NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

VETERINERIAN APPLICATION PROGRAM WITH DELPHI

Graduation Project
COM400

Student: Ahmet KAYABAŞ (20021329)

Supervisor: Mr. Elburus IMANOV

Lefkoşa-2007

TABLE OF CONTENT

ACKNOWLEDGMENT

When people start a new work they get excited.Because who do not know any thing about the future of work.When a time passed human becomes familiar for this work.Afterthat may be borred,maybe want to leave this work.That may be true maybe false.It changes from people to people.But I believe that the important thing in the life do not leave such who should embrace very tightly.When we get this it makes us happy.

In the life what is important for you.Business? Money? Science? Power? Family? Love? Humanity? or purpose of existence? In my opinion first of all aim of existence comes.Rest of all things involved in aim of existence.After that comes Love.The world exists of love.With love person gets power, gains working perseverence .

Well in this project I gained perseverence from ***Allah*** *and from my fiancee.I am happy to complete the task which I had given with blessing of* ***Allah*** *and also I am grateful to my fiancee and all the people in my life who have supported me, advised me. They all the time helped and encouraged me to follow my dreams and ambitions.*

For intellectual support, encouragement I want to thank to my supervisor ***Mr. Elburus Imanov*** *who made this project contributions.*

And thank ***my dearest parents*** *who supported me to continue beyond my undergraduate studies, and also many thanks to my dear familiy who brought me till such meaning days.*

To all my friends, especially ***M.Fethullah Akatay, Selman Kayabaş, Metin Yenigün , Kadir Bekiroğlu and My dear fiancee*** *for sharing wonderful moments, advice, and for making me feel at home and in life. And above, I thank God for giving me stamina and courage to achieve my objectives.*

AHMET KAYABAŞ

ABSTRACT

In the world not only human life is important.In the same time other entity lives with us.We are not alone on the earth.Animals share life with us.Ilnesses are not only for human.In the same time whole alive interested with illnesses.How Doctor is important for us like Veterinerian is important for animals.Todays Doctors use application program.Because of to keep knowledge of patient, to facility diagnosis of illness, to reach background of patient efficiently and easly.

Well Veterinerian application program is important like the program that is used human health.Also much more important then others.Because animal can not keep the illnesses knowledge.And also papers of the animal can lost.

This project has as its goal to develop software, processing information about activities of a veterinerian application software. Software developed in this project like not only for animal.In the same time for staff and for owner of the animal.All records keep in the other Database program.It acts easly and fast access.Veterinerian can keep all records in the program as concentment.

INTRODUCTION

Since human created by the powerful Allah, Human wonder everything.Well who tried to satisfy wonder.Such humanity came to nowadays as develop.Todays everyone says technology perfect developed.Yes that is right.By means of technology all process gained velocity.This development acts to spend time to the people.

Technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.The automation is also became a part of our lives. The people operate with automation systems in everywhere.

Veterinerian Application project which is my project.In this software veterinerian can keep animal knowledge, patient background knowledge of the animal, owner of the animal knowledge.With this software veterinerian will make record process easily and safetly.

In Software there are five types user.They can access to only their task process.In the same time in the program veterinerian can get obligation as daily.The software can be used at every animal clinic easly.

CHAPTER 1

DELPHI

1.1 INTRODUCTION TO DELPHI

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

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

It is a method for predicting future events.

It is a method for generating a quick consensus by a group.

It is the use of a survey to collect information.

It is the use of anonymity on the part of the participants.

It is the use of voting to reduce the need for long discussions.

It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing detailed critical examination and discussion, not at forcing a quick

compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective:

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

Additional opportunity has been added by the introduction of Computer Mediated Communication Systems (Hiltz and Turoff, 1978; Rice and Associates, 1984; Turoff, 1989; Turoff, 1991). These are computer systems that support group communications in either a synchronous (Group Decision Support Systems, Desanctis et. al., 1987) or an asynchronous manner (Computer Conferencing). Techniques that were developed and refined in the evolution of the Delphi Method (e.g. anonymity, voting) have been incorporated as basic facilities or tools in many of these computer based systems. As a result, any of these systems can be used to carry out some form of a Delphi process or Nominal Group Technique (Delbecq, et. al., 1975).

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

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

1.2 WHAT IS DELPHI?

Delphi is an object oriented, 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 What's more, 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 tools.

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!

1.3 WHAT KIND OF PROGRAMMING CAN YOU DO WITH DELPHI?

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

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

Some projects to which Delphi is suited:

- Simple, single user database applications

- Intermediate multi-user database applications

- Large scale multi-tier, multi-user database applications

- Internet applications

- Graphics Applications

- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools

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.

1.4 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 quitcapable ofthis.

1.5 SOME KNOWLEDGE ABOUT DELPHI

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

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

For the purposes of this series I will be using Delphi 7. There are more recent versions available (2005 and 2006) however Delphi 7 should be available inexpensively compared to the new versions which will set you back a lot of money. Delphi 7 will more than likely be available in a magazine for free.

1.5.2 Example: Try First Delphi Program

First thing is first, fire up your copy of Delphi and open the Project > Options menu. To compile a console application you need to change a setting on the Linker tab called 'Generate console application', check the box and click OK. Now select File > Close All if anything is already loaded. Then select File > New > Other > Console Application.

Notice the first line refers to the keyword program. You can rename this to HelloWorld. You can also remove the commented portion enclosed in curly brackets. The uses keyword allows you to list all units that you want to use in the program. At the moment just leave it as it is, SysUtils is all we need.

Your unit should now look like this:

Delphi Code:

```
program HelloWorld;

  {$APPTYPE CONSOLE}

 uses

  SysUtils;

 begin

 end.
```

Now what we have just done is written a program, it currently doesn't do a thing however. Hit the run button and see the result. Now wasn't that completely worthless.

Luckily this isn't the end of the article so we'll actually have a worthwhile program at the end of it. All we need to do is insert some code in the main procedure we have just made.

Every good programmer's first program was 'Hello World' and you'll be no exception. All we need to do is use the WriteLn procedure to write 'Hello World!' to the console, simple.Notice the semicolon at the end of the line, at the end of any statement you need to add a semicolon. Run the program and see the results…

Now I don't know about you but I saw hello world flash up and go away in a second, if you didn't write the program you wouldn't even know what it said. To solve this problem we need to tell the program to leave the console open until the user is ready to close it. We can use ReadLn for this which reads the users input from the console.

Delphi Code:

```
program HelloWorld;

 {$APPTYPE CONSOLE}

 uses

   SysUtils;

 begin

   WriteLn('Hello World!' + #13#10 + #13#10 +

     'Press RETURN to end...');

   ReadLn;

 end.
```

I have added a few extra things into the 'Hello World' string so the user knows what to do to end the program as it could be a bit confusing. '#13#10' is to insert a carriage

return as 13 and 10 are the ASCII codes for a carriage return followed by a new line feed. ASCII can be inserted in this way into strings.

1.5.2 Delphi Style

Coding style, the way you format your code and the way in which you present it on the page.At the end of the day who cares about my style, I can read it, and Delphi strips all the spaces out of it and doesn't care if I indent. Why waste my time?

Neatly present code which conforms to the accepted standards not only makes your code much easier for you to read and debug but also but any one else who might read your code to help you, or learn from you can do so with ease. After all which code is easier to follow, example 1 or 2?

Delphi Code:

```
// Example 1

procedure xyz();

var

x,y,z,a:integer;

begin

x:=1;y:=2;

for z:=x to y do begin

a:=power(z,y);

showmessage(inttostr(a));

end;
```

```
end;
```

Delphi Code:

```
// Example 2
procedure XYZ();
var
X,Y,Z,A: Integer;
begin
  X := 1;
  Y := 2;
  for Z := X to Y do
  begin
        A := Power(Z, Y);
        ShowMessage(IntToStr(A));
  end; // for end
end; // procedure end
```

Design patterns are frequently recurring structures and relationships in object-oriented design. Getting to know them can help you design better, more reusable code and also help you learn to design more complex systems.

Much of the ground-breaking work on design patterns was presented in the book Design Patterns: Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides. You might also have heard of the authors referred to as "the Gang of Four". If you haven't read this book before and you're designing objects, it's an excellent

primer to help structure your design. To get the most out of these examples, I recommend reading the book as well.

Another good source of pattern concepts is the book Object Models: Strategies, Patterns and Applications by Peter Coad. Coad's examples are more business oriented and he emphasises learning strategies to identify patterns in your own work.

1.6 HOW DELPHI HELPS YOU DEFINE PATTERNS

Delphi implements a fully object-oriented language with many practical refinements that simplify development.

The most important class attributes from a pattern perspective are the basic inheritance of classes; virtual and abstract methods; and use of protected and public scope. These give you the tools to create patterns that can be reused and extended, and let you isolate varying functionality from base attributes that are unchanging.

Delphi is a great example of an extensible application, through its component architecture, IDE interfaces and tool interfaces. These interfaces define many virtual and abstract constructors and operations.

1.6.1 Delphi Examples of Design Patterns

I should note from the outset, there may be alternative or better ways to implement these patterns and I welcome your suggestions on ways to improve the design. The following patterns from the book *Design Patterns* are discussed and illustrated in Delphi to give you a starting point for implementing your own Delphi patterns.

Pattern Name	*Definition*
Singleton	"Ensure a class has only one instance, and provide a global point of access to it."
Adapter	"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't

	otherwise because of incompatible interfaces."
Template Method	"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."
Builder	"Separate the construction of a complex object from its representation so that the same construction process can create different representations."
Abstract Factory	"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."
Factory Method	"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

Note: These definitions are taken from *Design Patterns*.

1.6.2 Pattern: Singleton

1.6.2.1 Definition

"Ensure a class has only one instance, and provide a global point of access to it."

This is one of the easiest patterns to implement.

1.6.2.2 Applications in Delphi

There are several examples of this sort of class in the Delphi VCL, such as TApplication, TScreen or TClipboard. The pattern is useful whenever you want a single global object in your application. Other uses might include a global exception handler, application security, or a single point of interface to another application.

1.6.2.3 Implementation Example

To implement a class of this type, override the constructor and destructor of the class to refer to a global (interface) variable of the class.

Abort the constructor if the variable is assigned, otherwise create the instance and assign the variable.

In the destructor, clear the variable if it refers to the instance being destroyed.

Note: To make the creation and destruction of the single instance automatic, include its creation in the initialization section of the unit. To destroy the instance, include its destruction in an ExitProc (Delphi 1) or in the finalization section of the unit (Delphi 2).

1.6.3 Pattern: Adapter

1.6.3.1 Definition

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

1.6.3.2 Applications in Delphi

A typical example of this is the wrapper Delphi generates when you import a VBX or OCX. Delphi generates a new class which translates the interface of the external control into a Pascal compatible interface. Another typical case is when you want to build a single interface to old and new systems.

Note Delphi does not allow class adaption through multiple inheritance in the way described in Design Patterns. Instead, the adapter needs to refer to a specific instance of the old class.

1.6.3.3 Implementation Example

The following example is a simple (read only) case of a new customer class, an adapter class and an old customer class. The adapter illustrates handling the year 2000 problem, translating an old customer record containing two digit years into a new date format. The client using this wrapper only knows about the new customer class. Translation between classes is handled by the use of virtual access methods for the properties. The old customer class and adapter class are hidden in the implementation of the unit.

1.6.4 Pattern: Template Method

1.6.4.1 Definition

"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

This pattern is essentially an extension of abstract methods to more complex algorithms.

1.6.4.2 Applications in Delphi

Abstraction is implemented in Delphi by abstract virtual methods. Abstract methods differ from virtual methods by the base class not providing any implementation. The descendant class is completely responsible for implementing an abstract method. Calling an abstract method that has not been overridden will result in a runtime error.

1.6.4.3 A typical example of abstraction is the TGraphic class.

TGraphic is an abstract class used to implement TBitmap, TIcon and TMetafile. Other developers have frequently used TGraphic as the basis for other graphics objects such as PCX, GIF, JPG representations. TGraphic defines abstract methods such as Draw, LoadFromFile and SaveToFile which are then overridden in the concrete classes. Other objects that use TGraphic, such as a TCanvas only know about the abstract Draw method, yet are used with the concrete class at runtime.

Many classes that use complex algorithms are likely to benefit from abstraction using the template method approach. Typical examples include data compression, encryption and advanced graphics processing.

1.6.4.4 Implementation Example

To implement template methods you need an abstract class and concrete classes for each alternate implementation. Define a public interface to an algorithm in an abstract base class. In that public method, implement the steps of the algorithm in calls to protected abstract methods of the class. In concrete classes derived from the base class, override each step of the algorithm with a concrete implementation specific to that class.

1.6.5 Pattern: Builder

1.6.5.1 Definition

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

A Builder seems similar in concept to the Abstract Factory. The difference as I see it is the Builder refers to single complex objects of different concrete classes but containing multiple parts, whereas the abstract factory lets you create whole families of concrete classes. For example, a builder might construct a house, cottage or office. You might employ a different builder for a brick house or a timber house, though you would give them both similar instructions about the size and shape of the house. On the other hand the factory generates parts and not the whole. It might produce a range of windows for buildings, or it might produce a quite different range of windows for cars.

1.6.5.2 Applications in Delphi

The functionality used in Delphi's VCL to create forms and components is similar in concept to the builder. Delphi creates forms using a common interface, through Application.CreateForm and through the TForm class constructor. TForm implements a

common constructor using the resource information (DFM file) to instantiate the components owned by the form. Many descendant classes reuse this same construction process to create different representations. Delphi also makes developer extensions easy. TForm's OnCreate event also adds a hook into the builder process to make the functionality easy to extend.

1.6.5.3 Implementation Example

The following example includes a class TAbstractFormBuilder and two concrete classes TRedFormBuilder and TBlueFormBuilder. For ease of development some common functionality of the concrete classes has been moved into the shared TAbstractFormBuilder class.

1.6.6 Pattern: Abstract Factory

1.6.6.1 Definition

"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."

The Factory Method pattern below is commonly used in this pattern.

1.6.6.2 Applications in Delphi

This pattern is ideal where you want to isolate your application from the implementation of the concrete classes. For example if you wanted to overlay Delphi's VCL with a common VCL layer for both 16 and 32 bit applications, you might start with the abstract factory as a base.

1.6.6.3 Implementation Example

The following example uses an abstract factory and two concrete factory classes to implement different styles of user interface components. TOAbstractFactory is a singleton class, since we usually want one factory to be used for the whole application.

At runtime, our client application instantiates the abstract factory with a concrete class and then uses the abstract interface. Parts of the client application that use the factory don't need to know which concrete class is actually in use.

1.6.7 Pattern: Factory Method

1.6.7.1 Definition

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

The Abstact Factory pattern can be viewed as a collection of Factory Methods.

1.6.7.2 Applications in Delphi

This pattern is useful when you want to encapsulate the construction of a class and isolate knowledge of the concrete class from the client application through an abstract interface.

One example of this might arise if you had an object oriented business application potentially interfacing to multiple target DBMS. The client application only wants to know about the business classes, not about their implementation-specific storage and retrieval.

1.6.7.3 Implementation Example

In the Abstract Factory example, each of the virtual widget constructor functions is a Factory Method. In their implementation we define a specific widget class to return.

1.7 KEY ELEMENTS OF DELPHI CLASS DEFINITIONS

1.7.1 Unit Structure

Delphi units (.PAS files) allow declaration of interface and implementation sections. The interface defines the part that is visible to other units using that unit. The keyword *uses* can be added to a unit's interface or implementation section to list the other units that your unit uses. This indicates to the compiler that your unit refers to parts of the used unit's interface. Parts of a unit declared in the implementation section are all private to that unit, i.e. never visible to any other unit. Types, functions and procedures declared in the interface of a unit must have a corresponding implementation, or be declared as external (e.g. a call to a function in a DLL).

1.7.2 Class Interfaces

Classes are defined as types in Delphi and may contain fields of standard data types or other objects, methods declared as functions or procedures, and properties. The type declaration of a class defines its interface and the scope of access to fields, methods and properties of the class. Class interfaces are usually defined in the interface of a unit to make them accessible to other modules using that unit. However they don't need to be. Sometimes a type declaration of a class may be used only within the implementation part of a unit.

1.7.3 Properties

Properties are a specialised interface to a field of a defined type, allowing access control through read and write methods. Properties are not virtual, you can replace a property with another property of the same name, but the parent class doesn't know about the new property. It is however possible to make the access methods of a property virtual.

1.7.4 Inheritance

Delphi's inheritance model is based on a single hierarchy. Every class inherits from TObject and can have only one parent.

A descendant class inherits all of the interface and functionality of its parent class, subject to the scope described below.

Multiple inheritance from more than one parent is not allowed directly. It can be implemented by using a container class to create instances one or more other classes and selectively expose parts of the contained classes.

Private, Protected, Public and Published ScopeScope refers to the visibility of methods and data defined in the interface of a class, i.e. what parts of the class are accessible to the rest of the application or to descendant classes.

The default scope is public, for instance the component instances you add to a form at design time. Public says "come and get me"; it makes the data or method visible to everything at runtime.

Published parts of a class are a specialized form of Public scope. They indicate special behaviour for classes derived from TPersistent. A persistent class can save and restore its published properties to persistent storage using Delphi's standard streaming methods. Published properties also interact with Delphi Object Inspector in the IDE. A class must descend from TPersistent in order to use Published. There's also not much point in publishing methods, since you can't store them, although Delphi's compiler doesn't stop you. Published also lets another application access details of the class through Delphi's runtime type information. This would be rarely used, except in Delphi's design time interaction with its VCL.

Encapsulation or information hiding is essential to object orientation, so Protected and Private scope let you narrow the access to parts of a class.

Protected parts are visible only to descendant classes, or to other classes defined in the same unit.

Private parts are visible only to the defining class, or to other classes defined in the same unit.

It's important to note that once something is given public or published scope, it cannot be hidden in descendant classes.

Static, Virtual and Dynamic Methods; Override and Inherited

Methods declared as virtual or dynamic let you change their behaviour using override in a descendant class. You're unlikely to see a virtual method in the private part of a class, since it could only be overridden in the same unit, although Delphi's compiler doesn't stop you from doing this.

Override indicates that your new method replaces the method of the same name from the parent class. The override must be declared with the same name and parameters as the original method.

When a method is overridden, a call to the parent class's method actually executes the override method in the real class of the object.

Static methods on the other hand have no virtual or override declaration. You can replace a method of a class in a descendant class by redeclaring another method, however this is not object oriented. If you reference your descendant class as the parent type and try to call the replaced method, the static method of the parent class is executed. So in most cases, it's a bad idea to replace a static method.

Virtual and dynamic methods can be used interchangeably. They differ only in their treatment by the compiler and runtime library. Delphi's help explains that dynamic methods have their implementation resolved at compile time and run slightly faster, whereas virtual methods are resolved at runtime, resulting in slightly slower access but a smaller compiled program. Virtual is usually the preferred declaration. Delphi's help suggests using dynamic when you have a base class with many descendants that may not override the method.

The inherited directive lets you refer back to a property or method as it was declared in the parent class. This is most often used in the implementation of an override method, to call the inherited method of the parent class and then supplement its behaviour.

1.7.5 Abstract Methods

Abstract is used in base classes to declare a method in the interface and defer its implementation to a descendant class. I.e. it defines an interface, but not the underlying

operation. Abstract must be used with the virtual or dynamic directive. Abstract methods are never implemented in the base class and must be implemented in descendant classes to be used. A runtime error occurs if you try to execute an abstract method that is not overridden. Calling inherited within the override implementation of an abstract method will also result in a runtime error, since there is no inherited behaviour.

1.7.6 Messages

Delphi's handling of Windows messages is a special case of virtual methods. Message handlers are implemented in classes that descend from TControl. I.e classes that have a handle and can receive messages. Message handlers are always virtual and can be declared in the private part of a class interface, yet still allow the inherited method to be called. Inherited in a message handler just uses the keyword inherited, there is no need to supply the name of the method to call.

1.7.7 Events

Events are also an important characteristic of Delphi, since they let you delegate extensible behaviour to instances of a class. Events are properties that refer to a method of another object. Events are not inherited in Delphi 1; Delphi 2 extends this behaviour to let you use inherited in an event. . Inherited in an event handler just uses the keyword inherited, there is no need to supply the name of the method to call.

Events are particularly important to component developers, since they provide a hook for the user of the component to modify its behaviour in a way that may not be foreseen at the time the component is written.

1.7.8 Constructors and Destructors

The constructor and destructor are two special types of methods. The constructor initializes a class instance (allocates memory initialized to 0) and returns a reference (pointer) to the object. The destructor deallocates memory used by the object (but not the memory of other objects created by the object).

Classes descended from TObject have a static constructor, Create, and a virtual destructor Destroy.

TComponent introduces a new public property, the Owner of the component and this must be initialized in the constructor. TComponent's constructor is declared virtual, i.e. it can be overridden in descendant classes.It is essential when you override a virtual constructor or destructor in a TComponent descendant to include a call to the inherited method.

1.8 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.

1.8.1 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.

1.8.2 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 a 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.

1.8.3 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.

1.8.4 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.

1.8.4.1 Standard Components

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox and Tedit, 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 Window's 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.

1.8.4.2 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.

1.8.4.3 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 can't 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.

1.8.4.4 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.

1.8.4.5 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.

1.8.4.6 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.

1.9 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.

1.9.1 Property-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 that 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.

1.9.2 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.

Property type	Object Inspector treatment
Simple	Numeric, character, and string properties appear in the Object Inspector as numbers, characters, and strings, respectively. The user can type and edit the value of the property directly.
Enumerated	Properties of enumerated types (including Boolean) display the value as defined in the source code. The user can cycle through the possible values by double-clicking the value column. There is also a drop-down list that shows all possible values of the enumerated type.
Set	Properties of set types appear in the Object Inspector looking like a set. By expanding the set, the user can treat each element of the set as a Boolean value: True if the element is included in the set or False if it's not included.
Object	Properties that are themselves objects often have their own property editors. However, if the object that is a property also has published properties, the Object Inspector allows the user to expand the list of object properties and edit them individually. Object properties must descend from TPersistent.
Array	Array properties must have their own property editors. The Object Inspector has no built-in support for editing array properties.

For more information on properties, refer to the "Component Writers Guide" which ships with Delphi.

1.9.3 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.

1.9.4 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.

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.

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;
```

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.

1.9.5 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.

1.9.6 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 code 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.

1.9.7 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 it's 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. This task is done automatically by Delphi's design-time environment when you drop a component from the Component Palette onto your form.

CHAPTER 2

DATABASE

Every thing around us has a particular identity. To identify anything system, actor or person in words we need a data or information. So this information is valuable and in this advanced era we can store it in database and access this data by the blink of eye.

For an instant if we go through the definitions of database we may find following definitions.

A database is a collection of related information.

A database is an organized body of related information.

2.1 DEMERITS OF ABSENCE OF DATABASE

A glance on the past will may help us to reveal the drawbacks in case of absence of database.

In the past when there wasn't proper system of database, Much paper work was need to do and to handle great deal of written paper documentation was giant among the problems itself.

In the huge networks to deal with equally bulky data, more workers are needed which affidavit cost much labor expanses.

The old criteria for saving data and making identification was much time consuming such as if we want to search the particular data of a person.

Before the Development of Computer database it was a great problem to search for some thing. Efforts to avoid the headache of search often results in new establishments of data.

Before the development of database it seemed very unsafe to keep the worthy information. In Some situation some big organization had to employee the special persons in order to secure the data.

Before the implementation of database any firm had to face the plenty of difficulties in order to maintain their Management. To hold the check on the expenses of the firm, the manager faced difficulties.

2.2 MERITS OF DATABASE

The modern era is known as the golden age computer sciences and technology. In a simple phrase we can express that the modern age is built on the foundation of database.

If we carefully watch our daily life we can examine that some how our daily life is being connected with database.

There are several benefits of database developments.

Now with the help of computerized database we can access data in a second.

By the development of the database we can make data more secure.

By the development of database we can reduce the cost.

2.3 DATABASE DESIGN

The design of a database has to do with the way data is stored and how that data is related. The design process is performed after you determine exactly what information needs to be stored and how it is to be retrieved.

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

Computerized library systems

Automated teller machines

Flight reservation systems

Computerized parts inventory systems

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query.

Database design is a complex subject. A properly designed database is a model of a business, Country Database or some other in the real world. Like their physical model counterparts, data models enable you to get answers about the facts that make up the objects being modeled. It's the questions that need answers that determine which facts need to be stored in the data model.

In the relational model, data is organized in tables that have the following characteristics: every record has the same number of facts, every field contains the same type of facts (Data) in each record, and there is only one entry for each fact. No two records are exactly the same.

The more carefully you design, the better the physical database meets users' needs. In the process of designing a complete system, you must consider user needs from a variety of viewpoints.

2.4 DATABASE MODELS

Various techniques are used to model data structures. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementation may be possible. An example of this is the relational model: in larger systems the physical implementation often has indexes which point to the data; this is similar to some aspects of common implementations of the network model. But in small relational database the data is often stored in a set of

files, one per table, in a flat, un-indexed structure. There is some confusion below and elsewhere in this article as to logical data model vs. its physical implementation.

2.4.1 Flat Model

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 might be used as a part of a system security database. Each row would have the specific password associated with a specific 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 the basis of the spreadsheet.

2.4.2 Network Model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.

2.4.3 Relational Model

The relational data model was introduced in an academic paper by E.F. Cod 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.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allow tables to be defined that allow duplicate rows an extension (or violation) of the relational model. In common English usage, a DBMS is

called relational if it supports relational operational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, not-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't 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 row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no nature key is suitable, an arbitrary key can be assigned (such as by given 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 can't 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).

2.4.3.1 Why we use a Relational Database Design

Maintaining a simple, so-called flat database consisting of a single table doesn't require much knowledge of database theory. On the other hand, most database worth maintaining are quite a bit more complicated than that. Real life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full-fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million

books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database.

2.5 RELATIONSHIPS BETWEEN TABLES

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many and many-to-many relationships.

2.5.2 One-To-One Relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their tracks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support in a table requires security, placing them in a separate table lets your application restrict to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to these fields.

2.5.3 One-To-Many Relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a Many-To-Many relationship as well.

2.6 DATA MODELING

In information system design, data modeling is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "Data Analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together), than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

2.6.1 Database Normalization

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMS lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case de-normalizations are sometimes used to improve performance, at the cost of reduced consistency.

2.6.2 Primary Key

In database design, a primary key is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions) and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a primary key is a candidate key chosen as the main method of uniquely identifying a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System Numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others. In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

2.6.3 Foreign Key

A foreign key (FK) is a field in a database record under one primary key that points to a key field of another database record in another table where the foreign key of one table refers to the primary key of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail needs not to include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book, or even the book itself. The ISBN is the primary key of the book, and it is used as a foreign key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

2.6.4 Compound Key

In database design, a compound key (also called a composite key) is a key that consists on 2 or more attributes.

No restriction is applied to the attribute regarding their (initial) ownership within the data model. This means that any one, none or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute value.

CHAPTER 3
MYSQL

3.1 INTRODUCTION TO MYSQL

This chapter provides a tutorial introduction to MySQL by showing how to use the mysql client program to create and use a simple database. mysql (sometimes referred to as the ``terminal monitor" or just ``monitor") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you place your queries in a file beforehand, then tell mysql to execute the contents of the file. Both ways of using mysql are covered here.

To see a list of options provided by mysql, invoke it with the --help option:

shell> mysql --help

This chapter assumes that mysql is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If you are the administrator, you will need to consult other sections of this manual.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an already-existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily left out. Consult the relevant sections of the manual for more information on the topics covered here.

3.2 WHAT IS MYSQL?

3.2.1 Definition

MySQL is an open source software relational database management system (RDBMS) which

uses a SQL (Structured Query Language)

SQL is the standard language used for interacting with databases.

3.3 WHY CHOOSE MYSQL?

There are many relational databases available to use, so why choose MySQL?

We are specifically interested in databases which PHP supports; these include Oracle, IBM's DB2 and Microsoft's SQL Server (all of which cost money).

The two main open source (free) alternatives to these are PostgreSQL and MySQL. PostgreSQL is arguably the better of the two, but MySQL is better supported on Windows, and is a popular choice among Web hosts that provide support for PHP.

Here are some of MySQL's advantages

- It's fast
- It's free to use, and commercial licenses are reasonable
- It's easy to use
- It is cross platform
- There is a wide community of technical support
- It's secure
- It supports large databases
- It is designed specifically for web base applications and hence works very well partnered with PHP

3.4 PREPARING THE WINDOWS MYSQL ENVIRONMENT

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL- Max server binaries. Here is a list of the different MySQL servers you can use:

mysqld	Compiled with full debugging and automatic memory allocation checking, symbolic links, InnoDB and DBD tables.
mysql-opt	Optimized binary with no support for transactional tables.
mysqld-nt	Optimized binary for NT with support for named pipes. You can run this version on Win98, but in this case no named pipes are created and you must have TCP/IP installed.
mysqld-max	Optimized binary with support for symbolic links, InnoDB and DBD tables.
mysqld-max-nt	Like mysqld-max, but compiled with support for named pipes.

All of the above binaries are optimized for the Pentium Pro processor but should work on any Intel processor >=i386

In the following circumstance, you will need to use the MySQL configuration file:

- The install/data directories are different than the default 'c:\mysql' and 'c:\mysql\data'.
- If you want to use one of these servers:
  - mysqld.exe
  - mysqld-max.exe
  - mysqld-max-nt.exe
- If you need to tune the server settings.

There are two configuration files with the same function: 'my.cnf' and 'my.ini' file, however, only one of these can/should be used. Both files are plain text. The 'my.cnf' file should be created in the root directory of drive C and the 'my.ini' file in the WinDir directory e.g.: C:\WINDOWS or C:\WINNT. If your PC uses a boot loader where the C drive isn't the boot drive, then your only option is to use the 'my.ini' file. Also note that if you use the WinMySQLAdmin tool, only the

'my.ini' file is used. The '\mysql\bin' directory contains a help file with instructions for using this tool.

Using Notepad, create the configuration file and edit the base section and keys:

[mysqld]

basedir = the_install_path # e.g. 'c:\mysql'

datadir = the_data_path # e.g. 'c:\mysql\data' or 'd:\mydata\data'

If the data directory is other than the default 'c:\mysql\data', you must cut the whole

'\data\mysql' directory and paste it on the your option new directory, e.g. 'd:\mydata\mysql'.

If you want to use the InnoDB transaction tables, you need to manually create two new directories to hold the InnoDB data and log files, e.g. 'c:\ibdata' and 'c:\iblogs'. You will also need to create some extra lines to the configuration file.

If you don't want to use, add the skip-innodb option to the configuration file.

Now you are ready to test starting the server.

3.5 STARTING THE SERVER FOR THE FIRST TIME

Testing from a DOS command prompt is the best thing to do because the server prints messages, so if something is wrong with your configuration, you will see a more accurate error message which will make it easier to identify and fix any problems.

Make sure you're in the right directory *(C:\>cd \mysql\bin),*

 # To install mysqld as a standalone program, enter:

C:\mysql\bin> mysqld-max --standalone

You should see the below print messages:

```
InnoDB: The first specified data file c:\ibdata\ibdata1 did not exist
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 200715200
InnoDB: Database physically writes the file full: wait
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

 # To install mysql as a service (Windows 2000), enter:

C:\mysql\bin> mysqld-nt --install

Now you can start and stop mysqld as follows:

C:\>NET START MySQL C:\>NET STOP MySQL

C:\>NET START MySQL

 # To start the MySQL Monitor, enter:

The MySql service is starting.

The MySQL service was started successfully.

C:\>cd \mysql

C:\mysql>bin\mysql

Welcome to the MySQL Monitor. Commands end with ; or \g. Your MySQL connection id

is 1 to server version 3.23.49-nt Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> (enter a command or enter 'QUIT' to quit)

mysql> QUIT Bye

C:\mysql>NET STOP MySQL The MySQL service is stopping.

The MySQL service was stopped successfully.

C:\mysql>

3.6 CONNECTING TO AND DISCONNECTING FROM THE SERVER

To connect to the server, you'll usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

shell> mysql -h host -u user -p

Enter password: ********

The ******** represents your password; enter it when mysql displays the Enter password:

prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

shell> mysql -h host -u user -p

Enter password: ********

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>

The prompt tells you that mysql is ready for you to enter commands.

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

shell> mysql

After you have connected successfully, you can disconnect any time by typing QUIT at the *mysql>*

prompt: mysql> QUIT Bye

You can also disconnect by pressing Control-D.

Most examples in the following sections assume you are connected to the server. They indicate this by the mysql> prompt.

3.7 ENTERING QUERIES

Make sure you are connected to the server, as discussed in the previous section. Doing so will not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how mysql works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown below following the mysql> prompt and hit the RETURN key:

mysql> SELECT VERSION( ), CURRENT_DATE;

version()	CURRENT_DATE
3.22.20a-log	1999-03-19

row in set (0.01 sec)

mysql>

This query illustrates several things about mysql:

A command normally consists of a SQL statement followed by a semicolon. (There are some exceptions where a semicolon is not needed. QUIT, mentioned earlier, is one of them. We'll get to others later.)

When you issue a command, mysql sends it to the server for execution and displays the results, then prints another mysql> to indicate that it is ready for another command.

Mysql displays query output as a table (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), mysql labels the column using the expression itself.

Mysql shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the ``rows in set'' line is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

mysql> SELECT VERSION(), CURRENT_DATE; mysql> select version(), current_date;
mysql> SELECT VERSION(), current_DATE;
*mysql> SELECT SIN(PI()/4), (4+1)*5;*

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

VETERINERIAN APPLICATION PROGRAM WITH DELPHI

Graduation Project
COM400

Student: Ahmet KAYABAŞ (20021329)

Supervisor: Mr. Elburus IMANOV

Lefkoşa-2007

TABLE OF CONTENT

ACKNOWLEDGMENT

When people start a new work they get excited.Because who do not know any thing about the future of work.When a time passed human becomes familiar for this work.Afterthat may be borred,maybe want to leave this work.That may be true maybe false.It changes from people to people.But I believe that the important thing in the life do not leave such who should embrace very tightly.When we get this it makes us happy.

In the life what is important for you.Business? Money? Science? Power? Family? Love? Humanity? or purpose of existence? In my opinion first of all aim of existence comes.Rest of all things involved in aim of existence.After that comes Love.The world exists of love.With love person gets power, gains working perseverence .

