.text:='';
edit2.text:='';
end;
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
edit2.PasswordChar:='*';
end;

procedure TForm1.Image1Click(Sender: TObject);
var
c:word;
begin
if (edit1.text='murat') and (edit2.text='20020559')  then
begin
form2.Show;
form1.Visible:=false;

end
else
begin
c:=messagedlg('Wrong Password Check Again',mterror,[mbok],0);
edit1.text:='';
edit2.text:='';
end;
end;
procedure TForm1.Label4Click(Sender: TObject);
var
c:word;
begin
if (edit1.text='murat') and (edit2.text='20020559')  then
begin
form2.Show;
form1.Visible:=false;
```

```
end
else
begin
c:=messagedlg('Wrong Password Check Again',mterror,[mbok],0);
edit1.text:=";
edit2.text:=";
end;
end;
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if (key=#13) then edit2.SetFocus;
end;

procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then edit1.SetFocus;
end;

end.


unit Unit2;

interface

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

type
  TForm2 = class(TForm)
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox8: TGroupBox;
    GroupBox11: TGroupBox;
    GroupBox15: TGroupBox;
    GroupBox16: TGroupBox;
    ComboBox1: TComboBox;
    GroupBox18: TGroupBox;
    Label4: TLabel;
    Label5: TLabel;
    GroupBox19: TGroupBox;
    Image1: TImage;
    Label1: TLabel;
    Label2: TLabel;
    Timer1: TTimer;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
```

```
  procedure FormCreate(Sender: TObject);
  procedure ComboBox1Change(Sender: TObject);
  procedure Label1Click(Sender: TObject);
  procedure Label2Click(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
 Form2: TForm2;

implementation

uses Unit1, Unit3, Unit4, Unit5, Unit6, Unit7, Unit8, Unit9, Unit10,
 Unit11, Unit12;

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
form1.close;
end;




procedure TForm2.FormCreate(Sender: TObject);
begin
combobox1.text:='NO SELECTED';
end;

procedure TForm2.ComboBox1Change(Sender: TObject);
begin
   if (combobox1.ItemIndex=0) then
       begin
       form3.show;
       end;
       begin
   if (combobox1.ItemIndex=1) then
       form4.show;
       end;
       begin
   if (combobox1.ItemIndex=2) then
       form5.show;
       end;
       begin
   if (combobox1.ItemIndex=3) then
       form10.show;
```

```
          end;
          begin
      if (combobox1.ItemIndex=4) then
          form7.show;
          end;
          begin
      if (combobox1.ItemIndex=5) then
          form8.show;
          end;
          begin
      if (combobox1.ItemIndex=6) then
          form9.show;
          end;

end;
procedure TForm2.Label1Click(Sender: TObject);
begin
form12.show;
end;

procedure TForm2.Label2Click(Sender: TObject);
begin
form1.Close;
end;

procedure TForm2.Timer1Timer(Sender: TObject);
begin
label3.Caption:=timetostr(time);
end;

end.


unit Unit3;

interface

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

type
  TForm3 = class(TForm)
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    GroupBox3: TGroupBox;
    Label2: TLabel;
```

```
GroupBox4: TGroupBox;
GroupBox6: TGroupBox;
Label3: TLabel;
GroupBox11: TGroupBox;
Button1: TButton;
GroupBox16: TGroupBox;
GroupBox17: TGroupBox;
DBGrid1: TDBGrid;
GroupBox20: TGroupBox;
DBNavigator1: TDBNavigator;
GroupBox21: TGroupBox;
DBMemo1: TDBMemo;
GroupBox15: TGroupBox;
Image1: TImage;
GroupBox19: TGroupBox;
Button7: TButton;
GroupBox8: TGroupBox;
Label4: TLabel;
GroupBox9: TGroupBox;
Label5: TLabel;
GroupBox10: TGroupBox;
Label6: TLabel;
GroupBox22: TGroupBox;
Label7: TLabel;
GroupBox23: TGroupBox;
Label8: TLabel;
GroupBox26: TGroupBox;
OpenPictureDialog1: TOpenPictureDialog;
Button2: TButton;
GroupBox5: TGroupBox;
GroupBox12: TGroupBox;
GroupBox13: TGroupBox;
GroupBox14: TGroupBox;
GroupBox18: TGroupBox;
GroupBox24: TGroupBox;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DataSource1: TDataSource;
Query1: TQuery;
GroupBox25: TGroupBox;
Label9: TLabel;
GroupBox27: TGroupBox;
Edit1: TEdit;
GroupBox28: TGroupBox;
```

