



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

**Ceylan Airlines Company Database Management
System**

**Graduation Project
COM-400**

Student: Yusuf Ceylan (20030732)

Supervisor: Assist.Prof.Dr.Adil AMIRJANOV

Nicosia - 2008

ACKNOWLEDGMENTS

Firstly, I would like to thank to my supervisor Dr.Adil AMIRJANOV, my advisor Dr.Kaan UYAR, and Mr Elbrus IMANOV,for their great advice and recommendation on finishing my project and guiding me in their lectures.

I am very grateful to my family for their endless support from the first day in my educational life until today. I will never forget the things that my father Mr.Nevzat CEYLAN did for me during my educational life, also I want to say thanks to my mother Mrs. Nurhan CEYLAN.

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

I thank my friends Murat EVEREKLI,Muhammet Emin TUTKUN, Mehmet TAHTA,Mehmet Akif GURSOY, Gökan CİL, Tahir CICEKLI, for their help, they get tired with me, and they helped me and give morale evvertime.

I thank them with all my heart.

ABSTRACT

The aim of this project is to design airlines database management system that contain information forms including flight,passenger,employee and airport.The forms are reached via logging into the database system. The program was prepared by using Delphi programming and using database. This program is practical and useful in the airlines business.It can be modified with the new developments and requirements of people in the technology in future.The program must be clear and easy to learn for users,hence it can be acceptable widely.

TABLE OF CONTENTS

<u>ACKNOWLEDGMENT</u>	i
<u>ABSTRACT</u>	ii
<u>TABLE OF CONTENTS</u>	iii

CHAPTER 1

INTRODUCTION 1

1. BASIC CONCEPT OF DELPHI

1.1. Introduction to Delphi	2
1.2. What is Delphi?	2
1.3. A Tour Of The Environment	3
1.3.1. Running Delphi For The First Time	3
1.3.2. The Delphi IDE	3
1.3.3. The Menus & Toolbar	4
1.3.4. The Component Palette	5
1.3.5. The Code Editor	6
1.3.6. The Object Inspector	7
1.3.7. The Object TreeView	8
1.3.8. Class Completion	9
1.3.9. Debugging applications	10
1.3.10. Exploring databases	11
1.3.11. Templates and the Object Repository	11
1.4. Programming With Delphi	13
1.4.1. Starting a New Application	13
1.4.1.1. Setting Property Values	14
1.4.2. Adding objects to the form	15
1.4.3. Add a Table and a StatusBar to the form	15
1.4.4. Add all include Standard Component to the form	17
1.4.5. Connecting to a Database	22

CHAPTER 2

2. DATA BASE SYSTEM

2.1. INTRODUCTION TO DATABASE	25
2.2. DATABASE MODELS	26
2.2.1. Relational model	27
2.2.1.1. Relational operations	28
2.3. SQL in DELPHI	29

CHAPTER 3

3. CEYLAN AIRLINE COMPANY DATABASE MANAGEMENT SYSTEM USERS MANUAL

3.1. LOGIN	32
3.2. USER	35
3.3. ADMIN	39

<u>CONCLUSION</u>	43
--------------------------	-----------

<u>REFERENCES</u>	44
--------------------------	-----------

<u>APPENDIX</u>	45
------------------------	-----------

CHAPTER 1

INTRODUCTION

This project is ceylan airlines company database management system, which begins with the login page and lead the users to the necessary forms for applications. But before all this progress, the users must be registered on the system to achieve what they want to do. After registration, the user first meets the passenger information form, then completes it and chooses the appropriate flight and departure for himself, makes the payment by credit card and gets the report about the ticket eventually. If we log into the system as admin, admin page automatically will be called. Each form is reachable through admin page, so admin can change any information in the system. He can even make a user an admin. And this program was prepared by using Borland Delphi 7 and PARADOX.

The subjects are chapter by chapter so let us go through the overview. The chapters are briefly:

In the first chapter Borland Delphi 7 programming language is described, its properties, components and some examples, I used Borland Delphi 7 in my project, because I find it easy and I liked its coding system. I think Borland Delphi 7 is easier compared to the other programming languages for applications.

In the Second Chapter I described Database system, I used PARADOX data base system in my program with Borland Delphi 7.

Third Chapter forms the main body of the project, it includes the total applications about what we aim here. The relationship between the whole datas (user-admin relation, the database connections between information forms) and what each information form is used for is explained.

Finally, the last chapter is the explanation of the program followed by the Appendixes. The codes forming the main program are recorded here. If we describe the functions of the codes, they are used to combine everything and show the relationship between any datas whatever they are. So by developing and moderating the technology of our program, it can be updated. Also new properties could be added into the program in the future.

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 7. Delphi 7 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.

1.3. A Tour Of The Environment:

This chapter explains how to start Delphi and gives you a quick tour of the main parts and tools of the Integrated Development Environment(IDE)

1.3.1. Running Delphi For The First Time:

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

- Choose Programs | Borland Delphi 7| Delphi 7 from the Windows Start menu
- Choose Run from the Windows Start menu and type Delphi32
- Double-click Delphi32.exe in the \$(DELPHI)\Bin folder. Where \$(DELPHI) is a folder where Delphi was installed. The default is C:\Program Files\Borland\Delphi7.
- Double-click the Delphi icon on the Desktop (if you've created a shortcut)

1.3.2. The Delphi IDE:

As explained before, one of the ways to start Delphi is to choose Programs | Borland Delphi 7 | Delphi 7 from the Windows Start menu.

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

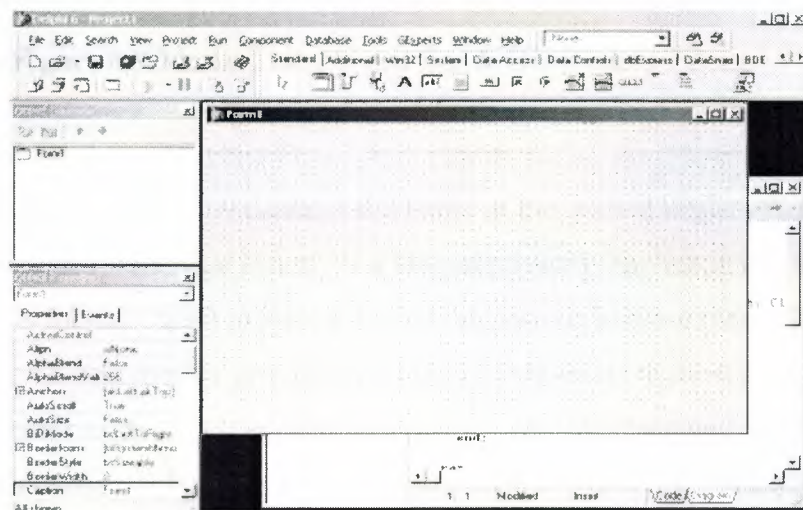


Figure 1.8.IDE

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

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

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

1.3.3. The Menus & Toolbar:

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

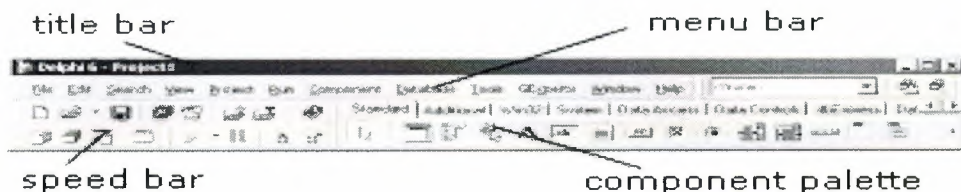


Figure 1.9.Menu ,Title , Speed Bar & Component Palette

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.

1.3.4. The Component Palette:

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

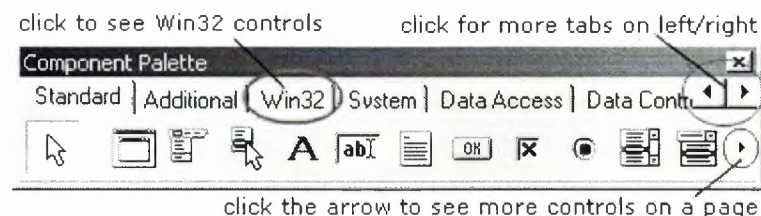


Figure 1.10.Component Palatte

Depending on the version of Delphi (assumed Delphi 7 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.

1.3.5. The Code Editor :

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

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

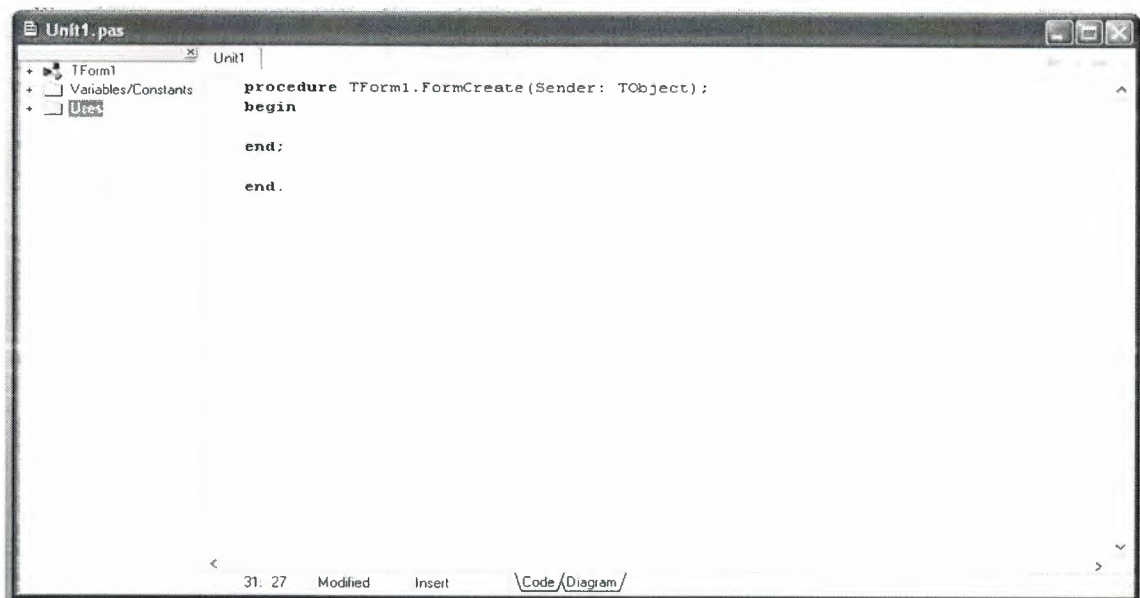


Fig.1.11.Code Editor Window

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

1.3.6. 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.

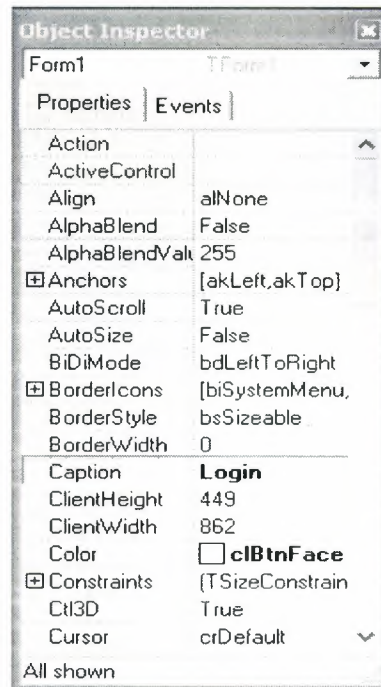


Figure 1.11.Object Inspector

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.

1.3.7. The Object TreeView:

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

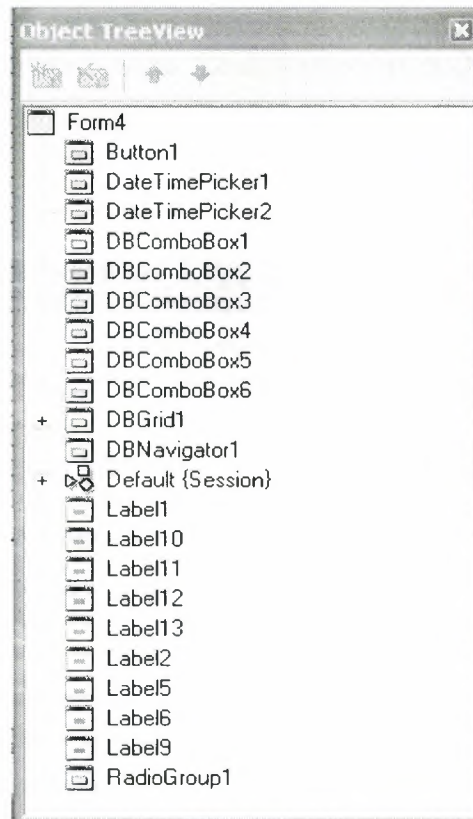


Figure 1.12.Object Tree View

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

1.3.8. 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.