Well in this project I gained perseverence from ***Allah*** *and from my fiancee.I am happy to complete the task which I had given with blessing of* ***Allah*** *and also I am grateful to my fiancee and all the people in my life who have supported me, advised me. They all the time helped and encouraged me to follow my dreams and ambitions.*

For intellectual support, encouragement I want to thank to my supervisor ***Mr. Elburus Imanov*** *who made this project contributions.*

And thank ***my dearest parents*** *who supported me to continue beyond my undergraduate studies, and also many thanks to my dear familiy who brought me till such meaning days.*

To all my friends, especially ***M.Fethullah Akatay, Selman Kayabaş, Metin Yenigün , Kadir Bekiroğlu and My dear fiancee*** *for sharing wonderful moments, advice, and for making me feel at home and in life. And above, I thank God for giving me stamina and courage to achieve my objectives.*

AHMET KAYABAŞ

ABSTRACT

In the world not only human life is important.In the same time other entity lives with us.We are not alone on the earth.Animals share life with us.Ilnesses are not only for human.In the same time whole alive interested with illnesses.How Doctor is important for us like Veterinerian is important for animals.Todays Doctors use application program.Because of to keep knowledge of patient, to facility diagnosis of illness, to reach background of patient efficiently and easly.

Well Veterinerian application program is important like the program that is used human health.Also much more important then others.Because animal can not keep the illnesses knowledge.And also papers of the animal can lost.

This project has as its goal to develop software, processing information about activities of a veterinerian application software. Software developed in this project like not only for animal.In the same time for staff and for owner of the animal.All records keep in the other Database program.It acts easly and fast access.Veterinerian can keep all records in the program as concentment.

INTRODUCTION

Since human created by the powerful Allah, Human wonder everything.Well who tried to satisfy wonder.Such humanity came to nowadays as develop.Todays everyone says technology perfect developed.Yes that is right.By means of technology all process gained velocity.This development acts to spend time to the people.

Technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.The automation is also became a part of our lives. The people operate with automation systems in everywhere.

Veterinerian Application project which is my project.In this software veterinerian can keep animal knowledge, patient background knowledge of the animal, owner of the animal knowledge.With this software veterinerian will make record process easily and safetly.

In Software there are five types user.They can access to only their task process.In the same time in the program veterinerian can get obligation as daily.The software can be used at every animal clinic easly.

CHAPTER 1

DELPHI

1.1 INTRODUCTION TO DELPHI

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

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

It is a method for predicting future events.

It is a method for generating a quick consensus by a group.

It is the use of a survey to collect information.

It is the use of anonymity on the part of the participants.

It is the use of voting to reduce the need for long discussions.

It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing detailed critical examination and discussion, not at forcing a quick

compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective:

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

Additional opportunity has been added by the introduction of Computer Mediated Communication Systems (Hiltz and Turoff, 1978; Rice and Associates, 1984; Turoff, 1989; Turoff, 1991). These are computer systems that support group communications in either a synchronous (Group Decision Support Systems, Desanctis et. al., 1987) or an asynchronous manner (Computer Conferencing). Techniques that were developed and refined in the evolution of the Delphi Method (e.g. anonymity, voting) have been incorporated as basic facilities or tools in many of these computer based systems. As a result, any of these systems can be used to carry out some form of a Delphi process or Nominal Group Technique (Delbecq, et. al., 1975).

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

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

1.2 WHAT IS DELPHI?

Delphi is an object oriented, 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 What's more, 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 tools.

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!

1.3 WHAT KIND OF PROGRAMMING CAN YOU DO WITH DELPHI?

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

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

Some projects to which Delphi is suited:

- Simple, single user database applications

- Intermediate multi-user database applications

- Large scale multi-tier, multi-user database applications

- Internet applications

- Graphics Applications

- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools

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.

1.4 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 quitcapable ofthis.

1.5 SOME KNOWLEDGE ABOUT DELPHI

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

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

For the purposes of this series I will be using Delphi 7. There are more recent versions available (2005 and 2006) however Delphi 7 should be available inexpensively compared to the new versions which will set you back a lot of money. Delphi 7 will more than likely be available in a magazine for free.

1.5.2 Example: Try First Delphi Program

First thing is first, fire up your copy of Delphi and open the Project > Options menu. To compile a console application you need to change a setting on the Linker tab called 'Generate console application', check the box and click OK. Now select File > Close All if anything is already loaded. Then select File > New > Other > Console Application.

Notice the first line refers to the keyword program. You can rename this to HelloWorld. You can also remove the commented portion enclosed in curly brackets. The uses keyword allows you to list all units that you want to use in the program. At the moment just leave it as it is, SysUtils is all we need.

Your unit should now look like this:

Delphi Code:

```
program HelloWorld;

  {$APPTYPE CONSOLE}

 uses

  SysUtils;

 begin

 end.
```

Now what we have just done is written a program, it currently doesn't do a thing however. Hit the run button and see the result. Now wasn't that completely worthless.

Luckily this isn't the end of the article so we'll actually have a worthwhile program at the end of it. All we need to do is insert some code in the main procedure we have just made.

Every good programmer's first program was 'Hello World' and you'll be no exception. All we need to do is use the WriteLn procedure to write 'Hello World!' to the console, simple.Notice the semicolon at the end of the line, at the end of any statement you need to add a semicolon. Run the program and see the results…

Now I don't know about you but I saw hello world flash up and go away in a second, if you didn't write the program you wouldn't even know what it said. To solve this problem we need to tell the program to leave the console open until the user is ready to close it. We can use ReadLn for this which reads the users input from the console.

Delphi Code:

```
program HelloWorld;

 {$APPTYPE CONSOLE}

 uses

   SysUtils;

 begin

   WriteLn('Hello World!' + #13#10 + #13#10 +

     'Press RETURN to end...');

   ReadLn;

 end.
```

I have added a few extra things into the 'Hello World' string so the user knows what to do to end the program as it could be a bit confusing. '#13#10' is to insert a carriage

return as 13 and 10 are the ASCII codes for a carriage return followed by a new line feed. ASCII can be inserted in this way into strings.

1.5.2 Delphi Style

Coding style, the way you format your code and the way in which you present it on the page.At the end of the day who cares about my style, I can read it, and Delphi strips all the spaces out of it and doesn't care if I indent. Why waste my time?

Neatly present code which conforms to the accepted standards not only makes your code much easier for you to read and debug but also but any one else who might read your code to help you, or learn from you can do so with ease. After all which code is easier to follow, example 1 or 2?

Delphi Code:

```
// Example 1

procedure xyz();

var

x,y,z,a:integer;

begin

x:=1;y:=2;

for z:=x to y do begin

a:=power(z,y);

showmessage(inttostr(a));

end;
```

```
end;
```

Delphi Code:

```
// Example 2
procedure XYZ();
var
X,Y,Z,A: Integer;
begin
  X := 1;
  Y := 2;
  for Z := X to Y do
  begin
        A := Power(Z, Y);
        ShowMessage(IntToStr(A));
  end; // for end
end; // procedure end
```

Design patterns are frequently recurring structures and relationships in object-oriented design. Getting to know them can help you design better, more reusable code and also help you learn to design more complex systems.

Much of the ground-breaking work on design patterns was presented in the book Design Patterns: Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides. You might also have heard of the authors referred to as "the Gang of Four". If you haven't read this book before and you're designing objects, it's an excellent

primer to help structure your design. To get the most out of these examples, I recommend reading the book as well.

Another good source of pattern concepts is the book Object Models: Strategies, Patterns and Applications by Peter Coad. Coad's examples are more business oriented and he emphasises learning strategies to identify patterns in your own work.

1.6 HOW DELPHI HELPS YOU DEFINE PATTERNS

Delphi implements a fully object-oriented language with many practical refinements that simplify development.

The most important class attributes from a pattern perspective are the basic inheritance of classes; virtual and abstract methods; and use of protected and public scope. These give you the tools to create patterns that can be reused and extended, and let you isolate varying functionality from base attributes that are unchanging.

Delphi is a great example of an extensible application, through its component architecture, IDE interfaces and tool interfaces. These interfaces define many virtual and abstract constructors and operations.

1.6.1 Delphi Examples of Design Patterns

I should note from the outset, there may be alternative or better ways to implement these patterns and I welcome your suggestions on ways to improve the design. The following patterns from the book *Design Patterns* are discussed and illustrated in Delphi to give you a starting point for implementing your own Delphi patterns.

Pattern Name	*Definition*
Singleton	"Ensure a class has only one instance, and provide a global point of access to it."
Adapter	"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't

	otherwise because of incompatible interfaces."
Template Method	"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."
Builder	"Separate the construction of a complex object from its representation so that the same construction process can create different representations."
Abstract Factory	"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."
Factory Method	"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

Note: These definitions are taken from *Design Patterns*.

1.6.2 Pattern: Singleton

1.6.2.1 Definition

"Ensure a class has only one instance, and provide a global point of access to it."

This is one of the easiest patterns to implement.

1.6.2.2 Applications in Delphi

There are several examples of this sort of class in the Delphi VCL, such as TApplication, TScreen or TClipboard. The pattern is useful whenever you want a single global object in your application. Other uses might include a global exception handler, application security, or a single point of interface to another application.

1.6.2.3 Implementation Example

To implement a class of this type, override the constructor and destructor of the class to refer to a global (interface) variable of the class.

Abort the constructor if the variable is assigned, otherwise create the instance and assign the variable.

In the destructor, clear the variable if it refers to the instance being destroyed.

Note: To make the creation and destruction of the single instance automatic, include its creation in the initialization section of the unit. To destroy the instance, include its destruction in an ExitProc (Delphi 1) or in the finalization section of the unit (Delphi 2).

1.6.3 Pattern: Adapter

1.6.3.1 Definition

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

1.6.3.2 Applications in Delphi

A typical example of this is the wrapper Delphi generates when you import a VBX or OCX. Delphi generates a new class which translates the interface of the external control into a Pascal compatible interface. Another typical case is when you want to build a single interface to old and new systems.

Note Delphi does not allow class adaption through multiple inheritance in the way described in Design Patterns. Instead, the adapter needs to refer to a specific instance of the old class.

1.6.3.3 Implementation Example

The following example is a simple (read only) case of a new customer class, an adapter class and an old customer class. The adapter illustrates handling the year 2000 problem, translating an old customer record containing two digit years into a new date format. The client using this wrapper only knows about the new customer class. Translation between classes is handled by the use of virtual access methods for the properties. The old customer class and adapter class are hidden in the implementation of the unit.

1.6.4 Pattern: Template Method

1.6.4.1 Definition

"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

This pattern is essentially an extension of abstract methods to more complex algorithms.

1.6.4.2 Applications in Delphi

Abstraction is implemented in Delphi by abstract virtual methods. Abstract methods differ from virtual methods by the base class not providing any implementation. The descendant class is completely responsible for implementing an abstract method. Calling an abstract method that has not been overridden will result in a runtime error.

1.6.4.3 A typical example of abstraction is the TGraphic class.

TGraphic is an abstract class used to implement TBitmap, TIcon and TMetafile. Other developers have frequently used TGraphic as the basis for other graphics objects such as PCX, GIF, JPG representations. TGraphic defines abstract methods such as Draw, LoadFromFile and SaveToFile which are then overridden in the concrete classes. Other objects that use TGraphic, such as a TCanvas only know about the abstract Draw method, yet are used with the concrete class at runtime.

Many classes that use complex algorithms are likely to benefit from abstraction using the template method approach. Typical examples include data compression, encryption and advanced graphics processing.

1.6.4.4 Implementation Example

To implement template methods you need an abstract class and concrete classes for each alternate implementation. Define a public interface to an algorithm in an abstract base class. In that public method, implement the steps of the algorithm in calls to protected abstract methods of the class. In concrete classes derived from the base class, override each step of the algorithm with a concrete implementation specific to that class.

1.6.5 Pattern: Builder

1.6.5.1 Definition

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

A Builder seems similar in concept to the Abstract Factory. The difference as I see it is the Builder refers to single complex objects of different concrete classes but containing multiple parts, whereas the abstract factory lets you create whole families of concrete classes. For example, a builder might construct a house, cottage or office. You might employ a different builder for a brick house or a timber house, though you would give them both similar instructions about the size and shape of the house. On the other hand the factory generates parts and not the whole. It might produce a range of windows for buildings, or it might produce a quite different range of windows for cars.

1.6.5.2 Applications in Delphi

The functionality used in Delphi's VCL to create forms and components is similar in concept to the builder. Delphi creates forms using a common interface, through Application.CreateForm and through the TForm class constructor. TForm implements a

common constructor using the resource information (DFM file) to instantiate the components owned by the form. Many descendant classes reuse this same construction process to create different representations. Delphi also makes developer extensions easy. TForm's OnCreate event also adds a hook into the builder process to make the functionality easy to extend.

1.6.5.3 Implementation Example

The following example includes a class TAbstractFormBuilder and two concrete classes TRedFormBuilder and TBlueFormBuilder. For ease of development some common functionality of the concrete classes has been moved into the shared TAbstractFormBuilder class.

1.6.6 Pattern: Abstract Factory

1.6.6.1 Definition

"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."

The Factory Method pattern below is commonly used in this pattern.

1.6.6.2 Applications in Delphi

This pattern is ideal where you want to isolate your application from the implementation of the concrete classes. For example if you wanted to overlay Delphi's VCL with a common VCL layer for both 16 and 32 bit applications, you might start with the abstract factory as a base.

1.6.6.3 Implementation Example

The following example uses an abstract factory and two concrete factory classes to implement different styles of user interface components. TOAbstractFactory is a singleton class, since we usually want one factory to be used for the whole application.

At runtime, our client application instantiates the abstract factory with a concrete class and then uses the abstract interface. Parts of the client application that use the factory don't need to know which concrete class is actually in use.

1.6.7 Pattern: Factory Method

1.6.7.1 Definition

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

The Abstact Factory pattern can be viewed as a collection of Factory Methods.

1.6.7.2 Applications in Delphi

This pattern is useful when you want to encapsulate the construction of a class and isolate knowledge of the concrete class from the client application through an abstract interface.

One example of this might arise if you had an object oriented business application potentially interfacing to multiple target DBMS. The client application only wants to know about the business classes, not about their implementation-specific storage and retrieval.

1.6.7.3 Implementation Example

In the Abstract Factory example, each of the virtual widget constructor functions is a Factory Method. In their implementation we define a specific widget class to return.

1.7 KEY ELEMENTS OF DELPHI CLASS DEFINITIONS

1.7.1 Unit Structure

Delphi units (.PAS files) allow declaration of interface and implementation sections. The interface defines the part that is visible to other units using that unit. The keyword *uses* can be added to a unit's interface or implementation section to list the other units that your unit uses. This indicates to the compiler that your unit refers to parts of the used unit's interface. Parts of a unit declared in the implementation section are all private to that unit, i.e. never visible to any other unit. Types, functions and procedures declared in the interface of a unit must have a corresponding implementation, or be declared as external (e.g. a call to a function in a DLL).

1.7.2 Class Interfaces

Classes are defined as types in Delphi and may contain fields of standard data types or other objects, methods declared as functions or procedures, and properties. The type declaration of a class defines its interface and the scope of access to fields, methods and properties of the class. Class interfaces are usually defined in the interface of a unit to make them accessible to other modules using that unit. However they don't need to be. Sometimes a type declaration of a class may be used only within the implementation part of a unit.

1.7.3 Properties

Properties are a specialised interface to a field of a defined type, allowing access control through read and write methods. Properties are not virtual, you can replace a property with another property of the same name, but the parent class doesn't know about the new property. It is however possible to make the access methods of a property virtual.

1.7.4 Inheritance

Delphi's inheritance model is based on a single hierarchy. Every class inherits from TObject and can have only one parent.

A descendant class inherits all of the interface and functionality of its parent class, subject to the scope described below.

Multiple inheritance from more than one parent is not allowed directly. It can be implemented by using a container class to create instances one or more other classes and selectively expose parts of the contained classes.

Private, Protected, Public and Published ScopeScope refers to the visibility of methods and data defined in the interface of a class, i.e. what parts of the class are accessible to the rest of the application or to descendant classes.

The default scope is public, for instance the component instances you add to a form at design time. Public says "come and get me"; it makes the data or method visible to everything at runtime.

Published parts of a class are a specialized form of Public scope. They indicate special behaviour for classes derived from TPersistent. A persistent class can save and restore its published properties to persistent storage using Delphi's standard streaming methods. Published properties also interact with Delphi Object Inspector in the IDE. A class must descend from TPersistent in order to use Published. There's also not much point in publishing methods, since you can't store them, although Delphi's compiler doesn't stop you. Published also lets another application access details of the class through Delphi's runtime type information. This would be rarely used, except in Delphi's design time interaction with its VCL.

Encapsulation or information hiding is essential to object orientation, so Protected and Private scope let you narrow the access to parts of a class.

Protected parts are visible only to descendant classes, or to other classes defined in the same unit.

Private parts are visible only to the defining class, or to other classes defined in the same unit.

It's important to note that once something is given public or published scope, it cannot be hidden in descendant classes.

Static, Virtual and Dynamic Methods; Override and Inherited

Methods declared as virtual or dynamic let you change their behaviour using override in a descendant class. You're unlikely to see a virtual method in the private part of a class, since it could only be overridden in the same unit, although Delphi's compiler doesn't stop you from doing this.

Override indicates that your new method replaces the method of the same name from the parent class. The override must be declared with the same name and parameters as the original method.

When a method is overridden, a call to the parent class's method actually executes the override method in the real class of the object.

Static methods on the other hand have no virtual or override declaration. You can replace a method of a class in a descendant class by redeclaring another method, however this is not object oriented. If you reference your descendant class as the parent type and try to call the replaced method, the static method of the parent class is executed. So in most cases, it's a bad idea to replace a static method.

Virtual and dynamic methods can be used interchangeably. They differ only in their treatment by the compiler and runtime library. Delphi's help explains that dynamic methods have their implementation resolved at compile time and run slightly faster, whereas virtual methods are resolved at runtime, resulting in slightly slower access but a smaller compiled program. Virtual is usually the preferred declaration. Delphi's help suggests using dynamic when you have a base class with many descendants that may not override the method.

The inherited directive lets you refer back to a property or method as it was declared in the parent class. This is most often used in the implementation of an override method, to call the inherited method of the parent class and then supplement its behaviour.

1.7.5 Abstract Methods

Abstract is used in base classes to declare a method in the interface and defer its implementation to a descendant class. I.e. it defines an interface, but not the underlying

operation. Abstract must be used with the virtual or dynamic directive. Abstract methods are never implemented in the base class and must be implemented in descendant classes to be used. A runtime error occurs if you try to execute an abstract method that is not overridden. Calling inherited within the override implementation of an abstract method will also result in a runtime error, since there is no inherited behaviour.

1.7.6 Messages

Delphi's handling of Windows messages is a special case of virtual methods. Message handlers are implemented in classes that descend from TControl. I.e classes that have a handle and can receive messages. Message handlers are always virtual and can be declared in the private part of a class interface, yet still allow the inherited method to be called. Inherited in a message handler just uses the keyword inherited, there is no need to supply the name of the method to call.

1.7.7 Events

Events are also an important characteristic of Delphi, since they let you delegate extensible behaviour to instances of a class. Events are properties that refer to a method of another object. Events are not inherited in Delphi 1; Delphi 2 extends this behaviour to let you use inherited in an event. . Inherited in an event handler just uses the keyword inherited, there is no need to supply the name of the method to call.

Events are particularly important to component developers, since they provide a hook for the user of the component to modify its behaviour in a way that may not be foreseen at the time the component is written.

1.7.8 Constructors and Destructors

The constructor and destructor are two special types of methods. The constructor initializes a class instance (allocates memory initialized to 0) and returns a reference (pointer) to the object. The destructor deallocates memory used by the object (but not the memory of other objects created by the object).

Classes descended from TObject have a static constructor, Create, and a virtual destructor Destroy.

TComponent introduces a new public property, the Owner of the component and this must be initialized in the constructor. TComponent's constructor is declared virtual, i.e. it can be overridden in descendant classes.It is essential when you override a virtual constructor or destructor in a TComponent descendant to include a call to the inherited method.

1.8 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.

1.8.1 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.

1.8.2 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 a 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.

1.8.3 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.

1.8.4 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.

1.8.4.1 Standard Components

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox and Tedit, 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 Window's 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.

1.8.4.2 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.

1.8.4.3 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 can't 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.

1.8.4.4 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.

1.8.4.5 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.

1.8.4.6 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.

1.9 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.

1.9.1 Property-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 that 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.

1.9.2 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.

Property type	Object Inspector treatment
Simple	Numeric, character, and string properties appear in the Object Inspector as numbers, characters, and strings, respectively. The user can type and edit the value of the property directly.
Enumerated	Properties of enumerated types (including Boolean) display the value as defined in the source code. The user can cycle through the possible values by double-clicking the value column. There is also a drop-down list that shows all possible values of the enumerated type.
Set	Properties of set types appear in the Object Inspector looking like a set. By expanding the set, the user can treat each element of the set as a Boolean value: True if the element is included in the set or False if it's not included.
Object	Properties that are themselves objects often have their own property editors. However, if the object that is a property also has published properties, the Object Inspector allows the user to expand the list of object properties and edit them individually. Object properties must descend from TPersistent.
Array	Array properties must have their own property editors. The Object Inspector has no built-in support for editing array properties.

For more information on properties, refer to the "Component Writers Guide" which ships with Delphi.

1.9.3 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.

1.9.4 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.

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.

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;
```

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.

1.9.5 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.

1.9.6 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 code 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.

1.9.7 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 it's 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. This task is done automatically by Delphi's design-time environment when you drop a component from the Component Palette onto your form.

CHAPTER 2

DATABASE

Every thing around us has a particular identity. To identify anything system, actor or person in words we need a data or information. So this information is valuable and in this advanced era we can store it in database and access this data by the blink of eye.

For an instant if we go through the definitions of database we may find following definitions.

A database is a collection of related information.

A database is an organized body of related information.

2.1 DEMERITS OF ABSENCE OF DATABASE

A glance on the past will may help us to reveal the drawbacks in case of absence of database.

In the past when there wasn't proper system of database, Much paper work was need to do and to handle great deal of written paper documentation was giant among the problems itself.

In the huge networks to deal with equally bulky data, more workers are needed which affidavit cost much labor expanses.

The old criteria for saving data and making identification was much time consuming such as if we want to search the particular data of a person.

Before the Development of Computer database it was a great problem to search for some thing. Efforts to avoid the headache of search often results in new establishments of data.

Before the development of database it seemed very unsafe to keep the worthy information. In Some situation some big organization had to employee the special persons in order to secure the data.

Before the implementation of database any firm had to face the plenty of difficulties in order to maintain their Management. To hold the check on the expenses of the firm, the manager faced difficulties.

2.2 MERITS OF DATABASE

The modern era is known as the golden age computer sciences and technology. In a simple phrase we can express that the modern age is built on the foundation of database.

If we carefully watch our daily life we can examine that some how our daily life is being connected with database.

There are several benefits of database developments.

Now with the help of computerized database we can access data in a second.

By the development of the database we can make data more secure.

By the development of database we can reduce the cost.

2.3 DATABASE DESIGN

The design of a database has to do with the way data is stored and how that data is related. The design process is performed after you determine exactly what information needs to be stored and how it is to be retrieved.

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

Computerized library systems

Automated teller machines

Flight reservation systems

Computerized parts inventory systems

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query.

Database design is a complex subject. A properly designed database is a model of a business, Country Database or some other in the real world. Like their physical model counterparts, data models enable you to get answers about the facts that make up the objects being modeled. It's the questions that need answers that determine which facts need to be stored in the data model.

In the relational model, data is organized in tables that have the following characteristics: every record has the same number of facts, every field contains the same type of facts (Data) in each record, and there is only one entry for each fact. No two records are exactly the same.

The more carefully you design, the better the physical database meets users' needs. In the process of designing a complete system, you must consider user needs from a variety of viewpoints.

2.4 DATABASE MODELS

Various techniques are used to model data structures. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementation may be possible. An example of this is the relational model: in larger systems the physical implementation often has indexes which point to the data; this is similar to some aspects of common implementations of the network model. But in small relational database the data is often stored in a set of

files, one per table, in a flat, un-indexed structure. There is some confusion below and elsewhere in this article as to logical data model vs. its physical implementation.

2.4.1 Flat Model

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 might be used as a part of a system security database. Each row would have the specific password associated with a specific 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 the basis of the spreadsheet.

2.4.2 Network Model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.

2.4.3 Relational Model

The relational data model was introduced in an academic paper by E.F. Cod 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.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allow tables to be defined that allow duplicate rows an extension (or violation) of the relational model. In common English usage, a DBMS is

called relational if it supports relational operational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, not-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't 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 row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no nature key is suitable, an arbitrary key can be assigned (such as by given 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 can't 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).

2.4.3.1 Why we use a Relational Database Design

Maintaining a simple, so-called flat database consisting of a single table doesn't require much knowledge of database theory. On the other hand, most database worth maintaining are quite a bit more complicated than that. Real life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full-fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million

books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database.

2.5 RELATIONSHIPS BETWEEN TABLES

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many and many-to-many relationships.

2.5.2 One-To-One Relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their tracks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support in a table requires security, placing them in a separate table lets your application restrict to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to these fields.

2.5.3 One-To-Many Relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a Many-To-Many relationship as well.

2.6 DATA MODELING

In information system design, data modeling is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "Data Analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together), than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

2.6.1 Database Normalization

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMS lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case de-normalizations are sometimes used to improve performance, at the cost of reduced consistency.

2.6.2 Primary Key

In database design, a primary key is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions) and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a primary key is a candidate key chosen as the main method of uniquely identifying a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System Numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others. In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

2.6.3 Foreign Key

A foreign key (FK) is a field in a database record under one primary key that points to a key field of another database record in another table where the foreign key of one table refers to the primary key of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail needs not to include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book, or even the book itself. The ISBN is the primary key of the book, and it is used as a foreign key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

2.6.4 Compound Key

In database design, a compound key (also called a composite key) is a key that consists on 2 or more attributes.

No restriction is applied to the attribute regarding their (initial) ownership within the data model. This means that any one, none or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute value.

CHAPTER 3
MYSQL

3.1 INTRODUCTION TO MYSQL

This chapter provides a tutorial introduction to MySQL by showing how to use the mysql client program to create and use a simple database. mysql (sometimes referred to as the ``terminal monitor" or just ``monitor") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you place your queries in a file beforehand, then tell mysql to execute the contents of the file. Both ways of using mysql are covered here.

To see a list of options provided by mysql, invoke it with the --help option:

shell> mysql --help

This chapter assumes that mysql is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If you are the administrator, you will need to consult other sections of this manual.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an already-existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily left out. Consult the relevant sections of the manual for more information on the topics covered here.

3.2 WHAT IS MYSQL?

3.2.1 Definition

MySQL is an open source software relational database management system (RDBMS) which

uses a SQL (Structured Query Language)

SQL is the standard language used for interacting with databases.

3.3 WHY CHOOSE MYSQL?

There are many relational databases available to use, so why choose MySQL?

We are specifically interested in databases which PHP supports; these include Oracle, IBM's DB2 and Microsoft's SQL Server (all of which cost money).

The two main open source (free) alternatives to these are PostgreSQL and MySQL. PostgreSQL is arguably the better of the two, but MySQL is better supported on Windows, and is a popular choice among Web hosts that provide support for PHP.

Here are some of MySQL's advantages

- It's fast
- It's free to use, and commercial licenses are reasonable
- It's easy to use
- It is cross platform
- There is a wide community of technical support
- It's secure
- It supports large databases
- It is designed specifically for web base applications and hence works very well partnered with PHP

3.4 PREPARING THE WINDOWS MYSQL ENVIRONMENT

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL- Max server binaries. Here is a list of the different MySQL servers you can use:

mysqld	Compiled with full debugging and automatic memory allocation checking, symbolic links, InnoDB and DBD tables.
mysql-opt	Optimized binary with no support for transactional tables.
mysqld-nt	Optimized binary for NT with support for named pipes. You can run this version on Win98, but in this case no named pipes are created and you must have TCP/IP installed.
mysqld-max	Optimized binary with support for symbolic links, InnoDB and DBD tables.
mysqld-max-nt	Like mysqld-max, but compiled with support for named pipes.

All of the above binaries are optimized for the Pentium Pro processor but should work on any Intel processor >=i386

In the following circumstance, you will need to use the MySQL configuration file:

- The install/data directories are different than the default 'c:\mysql' and 'c:\mysql\data'.
- If you want to use one of these servers:
  - mysqld.exe
  - mysqld-max.exe
  - mysqld-max-nt.exe
- If you need to tune the server settings.

There are two configuration files with the same function: 'my.cnf' and 'my.ini' file, however, only one of these can/should be used. Both files are plain text. The 'my.cnf' file should be created in the root directory of drive C and the 'my.ini' file in the WinDir directory e.g.: C:\WINDOWS or C:\WINNT. If your PC uses a boot loader where the C drive isn't the boot drive, then your only option is to use the 'my.ini' file. Also note that if you use the WinMySQLAdmin tool, only the

'my.ini' file is used. The '\mysql\bin' directory contains a help file with instructions for using this tool.

Using Notepad, create the configuration file and edit the base section and keys:

[mysqld]

basedir = the_install_path # e.g. 'c:\mysql'

datadir = the_data_path # e.g. 'c:\mysql\data' or 'd:\mydata\data'

If the data directory is other than the default 'c:\mysql\data', you must cut the whole

'\data\mysql' directory and paste it on the your option new directory, e.g. 'd:\mydata\mysql'.

If you want to use the InnoDB transaction tables, you need to manually create two new directories to hold the InnoDB data and log files, e.g. 'c:\ibdata' and 'c:\iblogs'. You will also need to create some extra lines to the configuration file.

If you don't want to use, add the skip-innodb option to the configuration file.

Now you are ready to test starting the server.

3.5 STARTING THE SERVER FOR THE FIRST TIME

Testing from a DOS command prompt is the best thing to do because the server prints messages, so if something is wrong with your configuration, you will see a more accurate error message which will make it easier to identify and fix any problems.

Make sure you're in the right directory *(C:\>cd \mysql\bin),*

 # To install mysqld as a standalone program, enter:

C:\mysql\bin> mysqld-max --standalone

You should see the below print messages:

```
InnoDB: The first specified data file c:\ibdata\ibdata1 did not exist
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 200715200
InnoDB: Database physically writes the file full: wait
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

 # To install mysql as a service (Windows 2000), enter:

C:\mysql\bin> mysqld-nt --install

Now you can start and stop mysqld as follows:

C:\>NET START MySQL C:\>NET STOP MySQL

C:\>NET START MySQL

 # To start the MySQL Monitor, enter:

The MySql service is starting.

The MySQL service was started successfully.

C:\>cd \mysql

C:\mysql>bin\mysql

Welcome to the MySQL Monitor. Commands end with ; or \g. Your MySQL connection id

is 1 to server version 3.23.49-nt Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> (enter a command or enter 'QUIT' to quit)

mysql> QUIT Bye

C:\mysql>NET STOP MySQL The MySQL service is stopping.

The MySQL service was stopped successfully.

C:\mysql>

3.6 CONNECTING TO AND DISCONNECTING FROM THE SERVER

To connect to the server, you'll usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

shell> mysql -h host -u user -p

Enter password: ********

The ******** represents your password; enter it when mysql displays the Enter password:

prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

shell> mysql -h host -u user -p

Enter password: ********

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>

The prompt tells you that mysql is ready for you to enter commands.

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

shell> mysql

After you have connected successfully, you can disconnect any time by typing QUIT at the *mysql>*

prompt: mysql> QUIT Bye

You can also disconnect by pressing Control-D.

Most examples in the following sections assume you are connected to the server. They indicate this by the mysql> prompt.

3.7 ENTERING QUERIES

Make sure you are connected to the server, as discussed in the previous section. Doing so will not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how mysql works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown below following the mysql> prompt and hit the RETURN key:

mysql> SELECT VERSION( ), CURRENT_DATE;

version()	CURRENT_DATE
3.22.20a-log	1999-03-19

row in set (0.01 sec)

mysql>

This query illustrates several things about mysql:

A command normally consists of a SQL statement followed by a semicolon. (There are some exceptions where a semicolon is not needed. QUIT, mentioned earlier, is one of them. We'll get to others later.)

When you issue a command, mysql sends it to the server for execution and displays the results, then prints another mysql> to indicate that it is ready for another command.

Mysql displays query output as a table (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), mysql labels the column using the expression itself.

Mysql shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the ``rows in set'' line is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

mysql> SELECT VERSION(), CURRENT_DATE; mysql> select version(), current_date;
mysql> SELECT VERSION(), current_DATE;
*mysql> SELECT SIN(PI()/4), (4+1)*5;*

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

mysql> SELECT VERSION(); SELECT NOW();

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. mysql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, mysql accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

mysql> SELECT USER(),CURRENT_DATE;

USER()	CURRENT_DATE
joesmith@localhost	1999-03-18

In this example, notice how the prompt changes from mysql> to -> after you enter the first line of a multiple-line query. This is how mysql indicates that it hasn't seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you will always be aware of what mysql is waiting for.

If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing \c:

mysql> SELECT USER() \c mysql>

Here, too, notice the prompt. It switches back to mysql> after you type \c, providing feedback to indicate that mysql is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that mysql is in:

Prompt	Meaning
mysql>	Ready for new command.
->	Waiting for next line of multiple-line command.
'>	Waiting for next line, collecting a string that begins with a single quote (`').
">	Waiting for next line, collecting a string that begins with a double quote (`"').

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, mysql waits for more input:

mysql> SELECT USER()
->

If this happens to you (you think you've entered a statement but the only response is a -> prompt), most likely mysql is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and mysql will execute it:

mysql> SELECT USER()
-> ;
USER()
joesmith@localhost

The '> and "> prompts occur during string collection. In MySQL, you can write strings surrounded by either `'` or `"` characters (for example, 'hello' or "goodbye"), and mysql lets you enter strings that span multiple lines. When you see a '> or "> prompt, it means that you've entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. That's fine if you really are entering a multiple-line string, but how likely is that? Not very. More often, the '> and "> prompts indicate that you've inadvertantly left out a quote character. For example:

*mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;*
">

If you enter this SELECT statement, then hit RETURN and wait for the result, nothing will happen. Instead of wondering why this query takes so long, notice the clue provided by the "> prompt. It tells you that mysql expects to see the rest of an unterminated string. (Do you see the error in the statement? The string "Smith is missing the second quote.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type \c in this case, because mysql interprets it as part of the string that it is collecting! Instead, enter the closing quote character (so mysql knows you've finished the string), then type

*\c:mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;*
"> "\c mysql>

The prompt changes back to mysql>, indicating that mysql is ready for a new command.

It's important to know what the '> and "> prompts signify, because if you mistakenly enter an unterminated string, any further lines you type will appear to be ignored by mysql -- including a line containing QUIT! This can be quite confusing, especially if you don't know that you need to supply the terminating quote before you can cancel the current command.

CHAPTER 4

USER MANUAL

In this chapter I will try to explain the veterinerian application program that when it run.If a someone run the program; firstly splash form will be shown for 3000ms like below.

