

NEAR EAST UNIVERSITY



Faculty Of Engineering

Department Of Computer Engineering

**STOCK AND REGISTER PROGRAM FOR
OPTICIAN BY USING DELPHI PROGRAMMING**

**Graduation Project
COM 400**

Student: Enis KASIMOĞLU

Supervisor : Asst.Prof.Dr Elbrus IMANOV

Nicosia - 2007

ACKNOWLEDGEMENTS

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

I am greatly indepted to my family for their endless support from my starting day in my educational life until today. I will never forget the things that my father Mr.Celal Kasımoğlu did for me during my educational life, also I want to say thanks to my mother Mrs. Nazife Kasımoğlu. I dedicate my project to them.

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

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

ABSTRACT

The aim of this project is to develop optician tracking program that contain registration, all applications and also account 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 program is designed for login . Which are login must user name and password. An individual who is working in any of these predefined type can login to the program by using a predefined password. After logging there will be a form. Which is belongs to authorised person. The authority of the user to reach, do changes and update the information in this program is limited with respect to the possition according to letting users. These are simply expressing how the program was designed to use in proper and secure way.

To show results show the efficiency of the program of optician tracking system in program of the using in other chapters

TABLE OF CONTENTS

ACKNOWLEDGMENT	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	v
CHAPTER ONE: DELPHI	1
1.1.Introduction to Delphi	1
1.2 What is Delphi?	1
1.2.1 What kind of programming can you do with Delphi	2
1.2.2 Versions are there and How do they differs?	3
1.3 The VCL to Applications Developers	5
1.3.1 The VCL Component Writers	6
1.3.2 The VCL is made up of components	6
1.3.3 Component Types, Sturcture, and VCL Hierarchy	7
1.3.4 Component Types	7
1.3.4.1 Standart Component	7
1.3.4.2 Custom Component	8
1.3.4.3 Graphical Component	9
1.3.4.4 Non-Visual Component	9
1.3.4.5 Structure of Component	9
1.3.5 Component Properties	10
1.3.6 Properties provide access to internal storage fields	10
1.4 Property-Access methods	11
1.5 Types of Properties	12
1.6 Methods	12
1.7 Events	12
1.8 Containership	15
1.9 Ownership	15
1.10 Parenthood	16
CHAPTER TWO: DATABASE	17
2.1 Introduction Database	17
2.2 History	18
2.3 Database Models	20
2.3.1 Flat Model	20
2.3.2 Hierarchical Model	20
2.3.3 Network Model	21
2.3.4 Relational Model	21
2.3.4.1 Relational Operations	23
2.3.5 Dimensional Model	24
2.3.6 Object Database Model	24
2.4 Database Internals	25
2.4.1 Indexing	25
2.4.2 Transactions and Concurrency	26
2.4.3 Replication	27
2.5 Applications of Databases	27

2.6 Database Brands	28
CHAPTER THREE: SQL	29
3.1 SQL	29
3.2 History	29
3.2.1 Standardization	30
3.3 Scope	31
3.4 SQL Keywords	32
3.4.1 Data Retrieval	32
3.4.2 Data Manipulation	33
3.4.3 Transaction Control	34
3.4.4 Data Definition	34
3.4.5 Data Control	35
3.4.6 Other	35
3.5 Criticisms of SQL	36
3.6 Logical Operators	37
CHAPTER FOUR: DESCRIPTION ABOUT PROJECT	38
CONCLUSION	49
APPENDIX1 : Program Code	50
APPENDIX2: Table of Database	112
REFERENCES	116

INTRODUCTION

This project is register and stock program for optician which uses SQL queries. This program was prepared by using Borland Delphi 7 and SQL (Structured Query Language).

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

CHAPTER1: About the Delphi general information.

CHAPTER2: About the database general information, database models, relational operations, and database brands.

CHAPTER3: About SQL. It is history , keywords and some of the commands of it.

CHAPTER4: About the project , how we create it, its forms and using the program.

CHAPTER 1

DELPHI

1.1 INTRODUCTION TO DELPHI

In this project we will answer some basic questions about Delphi, to give a feel for where it came from, what it has to offer, and where it is going in the future. This is an essential part of any course. We feel it is important for those studying a new programming language to understand the ideology and intended use of the language. Too many programmers are tempted to use the language that they know, rather than learn a new one to cope with the specific demands of the project that they have at the end of this lecture, we should have gained sufficient understanding of the Delphi ideology to decide if it is a suitable language for a specific project that we have.

1.2 WHAT IS DELPHI?

Delphi is an object oriented, component based, visual, rapid development environment for event driven Windows applications, based on the Pascal language. Unlike other popular competing Rapid Application Development (RAD) tools, Delphi compiles the code you write and produces really tight, natively executable code for the target platform. In fact the most recent versions of Delphi optimise the compiled code and the resulting executables are as efficient as those compiled with any other compiler currently on the market. The term "visual" describes Delphi very well. All of the user interface development is conducted in a What You See Is What You Get environment (WYSIWYG), which means you can create polished, user friendly interfaces in a very short time, or prototype whole applications in a few hours.

Delphi is, in effect, the latest in a long and distinguished line of Pascal compilers (the previous versions of which went by the name "Turbo Pascal") from the company formerly known as Borland, now known as Inprise. In common with the Turbo Pascal compilers that preceded it, Delphi is not just a compiler, but a complete development environment. 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

The development environment itself is extensible, and there are a number of add ins available to perform functions such as memory leak detection and profiling. In short, Delphi includes just about everything you need to write applications that will run on an Intel platform under Windows, but if your target platform is a Silicon Graphics running IRIX, or a Sun Sparc running SOLARIS, or even a PC running LINUX, then you will need to look elsewhere for your development tool.

This specialisation on one platform and one operating system, makes Delphi a very strong tool. The code it generates runs very rapidly, and is very stable, once your own bugs have been ironed out.

1.2.1 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.

Some projects to which Delphi is suited:

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

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

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

1.2.2 Versions are there and How do they differ?

Borland (as they were then) has a long tradition in the creation of high speed compilers. One of their best known products was Turbo Pascal - a tool that many programmers cut their teeth on. With the rise in importance of the Windows environment, it was only a matter of time before development tools started to appear

that were specific to this new environment. In the very beginning, Windows produced SDKs (software development kits) that were totally non-visual (user interface development was totally separated from the development of the actual application), and required great patience and some genius to get anything working with. Whilst these tools slowly improved, they still required a really good understanding of the inner workings of Windows. To a great extent these criticisms were dispatched by the release of Microsoft's Visual Basic product, which attempted to bring Windows development to the masses. It achieved this to a great extent too, and remains a popular product today. However, it suffered from several drawbacks:

- 1) It wasn't as stable as it might have been
- 2) It was an interpreted language and hence was slow to run
- 3) It had as its underlying language BASIC, and most "real" programmers weren't so keen!

Into this environment arrived the eye opening Delphi I product, and in many ways the standard for visual development tools for Windows was set. This first version was a 16 bit compiler, and produced executable code that would run on Windows 3.1 and Windows 3.11. Of course, Microsoft have ensured (up to now) that their 32 bit operating systems (Win95, Win98, and Win NT) will all run 16 bit applications, however, many of the features that were introduced in these newer operating systems are not accessible to the 16 bit applications developed with Delphi I. Delphi 2 was released quite soon after Delphi I, and in fact included a full distribution of Delphi I on the same CD. Delphi 2, (and all subsequent versions) have been 32 bit compilers, producing code that runs exclusively on 32bit Windows platforms. (We ignore for simplicity the WIN32S DLLs which allow Win 3.1x to run some 32 bit applications).

Delphi is currently standing at Version 4.0, with a new release (version 5.0) expected shortly. In its latest version, Delphi has become somewhat feature loaded, and as a result, we would argue, less stable than the earlier versions. However, in its defence, Delphi (and Borland products in general) have always been more stable than their competitors products, and the majority of Delphi 4's glitches are minor and

forgivable - just don't try and copy/paste a selection of your code, midway through a debugging session!

The reasons for the version progression include the addition of new components, improvements in the development environment, the inclusion of more internet related support and improvements in the documentation. Delphi at version 4 is a very mature product, and Inprise has always been responsive in developing the product in the direction that the market requires it to go. Predominantly this means right now, the inclusion of more and more Internet, Web and CORBA related tools and components - a trend we are assured continues with the release of version 5.0

For each version of Delphi there are several sub-versions, varying in cost and features, from the most basic "Developer" version to the most complete (and expensive) "Client Server" version. The variation in price is substantial, and if you are contemplating a purchase, you should study the feature list carefully to ensure you are not paying for features you will never use. Even the most basic "Developer" version contains the vast majority of the features you are likely to need on a day to day basis. Don't assume that you will need Client Server, simply because you are intending to write a large database application - The developer edition is quite capable of this.

1.3 The VCL to Applications Developers

Applications Developers create complete applications by interacting with the Delphi visual environment (as mentioned earlier, this is a concept nonexistent in many other frameworks). These people use the VCL to create their user-interface and the other elements of their application: database connectivity, data validation, business rules, etc..

Applications Developers should know which properties, events, and methods each component makes available. Additionally, by understanding the VCL architecture, Applications Developers will be able to easily identify where they can improve their applications by extending components or creating new ones. Then they can maximize the capabilities of these components, and create better applications.

1.3.1 The VCL to Component Writers

Component Writers expand on the existing VCL, either by developing new components, or by increasing the functionality of existing ones. Many component writers make their components available for Applications Developers to use.

A Component Writer must take their knowledge of the VCL a step further than that of the Application Developer. For example, they must know whether to write a new component or to extend an existing one when the need for a certain characteristic arises. This requires a greater knowledge of the VCL's inner workings.

1.3.2 The VCL is made up of components

Components are the building blocks that developers use to design the user-interface and to provide some non-visual capabilities to their applications. To an Application Developer, a component is an object most commonly dragged from the Component palette and placed onto a form. Once on the form, one can manipulate the component's properties and add code to the component's various events to give the component a specific behavior. To a Component Writer, components are objects in Object Pascal code. Some components encapsulate the behavior of elements provided by the system, such as the standard Windows 95 controls. Other objects introduce entirely new visual or non-visual elements, in which case the component's code makes up the entire behavior of the component.

The complexity of different components varies widely. Some might be simple while others might encapsulate an elaborate task. There is no limit to what a component can do or be made up of. You can have a very simple component like a **TLabel**, or a much more complex component which encapsulates the complete functionality of a spreadsheet.

1.3.3 Component types, structure, and VCL hierarchy

Components are really just special types of objects. In fact, a component's structure is based on the rules that apply to Object Pascal. There are three fundamental keys to understanding the VCL.

First, you should know the special characteristics of the four basic component types: standard controls, custom controls, graphical controls and non-visual components.

Second, you must understand the VCL structure with which components are built. This really ties into your understanding of Object Pascal's implementation.

Third, you should be familiar with the VCL hierarchy and you should also know where the four component types previously mentioned fit into the VCL hierarchy. The following paragraphs will discuss each of these keys to understanding the VCL.

1.3.4 Component Types

As a component writer, there four primary types of components that you will work with in Delphi: standard controls, custom controls, graphical controls, and non-visual components. Although these component types are primarily of interest to component writers, it's not a bad idea for applications developers to be familiar with them. They are the foundations on which applications are built.

1.3.4.1 Standard Components

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox As a component writer, there four primary types of components that you will work with in Delphi: standard controls, custom controls, graphical controls, and non-visual components. Although these component types are primarily of interest to component writers, it's not a bad idea for applications developers to be familiar with them. They are the foundations on which applications are built.

For example. You will find these components on the Standard page of the Component Palette. These components are Windows' common controls with Object Pascal wrappers around them.

Each standard component looks and works like the Windows' common control which it encapsulates. The VCL wrapper's simply makes the control available to you in the form of a Delphi component-it doesn't define the common control's appearance or functionality, but rather, surfaces the ability to modify a control's appearance/functionality in the form of methods and properties. If you have the VCL source code, you can examine how the VCL wraps these controls in the file `STDCTRLS.PAS`.

If you want to use these standard components unchanged, there is no need to understand how the VCL wraps them. If, however, you want to extend or change one of these components, then you must understand how the Windows' common control is wrapped by the VCL into a Delphi component.

For example, the Windows class `LISTBOX` can display the list box items in multiple columns. This capability, however, isn't surfaced by Delphi's `TListBox` component (which encapsulates the Windows `LISTBOX` class). (`TListBox` only displays items in a single column.) Surfacing this capability requires that you override the default creation of the `TListBox` component.

This example also serves to illustrate why it is important for Applications Developers to understand the VCL. Just knowing this tidbit of information helps you to identify where enhancements to the existing library of components can help make your life easier and more productive.

1.3.4.2 Custom components

Unlike standard components, custom components are controls that don't already have a method for displaying themselves, nor do they have a defined behavior. The Component Writer must provide to code that tells the component how to draw itself and determines how the component behaves when the user interacts with it. Examples of existing custom components are the `TPanel` and `TStringGrid` components.

It should be mentioned here that both standard and custom components are *windowed* controls. A "windowed control" has a window associated with it and, therefore, has a window handle. Windowed controls have three characteristics: they can

receive the input focus, they use system resources, and they can be parents to other controls. (Parents is related to containership, discussed later in this paper.) An example of a component which can be a container is the TPanel component.

1.3.4.3 Graphical components

Graphical components are visual controls which cannot receive the input focus from the user. They are non-windowed controls. Graphical components allow you to display something to the user without using up any system resources; they have less "overhead" than standard or custom components. Graphical components don't require a window handle-thus, they cannot get focus. Some examples of graphical components are the TLabel and TShape components.

Graphical components cannot be containers of other components. This means that they cannot own other components which are placed on top of them.

1.3.4.4 Non-visual components

Non-visual components are components that do not appear on the form as controls at run-time. These components allow you to encapsulate some functionality of an entity within an object. You can manipulate how the component will behave, at design-time, through the Object Inspector. Using the Object Inspector, you can modify a non-visual component's properties and provide event handlers for its events. Examples of such components are the TOpenDialog, TTable, and TTimer components

1.3.4.5 Structure of a component

All components share a similar structure. Each component consists of common elements that allow developers to manipulate its appearance and function via properties, methods and events. The following sections in this paper will discuss these common elements as well as talk about a few other characteristics of components which don't apply to all components

1.3.5 Component properties

Properties provide an extension of an object's fields. Unlike fields, properties do not store data: they provide other capabilities. For example, properties may use methods to read or write data to an object field to which the user has no access. This adds a certain level of protection as to how a given field is assigned data. Properties also cause "side effects" to occur when the user makes a particular assignment to the property. Thus what appears as a simple field assignment to the component user could trigger a complex operation to occur behind the scenes.

1.3.6 Properties provide access to internal storage fields

There are two ways that properties provide access to internal storage fields of components: directly or through access methods. Examine the code below which illustrates this process.

```
TCustomEdit = class(TWinControl)
private
    FMaxLength: Integer;
protected
    procedure SetMaxLength(Value: Integer);
...
published
    property MaxLength: Integer read
        FMaxLength write SetMaxLength default 0;
...
end;
```

The code above is snippet of the TCustomEdit component class. TCustomEdit is the base class for edit boxes and memo components such as TEdit, and TMemo.

TCustomEdit has an internal field FMaxLength of type Integer which specifies the maximum length of characters which the user can enter into the control. The user doesn't directly access the FMaxLength field to specify this value. Instead, a value is added to this field by making an assignment to the MaxLength property.

The property `MaxLength` provides the access to the storage field `FMaxLength`. The property definition is comprised of the property name, the property type, a read declaration, a write declaration and optional default value.

The read declaration specifies how the property is used to read the value of an internal storage field. For instance, the `MaxLength` property has direct read access to `FMaxLength`. The write declaration for `MaxLength` shows that assignments made to the `MaxLength` property result in a call to an *access method* which is responsible for assigning a value to the `FMaxLength` storage field. This access method is `SetMaxLength`.