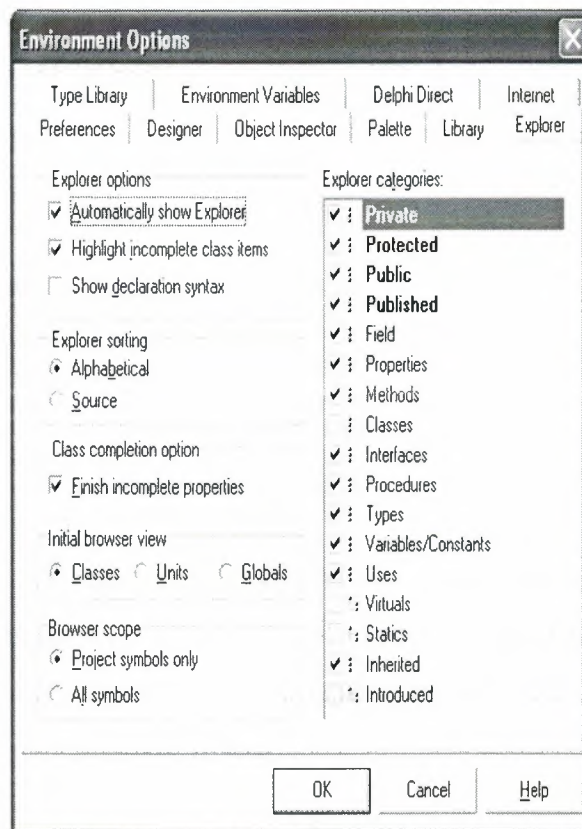
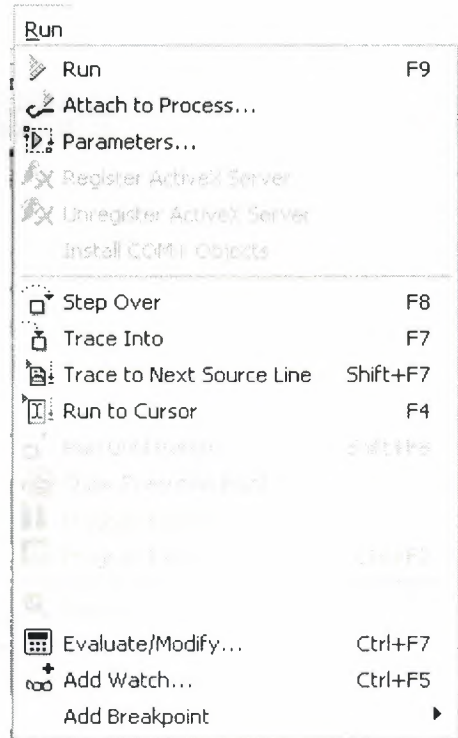


Fig.1.13.Class

1.3.9.Debugging applications:

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



Choose any of the debugging commands from the Run menu.

Some commands are also available on the toolbar.



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](#)".

1.3.10.Exploring databases:

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

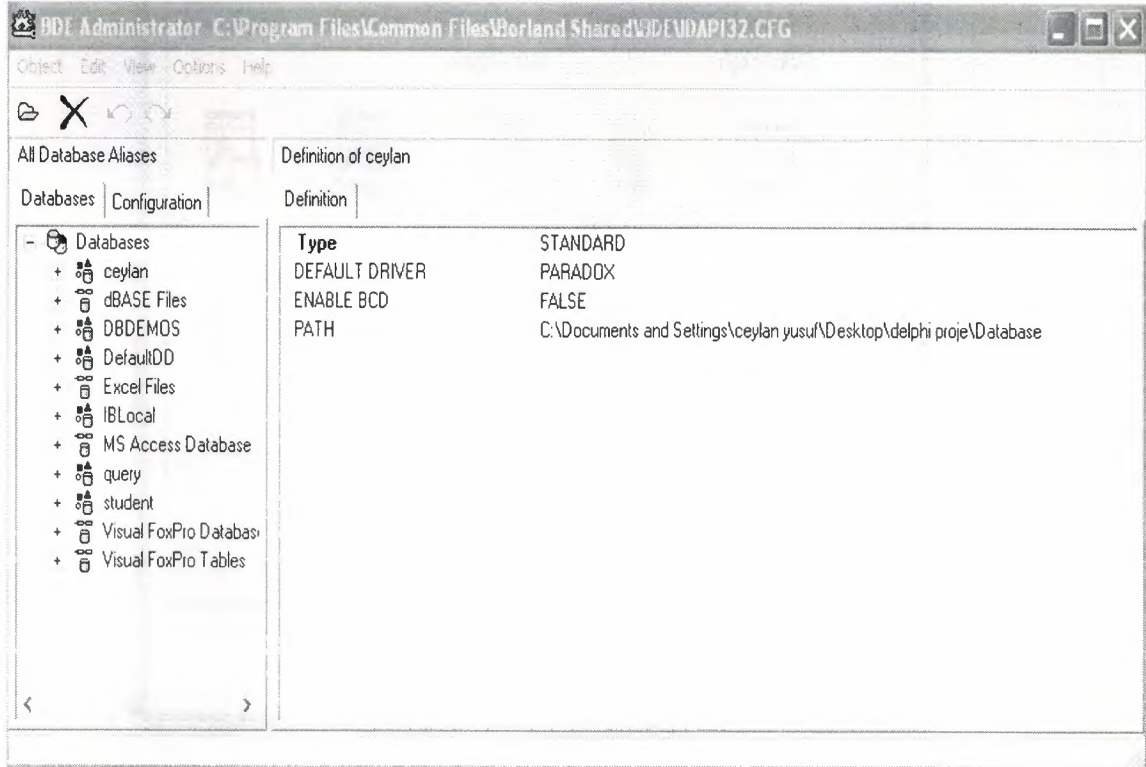


Figure 1.15.SQL Explorer

1.3.11.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.

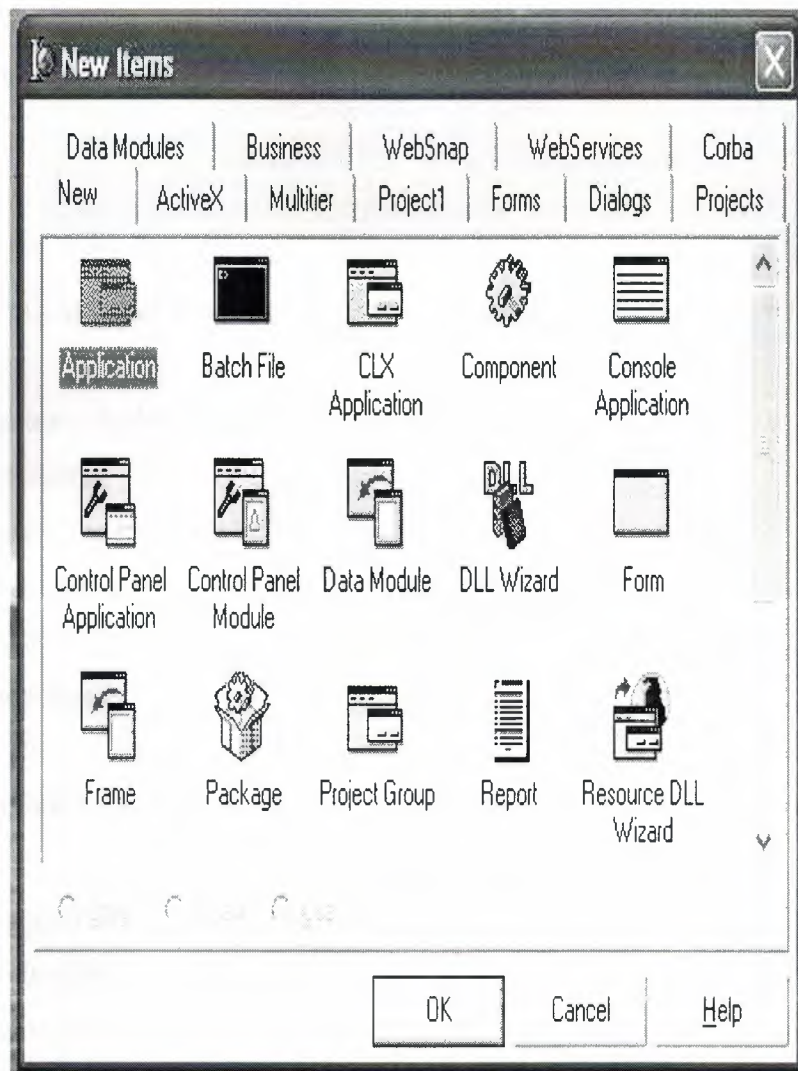


Figure 1.16.New Item

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.

1.4.Programming With Delphi:

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

1.4.1.Starting a New Application:

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

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

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

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

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

Each form has its own unit and form files.

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

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

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

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

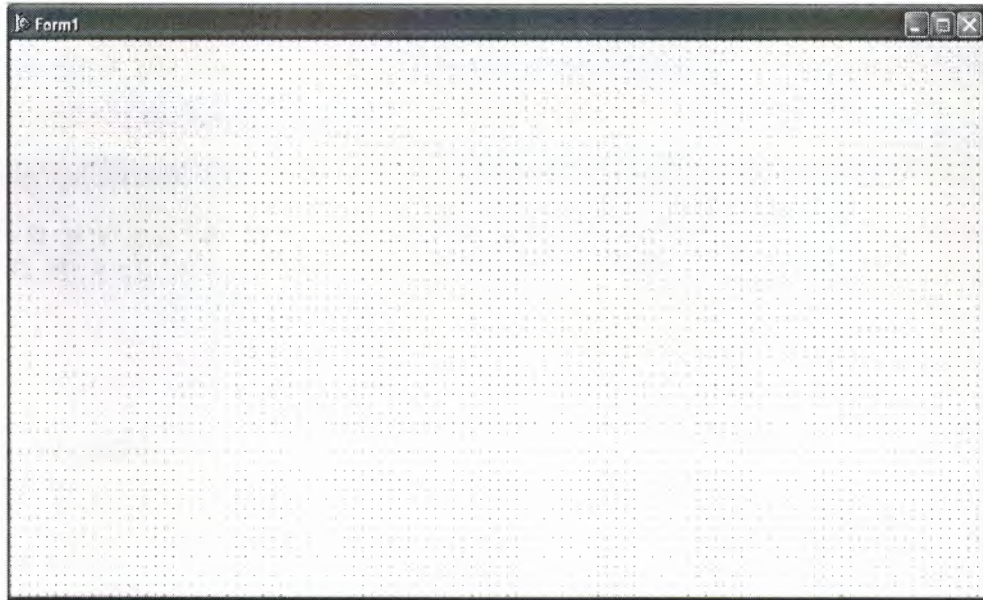


Figure 1.17.Form Screen

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

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

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

1.4.1.1. 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.

1.4.2. Adding objects to the form:

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

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

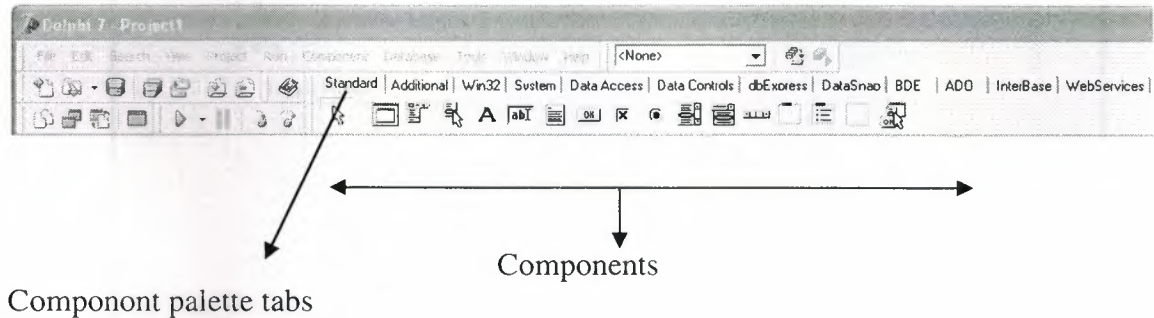


Figure 1.18. Standart Button

1.4.3. 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.