Figure 4.1

After 3000ms entry page (Secure Page) will be shown. (Figure 4.2)

Figure 4.2

On this page (Figure 4.2) the user must enter the user name and password.If user name and password found then the program check the user state for still working or has left.If still working; now the program check the user position for Admin,Veterinerian,Manager,User and Temporary.If the user has left then who can not enter the system; in the same time there is no user name and password program gives three trying chance to enter the system; when is thirth the program will be terminate.

If user is Admin then who can access everything to make on application program; If user is manager then who access everything to make exclusive of wrong password application; If user is veterinerian then who can not access process of user after that who can access; If user is a normal user then who can see some knowledge and can change the program settings; If the user is temporary then who can access only amusements, internet explorer, find folder and drug knowledge.

Then main page comes (Figure 4.3) it is shown below

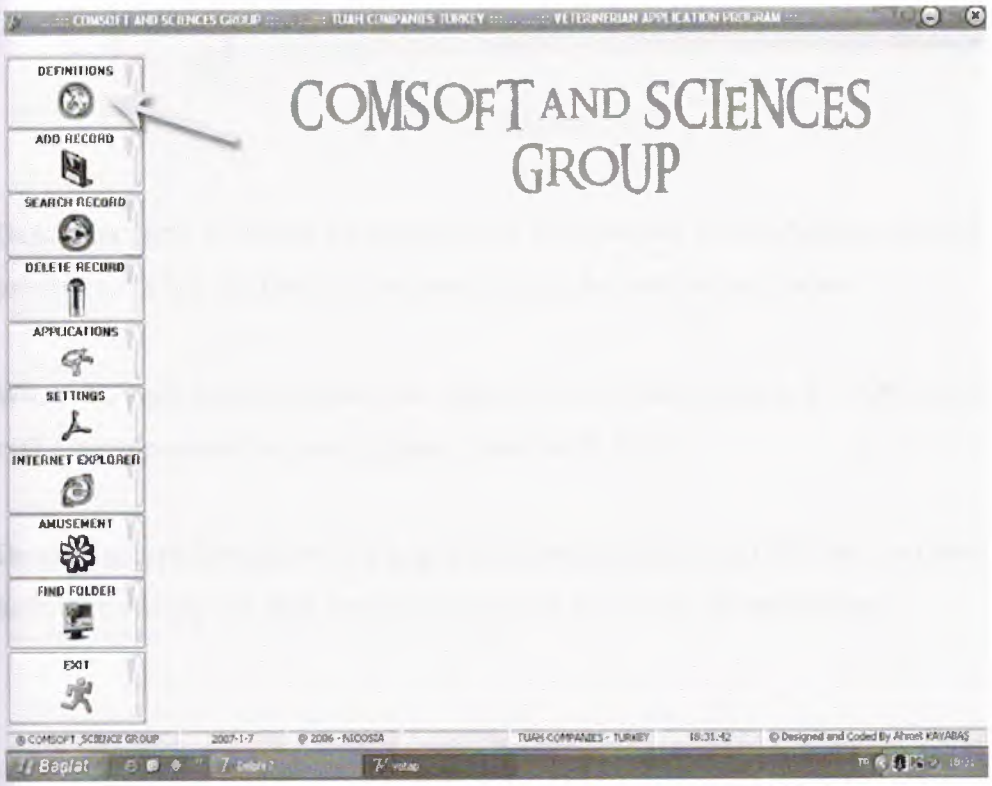

Figure 4.3

This is main page; other pages shown on it. There are ten button on this. User click one of them and access the page that wanted by the user.

When button of definition clicked definition selection page is shown like below figure

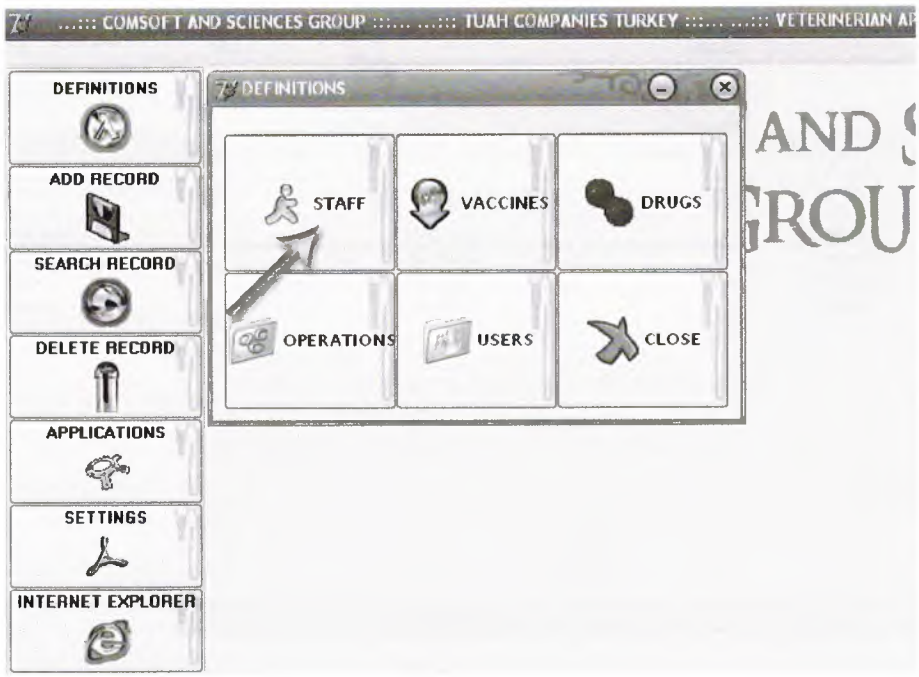

Figure 4.4

Definitions acts to create knowledge that is necessary for application process.User decide process and click the button to access the page for needed application.

When the staff button clicked; the page is shown that is figure 4.5. On this page user can make some process like save, update, delete and find.

On staff record form there is a magnifier button that acts that if there is a person who saved before;knowledge of that person is shown on form with all knowledge.

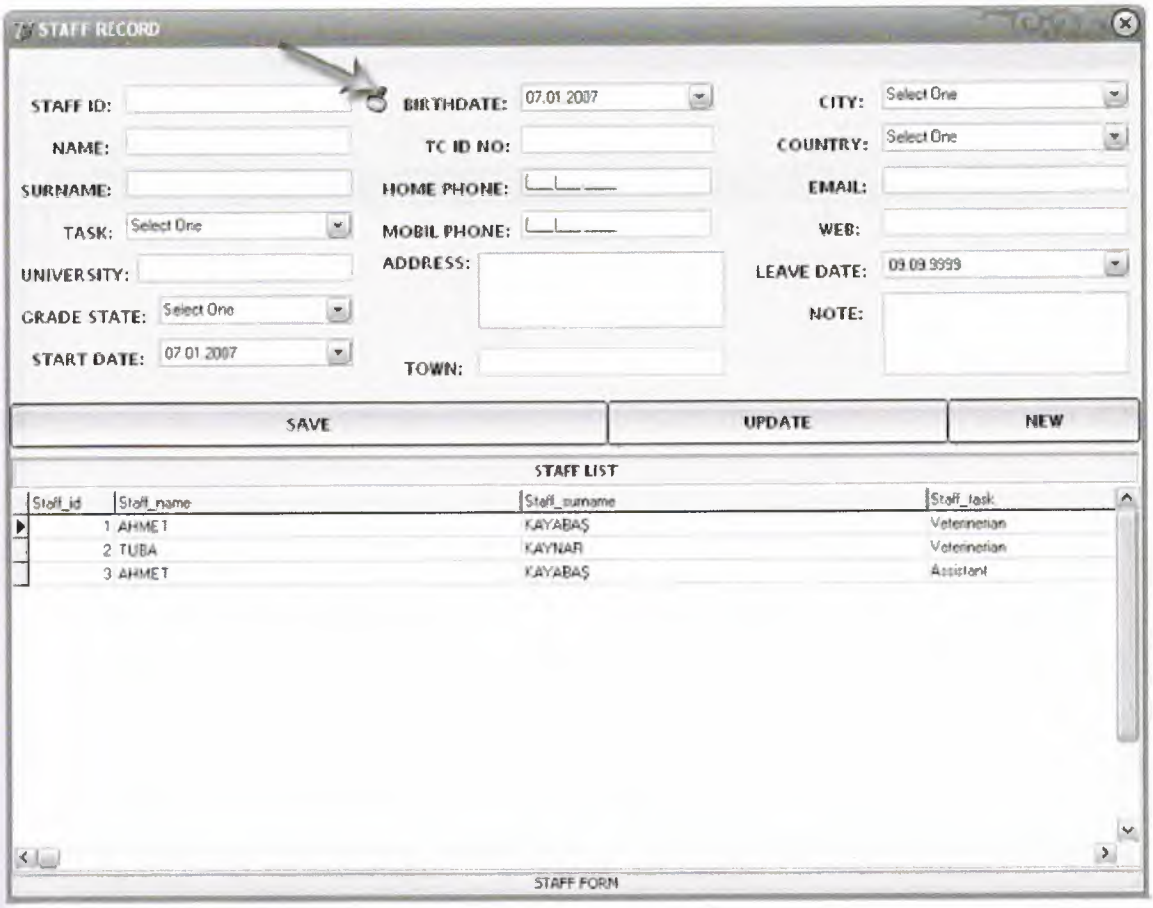

Figure 4.5

When the magnifier button clicked figure 4.6 is shown

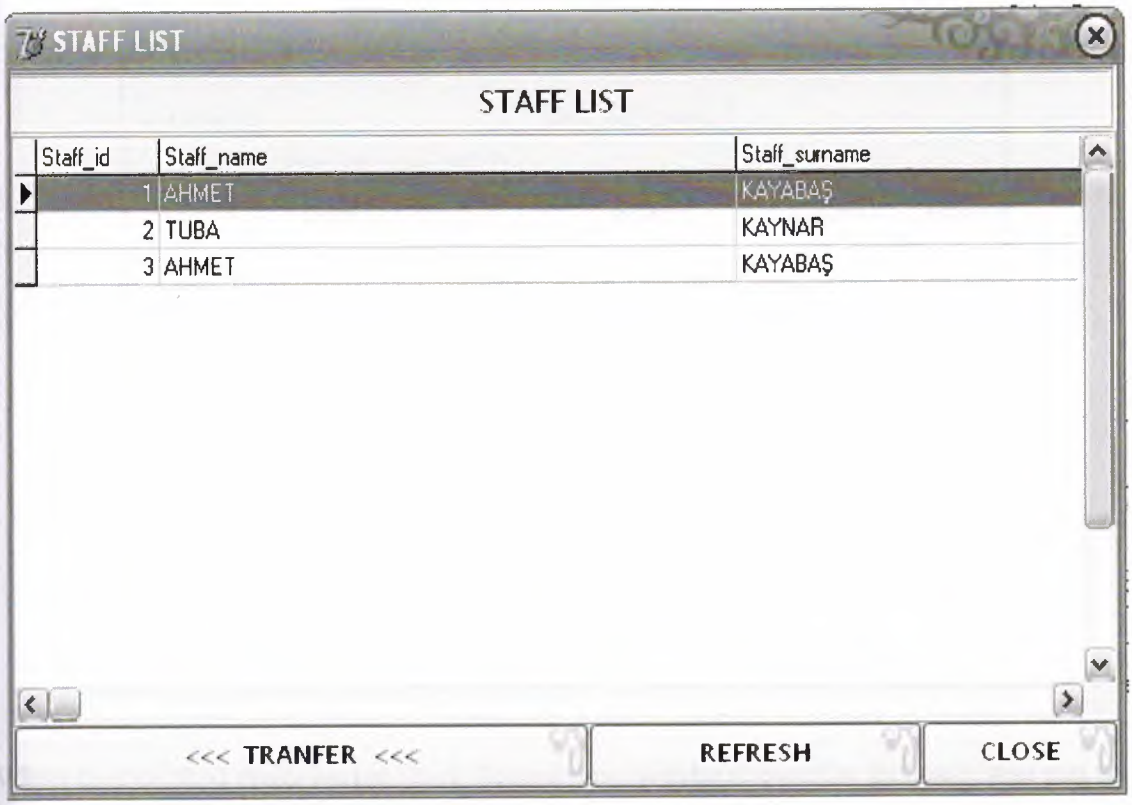

Figure 4.6

When the vaccines record clicked; the page is shown that is figure 4.5. On this page user can add new vaccine, can delete or update it.For process of vaccinates vaccine name called from here (Figure 4.7)

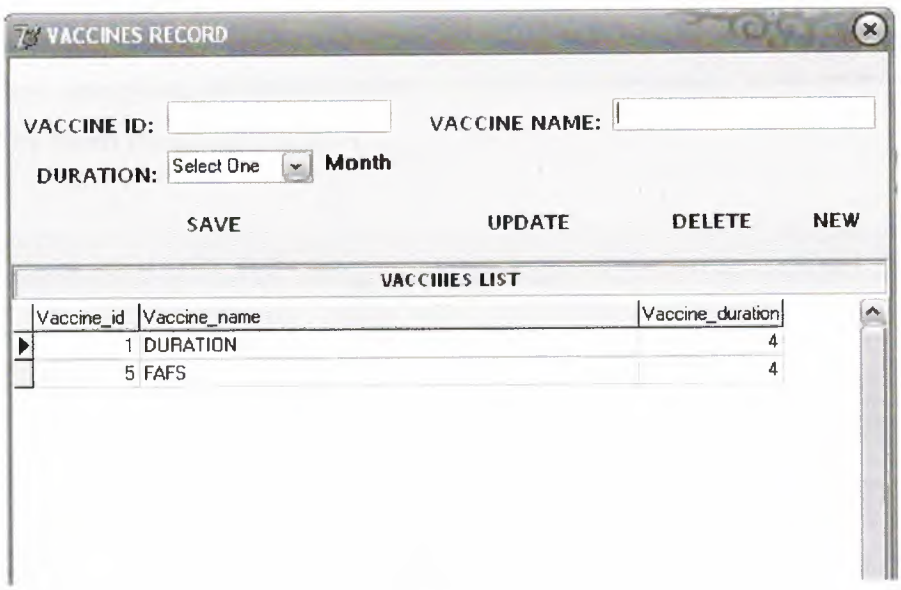

Figure 4.7

When user clicked drugs button Drug Record page will be shown.On this page user can add new drug, delete drug and update old drug record.For process of drug application drug name will absorb from here Figure 4.8

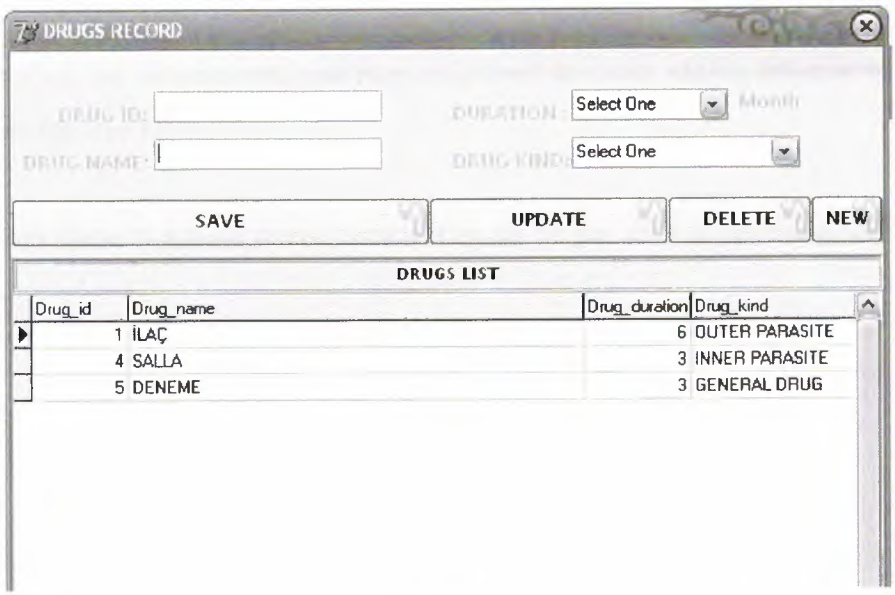

Figure 4.8

When operations button clicked Operation Record page will be shown.On this page user can add new operation, deleteand update operation.For operation application operation name will be taken from here Figure 4.9

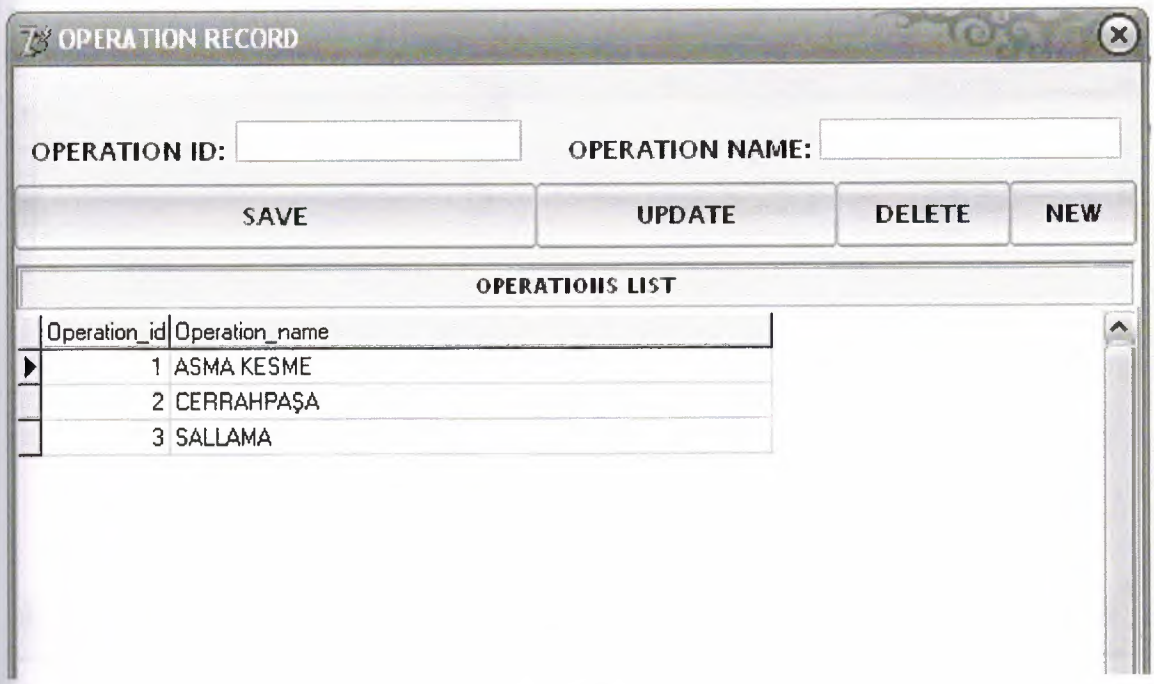

Figure 4.9

If the user click the user button; user page is opened to make adding,deleting and updating user knowledge like Figure 4.10.

On this form there is a mini arrow button.It is act to get staff to combine with users and staff.Because after when a knowledge is needed it stafisfied directly.

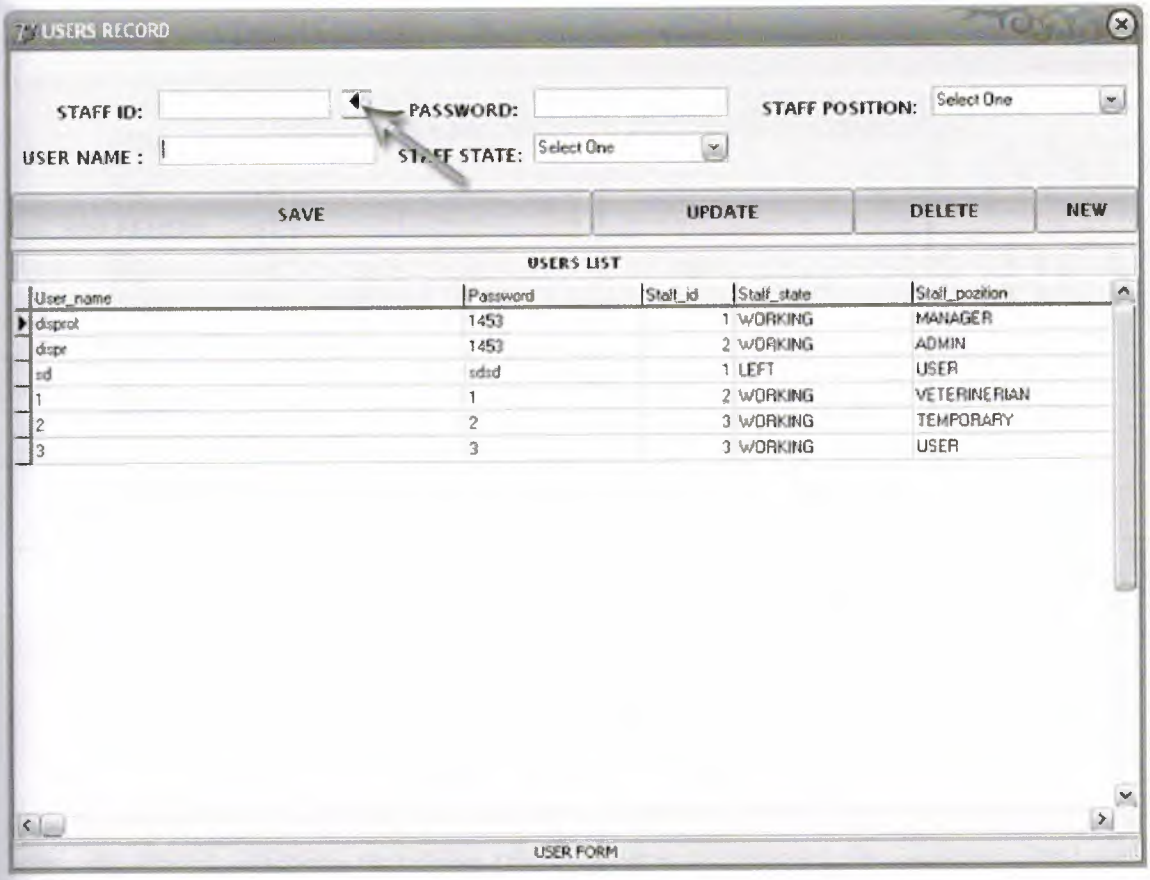

Figure 4.10

ADD Record button thet on main form acts to create knowledge that is necessary for continuity of program.Because Customer and Animal is defined here.User decide process and click the button to access the page for needed application.Figure 4.11 act transaction of this process from main menu.

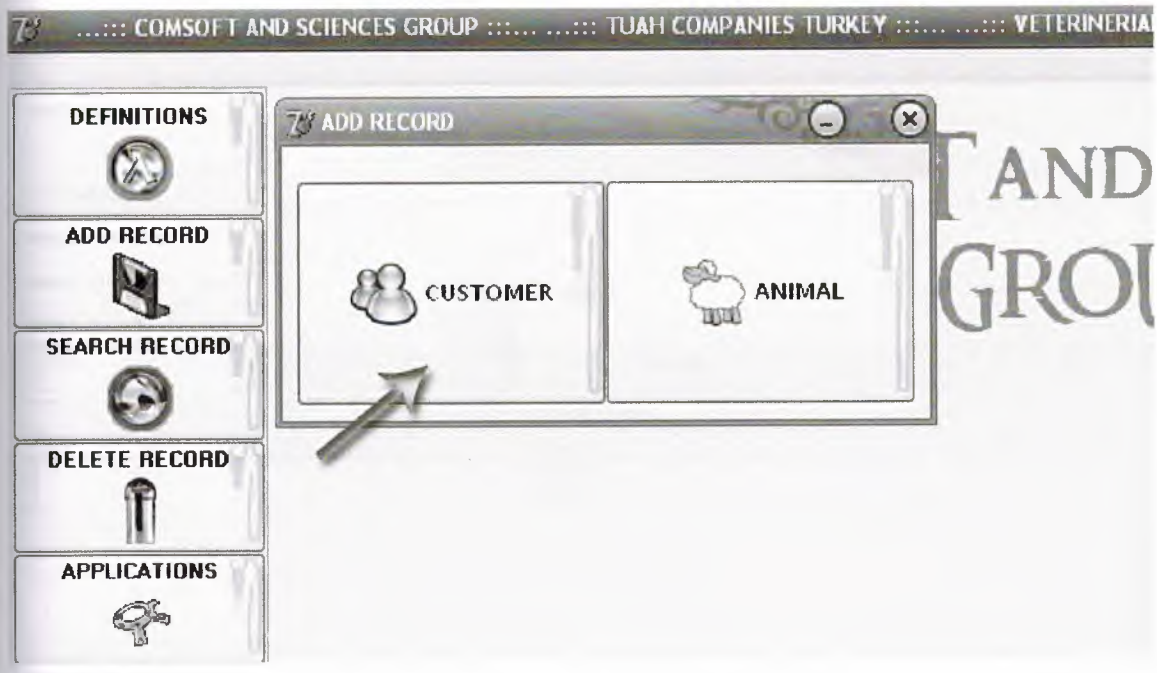

Figure 4.11

User can decide customer or animal.who if decide to continue for customer must click customer button.When he/she made this a new form is shown,Customer record form.With this form user can add an new customer or delete or update an old customer.Update or delete is needed.Well may be customer transferred to other city or transferred to other veterinerian.Figure 4.12 include a customer record page figure

The Program acts all of them easly.Interface is basic as shown.Every user can adapt easly to make operation.

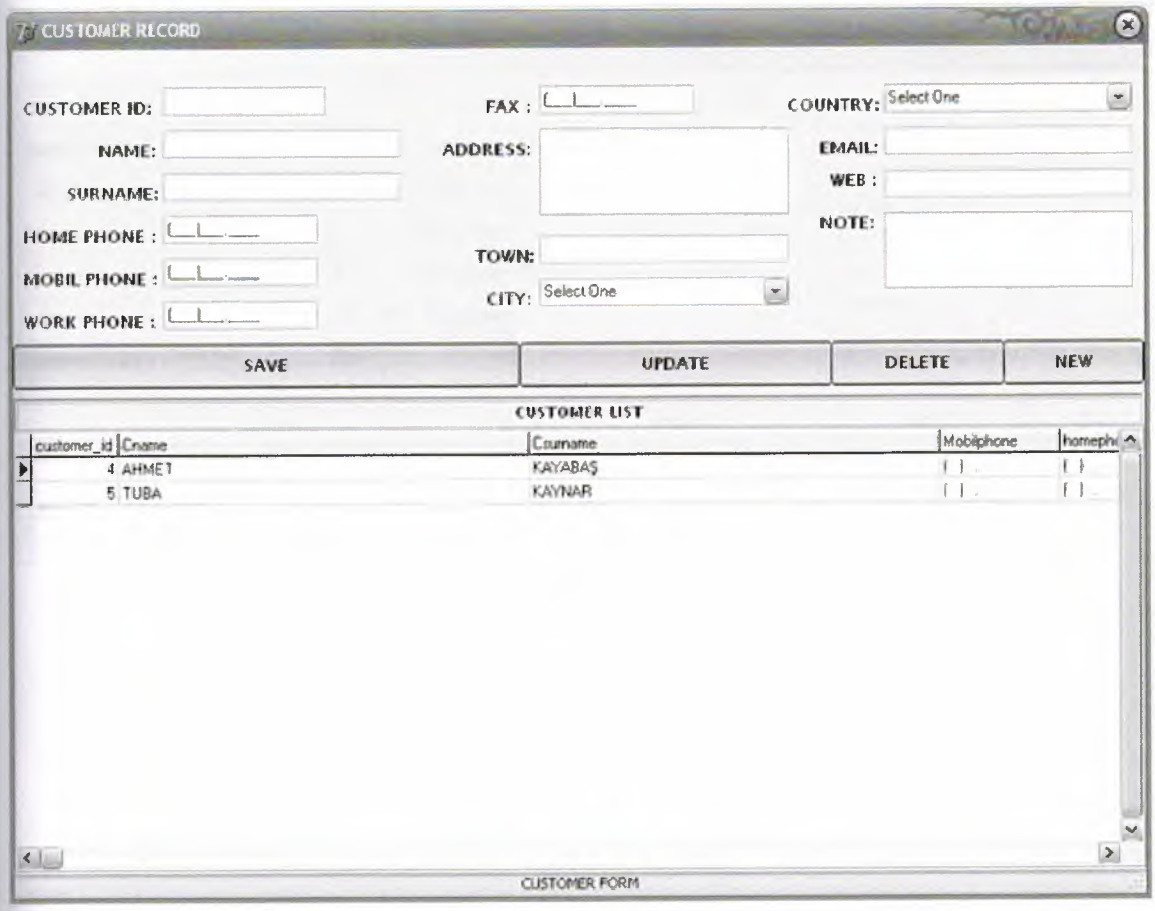

Figure 4.12

If user decided for animal must click animal button on Add Record Form (Figure 4.11). When he/she clicked animal button Animal Record Form will be displayed.With this form user can add an new animal or delete or update an old animal with their owner.Update or delete is needed.Well may be animal transferred to other veterinerian or may be died.

Figure 4.13 shows animal record form.

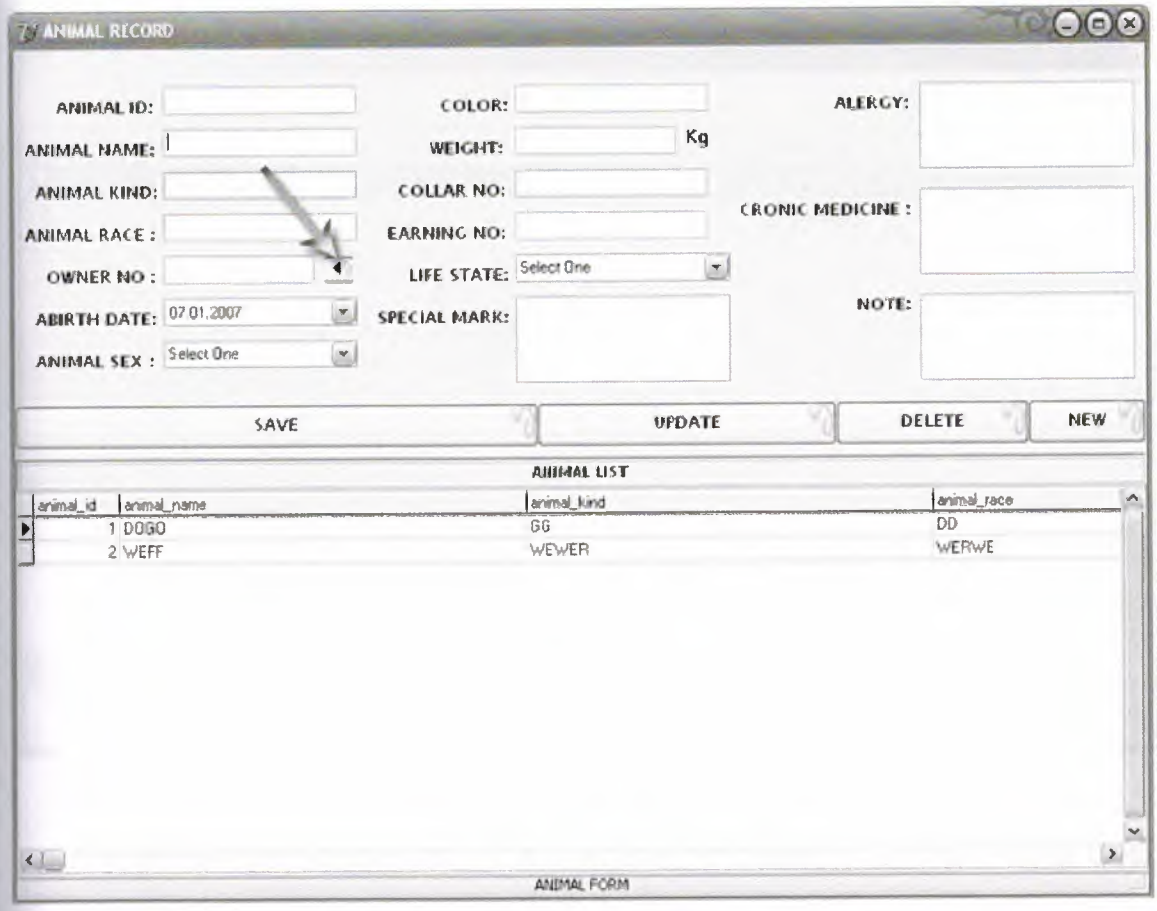

Figure 4.13

On this form (Figure 4.13) there is a mini arrow button.It find owner.Thats why initially customer must save then animal can save.Because as seen owner only called from other form directly.(Figure 4.14)

This Page (Figure 4.14) absorb the knowledge directly database through queries.When it opened datas comes onto dbgrid that on page.

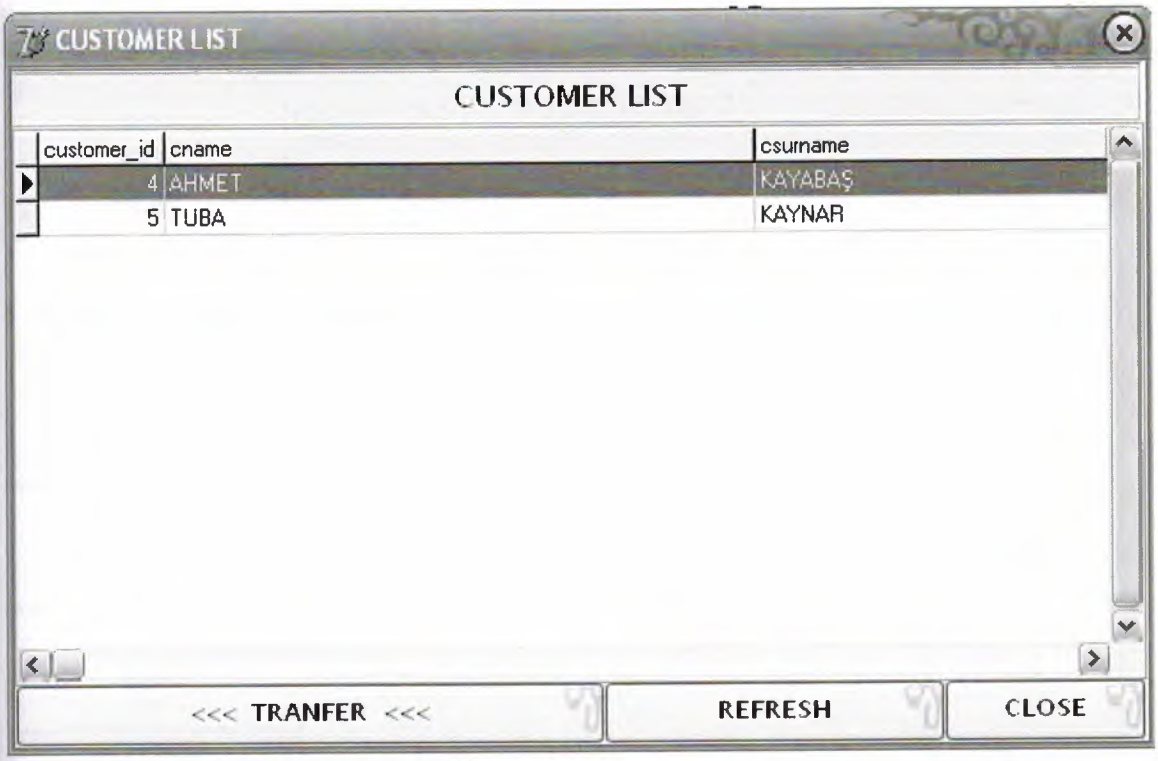

Figure 4.14

Search Record button that on main form acts to show knowledge.The knowledge stored in database.User can access data through this pages (Search pages).When Search Button clicked on main menu a new page will appear (Figure 4.15)

On this form (Figure 4.15) there are all states, applications.Well users can see, collect the datas easly.They must decide Only 'What do I need' then click button and access knowledge that needed by your own.