1.4Property-access methods

Access methods take a single parameter of the same type as the property. One of the primary reasons for write access methods is to cause some side-effect to occur as a result of an assignment to a property. Write access methods also provide a method layer over assignments made to a component's fields. Instead of the component user making the assignment to the field directly, the property's write access method will assign the value to the storage field if the property refers to a particular storage field. For example, examine the implementation of the `SetMaxLength` method below.

```
procedure TCustomEdit.SetMaxLength(Value: Integer);
begin
  if FMaxLength <> Value then
  begin
    FMaxLength := Value;
    if HandleAllocated then
      SendMessage(Handle, EM_LIMITTEXT, Value, 0);
  end;
end;
```

The code in the `SetMaxLength` method checks if the user is assigning the same value as that which the property already holds. This is done as a simple optimization. The method then assigns the new value to the internal storage field, `FMaxLength`. Additionally, the method then sends an `EM_LIMITTEXT` Windows message to the window which the `TCustomEdit` encapsulates. The `EM_LIMITTEXT` message places a limit on the amount of text that a user can enter into an edit control. This last step is

what is referred to as a *side-effect* when assigning property values. Side effects are any additional actions that occur when assigning a value to a property and can be quite sophisticated.

Providing access to internal storage fields through property access methods offers the advantage that the Component Writer can modify the implementation of a class without modifying the interface. It is also possible to have access methods for the read access of a property. The read access method might, for example, return a type which is different than that of a properties storage field. For instance, it could return the string representation of an integer storage field.

Another fundamental reason for properties is that properties are accessible for modification at run-time through Delphi's Object Inspector. This occurs whenever the declaration of the property appears in the published section of a component's declaration.

1.5 Types of properties

Properties can be of the standard data types defined by the Object Pascal rules. Property types also determine how they are edited in Delphi's Object Inspector. The table below shows the different property types as they are defined in Delphi's online help.

1.6 Methods

Since components are really just objects, they can have methods. We will discuss some of the more commonly used methods later in this paper when we discuss the different levels of the VCL hierarchy.

1.7 Events

Events provide a means for a component to notify the user of some pre-defined occurrence within the component. Such an occurrence might be a button click or the pressing of a key on a keyboard.

Components contain special properties called events to which the component user assigns code. This code will be executed whenever a certain event occurs. For instance, if you look at the events page of a TEdit component, you'll see such events as OnChange, OnClick and OnDblClick. These events are nothing more than pointers to methods.

When the user of a component assigns code to one of those events, the user's code is referred to as an event handler. For example, by double clicking on the events page for a particular event causes Delphi to generate a method and places you in the Code Editor where you can add your code for that method. An example of this is shown in the code below, which is an OnClick event for a TButton component.

TButton component.

```
TForm1 = class(TForm)
    Button1: Tbutton;
    procedure Button1Click(Sender: TObject);
end;
...
procedure TForm1.Button1Click(Sender: TObject);
begin
    { Event code goes here }
end;
```

It becomes clearer that events are method pointers when you assign an event handler to an event programmatically. The above example was Delphi generated code. To link your own an event handler to a TButton's OnClick event at run time you must first create a method that you will assign to this event. Since this is a method, it must belong to an existing object. This object can be the form which owns the TButton component although it doesn't have to be. In fact, the event handlers which Delphi creates belong to the form on which the component resides. The code below illustrates how you would create an event handler method.

```
TForm1 = class(TForm)
    Button1: TButton;
private
    MyOnClickEvent(Sender: TObject); // Your method declaration
end;
...
```

```

{ Your method definition below }
procedure TForm1.MyOnClickEvent(Sender: TObject);
begin
    { Your code goes here }
end;

```

The MyOnClickEvent method becomes the event handler for Button1.OnClick when it is assigned to Button1.OnClick in code as shown below.

```
Button1.OnClick := MyOnClickEvent
```

This assignment can be made anytime at runtime, such as in the form's OnCreate event handler. This is essentially the same thing that happens when you create an event handler through Delphi's Object Inspector except that Delphi generates the method declaration.

When you define methods for event handlers, these methods must be defined as the same type as the event property and the field to which the event property refers. For instance, the OnClick event refers to an internal data field, FOnClick. Both the property OnClick, and field FOnClick are of the type TNotifyEvent. TNotifyEvent is a procedural type as shown below:

```
TNotifyEvent = procedure (Sender: TObject) of object;
```

Therefore, if you are creating a method for an OnClick event, it must be defined with the same type and number of parameters as shown below.

```

TForm1 = class(TForm)
...
    procedure (Sender: TObject);
end;

```

Note the use of the of object specification. This tells the compiler that the procedure definition is actually a method and performs some additional logic like ensuring that an implicit Self parameter is also passed to this method when called. Self is just a pointer reference to the class to which a method belongs.

1.8 Containership

Some components in the VCL can own other components as well as be parents to other components. These two concepts have a different meaning as will be discussed in the section to follow.

1.9 Ownership

All components may be owned by other components but not all components can own other components. A component's Owner property contains a reference to the component which owns it.

The basic responsibility of the owner is one of resource management. The owner is responsible for freeing those components which it owns whenever it is destroyed. Typically, the form owns all components which appear on it, even if those components are placed on another component such as a TPanel. At design-time, the form automatically becomes the owner for components which you place on it. At run-time, when you create a component, you pass the owner as a parameter to the component's constructor. For instance, the below shows how to create a TButton component at run-time and passes the form's implicit Self variable to the TButton's Create constructor. TButton.Create will then assign whatever is passed to it, in this case Self or rather the form, and assign it to the button's Owner property.

```
MyButton := TButton.Create(self);
```

When the form that now owns this TButton component gets freed, MyButton will also be freed.

You can create a component without an owner by passing nil to the component's Create constructor, however, you must ensure that the component is freed when it is no longer needed. The code below shows you how to do this for a TTable component.

```
MyTable := TTable.Create(nil)
try
  { Do stuff with MyTable }
finally
  MyTable.Free;
end;
```

As shown in the code above, it is best to use a try..finally block to ensure that the component gets freed even if an exception were to be raised.

The Components property of a component is an array property which contains a list of the components which it owns. For instance, the code below shows how to loop through a form's components and then shows their class name.

```
var
  I: integer;
begin
  for I := 0 to ComponentCount - 1 do
    ShowMessage(Components[i].ClassName);
  end;
```

1.10 Parenthood

Parenthood is a much different concept from ownership. It applies only to windowed components, which can be parents to other components. Later, when we discuss the VCL hierarchy, you will see the level in the hierarchy which introduces windowed controls.

Parent components are responsible for the display of other components. They call the appropriate methods internally that cause the children components to draw themselves. The Parent property of a component refers to the component which is its parent. Also, a component's parent does not have to be its owner. Although the parent component is mainly responsible for the display of components, it also frees children components when it is destroyed.

Windowed components are controls which are visible user interface elements such as edit controls, list boxes and memo controls. In order for a windowed component to be displayed, it must be assigned a parent on which to display itself.

CHAPTER 2

2.1 INTRODUCTION TO DATABASE

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

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

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

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

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

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

2.2 HISTORY

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

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

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

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

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

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

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

2.3 DATABASE MODELS

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

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

2.3.1 Flat model

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

2.3.2 Hierarchical model

In a hierarchical model, data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list. Hierarchical structures were widely used in the early mainframe database management systems, such as the Information Management System (IMS) by IBM, and now describe the structure of XML documents. This structure allows one 1:N relationship between two types of data. This structure is very

efficient to describe many relationships in the real world; recipes, table of contents, ordering of paragraphs/verses, any nested and sorted information. However, the hierarchical structure is inefficient for certain database operations when a full path (as opposed to upward link and sort field) is not also included for each record.

2.3.3 Network model

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

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

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

2.3.4 Relational model

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

The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model defined by Codd. The data structures in these products are tables, rather than relations: the main differences being that tables can contain duplicate rows, and that the rows (and columns) can be treated as being ordered. The same criticism applies to the SQL language which is the primary interface to these products. There has been considerable controversy, mainly

due to Codd himself, as to whether it is correct to describe SQL implementations as "relational": but the fact is that the world does so, and the following description uses the term in its popular sense.

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

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

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

2.3.4.1 Relational operations

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

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

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

There are a number of relational operations in addition to join. These include project (the process of eliminating some of the columns), restrict (the process of eliminating some of the rows), union (a way of combining two tables with similar structures), difference (which lists the rows in one table that are not found in the other), intersect (which lists the rows found in both tables), and product (mentioned above, which combines each row of one table with each row of the other). Depending on which other sources you consult, there are a number of other operators - many of which can be defined in terms of those listed above. These include semi-join, outer operators such as outer join and outer union, and various forms of division. Then there are operators to rename columns, and summarizing or aggregating operators, and if you permit relation values as attributes (RVA - relation-valued attribute), then operators such as group and ungroup. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially

important for databases that might be used for decades. This has made the idea and implementation of relational databases very popular with businesses.

2.3.5 Dimensional model

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

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

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

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

2.3.6 Object database models

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

example as rows in tables) and its representation in the application program (typically as objects). At the same time object databases attempt to introduce the key ideas of object programming, such as encapsulation and polymorphism, into the world of databases.

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

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

2.4 DATABASE INTERNALS

2.4.1 Indexing

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

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

affect performance, any SQL command will run with or without indexes existing in the database.

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

2.4.2 Transactions and concurrency

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

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

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

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

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

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

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

2.4.3 Replication

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

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

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

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

2.5 APPLICATIONS OF DATABASES

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

2.6 DATABASE BRANDS

(In alphabetical order)

4D
Adabas
Adaptive Server Enterprise
Corel Paradox
Dataflex
Dataphor
DB2
Filemaker
Firebird
Information Management System
Informix
Ingres
Intersystem Cache
Kx
Microsoft Access
Microsoft SQL Server
MySQL
Netezza
Openoffice.org
Oracle
PostgreSQL
Progress
Rel(DBMS)
SQLite
SQL Anywhere Studio
Teradata
VistaDB

CHAPTER 3

3.1 SQL (STRUCTURED QUERY LANGUAGE):

SQL (Structured Query Language) is the most popular computer language used to create, modify, retrieve and manipulate data from relational database management systems. The language has evolved beyond its original purpose to support object-relational database management systems. It is an ANSI/ISO standard.

3.2 HISTORY

An influential paper, "A Relational Model of Data for Large Shared Data Banks", by Dr. Edgar F. Codd, was published in June, 1970 in the Association for Computing Machinery (ACM) journal, Communications of the ACM, although drafts of it were circulated internally within IBM in 1969. Codd's model became widely accepted as the definitive model for relational database management systems (RDBMS or RDMS).

During the 1970s, a group at IBM's San Jose research center developed a database system "System R" based upon, but not strictly faithful to, Codd's model. Structured English Query Language ("SEQUEL") was designed to manipulate and retrieve data stored in System R. The acronym SEQUEL was later condensed to **SQL** because the word 'SEQUEL' was held as a trademark by the Hawker-Siddeley aircraft company of the UK. Although SQL was influenced by Codd's work, Donald D. Chamberlin and Raymond F. Boyce at IBM were the authors of the SEQUEL language design. Their concepts were published to increase interest in SQL.

The first non-commercial, relational, non-SQL database, Ingres, was developed in 1974 at U.C. Berkeley.

In 1978, methodical testing commenced at customer test sites. Demonstrating both the usefulness and practicality of the system, this testing proved to be a success for IBM. As a result, IBM began to develop commercial products based on their System R prototype that implemented SQL, including the System/38 (announced in 1978 and commercially available in August 1979), SQL/DS (introduced in 1981), and DB2 (in 1983).

At the same time Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Chamberlin and Boyce and developed their own version of a RDBMS for the Navy, CIA and others. In the summer of 1979 Relational Software,

Inc. introduced Oracle V2 (Version2) for VAX computers as the first commercially available implementation of SQL. Oracle is often incorrectly cited as beating IBM to market by two years, when in fact they only beat IBM's release of the System/38 by a few weeks. Considerable public interest then developed; soon many other vendors developed versions, and Oracle's future was ensured.

3.2.1 Standardization

SQL was adopted as a standard by ANSI (American National Standards Institute) in 1986 and ISO (International Organization for Standardization) in 1987. ANSI has declared that the official pronunciation for SQL is /ɛs kju ɛl/, although many English-speaking database professionals still pronounce it as sequel.

The SQL standard has gone through a number of revisions:

Year	Name	Alias	Comments
------	------	-------	----------

1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
------	--------	--------	---

1989	SQL-89		Minor revision.
------	--------	--	-----------------

1992	SQL-92	SQL2	Major revision (ISO 9075).
------	--------	------	----------------------------

1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.)
------	----------	------	---

2003	SQL:2003		Introduced XML-related features, window functions, standardized sequences and columns with auto-generated values (including identity-columns).
------	----------	--	--

3.3 SCOPE

SQL is defined by both ANSI and ISO. Extensions to and variations of the standards exist: Oracle Corporation's PL/SQL, IBM's SQL PL (SQL Procedural Language) and Sybase / Microsoft's Transact-SQL, which are of a proprietary nature. Commercial implementations commonly omit support for basic features of the standard, such as the `DATE` or `TIME` data types, preferring variations of their own. SQL code can rarely be ported between database systems without major modifications, in contrast to ANSI C or ANSI Fortran, which can usually be ported from platform to platform without major structural changes.

SQL is designed for a specific, limited purpose — querying data contained in a relational database. As such, it is a set-based, declarative computer language rather than an imperative language such as C or BASIC which, being general-purpose, are designed to solve a much broader set of problems.

Language extensions such as PL/SQL bridge this gap to some extent by adding procedural elements, such as flow-of-control constructs. Another approach is to allow programming language code to be embedded in and interact with the database. For example, Oracle and others include Java in the database, and SQL Server 2005 allows any .NET language to be hosted within the database server process, while PostgreSQL allows functions to be written in a wide variety of languages, including Perl, Tcl, and C.

There are several reasons for this lack of portability between database systems:

The complexity and size of the SQL standard means that most databases do not implement the entire standard.

The standard does not specify database behavior in several important areas (e.g. indexes), leaving it up to implementations of the standard to decide how to behave.

The SQL standard precisely specifies the syntax that a conforming database system must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to areas of ambiguity.

Many database vendors have large existing customer bases; where the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.

Some believe the lack of compatibility between database systems is intentional in order to ensure vendor lock-in.

3.4 SQL KEYWORDS

SQL keywords fall into several groups.

3.4.1 Data retrieval

The most frequently used operation in transactional databases is the data retrieval operation. When restricted to data retrieval commands, SQL acts as a declarative language.

SELECT is used to retrieve zero or more rows from one or more tables in a database. In most applications, SELECT is the most commonly used Data Manipulation Language command. In specifying a SELECT query, the user specifies a description of the desired result set, but they do not specify what physical operations must be executed to produce that result set. Translating the query into an efficient query plan is left to the database system, more specifically to the query optimizer.

Commonly available keywords related to SELECT include:

FROM is used to indicate from which tables the data is to be taken, as well as how the tables JOIN to each other.

WHERE is used to identify which rows to be retrieved, or applied to GROUP BY. WHERE is evaluated before the GROUP BY.

GROUP BY is used to combine rows with related values into elements of a smaller set of rows.

HAVING is used to identify which of the "combined rows" (combined rows are produced when the query has a GROUP BY keyword or when the SELECT part contains aggregates), are to be retrieved. HAVING acts much like a WHERE, but it operates on the results of the GROUP BY and hence can use aggregate functions.

ORDER BY is used to identify which columns are used to sort the resulting data. Data retrieval is very often combined with data projection; usually it isn't the verbatim data stored in primitive data types that a user is looking for or a query is written to serve. Often the data needs to be expressed differently from how it's stored. SQL allows a wide variety of formulas included in the select list to project data.

Ex: To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

SELECT LastName,FirstName FROM Persons
--

The database table "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Karina	Storgt 20	Stavanger

The result

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Karina

3.4.2 Data manipulation

First there are the standard Data Manipulation Language (DML) elements. DML is the subset of the language used to add, update and delete data.

INSERT is used to add zero or more rows (formally tuples) to an existing table.

UPDATE is used to modify the values of a set of existing table rows.

MERGE is used to combine the data of multiple tables. It is something of a combination of the INSERT and UPDATE elements. It is defined in the SQL:2003 standard; prior to that, some databases provided similar functionality via different syntax, sometimes called an "upsert".