```
    Label10: TLabel;
    Timer1: TTimer;
    DateTimePicker1: TDateTimePicker;
    GroupBox29: TGroupBox;
    Label11: TLabel;
    GroupBox30: TGroupBox;
    DBEdit9: TDBEdit;
    GroupBox31: TGroupBox;
    Button3: TButton;
    Timer2: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure DataSource1DataChange(Sender: TObject; Field: TField);
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
    procedure Timer2Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form3: TForm3;

implementation

uses Unit4, Unit5, Unit6, Unit7, Unit8, Unit9;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  form3.Close;
end;

procedure TForm3.Button2Click(Sender: TObject);
```

```
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));

end;

procedure TForm3.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm3.DataSource1DataChange(Sender: TObject; Field: TField);
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>" then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
else
showmessage ('Dont find picture');

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from engineering';
query1.Active:=true;
edit1.Text:=";
end;

procedure TForm3.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from engineering where BookName like'+#39+(edit1.text)+'%'+#39);
query1.Open;

end;

procedure TForm3.Timer1Timer(Sender: TObject);
begin
```

```
label10.caption:=timetostr(time);
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
form6.Show;
end;

procedure TForm3.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm3.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm3.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm3.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm3.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm3.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;

procedure TForm3.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;

procedure TForm3.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm3.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
```

```pascal
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm3.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm3.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.


unit Unit4;

interface

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

type
  TForm4 = class(TForm)
    DataSource1: TDataSource;
    OpenPictureDialog1: TOpenPictureDialog;
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    GroupBox4: TGroupBox;
    DBEdit1: TDBEdit;
    GroupBox6: TGroupBox;
    Label3: TLabel;
    GroupBox11: TGroupBox;
    Button1: TButton;
    GroupBox16: TGroupBox;
    GroupBox17: TGroupBox;
    DBGrid1: TDBGrid;
    GroupBox20: TGroupBox;
    DBNavigator1: TDBNavigator;
    GroupBox21: TGroupBox;
    DBMemo1: TDBMemo;
    GroupBox15: TGroupBox;
    Image1: TImage;
```

```
GroupBox19: TGroupBox;
Button7: TButton;
GroupBox8: TGroupBox;
Label4: TLabel;
GroupBox9: TGroupBox;
Label5: TLabel;
GroupBox10: TGroupBox;
Label6: TLabel;
GroupBox22: TGroupBox;
Label7: TLabel;
GroupBox23: TGroupBox;
Label8: TLabel;
GroupBox26: TGroupBox;
Button2: TButton;
DBEdit8: TDBEdit;
GroupBox5: TGroupBox;
DBEdit2: TDBEdit;
GroupBox12: TGroupBox;
DBEdit3: TDBEdit;
GroupBox13: TGroupBox;
DBEdit4: TDBEdit;
GroupBox14: TGroupBox;
DBEdit5: TDBEdit;
GroupBox18: TGroupBox;
DBEdit6: TDBEdit;
GroupBox24: TGroupBox;
DBEdit7: TDBEdit;
GroupBox25: TGroupBox;
Label9: TLabel;
GroupBox27: TGroupBox;
Edit1: TEdit;
GroupBox28: TGroupBox;
Label10: TLabel;
DateTimePicker1: TDateTimePicker;
GroupBox29: TGroupBox;
Label11: TLabel;
GroupBox30: TGroupBox;
DBEdit9: TDBEdit;
GroupBox31: TGroupBox;
Button3: TButton;
Query1: TQuery;
Timer1: TTimer;
Timer2: TTimer;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure DataSource1DataChange(Sender: TObject; Field: TField);
procedure FormCreate(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
```

```
    procedure Button3Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
    procedure Timer2Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses Unit3, Unit5, Unit6, Unit7, Unit8, Unit9;

{$R *.dfm}

procedure TForm4.Button1Click(Sender: TObject);
begin
form4.Close;
end;

procedure TForm4.Button2Click(Sender: TObject);
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));
end;

procedure TForm4.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm4.DataSource1DataChange(Sender: TObject; Field: TField);
```