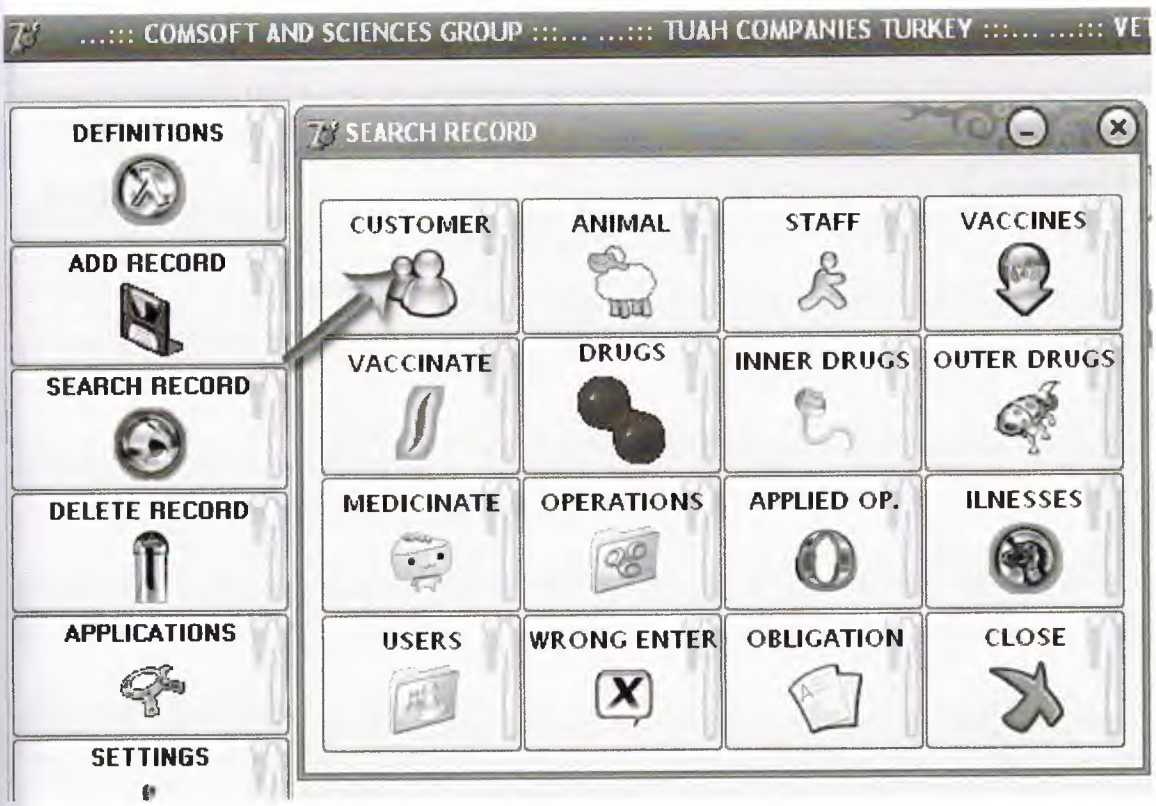

Figure 4.15

If user want to see customer knowledge, he/she must click customer button.Than customer search form will be displayed.Well easly got the data.Figure 4.16 has a customer search page image.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.16 has only one of them.All figure will append end of project as appendix.

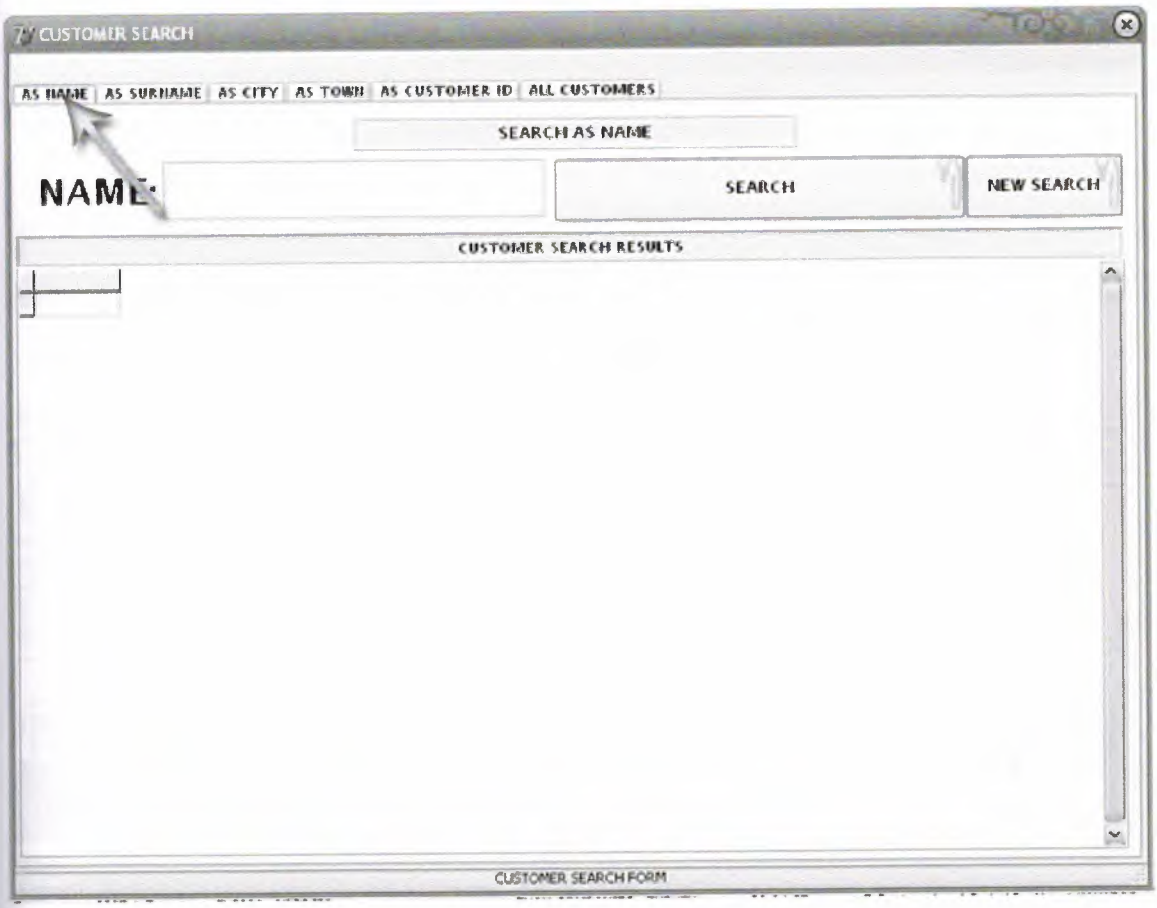

Figure 4.16

If user want to see animal knowledge, he/she must click animal button.Than animal search form will be displayed.Figure 4.17 shows an animal search page.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.17 has only one of them.

When user write character from keyboard the program will check the animals.

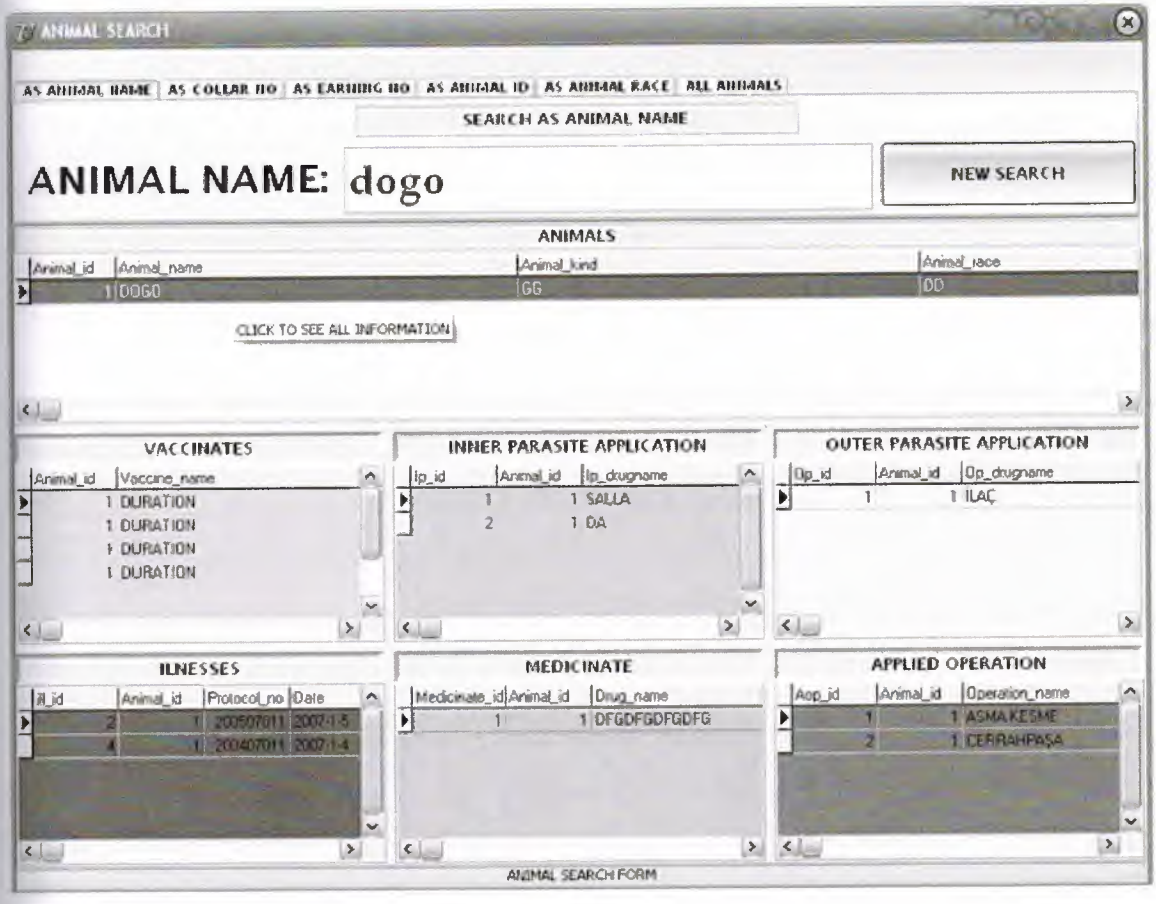

Figure 4.17

If user want to see staff knowledge, he/she must click staff button that is on main page.Than staff search form will be displayed.Figure 4.18 shows an staff search form.

As it seen there are seven criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.18 has only one of them.

When user write character from keyboard the program will check the staff name

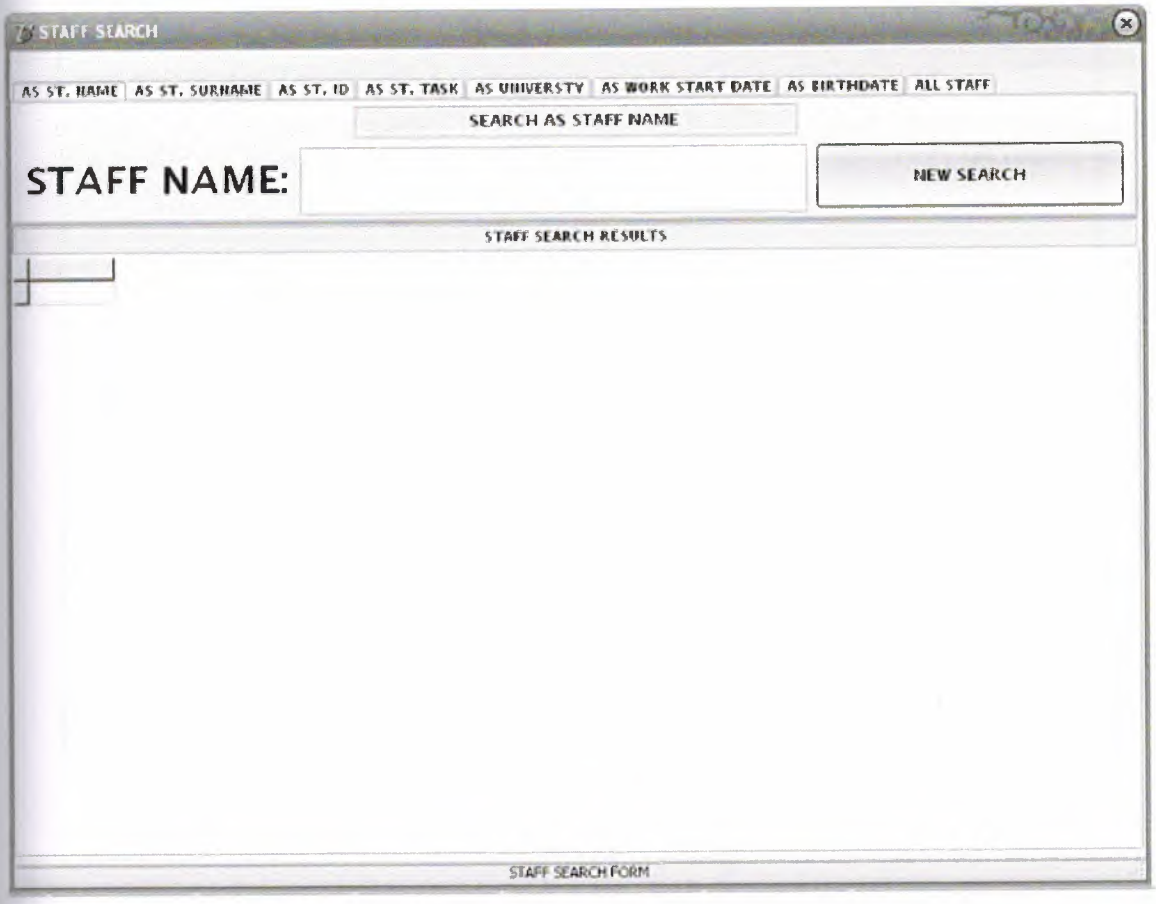

Figure 4.18

If user want to see vaccinate knowledge, he/she must click vaccinate button that is on main page.Than vaccinate search form will be displayed.Figure 4.19 shows an staff search form.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.19 has only one of them.

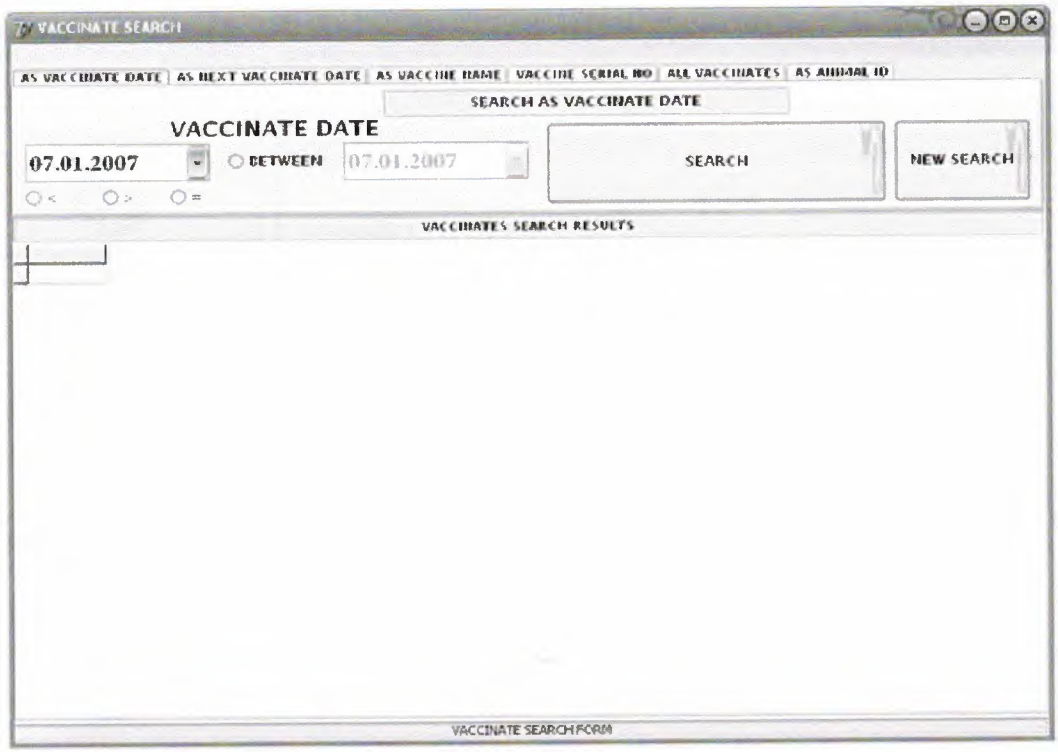

Figure 4.19

When User want to change the settings, he/she must click settings button that is on main page.Than setting page will be displayed.Figure 4.20 shows an settings form

As it seen there are two criteria to make search.Well user can change setting to various situation.User can change form color,can disable or enable skins, disable or enable picture, change skins and picture.

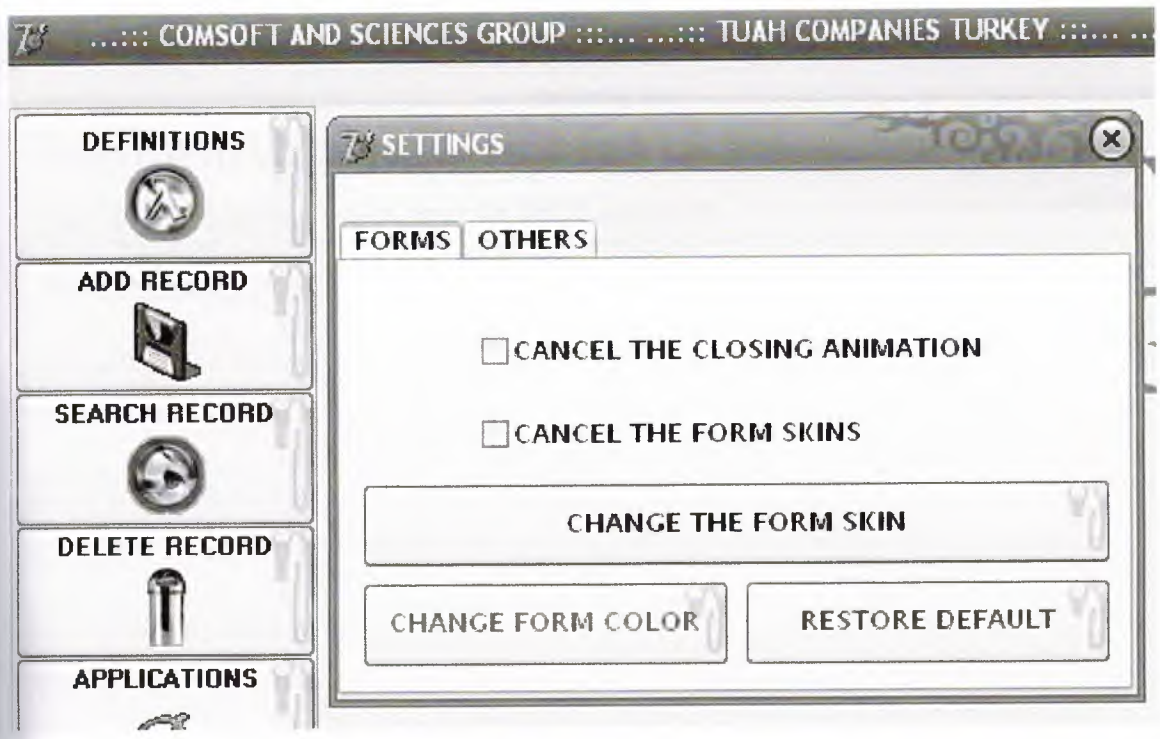

Figure 4.20

If User want to see 'What will I do today?','Which process will be made today ?', he/she must click obligation button that is on search record page.Than obligation page will be displayed.Figure 4.21 shows an settings form

As it seen there are three criteria to make search.Well user can learn to satisfy vaccinate process, inner parasite application process, outer parasite application process.

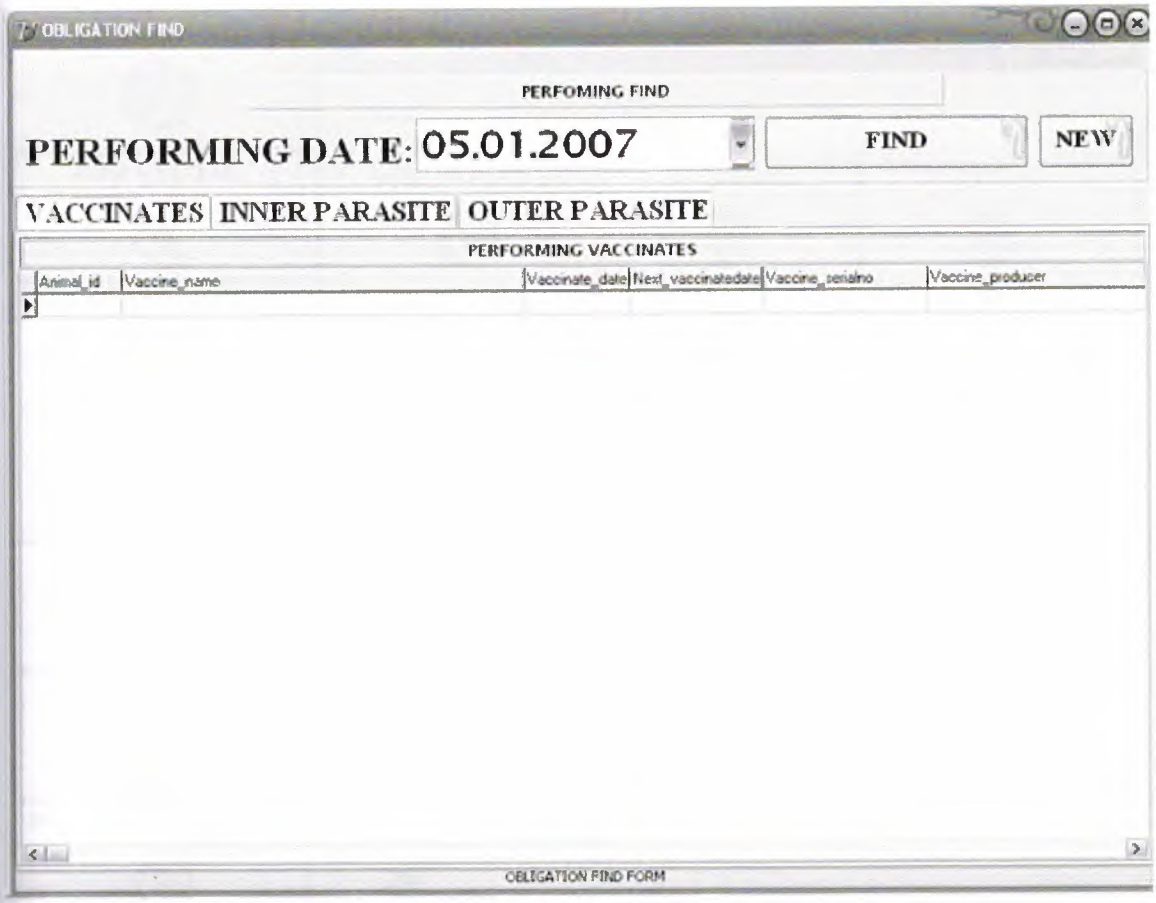

Figure 4.21

When User want to arrvive the amusement. He/she must click amusement button that is on main page.Than amusement selection page will be displayed.Figure 4.22 shows an amusement selection form

As it seen there are six selection object to fun.Well user can arrive various fun.

Figure 4.22

If User want to open a web page. He/she must click internet explorer button that is on main page.Than internet explorer page will be displayed.Figure 4.23 shows that.

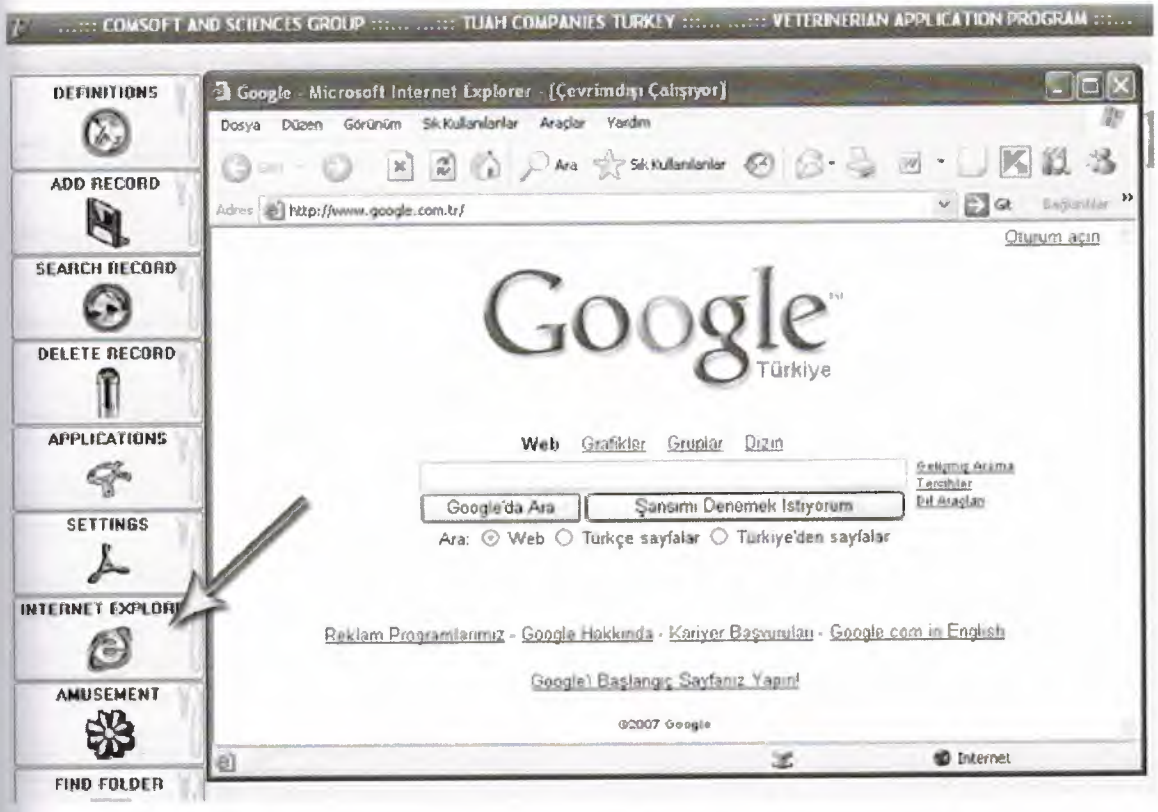

Figure 4.23

If User want toget an windows screen. He/she must click find folder button that is on main page.Than windows screen page will be displayed.Figure 4.24 shows that.In here can find folder, files.And also can process some operation about other application.

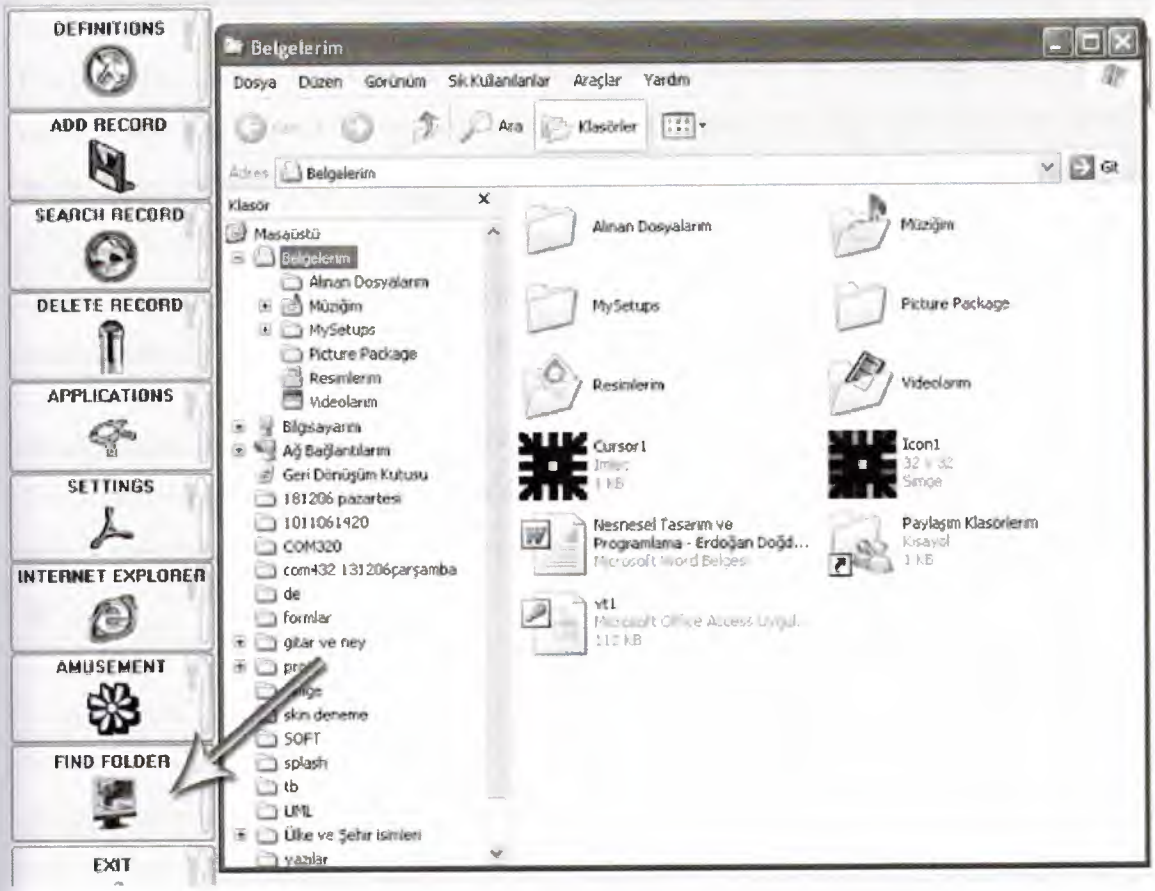
DEFINITIONS
ADD RECORD
SEARCH RECORD
DELETE RECORD
APPLICATIONS
SETTINGS
INTERNET EXPLORER
AMUSEMENT
FIND FOLDER
EXIT
Belgelerim
Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım
Ara
Klasörler
Adres
Belgelerim
Git
Klasör
Masaüstü
Belgelerim
Alınan Dosyalarım
Müziğim
MySetups
Picture Package
Resimlerim
Videolarım
Bilgisayarım
Ağ Bağlantılarım
Geri Dönüşüm Kutusu
181206 pazartesi
1011061420
COM320
com432 131206çarşamba
de
formlar
gitar ve ney
skin deneme
SOFT
splash
tb
UML
Ülke ve Şehir isimleri
yazılar
Alınan Dosyalarım
Müziğim
MySetups
Picture Package
Resimlerim
Videolarım
Cursor1
İmleç
1 KB
Icon1
32 x 32
Simge
Nesnesel Tasarım ve Programlama - Erdoğan Doğd...
Microsoft Word Belgesi
Paylaşım Klasörlerim
Kısayol
1 KB
vt1
Microsoft Office Access Uygul...
212 KB

Figure 4.24

CONCLUSION

MySQL and Delphi are powerful program.When I study with these two program,I get fun.Because these program are wonderful.Examination of the data for internal consistency and comparisons with externally available data indicates that the Delphi study appears reliable. However, the study was difficult to carry out owing to difficulties in obtaining answers from possible respondents. Thus, if a larger survey is to be undertaken to include all building components, it is recommended that committed respondents be obtained before devising the survey.

Veterinerian Application program for veterinerian and users act more facility.However Users adapt easly to the program and use it safetly.Nowadays in everywhere, in every job is combined with the computer.Well Veterinerian clinic will combine with this project.