DELETE removes zero or more existing rows from a table.

Example:

```
INSERT INTO my_table (field1, field2, field3) VALUES ('test', 'N', NULL);
UPDATE my_table SET field1 = 'updated value' WHERE field2 = 'N';
DELETE FROM my_table WHERE field2 = 'N';
```


3.4.3 Transaction Control

Transaction, if available, can be used to wrap around the DML operations.

BEGIN WORK (or START TRANSACTION, depending on SQL dialect) can be used to mark the start of a database transaction, which either completes completely or not at all.

COMMIT causes all data changes in a transaction to be made permanent.

ROLLBACK causes all data changes since the last COMMIT or ROLLBACK to be discarded, so that the state of the data is "rolled back" to the way it was prior to those changes being requested.

COMMIT and ROLLBACK interact with areas such as transaction control and locking. Strictly, both terminate any open transaction and release any locks held on data. In the absence of a BEGIN WORK or similar statement, the semantics of SQL are implementation-dependent.

Example:

```
BEGIN WORK;  
UPDATE inventory SET quantity = quantity - 3 WHERE item = 'pants';  
COMMIT;
```

3.4.4 Data definition

The second group of keywords is the Data Definition Language (DDL). DDL allows the user to define new tables and associated elements. Most commercial SQL databases have proprietary extensions in their DDL, which allow control over nonstandard features of the database system.

The most basic items of DDL are the CREATE, ALTER, RENAME, TRUNCATE and DROP commands.

CREATE causes an object (a table, for example) to be created within the database.

DROP causes an existing object within the database to be deleted, usually irretrievably.

TRUNCATE deletes all data from a table (non-standard, but common SQL command).

ALTER command permits the user to modify an existing object in various ways -- for example, adding a column to an existing table.

Example:

```
CREATE TABLE my_table {  
my_field1 INT,  
my_field2 VARCHAR (50),  
my_field3 DATE NOT NULL,  
PRIMARY KEY (my_field1, my_field2)  
}
```

3.4.5 Data control

The third group of SQL keywords is the Data Control Language (DCL). DCL handles the authorization aspects of data and permits the user to control who has access to see or manipulate data within the database.

Its two main keywords are:

GRANT — authorizes one or more users to perform an operation or a set of operations on an object.

REVOKE — removes or restricts the capability of a user to perform an operation or a set of operations.

Example:

```
GRANT SELECT, UPDATE ON my_table TO some_user, another_user
```

3.4.6 Other

ANSI-standard SQL supports -- as a single line comment identifier (some extensions also support curly brackets or C-style /* comments */ for multi-line comments).

Example:

```
SELECT * FROM inventory -- Retrieve everything from inventory table
```

Database system using SQL

3.5 CRITICISMS OF SQL

Technically, SQL is a declarative computer language for use with "SQL databases". Theorists and some practitioners note that many of the original SQL features were inspired by, but in violation of, the relational model for database management and its tuple calculus realization. Recent extensions to SQL achieved relational completeness, but have worsened the violations, as documented in The Third Manifesto.

In addition, there are also some criticisms about the practical use of SQL:

Implementations are inconsistent and, usually, incompatible between vendors. In particular date and time syntax, string concatenation, nulls, and comparison case sensitivity often vary from vendor-to-vendor.

The language makes it too easy to do a Cartesian join, which results in "run-away" result sets when `WHERE` clauses are mistyped. Cartesian joins are so rarely used in practice that requiring an explicit `CARTESIAN` keyword may be warranted.

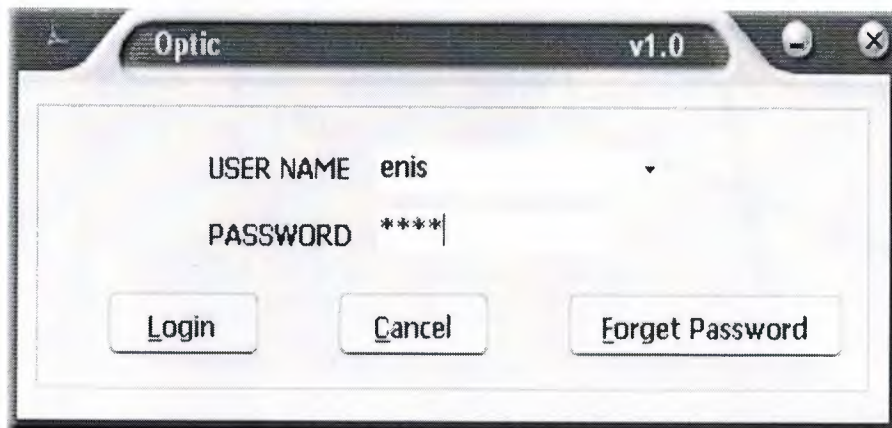
3.6 LOGICAL OPERATORS:

Logical constructs consist of other logical constructs, formulas and values, which are connected through logical operators.

Operator	Description	Example	Description
AND	Means that the values at both sides of the operator must be TRUE, otherwise this operator returns FALSE. The execution priority is (in SQL) bigger than OR, but smaller than NOT.	Price>10 AND Preis<100	Price is bigger than 10 and smaller than 100....
NOT	Logical negation, makes from FALSE an TRUE value and vice versa. Biggest execution priority of all logical operators.	NOT preis=0	Price is not zero
OR	Means that at least one of the values at both sides of the operator must be TRUE, otherwise this operator returns FALSE. This operator is executed after NOT and AND.	Preis>10 AND Preis<100 OR Preis>1000	Price must be bigger than 10 and smaller than 100 or bigger than 1000
=	Is TRUE when the values at both sides of the operator are equal. The execution priority is bigger than these of NOT, AND and OR.	Preis=10	Price is equal 10
>	Is TRUE when the value at the left side of the operator is greater than the value at the right side. The execution priority is bigger than these of NOT, AND and OR.	Preis>0	Price is bigger than 0
>=	Is TRUE when the value at the left side of the operator is greater or equal to the value at the right side. The execution priority is bigger than these of NOT, AND and OR.	Preis>=300	Price is bigger than or equal to 300
<	Is TRUE when the value at the left side of the operator is smaller than the value at the right side. The execution priority is bigger than these of NOT, AND and OR.	Preis<300	Price is smaller than 300
<=	Is TRUE when the value at the left side of the operator is smaller or equal to the value at the right side. The execution priority is bigger than these of NOT, AND and OR.	Price<=300	Price is smaller than or equal to 300
<>	Is TRUE when the value at the left side of the operator is not equal to the value at the right side. The execution priority is bigger than these of NOT, AND and OR.	Price <>0	Price is not equal 0

CHAPTER 4

Description About Project

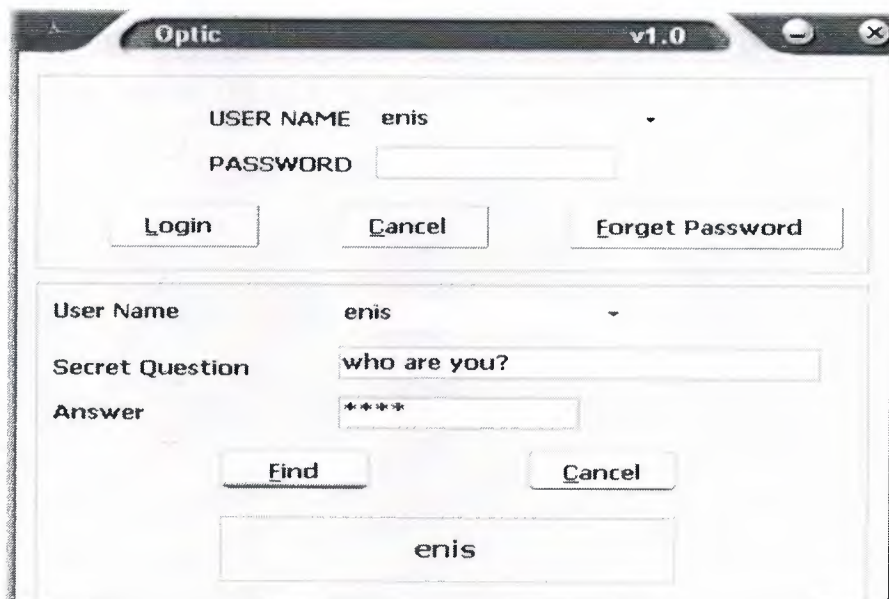


Enter the Program

The first form of the program is designed for login. For entering the program, we must enter the user name and password. If password is true then login the optician register and stock program. An individual, who is working in any of these predefined type can login to the program by using a predefined password.

If you forget the password click the 'Forget Password' button and we will see new form. Then enter the username we will see your secret question after answer the your question and click the find button, you can see your password if your answer is true.

If you have not a register you do not use this optic program.



Forget password form



Main form

In main form contain the main menu which include your requirement. It has customer, stock , account, and uses tools. If click the customer button we can see some concerned applications. If click the stock button we will see the glass stock and frame stock applications. If you choose the account section we can see revenue and expenditure register. Tools section has a contain new user register , add frame trademark , and other applications which calculator, word, excel , notepad. At last section is a exit button, if we click this we can see some alternatives which save and not save exit.

And in main form include the current date and current time in the screen every time. Shortly this form include the all applications in the program.

New Customer Register

Customer Number Automatic Number

Name Sex

Surname

Register Date

Mobile Phone 0() -

Home Phone 0() -

Address

MÜŞTERİ NUMARASI	ADI	SOYADI	CEP TEL.
1	MSDNDSA	DMNDAS	0(121)34
672893	OZAN	EKİZ	0(553)53
2	ED	AAA	0(538)25
58346	DDA	SDSDA	0() -
150726	EN	IS	0(533)84

New Customer Register

Click the customer and new customer register on main form and will open form on top. In this form you can save , edit ,delete , clear about the customer data which name, surname, phone number, address ,date then you save this information your database. And you can see your customer details.

You give the automatic number everyone customer. Because you have a many customer and you dont remember the customers number. And use the this button give the automatic number.

Register List

List

☒ Customer Number

☐ Name

☐ Surname

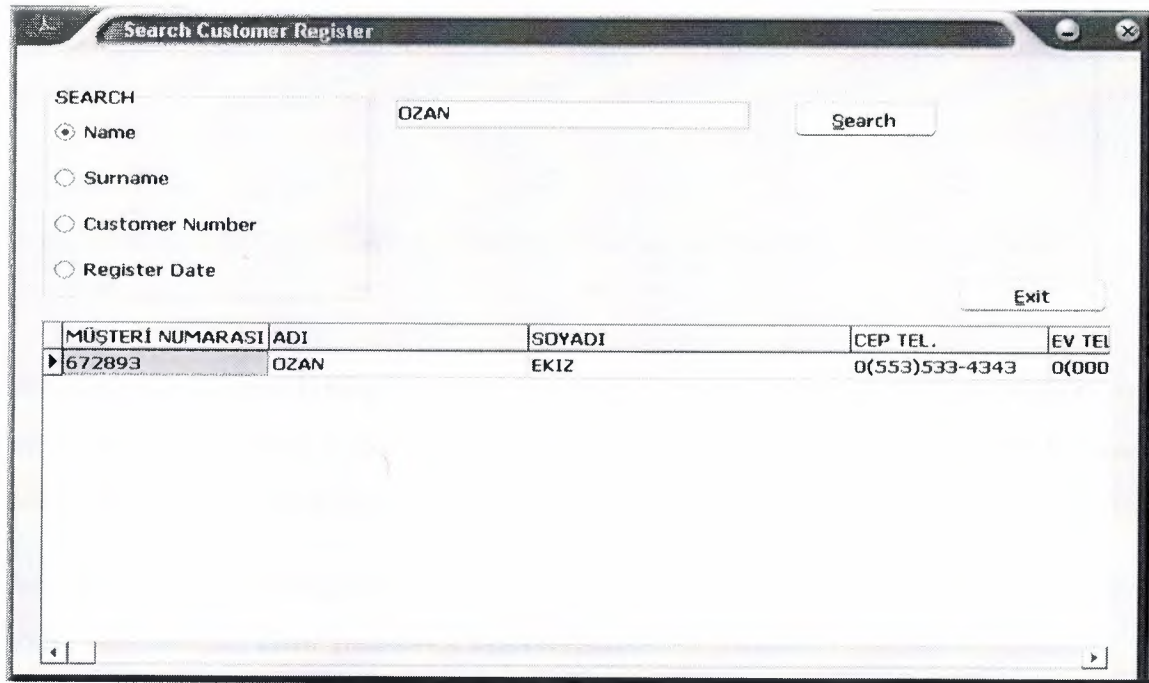
☐ Sex

MÜŞTERİ NUMARASI	ADI	SOYADI	CEP TEL.	EV TEL.
1	MSDNDSA	DMNDAS	0(121)343-3122	0(213)21
150726	EN	IS	0(533)845-7475	0(033)28
2	ED	AAA	0(538)254-7568	0(234)56
58346	DDA	SDSDA	0() -	0() -
672893	OZAN	EKİZ	0(553)533-4343	0(000)00

Customer Register List

If you want to see your customer list , you can use this form.

You will listed by number , name , surname and sex in your database and see MDATA table.



MÜŞTERİ NUMARASI	ADI	SOYADI	CEP TEL.	EV TEL.
672893	OZAN	EKİZ	0(553)533-4343	0(000

Customer search Form

If you search the customer in your database, you use the search customer form you can see customers information. Why you need this form? Because, your database have a many customer and not easy the find and you search according as name , surname, customer number and register date. Therefore you need the customer search form.

You click the stock and glass stock button you open the below form

Glass Stock

Kind of Glass Total .00YTL

Price .00YTL

ADET

Come From Company

Address

Come Date 05.01.2007

Save Edit Edit Clear Exit

Camcinsi	Bfiyat	Adet	Gelsirket	Geladr
INCE	120	12	ORIMEX	(MEMC
INCE	120	12	ORIMEX	(MEMC
ASDDSA	23	12	BASFB	(MEMC

We can enter the new kind of glass in our stock. And we secrete some information our memory which kind of glass, price, grain, company name, address, and date. We can see all information in the StockCam Table.

And enter the price and grain about the glass, it can calculate the total price. Then on the form, edit , delete , clear your stock information.

List of Glass Stock

LIST

☒ Date

☐ Price

☐ Company

☐ Total Kind of Glass INCE

This kind of glass in stock 24

Exit

Camcinsi	Bfiyat	Adet	Gelsirket	Geladresi	Genelto
ASDDSA	23	12	BASFB	(MEMO)	276
INCE	120	12	ORIMEX	(MEMO)	1440
INCE	120	12	ORIMEX	(MEMO)	1440

List of Glass Stock

In this form , we can list of the our stock, according the some values which date, price company name. And if choose the total kind of glass checkbox you will see total value selected glass in stock.

Frame Stock Register

Frame Trademark: ENISSS

Kind of Frame:

Price: .00YTL

ADET:

Company:

Address:

Date: 05.01.2007

General Total: .00YTL

Cmarka	Ccins	Bfiyat	Adet	Gsirket
EAX	1	2000	50	2
TOPTEN	2	1500	10	2
CERMAR	INCE	20000	10	ADANAOP

Save Edit Delete Clear Exit

Frame Stock Register

In the form we can register the new frame for our stock. This form like the glass stock. We can enter the new frame in our stock. And we secrete some information our memory which kind of frame, price, grain, company name, address, date and general total.