```
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>'' then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
else
showmessage ('Dont find picture');

end;

procedure TForm4.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from Politics';
query1.Active:=true;
edit1.Text:='';
end;

procedure TForm4.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from Politics where BookName like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm4.Timer1Timer(Sender: TObject);
begin
 label10.caption:=timetostr(time);
end;

procedure TForm4.Button3Click(Sender: TObject);
begin
form6.Show;
end;

procedure TForm4.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm4.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm4.DBEdit3KeyPress(Sender: TObject; var Key: Char);
```

```
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm4.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm4.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm4.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;

procedure TForm4.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;

procedure TForm4.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm4.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm4.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm4.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.


unit Unit5;
```

interface

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

type
  TForm5 = class(TForm)
    DataSource1: TDataSource;
    OpenPictureDialog1: TOpenPictureDialog;
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    GroupBox4: TGroupBox;
    DBEdit1: TDBEdit;
    GroupBox6: TGroupBox;
    Label3: TLabel;
    GroupBox11: TGroupBox;
    Button1: TButton;
    GroupBox16: TGroupBox;
    GroupBox17: TGroupBox;
    DBGrid1: TDBGrid;
    GroupBox20: TGroupBox;
    DBNavigator1: TDBNavigator;
    GroupBox21: TGroupBox;
    DBMemo1: TDBMemo;
    GroupBox15: TGroupBox;
    Image1: TImage;
    GroupBox19: TGroupBox;
    Button7: TButton;
    GroupBox8: TGroupBox;
    Label4: TLabel;
    GroupBox9: TGroupBox;
    Label5: TLabel;
    GroupBox10: TGroupBox;
    Label6: TLabel;
    GroupBox22: TGroupBox;
    Label7: TLabel;
    GroupBox23: TGroupBox;
    Label8: TLabel;
    GroupBox26: TGroupBox;
    Button2: TButton;
    DBEdit8: TDBEdit;
    GroupBox5: TGroupBox;
    DBEdit2: TDBEdit;
    GroupBox12: TGroupBox;

```
    DBEdit3: TDBEdit;
    GroupBox13: TGroupBox;
    DBEdit4: TDBEdit;
    GroupBox14: TGroupBox;
    DBEdit5: TDBEdit;
    GroupBox18: TGroupBox;
    DBEdit6: TDBEdit;
    GroupBox24: TGroupBox;
    DBEdit7: TDBEdit;
    GroupBox25: TGroupBox;
    Label9: TLabel;
    GroupBox27: TGroupBox;
    Edit1: TEdit;
    GroupBox28: TGroupBox;
    Label10: TLabel;
    DateTimePicker1: TDateTimePicker;
    GroupBox29: TGroupBox;
    Label11: TLabel;
    GroupBox30: TGroupBox;
    DBEdit9: TDBEdit;
    GroupBox31: TGroupBox;
    Button3: TButton;
    Query1: TQuery;
    Timer1: TTimer;
    Timer2: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure DataSource1DataChange(Sender: TObject; Field: TField);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
    procedure Timer2Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form5: TForm5;

implementation

uses Unit3, Unit4, Unit6, Unit7, Unit8, Unit9;

{$R *.dfm}

procedure TForm5.Button1Click(Sender: TObject);
begin
form5.close;
end;

procedure TForm5.Button2Click(Sender: TObject);
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));
end;

procedure TForm5.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm5.DataSource1DataChange(Sender: TObject; Field: TField);
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>" then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
else
showmessage ('Dont find picture');
end;

procedure TForm5.Timer1Timer(Sender: TObject);
begin
 label10.caption:=timetostr(time);
end;

procedure TForm5.FormCreate(Sender: TObject);
begin
```

```
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from physics';
query1.Active:=true;
edit1.Text:='';
end;

procedure TForm5.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from physics where BookName like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm5.Button3Click(Sender: TObject);
begin
form6.Show;
end;

procedure TForm5.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm5.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm5.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm5.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm5.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm5.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;
```

73

```
procedure TForm5.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;

procedure TForm5.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm5.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm5.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm5.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.

unit Unit7;

interface

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

type
  TForm7 = class(TForm)
    OpenPictureDialog1: TOpenPictureDialog;
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    GroupBox4: TGroupBox;
    DBEdit1: TDBEdit;
    GroupBox6: TGroupBox;
    Label3: TLabel;
    GroupBox11: TGroupBox;
```