APPENDIX

VETARINERIAN APPLICATION PROGRAM SOURCE CODE

FORM 1 CODES

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, Menus, ExtCtrls, WinSkinData, jpeg, StdCtrls,
  XPMan;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    StatusBar1: TStatusBar;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    Shape1: TShape;
    SkinData1: TSkinData;
    Label1: TLabel;
    Timer1: TTimer;
    Image1: TImage;
    XPManifest1: TXPManifest;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);

    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
```

```
  procedure SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton3MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton4MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton7MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton8MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton9MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton6MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton10MouseMove(Sender: TObject; Shift: TShiftState;
    X, Y: Integer);
  procedure SpeedButton6Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit9, Unit41;

{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
//if (form1.Label1.Top <> 600) then//and (form1.Label1.Top > 1) then
//form1.Label1.Top:=form1.Label1.Top-1;
//if form1.Label1.Top <> 1 then
//form1.Label1.Top:=form1.Label1.Top+1;
FORM1.StatusBar1.Panels[5].Text:=TIMETOSTR(TIME);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Label1.Caption:='COMSOFT and SCIENCES'+#13+'                GROUP';
FORM1.StatusBar1.Panels[1].Text:=DATETOSTR(DATE);
FORM1.StatusBar1.Panels[5].Text:=TIMETOSTR(TIME);
```

```
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM2.SHOW;
end;

procedure TForm1.SpeedButton10Click(Sender: TObject);
begin
form41.CLOSE;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
FORM6.CLOSE;
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
 FORM3.SHOW;
end;



procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
form2.CLOSE;
form3.CLOSE;
form4.CLOSE;
form5.CLOSE;
form6.CLOSE;
form41.close;

end;

procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
```

```
form4.show;
end;

procedure TForm1.SpeedButton8Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
FORM5.SHOW;
end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM7.CLOSE;
form6.show;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM7.SHOW;
end;

procedure TForm1.SpeedButton9Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
 if FileExists('C:\WINDOWS\explorer.exe') then
   winexec('C:\WINDOWS\explorer.exe',sw_shownormal);
end;

procedure TForm1.SpeedButton7Click(Sender: TObject);
```

```
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
  if FileExists('C:\Program Files\Internet Explorer\iexplore.exe') then
    winexec('C:\Program Files\Internet Explorer\iexplore.exe',sw_shownormal);
end;

procedure TForm1.SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
FORM1.SpeedButton1.Hint:='     THIS ACTS TO DEFINE NEW
KNOWLEDGE'+#13+
                  '(STAFF, VACCINE, DRUGS, OPERATIONS, USERS)';

end;

procedure TForm1.SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton2.Hint:='USES TO SAVE NEW RECORD'+#13+
                  '   (CUSTOMER, ANIMAL)';
end;

procedure TForm1.SpeedButton3MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton3.Hint:='USE TO FIND RECORD'+#13+
                  '   (ALL CRITERIA)';
end;

procedure TForm1.SpeedButton4MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton4.Hint:='ACTS TO DELETE RECORD'+#13+
                  '    (ALL CRITERIA)';
end;

procedure TForm1.SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton5.Hint:='     USES TO SAVE NEW APPLICATION'+#13+
                  '(VACCINATE, INNER PARASITE, OUTER PARASITE)'+#13+
                  '(MEDICINATE, APPLIED OPERATIONS, ILNESSES )';
end;
```

```
procedure TForm1.SpeedButton7MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton7.Hint:='USES TO OPEN THE INTERNET EXPLORER';
end;

procedure TForm1.SpeedButton8MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton8.Hint:='USE TO HAVE FUN';
end;

procedure TForm1.SpeedButton9MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton9.Hint:='USES TO SEE WINDOWS FILES OR FOLDERS';
end;

procedure TForm1.SpeedButton6MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
FORM1.SpeedButton6.Hint:='ACTS TO CHANGE THE PROGRAM SETTINGS';
end;

procedure TForm1.SpeedButton10MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
FORM1.SpeedButton10.Hint:='ACTS TO CLOSE THE PROGRAM';
end;

procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
FORM4.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM9.SHOW;
end;

end.
```

FORM 2 CODES

```
unit Unit2;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm2 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton6: TSpeedButton;
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

uses Unit10, Unit11, Unit12, Unit13, Unit14;

{$R *.dfm}

procedure TForm2.SpeedButton6Click(Sender: TObject);
begin
  form2.hide;
end;

procedure TForm2.SpeedButton1Click(Sender: TObject);
begin
  FORM10.SHOW;
  form2.Hide;
end;

procedure TForm2.SpeedButton2Click(Sender: TObject);
begin
  form11.show;
  form2.Hide;
```

```
end;

procedure TForm2.SpeedButton3Click(Sender: TObject);
begin
  FORM12.SHOW;
  FORM2.Hide;
end;

procedure TForm2.SpeedButton4Click(Sender: TObject);
begin
  FORM13.SHOW;
  FORM2.HIDE;
end;

procedure TForm2.SpeedButton5Click(Sender: TObject);
begin
  FORM14.SHOW;
  FORM2.Hide;
end;

end.
```

FORM 3 CODES

```
unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm3 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    SpeedButton11: TSpeedButton;
    SpeedButton12: TSpeedButton;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    SpeedButton15: TSpeedButton;
```

```
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton16: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton15Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton16Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);


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

var
  Form3: TForm3;

implementation

uses Unit1, Unit23, Unit24, Unit25, Unit28, Unit27, Unit29, Unit30, Unit31,
  Unit32, Unit26, Unit33, Unit34, Unit35, Unit36, Unit37;

{$R *.dfm}


procedure TForm3.SpeedButton1Click(Sender: TObject);
begin
  FORM23.SHOW;
  FORM3.Hide;
end;

procedure TForm3.SpeedButton7Click(Sender: TObject);
begin
  FORM24.SHOW;
  FORM3.Hide;
end;
```

```
procedure TForm3.SpeedButton14Click(Sender: TObject);
begin
  FORM25.SHOW;
  FORM3.Hide;
end;

procedure TForm3.SpeedButton15Click(Sender: TObject);
begin
FORM27.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton2Click(Sender: TObject);
begin
FORM28.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton5Click(Sender: TObject);
begin
form29.show;
form3.Hide;
end;

procedure TForm3.SpeedButton9Click(Sender: TObject);
begin
FORM30.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton10Click(Sender: TObject);
begin
FORM31.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton3Click(Sender: TObject);
begin
FORM32.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton4Click(Sender: TObject);
begin
FORM26.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton8Click(Sender: TObject);
begin
```

```
FORM33.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton11Click(Sender: TObject);
begin
FORM34.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton6Click(Sender: TObject);
begin
FORM35.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton12Click(Sender: TObject);
begin
FORM36.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton16Click(Sender: TObject);
begin
FORM3.Hide;
end;

procedure TForm3.SpeedButton13Click(Sender: TObject);
begin
FORM37.SHOW;
FORM3.Hide;
end;

end.
```

FORM 4 CODES

```
unit Unit4;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm4 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses Unit17, Unit18, Unit19, Unit20, Unit21, Unit22;

{$R *.dfm}

procedure TForm4.SpeedButton1Click(Sender: TObject);
begin
  FORM17.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton2Click(Sender: TObject);
begin
  FORM18.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton5Click(Sender: TObject);
begin
  FORM19.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton3Click(Sender: TObject);
begin
  FORM20.SHOW;
```

```
  FORM4.Hide;
end;

procedure TForm4.SpeedButton4Click(Sender: TObject);
begin
  FORM21.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton6Click(Sender: TObject);
begin
form22.show;
form4.Hide;
end;

procedure TForm4.SpeedButton7Click(Sender: TObject);
begin
FORM4.Hide;
end;

end.
```

FORM 5 CODES

```
unit Unit5;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm5 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
```

```
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form5: TForm5;

implementation

{$R *.dfm}

procedure TForm5.SpeedButton1Click(Sender: TObject);
begin
  if FileExists('C:\Program Files\Windows Media Player\wmplayer.exe') then
    winexec('C:\Program Files\Windows Media Player\wmplayer.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton2Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\sol.exe') then
    winexec('C:\WINDOWS\system32\sol.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton3Click(Sender: TObject);
begin
  if FileExists('C:\windows\system32\freecell.exe') then
    winexec('C:\windows\system32\freecell.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton4Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\winmine.exe') then
    winexec('C:\WINDOWS\system32\winmine.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton5Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\calc.exe') then
  winexec('C:\WINDOWS\system32\calc.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton8Click(Sender: TObject);
begin
```

```
  if FileExists('C:\WINDOWS\notepad.exe') then
    winexec('C:\WINDOWS\notepad.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton6Click(Sender: TObject);
begin
  if FileExists('C:\Program Files\MSN Messenger\msnmsgr.exe') then
    winexec('C:\Program Files\MSN Messenger\msnmsgr.exe',sw_shownormal)
  else if FileExists('C:\Program Files\Messenger\msmsgs.exe') then
    winexec('C:\Program Files\Messenger\msmsgs.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton7Click(Sender: TObject);
begin
form5.Hide;
end;

procedure TForm5.SpeedButton9Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\mshearts.exe') then
    winexec('C:\WINDOWS\system32\mshearts.exe', sw_shownormal);
end;

end.
```

FORM 6 CODES

```
unit Unit6;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm6 = class(TForm)
    SpeedButton1: TSpeedButton;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton11: TSpeedButton;
```

```
    SpeedButton12: TSpeedButton;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form6: TForm6;

implementation

uses Unit15, Unit11, Unit12, Unit13, Unit14, Unit16, Unit17, Unit18,
  Unit19, Unit21, Unit20, Unit22;

{$R *.dfm}

procedure TForm6.SpeedButton6Click(Sender: TObject);
begin
FORM6.CLOSE;
end;

procedure TForm6.SpeedButton1Click(Sender: TObject);
begin
  Form15.LbSpeedButton1.Enabled:=FALSE;
  Form15.LbSpeedButton2.Enabled:=FALSE;
  FORM15.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton5Click(Sender: TObject);
begin
  FORM11.SpeedButton2.Enabled:=FALSE;
  FORM11.SpeedButton3.Enabled:=FALSE;
  FORM11.SHOW;
```

```
  FORM6.Close;
end;

procedure TForm6.SpeedButton9Click(Sender: TObject);
begin
  FORM12.SpeedButton2.Enabled:=FALSE;
  FORM12.SpeedButton3.Enabled:=FALSE;
  FORM12.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton4Click(Sender: TObject);
begin
  Form13.LbSpeedButton1.Enabled:=FALSE;
  Form13.LbSpeedButton2.Enabled:=FALSE;
  FORM13.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton7Click(Sender: TObject);
begin
  Form14.LbSpeedButton1.Enabled:=FALSE;
  Form14.LbSpeedButton2.Enabled:=FALSE;
  FORM14.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton8Click(Sender: TObject);
begin
  FORM16.SpeedButton3.Enabled:=FALSE;
  FORM16.SpeedButton4.Enabled:=FALSE;
  FORM16.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton2Click(Sender: TObject);
begin
  Form17.LbSpeedButton1.Enabled:=FALSE;
  Form17.LbSpeedButton2.Enabled:=FALSE;
  FORM17.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton11Click(Sender: TObject);
begin
  Form18.SpeedButton3.Enabled:=FALSE;
  Form18.SpeedButton4.Enabled:=FALSE;
  FORM18.SHOW;
  FORM6.Close;
end;
```

```
procedure TForm6.SpeedButton14Click(Sender: TObject);
begin
  Form19.LbSpeedButton1.Enabled:=FALSE;
  Form19.LbSpeedButton2.Enabled:=FALSE;
  FORM19.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton12Click(Sender: TObject);
begin
  Form21.LbSpeedButton1.Enabled:=FALSE;
  Form21.LbSpeedButton2.Enabled:=FALSE;
  FORM21.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton3Click(Sender: TObject);
begin
  Form20.SpeedButton3.Enabled:=FALSE;
  Form20.SpeedButton4.Enabled:=FALSE;
  FORM20.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton13Click(Sender: TObject);
begin
  Form22.SpeedButton3.Enabled:=FALSE;
  Form22.SpeedButton4.Enabled:=FALSE;
  FORM22.SHOW;
  FORM6.Close;
end;

end.
```

FORM 7 CODES

```
unit Unit7;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm7 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
  SpeedButton1: TSpeedButton;
  SpeedButton2: TSpeedButton;
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form7: TForm7;

implementation

uses Unit15, Unit16;

{$R *.dfm}

procedure TForm7.SpeedButton1Click(Sender: TObject);
begin
 FORM15.SHOW;
 FORM7.HIDE;
end;

procedure TForm7.SpeedButton2Click(Sender: TObject);
begin
 FORM16.SHOW;
 FORM7.Hide;
end;

end.
```

FORM 8 CODES

```
unit Unit8;

interface

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

type
 TForm8 = class(TForm)
  Image1: TImage;
 private
  { Private declarations }
 public
```

```
    { Public declarations }
  end;

var
  Form8: TForm8;

implementation

{$R *.dfm}

end.
```

FORM 9 CODES

```
unit Unit9;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, Menus, StdCtrls, jpeg, ExtDlgs;

type
  TForm9 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    PageControl1: TPageControl;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    ColorDialog1: TColorDialog;
    FontDialog1: TFontDialog;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    SpeedButton4: TSpeedButton;
    CheckBox3: TCheckBox;
    CheckBox6: TCheckBox;
    SpeedButton5: TSpeedButton;
    OpenDialog1: TOpenDialog;
    OpenPictureDialog1: TOpenPictureDialog;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    procedure SpeedButton4Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
    procedure CheckBox6Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
  private
    { Private declarations }
```

```
 public
  { Public declarations }
 end;

var
 Form9: TForm9;

implementation

uses Unit1, Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit10, Unit11,
 Unit12, Unit13, Unit14, Unit15, Unit16, Unit17, Unit18, Unit19, Unit20,
 Unit21, Unit22, Unit23, Unit24, Unit25, Unit26, Unit27, Unit28, Unit29,
 Unit30, Unit31, Unit32, Unit33, Unit34, Unit35, Unit36, Unit37, Unit38,
 Unit39, Unit40, Unit41;

{$R *.dfm}

procedure TForm9.SpeedButton4Click(Sender: TObject);
begin
 if form9.CheckBox2.Checked <> true then
 begin
  form9.OpenDialog1.Filter:='Skin Files (skn)|*.skn';
  if form9.OpenDialog1.Execute then
  begin
   form1.SkinData1.LoadFromFile(form9.OpenDialog1.FileName);
   //form1.Label1.Caption:=form1.SkinData1.SkinFile;
  end;
 end
 else
 begin
  beep;
  showmessage('YOU HAVE CANCELED THE SKINS BEFORE');
 end;

end;

procedure TForm9.CheckBox2Click(Sender: TObject);
begin
 if form9.CheckBox2.Checked = true then
  form1.SkinData1.Active:=false;
 if form9.CheckBox2.Checked = false then
  form1.SkinData1.Active:=true;

end;

procedure TForm9.CheckBox6Click(Sender: TObject);
begin
 if form9.CheckBox6.Checked = true then
 begin
  form1.Image1.Visible:=true;
```

```
 end;

 if form9.CheckBox6.Checked = false then
 begin
  form1.Image1.Visible:=false;
 end;
end;

procedure TForm9.SpeedButton5Click(Sender: TObject);
begin
 if form9.CheckBox6.Checked = true then
 begin
  if form9.OpenPictureDialog1.Execute then
   form1.Image1.Picture.LoadFromFile(form9.OpenPictureDialog1.FileName);
 end
 else
 begin
  beep;
  showmessage('YOU HAVE CANCELED WALLPAPERS BEFORE');
 end;
end;

procedure TForm9.SpeedButton1Click(Sender: TObject);
begin
 if form9.ColorDialog1.Execute then
 begin
  form1.Color:=form9.ColorDialog1.Color;
  form2.Color:=form9.ColorDialog1.Color;
  form3.Color:=form9.ColorDialog1.Color;
  form4.Color:=form9.ColorDialog1.Color;
  form5.Color:=form9.ColorDialog1.Color;
  form6.Color:=form9.ColorDialog1.Color;
  form7.Color:=form9.ColorDialog1.Color;
  form9.Color:=form9.ColorDialog1.Color;
  form10.Color:=form9.ColorDialog1.Color;
  form11.Color:=form9.ColorDialog1.Color;
  form12.Color:=form9.ColorDialog1.Color;
  form13.Color:=form9.ColorDialog1.Color;
  form14.Color:=form9.ColorDialog1.Color;
  form15.Color:=form9.ColorDialog1.Color;
  form16.Color:=form9.ColorDialog1.Color;
  form17.Color:=form9.ColorDialog1.Color;
  form18.Color:=form9.ColorDialog1.Color;
  form19.Color:=form9.ColorDialog1.Color;
  form20.Color:=form9.ColorDialog1.Color;
  form21.Color:=form9.ColorDialog1.Color;
  form22.Color:=form9.ColorDialog1.Color;
  form23.Color:=form9.ColorDialog1.Color;
  form24.Color:=form9.ColorDialog1.Color;
  form25.Color:=form9.ColorDialog1.Color;
```

```
    form26.Color:=form9.ColorDialog1.Color;
    form27.Color:=form9.ColorDialog1.Color;
    form28.Color:=form9.ColorDialog1.Color;
    form29.Color:=form9.ColorDialog1.Color;
    form30.Color:=form9.ColorDialog1.Color;
    form31.Color:=form9.ColorDialog1.Color;
    form32.Color:=form9.ColorDialog1.Color;
    form33.Color:=form9.ColorDialog1.Color;
    form34.Color:=form9.ColorDialog1.Color;
    form35.Color:=form9.ColorDialog1.Color;
    form36.Color:=form9.ColorDialog1.Color;
    form37.Color:=form9.ColorDialog1.Color;
    form38.Color:=form9.ColorDialog1.Color;
    form39.Color:=form9.ColorDialog1.Color;
    form40.Color:=form9.ColorDialog1.Color;
    form41.Color:=form9.ColorDialog1.Color;
  end;

end;

procedure TForm9.SpeedButton2Click(Sender: TObject);
begin
form1.color:=clBlack;
form2.color:=clBtnFace;
form3.color:=clBtnFace;
form4.color:=clBtnFace;
form5.color:=clBtnFace;
form6.color:=clBtnFace;
form7.color:=clBtnFace;
form9.color:=clBtnFace;
form10.color:=$004080FF;
form11.color:=$00C08080;
form12.color:=$00400040;
form13.color:=clGray;
form14.color:=clSilver;
form15.color:=$00404080;
form16.color:=clBtnFace;
form17.color:=clMoneyGreen;
form18.color:=$00400000;
form19.color:=clBlack;
form20.color:=clBtnFace;
form21.color:=$00404080;
form22.color:=clInactiveCaptionText;
form23.color:=clBtnFace;
form24.color:=clBtnFace;
form25.color:=clBtnFace;
form26.color:=clBtnFace;
form27.color:=clBtnFace;
form28.color:=clBtnFace;
form29.color:=clBtnFace;
```

```
form30.color:=clBtnFace;
form31.color:=clBtnFace;
form32.color:=clBtnFace;
form33.color:=clBtnFace;
form34.color:=clBtnFace;
form35.color:=clBtnFace;
form36.color:=clBtnFace;
form37.color:=clBtnFace;
form38.color:=clBtnFace;
form39.color:=$00D0A5A4;
form40.color:=clBtnFace;
form41.color:=clBtnFace;
end;

end.
```

FORM 10 CODES

```
unit Unit10;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Mask, Menus, DB, ADODB, Buttons, Grids,
  DBGrids, LbSpeedButton, ExtCtrls;

type
  TForm10 = class(TForm)
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Label8: TLabel;
```

```
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Edit4: TEdit;
    MaskEdit1: TMaskEdit;
    Memo1: TMemo;
    Edit5: TEdit;
    ComboBox3: TComboBox;
    ComboBox4: TComboBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Edit6: TEdit;
    DateTimePicker3: TDateTimePicker;
    MaskEdit2: TMaskEdit;
    Edit7: TEdit;
    Label18: TLabel;
    Memo2: TMemo;
    StatusBar1: TStatusBar;
    Label19: TLabel;
    Edit8: TEdit;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    DBGrid1: TDBGrid;
    SpeedButton1: TSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    Panel1: TPanel;
    ADOQuery2: TADOQuery;
    DataSource2: TDataSource;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;
```

```
implementation

uses Unit38;

{$R *.dfm}

procedure TForm10.FormCreate(Sender: TObject);
begin
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm10.SpeedButton1Click(Sender: TObject);
begin
 FORM38.SHOW;
 TA:=10;
end;

procedure TForm10.FormShow(Sender: TObject);
begin
form10.DateTimePicker1.Date:=date;
form10.DateTimePicker2.Date:=date;
//form10.DateTimePicker3.Date:=date;
form10.ADOQuery2.Close;
form10.ADOQuery2.SQL.Text:='select * from staff';
form10.ADOQuery2.Open;
end;

procedure TForm10.LbSpeedButton1Click(Sender: TObject);
begin
 form10.ADOQuery1.Close;
 form10.ADOQuery1.SQL.Text:='select * from staff where
Staff_name='+#39+form10.Edit2.Text+#39+' and
Staff_surname='+#39+form10.Edit3.Text+#39+' and
S_birthdate='+#39+datetostr(form10.DateTimePicker2.date)+#39;
 form10.ADOQuery1.Open;

 if form10.ADOQuery1.RecordCount = 0 then
 begin
  if (form10.Edit2.Text <> '') or (form10.Edit3.Text <> '') then
  begin
   form10.ADOQuery1.Close;
   form10.ADOQuery1.SQL.Text:='insert into staff
(Staff_name,Staff_surname,Staff_task,University,Grade_state,S_workstartdate,S_birthd
ate,S_TCidno,S_homephone,S_mobilphone,S_address,S_town,S_city,S_country,S_ema
il,S_web,S_leavingdate,S_note) values
('+#39+Form10.Edit2.Text+#39+','+#39+form10.Edit3.Text+#39+','+#39+form10.Com
boBox1.Text+#39+','+#39+form10.Edit4.Text+#39+','+#39+form10.ComboBox2.Text+
#39+','+#39+datetostr(form10.DateTimePicker1.date)+#39+','+#39+datetostr(form10.D
```

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

VETERINERIAN APPLICATION PROGRAM WITH DELPHI

Graduation Project
COM400

Student: Ahmet KAYABAŞ (20021329)

Supervisor: Mr. Elburus IMANOV

Lefkoşa-2007

TABLE OF CONTENT

ACKNOWLEDGMENT

When people start a new work they get excited.Because who do not know any thing about the future of work.When a time passed human becomes familiar for this work.Afterthat may be borred,maybe want to leave this work.That may be true maybe false.It changes from people to people.But I believe that the important thing in the life do not leave such who should embrace very tightly.When we get this it makes us happy.

In the life what is important for you.Business? Money? Science? Power? Family? Love? Humanity? or purpose of existence? In my opinion first of all aim of existence comes.Rest of all things involved in aim of existence.After that comes Love.The world exists of love.With love person gets power, gains working perseverence .

Well in this project I gained perseverence from ***Allah*** *and from my fiancee.I am happy to complete the task which I had given with blessing of* ***Allah*** *and also I am grateful to my fiancee and all the people in my life who have supported me, advised me. They all the time helped and encouraged me to follow my dreams and ambitions.*

For intellectual support, encouragement I want to thank to my supervisor ***Mr. Elburus Imanov*** *who made this project contributions.*

And thank ***my dearest parents*** *who supported me to continue beyond my undergraduate studies, and also many thanks to my dear familiy who brought me till such meaning days.*

To all my friends, especially ***M.Fethullah Akatay, Selman Kayabaş, Metin Yenigün , Kadir Bekiroğlu and My dear fiancee*** *for sharing wonderful moments, advice, and for making me feel at home and in life. And above, I thank God for giving me stamina and courage to achieve my objectives.*

AHMET KAYABAŞ

ABSTRACT

In the world not only human life is important.In the same time other entity lives with us.We are not alone on the earth.Animals share life with us.Ilnesses are not only for human.In the same time whole alive interested with illnesses.How Doctor is important for us like Veterinerian is important for animals.Todays Doctors use application program.Because of to keep knowledge of patient, to facility diagnosis of illness, to reach background of patient efficiently and easly.

Well Veterinerian application program is important like the program that is used human health.Also much more important then others.Because animal can not keep the illnesses knowledge.And also papers of the animal can lost.

This project has as its goal to develop software, processing information about activities of a veterinerian application software. Software developed in this project like not only for animal.In the same time for staff and for owner of the animal.All records keep in the other Database program.It acts easly and fast access.Veterinerian can keep all records in the program as concentment.

INTRODUCTION

Since human created by the powerful Allah, Human wonder everything.Well who tried to satisfy wonder.Such humanity came to nowadays as develop.Todays everyone says technology perfect developed.Yes that is right.By means of technology all process gained velocity.This development acts to spend time to the people.

Technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.The automation is also became a part of our lives. The people operate with automation systems in everywhere.

Veterinerian Application project which is my project.In this software veterinerian can keep animal knowledge, patient background knowledge of the animal, owner of the animal knowledge.With this software veterinerian will make record process easily and safetly.

In Software there are five types user.They can access to only their task process.In the same time in the program veterinerian can get obligation as daily.The software can be used at every animal clinic easly.

CHAPTER 1

DELPHI

1.1 INTRODUCTION TO DELPHI

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

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

It is a method for predicting future events.

It is a method for generating a quick consensus by a group.

It is the use of a survey to collect information.

It is the use of anonymity on the part of the participants.

It is the use of voting to reduce the need for long discussions.

It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing detailed critical examination and discussion, not at forcing a quick

compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective:

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

Additional opportunity has been added by the introduction of Computer Mediated Communication Systems (Hiltz and Turoff, 1978; Rice and Associates, 1984; Turoff, 1989; Turoff, 1991). These are computer systems that support group communications in either a synchronous (Group Decision Support Systems, Desanctis et. al., 1987) or an asynchronous manner (Computer Conferencing). Techniques that were developed and refined in the evolution of the Delphi Method (e.g. anonymity, voting) have been incorporated as basic facilities or tools in many of these computer based systems. As a result, any of these systems can be used to carry out some form of a Delphi process or Nominal Group Technique (Delbecq, et. al., 1975).

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

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

1.2 WHAT IS DELPHI?

Delphi is an object oriented, 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 What's more, 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 tools.

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!

1.3 WHAT KIND OF PROGRAMMING CAN YOU DO WITH DELPHI?

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

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

Some projects to which Delphi is suited:

- Simple, single user database applications

- Intermediate multi-user database applications

- Large scale multi-tier, multi-user database applications

- Internet applications

- Graphics Applications

- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools

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.

1.4 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 quitcapable ofthis.

1.5 SOME KNOWLEDGE ABOUT DELPHI

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

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

For the purposes of this series I will be using Delphi 7. There are more recent versions available (2005 and 2006) however Delphi 7 should be available inexpensively compared to the new versions which will set you back a lot of money. Delphi 7 will more than likely be available in a magazine for free.

1.5.2 Example: Try First Delphi Program

First thing is first, fire up your copy of Delphi and open the Project > Options menu. To compile a console application you need to change a setting on the Linker tab called 'Generate console application', check the box and click OK. Now select File > Close All if anything is already loaded. Then select File > New > Other > Console Application.

Notice the first line refers to the keyword program. You can rename this to HelloWorld. You can also remove the commented portion enclosed in curly brackets. The uses keyword allows you to list all units that you want to use in the program. At the moment just leave it as it is, SysUtils is all we need.

Your unit should now look like this:

Delphi Code:

```
program HelloWorld;

  {$APPTYPE CONSOLE}

 uses

  SysUtils;

 begin

 end.
```

Now what we have just done is written a program, it currently doesn't do a thing however. Hit the run button and see the result. Now wasn't that completely worthless.

Luckily this isn't the end of the article so we'll actually have a worthwhile program at the end of it. All we need to do is insert some code in the main procedure we have just made.

Every good programmer's first program was 'Hello World' and you'll be no exception. All we need to do is use the WriteLn procedure to write 'Hello World!' to the console, simple.Notice the semicolon at the end of the line, at the end of any statement you need to add a semicolon. Run the program and see the results…

Now I don't know about you but I saw hello world flash up and go away in a second, if you didn't write the program you wouldn't even know what it said. To solve this problem we need to tell the program to leave the console open until the user is ready to close it. We can use ReadLn for this which reads the users input from the console.

Delphi Code:

```
program HelloWorld;

 {$APPTYPE CONSOLE}

 uses

   SysUtils;

 begin

   WriteLn('Hello World!' + #13#10 + #13#10 +

     'Press RETURN to end...');

   ReadLn;

 end.
```

I have added a few extra things into the 'Hello World' string so the user knows what to do to end the program as it could be a bit confusing. '#13#10' is to insert a carriage

return as 13 and 10 are the ASCII codes for a carriage return followed by a new line feed. ASCII can be inserted in this way into strings.

1.5.2 Delphi Style

Coding style, the way you format your code and the way in which you present it on the page.At the end of the day who cares about my style, I can read it, and Delphi strips all the spaces out of it and doesn't care if I indent. Why waste my time?

Neatly present code which conforms to the accepted standards not only makes your code much easier for you to read and debug but also but any one else who might read your code to help you, or learn from you can do so with ease. After all which code is easier to follow, example 1 or 2?

Delphi Code:

```
// Example 1

procedure xyz();

var

x,y,z,a:integer;

begin

x:=1;y:=2;

for z:=x to y do begin

a:=power(z,y);

showmessage(inttostr(a));

end;
```

```
end;
```

Delphi Code:

```
// Example 2
procedure XYZ();
var
X,Y,Z,A: Integer;
begin
  X := 1;
  Y := 2;
  for Z := X to Y do
  begin
        A := Power(Z, Y);
        ShowMessage(IntToStr(A));
  end; // for end
end; // procedure end
```

Design patterns are frequently recurring structures and relationships in object-oriented design. Getting to know them can help you design better, more reusable code and also help you learn to design more complex systems.

Much of the ground-breaking work on design patterns was presented in the book Design Patterns: Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides. You might also have heard of the authors referred to as "the Gang of Four". If you haven't read this book before and you're designing objects, it's an excellent

primer to help structure your design. To get the most out of these examples, I recommend reading the book as well.

Another good source of pattern concepts is the book Object Models: Strategies, Patterns and Applications by Peter Coad. Coad's examples are more business oriented and he emphasises learning strategies to identify patterns in your own work.

1.6 HOW DELPHI HELPS YOU DEFINE PATTERNS

Delphi implements a fully object-oriented language with many practical refinements that simplify development.

The most important class attributes from a pattern perspective are the basic inheritance of classes; virtual and abstract methods; and use of protected and public scope. These give you the tools to create patterns that can be reused and extended, and let you isolate varying functionality from base attributes that are unchanging.

Delphi is a great example of an extensible application, through its component architecture, IDE interfaces and tool interfaces. These interfaces define many virtual and abstract constructors and operations.

1.6.1 Delphi Examples of Design Patterns

I should note from the outset, there may be alternative or better ways to implement these patterns and I welcome your suggestions on ways to improve the design. The following patterns from the book *Design Patterns* are discussed and illustrated in Delphi to give you a starting point for implementing your own Delphi patterns.

Pattern Name	*Definition*
Singleton	"Ensure a class has only one instance, and provide a global point of access to it."
Adapter	"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't

	otherwise because of incompatible interfaces."
Template Method	"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."
Builder	"Separate the construction of a complex object from its representation so that the same construction process can create different representations."
Abstract Factory	"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."
Factory Method	"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

Note: These definitions are taken from *Design Patterns*.

1.6.2 Pattern: Singleton

1.6.2.1 Definition

"Ensure a class has only one instance, and provide a global point of access to it."

This is one of the easiest patterns to implement.

1.6.2.2 Applications in Delphi

There are several examples of this sort of class in the Delphi VCL, such as TApplication, TScreen or TClipboard. The pattern is useful whenever you want a single global object in your application. Other uses might include a global exception handler, application security, or a single point of interface to another application.

1.6.2.3 Implementation Example

To implement a class of this type, override the constructor and destructor of the class to refer to a global (interface) variable of the class.

Abort the constructor if the variable is assigned, otherwise create the instance and assign the variable.

In the destructor, clear the variable if it refers to the instance being destroyed.

Note: To make the creation and destruction of the single instance automatic, include its creation in the initialization section of the unit. To destroy the instance, include its destruction in an ExitProc (Delphi 1) or in the finalization section of the unit (Delphi 2).

1.6.3 Pattern: Adapter

1.6.3.1 Definition

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

1.6.3.2 Applications in Delphi

A typical example of this is the wrapper Delphi generates when you import a VBX or OCX. Delphi generates a new class which translates the interface of the external control into a Pascal compatible interface. Another typical case is when you want to build a single interface to old and new systems.

Note Delphi does not allow class adaption through multiple inheritance in the way described in Design Patterns. Instead, the adapter needs to refer to a specific instance of the old class.

1.6.3.3 Implementation Example

The following example is a simple (read only) case of a new customer class, an adapter class and an old customer class. The adapter illustrates handling the year 2000 problem, translating an old customer record containing two digit years into a new date format. The client using this wrapper only knows about the new customer class. Translation between classes is handled by the use of virtual access methods for the properties. The old customer class and adapter class are hidden in the implementation of the unit.

1.6.4 Pattern: Template Method

1.6.4.1 Definition

"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

This pattern is essentially an extension of abstract methods to more complex algorithms.

1.6.4.2 Applications in Delphi

Abstraction is implemented in Delphi by abstract virtual methods. Abstract methods differ from virtual methods by the base class not providing any implementation. The descendant class is completely responsible for implementing an abstract method. Calling an abstract method that has not been overridden will result in a runtime error.

1.6.4.3 A typical example of abstraction is the TGraphic class.

TGraphic is an abstract class used to implement TBitmap, TIcon and TMetafile. Other developers have frequently used TGraphic as the basis for other graphics objects such as PCX, GIF, JPG representations. TGraphic defines abstract methods such as Draw, LoadFromFile and SaveToFile which are then overridden in the concrete classes. Other objects that use TGraphic, such as a TCanvas only know about the abstract Draw method, yet are used with the concrete class at runtime.

Many classes that use complex algorithms are likely to benefit from abstraction using the template method approach. Typical examples include data compression, encryption and advanced graphics processing.

1.6.4.4 Implementation Example

To implement template methods you need an abstract class and concrete classes for each alternate implementation. Define a public interface to an algorithm in an abstract base class. In that public method, implement the steps of the algorithm in calls to protected abstract methods of the class. In concrete classes derived from the base class, override each step of the algorithm with a concrete implementation specific to that class.

1.6.5 Pattern: Builder

1.6.5.1 Definition

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

A Builder seems similar in concept to the Abstract Factory. The difference as I see it is the Builder refers to single complex objects of different concrete classes but containing multiple parts, whereas the abstract factory lets you create whole families of concrete classes. For example, a builder might construct a house, cottage or office. You might employ a different builder for a brick house or a timber house, though you would give them both similar instructions about the size and shape of the house. On the other hand the factory generates parts and not the whole. It might produce a range of windows for buildings, or it might produce a quite different range of windows for cars.

1.6.5.2 Applications in Delphi

The functionality used in Delphi's VCL to create forms and components is similar in concept to the builder. Delphi creates forms using a common interface, through Application.CreateForm and through the TForm class constructor. TForm implements a

common constructor using the resource information (DFM file) to instantiate the components owned by the form. Many descendant classes reuse this same construction process to create different representations. Delphi also makes developer extensions easy. TForm's OnCreate event also adds a hook into the builder process to make the functionality easy to extend.

1.6.5.3 Implementation Example

The following example includes a class TAbstractFormBuilder and two concrete classes TRedFormBuilder and TBlueFormBuilder. For ease of development some common functionality of the concrete classes has been moved into the shared TAbstractFormBuilder class.

1.6.6 Pattern: Abstract Factory

1.6.6.1 Definition

"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."

The Factory Method pattern below is commonly used in this pattern.

1.6.6.2 Applications in Delphi

This pattern is ideal where you want to isolate your application from the implementation of the concrete classes. For example if you wanted to overlay Delphi's VCL with a common VCL layer for both 16 and 32 bit applications, you might start with the abstract factory as a base.

1.6.6.3 Implementation Example

The following example uses an abstract factory and two concrete factory classes to implement different styles of user interface components. TOAbstractFactory is a singleton class, since we usually want one factory to be used for the whole application.

At runtime, our client application instantiates the abstract factory with a concrete class and then uses the abstract interface. Parts of the client application that use the factory don't need to know which concrete class is actually in use.

1.6.7 Pattern: Factory Method

1.6.7.1 Definition

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

The Abstact Factory pattern can be viewed as a collection of Factory Methods.

1.6.7.2 Applications in Delphi

This pattern is useful when you want to encapsulate the construction of a class and isolate knowledge of the concrete class from the client application through an abstract interface.

One example of this might arise if you had an object oriented business application potentially interfacing to multiple target DBMS. The client application only wants to know about the business classes, not about their implementation-specific storage and retrieval.

1.6.7.3 Implementation Example

In the Abstract Factory example, each of the virtual widget constructor functions is a Factory Method. In their implementation we define a specific widget class to return.

1.7 KEY ELEMENTS OF DELPHI CLASS DEFINITIONS

1.7.1 Unit Structure

Delphi units (.PAS files) allow declaration of interface and implementation sections. The interface defines the part that is visible to other units using that unit. The keyword *uses* can be added to a unit's interface or implementation section to list the other units that your unit uses. This indicates to the compiler that your unit refers to parts of the used unit's interface. Parts of a unit declared in the implementation section are all private to that unit, i.e. never visible to any other unit. Types, functions and procedures declared in the interface of a unit must have a corresponding implementation, or be declared as external (e.g. a call to a function in a DLL).

1.7.2 Class Interfaces

Classes are defined as types in Delphi and may contain fields of standard data types or other objects, methods declared as functions or procedures, and properties. The type declaration of a class defines its interface and the scope of access to fields, methods and properties of the class. Class interfaces are usually defined in the interface of a unit to make them accessible to other modules using that unit. However they don't need to be. Sometimes a type declaration of a class may be used only within the implementation part of a unit.

1.7.3 Properties

Properties are a specialised interface to a field of a defined type, allowing access control through read and write methods. Properties are not virtual, you can replace a property with another property of the same name, but the parent class doesn't know about the new property. It is however possible to make the access methods of a property virtual.

1.7.4 Inheritance

Delphi's inheritance model is based on a single hierarchy. Every class inherits from TObject and can have only one parent.

A descendant class inherits all of the interface and functionality of its parent class, subject to the scope described below.

Multiple inheritance from more than one parent is not allowed directly. It can be implemented by using a container class to create instances one or more other classes and selectively expose parts of the contained classes.

Private, Protected, Public and Published ScopeScope refers to the visibility of methods and data defined in the interface of a class, i.e. what parts of the class are accessible to the rest of the application or to descendant classes.

The default scope is public, for instance the component instances you add to a form at design time. Public says "come and get me"; it makes the data or method visible to everything at runtime.

Published parts of a class are a specialized form of Public scope. They indicate special behaviour for classes derived from TPersistent. A persistent class can save and restore its published properties to persistent storage using Delphi's standard streaming methods. Published properties also interact with Delphi Object Inspector in the IDE. A class must descend from TPersistent in order to use Published. There's also not much point in publishing methods, since you can't store them, although Delphi's compiler doesn't stop you. Published also lets another application access details of the class through Delphi's runtime type information. This would be rarely used, except in Delphi's design time interaction with its VCL.

Encapsulation or information hiding is essential to object orientation, so Protected and Private scope let you narrow the access to parts of a class.

Protected parts are visible only to descendant classes, or to other classes defined in the same unit.

Private parts are visible only to the defining class, or to other classes defined in the same unit.

It's important to note that once something is given public or published scope, it cannot be hidden in descendant classes.

Static, Virtual and Dynamic Methods; Override and Inherited

Methods declared as virtual or dynamic let you change their behaviour using override in a descendant class. You're unlikely to see a virtual method in the private part of a class, since it could only be overridden in the same unit, although Delphi's compiler doesn't stop you from doing this.

Override indicates that your new method replaces the method of the same name from the parent class. The override must be declared with the same name and parameters as the original method.

When a method is overridden, a call to the parent class's method actually executes the override method in the real class of the object.

Static methods on the other hand have no virtual or override declaration. You can replace a method of a class in a descendant class by redeclaring another method, however this is not object oriented. If you reference your descendant class as the parent type and try to call the replaced method, the static method of the parent class is executed. So in most cases, it's a bad idea to replace a static method.

Virtual and dynamic methods can be used interchangeably. They differ only in their treatment by the compiler and runtime library. Delphi's help explains that dynamic methods have their implementation resolved at compile time and run slightly faster, whereas virtual methods are resolved at runtime, resulting in slightly slower access but a smaller compiled program. Virtual is usually the preferred declaration. Delphi's help suggests using dynamic when you have a base class with many descendants that may not override the method.

The inherited directive lets you refer back to a property or method as it was declared in the parent class. This is most often used in the implementation of an override method, to call the inherited method of the parent class and then supplement its behaviour.

1.7.5 Abstract Methods

Abstract is used in base classes to declare a method in the interface and defer its implementation to a descendant class. I.e. it defines an interface, but not the underlying

operation. Abstract must be used with the virtual or dynamic directive. Abstract methods are never implemented in the base class and must be implemented in descendant classes to be used. A runtime error occurs if you try to execute an abstract method that is not overridden. Calling inherited within the override implementation of an abstract method will also result in a runtime error, since there is no inherited behaviour.

1.7.6 Messages

Delphi's handling of Windows messages is a special case of virtual methods. Message handlers are implemented in classes that descend from TControl. I.e classes that have a handle and can receive messages. Message handlers are always virtual and can be declared in the private part of a class interface, yet still allow the inherited method to be called. Inherited in a message handler just uses the keyword inherited, there is no need to supply the name of the method to call.

1.7.7 Events

Events are also an important characteristic of Delphi, since they let you delegate extensible behaviour to instances of a class. Events are properties that refer to a method of another object. Events are not inherited in Delphi 1; Delphi 2 extends this behaviour to let you use inherited in an event. . Inherited in an event handler just uses the keyword inherited, there is no need to supply the name of the method to call.

Events are particularly important to component developers, since they provide a hook for the user of the component to modify its behaviour in a way that may not be foreseen at the time the component is written.

1.7.8 Constructors and Destructors

The constructor and destructor are two special types of methods. The constructor initializes a class instance (allocates memory initialized to 0) and returns a reference (pointer) to the object. The destructor deallocates memory used by the object (but not the memory of other objects created by the object).

Classes descended from TObject have a static constructor, Create, and a virtual destructor Destroy.

TComponent introduces a new public property, the Owner of the component and this must be initialized in the constructor. TComponent's constructor is declared virtual, i.e. it can be overridden in descendant classes.It is essential when you override a virtual constructor or destructor in a TComponent descendant to include a call to the inherited method.

1.8 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.

1.8.1 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.

1.8.2 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 a 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.

1.8.3 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.

1.8.4 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.

1.8.4.1 Standard Components

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox and Tedit, 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 Window's 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.

1.8.4.2 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.

1.8.4.3 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 can't 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.

1.8.4.4 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.

1.8.4.5 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.

1.8.4.6 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.

1.9 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.

1.9.1 Property-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 that 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.

1.9.2 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.

Property type	Object Inspector treatment
Simple	Numeric, character, and string properties appear in the Object Inspector as numbers, characters, and strings, respectively. The user can type and edit the value of the property directly.
Enumerated	Properties of enumerated types (including Boolean) display the value as defined in the source code. The user can cycle through the possible values by double-clicking the value column. There is also a drop-down list that shows all possible values of the enumerated type.
Set	Properties of set types appear in the Object Inspector looking like a set. By expanding the set, the user can treat each element of the set as a Boolean value: True if the element is included in the set or False if it's not included.
Object	Properties that are themselves objects often have their own property editors. However, if the object that is a property also has published properties, the Object Inspector allows the user to expand the list of object properties and edit them individually. Object properties must descend from TPersistent.
Array	Array properties must have their own property editors. The Object Inspector has no built-in support for editing array properties.

For more information on properties, refer to the "Component Writers Guide" which ships with Delphi.

1.9.3 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.

1.9.4 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.

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.

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;
```

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.

1.9.5 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.

1.9.6 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 code 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.

1.9.7 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 it's 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. This task is done automatically by Delphi's design-time environment when you drop a component from the Component Palette onto your form.

CHAPTER 2

DATABASE

Every thing around us has a particular identity. To identify anything system, actor or person in words we need a data or information. So this information is valuable and in this advanced era we can store it in database and access this data by the blink of eye.

For an instant if we go through the definitions of database we may find following definitions.

A database is a collection of related information.

A database is an organized body of related information.

2.1 DEMERITS OF ABSENCE OF DATABASE

A glance on the past will may help us to reveal the drawbacks in case of absence of database.

In the past when there wasn't proper system of database, Much paper work was need to do and to handle great deal of written paper documentation was giant among the problems itself.

In the huge networks to deal with equally bulky data, more workers are needed which affidavit cost much labor expanses.

The old criteria for saving data and making identification was much time consuming such as if we want to search the particular data of a person.

Before the Development of Computer database it was a great problem to search for some thing. Efforts to avoid the headache of search often results in new establishments of data.

Before the development of database it seemed very unsafe to keep the worthy information. In Some situation some big organization had to employee the special persons in order to secure the data.

Before the implementation of database any firm had to face the plenty of difficulties in order to maintain their Management. To hold the check on the expenses of the firm, the manager faced difficulties.

2.2 MERITS OF DATABASE

The modern era is known as the golden age computer sciences and technology. In a simple phrase we can express that the modern age is built on the foundation of database.

If we carefully watch our daily life we can examine that some how our daily life is being connected with database.

There are several benefits of database developments.

Now with the help of computerized database we can access data in a second.

By the development of the database we can make data more secure.

By the development of database we can reduce the cost.

2.3 DATABASE DESIGN

The design of a database has to do with the way data is stored and how that data is related. The design process is performed after you determine exactly what information needs to be stored and how it is to be retrieved.

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

Computerized library systems

Automated teller machines

Flight reservation systems

Computerized parts inventory systems

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query.

Database design is a complex subject. A properly designed database is a model of a business, Country Database or some other in the real world. Like their physical model counterparts, data models enable you to get answers about the facts that make up the objects being modeled. It's the questions that need answers that determine which facts need to be stored in the data model.

In the relational model, data is organized in tables that have the following characteristics: every record has the same number of facts, every field contains the same type of facts (Data) in each record, and there is only one entry for each fact. No two records are exactly the same.

The more carefully you design, the better the physical database meets users' needs. In the process of designing a complete system, you must consider user needs from a variety of viewpoints.

2.4 DATABASE MODELS

Various techniques are used to model data structures. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementation may be possible. An example of this is the relational model: in larger systems the physical implementation often has indexes which point to the data; this is similar to some aspects of common implementations of the network model. But in small relational database the data is often stored in a set of

files, one per table, in a flat, un-indexed structure. There is some confusion below and elsewhere in this article as to logical data model vs. its physical implementation.

2.4.1 Flat Model

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 might be used as a part of a system security database. Each row would have the specific password associated with a specific 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 the basis of the spreadsheet.

2.4.2 Network Model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.

2.4.3 Relational Model

The relational data model was introduced in an academic paper by E.F. Cod 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.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allow tables to be defined that allow duplicate rows an extension (or violation) of the relational model. In common English usage, a DBMS is

called relational if it supports relational operational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, not-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't 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 row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no nature key is suitable, an arbitrary key can be assigned (such as by given 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 can't 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).

2.4.3.1 Why we use a Relational Database Design

Maintaining a simple, so-called flat database consisting of a single table doesn't require much knowledge of database theory. On the other hand, most database worth maintaining are quite a bit more complicated than that. Real life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full-fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million

books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database.

2.5 RELATIONSHIPS BETWEEN TABLES

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many and many-to-many relationships.

2.5.2 One-To-One Relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their tracks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support in a table requires security, placing them in a separate table lets your application restrict to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to these fields.

2.5.3 One-To-Many Relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a Many-To-Many relationship as well.

2.6 DATA MODELING

In information system design, data modeling is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "Data Analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together), than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

2.6.1 Database Normalization

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMS lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case de-normalizations are sometimes used to improve performance, at the cost of reduced consistency.

2.6.2 Primary Key

In database design, a primary key is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions) and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a primary key is a candidate key chosen as the main method of uniquely identifying a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System Numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others. In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

2.6.3 Foreign Key

A foreign key (FK) is a field in a database record under one primary key that points to a key field of another database record in another table where the foreign key of one table refers to the primary key of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail needs not to include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book, or even the book itself. The ISBN is the primary key of the book, and it is used as a foreign key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

2.6.4 Compound Key

In database design, a compound key (also called a composite key) is a key that consists on 2 or more attributes.

No restriction is applied to the attribute regarding their (initial) ownership within the data model. This means that any one, none or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute value.

CHAPTER 3
MYSQL

3.1 INTRODUCTION TO MYSQL

This chapter provides a tutorial introduction to MySQL by showing how to use the mysql client program to create and use a simple database. mysql (sometimes referred to as the ``terminal monitor" or just ``monitor") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you place your queries in a file beforehand, then tell mysql to execute the contents of the file. Both ways of using mysql are covered here.

To see a list of options provided by mysql, invoke it with the --help option:

shell> mysql --help

This chapter assumes that mysql is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If you are the administrator, you will need to consult other sections of this manual.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an already-existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily left out. Consult the relevant sections of the manual for more information on the topics covered here.

3.2 WHAT IS MYSQL?

3.2.1 Definition

MySQL is an open source software relational database management system (RDBMS) which

uses a SQL (Structured Query Language)

SQL is the standard language used for interacting with databases.

3.3 WHY CHOOSE MYSQL?

There are many relational databases available to use, so why choose MySQL?

We are specifically interested in databases which PHP supports; these include Oracle, IBM's DB2 and Microsoft's SQL Server (all of which cost money).

The two main open source (free) alternatives to these are PostgreSQL and MySQL. PostgreSQL is arguably the better of the two, but MySQL is better supported on Windows, and is a popular choice among Web hosts that provide support for PHP.

Here are some of MySQL's advantages

- It's fast
- It's free to use, and commercial licenses are reasonable
- It's easy to use
- It is cross platform
- There is a wide community of technical support
- It's secure
- It supports large databases
- It is designed specifically for web base applications and hence works very well partnered with PHP

3.4 PREPARING THE WINDOWS MYSQL ENVIRONMENT

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL- Max server binaries. Here is a list of the different MySQL servers you can use:

mysqld	Compiled with full debugging and automatic memory allocation checking, symbolic links, InnoDB and DBD tables.
mysql-opt	Optimized binary with no support for transactional tables.
mysqld-nt	Optimized binary for NT with support for named pipes. You can run this version on Win98, but in this case no named pipes are created and you must have TCP/IP installed.
mysqld-max	Optimized binary with support for symbolic links, InnoDB and DBD tables.
mysqld-max-nt	Like mysqld-max, but compiled with support for named pipes.

All of the above binaries are optimized for the Pentium Pro processor but should work on any Intel processor >=i386

In the following circumstance, you will need to use the MySQL configuration file:

- The install/data directories are different than the default 'c:\mysql' and 'c:\mysql\data'.
- If you want to use one of these servers:
  - mysqld.exe
  - mysqld-max.exe
  - mysqld-max-nt.exe
- If you need to tune the server settings.

There are two configuration files with the same function: 'my.cnf' and 'my.ini' file, however, only one of these can/should be used. Both files are plain text. The 'my.cnf' file should be created in the root directory of drive C and the 'my.ini' file in the WinDir directory e.g.: C:\WINDOWS or C:\WINNT. If your PC uses a boot loader where the C drive isn't the boot drive, then your only option is to use the 'my.ini' file. Also note that if you use the WinMySQLAdmin tool, only the

'my.ini' file is used. The '\mysql\bin' directory contains a help file with instructions for using this tool.

Using Notepad, create the configuration file and edit the base section and keys:

[mysqld]

basedir = the_install_path # e.g. 'c:\mysql'

datadir = the_data_path # e.g. 'c:\mysql\data' or 'd:\mydata\data'

If the data directory is other than the default 'c:\mysql\data', you must cut the whole

'\data\mysql' directory and paste it on the your option new directory, e.g. 'd:\mydata\mysql'.

If you want to use the InnoDB transaction tables, you need to manually create two new directories to hold the InnoDB data and log files, e.g. 'c:\ibdata' and 'c:\iblogs'. You will also need to create some extra lines to the configuration file.

If you don't want to use, add the skip-innodb option to the configuration file.

Now you are ready to test starting the server.

3.5 STARTING THE SERVER FOR THE FIRST TIME

Testing from a DOS command prompt is the best thing to do because the server prints messages, so if something is wrong with your configuration, you will see a more accurate error message which will make it easier to identify and fix any problems.