Frame Stock List

LIST

☐ Date

☐ Price

☐ Company

☒ I want to search for frame mark

ENISSS

CERMAR

Total frame in stock

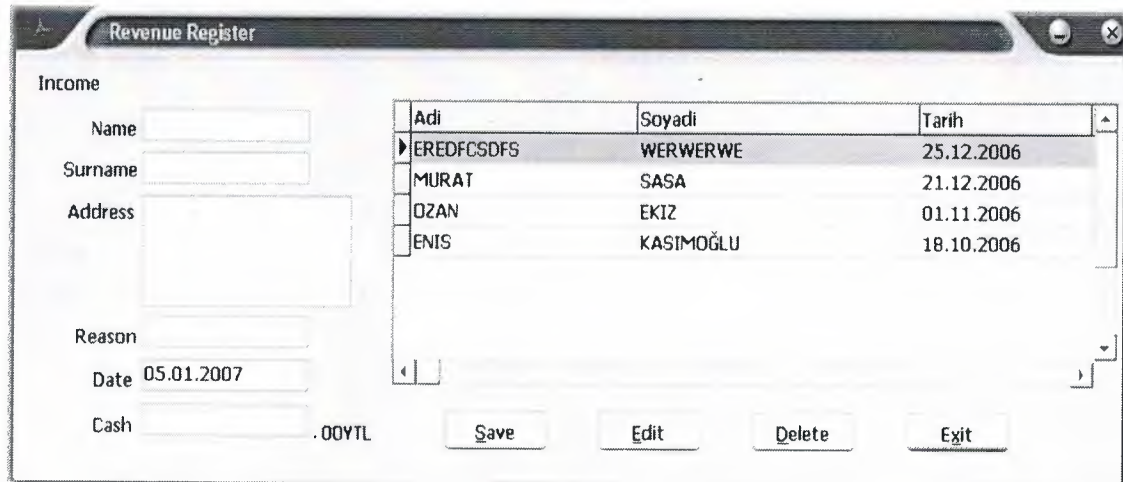
Exit

MARKASI	CİNSİ	BİRİM FİYATI	ADET	ŞİRKET	ADRES	GENEL TOPLAM	TARİH
EAX	1	2000	50	2	(MEMO)	100000	28.12.2006
TOPTEN	2	1500	10	2	(MEMO)	15000	28.05.2006
CERMAR	INCE	20000	10	ADANAOP	(MEMO)	200000	28.05.2006

Frame Stock List

In this form we can list the frame in our stock. we want to know , how many frame have got stock. And use the this form then listed an according the some values which date, price company name.

Click the account and revenue ;

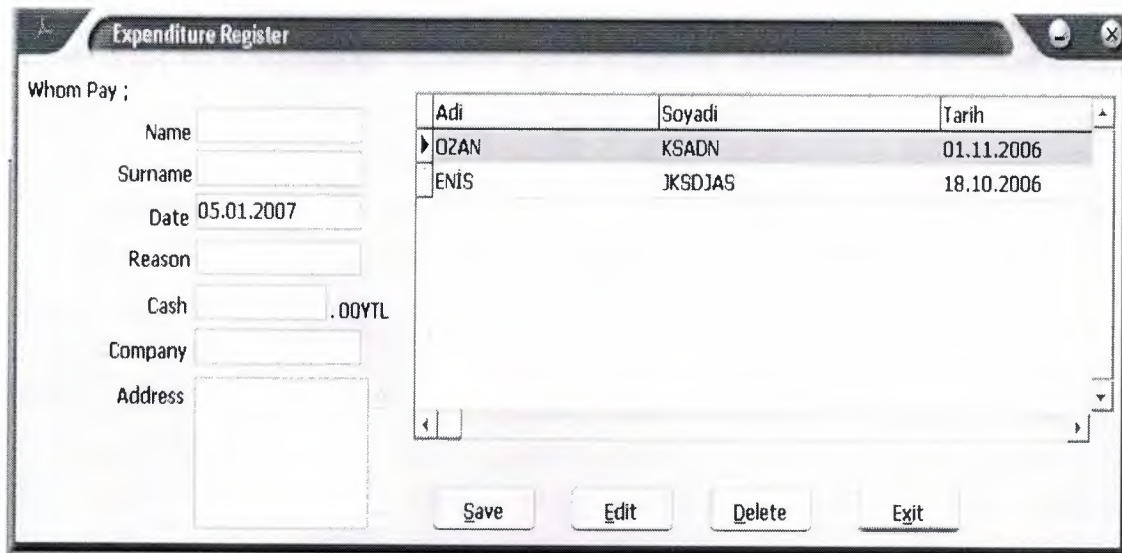


The Revenue Register form is a software window with a title bar. It contains several input fields on the left for 'Income' information: Name, Surname, Address, Reason, Date (pre-filled with 05.01.2007), and Cash (pre-filled with .00YTL). On the right, there is a table with three columns: 'Adi', 'Soyadi', and 'Tarih'. The table contains four rows of data. Below the table are four buttons: 'Save', 'Edit', 'Delete', and 'Exit'.

Adi	Soyadi	Tarih
EREDFCSDFS	WERWERWE	25.12.2006
MURAT	SASA	21.12.2006
OZAN	EKIZ	01.11.2006
ENIS	KASIMOĞLU	18.10.2006

Revenue Register

In this form, we can register the income information the our company which include the customer name, surname, address information and why take the money from the customer.



The Expenditure Register form is a software window with a title bar. It contains several input fields on the left for 'Whom Pay' information: Name, Surname, Date (pre-filled with 05.01.2007), Reason, Cash (pre-filled with .00YTL), Company, and Address. On the right, there is a table with three columns: 'Adi', 'Soyadi', and 'Tarih'. The table contains two rows of data. Below the table are four buttons: 'Save', 'Edit', 'Delete', and 'Exit'.

Adi	Soyadi	Tarih
OZAN	KSADN	01.11.2006
ENIS	JKSDJAS	18.10.2006

Expenditure Register

Expenditure register form for account the contain the expenses information about our company. Informations which name, surname, company name, company address, reason , cash.

Revenue - Expenditure List

Account

☐ List of Revenue

☒ List of Expenditure

Exit

Adi	Soyadi	Tarih	Sebeb	Miktar	K
DZAN	KSADN	01.11.2006	JDFJNF	53	0
ENİS	JKSDJAS	18.10.2006	SDSBDŞ	2138	8

Account List

Account list form include the revenue and expenditure informations. We listed revenue and expences. We can easly see the account informations.

Tools section contains the some applications. That include new user register, useful programs (word, excel, nodpad, calculator), add frame trademark, save section, and change user.

New User Register:

KULLANICI ADI	KAYIT TARİHİ	PROGRAM YETKİSİ
ed	21.12.2006	USER
enis	18.10.2006	ADMIN

New User Register

For entering the program , we can create the new user. And also necessary some informations which username, password, confirm password, secret qestion, answer, and authority.

Secret question necessary for forget the password, if we forget our password we answer the our question, if it is true it will be our password in the entering form.

In this form , confirm password not equal to password we can see warning message and edit our password.

Another point is a Authorization. If we selected ADMIN we can use the all application in the program, if we select USER we can use the limited the application in project. The authority of the user to reach , do changes and update the information in this program is limited with respect to the position according to letting users.

Kullanıcı.db has include Username , Register Date , Authorization informations you can see.

Another application is;

Add Frame Trademark:

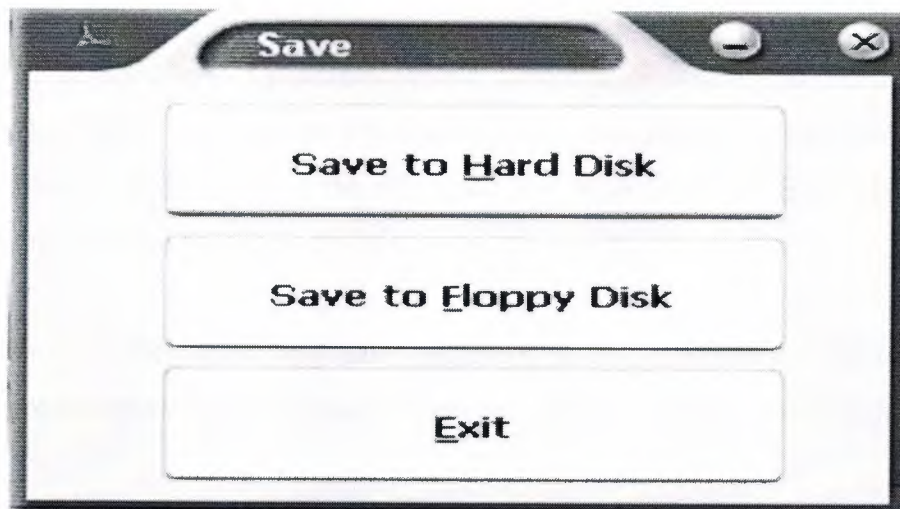
Frame Trademark	
	Marka
▶	ENISSS
	CERMAR

Add Frame Trademark

We can add the new frame trademark this form. And use the automatically this mark in frame stock register and frame stock list.

Tools section have a calculator, microsoft word, microsoft excel, and notepad for easily use.

Save Section:



Save

If we want to take the reserve, about the information in our computer harddisk or floppy disk we can use this section.

CONCLUSION

The OPTIC is optician automation. The program prepared this software by using Borland Delphi 7 and the SQL and Database Desktop. Both of them are powerful software and help me very much.

Borland Delphi is very easy language. It has many tools and components to help programmer. Delphi has got a many component, that easily used in the program.

Database desktop is very easy to create tables. Create tables without much effort with it.

SQL helped me very much. In one line of program code I solved many problems with it. OPTIC is for optician shops which is small but has to manage lot of data. It is easy to use and user friendly.

APPENDIX 1

Program Code

```
program optik;
uses
  windows,
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas' {Form3},
  Unit4 in 'Unit4.pas' {Form4},
  Unit5 in 'Unit5.pas' {Form5},
  Unit6 in 'Unit6.pas' {Form6},
  Unit7 in 'Unit7.pas' {Form7},
  Unit8 in 'Unit8.pas' {Form8},
  Unit9 in 'Unit9.pas' {Form9},
  Unit10 in 'Unit10.pas' {Form10},
  Unit11 in 'Unit11.pas' {Form11},
  Unit12 in 'Unit12.pas' {Form12},
  Unit13 in 'Unit13.pas' {Form13},
  Unit14 in 'Unit14.pas' {Form14},
  Unit15 in 'Unit15.pas' {Form15},
  Unit16 in 'Unit16.pas' {Form16},
  Unit17 in 'Unit17.pas' {Form17},
  Unit18 in 'Unit18.pas' {Form18},
  Unit19 in 'Unit19.pas' {Form19};
{$R *.res}
begin
  form19:=tform19.Create(nil);
  form19.Show;
  Application.ProcessMessages;
  Application.Initialize;
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm4, Form4);
  Application.CreateForm(TForm5, Form5);
  Application.CreateForm(TForm6, Form6);
  Application.CreateForm(TForm7, Form7);
  Application.CreateForm(TForm8, Form8);
  Application.CreateForm(TForm9, Form9);
  Application.CreateForm(TForm10, Form10);
  Application.CreateForm(TForm11, Form11);
  Application.CreateForm(TForm12, Form12);
  Application.CreateForm(TForm13, Form13);
  Application.CreateForm(TForm14, Form14);
  Application.CreateForm(TForm15, Form15);
  Application.CreateForm(TForm16, Form16);
  Application.CreateForm(TForm17, Form17);
```

```
Application.CreateForm(TForm18, Form18);
```

```
sleep(30000);  
form19.Hide;  
form19.Release;  
Application.Run;  
end.
```

Form 1

```
unit Unit1;  
interface  
uses  
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
  Dialogs, Menus, XPMenu, ExtCtrls, jpeg, StdCtrls, DB, DBTables;  
type  
  TForm1 = class(TForm)  
    MainMenu1: TMainMenu;  
    mterlemler1: TMenuItem;  
    stoklemler1: TMenuItem;  
    amstoul: TMenuItem;  
    stokgr1: TMenuItem;  
    stoklstes1: TMenuItem;  
    ercevelstes1: TMenuItem;  
    stokkaydı1: TMenuItem;  
    stoklstes2: TMenuItem;  
    carlemler1: TMenuItem;  
    aralar1: TMenuItem;  
    yenkullanıcıkaydı1: TMenuItem;  
    yardımcıprogramlar1: TMenuItem;  
    yedeklem1: TMenuItem;  
    hardskeyedeklem1: TMenuItem;  
    dsketeyedeklemel: TMenuItem;  
    hesabmaknes1: TMenuItem;  
    mcrosoftword1: TMenuItem;  
    mcrosoftexcel1: TMenuItem;  
    nodpad1: TMenuItem;  
    hakkında1: TMenuItem;  
    programdanıkı1: TMenuItem;  
    yedekeleık1: TMenuItem;  
    yedeklemedenık1: TMenuItem;  
    XPMenu1: TXPMenu;  
    Panel2: TPanel;  
    Timer1: TTimer;  
    mterkaydı1: TMenuItem;  
    yenkayıtl: TMenuItem;  
    slme1: TMenuItem;  
    dzeltme1: TMenuItem;  
    arama1: TMenuItem;  
    lstelemel: TMenuItem;
```

```

DataSource1: TDataSource;
Query1: TQuery;
kasal: TMenuItem;
kasagrds1: TMenuItem;
kasaiktısı1: TMenuItem;
kurumlemler1: TMenuItem;
yenkurumkaydı1: TMenuItem;
kasazet1: TMenuItem;
kasasinsondurumu1: TMenuItem;
ercevemarkasıekleme1: TMenuItem;
Image1: TImage;
About1: TMenuItem;
Edit1: TEdit;
changeuser1: TMenuItem;
procedure yedeklemedenık1click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure yenkullanıcıkaydı1click(Sender: TObject);
procedure slme1click(Sender: TObject);
procedure yenkayıt6click(Sender: TObject);
procedure lstel1click(Sender: TObject);
procedure arama2click(Sender: TObject);
procedure dzeltme2click(Sender: TObject);
procedure slme2click(Sender: TObject);
procedure kayıtslme2click(Sender: TObject);
procedure kayıtdzeltme4click(Sender: TObject);
procedure kayıtdzeltme5click(Sender: TObject);
procedure kayıtslme6click(Sender: TObject);
procedure kasanınsonhal1click(Sender: TObject);
procedure kasazetleme1click(Sender: TObject);
procedure kayıtslme5click(Sender: TObject);
procedure yenkayıt1click(Sender: TObject);
procedure dzeltme1click(Sender: TObject);
procedure lsteleme1click(Sender: TObject);
procedure arama1click(Sender: TObject);
procedure yenkayıt4click(Sender: TObject);
procedure kayıtslme4click(Sender: TObject);
procedure yenkayıt5click(Sender: TObject);
procedure kurumlstes1click(Sender: TObject);
procedure hesabmaknes1click(Sender: TObject);
procedure mcrosoftword1click(Sender: TObject);
procedure mcrosoftexcel1click(Sender: TObject);
procedure nodpad1click(Sender: TObject);
procedure hardskeyedeklem1click(Sender: TObject);
procedure dsketeyedekleme1click(Sender: TObject);
procedure yedekeleık1click(Sender: TObject);
procedure yenkurumkaydı1click(Sender: TObject);
procedure kasagrds1click(Sender: TObject);
procedure kasaiktısı1click(Sender: TObject);
procedure kasasinsondurumu1click(Sender: TObject);
procedure stoklstes1click(Sender: TObject);

```

```

procedure stokgr1click(Sender: TObject);
procedure ercevemarkasıkleme1click(Sender: TObject);
procedure stokkaydı1click(Sender: TObject);
procedure stoklstes2click(Sender: TObject);
procedure About1Click(Sender: TObject);
procedure changeuser1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
var
  Form1: TForm1;
implementation
uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit8, Unit9, Unit10, Unit11,
  Unit12, Unit14, Unit15, Unit16, Unit17, Unit18;
{$R *.dfm}
function ye(a:string):boolean;
begin
  ye:=false;
  form1.Query1.First;
  while not form1.Query1.eof do
  if (a=form1.Query1.Fields[0].asString) then
  begin
    ye:=true;
    exit;
  end
  else
    form1.Query1.Next;
  end;
procedure TForm1.yedeklemedenik1click(Sender: TObject);
begin
  form1.close;
  form3.close;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var zaman:TDateTime;
begin
  panel2.caption:=FormatDateTime('d mmmm yyyy dddd hh:nn:ss am/pm',Date+Time);
end;

procedure TForm1.yenkullanıcıkaydı1click(Sender: TObject);
begin
  ye(edit1.Text);
  if (query1.Fields[0].AsString=edit1.Text) then
  begin
    if (query1.Fields[1].AsString='1') then
    begin
      form2.Show;