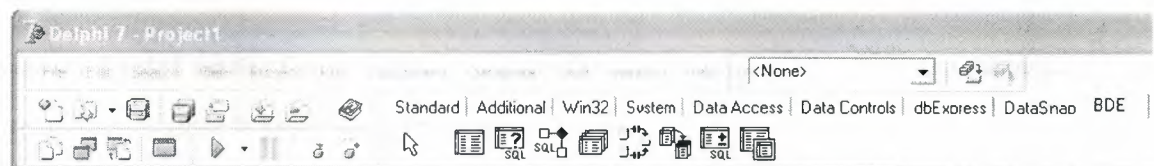


Fig.1.19.BDE Component palette

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.)

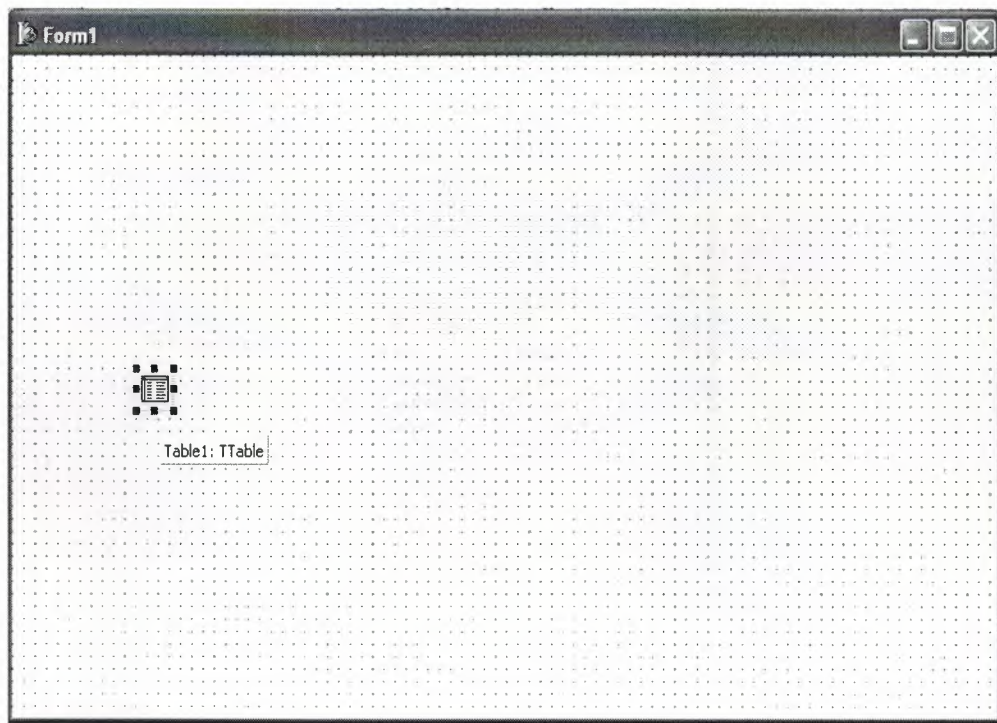


Figure 1.20.Table In The Form

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 ceylan. (ceylan 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 ceylan from the drop-down list.

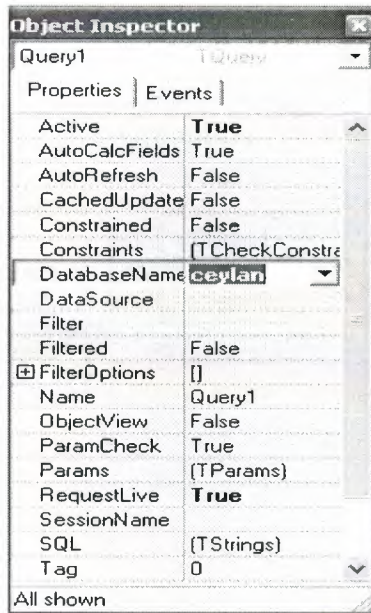


Fig.1.21.Select DatabaseName

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.)

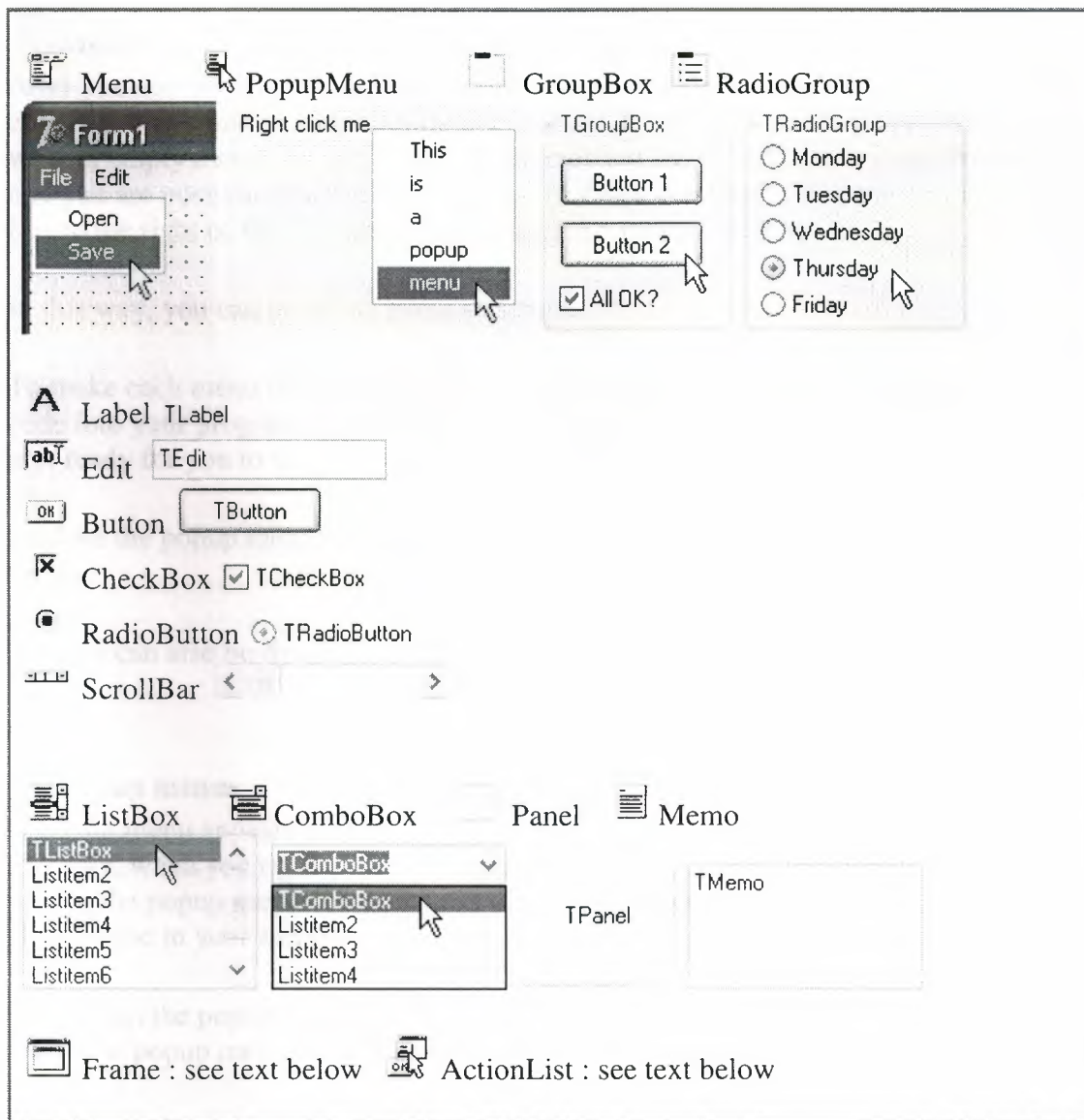
1.4.4.Add all include Standard Component to the form:

GUI stands for Graphical User Interface. It refers to the windows, buttons, dialogs, menus and everything visual in a modern application. A GUI component is one of these graphical building blocks. Delphi lets you build powerful applications using a rich variety of these components.

These components are grouped under a long set of tabs in the top part of the Delphi screen, starting with **Standard** at the left. We'll look at this Standard tab here. It looks something like this (Delphi allows you to tinker with nearly everything in its interface, so it may look different on your system):



Each of the components is itemised below with a picture of a typical GUI object they can create:



Frame objects

These were introduced in Delphi 5. They represent a powerful mechanism, albeit one that is a little advanced for a Delphi Basics site. However, it is worth describing their role if you want to research further.

A frame is essentially a new object. It is defined using the **File|New** menu. Only then can you add the frame to your form using the **Frame** component. You can add the same frame to as many forms of your application as you want. This is because the frame is designed as a kind of template for a part of a form. It allows you to define the same look and feel for that part of each form. And more importantly, each instance of the frame inherits everything from the original frame.

For further reading, **Mastering Delphi** by Cantu covers this topic with example code.



Menus

After you add a TMenu component to your form, you can design the menu by double clicking it (or using the right button popup menu for it). You are then shown a panel with an empty menu. As you type, you are creating the top left menu item. Press enter and you are positioned at the first sub item of this menu item. Click the new empty box to the right of the first menu item to create a new menu item.

In this way, you can build the menu structure.

To make each menu item do something, just double click it. Delphi will then insert code into your program to handle the menu item, and position your cursor in the form unit ready for you to write your code.

Explore the popup menu for the menu editor to discover more options, such as sub-menus.

A menu can also be dynamically updated by your code.



Popup menus

A popup menu appears in many applications when you right click on something. For example, when you right click the Windows desktop. You create a popup menu by adding the popup menu component to your form and double clicking it. You then simply type in your menu item list.

You attach the popup menu to an existing form object (or the form itself) by selecting your new popup menu in the **PopupMenu** property of the object.

To activate the popup menu items, double click each in turn. Delphi will add the appropriate code to your form unit. You can then type in the code that each menu item should perform.

A popup menu can also be dynamically updated by your code.

A Labels

Labels are the simplest component. They are used to literally label things on a form, but the text, colours and so on can be changed by your code. For example, you can change the label colour when the mouse hovers over it, and can run code when the user clicks it. This makes the label like a web page link. Normally, they are just kept as plain, unchanging text.

Edit boxes

An edit box allows the user to type in a single line of text. For example, the name of the user. You set up the initial value with the **Text** property either at design time or when your code runs.

Memo boxes

A memo box displays a single string as a multi line wrapped text display. You cannot apply any formatting. The displayed lines are set using the **Lines** property. This may be set at design time as well as at run time.

Buttons

A button is the simplest active item. When clicked by a user, it performs some action. You can change the button label by setting the **Caption** property. Double clicking the button when designing adds code to your form to run when the button is clicked at run time.

Check boxes

Check boxes are used to give a user a **yes/no** choice. For example, whether to wrap text or not. The label is set using the **Caption** property. You can preset the check box to ticked by setting the **Checked** property to **true**.

Radio buttons

Radio buttons are used to give a user **multiple** choices. For example, whether to left, centre or right align text. The label is set using the **Caption** property. You can preset a radio button to selected by setting the **Checked** property to **true**.

You would normally use radio buttons in groups of two or more. The **TRadioGroup** component allows you to do this in a neat and dynamic way.

List boxes

List boxes provide selectable items. For example, a collection of fish names. If you set the **MultiSelect** property to **true**, you allow the user to select more than one. The items in the list are added using the **Items.Add** method, passing the string of each item as a parameter.

You can act upon an item being selected by setting the **OnClick** event (by double clicking it) to a procedure in your form unit.