Make sure you're in the right directory *(C:\>cd \mysql\bin),*

 # To install mysqld as a standalone program, enter:

C:\mysql\bin> mysqld-max --standalone

You should see the below print messages:

```
InnoDB: The first specified data file c:\ibdata\ibdata1 did not exist
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 200715200
InnoDB: Database physically writes the file full: wait
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

 # To install mysql as a service (Windows 2000), enter:

C:\mysql\bin> mysqld-nt --install

Now you can start and stop mysqld as follows:

C:\>NET START MySQL C:\>NET STOP MySQL

C:\>NET START MySQL

 # To start the MySQL Monitor, enter:

The MySql service is starting.

The MySQL service was started successfully.

C:\>cd \mysql

C:\mysql>bin\mysql

Welcome to the MySQL Monitor. Commands end with ; or \g. Your MySQL connection id

is 1 to server version 3.23.49-nt Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> (enter a command or enter 'QUIT' to quit)

mysql> QUIT Bye

C:\mysql>NET STOP MySQL The MySQL service is stopping.

The MySQL service was stopped successfully.

C:\mysql>

3.6 CONNECTING TO AND DISCONNECTING FROM THE SERVER

To connect to the server, you'll usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

shell> mysql -h host -u user -p

Enter password: ********

The ******** represents your password; enter it when mysql displays the Enter password:

prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

shell> mysql -h host -u user -p

Enter password: ********

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>

The prompt tells you that mysql is ready for you to enter commands.

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

shell> mysql

After you have connected successfully, you can disconnect any time by typing QUIT at the *mysql>*

prompt: mysql> QUIT Bye

You can also disconnect by pressing Control-D.

Most examples in the following sections assume you are connected to the server. They indicate this by the mysql> prompt.

3.7 ENTERING QUERIES

Make sure you are connected to the server, as discussed in the previous section. Doing so will not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how mysql works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown below following the mysql> prompt and hit the RETURN key:

mysql> SELECT VERSION( ), CURRENT_DATE;

version()	CURRENT_DATE
3.22.20a-log	1999-03-19

row in set (0.01 sec)

mysql>

This query illustrates several things about mysql:

A command normally consists of a SQL statement followed by a semicolon. (There are some exceptions where a semicolon is not needed. QUIT, mentioned earlier, is one of them. We'll get to others later.)

When you issue a command, mysql sends it to the server for execution and displays the results, then prints another mysql> to indicate that it is ready for another command.

Mysql displays query output as a table (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), mysql labels the column using the expression itself.

Mysql shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the ``rows in set'' line is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

mysql> SELECT VERSION(), CURRENT_DATE; mysql> select version(), current_date;
mysql> SELECT VERSION(), current_DATE;
*mysql> SELECT SIN(PI()/4), (4+1)*5;*

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

mysql> SELECT VERSION(); SELECT NOW();

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. mysql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, mysql accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

mysql> SELECT USER(),CURRENT_DATE;

USER()	CURRENT_DATE
joesmith@localhost	1999-03-18

In this example, notice how the prompt changes from mysql> to -> after you enter the first line of a multiple-line query. This is how mysql indicates that it hasn't seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you will always be aware of what mysql is waiting for.

If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing \c:

mysql> SELECT USER() \c mysql>

Here, too, notice the prompt. It switches back to mysql> after you type \c, providing feedback to indicate that mysql is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that mysql is in:

Prompt	Meaning
mysql>	Ready for new command.
->	Waiting for next line of multiple-line command.
'>	Waiting for next line, collecting a string that begins with a single quote (`').
">	Waiting for next line, collecting a string that begins with a double quote (`"').

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, mysql waits for more input:

mysql> SELECT USER()
->

If this happens to you (you think you've entered a statement but the only response is a -> prompt), most likely mysql is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and mysql will execute it:

mysql> SELECT USER()
-> ;
USER()
joesmith@localhost

The '> and "> prompts occur during string collection. In MySQL, you can write strings surrounded by either `'` or `"` characters (for example, 'hello' or "goodbye"), and mysql lets you enter strings that span multiple lines. When you see a '> or "> prompt, it means that you've entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. That's fine if you really are entering a multiple-line string, but how likely is that? Not very. More often, the '> and "> prompts indicate that you've inadvertantly left out a quote character. For example:

*mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;*
">

If you enter this SELECT statement, then hit RETURN and wait for the result, nothing will happen. Instead of wondering why this query takes so long, notice the clue provided by the "> prompt. It tells you that mysql expects to see the rest of an unterminated string. (Do you see the error in the statement? The string "Smith is missing the second quote.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type \c in this case, because mysql interprets it as part of the string that it is collecting! Instead, enter the closing quote character (so mysql knows you've finished the string), then type

*\c:mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;*
"> "\c mysql>

The prompt changes back to mysql>, indicating that mysql is ready for a new command.

It's important to know what the '> and "> prompts signify, because if you mistakenly enter an unterminated string, any further lines you type will appear to be ignored by mysql -- including a line containing QUIT! This can be quite confusing, especially if you don't know that you need to supply the terminating quote before you can cancel the current command.

CHAPTER 4

USER MANUAL

In this chapter I will try to explain the veterinerian application program that when it run.If a someone run the program; firstly splash form will be shown for 3000ms like below.

Figure 4.1

After 3000ms entry page (Secure Page) will be shown. (Figure 4.2)

Figure 4.2

On this page (Figure 4.2) the user must enter the user name and password.If user name and password found then the program check the user state for still working or has left.If still working; now the program check the user position for Admin,Veterinerian,Manager,User and Temporary.If the user has left then who can not enter the system; in the same time there is no user name and password program gives three trying chance to enter the system; when is thirth the program will be terminate.

If user is Admin then who can access everything to make on application program; If user is manager then who access everything to make exclusive of wrong password application; If user is veterinerian then who can not access process of user after that who can access; If user is a normal user then who can see some knowledge and can change the program settings; If the user is temporary then who can access only amusements, internet explorer, find folder and drug knowledge.

Then main page comes (Figure 4.3) it is shown below

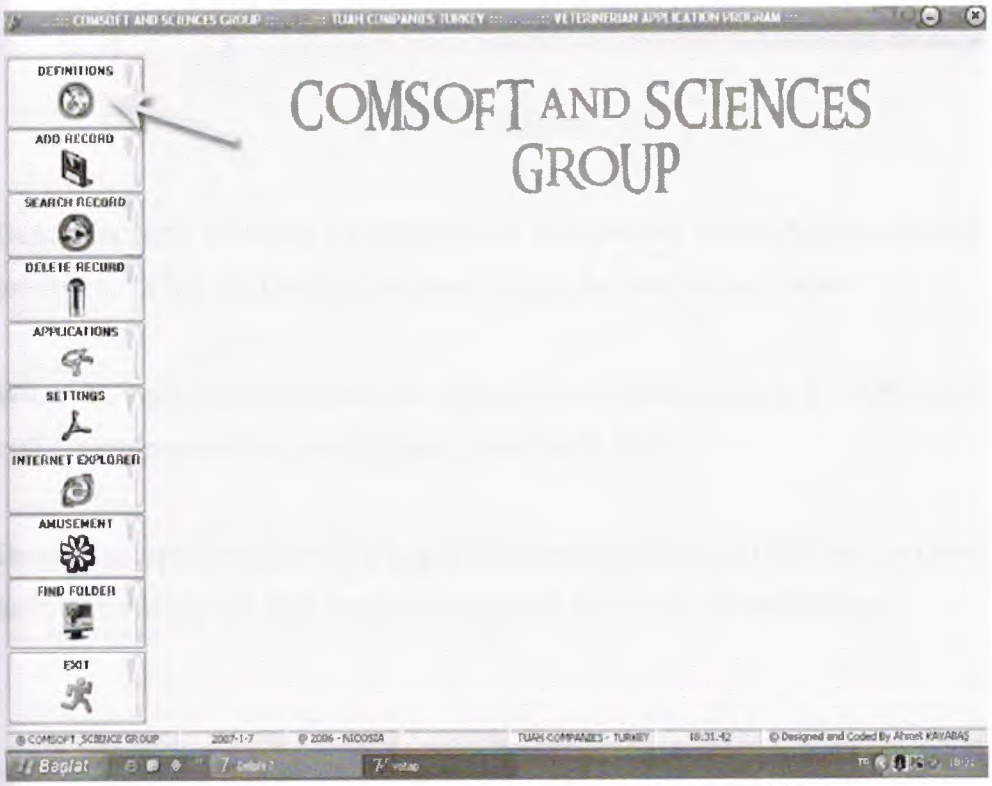

Figure 4.3

This is main page; other pages shown on it. There are ten button on this. User click one of them and access the page that wanted by the user.

When button of definition clicked definition selection page is shown like below figure

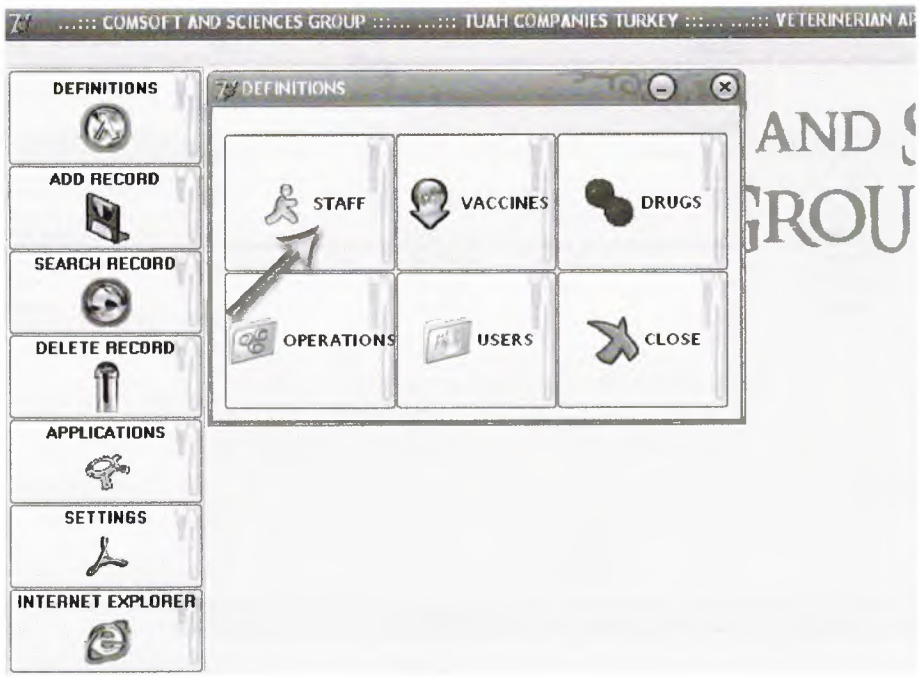

Figure 4.4

Definitions acts to create knowledge that is necessary for application process.User decide process and click the button to access the page for needed application.

When the staff button clicked; the page is shown that is figure 4.5. On this page user can make some process like save, update, delete and find.

On staff record form there is a magnifier button that acts that if there is a person who saved before;knowledge of that person is shown on form with all knowledge.

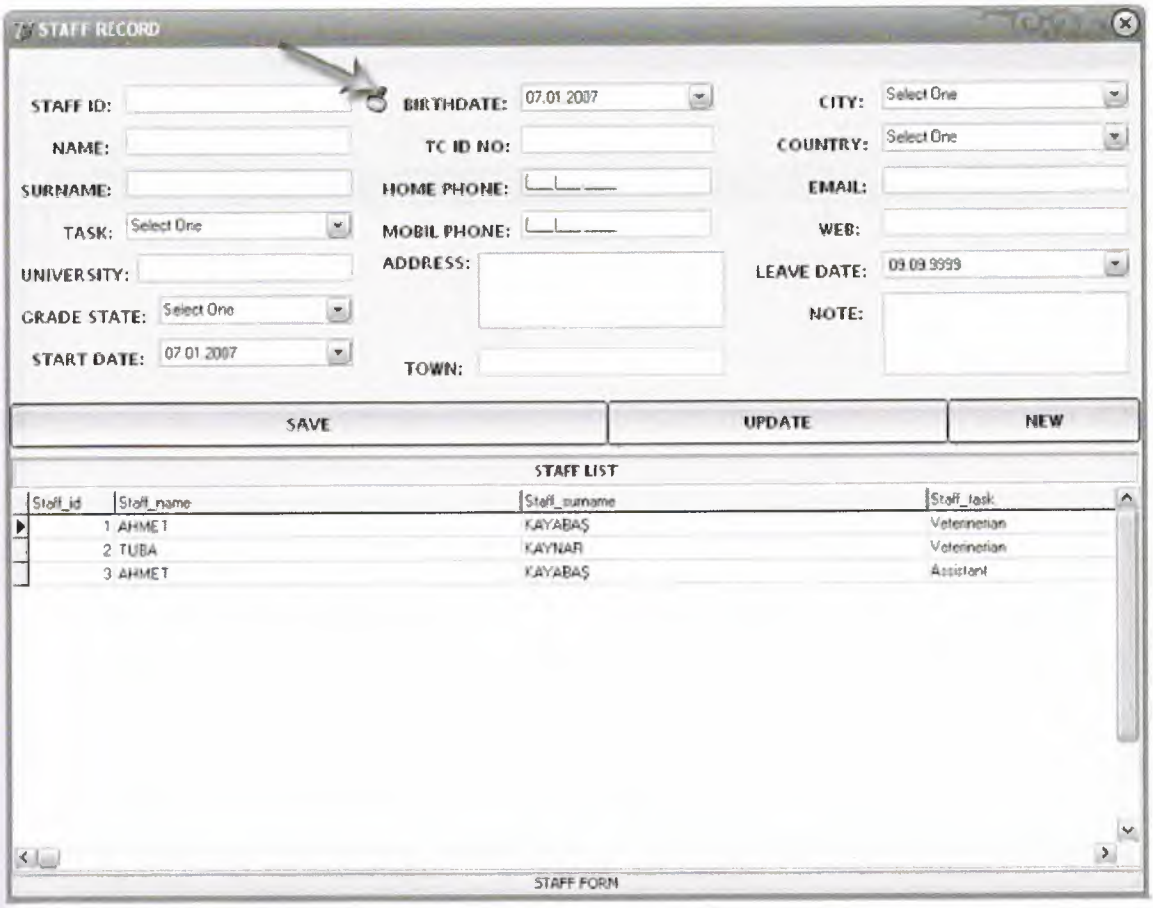

Figure 4.5

When the magnifier button clicked figure 4.6 is shown

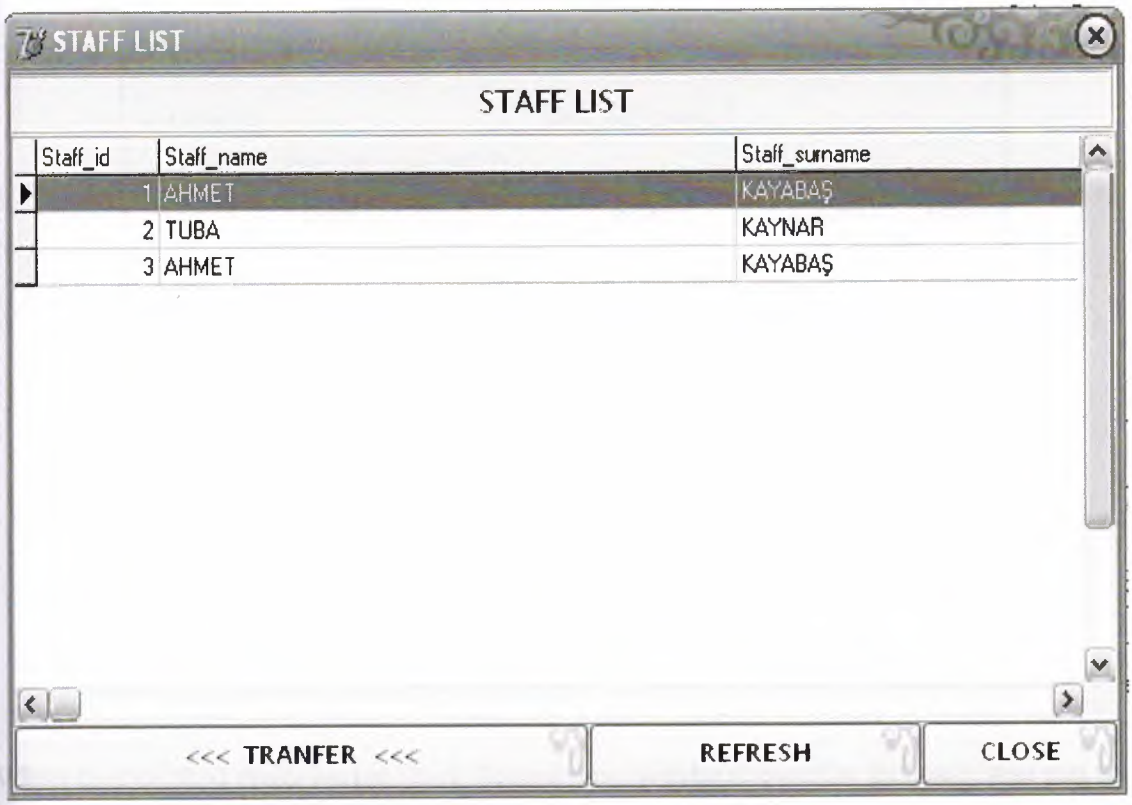

Figure 4.6

When the vaccines record clicked; the page is shown that is figure 4.5. On this page user can add new vaccine, can delete or update it.For process of vaccinates vaccine name called from here (Figure 4.7)

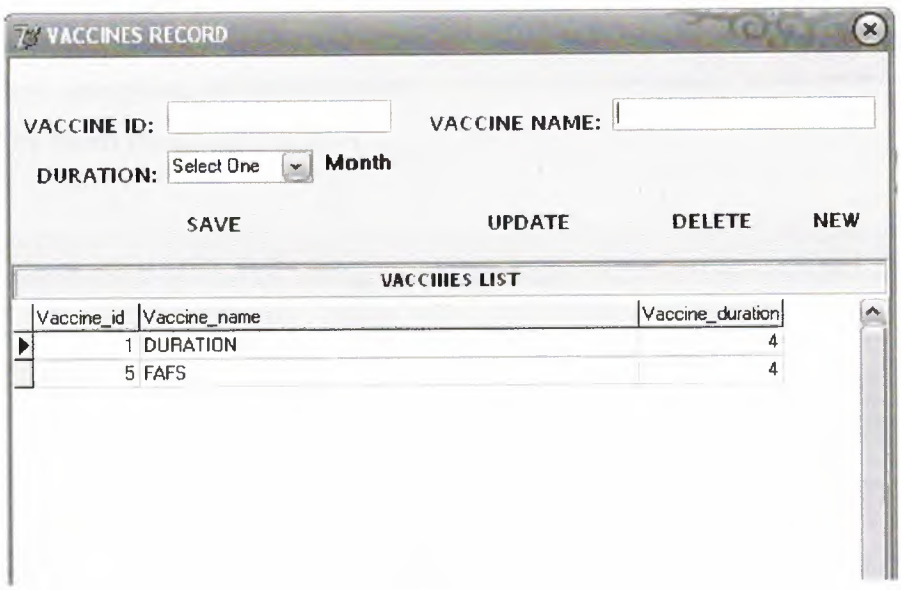

Figure 4.7

When user clicked drugs button Drug Record page will be shown.On this page user can add new drug, delete drug and update old drug record.For process of drug application drug name will absorb from here Figure 4.8

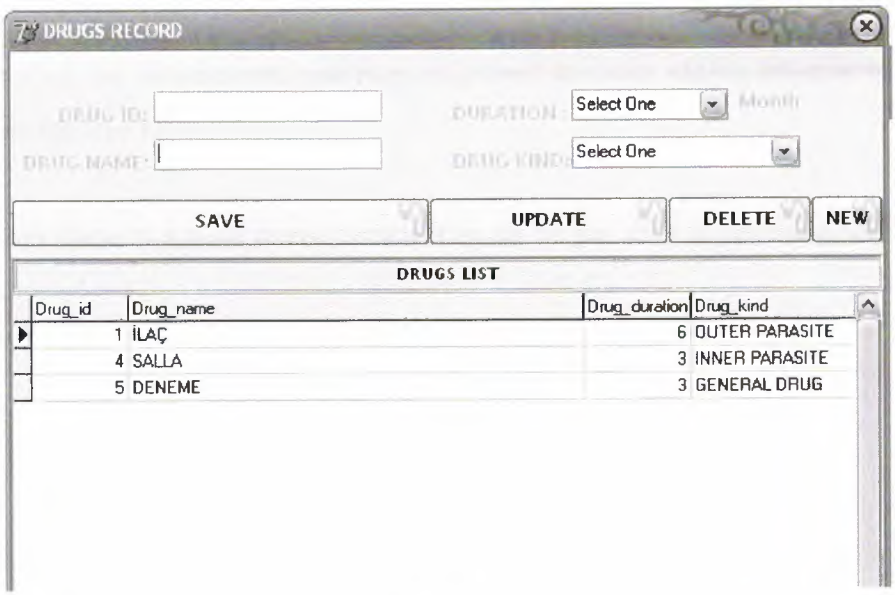

Figure 4.8

When operations button clicked Operation Record page will be shown.On this page user can add new operation, deleteand update operation.For operation application operation name will be taken from here Figure 4.9

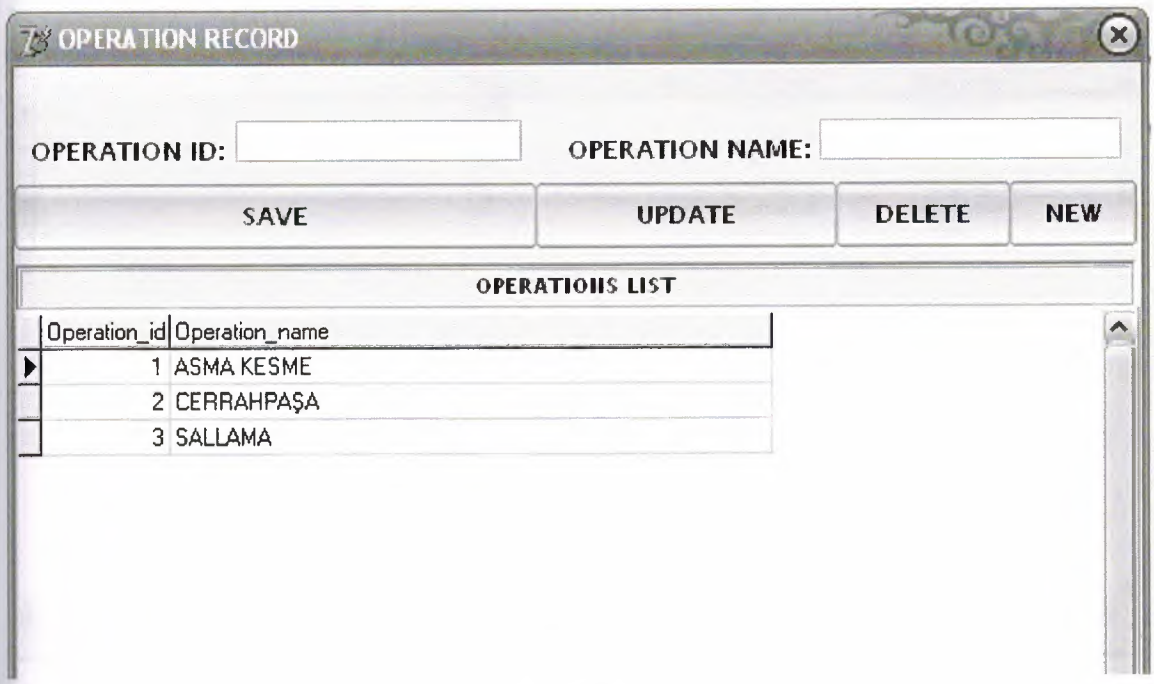

Figure 4.9

If the user click the user button; user page is opened to make adding,deleting and updating user knowledge like Figure 4.10.

On this form there is a mini arrow button.It is act to get staff to combine with users and staff.Because after when a knowledge is needed it stafisfied directly.

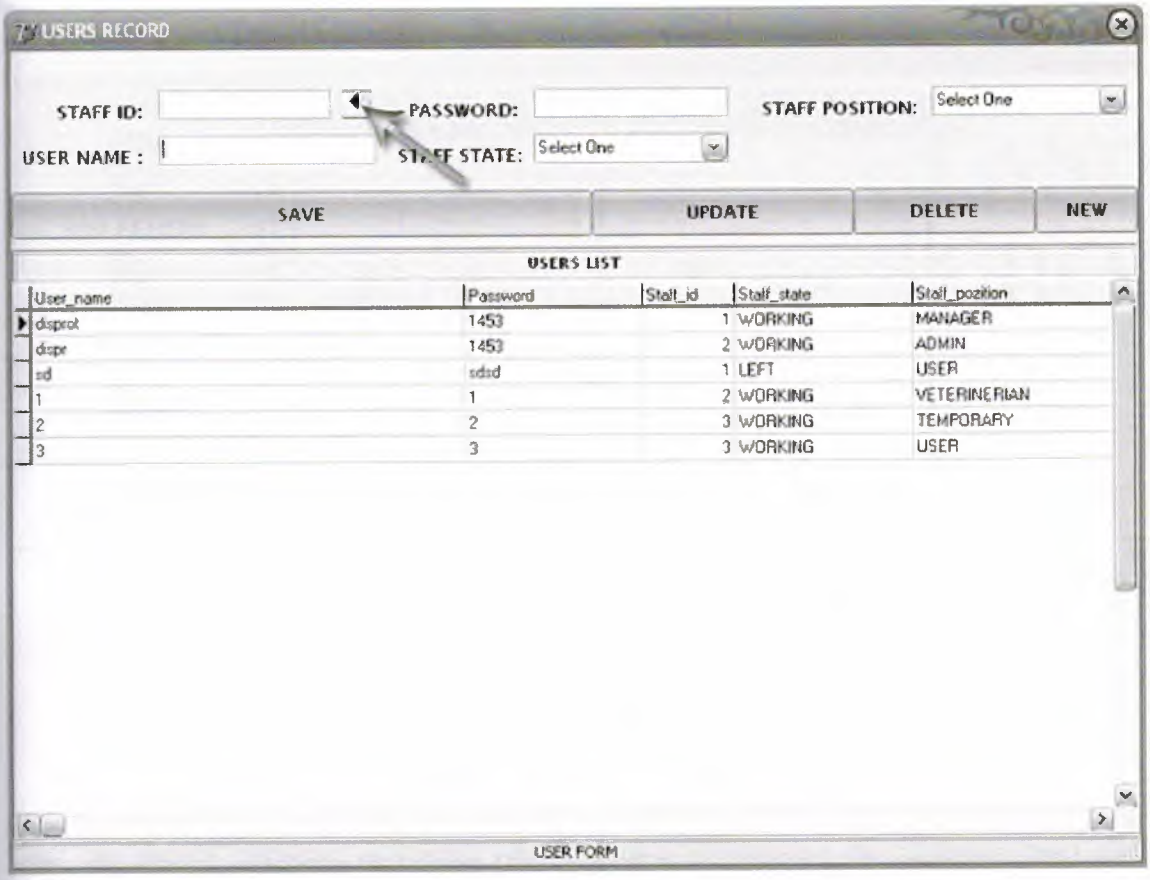

Figure 4.10

ADD Record button thet on main form acts to create knowledge that is necessary for continuity of program.Because Customer and Animal is defined here.User decide process and click the button to access the page for needed application.Figure 4.11 act transaction of this process from main menu.

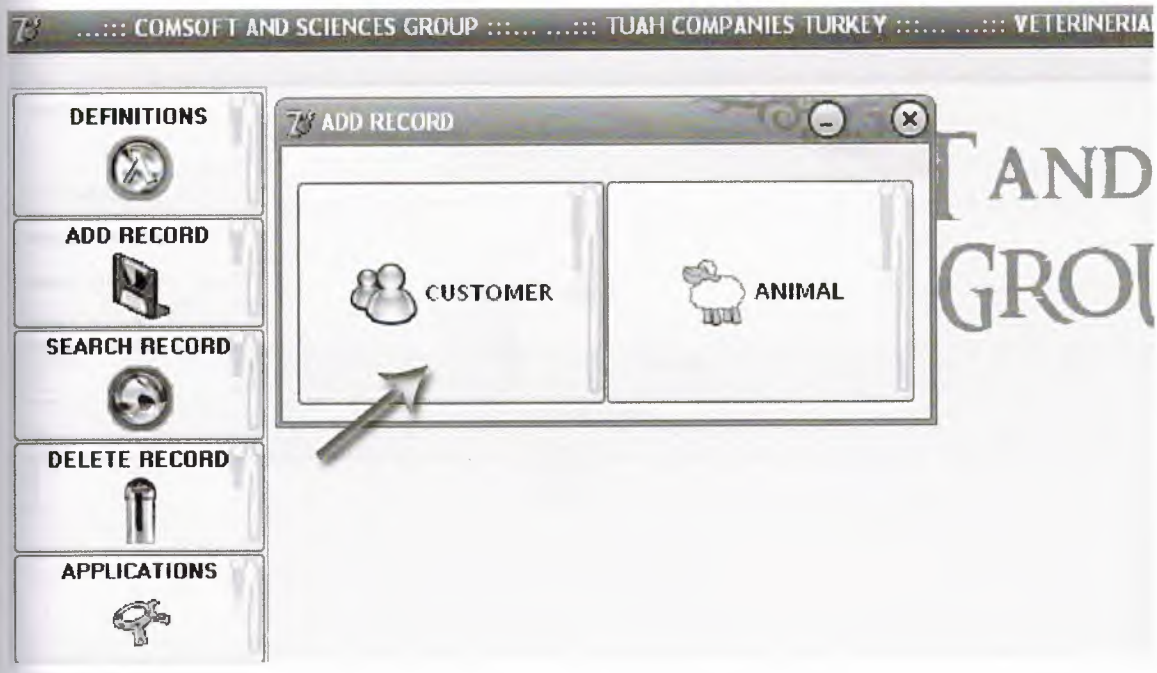

Figure 4.11

User can decide customer or animal.who if decide to continue for customer must click customer button.When he/she made this a new form is shown,Customer record form.With this form user can add an new customer or delete or update an old customer.Update or delete is needed.Well may be customer transferred to other city or transferred to other veterinerian.Figure 4.12 include a customer record page figure

The Program acts all of them easly.Interface is basic as shown.Every user can adapt easly to make operation.

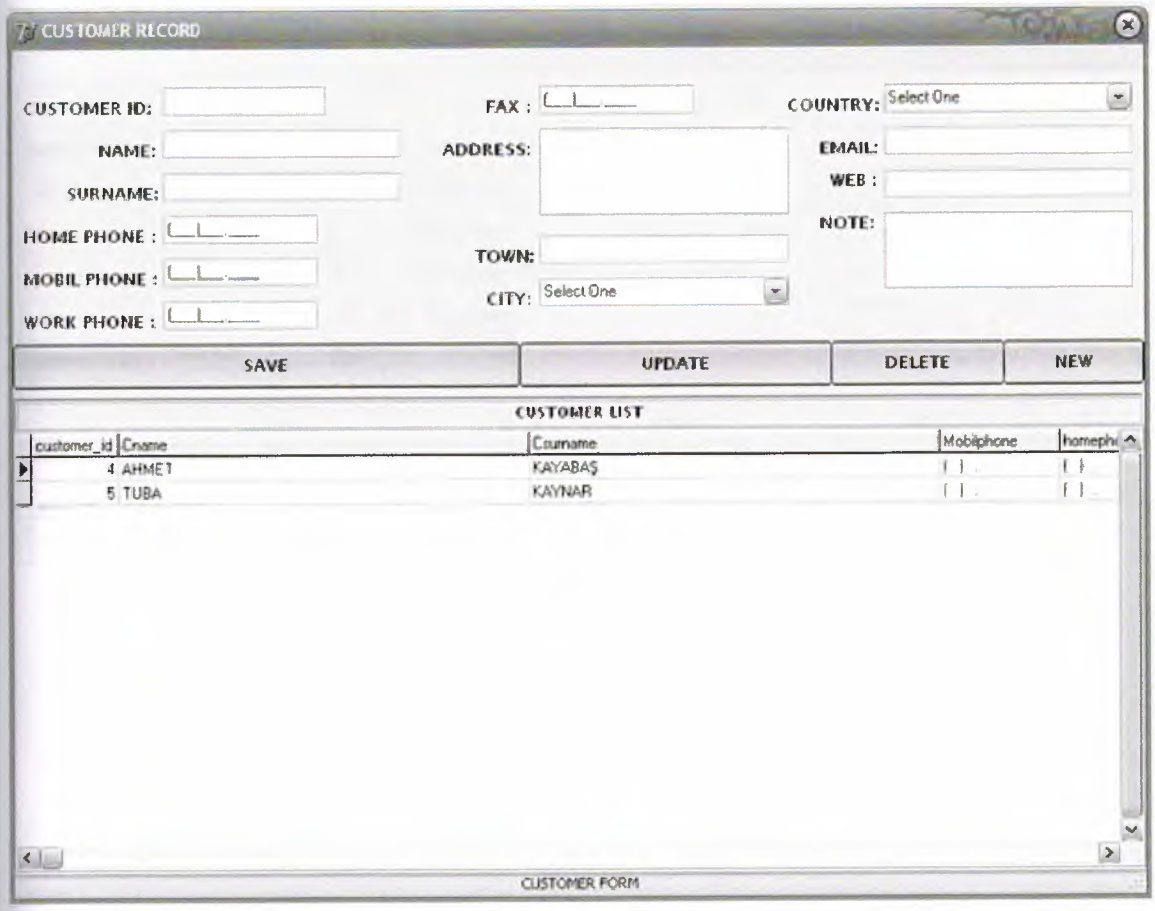

Figure 4.12

If user decided for animal must click animal button on Add Record Form (Figure 4.11). When he/she clicked animal button Animal Record Form will be displayed.With this form user can add an new animal or delete or update an old animal with their owner.Update or delete is needed.Well may be animal transferred to other veterinerian or may be died.

Figure 4.13 shows animal record form.

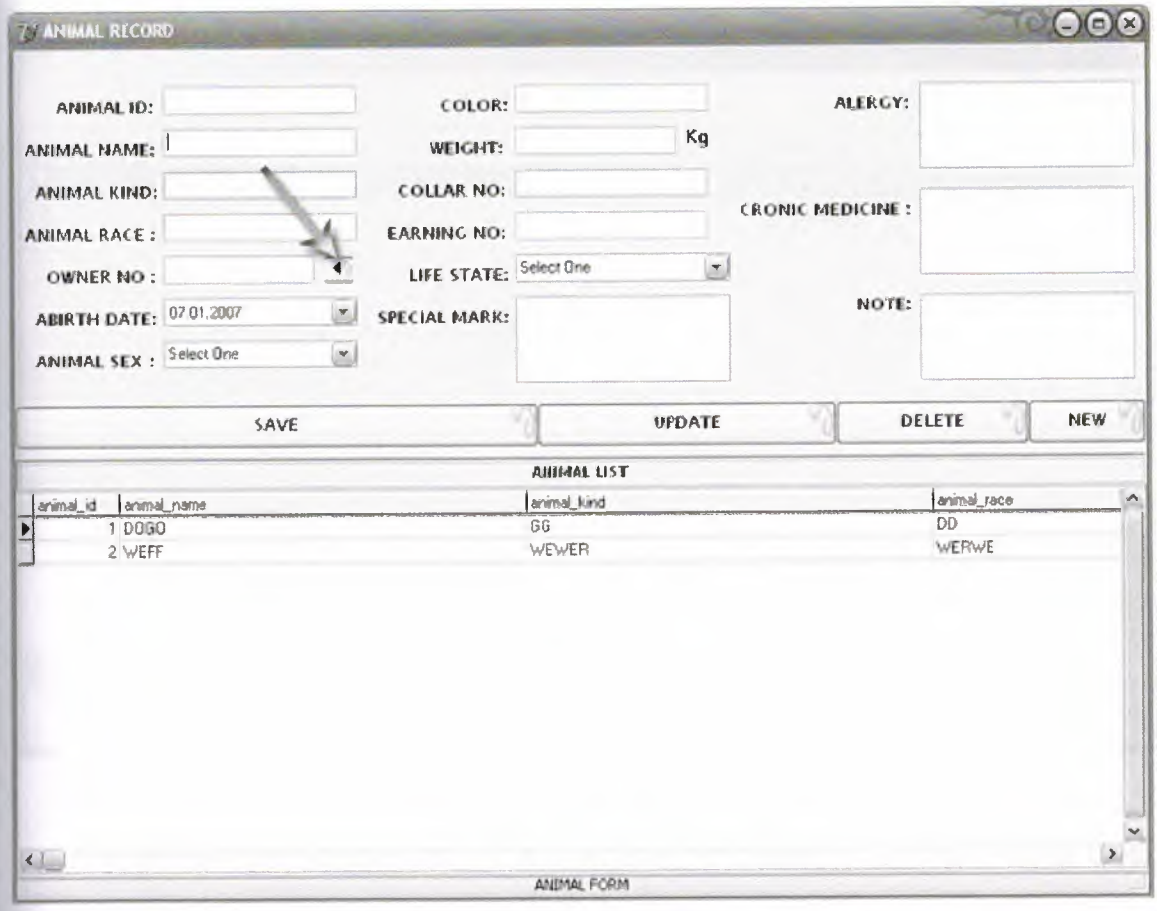

Figure 4.13

On this form (Figure 4.13) there is a mini arrow button.It find owner.Thats why initially customer must save then animal can save.Because as seen owner only called from other form directly.(Figure 4.14)

This Page (Figure 4.14) absorb the knowledge directly database through queries.When it opened datas comes onto dbgrid that on page.

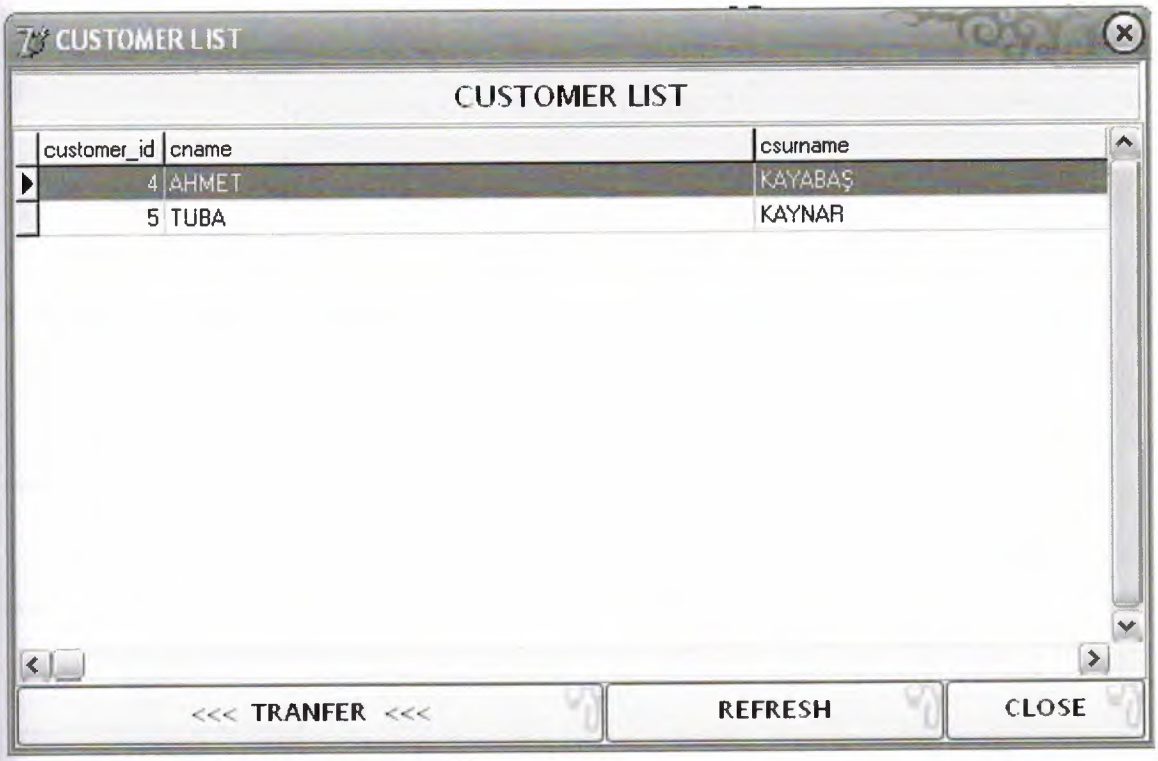

Figure 4.14

Search Record button that on main form acts to show knowledge.The knowledge stored in database.User can access data through this pages (Search pages).When Search Button clicked on main menu a new page will appear (Figure 4.15)

On this form (Figure 4.15) there are all states, applications.Well users can see, collect the datas easly.They must decide Only 'What do I need' then click button and access knowledge that needed by your own.

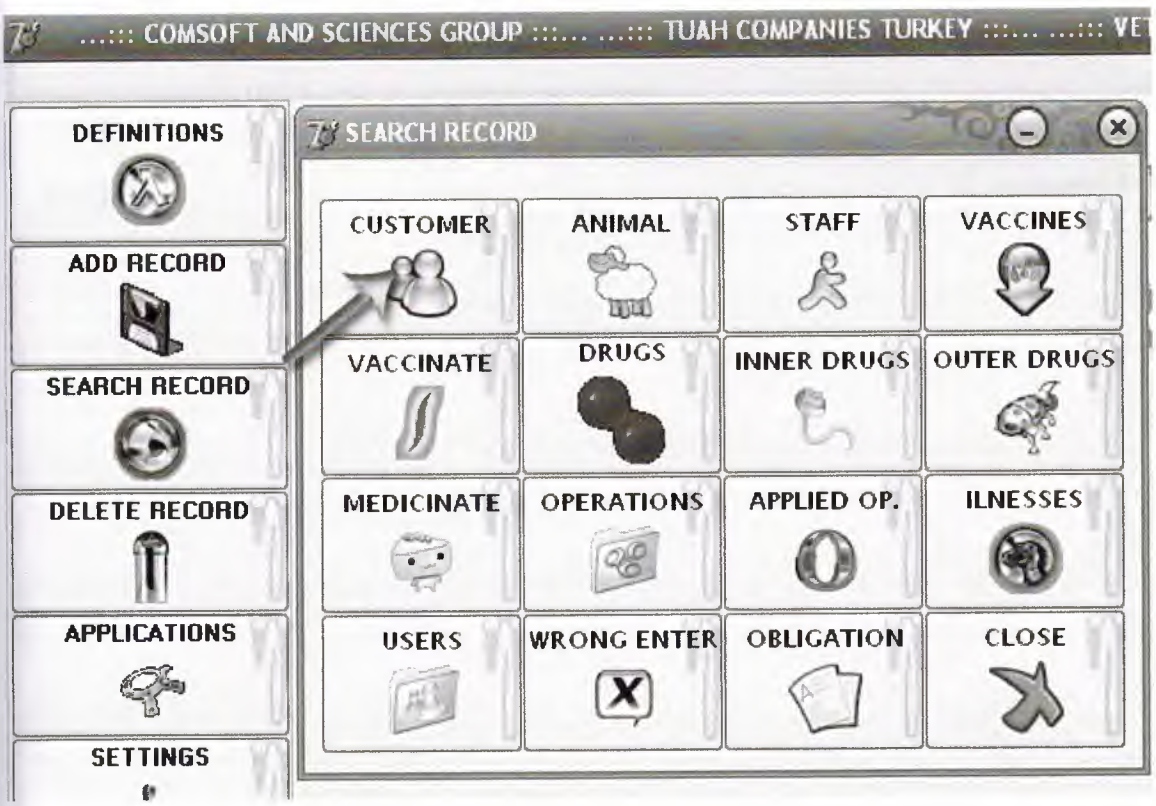

Figure 4.15

If user want to see customer knowledge, he/she must click customer button.Than customer search form will be displayed.Well easly got the data.Figure 4.16 has a customer search page image.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.16 has only one of them.All figure will append end of project as appendix.

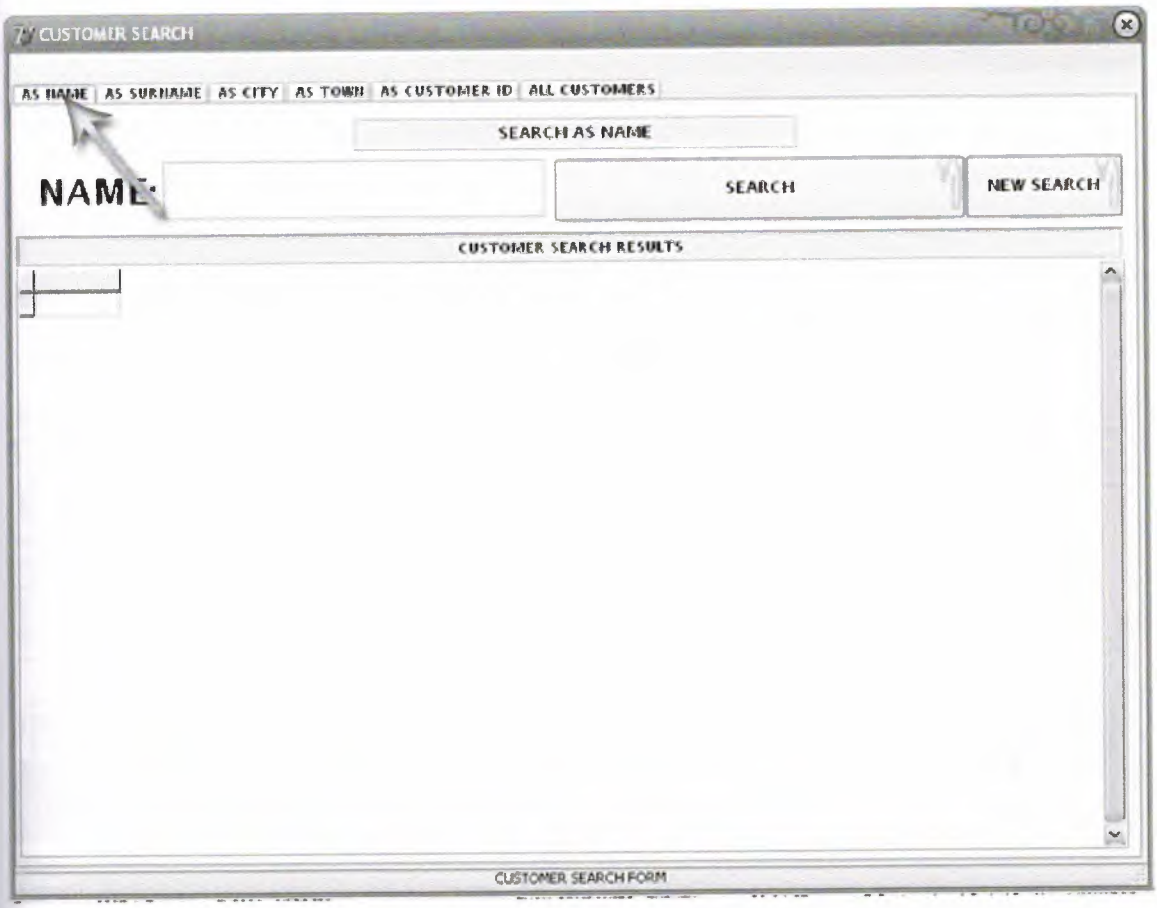

Figure 4.16

If user want to see animal knowledge, he/she must click animal button.Than animal search form will be displayed.Figure 4.17 shows an animal search page.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.17 has only one of them.

When user write character from keyboard the program will check the animals.

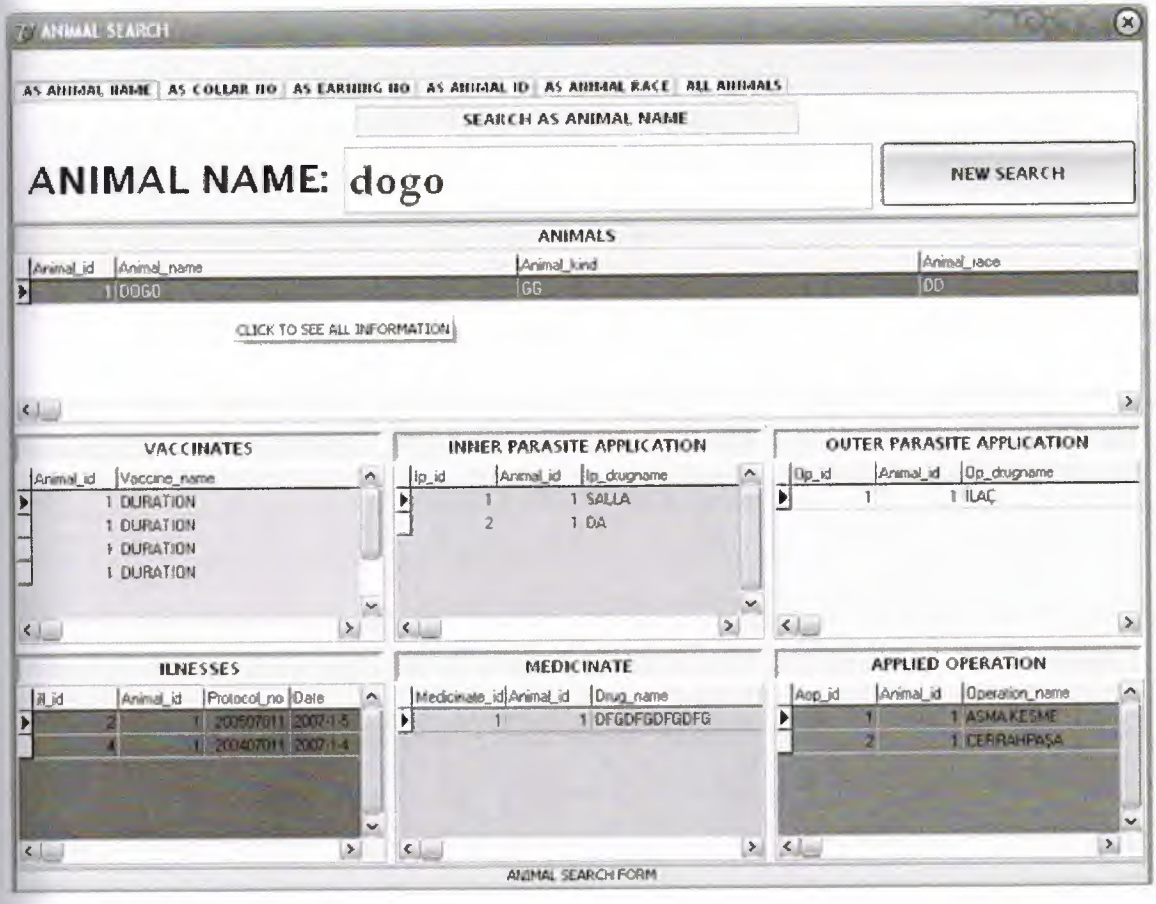

Figure 4.17

If user want to see staff knowledge, he/she must click staff button that is on main page.Than staff search form will be displayed.Figure 4.18 shows an staff search form.

As it seen there are seven criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.18 has only one of them.

When user write character from keyboard the program will check the staff name

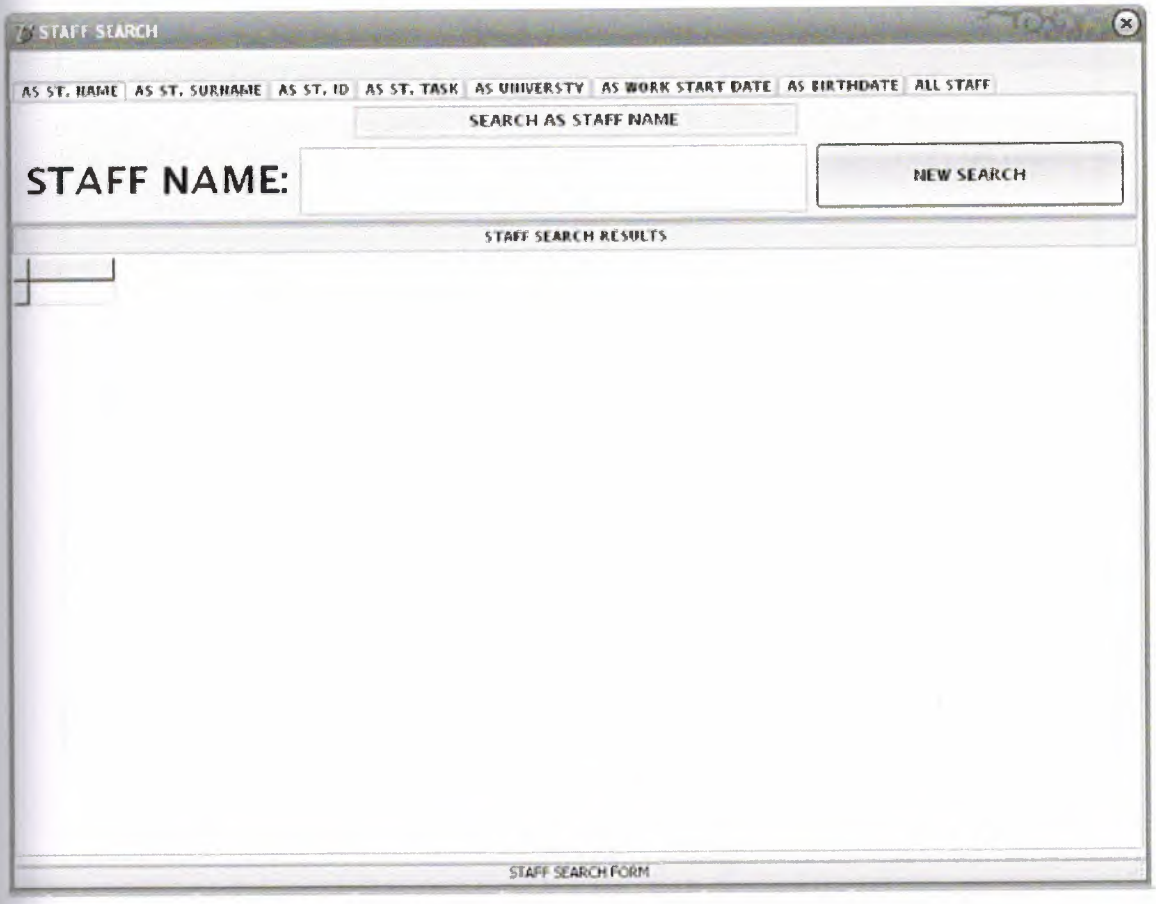

Figure 4.18

If user want to see vaccinate knowledge, he/she must click vaccinate button that is on main page.Than vaccinate search form will be displayed.Figure 4.19 shows an staff search form.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.19 has only one of them.

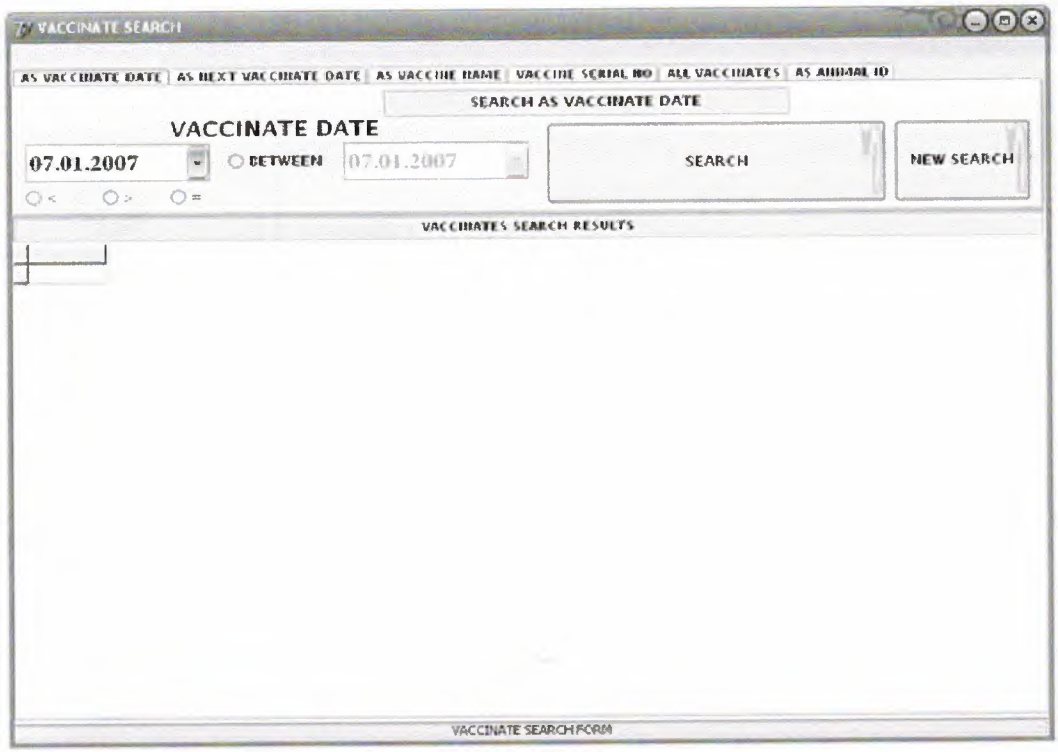

Figure 4.19

When User want to change the settings, he/she must click settings button that is on main page.Than setting page will be displayed.Figure 4.20 shows an settings form

As it seen there are two criteria to make search.Well user can change setting to various situation.User can change form color,can disable or enable skins, disable or enable picture, change skins and picture.

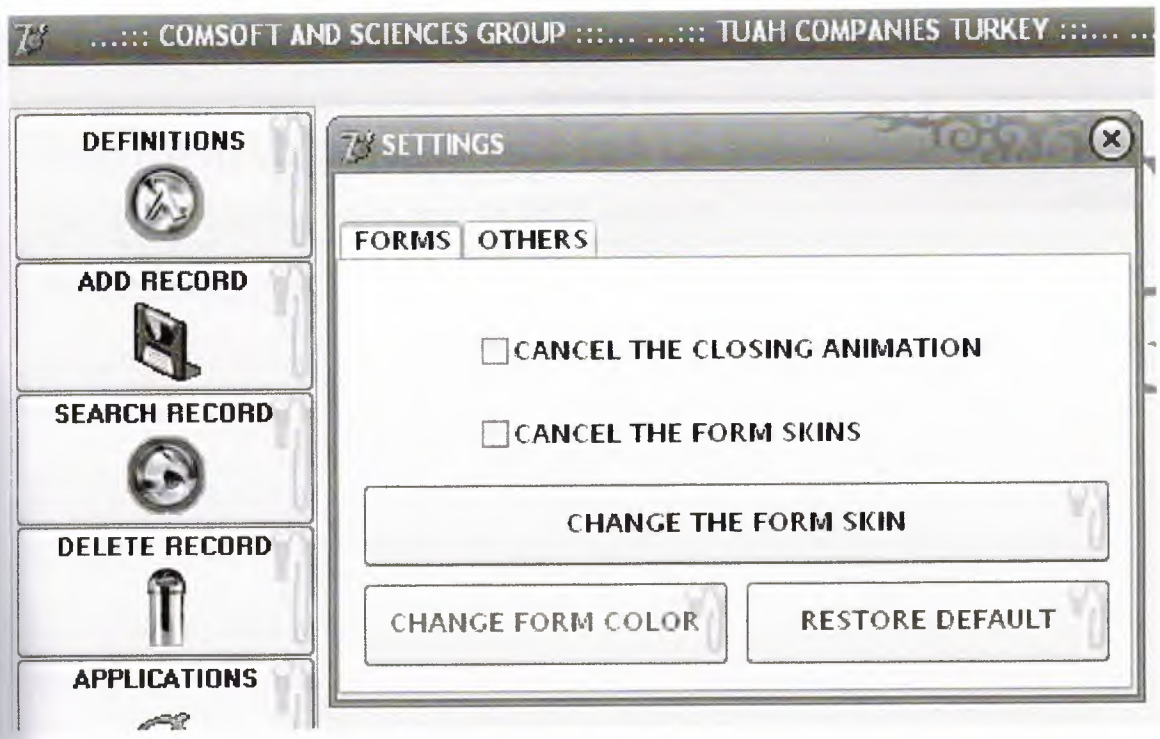

Figure 4.20

If User want to see 'What will I do today?','Which process will be made today ?', he/she must click obligation button that is on search record page.Than obligation page will be displayed.Figure 4.21 shows an settings form

As it seen there are three criteria to make search.Well user can learn to satisfy vaccinate process, inner parasite application process, outer parasite application process.

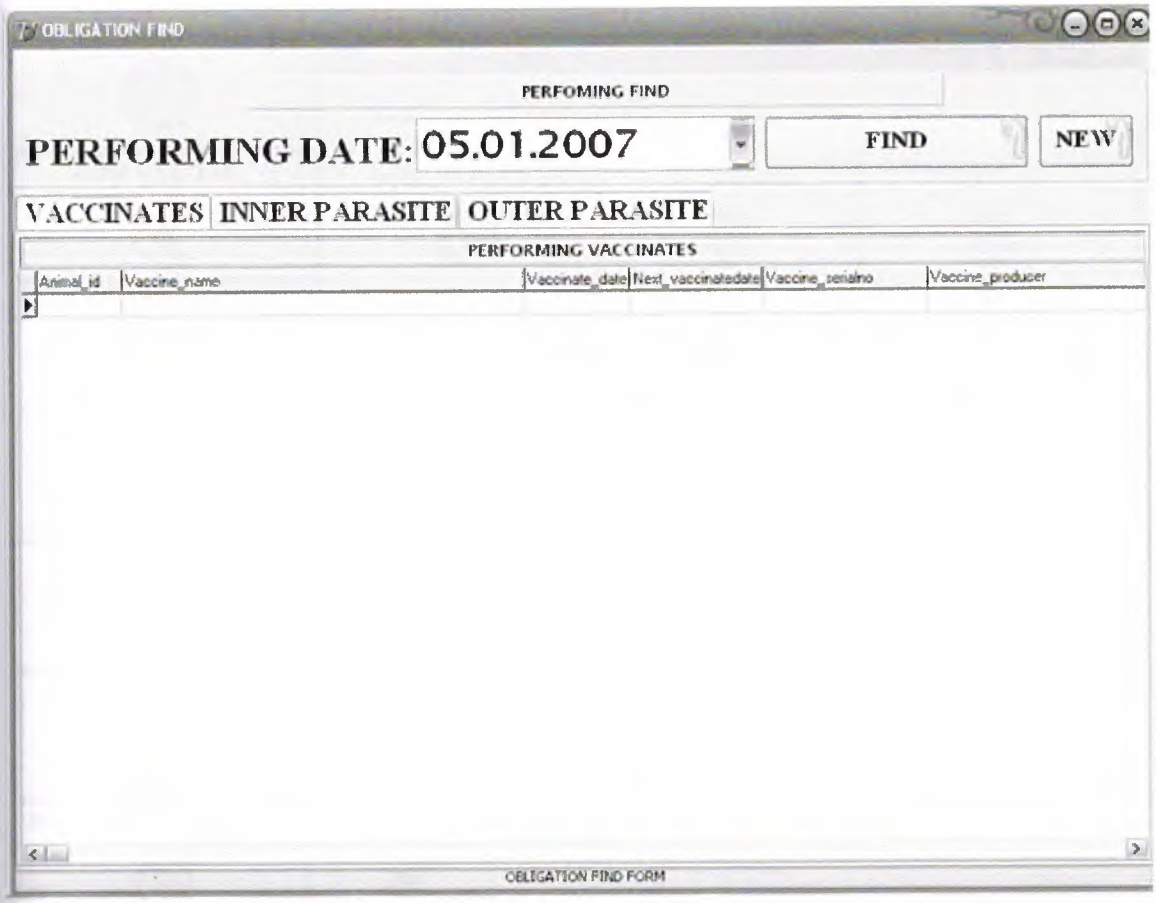

Figure 4.21

When User want to arrvive the amusement. He/she must click amusement button that is on main page.Than amusement selection page will be displayed.Figure 4.22 shows an amusement selection form

As it seen there are six selection object to fun.Well user can arrive various fun.

Figure 4.22

If User want to open a web page. He/she must click internet explorer button that is on main page.Than internet explorer page will be displayed.Figure 4.23 shows that.

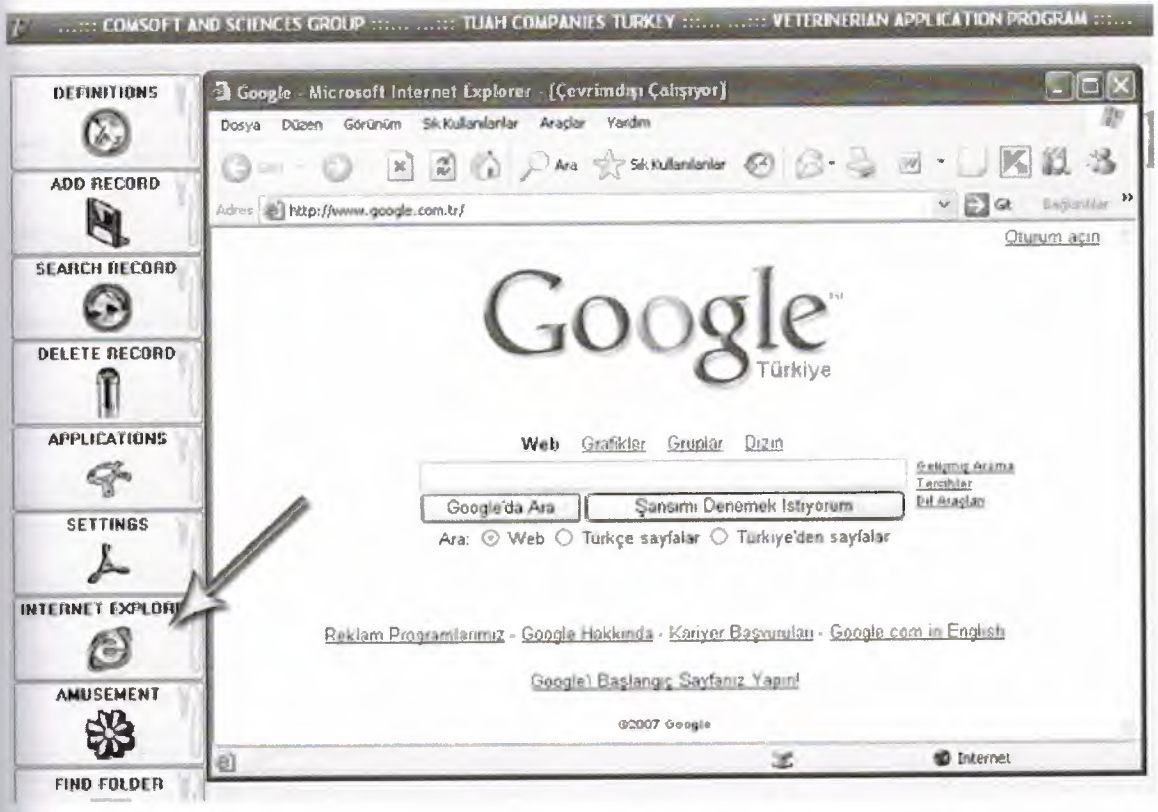

Figure 4.23

If User want toget an windows screen. He/she must click find folder button that is on main page.Than windows screen page will be displayed.Figure 4.24 shows that.In here can find folder, files.And also can process some operation about other application.

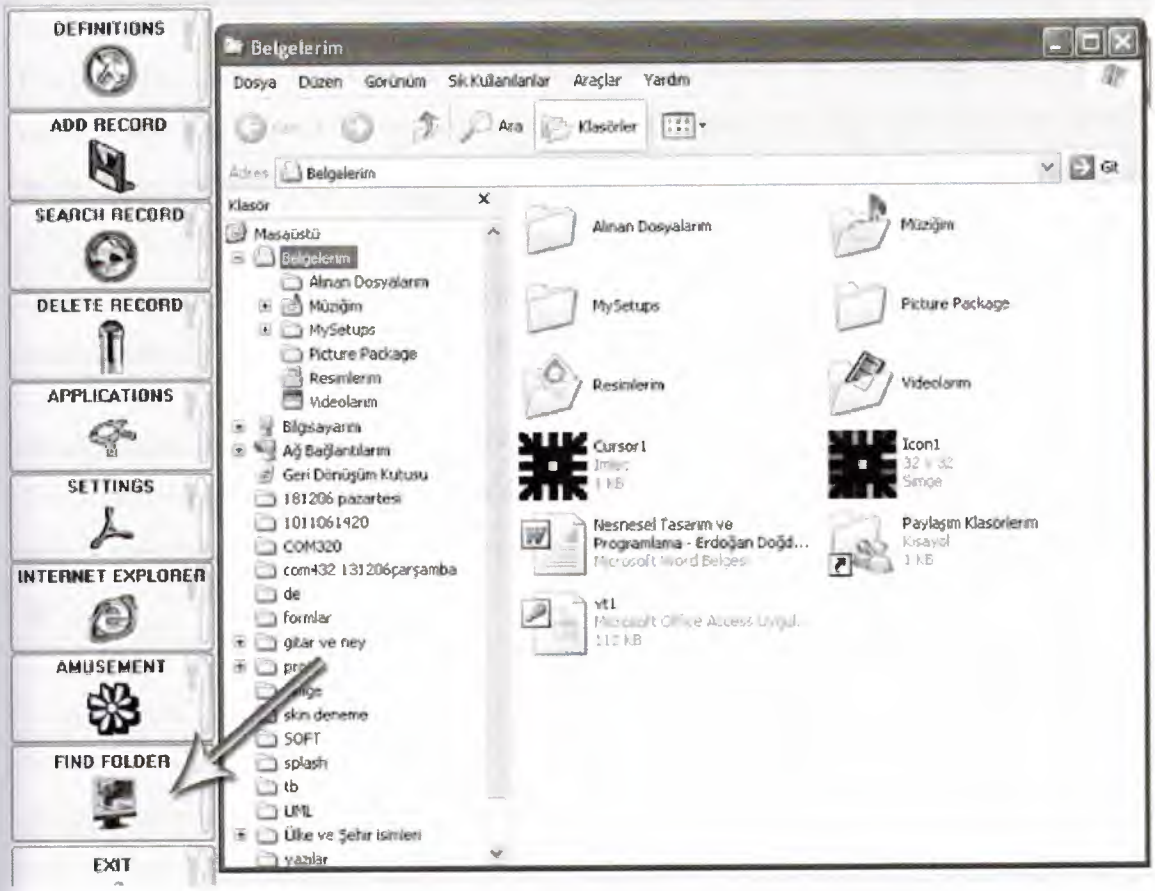
DEFINITIONS
ADD RECORD
SEARCH RECORD
DELETE RECORD
APPLICATIONS
SETTINGS
INTERNET EXPLORER
AMUSEMENT
FIND FOLDER
EXIT
Belgelerim
Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım
Ara
Klasörler
Adres
Belgelerim
Git
Klasör
Masaüstü
Belgelerim
Alınan Dosyalarım
Müziğim
MySetups
Picture Package
Resimlerim
Videolarım
Bilgisayarım
Ağ Bağlantılarım
Geri Dönüşüm Kutusu
181206 pazartesi
1011061420
COM320
com432 131206çarşamba
de
formlar
gitar ve ney
skin deneme
SOFT
splash
tb
UML
Ülke ve Şehir isimleri
yazılar
Alınan Dosyalarım
Müziğim
MySetups
Picture Package
Resimlerim
Videolarım
Cursor1
İmleç
1 KB
Icon1
32 x 32
Simge
Nesnesel Tasarım ve Programlama - Erdoğan Doğd...
Microsoft Word Belgesi
Paylaşım Klasörlerim
Kısayol
1 KB
vt1
Microsoft Office Access Uygul...
212 KB

Figure 4.24

CONCLUSION

MySQL and Delphi are powerful program.When I study with these two program,I get fun.Because these program are wonderful.Examination of the data for internal consistency and comparisons with externally available data indicates that the Delphi study appears reliable. However, the study was difficult to carry out owing to difficulties in obtaining answers from possible respondents. Thus, if a larger survey is to be undertaken to include all building components, it is recommended that committed respondents be obtained before devising the survey.

Veterinerian Application program for veterinerian and users act more facility.However Users adapt easly to the program and use it safetly.Nowadays in everywhere, in every job is combined with the computer.Well Veterinerian clinic will combine with this project.

APPENDIX

VETARINERIAN APPLICATION PROGRAM SOURCE CODE

FORM 1 CODES

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, Menus, ExtCtrls, WinSkinData, jpeg, StdCtrls,
  XPMan;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    StatusBar1: TStatusBar;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    Shape1: TShape;
    SkinData1: TSkinData;
    Label1: TLabel;
    Timer1: TTimer;
    Image1: TImage;
    XPManifest1: TXPManifest;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);

    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
```

```
  procedure SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton3MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton4MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton7MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton8MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton9MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton6MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton10MouseMove(Sender: TObject; Shift: TShiftState;
    X, Y: Integer);
  procedure SpeedButton6Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit9, Unit41;

{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
//if (form1.Label1.Top <> 600) then//and (form1.Label1.Top > 1) then
//form1.Label1.Top:=form1.Label1.Top-1;
//if form1.Label1.Top <> 1 then
//form1.Label1.Top:=form1.Label1.Top+1;
FORM1.StatusBar1.Panels[5].Text:=TIMETOSTR(TIME);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Label1.Caption:='COMSOFT and SCIENCES'+#13+'                GROUP';
FORM1.StatusBar1.Panels[1].Text:=DATETOSTR(DATE);
FORM1.StatusBar1.Panels[5].Text:=TIMETOSTR(TIME);
```

```
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM2.SHOW;
end;

procedure TForm1.SpeedButton10Click(Sender: TObject);
begin
form41.CLOSE;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
FORM6.CLOSE;
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
 FORM3.SHOW;
end;



procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
form2.CLOSE;
form3.CLOSE;
form4.CLOSE;
form5.CLOSE;
form6.CLOSE;
form41.close;

end;

procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
```

```
form4.show;
end;

procedure TForm1.SpeedButton8Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
FORM5.SHOW;
end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM7.CLOSE;
form6.show;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM7.SHOW;
end;

procedure TForm1.SpeedButton9Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
 if FileExists('C:\WINDOWS\explorer.exe') then
   winexec('C:\WINDOWS\explorer.exe',sw_shownormal);
end;

procedure TForm1.SpeedButton7Click(Sender: TObject);
```

```
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
  if FileExists('C:\Program Files\Internet Explorer\iexplore.exe') then
    winexec('C:\Program Files\Internet Explorer\iexplore.exe',sw_shownormal);
end;

procedure TForm1.SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
FORM1.SpeedButton1.Hint:='     THIS ACTS TO DEFINE NEW
KNOWLEDGE'+#13+
                  '(STAFF, VACCINE, DRUGS, OPERATIONS, USERS)';

end;

procedure TForm1.SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton2.Hint:='USES TO SAVE NEW RECORD'+#13+
                  '   (CUSTOMER, ANIMAL)';
end;

procedure TForm1.SpeedButton3MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton3.Hint:='USE TO FIND RECORD'+#13+
                  '   (ALL CRITERIA)';
end;

procedure TForm1.SpeedButton4MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton4.Hint:='ACTS TO DELETE RECORD'+#13+
                  '    (ALL CRITERIA)';
end;

procedure TForm1.SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton5.Hint:='     USES TO SAVE NEW APPLICATION'+#13+
                  '(VACCINATE, INNER PARASITE, OUTER PARASITE)'+#13+
                  '(MEDICINATE, APPLIED OPERATIONS, ILNESSES )';
end;
```

```
procedure TForm1.SpeedButton7MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton7.Hint:='USES TO OPEN THE INTERNET EXPLORER';
end;

procedure TForm1.SpeedButton8MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton8.Hint:='USE TO HAVE FUN';
end;

procedure TForm1.SpeedButton9MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton9.Hint:='USES TO SEE WINDOWS FILES OR FOLDERS';
end;

procedure TForm1.SpeedButton6MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
FORM1.SpeedButton6.Hint:='ACTS TO CHANGE THE PROGRAM SETTINGS';
end;

procedure TForm1.SpeedButton10MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
FORM1.SpeedButton10.Hint:='ACTS TO CLOSE THE PROGRAM';
end;

procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
FORM4.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM9.SHOW;
end;

end.
```