```



```

form1.Enabled:=false;
end
else application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
!!','Warning',48);
end;
end;

```

```

procedure TForm1.slme1click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query1.Fields[1].AsString='0') then
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
!!','Warning',48)
else begin
form4.show;
with form4 do begin
with panel1 do begin
bitbtn2.enabled:=false;
bitbtn1.enabled:=false;
bitbtn3.enabled:=true;
end;
end;
form1.enabled:=false;
end;
end;
end;
end;

```

```

procedure TForm1.yenkayıt6click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query1.Fields[1].AsString='0') then
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
!!','Warning',48)
else begin
form13.show;
form1.Enabled:=false;
end; end; end;

```

```

procedure TForm1.lstel1click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query1.Fields[1].AsString='0') then
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
!!','Warning',48)

```



```
else begin  
end; end; end;
```

```
procedure TForm1.arama2click(Sender: TObject);  
begin  
ye(edit1.Text);  
if (query1.Fields[0].AsString=edit1.Text) then  
begin  
if (query1.Fields[1].AsString='0') then  
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS  
!!','Warning',48)  
else begin  
end; end;  
end;
```

```
procedure TForm1.dzeltme2click(Sender: TObject);  
begin  
ye(edit1.Text);  
if (query1.Fields[0].AsString=edit1.Text) then  
begin  
if (query1.Fields[1].AsString='0') then  
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS  
!!','Warning',48)  
else begin  
end;  
end;  
end;
```

```
procedure TForm1.slme2click(Sender: TObject);  
begin  
ye(edit1.Text);  
if (query1.Fields[0].AsString=edit1.Text) then  
begin  
if (query1.Fields[1].AsString='0') then  
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS  
!!','Warning',48)  
else begin
```

```
// form çağırma komutlari
```

```
end;
```

```
end;
```

```
end;
```

```
procedure TForm1.kayıtlme2click(Sender: TObject);
```

```
begin
```

```
ye(edit1.Text);
```

```
if (query1.Fields[0].AsString=edit1.Text) then
```

```
begin
```

```
if (query1.Fields[1].AsString='0') then
```

```
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
```

```
!!','Warning',48)
```

```
else begin
```

```
// form çağırma komutlari  
end; end; end;
```

```
procedure TForm1.kayıtdzeltme4click(Sender: TObject);  
begin  
form7.show;  
form1.Enabled:=false;  
with form7 do begin  
with panel1 do begin  
bitbtn1.enabled:=false;  
bitbtn2.enabled:=true;  
bitbtn4.enabled:=false;  
end;  
end;  
end;
```

```
procedure TForm1.kayıtdzeltme5click(Sender: TObject);  
begin  
form9.show;  
form1.enabled:=false;  
with form9 do begin  
bitbtn1.enabled:=false;  
bitbtn2.enabled:=true;  
bitbtn3.enabled:=false;  
end;  
end;
```

```
procedure TForm1.kayıtslme6click(Sender: TObject);  
begin  
form9.show;  
form1.enabled:=false;  
with form9 do begin  
bitbtn3.enabled:=true;  
bitbtn2.enabled:=false;  
bitbtn1.enabled:=false;  
end;  
end;
```

```
procedure TForm1.kasanınsonhal1click(Sender: TObject);  
begin  
ye(edit1.Text);  
if (query1.Fields[0].AsString=edit1.Text) then  
begin  
if (query1.Fields[1].AsString='0') then  
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS  
!!','Warning',48)  
else begin  
// form çağırma komutlari  
end;  
end; end;
```

```

procedure TForm1.kasazetleme1click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query1.Fields[1].AsString='0') then
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
!!','Warning',48)
else begin
// form çağırma komutlari
end;
end;
end;

```

```

procedure TForm1.kayıtlme5click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query1.Fields[1].AsString='0') then
application.MessageBox('THIS SECTION CAN USE ONLY ADMIN USERS
!!','Warning',48)
else begin
form7.show;
form1.Enabled:=false;
with form7 do begin
with panel1 do begin
bitbtn4.enabled:=true;
bitbtn2.enabled:=false;
bitbtn1.enabled:=false;
end;
end;
end;
end;
end;

```

```

procedure TForm1.yenkayıtlclick(Sender: TObject);
begin
form4.show;
with form4 do begin
with panel1 do begin
bitbtn2.enabled:=false;
bitbtn3.enabled:=false;
bitbtn1.enabled:=true;
end;
end;
form1.enabled:=false;
end;

```



```

procedure TForm1.dzeltme1click(Sender: TObject);
begin
  form4.show;
  with form4 do begin
    with panel1 do begin
      bitbtn1.enabled:=false;
      bitbtn3.enabled:=false;
      bitbtn2.enabled:=true;
    end;
  end;
  form1.enabled:=false;
end;

```

```

procedure TForm1.lsteleme1click(Sender: TObject);
begin
  form5.show;
  form1.Enabled:=false;
end;

```

```

procedure TForm1.arama1click(Sender: TObject);
begin
  form6.show;
  form1.Enabled:=false;
end;

```

```

procedure TForm1.yenkayıt4click(Sender: TObject);
begin
  form7.show;
  form1.Enabled:=false;
  with form7 do begin
    with panel1 do begin
      bitbtn1.enabled:=true;
      bitbtn2.enabled:=false;
      bitbtn4.enabled:=false;
    end;
  end;
end;

```

```

procedure TForm1.kayıtsıme4click(Sender: TObject);
begin
  form8.show;
  form1.Enabled:=false;
end;

```

```

procedure TForm1.yenkayıt5click(Sender: TObject);
begin
  form9.show;
  form1.enabled:=false;
  with form9 do begin

```

```
bitbtn1.enabled:=true;  
bitbtn2.enabled:=false;  
bitbtn3.enabled:=false;  
end;  
end;
```

```
procedure TForm1.kurumlstes1click(Sender: TObject);  
begin  
form10.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.hesabmaknes1click(Sender: TObject);  
begin  
winexec('C:\windows\system32\calc.exe',1);  
end;
```

```
procedure TForm1.mcrosoftword1click(Sender: TObject);  
begin  
winexec('C:\Program Files\Microsoft Office\Office11\winword.exe',1);  
end;
```

```
procedure TForm1.mcrosoftexcel1click(Sender: TObject);  
begin  
winexec('C:\Program Files\Microsoft Office\Office11\excel.exe',1);  
end;
```

```
procedure TForm1.nodpad1click(Sender: TObject);  
begin  
winexec('C:\windows\notepad.exe',1);  
end;
```

```
procedure TForm1.hardskeyedeklem1click(Sender: TObject);  
begin  
form11.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.dsketeyedekleme1click(Sender: TObject);  
begin  
form11.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.yedekle1click(Sender: TObject);  
begin  
winexec('yedeklehdd.bat',0);  
application.MessageBox('Saved C:\YEDEK.', 'Warning', 32);  
form1.Close;  
form3.close; end;
```

```
procedure TForm1.yenkurumkaydı1click(Sender: TObject);  
begin  
form7.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.kasagrds1click(Sender: TObject);  
begin  
form9.Show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.kasaıktısı1click(Sender: TObject);  
begin  
form8.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.kasasınsondurumu1click(Sender: TObject);  
begin  
form10.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.stoklstes1click(Sender: TObject);  
begin  
form15.show;  
form1.enabled:=false;  
end;
```

```
procedure TForm1.stokgr1click(Sender: TObject);  
begin  
form14.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.ercevemarkasıekleme1click(Sender: TObject);  
begin  
form16.show;  
form1.Enabled:=false;  
end;
```

```
procedure TForm1.stokkaydı1click(Sender: TObject);  
begin  
form17.show;  
form1.Enabled:=false;  
end;
```

```

procedure TForm1.Stoklstes2click(Sender: TObject);
begin
form18.show;
form1.enabled:=false;
end;

```

```

procedure TForm1.About1Click(Sender: TObject);
begin
form12.show;
form1.enabled:=false;
end;

```

```

procedure TForm1.changeuser1Click(Sender: TObject);
begin
form3.show;
form1.close;
end;
end.

```

Form 2

```

unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls, DB, DBTables,
  XPMenu, Menus;
type
  TForm2 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    ComboBox1: TComboBox;
    Panel1: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    Query1: TQuery;

```



```

DataSource2: TDataSource;
Query2: TQuery;
Query3: TQuery;
DataSource3: TDataSource;
BitBtn5: TBitBtn;
XPMenu1: TXPMenu;
PopupMenu1: TPopupMenu;
Clear1: TMenuItem;
Save1: TMenuItem;
Edit6: TMenuItem;
Delete1: TMenuItem;
Exit1: TMenuItem;
procedure BitBtn4Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBGrid1DbClick(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure Clear1Click(Sender: TObject);
procedure Save1Click(Sender: TObject);
procedure Edit6Click(Sender: TObject);
procedure Delete1Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form2: TForm2;
implementation
uses Unit1;
{$R *.dfm}
function ye(a:string):boolean;
begin
  ye:=false;
  form2.Query1.First;
  while not form2.Query1.eof do
    if (a=form2.Query1.Fields[0].asString) then
      begin
        ye:=true;
        exit;
      end
    else
      form2.Query1.Next;
  end;

  function ye1(a:string):boolean;
  begin

```



```

procedure TForm2.BitBtn2Click(Sender: TObject);
begin
if (edit2.text<>edit3.Text) then
application.MessageBox('Check confirm Password !!!','Warning',48)
ELSE begin
if (edit1.text='') and (edit2.text='') then
application.MessageBox('You must write username and password','Warning',48)
else begin
query1.edit;
query1.Fields[0].AsString:=edit1.Text;
query1.Fields[1].AsString:=edit2.Text;
query1.Fields[2].AsString:=edit4.Text;
query1.Fields[3].AsString:=edit5.Text;
query1.Fields[4].AsString:=datetostr(date());
query1.Fields[5].AsString:=combobox1.Text;
query1.post;
query2.edit;
query2.Fields[0].AsString:=edit1.Text;
query2.Fields[1].AsString:='0';
query2.Post;
edit1.Text:='';
edit2.Text:='';
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit1.setfocus;
query3.Refresh;
end;
end;
end;

```

```

procedure TForm2.DBGrid1DblClick(Sender: TObject);
var a:string;
begin
a:=dbgrid1.Fields[0].AsString;
ye(a);
if (query1.Fields[0].AsString=a) then
begin
edit1.Text:=query1.Fields[0].AsString;
edit2.Text:=query1.Fields[1].AsString;
edit3.Text:=query1.Fields[1].AsString;
edit4.Text:=query1.Fields[2].AsString;
edit5.Text:=query1.Fields[3].AsString;
combobox1.Text:=query1.Fields[5].AsString;
end;
end;

```

```

procedure TForm2.BitBtn3Click(Sender: TObject);
begin
ye(edit1.Text);
ye1(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query2.Fields[0].AsString=edit1.Text) then
begin
query2.delete;
end;
query1.Delete;
query3.Refresh;
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
end;
end;

```

```

procedure TForm2.BitBtn5Click(Sender: TObject);
begin
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
end;

```

```

procedure TForm2.Clear1Click(Sender: TObject);
begin
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
end;

```

```

procedure TForm2.Save1Click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
application.MessageBox('This username was using !!!','Warning',32)
else begin
if (edit2.text<>edit3.Text) then
application.MessageBox('Check confirm password !!!','Warning',32)
else begin
if (edit1.text="") and (edit2.text="") then
application.MessageBox('You must write Username and password ','Warning',32)
else begin

```



```

query1.Insert;
query1.Fields[0].AsString:=edit1.Text;
query1.Fields[1].AsString:=edit2.Text;
query1.Fields[2].AsString:=edit4.Text;
query1.Fields[3].AsString:=edit5.Text;
query1.Fields[4].AsString:=datetostr(date());
query1.Fields[5].AsString:=combobox1.Text;
query1.post;
query2.Insert;
query2.Fields[0].AsString:=edit1.Text;
query2.Fields[1].AsString:='0';
query2.Post;
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
edit1.setfocus;
query3.Refresh;
end;
end;
end;
end;

```

```

procedure TForm2.Edit6Click(Sender: TObject);
begin
if (edit2.text<>edit3.Text) then
application.MessageBox('Check confirm Password !!!','Warning',48)
ELSE begin
if (edit1.text="") and (edit2.text="") then
application.MessageBox('You must write username and password','Warning',48)
else begin
query1.edit;
query1.Fields[0].AsString:=edit1.Text;
query1.Fields[1].AsString:=edit2.Text;
query1.Fields[2].AsString:=edit4.Text;
query1.Fields[3].AsString:=edit5.Text;
query1.Fields[4].AsString:=datetostr(date());
query1.Fields[5].AsString:=combobox1.Text;
query1.post;
query2.edit;
query2.Fields[0].AsString:=edit1.Text;
query2.Fields[1].AsString:='0';
query2.Post;
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
edit1.setfocus;

```

```

query3.Refresh;
end;
end;
end;

procedure TForm2.Delete1Click(Sender: TObject);
begin
ye(edit1.Text);
ye1(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
if (query2.Fields[0].AsString=edit1.Text) then
begin
query2.delete;
end;
query1.Delete;
query3.Refresh;
edit1.Text="";
edit2.Text="";
edit3.Text="";
edit4.Text="";
edit5.Text="";
end;
end;

procedure TForm2.Exit1Click(Sender: TObject);
begin
form1.enabled:=true;
form2.Close;
end;
end.

```

Form 3

```

unit Unit3;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Buttons, DB, DBTables, XPMenu, WinSkinData;

type
  TForm3 = class(TForm)
    Panel1: TPanel;
    Label3: TLabel;
    ComboBox2: TComboBox;
    Label4: TLabel;
    Label5: TLabel;
    Edit2: TEdit;
    Edit3: TEdit;

```

```

BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
DataSource1: TDataSource;
Query1: TQuery;
Query2: TQuery;
DataSource2: TDataSource;
Panel2: TPanel;
Label1: TLabel;
Label2: TLabel;
ComboBox1: TComboBox;
Edit1: TEdit;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
Panel3: TPanel;
XPMenu1: TXPMenu;
SkinData1: TSkinData;
procedure BitBtn1Click(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure ComboBox2Change(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form3: TForm3;
implementation
uses Unit1;
{$R *.dfm}
function ye(a:string):boolean;
begin
  ye:=false;
  form3.Query2.First;
  while not form3.Query2.eof do
  if (a=form3.Query2.Fields[0].asString) then
  begin
    ye:=true;
    exit;
  end
  else
    form3.Query2.Next;
  end;
end;

```

```

function bul(a:string;b:string):boolean;
begin
bul:=false;
form3.Query1.First;
while not form3.Query1.eof do begin
if (a=form3.Query1.Fields[0].asstring)and (b=form3.Query1.Fields[1].asstring)then
begin
bul:=true;
exit;
end
else
form3.Query1.Next;
end;
end;

```

```

function bul2(a:string;b:string):boolean;
begin
bul2:=false;
form3.Query1.First;
while not form3.Query1.eof do
if (a=form3.Query1.Fields[0].asstring)and (b=form3.Query1.Fields[3].asstring)then
begin
bul2:=true;
exit;
end
else
form3.Query1.Next;
end;

```

```

function bul1(a:string):boolean;
begin
bul1:=false;
form3.Query1.First;
while not form3.Query1.eof do
if (a=form3.Query1.Fields[0].asstring)then
begin
bul1:=true;
exit;
end
else
form3.Query1.Next;
end;

```

```

procedure TForm3.BitBtn1Click(Sender: TObject);
begin
if (edit1.text="" and (edit2.text="" ) then begin
application.MessageBox('You must write username and password','Warning',48);
end;
begin
ye(combobox1.Text);

```



```

    bul(form3.combobox1.Text,form3.edit1.Text);
    if (query1.Fields[0].AsString=combobox1.Text) and
    (query1.Fields[1].AsString=edit1.Text) then begin
    if (query1.Fields[5].AsString='ADMIN') then begin
    if (query2.Fields[0].AsString=combobox1.Text) then begin
    query2.Edit;
    query2.Fields[1].AsString:='1';
    query2.Post;
    form1.show;
    form3.Hide;
    form1.Edit1.Text:=form3.ComboBox1.Text;
    end;
    end else
    begin
    query2.Edit;
    query2.Fields[1].AsString:='0';
    query2.Post;
    form1.show;
    form3.Hide;
    form1.Edit1.Text:=form3.ComboBox1.Text;
    end;
    end;
    end;
    end;

```

```

procedure TForm3.FormActivate(Sender: TObject);
begin
    panel1.Visible:=false;
    combobox1.Items.Clear;
    combobox2.Items.Clear;
    while(not query1.Eof) do begin
    combobox1.Items.Add(query1.Fields[0].asString);
    combobox2.Items.Add(query1.Fields[0].asString);
    query1.Next;
    end;
    form3.Height:=176;
end;

```

```

procedure TForm3.BitBtn3Click(Sender: TObject);
begin
    combobox2.Text:="";
    edit2.Text:="";
    edit3.Text:="";
    panel3.Caption:="";
    form3.Height:=372;
    panel1.Visible:=true;
    panel2.Enabled:=false;
end;

```

```

procedure TForm3.BitBtn5Click(Sender: TObject);
begin
form3.Height:=176;
panel1.Visible:=false;
panel2.Enabled:=true;
end;

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

procedure TForm3.ComboBox2Change(Sender: TObject);
begin
bul1(combobox2.Text);
if (query1.Fields[0].AsString=combobox2.Text) then
begin
edit2.Text:=query1.Fields[2].AsString;
edit3.SetFocus;
end;
end;

procedure TForm3.BitBtn4Click(Sender: TObject);
begin
bul2(combobox2.Text,edit3.Text);
if (query1.Fields[0].AsString=combobox2.Text) and
(query1.Fields[3].AsString=edit3.Text) then
begin
panel3.Caption:=query1.Fields[1].AsString;;
end;
end;

procedure TForm3.Timer1Timer(Sender: TObject);
begin
if form1.AlphaBlendValue<255 then
form1.AlphaBlendValue:=form1.AlphaBlendValue+5
end;
end.