The following example displays the selected list item in a dialog box:

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
  listBox : TListBox;
  index   : Integer;
begin
  // Cast the passed object to its correct type
  listBox := TListBox(Sender);

  // Get the index of the selected list item
  index := listBox.ItemIndex;

  // Display the selected list item value
  ShowMessage(listBox.Items[index]);
end;
```

Combo boxes

A combo box is like a list box, and is set up in the same way (see above). It just takes up less space on your form by collapsing to a single line when deselected, showing the chosen list item. It is not recommend to use one for multi line selection.

Scroll bars

Many components have built in scroll bars. For those that don't, you can use this to do your own scrolling. You link the scrollbar to your component by setting the **OnScroll** event. This gives you the details of the last scroll activity made by the user.

Group boxes

A group box is like a panel. It differs in that it gives a name to the collection of components that you add to it. This title is set with the **Caption** property. Use a group box to help the user see what controls affect one particular aspect of the application.

Radio group panels

Radio buttons are used to give a user a **multiple** choices. For example, whether to left, centre or right align text. Unlike individual radio buttons, a group is only set up by your code. You define the buttons by calling the **Items.Add** method of the TRadioGroup object, passing the caption string of each radio button as a parameter. You can reference each button by using the **Buttons** indexed property. You might, for example, choose the third button to be checked. For example :

```
// Set the third button to be pre-selected (index starts at 0)
RadioGroup1.Buttons[2].Checked := true;
```

Empty panels

When building your form, you might want to add many components. These may fall into logical groups. If so, you can add each group to a panel, and use the panel to position the whole group on the form. The panel name can be blanked out by setting the **Caption** property.

You can even hide the panel by setting the **BevelOuter** and **BevelInner** properties to **bvNone**.

Action lists

Action lists are a large topic on their own. They allow you to define, for example, menus with sub-items that are also shown as buttons on your application. Only one **action** is defined, regardless of the number of references to it.

1.4.5. Connecting to a Database:

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

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

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

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

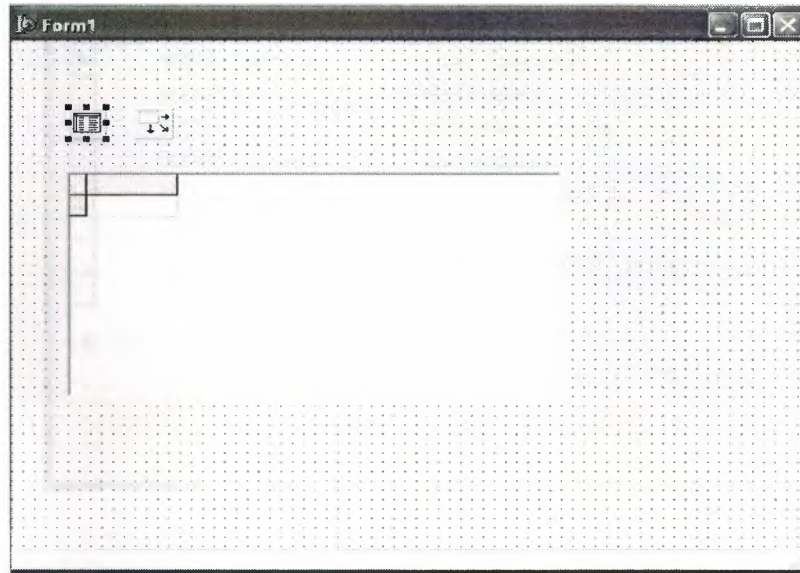


Figure 1.22.DBGrid In The Form

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

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

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

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

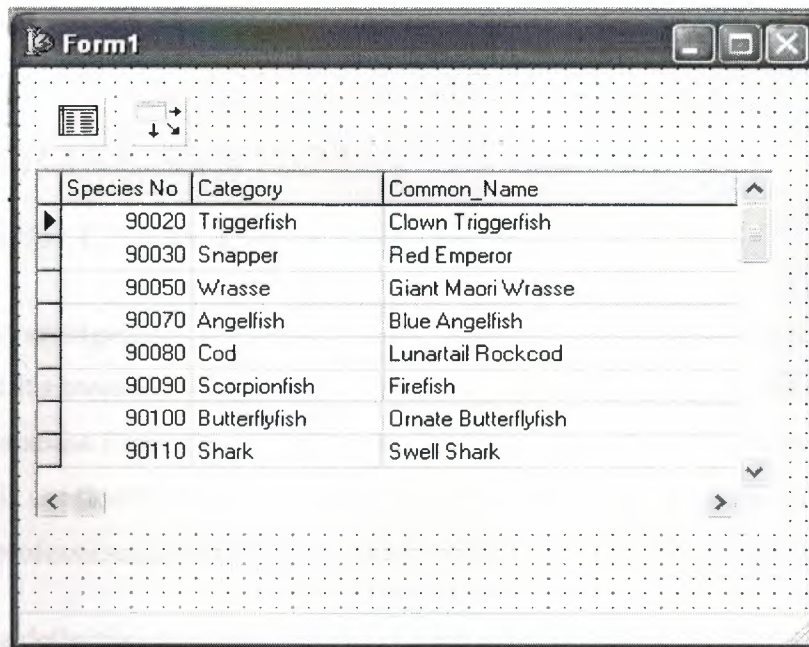


Figure 1.23.Show Table

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

6. Press F9 to compile and run the project. (You can also run the project by clicking the Run button on the Debug toolbar, or by choosing Run from the Run menu.)
7. In connecting our application to a database, we've used three components and several levels of indirection. A data-aware control (in this case, a DBGrid) points to a DataSource object, which in turn points to a dataset object (in this case, a Table). Finally, the dataset (Table1) points to an actual database table (BIOLIFE), which is accessed through the BDE alias DBDEMOS. (BDE aliases are configured through the BDE Administrator.)



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

CHAPTER 2

2.DATABASE SYSTEM

2.1 INTRODUCTION TO DATABASE:

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

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

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

The term database refers to the collection of related records, and the software should be referred to as the database management system or DBMS. When the context is

unambiguous, however, many database administrators and programmers use the term database to cover both meanings.

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

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

2.2 DATABASE MODELS:

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

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

2.2.1 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.)

2.2.1.1 Relational operations:

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

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

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

There are a number of relational operations in addition to join. These include project (the process of eliminating some of the columns), restrict (the process of eliminating some of the rows), union (a way of combining two tables with similar structures), difference (which lists the rows in one table that are not found in the other), intersect (which lists the rows found in both tables), and product (mentioned above, which combines each row of one table with each row of the other). Depending on which other sources you consult, there are a number of other operators - many of which can be defined in terms of those listed above. These include semi-join, outer operators such as outer join and outer union, and various forms of division. Then there are operators to

rename columns, and summarizing or aggregating operators, and if you permit relation values as attributes (RVA - relation-valued attribute), then operators such as group and ungroup. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.

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

2.3 SQL IN DELPHI :

Using Structured Query Language in Delphi:

SQL:

SQL (Structured Query Language) is a standardized language for defining and manipulating data in a relational database. In accordance with the relational model of data, the database is perceived as a set of tables, relationships are represented by values in tables, and data is retrieved by specifying a result table that can be derived from one or more base tables.

Queries take the form of a command language that lets you *select*, *insert*, *update*, *find* out the location of data, and so forth.

In Delphi ... TQuery



If you are going to use SQL in your applications, you will become very familiar with the *TQuery* component. Delphi enables your applications to use SQL syntax directly through the *TQuery* component to access data from: Paradox and dBase tables (using local SQL - subset of ANSI standard SQL), Databases on the Local InterBase Server, and Databases on remote database servers.

Delphi also supports heterogeneous queries against more than one server or table type (for example, data from an Oracle table and a Paradox table).

TQuery has a property called *SQL*, which is used to store the SQL statement.

TQuery encapsulates one or more SQL statements, executes them and provides methods by which we can manipulate the results. Queries can be divided into two categories: those that produce result sets (such as a *SELECT* statement), and those that don't (such as an *UPDATE* or *INSERT* statement). Use *TQuery.Open* to execute a query that produces a result set; use *TQuery.ExecSQL* to execute queries that do not produce result sets.

The SQL statements can be either *static* or *dynamic*, that is, they can be set at design time or include parameters (*TQuery.Params*) that vary at run time. Using parameterized queries is very flexible, because you can change a user's view of and access to data on the fly at run time.

All executable SQL statements must be prepared before they can be executed. The result of preparation is the executable or operational form of the statement. The method of preparing an SQL statement and the persistence of its operational form distinguish static SQL from dynamic SQL. At design time a query is prepared and executed automatically when you set the query component's Active property to True. At run time, a query is prepared with a call to Prepare, and executed when the application calls the component's Open or ExecSQL methods.

A TQuery can return two kinds of result sets: "live" as with TTable component (users can edit data with data controls, and when a call to Post occurs changes are sent to database), "read only" for display purposes only. To request a live result set, set a query component's RequestLive property to True, and be aware that SQL statement must meet some specific requirements (no ORDER BY, SUM, AVG, etc.)

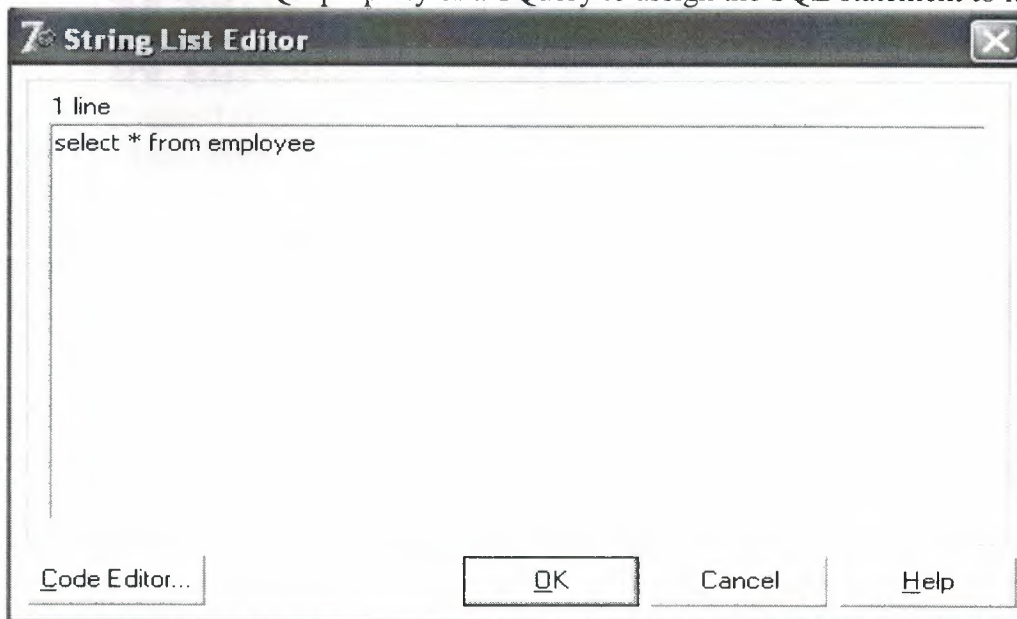
A query behaves in many ways very much like a table filter, and in some ways a query is even more powerful than a filter because it lets you access:

- more than one table at a time ("join" in SQL),
- a specified subset of rows and columns from its underlying table(s), rather than always returning all of them.

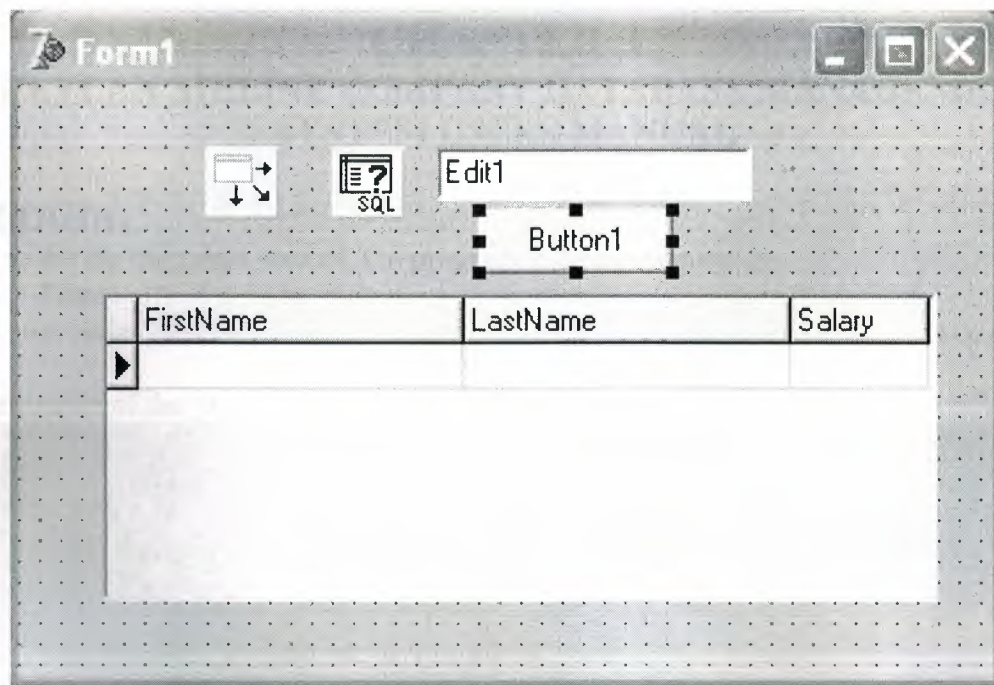
Simple example:

Now let's see some SQL in action. Although we could use the Database Form Wizard to create some SQL examples for this example we will do it manually, step by step:

1. Place a TQuery, TDataSource, TDBGrid, TEdit, and a TButton component on the main form.
2. Set TDataSource component's DataSet property to Query1.
3. Set TDBGrid component's DataSource property to DataSource1.
4. Set TQuery component's DatabaseName property to query
5. Double-click on SQL property of a TQuery to assign the SQL statement to it.



6. To make the grid display data at design time, change TQuery component's Active property to True.



As you can see, the grid displays data from Employee.db table in three columns (FirstName, LastName, Salary) even if Employee.db has 7 fields, and the result set is restricted to those records where the FirstName begins with 'R'.

7. Now assign the following code to the OnClick event of the Button1.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Query1.Close; {close the query}
  //assign new SQL expression
  Query1.SQL.Clear;
  Query1.SQL.Add ('Select EmpNo, FirstName, LastName');
  Query1.SQL.Add ('FROM employee.db');
  Query1.SQL.Add ('WHERE Salary > ' + Edit1.Text);
  Query1.RequestLive := true;
  Query1.Open; {open query + display data}
end;
```

8. Run your application. When you click on the Button (as long as Edit 1 has a valid currency value in it), the grid will display the EmpNo, FirstName and LastName fields for all records where Salary is greater than the specified currency value.