```
Button1: TButton;
GroupBox16: TGroupBox;
GroupBox17: TGroupBox;
DBGrid1: TDBGrid;
GroupBox20: TGroupBox;
DBNavigator1: TDBNavigator;
GroupBox21: TGroupBox;
DBMemo1: TDBMemo;
GroupBox15: TGroupBox;
Image1: TImage;
GroupBox19: TGroupBox;
Button7: TButton;
GroupBox8: TGroupBox;
Label4: TLabel;
GroupBox9: TGroupBox;
Label5: TLabel;
GroupBox10: TGroupBox;
Label6: TLabel;
GroupBox22: TGroupBox;
Label7: TLabel;
GroupBox23: TGroupBox;
Label8: TLabel;
GroupBox26: TGroupBox;
Button2: TButton;
DBEdit8: TDBEdit;
GroupBox5: TGroupBox;
DBEdit2: TDBEdit;
GroupBox12: TGroupBox;
DBEdit3: TDBEdit;
GroupBox13: TGroupBox;
DBEdit4: TDBEdit;
GroupBox14: TGroupBox;
DBEdit5: TDBEdit;
GroupBox18: TGroupBox;
DBEdit6: TDBEdit;
GroupBox24: TGroupBox;
DBEdit7: TDBEdit;
GroupBox25: TGroupBox;
Label9: TLabel;
GroupBox27: TGroupBox;
Edit1: TEdit;
GroupBox28: TGroupBox;
Label10: TLabel;
DateTimePicker1: TDateTimePicker;
GroupBox29: TGroupBox;
Label11: TLabel;
GroupBox30: TGroupBox;
DBEdit9: TDBEdit;
GroupBox31: TGroupBox;
Button3: TButton;
```

```
    Query1: TQuery;
    DataSource1: TDataSource;
    Timer1: TTimer;
    Timer2: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure DataSource1DataChange(Sender: TObject; Field: TField);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
    procedure Timer2Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form7: TForm7;

implementation

uses Unit3, Unit4, Unit5, Unit6, Unit8, Unit9;

{$R *.dfm}

procedure TForm7.Button1Click(Sender: TObject);
begin
form7.Close;
end;

procedure TForm7.Button2Click(Sender: TObject);
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));

end;
```

```
procedure TForm7.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm7.DataSource1DataChange(Sender: TObject; Field: TField);
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>'' then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
else
showmessage ('Dont find picture');

end;
procedure TForm7.Timer1Timer(Sender: TObject);
begin
label10.caption:=timetostr(time);
end;

procedure TForm7.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from computers';
query1.Active:=true;
edit1.Text:='';
end;

procedure TForm7.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from computers where BookName like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm7.Button3Click(Sender: TObject);
begin
form6.Show;
end;
```

```
procedure TForm7.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm7.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm7.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm7.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm7.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm7.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;

procedure TForm7.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;

procedure TForm7.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm7.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm7.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;
```

```
procedure TForm7.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.

unit Unit8;

interface

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

type
  TForm8 = class(TForm)
    DataSource1: TDataSource;
    OpenPictureDialog1: TOpenPictureDialog;
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    GroupBox4: TGroupBox;
    DBEdit1: TDBEdit;
    GroupBox6: TGroupBox;
    Label3: TLabel;
    GroupBox11: TGroupBox;
    Button1: TButton;
    GroupBox16: TGroupBox;
    GroupBox17: TGroupBox;
    DBGrid1: TDBGrid;
    GroupBox20: TGroupBox;
    DBNavigator1: TDBNavigator;
    GroupBox21: TGroupBox;
    DBMemo1: TDBMemo;
    GroupBox15: TGroupBox;
    Image1: TImage;
    GroupBox19: TGroupBox;
    Button7: TButton;
    GroupBox8: TGroupBox;
    Label4: TLabel;
    GroupBox9: TGroupBox;
    Label5: TLabel;
    GroupBox10: TGroupBox;
    Label6: TLabel;
    GroupBox22: TGroupBox;
```