```

Form 4

```

unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls, XPMenu, Mask, DB,
  DBTables, Menus;

type
  TForm4 = class(TForm)

```

```

Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
MaskEdit1: TMaskEdit;
MaskEdit2: TMaskEdit;
Memo1: TMemo;
XPMenu1: TXPMenu;
Panel1: TPanel;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
DBGrid1: TDBGrid;
DataSource1: TDataSource;
Query1: TQuery;
Query2: TQuery;
DataSource2: TDataSource;
Label7: TLabel;
ComboBox1: TComboBox;
DataSource3: TDataSource;
Query3: TQuery;
Label8: TLabel;
MaskEdit3: TMaskEdit;
BitBtn6: TBitBtn;
BitBtn7: TBitBtn;
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure DBGrid1DblClick(Sender: TObject);
procedure BitBtn6Click(Sender: TObject);
procedure BitBtn7Click(Sender: TObject);

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

var
    Form4: TForm4;

```

```

implementation
uses Unit1;
{$R *.dfm}
function RandomString(PWLen: integer): string;
const StrTable: string =
    '1234567890';
var
    N, K, X, Y: integer;
begin
    Randomize;
    if (PWLen > Length(StrTable)) then K := Length(StrTable)-1
    else K := PWLen;
    SetLength(result, K);
    Y := Length(StrTable);
    N := 0;

    while N < K do begin
        X := Random(Y) + 1;
        if (pos(StrTable[X], result) = 0) then begin
            inc(N);
            Result[N] := StrTable[X];
        end;
    end;
end;

procedure kayit;
begin
    with form4 do begin
        query1.Fields[0].AsString:=edit1.Text;
        query1.Fields[1].AsString:=edit2.Text;
        query1.Fields[2].AsString:=edit3.Text;
        query1.Fields[3].AsString:=maskedit1.Text;
        query1.Fields[4].AsString:=maskedit2.Text;
        query1.Fields[5].AsString:=memo1.Text;
        query1.Fields[6].AsString:=combobox1.Text;
        query1.Fields[7].AsString:=maskedit3.Text;
    end;
end;

procedure oku;
begin
    with form4 do begin
        edit1.Text:=query1.Fields[0].AsString;
        edit2.Text:=query1.Fields[1].AsString;
        edit3.Text:=query1.Fields[2].AsString;
        maskedit1.Text:=query1.Fields[3].AsString;
        maskedit2.Text:=query1.Fields[4].AsString;
        memo1.Text:=query1.Fields[5].AsString;
        combobox1.Text:=query1.Fields[6].AsString;
        maskedit3.Text:=query1.Fields[7].AsString;
    end;
end;

```



```

procedure temizle;
begin
with form4 do begin
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
maskedit1.Text:="";
maskedit2.Text:="";
memo1.Text:="";
combobox1.Text:="";
maskedit3.Text:="";
edit1.SetFocus;
end;
end;
function mnum(a:string):boolean;
begin
mnum:=false;
form4.Query1.First;
while not form4.Query1.eof do
if (a=form4.Query1.Fields[0].asString) then
begin
mnum:=true;
exit;
end
else
form4.Query1.Next;
end;

```

```

function ye(a:string):boolean;
begin
ye:=false;
form4.Query1.First;
while not form4.Query1.eof do
if (a=form4.Query1.Fields[0].asString) then
begin
ye:=true;
exit;
end
else
form4.Query1.Next;
end;

```

```

procedure TForm4.BitBtn5Click(Sender: TObject);
begin
form1.enabled:=true;
form4.Close;
end;

```

```

procedure TForm4.BitBtn4Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('informations will clear ?','warning',36);
if a=IDYES then
begin
temizle;
end;
end;

```

```

procedure TForm4.BitBtn1Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('informations true?','Warning',36);
if a=IDYES then
begin
ye(edit1.Text);
if (query1.Fields[0].AsString<>edit1.Text) then
begin
query1.Insert;
kayit;
query1.Post;
temizle;
query3.Refresh;
end else application.MessageBox('This customer number is saved !!!','Warning',48);
end;
end;

```

```

procedure TForm4.BitBtn2Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('informations true?','Warning',36);
if a=IDYES then
begin
query1.Edit;
kayit;
query1.Post;
query3.Refresh;
temizle;
end;
end;
procedure TForm4.BitBtn3Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Are You Sure Delete?','Warning',36);
if a=IDYES then
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then begin
query1.Delete;

```

```

query3.Refresh;
temizle;
end;
end;
end;

procedure TForm4.DBGrid1DbClick(Sender: TObject);
var a:string;
begin
a:=dbgrid1.Fields[0].AsString;
ye(a);
if (query1.Fields[0].AsString=a) then
begin
oku;
end;
end;

procedure TForm4.BitBtn6Click(Sender: TObject);
begin
ye(edit1.Text);
if(query1.Fields[0].asString=edit1.Text) then
begin
oku;
end; end;

procedure TForm4.BitBtn7Click(Sender: TObject);
begin
mnum(edit1.Text);
if (query1.Fields[0].AsString<>edit1.Text) then
begin edit1.Text:=RandomString(5); end
else begin edit1.Text:=RandomString(5); end;
end;
end.

```

Form 5

```

unit Unit5;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, Grids, DBGrids, DB, DBTables,
  XPMenu;

type
  TForm5 = class(TForm)
    DataSource1: TDataSource;
    Query1: TQuery;
    DBGrid1: TDBGrid;
    RadioGroup1: TRadioGroup;

```

```

    BitBtn2: TBitBtn;
    XPMenu1: TXPMenu;
    procedure RadioGroup1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure DBGrid1DbClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form5: TForm5;

implementation
uses Unit1, Unit4;
{$R *.dfm}

procedure TForm5.RadioGroup1Click(Sender: TObject);
begin
    if (radiogroup1.ItemIndex=0) then
    begin
        query1.Active:=false;
        query1.SQL.Clear;
        query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from mdata order
        by mno ASC'
        end;

    if (radiogroup1.ItemIndex=1) then
    begin
        query1.Active:=false;
        query1.SQL.Clear;
        query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from mdata order
        by adi ASC'
        end;

    if (radiogroup1.ItemIndex=2) then
    begin
        query1.Active:=false;
        query1.SQL.Clear;
        query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from mdata order
        by soyadi ASC'
        end;
    if (radiogroup1.ItemIndex=3) then
    begin
        query1.Active:=false;
        query1.SQL.Clear;
        query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from mdata order
        by cinsiyet ASC'
    end;

```



```

end;
query1.Active:=true;
end;

procedure TForm5.FormActivate(Sender: TObject);
begin
query1.Active:=false;
query1.Active:=true;
end;

procedure TForm5.BitBtn2Click(Sender: TObject);
begin
form5.Close;
form1.enabled:=true;
end;

procedure TForm5.DBGrid1DblClick(Sender: TObject);
var a:string;
begin
a:=dbgrid1.Fields[0].AsString;
form4.show;
form4.edit1.text:=a;
form4.bitbtn6.click;
end;
end.

```

Form 6

```

unit Unit6;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, DB, DBTables, StdCtrls, Buttons, ExtCtrls,
  XPMenu;

type
  TForm6 = class(TForm)
    XPMenu1: TXPMenu;
    RadioGroup1: TRadioGroup;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Edit1: TEdit;
    DataSource1: TDataSource;
    Query1: TQuery;
    DBGrid1: TDBGrid;
  procedure FormActivate(Sender: TObject);
  procedure BitBtn1Click(Sender: TObject);
  procedure RadioGroup1Click(Sender: TObject);

```

```

    procedure BitBtn2Click(Sender: TObject);
    procedure DBGrid1DblClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form6: TForm6;

implementation
uses Unit1, Unit4;
{$R *.dfm}
function bulad(a:string):boolean;
begin
    bulad:=false;
    form6.Query1.SQL.Clear;
    form6.Query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from
mdata where adi like'+#39+(form6.edit1.text)+'%'+#39;
    form6.Query1.Open;
    if not form6.Query1.IsEmpty then bulad:=true;
end;
function bulsoy(a:string):boolean;
begin
    bulsoy:=false;
    form6.Query1.SQL.Clear;
    form6.Query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from
mdata where soyadi like'+#39+(form6.edit1.text)+'%'+#39;
    form6.Query1.Open;
    if not form6.Query1.IsEmpty then bulsoy:=true;
end;
function bulmno(a:string):boolean;
begin
    bulmno:=false;
    form6.Query1.SQL.Clear;
    form6.Query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from
mdata where mno like'+#39+(form6.edit1.text)+'%'+#39;
    form6.Query1.Open;
    if not form6.Query1.IsEmpty then bulmno:=true;
end;
function bulc(a:string):boolean;
begin
    bulc:=false;
    form6.Query1.SQL.Clear;
    form6.Query1.SQL.Text:='select mno,adi,soyadi,ceptel,evtel,ktarihi,cinsiyet from
mdata where ktarihi like'+#39+(form6.edit1.text)+'%'+#39;
    form6.Query1.Open;
    if not form6.Query1.IsEmpty then bulc:=true;
end;

```

```
procedure TForm6.FormActivate(Sender: TObject);
begin
query1.Active:=false;
edit1.Visible:=false;
end;
```

```
procedure TForm6.BitBtn1Click(Sender: TObject);
var a:integer;
begin
a:=radiogroup1.ItemIndex;
if a=0 then
bulad(edit1.Text);
if a=1 then
bulsoy(edit1.Text);
if a=2 then
bulmno(edit1.Text);
if a=3 then
bulc(edit1.Text);
end;
```

```
procedure TForm6.RadioGroup1Click(Sender: TObject);
begin
edit1.Visible:=true;
end;
```

```
procedure TForm6.BitBtn2Click(Sender: TObject);
begin
form6.Close;
form1.Enabled:=true;
end;
```

```
procedure TForm6.DBGrid1DblClick(Sender: TObject);
var a:string;
begin
form4.show;
a:=dbgrid1.Fields[0].AsString;
form4.edit1.Text:=a;
form4.bitbtn6.click;
end;
end.
```

Form 7

```
unit Unit7;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls, Mask,
  XPMenu;

type
  TForm7 = class(TForm)
    XPMenu1: TXPMenu;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Panel1: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    DBGrid1: TDBGrid;
    Edit3: TEdit;
    Query1: TQuery;
    DataSource1: TDataSource;
    Query2: TQuery;
    DataSource2: TDataSource;
    Label4: TLabel;
    Label5: TLabel;
    procedure BitBtn3Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure DBGrid1DblClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form7: TForm7;

implementation
uses Unit1;
{$R *.dfm}
procedure kaydet;
begin
```



```

with form7 do begin
query1.Fields[0].asString:=edit1.Text;
query1.Fields[2].asString:=edit2.Text;
query1.Fields[1].asString:=edit3.Text;
end;
end;
procedure oku;
begin
with form7 do begin
edit1.Text:=query1.Fields[0].asString;
edit2.Text:=query1.Fields[1].asString;
edit3.Text:=query1.Fields[2].asString;
end;
end;
procedure temizle;
begin
with form7 do begin
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit1.SetFocus;
end;
end;
function ye(a:string):boolean;
begin
ye:=false;
form7.Query1.First;
while not form7.Query1.eof do
if (a=form7.Query1.Fields[0].asString) then
begin
ye:=true;
exit;
end
else
form7.Query1.Next;
end;

```

```

procedure TForm7.BitBtn3Click(Sender: TObject);
begin
form7.Close;
form1.enabled:=true;
end;

```

```

procedure TForm7.FormActivate(Sender: TObject);
begin
edit3.Text:=datetostr(date());
end;
function ye1(a:string):boolean;
begin

```

```

yel:=false;
form7.Query2.First;
while not form7.Query2.eof do
if (a=form7.Query2.Fields[0].asString) then
begin
yel:=true;
exit;
end
else
form7.Query2.Next;
end;
procedure TForm7.BitBtn1Click(Sender: TObject);
var a:word;
begin
a:=application.messagebox('informations true?','Warning',36);
if (a=IDYES ) then
begin
yel(edit1.Text);
if(query2.fields[0].asString=edit1.Text) then begin
query1.Insert;
kaydet;
query1.Post;
query1.Refresh;
temizle;
edit3.Text:=datetostr(date());
end else begin
query2.insert;
query2.Fields[0].AsString:=edit1.Text;
query2.Post;
query1.Insert;
kaydet;
query1.Post;
query1.Refresh;
temizle;
edit3.Text:=datetostr(date());
end;
end;
end;
procedure TForm7.BitBtn2Click(Sender: TObject);
var a:word;
begin
a:=application.messagebox('informations true?','Warning',36);
if (a=IDYES ) then
begin
query1.edit;
kaydet;
query1.Post;
query1.Refresh;
temizle;
edit3.Text:=datetostr(date());

```

```

end;
end;
procedure TForm7.BitBtn4Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Delete?','Warning',36);
if a=IDYES then
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
query1.Delete;
query1.Refresh;
temizle;
end;
end;
end;
end;

procedure TForm7.DBGrid1DbClick(Sender: TObject);
var a:string;
begin
a:=dbgrid1.Fields[0].AsString;
ye(a);
if (query1.Fields[0].AsString=a) then
begin
oku;
end;
end;
end.