In this example we created simple static SQL statement with live result set (we haven't changed any of displayed records) just for displaying purposes.

CHAPTER 3

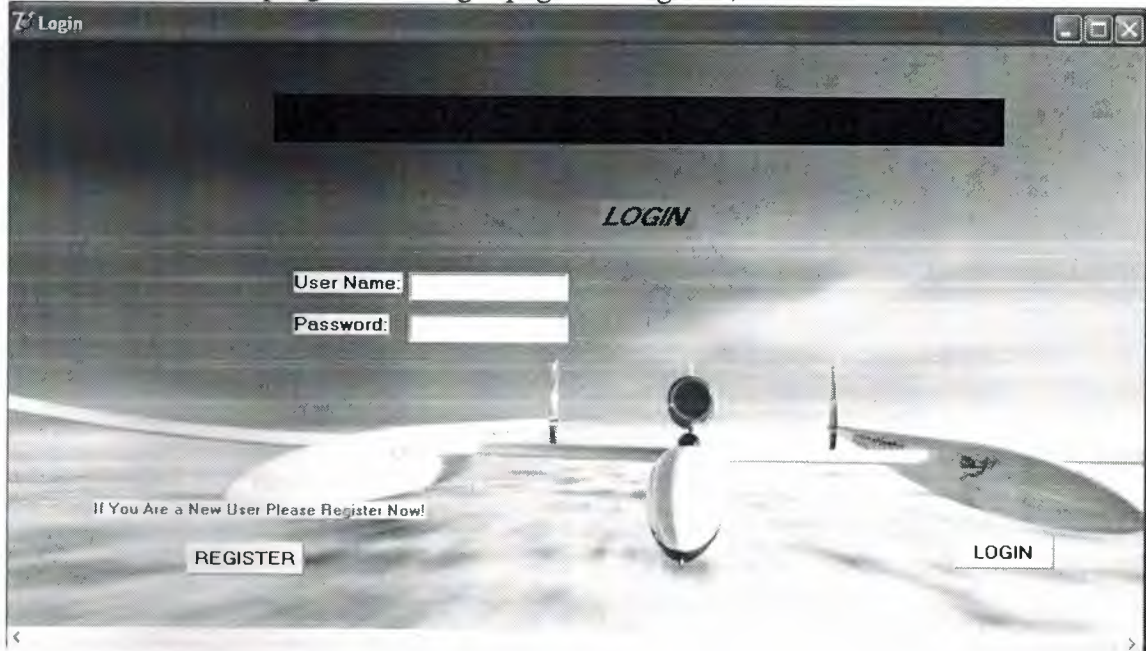
3.CEYLAN AIRLINE COMPANY DATABASE MANAGEMENT SYSTEM USERS MANUAL

3.1 LOGIN:

We can divide the target user of the program in two parts such as:

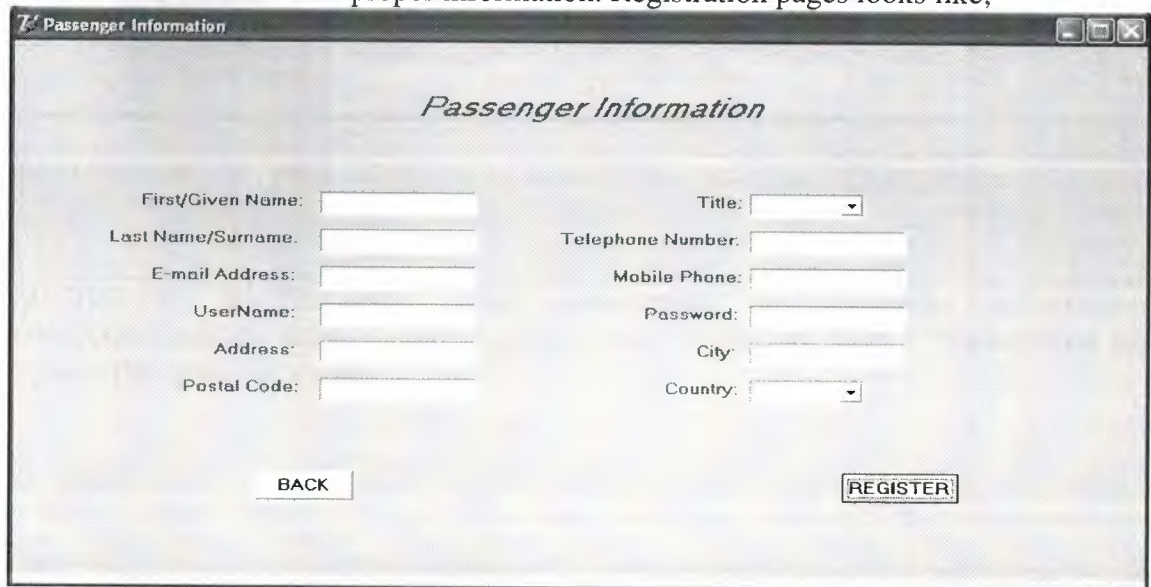
- User
- Admin

The entrance of the program is a login page looking like;



The screenshot shows a web browser window titled "Login". The background is a grayscale image of an airplane on a runway. The word "LOGIN" is centered in a stylized font. Below it, there are two input fields: "User Name:" and "Password:". Below these fields, there is a message: "If You Are a New User Please Register Now!". At the bottom, there are two buttons: "REGISTER" on the left and "LOGIN" on the right.

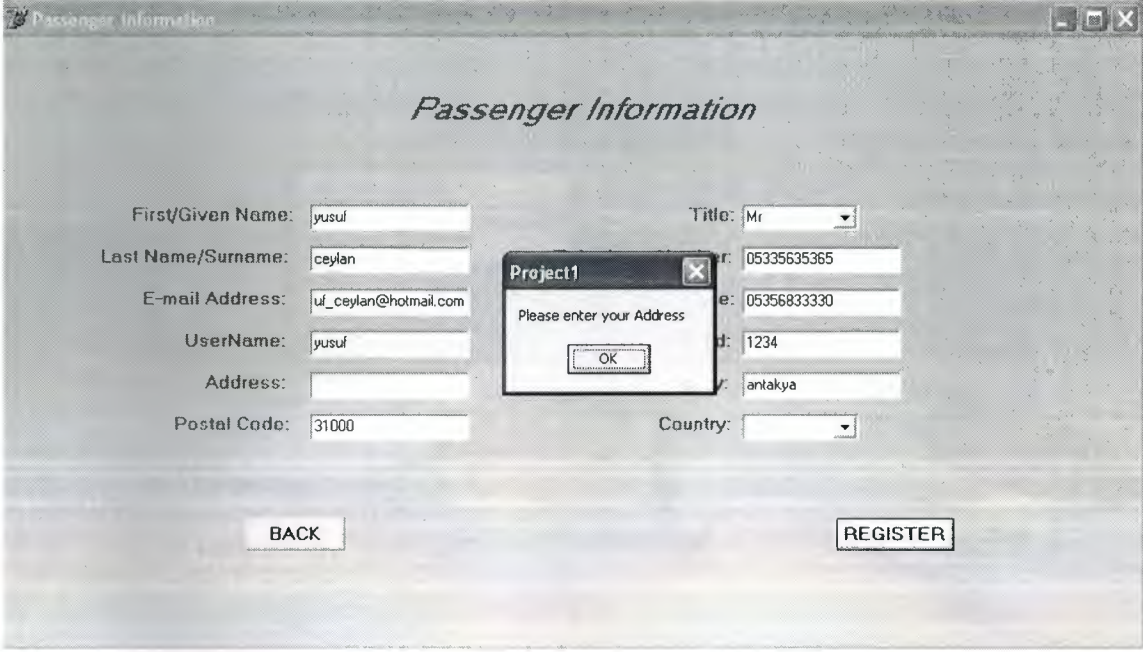
According to user name and password, users are redirected to the relevant pages from login page. If the user is not registered to the system yet s/he should be registered using the register button. When this button is clicked, the registration window comes up, user should fill the blanks with proper information. Registration pages looks like;



The screenshot shows a web browser window titled "Passenger Information". The background is a light gray. The title "Passenger Information" is centered in a stylized font. Below it, there are several input fields arranged in two columns. The left column contains: "First/Given Name:", "Last Name/Surname:", "E-mail Address:", "UserName:", "Address:", and "Postal Code:". The right column contains: "Title:" (with a dropdown arrow), "Telephone Number:", "Mobile Phone:", "Password:", "City:", and "Country:" (with a dropdown arrow). At the bottom, there are two buttons: "BACK" on the left and "REGISTER" on the right.

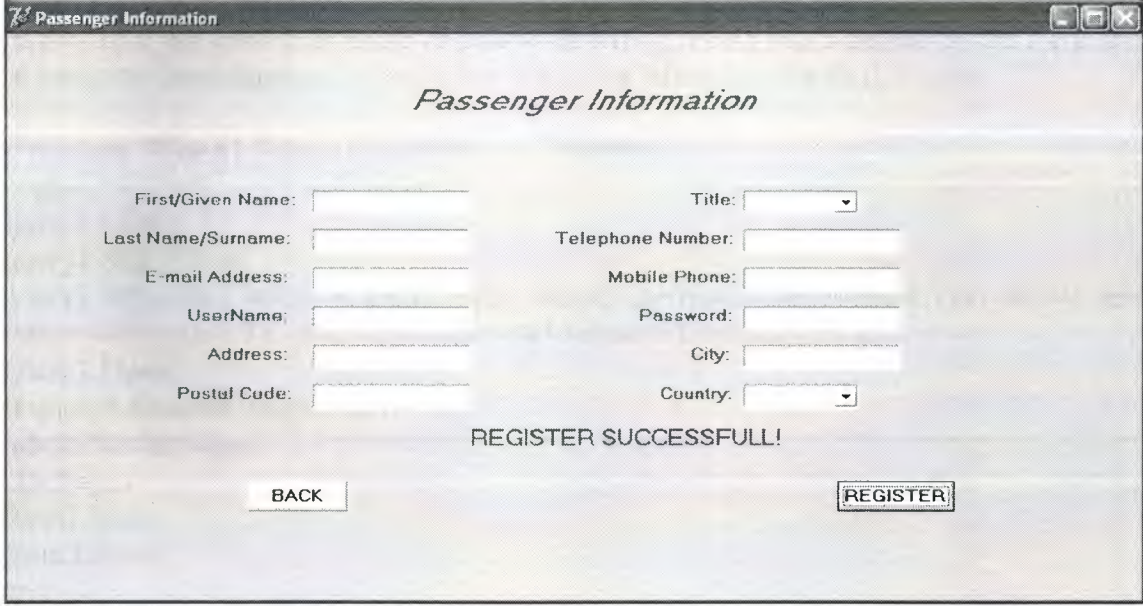
The user must complete this form for registration. The informations in the form must be true not to have a problem during the progress.