```
Label7: TLabel;
GroupBox23: TGroupBox;
Label8: TLabel;
GroupBox26: TGroupBox;
Button2: TButton;
DBEdit8: TDBEdit;
GroupBox5: TGroupBox;
DBEdit2: TDBEdit;
GroupBox12: TGroupBox;
DBEdit3: TDBEdit;
GroupBox13: TGroupBox;
DBEdit4: TDBEdit;
GroupBox14: TGroupBox;
DBEdit5: TDBEdit;
GroupBox18: TGroupBox;
DBEdit6: TDBEdit;
GroupBox24: TGroupBox;
DBEdit7: TDBEdit;
GroupBox25: TGroupBox;
Label9: TLabel;
GroupBox27: TGroupBox;
Edit1: TEdit;
GroupBox28: TGroupBox;
Label10: TLabel;
DateTimePicker1: TDateTimePicker;
GroupBox29: TGroupBox;
Label11: TLabel;
GroupBox30: TGroupBox;
DBEdit9: TDBEdit;
GroupBox31: TGroupBox;
Button3: TButton;
Query1: TQuery;
Timer1: TTimer;
Timer2: TTimer;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure DataSource1DataChange(Sender: TObject; Field: TField);
procedure Timer1Timer(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
```

```
  procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
  procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
  procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
  procedure Timer2Timer(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form8: TForm8;

implementation

uses Unit3, Unit4, Unit5, Unit6, Unit7, Unit9;

{$R *.dfm}

procedure TForm8.Button1Click(Sender: TObject);
begin
form8.Close;
end;

procedure TForm8.Button2Click(Sender: TObject);
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));

end;

procedure TForm8.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm8.DataSource1DataChange(Sender: TObject; Field: TField);
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>" then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
```

81

```
else
showmessage ('Dont find picture');

end;

procedure TForm8.Timer1Timer(Sender: TObject);
begin
label10.caption:=timetostr(time);
end;

procedure TForm8.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from civil_engineering';
query1.Active:=true;
edit1.Text:='';
end;

procedure TForm8.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from civil_engineering where BookName
like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm8.Button3Click(Sender: TObject);
begin
form6.Show;
end;

procedure TForm8.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm8.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm8.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm8.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
```

```pascal
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm8.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm8.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;

procedure TForm8.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;

procedure TForm8.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm8.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm8.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm8.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.

unit Unit9;

interface

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

type
```

```
TForm9 = class(TForm)
  OpenPictureDialog1: TOpenPictureDialog;
  DataSource1: TDataSource;
  GroupBox7: TGroupBox;
  GroupBox1: TGroupBox;
  GroupBox2: TGroupBox;
  Label1: TLabel;
  GroupBox3: TGroupBox;
  Label2: TLabel;
  GroupBox4: TGroupBox;
  DBEdit1: TDBEdit;
  GroupBox6: TGroupBox;
  Label3: TLabel;
  GroupBox11: TGroupBox;
  Button1: TButton;
  GroupBox16: TGroupBox;
  GroupBox17: TGroupBox;
  DBGrid1: TDBGrid;
  GroupBox20: TGroupBox;
  DBNavigator1: TDBNavigator;
  GroupBox21: TGroupBox;
  DBMemo1: TDBMemo;
  GroupBox15: TGroupBox;
  Image1: TImage;
  GroupBox19: TGroupBox;
  Button7: TButton;
  GroupBox8: TGroupBox;
  Label4: TLabel;
  GroupBox9: TGroupBox;
  Label5: TLabel;
  GroupBox10: TGroupBox;
  Label6: TLabel;
  GroupBox22: TGroupBox;
  Label7: TLabel;
  GroupBox23: TGroupBox;
  Label8: TLabel;
  GroupBox26: TGroupBox;
  Button2: TButton;
  DBEdit8: TDBEdit;
  GroupBox5: TGroupBox;
  DBEdit2: TDBEdit;
  GroupBox12: TGroupBox;
  DBEdit3: TDBEdit;
  GroupBox13: TGroupBox;
  DBEdit4: TDBEdit;
  GroupBox14: TGroupBox;
  DBEdit5: TDBEdit;
  GroupBox18: TGroupBox;
  DBEdit6: TDBEdit;
  GroupBox24: TGroupBox;
```

```
    DBEdit7: TDBEdit;
    GroupBox25: TGroupBox;
    Label9: TLabel;
    GroupBox27: TGroupBox;
    Edit1: TEdit;
    GroupBox28: TGroupBox;
    Label10: TLabel;
    DateTimePicker1: TDateTimePicker;
    GroupBox29: TGroupBox;
    Label11: TLabel;
    GroupBox30: TGroupBox;
    DBEdit9: TDBEdit;
    GroupBox31: TGroupBox;
    Button3: TButton;
    Query1: TQuery;
    Timer1: TTimer;
    Timer2: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure DataSource1DataChange(Sender: TObject; Field: TField);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
    procedure Timer2Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form9: TForm9;

implementation

uses Unit3, Unit4, Unit5, Unit6, Unit7, Unit8;