```

Form 8

```

unit Unit8;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Grids, DBGrids, DB, DBTables, XPMenu;

type
  TForm8 = class(TForm)
    BitBtn1: TBitBtn;
    XPMenu1: TXPMenu;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Edit2: TEdit;
    Label4: TLabel;
    Label5: TLabel;
    Edit3: TEdit;

```

```

Edit4: TEdit;
Label6: TLabel;
Edit5: TEdit;
Edit6: TEdit;
Label7: TLabel;
DBGrid1: TDBGrid;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
BitBtn4: TBitBtn;
Label9: TLabel;
Memo1: TMemo;
DataSource1: TDataSource;
Query1: TQuery;
Query2: TQuery;
DataSource2: TDataSource;
Query3: TQuery;
DataSource3: TDataSource;
Label8: TLabel;
Label10: TLabel;
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure DBGrid1DblClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form8: TForm8;

implementation
uses Unit1;
{$R *.dfm}

procedure kaydet;
begin
  with form8 do begin
    query1.Fields[0].AsString:=edit1.text;
    query1.Fields[1].AsString:=edit2.text;
    query1.Fields[2].AsString:=edit3.text;
    query1.Fields[3].AsString:=edit4.text;
    query1.Fields[4].AsString:=edit5.text;
    query1.Fields[5].AsString:=edit6.text;
    query1.Fields[6].AsString:=memo1.text;
  end;
end;

```



```

procedure oku;
begin
with form8 do begin
edit1.text:=query1.Fields[0].AsString;
edit2.text:=query1.Fields[1].AsString;
edit3.text:=query1.Fields[2].AsString;
edit4.text:=query1.Fields[3].AsString;
edit5.text:=query1.Fields[4].AsString;
edit6.text:=query1.Fields[5].AsString;
memo1.text:=query1.Fields[6].AsString;
end;
end;
procedure temizle;
begin
with form8 do begin
edit1.text:="";
edit2.text:="";
edit3.text:="";
edit4.text:="";
edit5.text:="";
edit6.text:="";
memo1.text:="";
edit1.SetFocus;
edit3.Text:=datetostr(date());
end;
end;
function ye(a:string;b:string;c:string):boolean;
begin
ye:=false;
form8.Query1.First;
while not form8.Query1.eof do
if (a=form8.Query1.Fields[0].asstring) and (b=form8.Query1.Fields[1].asstring)and
(c=form8.Query1.Fields[3].asstring)then
begin
ye:=true;
exit;
end
else
form8.Query1.Next;
end;

procedure TForm8.BitBtn1Click(Sender: TObject);
begin
form8.Close;
form1.enabled:=true;
end;

procedure TForm8.BitBtn3Click(Sender: TObject);
var a:word;
begin

```

```

a:=application.MessageBox('informations true?','Warning',36);
if (a=IDYES) then
begin
query1.Insert;
kaydet;
query1.Post;
query3.Insert;
query3.Fields[0].AsString:=edit3.Text;
query3.Fields[1].AsString:=edit5.Text;
query3.Post;
query2.Refresh;
temizle;
end;
end;

```

```

procedure TForm8.BitBtn2Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Delete?','Warning',36);
if (a=IDYES) then
begin
ye(edit1.Text,edit2.Text,edit4.Text);
if (query1.Fields[0].AsString=edit1.text) and (query1.Fields[1].AsString=edit2.text) and
(query1.Fields[3].AsString=edit4.text) then begin
query1.Delete;
query2.Refresh;
temizle;
end;
end;
end;

```

```

procedure TForm8.BitBtn4Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Information true ?','Warning',36);
if (a=IDYES) then
begin
query1.edit;
kaydet;
query1.Post;
query3.edit;
query3.Fields[0].AsString:=edit3.Text;
query3.Fields[1].AsString:=edit5.Text;
query3.Post;
query2.Refresh;
temizle;
end;
end;

```

```

procedure TForm8.DBGrid1DbClick(Sender: TObject);

```

```

var a,b,c:string;
begin
a:=dbgrid1.Fields[0].AsString;
b:=dbgrid1.Fields[1].AsString;
c:=dbgrid1.Fields[4].AsString;
ye(a,b,c);
if (query1.Fields[0].AsString=a) and (query1.Fields[1].AsString=b) and
(query1.Fields[3].AsString=c) then begin
oku;
end;
end;

procedure TForm8.FormActivate(Sender: TObject);
begin
edit3.Text:=datetostr(date());
end;
end.

```

Form 9

```

unit Unit9;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, XPMenu, DB, DBTables, Grids, DBGrids, StdCtrls, Buttons;

type
  TForm9 = class(TForm)
    XPMenu1: TXPMenu;
    BitBtn4: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Memo1: TMemo;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DataSource1: TDataSource;

```

```

Query2: TQuery;
DataSource2: TDataSource;
Query3: TQuery;
DataSource3: TDataSource;
DataSource4: TDataSource;
Query4: TQuery;
Label8: TLabel;
Label9: TLabel;
procedure FormActivate(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure DBGrid1DblClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form9: TForm9;

implementation
uses Unit1;
{$R *.dfm}

procedure kaydet;
begin
  with form9 do begin
    query1.Fields[0].AsString:=edit1.Text;
    query1.Fields[1].AsString:=edit2.Text;
    query1.Fields[2].AsString:=memo1.Text;
    query1.Fields[3].AsString:=edit3.Text;
    query1.Fields[4].AsString:=edit4.Text;
    query1.Fields[5].AsString:=edit5.Text;
  end;
end;

procedure oku;
begin
  with form9 do begin
    edit1.Text:=query1.Fields[0].AsString;
    edit2.Text:=query1.Fields[1].AsString;
    memo1.Text:=query1.Fields[2].AsString;
    edit3.Text:=query1.Fields[3].AsString;
    edit4.Text:=query1.Fields[4].AsString;
    edit5.Text:=query1.Fields[5].AsString;
  end;
end;

procedure temizle;

```



```

begin
with form9 do begin
edit1.Text:="";
edit2.Text:="";
memo1.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
edit1.SetFocus;
edit4.Text:=datetostr(date());
end;
end;
function ye(a:string;b:string;c:string):boolean;
begin
ye:=false;
form9.Query1.First;
while not form9.Query1.eof do
if (a=form9.Query1.Fields[0].asString) and (b=form9.Query1.Fields[1].asString)and
(c=form9.Query1.Fields[3].asString)then
begin
ye:=true;
exit;
end
else
form9.Query1.Next;
end;

```

```

procedure TForm9.FormActivate(Sender: TObject);
begin
edit4.Text:=datetostr(date());
end;

```

```

procedure TForm9.BitBtn4Click(Sender: TObject);
begin
form9.Close;
form1.enabled:=true;
end;

```

```

procedure TForm9.BitBtn1Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Informations true ?','Warning',36);
if (a=IDYES ) then
begin
query1.Insert;
kaydet;
query1.Post;
query3.Insert;
query3.Fields[0].AsString:=edit4.Text;

```

```

query3.Fields[1].AsString:=edit5.Text;
query3.Post;
query2.Refresh;
temizle;
end;
end;
procedure TForm9.BitBtn2Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Informations true ?','Warning',36);
if (a=IDYES ) then
begin
query1.edit;
kaydet;
query1.Post;
query3.edit;
query3.Fields[0].AsString:=edit4.Text;
query3.Fields[1].AsString:=edit5.Text;
query3.Post;
query2.Refresh;
temizle;
end;
end;

```

```

procedure TForm9.BitBtn3Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Delete?','Warning',36);
if (a=IDYES ) then
begin
ye(edit1.Text,edit2.Text,edit3.Text);
if (query1.Fields[0].AsString=edit1.Text) and (query1.Fields[1].AsString=edit2.Text)
and (query1.Fields[3].AsString=edit3.Text) then
begin
query1.Delete;
query2.Refresh;
temizle;
end;
end;
end;

```

```

procedure TForm9.DBGrid1DbClick(Sender: TObject);
var a,b,c:string;
begin
a:=dbgrid1.Fields[0].AsString;
b:=dbgrid1.Fields[1].AsString;
c:=dbgrid1.Fields[4].AsString;

ye(a,b,c);

```

```

if (query1.Fields[0].AsString=a) and (query1.Fields[1].AsString=b) and
(query1.Fields[3].AsString=c) then
begin
oku;
end;
end;
end.

```

Form 10

```

unit Unit10;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, DB, DBTables, Grids, DBGrids, XPMenu,
  ExtCtrls;

type
  TForm10 = class(TForm)
    BitBtn1: TBitBtn;
    XPMenu1: TXPMenu;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DataSource1: TDataSource;
    Panel1: TPanel;
    RadioGroup1: TRadioGroup;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    Query2: TQuery;
    Query3: TQuery;
    procedure BitBtn1Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;

implementation
uses Unit1;
{$R *.dfm}
procedure yel;
begin
form10.Query1.SQL.Clear;
form10.Query1.SQL.Text:='select * from kasacikti';
form10.Query1.Open;
end;

```

```

procedure ye2;
begin
form10.Query1.SQL.Clear;
form10.Query1.SQL.Text:='select * from kasagirdi';
form10.Query1.Open;
end;

procedure TForm10.BitBtn1Click(Sender: TObject);
begin
form1.enabled:=true;
form10.Close;
end;

procedure TForm10.RadioGroup1Click(Sender: TObject);
begin
if radiogroup1.ItemIndex=0 then
ye2;
if radiogroup1.ItemIndex=1 then
ye1;
end;
end.

```

Form 11

```

unit Unit11;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, XPMenu;

type
  TForm11 = class(TForm)
    XPMenu1: TXPMenu;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form11: TForm11;
implementation
uses Unit1;
{$R *.dfm}

```



```

procedure TForm11.BitBtn1Click(Sender: TObject);
begin
winexec('yedeklehdd.bat',0);
application.MessageBox('Saved C:\YEDEK.', 'Warning', 48);
end;

procedure TForm11.BitBtn2Click(Sender: TObject);
begin
application.MessageBox('Please insert Floppy Disk', 'Warning', 48);
winexec('yedeklefdh.bat',0);
end;

procedure TForm11.BitBtn3Click(Sender: TObject);
begin
form1.Enabled:=true;
form11.Close;
end;
end.

```

Form 12

```

unit Unit12;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, XPMenu;

type
  TForm12 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    XPMenu1: TXPMenu;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form12: TForm12;
implementation
uses Unit1;
{$R *.dfm}

procedure TForm12.Button1Click(Sender: TObject);
begin
form12.Close;
form1.enabled:=true;
end;
end.

```

Form 14

```
unit Unit14;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Grids, DBGrids, ExtCtrls, XPMenu, DB,
  DBTables;

type
  TForm14 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    Edit3: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    Edit4: TEdit;
    Label5: TLabel;
    Memo1: TMemo;
    Label10: TLabel;
    Edit9: TEdit;
    BitBtn1: TBitBtn;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    XPMenu1: TXPMenu;
    Query1: TQuery;
    DataSource1: TDataSource;
    Label11: TLabel;
    Edit10: TEdit;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    DataSource2: TDataSource;
    Query2: TQuery;
    procedure BitBtn5Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure DBGrid1Db1Click(Sender: TObject);
```

```

    procedure BitBtn6Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form14: TForm14;
implementation
uses Unit1;
{$R *.dfm}
procedure kaydet;
begin
    with form14 do begin
        query1.Fields[0].AsString:=edit1.Text;
        query1.Fields[1].AsString:=edit2.Text;
        query1.Fields[2].AsString:=edit3.Text;
        query1.Fields[3].AsString:=edit4.Text;
        query1.Fields[4].AsString:=memo1.Text;
        query1.Fields[5].AsString:=edit9.Text;
        query1.Fields[6].AsString:=edit10.Text;
    end;
end;
procedure oku;
begin
    with form14 do begin
        edit1.Text:=query1.Fields[0].AsString;
        edit2.Text:=query1.Fields[1].AsString;
        edit3.Text:=query1.Fields[2].AsString;
        edit4.Text:=query1.Fields[3].AsString;
        memo1.Text:=query1.Fields[4].AsString;
        edit9.Text:=query1.Fields[5].AsString;
        edit10.Text:=query1.Fields[6].AsString;
    end;
end;
procedure temizle;
begin
    with form14 do begin
        edit1.Text:="";
        edit2.Text:="";
        edit3.Text:="";
        edit4.Text:="";
        memo1.Text:="";
        edit9.Text:="";
        edit1.SetFocus;
    end;
end;
function ye(a:string):boolean;
begin

```

```

ye:=false;
form14.Query2.First;
while not form14.Query2.eof do
if (a=form14.Query1.Fields[0].asString) then
begin
ye:=true;
exit;
end
else
form14.Query2.Next;
end;
procedure TForm14.BitBtn5Click(Sender: TObject);
begin
form14.close;
form1.enabled:=true;
end;
procedure TForm14.FormActivate(Sender: TObject);
begin
edit10.Text:=datetostr(date());
end;

procedure TForm14.BitBtn2Click(Sender: TObject);
var a:word;
begin
bitbtn1.Click;
a:=application.MessageBox('INFORMATIONS TRUE ?','Warning',36);
if (a=IDYES) then
begin
ye(edit1.Text);
if (query2.Fields[0].AsString<>edit1.Text) then begin
query2.Insert;
query2.Fields[0].AsString:=edit1.Text;
query2.Post;
query1.Insert;
kaydet;
query1.Post;
query1.Refresh;
temizle;
end else begin
query1.Insert;
kaydet;
query1.Post;
query1.Refresh;
temizle;
end;
end;
end;
end;

```



```

procedure TForm14.BitBtn1Click(Sender: TObject);
var
a:integer;
d,c:integer;
begin
a:=0;
c:=0;
d:=0;
a:=strtoint(edit2.Text);
c:=strtoint(edit3.Text);
d:=a*c;
edit9.Text:=inttostr(d);
end;

```

```

procedure TForm14.BitBtn3Click(Sender: TObject);
var a:word;
begin
bitbtn1.Click;
a:=application.MessageBox('INFORMATIONS TRUE ?','Warning',36);
if (a=IDYES) then
begin
query1.edit;
kaydet;
query1.Post;
query1.Refresh;
temizle;
end;
end;

```

```

procedure TForm14.BitBtn4Click(Sender: TObject);
begin
ye(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
query1.Delete;
query1.Refresh;
temizle;
end;
end;

```

```

procedure TForm14.DBGrid1DblClick(Sender: TObject);
var a:string;
begin
a:=dbgrid1.Fields[0].AsString;
ye(a);
if (query1.Fields[0].AsString=a) then begin
oku;
end;
end;

```

```

procedure TForm14.BitBtn6Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Do You Want to Clear ?','Warning',36);
if a=IDYES then
temizle;
end;
end.

```

Form 15

```

unit Unit15;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, Grids, DBGrids, StdCtrls, ExtCtrls, Buttons,
  XPMenu;

type
  TForm15 = class(TForm)
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    ComboBox1: TComboBox;
    CheckBox1: TCheckBox;
    Label1: TLabel;
    Edit1: TEdit;
    BitBtn2: TBitBtn;
    Query2: TQuery;
    DataSource2: TDataSource;
    XPMenu1: TXPMenu;
    Panel1: TPanel;
    RadioGroup1: TRadioGroup;
    procedure CheckBox1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form15: TForm15;

implementation
uses Unit1;

```

```

{$R *.dfm}
procedure TForm15.CheckBox1Click(Sender: TObject);
var a:string;
begin
if (checkbox1.Checked=true) then
begin
combobox1.Enabled:=true;
combobox1.Items.Clear;
label1.Visible:=true;
edit1.Visible:=true;
query1.First;
while (not query2.Eof) do begin
combobox1.Items.Add(query2.Fields[0].AsString);
query2.Next;
end;
end else begin
combobox1.Enabled:=false;
label1.Visible:=false;
edit1.Visible:=false;
end;end;

procedure TForm15.BitBtn2Click(Sender: TObject);
begin
checkbox1.Checked:=false;
form1.enabled:=true;
form15.Close;
end;
function ye(a:string):boolean;
begin
ye:=false;
form15.Query1.First;
while not form15.Query1.eof do
if (a=form15.Query1.Fields[0].asString) then
begin
ye:=true;
exit;
end
else
form15.Query1.Next;
end;
function bulad(a:string):boolean;
begin
bulad:=false;
form15.Query1.SQL.Clear;
form15.Query1.SQL.Text:='select * from stokcam where camcinsi
='+#39+(form15.ComboBox1.Text)+#39;
form15.Query1.Open;
if not form15.Query1.IsEmpty then bulad:=true;
end;

```

```

procedure TForm15.ComboBox1Change(Sender: TObject);
var a,b,c:integer;
begin
bulad(combobox1.Text);
query1.First;
while(not query1.Eof) do begin
a:=query1.Fields[2].AsInteger;
b:=b+a;
query1.Next;
end;
edit1.Text:=inttostr(b);
end;

```

```

procedure TForm15.FormActivate(Sender: TObject);
begin
query1.Refresh;
end;

```

```

procedure TForm15.RadioGroup1Click(Sender: TObject);
begin
if (radiogroup1.ItemIndex=0) then
begin
query1.Active:=false;
query1.SQL.Clear;
query1.SQL.Text:='select * from stokcam order by tarih ASC'
end;
if (radiogroup1.ItemIndex=1) then
begin
query1.Active:=false;
query1.SQL.Clear;
query1.SQL.Text:='select * from stokcam order by bfiyat ASC'
end;
if (radiogroup1.ItemIndex=2) then
begin
query1.Active:=false;
query1.SQL.Clear;
query1.SQL.Text:='select * from stokcam order by gelsirket ASC'
end;
query1.Active:=true;
end;
end.