Then fill all passenger information. If there is a free area on register, a warning will be shown:



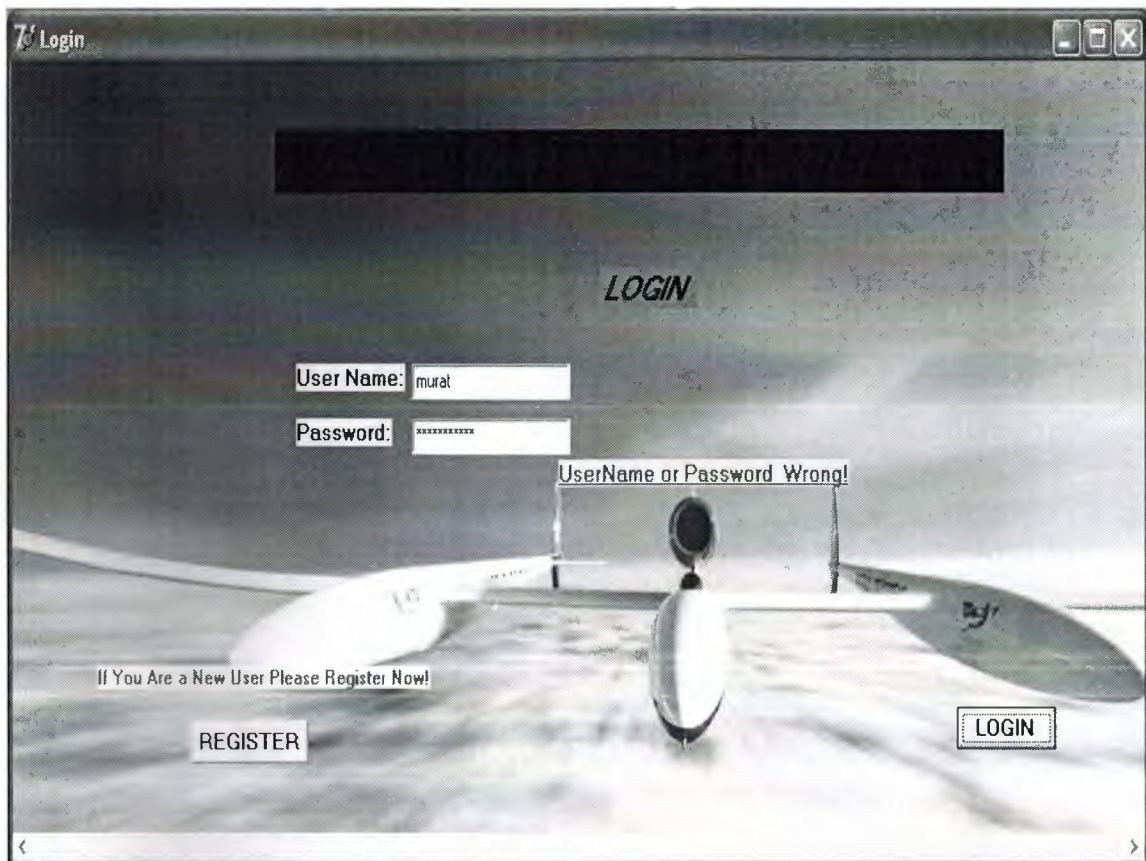
The screenshot shows a web application window titled "Passenger Information". The form contains the following fields: First/Given Name (yusuf), Last Name/Surname (ceylan), E-mail Address (uf_ceylan@hotmail.com), UserName (yusuf), Address (empty), Postal Code (31000), Title (Mr), Telephone Number (05335635365), Mobile Phone (05356833330), City (1234), and Country (antakya). A small dialog box titled "Project1" is overlaid on the form, displaying the message "Please enter your Address" and an "OK" button. At the bottom of the form, there are "BACK" and "REGISTER" buttons.

After clicking the "register" button a result page comes looking like;



The screenshot shows the same "Passenger Information" form, but now it displays the message "REGISTER SUCCESSFULL!" in the center. The form fields are empty, and the "BACK" and "REGISTER" buttons are still present at the bottom.

"REGISTER SUCCESSFUL" after completing the passenger information form. Afterwards, we return to the login page by clicking the "BACK" button. And we log into the system as a user.



When we return login form we try to log-in into the system with new username and password if we write user name or password wrong, seeing this warning on the form and to compare from database information to writing information with this code;

```

procedure TForm1.Button1Click(Sender: TObject);
begin
  query1.Close;
  query1.SQL.Clear;
  query1.SQL.Add('select * from login where username='+#39+edit1.Text+#39+' and
  Pass='+#39+edit2.Text+#39+'and type='+#39+label7.Caption+#39 );
  query1.Open;
  if query1.RecordCount=0 then
    label6.Visible:=true
  else begin
    form1.Hide;
    form3.show;
  end;
end;

```


3.2USER:

Departure	Destination	DepartureDate	ReturnDate	Class	Adult	Children	Infant	Round trip	One way	Round open
ADANA	ERCAN	23.06.2008	01.07.2008	ALL	1					

In this form, the user selects the departure and arrival airports by using the DBCombobox located at the top left of the form, then selects the flight dates using the datepicker application located at the upper part of the left page and the flexibility of flight days by clicking one of the radio buttons located at the lower part of middle of the page. Additionally, user should select the class (by default it is selected as 'Economy') and the number of passengers for each group by using the lists called adult, children and infant. The whole choices that we made above comes up at the lower part of the screen by using DBGrid application. And we can change these informations by using DBNavigator application. Then clicking the 'Continue' button, user can see the lists of the flights at the pre-determined dates. Page can be seen like:

Departure and Return Flight

CHOOSE DEPARTURE AND RETURN FLIGHT

Departure	Date	Time	Flight	Price+tax
<input checked="" type="radio"/> ADANA-ERCAN	23.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	25.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	27.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	29.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	01.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	03.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	05.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	07.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	09.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	11.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	13.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	15.07.2008	11:30	C301	70YTL

Return	Date	Time	Flight	Price+tax
<input type="radio"/> ERCAN-ADANA	23.06.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	25.06.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	27.06.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	29.06.2008	13:00	C301	70YTL
<input checked="" type="radio"/> ERCAN-ADANA	01.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	03.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	05.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	07.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	09.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	11.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	13.07.2008	13:00	C301	70YTL
<input type="radio"/> ERCAN-ADANA	15.07.2008	13:00	C301	70YTL

CONTINUE

At this page, user selects the most suitable actual date of any flight using radio group firstly if he selected round trip at the previous form(by default none). After selection, we automatically go the payment form clicking "Continue" button.

Flights

PLAN and BOOK YOUR FLIGHTS

DEPARTURE:
 ADULT(12+):
 CHILDREN(2-12):
 INFANT(7days-2):

DESTINATION:

DEPARTURE DATE:

RETURN DATE:

CLASS:

☒ ONE WAY
☐ ROUND TRIP
☐ ROUND TRIP(open)

Departure	Destination	DepartureDate	ReturnDate	Class	Adult	Children	Infant	Round trip	One way	Round open
ADANA	ERCAN	23.06.2008	01.07.2008	ALL	1					

Departure and Return Flight

CHOOSE DEPARTURE AND RETURN FLIGHT

Departure	Date	Time	Flight	Price+tax
<input type="radio"/> ADANA-ERCAN	23.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	25.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	27.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	29.06.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	01.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	03.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	05.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	07.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	09.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	11.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	13.07.2008	11:30	C301	70YTL
<input type="radio"/> ADANA-ERCAN	15.07.2008	11:30	C301	70YTL

Before payment form, we have one more issue about the selection. If we selected one-way trip from the flexibility of the flight dates in the flights form, our form would be such like above:

Form10

PLEASE FILL IN YOUR PAYMENT DETAILS

Last Name: Payment Method:

Card Number: Expiry Date:

Price+Tax:

The payment method has to be declared and informations about the credit card have to be entered correctly here by adding the credit card owner's last name. And we go to the report form automatically by clicking the "Show report" button. If we click back button, we return to the flights form.

Print Preview

CEYLAN AIRLINES

DEPARTURE	DESTINATION	DEPARTURE DATE	RETURN DATE	TIME	ADULT	CHILDREN	INFANT	PRICE(+Tax)
ADANA	ANTALYA	23.06.2008	01.07.2008	11:00	1			60YTL
ANTALYA	ADANA	01.07.2008	23.06.2008	11:00	1			60YTL

Page 1 of 1

This is the final report revealing the whole flight informations ready to be printed.

3.3ADMIN:

Admin can reach the related pages via the main page of the program seen below and s- he uses the login part by entering his-her username and password. Afterwards, admin menu page will be displayed and look like:

The screenshot shows a software window titled 'ADMIN' with a menu bar containing 'BOOK FLIGHT INFORMATION', 'PERSONEL INFORMATION', 'AIRPORT INFORMATION', and 'DEPARTURE AND RETURN INFORMATION'. The main area is titled 'PASSENGER INFORMATION' and contains a form with the following fields:

- First/Given Name:
- Title:
- Last Name/Surname:
- Telephone Number:
- E-mail Address:
- Mobile Phone:
- UserName:
- Password:
- Address:
- City:
- Postal Code:
- Country:
- Type:

Below the form is a 'BACK' button and a 'Search by Username' field. At the bottom, there is a data grid with the following columns: Username, Pass, First/GivenName, Surname, E-mail, Address, PostalCode, Title, and a checkbox column. The grid contains four rows of data:

Username	Pass	First/GivenName	Surname	E-mail	Address	PostalCode	Title	
ahmet	63486876878	ahmet	kel	ahmet.kel@hotmail.com	lefkosa	90000	mr	<input checked="" type="checkbox"/>
emin	6743576345	emin	tutkun	emin_tutkun@hotmail.com	bodrum	76576	mr	<input type="checkbox"/>
murat	1234	murat	everekli	murateverekli@hotmail.com	antalya	07000	mr	<input type="checkbox"/>
tahir	cicek	tahir	cicekli	tahir_cicekli@hotmail.com	malatya	44000	mr	<input type="checkbox"/>

In this form, we used main menu. The function of the main menu is to create the buttons which lead us from a form to another one. This page, admin can change the whole informations (ex: user can be made an admin by changing the type in the passenger information) and interact with the rest of the forms. The passenger informations in our database are listed by using DBGrid and are inserted or deleted by using DBNavigator. And all users in the database can be searched writing the codes below:

```
procedure TForm3.Edit1Change(Sender: TObject);
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from login where Username
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
```


Form6

FLIGHT INFORMATION

PLANE: ceylan1

DEPART AIRPORT: Ataturk ARRIVE AIRPORT: Ercan

DEPART GATE: yxcyx ARRIVE GATE: dasdard

DEPART TIME: 30.05.2008 ARRIVE TIME: 30.05.2008

SEATS AVAILABLE: dasdard SEATS BOOKED: dasdard

ACTUAL DEPART TIME: 30.05.2008 ACTUAL ARRIVE TIME: 15.05.2008

BACK

Plane	DepartAirport	DepartGate	DepartTime	Available	ActualDepart	ArriveAirport	ArriveGate	ArriveTime	Booked
* ceylan1	Ataturk	yxcyx	30.05.2008	dasdard	30.05.2008	Ercan	dasdard	30.05.2008	

We automatically reached this form clicking the flight information button on the admin page. The whole necessary informations about the flight are selected by the admin. After selection, the choices are listed via DBGrid and changed if desired via DBNavigator.

Employee Information

EMPLOYEE INFORMATION

NAME: yusuf SURNAME: ceylan

PASSWORD: 1234 SALARY: 2000YTL

ADDRESS: lefkosa/cyprus TELEPHONE NUMBER: 05338485288

POSTAL CODE: 90000

Search by Name

BACK

Name	Surname	Password	Department	Address	PostalCode	Teleph
yusuf	ceylan	1234	manager	lefkosa/cyprus	90000	053384
yagmur	gezer	1212	staff	antakya/hatay	31000	050676
emin	tuikun	3233232323	security	bodrum/mugla	53265	053376
tahir	cicekli	7623572536	employee	malatya	88900	053354

The employee information comes up just like how we meet the flight information form above. Clicking the employee information form on the admin page leads us directly to this form. The same procedure is applied here as well as the other forms which we go through the admin page. The necessary information about the staff is listed, we can search any staff using the name with applications that we discussed above. (DBGrid, DBNavigator)

Employee Information

EMPLOYEE INFORMATION

NAME: SURNAME:

PASSWORD: SALARY:

ADDRESS: TELEPHONE NUMBER:

POSTAL CODE:

Search by Name

Name	Surname	Password	Department	Address	PostalCode	Telephone
yusuf	ceylan	1234	manager	lefkosa/cyprus	90000	05338485288

The employee information form you see above just figures how the staff information is listed by making a search. when we type the first letter of the name in the search box, all names beginning with that letter are going to appear on the list.