{$R *.dfm}
```

```
procedure TForm9.Button1Click(Sender: TObject);
begin
form9.Close;
end;

procedure TForm9.Button2Click(Sender: TObject);
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));

end;

procedure TForm9.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm9.DataSource1DataChange(Sender: TObject; Field: TField);
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>" then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
else
showmessage ('Dont find picture');

end;

procedure TForm9.Timer1Timer(Sender: TObject);
begin
label10.caption:=timetostr(time);
end;

procedure TForm9.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from popular_sienence';
query1.Active:=true;
edit1.Text:=";
end;
```

```
procedure TForm9.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from popular_sienence where BookName
like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm9.Button3Click(Sender: TObject);
begin
form6.Show;
end;

procedure TForm9.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm9.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm9.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm9.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm9.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm9.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;

procedure TForm9.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;
```

```
procedure TForm9.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm9.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm9.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm9.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.

unit Unit10;

interface

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

type
  TForm10 = class(TForm)
    GroupBox7: TGroupBox;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    GroupBox4: TGroupBox;
    DBEdit1: TDBEdit;
    GroupBox6: TGroupBox;
    Label3: TLabel;
    GroupBox11: TGroupBox;
    Button1: TButton;
    GroupBox16: TGroupBox;
    GroupBox17: TGroupBox;
    DBGrid1: TDBGrid;
    GroupBox20: TGroupBox;
    DBNavigator1: TDBNavigator;
```

```
GroupBox21: TGroupBox;
DBMemo1: TDBMemo;
GroupBox15: TGroupBox;
Image1: TImage;
GroupBox19: TGroupBox;
Button7: TButton;
GroupBox8: TGroupBox;
Label4: TLabel;
GroupBox9: TGroupBox;
Label5: TLabel;
GroupBox10: TGroupBox;
Label6: TLabel;
GroupBox22: TGroupBox;
Label7: TLabel;
GroupBox23: TGroupBox;
Label8: TLabel;
GroupBox26: TGroupBox;
Button2: TButton;
DBEdit8: TDBEdit;
GroupBox5: TGroupBox;
DBEdit2: TDBEdit;
GroupBox12: TGroupBox;
DBEdit3: TDBEdit;
GroupBox13: TGroupBox;
DBEdit4: TDBEdit;
GroupBox14: TGroupBox;
DBEdit5: TDBEdit;
GroupBox18: TGroupBox;
DBEdit6: TDBEdit;
GroupBox24: TGroupBox;
DBEdit7: TDBEdit;
GroupBox25: TGroupBox;
Label9: TLabel;
GroupBox27: TGroupBox;
Edit1: TEdit;
GroupBox28: TGroupBox;
Label10: TLabel;
DateTimePicker1: TDateTimePicker;
GroupBox29: TGroupBox;
Label11: TLabel;
GroupBox30: TGroupBox;
DBEdit9: TDBEdit;
GroupBox31: TGroupBox;
Button3: TButton;
Query1: TQuery;
DataSource1: TDataSource;
OpenPictureDialog1: TOpenPictureDialog;
Timer1: TTimer;
Timer2: TTimer;
procedure Button1Click(Sender: TObject);
```

```
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure DataSource1DataChange(Sender: TObject; Field: TField);
    procedure Edit1Change(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBMemo1KeyPress(Sender: TObject; var Key: Char);
    procedure Timer2Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;

implementation

uses Unit6;

{$R *.dfm}

procedure TForm10.Button1Click(Sender: TObject);
begin
form10.Close;
end;

procedure TForm10.Timer1Timer(Sender: TObject);
begin
label10.caption:=timetostr(time);
end;

procedure TForm10.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from communications';
query1.Active:=true;
edit1.Text:='';
```