```


Form 16

```
unit Unit16;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, Buttons, DB, DBTables, XPMenu, Menus;

type
  TForm16 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    DBGrid1: TDBGrid;
    BitBtn4: TBitBtn;
    DataSource1: TDataSource;
    Query1: TQuery;
    XPMenu1: TXPMenu;
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure DBGrid1DblClick(Sender: TObject);

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

var
  Form16: TForm16;
implementation
uses Unit1;
{$R *.dfm}

procedure TForm16.BitBtn4Click(Sender: TObject);
begin
  form1.enabled:=true;
  form16.close;
end;
function ye(a:string):boolean;
begin
  ye:=false;
  form16.Query1.First;
  while not form16.Query1.eof do
    if (a=form16.Query1.Fields[0].asString) then
```

```

begin
ye:=true;
exit;
end
else
form16.Query1.Next;
end;

```

```

procedure TForm16.BitBtn1Click(Sender: TObject);
begin
ye(edit1.text);
if (query1.Fields[0].AsString<>edit1.Text) then begin
query1.Insert;
query1.Fields[0].AsString:=edit1.Text;
query1.Post;
edit1.Text:="";
edit1.SetFocus;
query1.Refresh;
end else application.MessageBox('This Trademark was using!!','Warning',48);
end;

```

```

procedure TForm16.BitBtn2Click(Sender: TObject);
begin
ye(edit1.text);
if (query1.Fields[0].AsString<>edit1.Text) then begin
query1.edit;
query1.Fields[0].AsString:=edit1.Text;
query1.Post;
edit1.Text:="";
edit1.SetFocus;
query1.Refresh;
end else application.MessageBox('This trademark was using !!','Warning',48);
end;

```

```

procedure TForm16.BitBtn3Click(Sender: TObject);
begin
ye(edit1.text);
if (query1.Fields[0].AsString=edit1.Text) then begin
query1.Delete;
end;
end;

```

```

procedure TForm16.DBGrid1DbClick(Sender: TObject);
var a:string;
begin
a:=dbgrid1.fields[0].asString;
ye(a);
if (query1.Fields[0].AsString=a) then begin
edit1.Text:=query1.Fields[0].AsString;
end; end; end.

```

Form 17

```
unit Unit17;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DB, DBTables, Grids, DBGrids, XPMenu, Buttons,
  ExtCtrls;

type
  TForm17 = class(TForm)
    Label1: TLabel;
    ComboBox1: TComboBox;
    Label2: TLabel;
    Edit1: TEdit;
    Label3: TLabel;
    Edit2: TEdit;
    Label4: TLabel;
    Edit3: TEdit;
    Label5: TLabel;
    Edit4: TEdit;
    Label6: TLabel;
    Memo1: TMemo;
    Label7: TLabel;
    Edit5: TEdit;
    DataSource1: TDataSource;
    Query1: TQuery;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Edit6: TEdit;
    DataSource2: TDataSource;
    Query2: TQuery;
    Panel1: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    XPMenu1: TXPMenu;
    Query3: TQuery;
    DataSource3: TDataSource;
    BitBtn6: TBitBtn;
    DBGrid1: TDBGrid;
    procedure FormActivate(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
```

```

    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure DBGrid1DblClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form17: TForm17;

implementation
uses Unit1;
{$R *.dfm}
procedure kaydet;
begin
    with form17 do begin
        query1.Fields[0].asString:=combobox1.Text;
        query1.Fields[1].asString:=edit1.Text;
        query1.Fields[2].asString:=edit2.Text;
        query1.Fields[3].asString:=edit3.Text;
        query1.Fields[4].asString:=edit4.Text;
        query1.Fields[5].asString:=memo1.Text;
        query1.Fields[7].asString:=edit5.Text;
        query1.Fields[6].asString:=edit6.Text;
    end;
end;
procedure oku;
begin
    with form17 do begin
        combobox1.Text:=query1.Fields[0].asString;
        edit1.Text:=query1.Fields[1].asString;
        edit2.Text:=query1.Fields[2].asString;
        edit3.Text:=query1.Fields[3].asString;
        edit4.Text:=query1.Fields[4].asString;
        memo1.Text:=query1.Fields[5].asString;
        edit5.Text:=query1.Fields[7].asString;
        edit6.Text:=query1.Fields[6].asString;
    end;
end;
procedure temizle;
begin
    with form17 do begin
        combobox1.Text:="";
        edit1.Text:="";
        edit2.Text:="";

```



```

edit3.Text:="";
edit4.Text:="";
memo1.Text:="";
edit6.Text:="";
edit1.SetFocus;
end;
end;
function bul(a:string;b:string):boolean;
begin
bul:=false;
form17.Query1.First;
while not form17.Query1.eof do
if (a=form1.Query1.Fields[0].asString)and (b=form1.Query1.Fields[1].asString)then
begin
bul:=true;
exit;
end
else
form17.Query1.Next;
end;
function bull(a:string):boolean;
begin
bull:=false;
form17.Query3.First;
while not form17.Query3.eof do
if (a=form17.Query3.Fields[0].asString)then
begin
bull:=true;
exit;
end
else
form17.Query1.Next;
end;

procedure TForm17.FormActivate(Sender: TObject);
begin
query1.Refresh;
edit5.Text:=datetostr(date());
query2.Refresh;
query2.First;
combobox1.Items.Clear;
while(not query2.Eof) do begin
combobox1.Items.Add(query2.Fields[0].AsString);
query2.Next;
end;
end;

```

```

procedure TForm17.BitBtn5Click(Sender: TObject);
begin
form1.enabled:=true;
form17.Close;
end;

procedure TForm17.BitBtn1Click(Sender: TObject);
var a:word;
begin
bitbtn6.Click;
a:=application.MessageBox('Informations True?','Warning',36);
if a=IDYES then
begin
query3.Insert;
query3.Fields[0].AsString:=combobox1.Text;
query3.Fields[1].AsString:=edit1.Text;
query3.Post;
query1.Insert;
kaydet;
query1.Post;
temizle;
query1.Refresh;
end;
end;

procedure TForm17.BitBtn2Click(Sender: TObject);
var a:word;
begin
bitbtn6.Click;
a:=application.MessageBox('Information true?','Warning',36);
if a=IDYES then
begin
query3.edit;
query3.Fields[0].AsString:=combobox1.Text;
query3.Fields[1].AsString:=edit1.Text;
query3.Post;
query1.edit;
kaydet;
query1.Post;
temizle;
query1.Refresh;
end;
end;

procedure TForm17.BitBtn3Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Delete?','Warning',36);
if a=IDYES then
begin

```

```

bul(combobox1.Text,edit1.Text);
if (query1.fields[0].asString=combobox1.Text) and
(query1.Fields[1].AsString=edit1.Text) then
begin
query1.Delete;
query1.Refresh;
temizle;
end;
end;
end;

```

```

procedure TForm17.BitBtn4Click(Sender: TObject);
var a:word;
begin
a:=application.MessageBox('Are You Sure ?','Warning',36);
if a=IDYES then
begin
temizle;
end;
end;

```

```

procedure TForm17.ComboBox1Change(Sender: TObject);
begin
edit1.SetFocus;
end;

```

```

procedure TForm17.BitBtn6Click(Sender: TObject);
var a,b,c:integer;
begin
a:=strtoint(edit2.Text);
b:=strtoint(edit3.Text);
c:=a*b;
edit6.Text:=inttostr(c);
end;

```

```

procedure TForm17.DBGrid1DbClick(Sender: TObject);
var a,b:string;
begin
a:=dbgrid1.Fields[0].AsString;
b:=dbgrid1.Fields[1].AsString;
bul(a,b);
if (query1.fields[0].asString=a) and (query1.Fields[1].AsString=b) then
oku;
end;
end.

```

Form 18

```
unit Unit18;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Grids, DBGrids, DB, DBTables, Buttons,
  XPMenu;

type
  TForm18 = class(TForm)
    Panel1: TPanel;
    RadioGroup1: TRadioGroup;
    DataSource1: TDataSource;
    Query1: TQuery;
    DBGrid1: TDBGrid;
    CheckBox1: TCheckBox;
    ComboBox1: TComboBox;
    DataSource2: TDataSource;
    Query2: TQuery;
    BitBtn2: TBitBtn;
    XPMenu1: TXPMenu;
    Label1: TLabel;
    Edit1: TEdit;
    procedure CheckBox1Click(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form18: TForm18;

implementation
uses Unit1;
{$R *.dfm}

procedure TForm18.CheckBox1Click(Sender: TObject);
begin
  if (checkbox1.Checked=true) then
  begin
    query2.First;
    combobox1.Enabled:=true;
    LABEL1.visible:=true;
    edit1.Visible:=true;
```



```

combobox1.Items.Clear;
while(not query2.Eof) do begin
combobox1.Items.Add(query2.Fields[0].AsString);
query2.Next;
end;
end else begin
combobox1.Text:="";
edit1.text:="";
combobox1.Enabled:=false;
label1.visible:=false;
edit1.Visible:=false;
end;
end;
function bulad(a:string):boolean;
begin
bulad:=false;
form18.Query1.SQL.Clear;
form18.Query1.SQL.Text:='select * from stokc where cmarka
='+#39+(form18.combobox1.text)+#39;
form18.Query1.Open;
if not form18.Query1.IsEmpty then bulad:=true;
end;
procedure TForm18.ComboBox1Change(Sender: TObject);
var a,b:integer;
begin
bulad(combobox1.Text);
while(not query1.Eof) do begin
a:=strtoint(query1.Fields[3].AsString);
b:=b+a;
query1.Next;
end;
edit1.Text:=inttostr(b);
end;

procedure TForm18.Radiogroup1Click(Sender: TObject);
begin
if (radiogroup1.ItemIndex=0) then
begin
query1.Active:=false;
query1.SQL.Clear;
query1.SQL.Text:='select * from stokc order by tarih ASC'
end;

if (radiogroup1.ItemIndex=1) then
begin
query1.Active:=false;
query1.SQL.Clear;
query1.SQL.Text:='select * from stokc order by bfiyat ASC'
end;

```

```

if (radiogroup1.ItemIndex=2) then
begin
query1.Active:=false;
query1.SQL.Clear;
query1.SQL.Text:='select * from stokc order by gsirket ASC'
end;
query1.Active:=true;
end;

```

```

procedure TForm18.BitBtn2Click(Sender: TObject);
begin
combobox1.Text:="";
edit1.text:="";
combobox1.Enabled:=false;
label1.visible:=false;
edit1.Visible:=false;
form18.Close;
form1.enabled:=true;
end;
end.

```

Form 19

```

unit Unit19;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, jpeg;

type
  TForm19 = class(TForm)
    Image1: TImage;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form19: TForm19;

implementation
uses Unit1;
{$R *.dfm}
end.

```

APPENDIX 2

Table of the Database

kullanicı	User Name	Password	Qestion	Answer	Date	Authority
1	enis	enis	who?	enis	18.10.2006	ADMIN
2	ed	ed	who are you?	ed	05.12.2006	USER
3						

Figure 1. User's Table

Kullanıcı.db for users. It has include the username, password, secret question , answer, date and authorization datas. For entering the program, needed this database table.

mdata	Customer Number	Name	Surname	Mobile phone	Home phone	Address	Sex	Register date
1	1	FATİH	EKİZ	0(533)645-6432	0() .	<BLOB Memo>	MALE	18,12,2006
2	2	OZAN	ALI	0(553)533-4343	0() .	<BLOB Memo>	MALE	01,11,2006
3	3	EN	İS	0(533)645-7475	0(033)283-8948	<BLOB Memo>	MALE	18,10,2006

Figure2. Customer Data Table

Mdata.db for customer's data. It contain the customer number, name, surname, mobile phone, address, sex, and register date.

yetki	User Name	State
1	enis	1
2	efe	2
3	fatih	2
4	deniz	1

Figure 3 Authority table

Yetki.db create for authorization. It has include username and state.
If state is '1' user is ADMIN, if state '2' user is USER.

kurum	Company name
1	BAGKUR
2	ADANA OPTIK
3	SSK

Figure 4 Company Table

Kurum.db create for company name.

kurumkaydi	Company Name	Date	Amount
1	ADANA OPTIK	18.10.2006	1200
2	ORIMEX	07.01.2007	200
3	XYZ FIRM	15.01.2007	3500

Figure 5 company register table

Kurumkaydi.db create for company register which has include the company name, register date, and take a cash amount.

marka	Trademark
1	cermar
2	abcdxyz
3	topten

Figure 6 Trademark glasses

Marka.db create for register of glass trademark

ka:	Name	Surname	Address	Reason	Date	Price
1	OZAN	EKIZ	<BLOB Memo>	mount installment	01.11.2006	456
2	ENIS	KASIMOGLU	<BLOB Memo>	mount installment	18.10.2006	1000

Figure 7 Income table

Kasagirdi.db use for revenue. It has include the name, surname, address, date, take cash amount, and reason which why the take money this person.

ka:	Name	Surname	Date	Reason	Price	Company name	Address
1	OZAN	EKIZ	01.11.2006	QWERTY	53	OYAKSA	<BLOB Memo>
2	ENIS	KASIMOGLU	18.10.2006	KYZABCD	2138	ORIMEX	<BLOB Memo>
3	FATIH	SALI	12.01.2007	ASDFGH	234	ADEBA	<BLOB Memo>

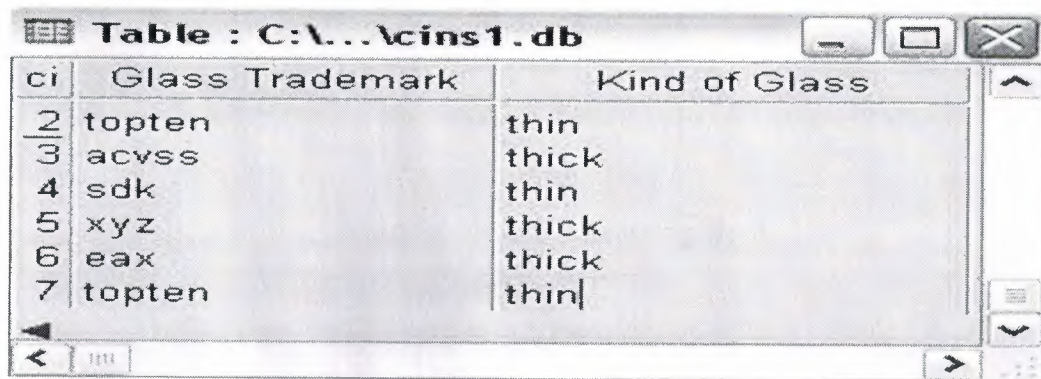
Figure 8 Expenditure table

Kasacikti.db create for expenditure. Contain the name , surname, date, address, company name, give money amount, and reason which why give the money this person and company.

stokc	Glass trademark	Kind of glass	Price	Grain	Company	Address	General total	Date
1	EAX	THIN	2000	50	GEWWR	<BLOB Memo>	100000	28.05.2006
2	TOPTEN	THIN	1500	10	ASDFF	<BLOB Memo>	15000	28.05.2006
3	CERMAR	THICK	20000	10	ADANA OPTIK	<BLOB Memo>	200000	28.05.2006

Figure 9 stock of glass

Stock.db create for stock amount. Contain the glass trademark, glass kind, price, address, genel total, and date.



ci	Glass Trademark	Kind of Glass
2	topten	thin
3	acvss	thick
4	sdk	thin
5	xyz	thick
6	eax	thick
7	topten	thin

Figure10.Kind of Glass

Create the cins1.db for kind of glass and glass trademark informations.

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] <http://www.w3schools.com/sql/>
- [7] <http://www.1keydata.com/sql>