Airport Information

AIRPORT INFORMATION

Airport Name:

Airport City:

Airport State:

Name	City	State

Airport information is again one of the pages seen via admin page. The airport name, the city in which it takes place and the state (if there is) can be listed by using the same method above.

Form11

DEPARTURE AND RETURN INFORMATION

FROM_ DATE.

TO_ TIME.

FLIGHT.

SEARCH BY NAME

From_	To_	Date	Time	Flight	P ^
ADANA	ANKARA	23.06.2008	10:30	C101	8
ADANA	ANKARA	25.06.2008	10:30	C101	8
ADANA	ANKARA	27.06.2008	10:30	C101	8
ADANA	ANKARA	29.06.2008	10:30	C101	8

The departure and return information form includes the necessary stuff about flight, we can search and find the flights using same methods. We additionally used date and time picker to determine the date and time for the departure or return. We can search the flights as well. How the search is made as follows:

Form11

DEPARTURE AND RETURN INFORMATION

FROM_ DATE.

TO_ TIME.

FLIGHT.

SEARCH BY NAME

From_	To_	Date	Time	Flight	P ^
ISTANBUL	ADANA	23.06.2008	14:00	C101	7
ISTANBUL	ADANA	25.06.2008	14:00	C101	7
ISTANBUL	ADANA	27.06.2008	14:00	C101	7
ISTANBUL	ADANA	29.06.2008	14:00	C101	7
ISTANBUL	ADANA	01.07.2008	14:00	C101	7

The search is made according to the first letter of the flight where it departs as you see in the figure. When we type the first letter of departure city, the flights from that city is going to be listed.

CONCLUSION

Airline Company Database Management System is a useful program for Airline Management. By using this program they can record and control register airline,passenger and customers.

The program is easy in use, and everything is in detail, I used borland Delphi 7 Programming Language in building it, also PARADOX 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 works in airlines registration,we can see about this. Also we can see all information about register airline company database management system...

REFERENCES

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

Program Code

unit Unit1;

interface

uses

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

type

TForm1 = class(TForm)

Image1: TImage;

Edit1: TEdit;

Edit2: TEdit;

Button1: TButton;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Button2: TButton;

Label5: TLabel;

DataSource1: TDataSource;

Label6: TLabel;

Label7: TLabel;

Query1: TQuery;

Label8: TLabel;

procedure Button2Click(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Edit2KeyPress(Sender: TObject; var Key: Char);

procedure Image1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4;

{ \$R *.dfm }

procedure TForm1.Button2Click(Sender: TObject);

begin

```
form2.show;  
form1.Hide;  
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
query1.Close;  
query1.SQL.Clear;  
query1.SQL.Add('select * from login where username='+#39+edit1.Text+#39+' and  
Pass='+#39+edit2.Text+#39+'and type='+#39+label7.Caption+#39 );  
query1.Open;  
if query1.RecordCount=0 then  
label6.Visible:=true  
else begin  
form1.Hide;  
form3.show;  
end;  
end;
```

```
query1.Close;  
query1.SQL.Clear;  
query1.SQL.Add('select * from login where username='+#39+edit1.Text+#39+' and  
Pass='+#39+edit2.Text+#39+'and type='+#39+label8.Caption+#39 );  
query1.Open;  
if query1.RecordCount=0 then  
label6.Visible:=true  
else begin  
form1.Hide;  
form4.Show;  
end;  
end;  
end;
```

```
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);  
begin  
if(key=#13)then button1.SetFocus;  
end;
```

```
procedure TForm1.Image1Click(Sender: TObject);
```

```
begin  
Query1.DatabaseName:='ceylan';  
Query1.requestlive:=true;  
query1.SQL.Text:='select * from login';  
query1.Active:=true;
```

```
end;
```

```
end.
```

unit Unit2;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, dblookup, StdCtrls, DB, DBTables, DBCtrls, Mask;

type

```
TForm2 = class(TForm)
  Button1: TButton;
  Button2: TButton;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  Label6: TLabel;
  Label7: TLabel;
  Label8: TLabel;
  Label9: TLabel;
  Label10: TLabel;
  Label11: TLabel;
  Label12: TLabel;
  Label13: TLabel;
  DBEdit1: TDBEdit;
  DBEdit2: TDBEdit;
  DBEdit3: TDBEdit;
  DBEdit4: TDBEdit;
  DBEdit5: TDBEdit;
  DBEdit6: TDBEdit;
  DBComboBox1: TDBComboBox;
  DBEdit7: TDBEdit;
  DBEdit8: TDBEdit;
  DBEdit9: TDBEdit;
  DBEdit10: TDBEdit;
  DBComboBox2: TDBComboBox;
  Query1: TQuery;
  DataSource1: TDataSource;
  Label14: TLabel;
  DBEdit11: TDBEdit;
  procedure Button2Click(Sender: TObject);
  procedure FormActivate(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```



```

var
  Form2: TForm2;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm2.Button2Click(Sender: TObject);
begin
  form1.show;
  form2.Hide;
end;

procedure TForm2.FormActivate(Sender: TObject);
begin
  query1.Insert;
  dbedit11.Text:='u';
end;

procedure TForm2.Button1Click(Sender: TObject);
begin
  if dbedit1.Text="" then
    showmessage('Please enter your name')
  else
    if dbedit2.Text="" then
      showmessage('please enter your surname')
    else
      if dbedit3.Text="" then
        showmessage('Please enter your E-mail Address')
      else
        if dbedit4.Text="" then
          showmessage('Please enter UserName ')
        else
          if dbedit5.Text="" then
            showmessage('Please enter your Address')
          else
            if dbedit6.Text="" then
              showmessage('Please enter your PostalCode')
            else
              if dbedit7.Text="" then
                showmessage('Please enter your TelephoneNumber')
              else
                if dbedit8.Text="" then
                  showmessage('Please enter your MobilePhone ')
                else
                  if dbedit9.Text="" then
                    showmessage('Please enter your Password')

```

```

else
if dbedit10.Text="" then
showmessage('Please enter your City')
else
begin
query1.Post;
dbedit1.Text:="";
dbedit2.Text:="";
dbedit3.Text:="";
dbedit4.Text:="";
dbedit5.Text:="";
dbedit6.Text:="";
dbedit7.Text:="";
dbedit8.Text:="";
dbedit9.Text:="";
dbedit10.Text:="";
dbcombobox1.Text:="";
dbcombobox2.Text:="";
label14.Caption:='REGISTER SUCCESSFUL!';
label14.Font.Size:=14;
label14.Font.Color:=clred;
label14.Visible:=true;
end;
end;
procedure TForm2.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='ceylan';
Query1.requestlive:=true;
query1.SQL.Text:='select * from login';
query1.Active:=true;
end;

end.

```

unit Unit3;

interface

uses

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

type

TForm3 = class(TForm)
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;

```

Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBComboBox1: TDBComboBox;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBComboBox2: TDBComboBox;
DBGrid1: TDBGrid;
DBNavigator1: TDBNavigator;
Edit1: TEdit;
DBComboBox3: TDBComboBox;
Label1: TLabel;
Button2: TButton;
MainMenu1: TMainMenu;
BOOKFLIGHT1: TMenuItem;
PERSONELINFORMATION1: TMenuItem;
AIRPORT1: TMenuItem;
PLANE1: TMenuItem;
Label14: TLabel;
Query1: TQuery;
DataSource1: TDataSource;
Label15: TLabel;
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
procedure Button2Click(Sender: TObject);
procedure BOOKFLIGHT1Click(Sender: TObject);
procedure PERSONELINFORMATION1Click(Sender: TObject);
procedure AIRPORT1Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure PLANE1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

```

var


```

Form3: TForm3;

implementation

uses Unit1, Unit6, Unit7, Unit8, Unit11;

{$R *.dfm}

procedure TForm3.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  dbedit1.Clear;
  dbedit2.Clear;
  dbedit3.Clear;
  dbedit4.Clear;
  dbedit5.Clear;
  dbedit6.Clear;
  dbedit7.Clear;
  dbedit8.Clear;
  dbedit9.Clear;
  dbedit10.Clear;
  dbcombobox1.Clear;
  dbcombobox2.Clear;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  form1.show;
  form3.Hide;
end;

procedure TForm3.BOOKFLIGHT1Click(Sender: TObject);
begin
  form6.show;
  form3.Hide;
end;

procedure TForm3.PERSONELINFORMATION1Click(Sender: TObject);
begin
  form7.show;
  form3.Hide;
end;

procedure TForm3.AIRPORT1Click(Sender: TObject);
begin
  form8.show;
  form3.Hide;
end;

```



```
procedure TForm3.Edit1Change(Sender: TObject);
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from login where Username
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
```

```
procedure TForm3.PLANE1Click(Sender: TObject);
begin
form11.show;
form3.Hide;
end;
```

```
procedure TForm3.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='ceylan';
Query1.requestlive:=true;
query1.SQL.Text:='select * from login';
query1.Active:=true;
edit1.Text:='';
```

```
end;
```

```
end.
```

unit Unit4;

```
interface
```

```
uses
```

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

```
type
```

```
TForm4 = class(TForm)
  Button1: TButton;
  DBComboBox1: TDBComboBox;
  DBComboBox2: TDBComboBox;
  DBComboBox3: TDBComboBox;
  DBComboBox4: TDBComboBox;
  DBComboBox5: TDBComboBox;
  DBComboBox6: TDBComboBox;
  Label5: TLabel;
  Label6: TLabel;
  Label9: TLabel;
  Label10: TLabel;
```

```

Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label1: TLabel;
Label2: TLabel;
DBGrid1: TDBGrid;
DataSource1: TDataSource;
Query1: TQuery;
DBNavigator1: TDBNavigator;
Query2: TQuery;
RadioGroup1: TRadioGroup;
Query3: TQuery;
DateTimePicker1: TDateTimePicker;
DateTimePicker2: TDateTimePicker;
Button2: TButton;
procedure Button1Click(Sender: TObject);
procedure DateTimePicker1Change(Sender: TObject);
procedure DateTimePicker2Change(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form4: TForm4;

implementation

uses Unit5, Unit9, Unit1;

{$R *.dfm}

procedure TForm4.Button1Click(Sender: TObject);

begin
    query1.Edit;
    if radiogroup1.ItemIndex=0 then begin
        dbgrid1.Fields[8].AsString:='T';
        dbgrid1.Fields[9].AsString:='F';
        dbgrid1.Fields[10].AsString:='F';
    end;
    if radiogroup1.ItemIndex=1 then begin
        dbgrid1.Fields[8].AsString:='F';
        dbgrid1.Fields[9].AsString:='T';
        dbgrid1.Fields[10].AsString:='F';
    end;
end;

```



```

if radiogroup1.ItemIndex=2 then begin
dbgrid1.Fields[8].AsString:='F';
dbgrid1.Fields[9].AsString:='F';
dbgrid1.Fields[10].AsString:='T';
end;

```

```

if radiogroup1.ItemIndex=0 then begin
query2.Close;
query2.SQL.Clear;
query2.SQL.Add('select * from details where From_='+#39+dbcombobox1.Text+#39+'
and To_='+#39+dbcombobox2.Text+#39);
query2.Open;
query2.First;
form9.radiogroup1.Items.Clear;
while not query2.Eof do begin
form9.RadioGroup1.Items.Add(query2.Fields[0].AsString+'-
'+query2.Fields[1].AsString+' '+query2.Fields[2].AsString+'
'+query2.Fields[3].AsString+' '+query2.Fields[4].AsString+'
'+query2.Fields[5].AsString);
query2.Next;
end;
form9.RadioGroup2.Visible:=false;
form5.SummaryBand1.Visible:=false;
end;

```