```pascal
end;

procedure TForm10.DataSource1DataChange(Sender: TObject; Field: TField);
var
pic:string;
begin
pic:=Query1.fieldbyname('Picture').AsString;
if pic<>'' then
if fileexists (pic) then
image1.Picture.loadfromfile(pic)
else
showmessage ('Dont find picture');
end;

procedure TForm10.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from communications where BookName
like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm10.Button2Click(Sender: TObject);
begin
dbedit8.Text:=inttostr(strtoint(dbedit7.Text)- strtoint(dbedit6.text));
dbedit9.Text:=inttostr(strtoint(dbedit9.Text)- strtoint('1'));

end;

procedure TForm10.Button7Click(Sender: TObject);
begin
if openpicturedialog1.Execute then
begin
Query1.Edit;
Image1.picture.loadfromfile(openpicturedialog1.filename);
Query1.FieldByName('picture').asstring:=openpicturedialog1.FileName;
Query1.post;
end;
end;

procedure TForm10.Button3Click(Sender: TObject);
begin
form6.Show;
end;

procedure TForm10.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;
```

```
procedure TForm10.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm10.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm10.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit9.SetFocus;
end;

procedure TForm10.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm10.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;

procedure TForm10.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit7.SetFocus;
end;

procedure TForm10.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit8.SetFocus;
end;

procedure TForm10.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbmemo1.SetFocus;
end;

procedure TForm10.DBMemo1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm10.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;
```

end.

unit Unit6;

interface

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

type
  TForm6 = class(TForm)
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    GroupBox3: TGroupBox;
    GroupBox4: TGroupBox;
    GroupBox5: TGroupBox;
    GroupBox6: TGroupBox;
    GroupBox7: TGroupBox;
    GroupBox8: TGroupBox;
    GroupBox9: TGroupBox;
    GroupBox10: TGroupBox;
    GroupBox11: TGroupBox;
    GroupBox12: TGroupBox;
    Label1: TLabel;
    Label6: TLabel;
    Label5: TLabel;
    Label4: TLabel;
    Label3: TLabel;
    Label2: TLabel;
    GroupBox13: TGroupBox;
    GroupBox14: TGroupBox;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;
    DBEdit5: TDBEdit;
    DBEdit6: TDBEdit;
    Edit1: TEdit;
    Label7: TLabel;
    GroupBox15: TGroupBox;
    DBNavigator1: TDBNavigator;
    GroupBox16: TGroupBox;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DataSource1: TDataSource;
    GroupBox17: TGroupBox;

```
    Label8: TLabel;
    GroupBox18: TGroupBox;
    Edit2: TEdit;
    Image1: TImage;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Timer1: TTimer;
    DateTimePicker1: TDateTimePicker;
    Label12: TLabel;
    Timer2: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure Label12Click(Sender: TObject);
    procedure Timer2Timer(Sender: TObject);



  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form6: TForm6;

implementation

uses Unit3, Unit4, Unit5, Unit7, Unit8, Unit9;

{$R *.dfm}


procedure TForm6.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='STANDARD2';
Query1.requestlive:=true;
query1.SQL.Text:='select * from customer';
query1.Active:=true;
```

```
edit1.Text:=";
edit2.Text:=";

end;

procedure TForm6.Edit1Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from customer where Name like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;

procedure TForm6.Edit2Change(Sender: TObject);
begin
query1.close;
query1.SQL.Clear;
query1.sql.add('select * from customer where RegisterCode like'+#39+(edit2.text)+'%'+#39);
query1.Open;
end;

procedure TForm6.Timer1Timer(Sender: TObject);
begin
label11.Caption:=timetostr(time);
end;

procedure TForm6.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit2.SetFocus;
end;

procedure TForm6.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit3.SetFocus;
end;

procedure TForm6.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit4.SetFocus;
end;

procedure TForm6.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit5.SetFocus;
end;

procedure TForm6.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbedit6.SetFocus;
end;
```

```
procedure TForm6.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then dbnavigator1.SetFocus;
end;

procedure TForm6.Label12Click(Sender: TObject);
begin
form6.Close;
end;

procedure TForm6.Timer2Timer(Sender: TObject);
begin
datetimepicker1.Format:=datetostr(date);
end;

end.

unit Unit12;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;

type
  TForm12 = class(TForm)
    GroupBox1: TGroupBox;
    Image1: TImage;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    procedure Image1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form12: TForm12;

implementation

uses Unit11;
```

```
{$R *.dfm}

procedure TForm12.Image1Click(Sender: TObject);
begin
form11.Show;
form12.Close;
end;

procedure TForm12.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit11;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls;

type
  TForm11 = class(TForm)
    GroupBox1: TGroupBox;
    Image1: TImage;
    procedure Image1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form11: TForm11;
implementation
uses Unit12;

{$R *.dfm}

procedure TForm11.Image1Click(Sender: TObject);
begin
form12.show;
form11.close;
end;

procedure TForm11.FormCreate(Sender: TObject);
begin
```