```

if radiogroup1.ItemIndex=1 then begin
query2.Close;
query2.SQL.Clear;
query2.SQL.Add('select * from details where From_='+#39+dbcombobox1.Text+#39+'
and To_='+#39+dbcombobox2.Text+#39);
query2.Open;
query2.First;
form9.RadioGroup1.Items.Clear;
while not query2.Eof do begin
form9.RadioGroup1.Items.Add(query2.Fields[0].AsString+'-
'+query2.Fields[1].AsString+' '+query2.Fields[2].AsString+'
'+query2.Fields[3].AsString+' '+query2.Fields[4].AsString+'
'+query2.Fields[5].AsString);
query2.Next;
end;
query3.Close;
query3.SQL.Clear;
query3.SQL.Add('select * from details where From_='+#39+dbcombobox2.Text+#39+'
and To_='+#39+dbcombobox1.Text+#39 );
query3.Open;
query3.First;
form9.RadioGroup2.Items.Clear;
while not query3.Eof do begin

```

```

form9.RadioGroup2.Items.Add(query3.Fields[0].AsString+'-
'+query3.Fields[1].AsString+' '+query3.Fields[2].AsString+'
'+query3.Fields[3].AsString+' '+query2.Fields[4].AsString+'
'+query2.Fields[5].AsString);
query3.Next;
end;
end;

form9.show;
form4.Hide;

end;

procedure TForm4.DateTimePicker1Change(Sender: TObject);
begin
query1.Edit;
query1.Fields[2].AsString:=datetostr(datetimepicker1.Date);
end;

procedure TForm4.DateTimePicker2Change(Sender: TObject);
begin
query1.Edit;
query1.Fields[3].AsString:=datetostr(datetimepicker2.Date);
end;

procedure TForm4.Button2Click(Sender: TObject);
begin
form1.show;
form4.Hide;
end;

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

end;

end.

```

unit Unit5;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls, QuickRpt, QRCtrls, DB, DBTables, jpeg;

type

```
TForm5 = class(TForm)
  Button1: TButton;
  Button2: TButton;
  QuickRep1: TQuickRep;
  Table1: TTable;
  DataSource1: TDataSource;
  ColumnHeaderBand1: TQRBand;
  DetailBand1: TQRBand;
  PageFooterBand1: TQRBand;
  PageHeaderBand1: TQRBand;
  QRLabel1: TQRLabel;
  QRLabel2: TQRLabel;
  QRLabel3: TQRLabel;
  QRLabel4: TQRLabel;
  QRLabel5: TQRLabel;
  QRLabel6: TQRLabel;
  QRLabel7: TQRLabel;
  QRLabel8: TQRLabel;
  QRDBText1: TQRDBText;
  QRDBText2: TQRDBText;
  QRDBText3: TQRDBText;
  QRDBText4: TQRDBText;
  QRDBText5: TQRDBText;
  QRDBText6: TQRDBText;
  QRDBText7: TQRDBText;
  QRDBText8: TQRDBText;
  QRLabel9: TQRLabel;
  QRDBText9: TQRDBText;
  QRLabel10: TQRLabel;
  SummaryBand1: TQRBand;
  QRDBText10: TQRDBText;
  QRDBText11: TQRDBText;
  QRDBText12: TQRDBText;
  QRDBText13: TQRDBText;
  QRDBText14: TQRDBText;
  QRDBText15: TQRDBText;
  QRDBText16: TQRDBText;
  QRDBText17: TQRDBText;
  QRDBText18: TQRDBText;
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
```



```

private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form5: TForm5;

implementation

uses Unit4;

{$R *.dfm}

procedure TForm5.Button1Click(Sender: TObject);
begin
  form4.show;
  form5.Hide;
end;

procedure TForm5.Button2Click(Sender: TObject);
begin
  quickrep1.Preview;
  if form4.RadioGroup1.ItemIndex=0 then
  begin
    qrdbtext10.Visible:=false;
  end;
end;

end;
end.

```

unit Unit6;

interface

uses

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

type

```

TForm6 = class(TForm)
  Label1: TLabel;
  DataSource1: TDataSource;
  DBNavigator1: TDBNavigator;
  DBEdit1: TDBEdit;
  DBEdit3: TDBEdit;
  DBEdit5: TDBEdit;

```

```

DBEdit7: TDBEdit;
DBComboBox1: TDBComboBox;
DBComboBox2: TDBComboBox;
DBComboBox3: TDBComboBox;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Button1: TButton;
DateTimePicker1: TDateTimePicker;
DateTimePicker2: TDateTimePicker;
DateTimePicker3: TDateTimePicker;
DateTimePicker4: TDateTimePicker;
Query1: TQuery;
DBGrid1: TDBGrid;
Table1: TTable;
procedure Button1Click(Sender: TObject);
procedure DateTimePicker1Change(Sender: TObject);
procedure DateTimePicker2Change(Sender: TObject);
procedure DateTimePicker3Change(Sender: TObject);
procedure DateTimePicker4Change(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure DBNavigator1Click(Sender: TObject; Button: TNavigateBtn);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form6: TForm6;

implementation

uses Unit3;

{$R *.dfm}

procedure TForm6.Button1Click(Sender: TObject);
begin
  form3.show;
  form6.Hide;
end;

```

```

procedure TForm6.DateTimePicker1Change(Sender: TObject);
begin
table1.Edit;
table1.Fields[3].AsString:=datetostr(datetimepicker1.DateTime)
end;

```

```

procedure TForm6.DateTimePicker2Change(Sender: TObject);
begin
table1.Edit;
table1.Fields[8].AsString:=datetostr(datetimepicker2.Date)
end;

```

```

procedure TForm6.DateTimePicker3Change(Sender: TObject);
begin
table1.Edit;
table1.Fields[5].AsString:=datetostr(datetimepicker3.Date)
end;

```

```

procedure TForm6.DateTimePicker4Change(Sender: TObject);
begin
table1.Edit;
table1.Fields[10].AsString:=datetostr(datetimepicker4.Date)
end;

```

```

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

```

```

procedure TForm6.DBNavigator1Click(Sender: TObject; Button: TNavigateBtn);
begin
if dbcombobox1.Text="" then
showmessage('Please enter plane')
else
if dbcombobox2.Text="" then
showmessage('please enter DepartAirport')
else
if dbedit1.Text="" then
showmessage('Please enter DepartGate')
else
if dbedit3.Text="" then
showmessage('Please enter SeatsAvailable ')
else
if dbcombobox3.Text="" then
showmessage('Please enter ArriveAirport')
else

```



```

if dbedit5.Text="" then
showmessage('Please enter ArriveGate')
else
if dbedit7.Text="" then
showmessage('Please enter SeatsBooked')
end;

```

```

end.

```

unit Unit7;

```

interface

```

```

uses

```

```

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

```

```

type

```

```

    TForm7 = class(TForm)
        Label1: TLabel;
        DataSource1: TDataSource;
        DBEdit1: TDBEdit;
        DBEdit2: TDBEdit;
        DBEdit3: TDBEdit;
        DBEdit4: TDBEdit;
        DBEdit5: TDBEdit;
        DBEdit6: TDBEdit;
        DBEdit7: TDBEdit;
        Button1: TButton;
        DBGrid1: TDBGrid;
        DBNavigator1: TDBNavigator;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        Label7: TLabel;
        Label8: TLabel;
        Edit1: TEdit;
        Label9: TLabel;
        Query1: TQuery;
        DBEdit8: TDBEdit;
        Label10: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure Edit1Change(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        procedure DBNavigator1Click(Sender: TObject; Button: TNavigateBtn);
    private
        { Private declarations }
    public

```

```

    { Public declarations }
end;

var
    Form7: TForm7;

implementation

uses Unit3;

{$R *.dfm}

procedure TForm7.Button1Click(Sender: TObject);
begin
    form3.show;
    form7.Hide;
end;

procedure TForm7.Edit1Change(Sender: TObject);
begin
    query1.Close;
    query1.SQL.Clear;
    query1.SQL.Add('select * from employee where Name
    like'+#39+(edit1.Text)+'%'+#39);
    query1.Open;
end;

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

end;

procedure TForm7.DBNavigator1Click(Sender: TObject; Button: TNavigateBtn);
begin
    if dbedit1.Text="" then
        showmessage('Please enter Name')
    else
        if dbedit2.Text="" then
            showmessage('please enter Surname')
        else
            if dbedit3.Text="" then
                showmessage('Please enter Password')
            else
                if dbedit4.Text="" then
                    showmessage('Please enter Address')

```

```

else
if dbedit5.Text="" then
showmessage('Please enter Postal Code')
else
if dbedit6.Text="" then
showmessage('Please enter Salary')
else
if dbedit7.Text="" then
showmessage('Please enter TelephoneNumber')
else
if dbedit8.Text="" then
showmessage('Please enter Department ')

end;

end.

```

unit Unit8;

```

interface

```

```

uses

```

```

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

```

```

type

```

```

  TForm8 = class(TForm)
    DataSource1: TDataSource;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    DBNavigator1: TDBNavigator;
    DBGrid1: TDBGrid;
    Button1: TButton;
    Query1: TQuery;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

```

```

var

```

```

  Form8: TForm8;

```


implementation

uses Unit3;

{ \$R *.dfm }

```
procedure TForm8.Button1Click(Sender: TObject);
begin
  form3.show;
  form8.Hide;
end;
```

```
procedure TForm8.FormCreate(Sender: TObject);
begin
  Query1.DatabaseName:='ceylan';
  Query1.requestlive:=true;
  query1.SQL.Text:='select * from airport';
  query1.Active:=true;
```

end;

end.

unit Unit9;

interface

uses

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

type

```
TForm9 = class(TForm)
  Label1: TLabel;
  Button1: TButton;
  Query1: TQuery;
  DataSource1: TDataSource;
  RadioGroup1: TRadioGroup;
  RadioGroup2: TRadioGroup;
  procedure Button1Click(Sender: TObject);
```

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form9: TForm9;

implementation

uses Unit4, Unit5, Unit10;

{ \$R *.dfm }

procedure TForm9.Button1Click(Sender: TObject);

begin

form10.show;

form9.Hide;

end;

end.

unit Unit10;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, DB, DBTables, ComCtrls, DBCtrls, Mask;

type

TForm10 = class(TForm)

DBEdit1: TDBEdit;

DBEdit2: TDBEdit;

DBComboBox1: TDBComboBox;

DateTimePicker1: TDateTimePicker;

DataSource1: TDataSource;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Button1: TButton;

Button2: TButton;

DBEdit3: TDBEdit;

Label6: TLabel;

Query1: TQuery;

procedure DateTimePicker1Change(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

```

    end;

var
    Form10: TForm10;

implementation

uses Unit5, Unit4;

{$R *.dfm}

procedure TForm10.DateTimePicker1Change(Sender: TObject);
begin
    query1.Edit;
    query1.Fields[3].AsString:=DateToStr(datetimepicker1.DateTime);
end;

procedure TForm10.Button2Click(Sender: TObject);
begin
    form5.show;
    form10.Hide;
    query1.Close;
    close;
end;

procedure TForm10.Button1Click(Sender: TObject);
begin
    form4.show;
    form10.Hide;
end;

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

end;

end.

```

unit Unit11;

```
interface
```

```
uses
```

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



```

ComCtrls;

type
  TForm11 = class(TForm)
    DBComboBox1: TDBComboBox;
    DBComboBox2: TDBComboBox;
    DBComboBox3: TDBComboBox;
    DBComboBox4: TDBComboBox;
    Label1: TLabel;
    Label2: TLabel;
    DateTimePicker1: TDateTimePicker;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    DBNavigator1: TDBNavigator;
    Label7: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    procedure Edit1Change(Sender: TObject);
    procedure DateTimePicker1Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form11: TForm11;

implementation

uses Unit3;

{$R *.dfm}

procedure TForm11.Edit1Change(Sender: TObject);
begin
  query1.Close;
  query1.SQL.Clear;
  query1.SQL.Add('select * from details where From_ like'+#39+(edit1.Text)+'%'+#39);
  query1.Open;
end;

procedure TForm11.DateTimePicker1Change(Sender: TObject);

```

```
begin
query1.Edit;
query1.Fields[2].AsString:=datetostr(datetimestr(Date));
end;
```

```
procedure TForm11.FormCreate(Sender: TObject);
begin
Query1.DatabaseName:='ceylan';
Query1.requestlive:=true;
query1.SQL.Text:='select * from details';
query1.Active:=true;
edit1.Text:='';
end;
```

```
procedure TForm11.Button1Click(Sender: TObject);
begin
form3.show;
form11.Hide;
end;

end.
```