FORM 2 CODES

```
unit Unit2;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm2 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton6: TSpeedButton;
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

uses Unit10, Unit11, Unit12, Unit13, Unit14;

{$R *.dfm}

procedure TForm2.SpeedButton6Click(Sender: TObject);
begin
  form2.hide;
end;

procedure TForm2.SpeedButton1Click(Sender: TObject);
begin
  FORM10.SHOW;
  form2.Hide;
end;

procedure TForm2.SpeedButton2Click(Sender: TObject);
begin
  form11.show;
  form2.Hide;
```

```
end;

procedure TForm2.SpeedButton3Click(Sender: TObject);
begin
  FORM12.SHOW;
  FORM2.Hide;
end;

procedure TForm2.SpeedButton4Click(Sender: TObject);
begin
  FORM13.SHOW;
  FORM2.HIDE;
end;

procedure TForm2.SpeedButton5Click(Sender: TObject);
begin
  FORM14.SHOW;
  FORM2.Hide;
end;

end.
```

FORM 3 CODES

```
unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm3 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    SpeedButton11: TSpeedButton;
    SpeedButton12: TSpeedButton;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    SpeedButton15: TSpeedButton;
```

```
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton16: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton15Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton16Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);


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

var
  Form3: TForm3;

implementation

uses Unit1, Unit23, Unit24, Unit25, Unit28, Unit27, Unit29, Unit30, Unit31,
  Unit32, Unit26, Unit33, Unit34, Unit35, Unit36, Unit37;

{$R *.dfm}


procedure TForm3.SpeedButton1Click(Sender: TObject);
begin
  FORM23.SHOW;
  FORM3.Hide;
end;

procedure TForm3.SpeedButton7Click(Sender: TObject);
begin
  FORM24.SHOW;
  FORM3.Hide;
end;
```

```
procedure TForm3.SpeedButton14Click(Sender: TObject);
begin
  FORM25.SHOW;
  FORM3.Hide;
end;

procedure TForm3.SpeedButton15Click(Sender: TObject);
begin
FORM27.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton2Click(Sender: TObject);
begin
FORM28.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton5Click(Sender: TObject);
begin
form29.show;
form3.Hide;
end;

procedure TForm3.SpeedButton9Click(Sender: TObject);
begin
FORM30.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton10Click(Sender: TObject);
begin
FORM31.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton3Click(Sender: TObject);
begin
FORM32.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton4Click(Sender: TObject);
begin
FORM26.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton8Click(Sender: TObject);
begin
```

```
FORM33.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton11Click(Sender: TObject);
begin
FORM34.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton6Click(Sender: TObject);
begin
FORM35.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton12Click(Sender: TObject);
begin
FORM36.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton16Click(Sender: TObject);
begin
FORM3.Hide;
end;

procedure TForm3.SpeedButton13Click(Sender: TObject);
begin
FORM37.SHOW;
FORM3.Hide;
end;

end.
```

FORM 4 CODES

```
unit Unit4;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm4 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses Unit17, Unit18, Unit19, Unit20, Unit21, Unit22;

{$R *.dfm}

procedure TForm4.SpeedButton1Click(Sender: TObject);
begin
  FORM17.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton2Click(Sender: TObject);
begin
  FORM18.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton5Click(Sender: TObject);
begin
  FORM19.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton3Click(Sender: TObject);
begin
  FORM20.SHOW;
```

```
  FORM4.Hide;
end;

procedure TForm4.SpeedButton4Click(Sender: TObject);
begin
  FORM21.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton6Click(Sender: TObject);
begin
form22.show;
form4.Hide;
end;

procedure TForm4.SpeedButton7Click(Sender: TObject);
begin
FORM4.Hide;
end;

end.
```

FORM 5 CODES

```
unit Unit5;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm5 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
```

```
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form5: TForm5;

implementation

{$R *.dfm}

procedure TForm5.SpeedButton1Click(Sender: TObject);
begin
  if FileExists('C:\Program Files\Windows Media Player\wmplayer.exe') then
    winexec('C:\Program Files\Windows Media Player\wmplayer.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton2Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\sol.exe') then
    winexec('C:\WINDOWS\system32\sol.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton3Click(Sender: TObject);
begin
  if FileExists('C:\windows\system32\freecell.exe') then
    winexec('C:\windows\system32\freecell.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton4Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\winmine.exe') then
    winexec('C:\WINDOWS\system32\winmine.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton5Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\calc.exe') then
  winexec('C:\WINDOWS\system32\calc.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton8Click(Sender: TObject);
begin
```

```
  if FileExists('C:\WINDOWS\notepad.exe') then
    winexec('C:\WINDOWS\notepad.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton6Click(Sender: TObject);
begin
  if FileExists('C:\Program Files\MSN Messenger\msnmsgr.exe') then
    winexec('C:\Program Files\MSN Messenger\msnmsgr.exe',sw_shownormal)
  else if FileExists('C:\Program Files\Messenger\msmsgs.exe') then
    winexec('C:\Program Files\Messenger\msmsgs.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton7Click(Sender: TObject);
begin
form5.Hide;
end;

procedure TForm5.SpeedButton9Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\mshearts.exe') then
    winexec('C:\WINDOWS\system32\mshearts.exe', sw_shownormal);
end;

end.
```

FORM 6 CODES

```
unit Unit6;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm6 = class(TForm)
    SpeedButton1: TSpeedButton;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton11: TSpeedButton;
```

```
    SpeedButton12: TSpeedButton;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form6: TForm6;

implementation

uses Unit15, Unit11, Unit12, Unit13, Unit14, Unit16, Unit17, Unit18,
  Unit19, Unit21, Unit20, Unit22;

{$R *.dfm}

procedure TForm6.SpeedButton6Click(Sender: TObject);
begin
FORM6.CLOSE;
end;

procedure TForm6.SpeedButton1Click(Sender: TObject);
begin
  Form15.LbSpeedButton1.Enabled:=FALSE;
  Form15.LbSpeedButton2.Enabled:=FALSE;
  FORM15.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton5Click(Sender: TObject);
begin
  FORM11.SpeedButton2.Enabled:=FALSE;
  FORM11.SpeedButton3.Enabled:=FALSE;
  FORM11.SHOW;
```

```
  FORM6.Close;
end;

procedure TForm6.SpeedButton9Click(Sender: TObject);
begin
  FORM12.SpeedButton2.Enabled:=FALSE;
  FORM12.SpeedButton3.Enabled:=FALSE;
  FORM12.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton4Click(Sender: TObject);
begin
  Form13.LbSpeedButton1.Enabled:=FALSE;
  Form13.LbSpeedButton2.Enabled:=FALSE;
  FORM13.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton7Click(Sender: TObject);
begin
  Form14.LbSpeedButton1.Enabled:=FALSE;
  Form14.LbSpeedButton2.Enabled:=FALSE;
  FORM14.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton8Click(Sender: TObject);
begin
  FORM16.SpeedButton3.Enabled:=FALSE;
  FORM16.SpeedButton4.Enabled:=FALSE;
  FORM16.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton2Click(Sender: TObject);
begin
  Form17.LbSpeedButton1.Enabled:=FALSE;
  Form17.LbSpeedButton2.Enabled:=FALSE;
  FORM17.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton11Click(Sender: TObject);
begin
  Form18.SpeedButton3.Enabled:=FALSE;
  Form18.SpeedButton4.Enabled:=FALSE;
  FORM18.SHOW;
  FORM6.Close;
end;
```

```
procedure TForm6.SpeedButton14Click(Sender: TObject);
begin
  Form19.LbSpeedButton1.Enabled:=FALSE;
  Form19.LbSpeedButton2.Enabled:=FALSE;
  FORM19.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton12Click(Sender: TObject);
begin
  Form21.LbSpeedButton1.Enabled:=FALSE;
  Form21.LbSpeedButton2.Enabled:=FALSE;
  FORM21.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton3Click(Sender: TObject);
begin
  Form20.SpeedButton3.Enabled:=FALSE;
  Form20.SpeedButton4.Enabled:=FALSE;
  FORM20.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton13Click(Sender: TObject);
begin
  Form22.SpeedButton3.Enabled:=FALSE;
  Form22.SpeedButton4.Enabled:=FALSE;
  FORM22.SHOW;
  FORM6.Close;
end;

end.
```

FORM 7 CODES

```
unit Unit7;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm7 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
  SpeedButton1: TSpeedButton;
  SpeedButton2: TSpeedButton;
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form7: TForm7;

implementation

uses Unit15, Unit16;

{$R *.dfm}

procedure TForm7.SpeedButton1Click(Sender: TObject);
begin
 FORM15.SHOW;
 FORM7.HIDE;
end;

procedure TForm7.SpeedButton2Click(Sender: TObject);
begin
 FORM16.SHOW;
 FORM7.Hide;
end;

end.
```

FORM 8 CODES

```
unit Unit8;

interface

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

type
 TForm8 = class(TForm)
  Image1: TImage;
 private
  { Private declarations }
 public
```

```
    { Public declarations }
  end;

var
  Form8: TForm8;

implementation

{$R *.dfm}

end.
```

FORM 9 CODES

```
unit Unit9;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, Menus, StdCtrls, jpeg, ExtDlgs;

type
  TForm9 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    PageControl1: TPageControl;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    ColorDialog1: TColorDialog;
    FontDialog1: TFontDialog;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    SpeedButton4: TSpeedButton;
    CheckBox3: TCheckBox;
    CheckBox6: TCheckBox;
    SpeedButton5: TSpeedButton;
    OpenDialog1: TOpenDialog;
    OpenPictureDialog1: TOpenPictureDialog;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    procedure SpeedButton4Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
    procedure CheckBox6Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
  private
    { Private declarations }
```

```
 public
  { Public declarations }
 end;

var
 Form9: TForm9;

implementation

uses Unit1, Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit10, Unit11,
 Unit12, Unit13, Unit14, Unit15, Unit16, Unit17, Unit18, Unit19, Unit20,
 Unit21, Unit22, Unit23, Unit24, Unit25, Unit26, Unit27, Unit28, Unit29,
 Unit30, Unit31, Unit32, Unit33, Unit34, Unit35, Unit36, Unit37, Unit38,
 Unit39, Unit40, Unit41;

{$R *.dfm}

procedure TForm9.SpeedButton4Click(Sender: TObject);
begin
 if form9.CheckBox2.Checked <> true then
 begin
  form9.OpenDialog1.Filter:='Skin Files (skn)|*.skn';
  if form9.OpenDialog1.Execute then
  begin
   form1.SkinData1.LoadFromFile(form9.OpenDialog1.FileName);
   //form1.Label1.Caption:=form1.SkinData1.SkinFile;
  end;
 end
 else
 begin
  beep;
  showmessage('YOU HAVE CANCELED THE SKINS BEFORE');
 end;

end;

procedure TForm9.CheckBox2Click(Sender: TObject);
begin
 if form9.CheckBox2.Checked = true then
  form1.SkinData1.Active:=false;
 if form9.CheckBox2.Checked = false then
  form1.SkinData1.Active:=true;

end;

procedure TForm9.CheckBox6Click(Sender: TObject);
begin
 if form9.CheckBox6.Checked = true then
 begin
  form1.Image1.Visible:=true;
```

```
 end;

 if form9.CheckBox6.Checked = false then
 begin
  form1.Image1.Visible:=false;
 end;
end;

procedure TForm9.SpeedButton5Click(Sender: TObject);
begin
 if form9.CheckBox6.Checked = true then
 begin
  if form9.OpenPictureDialog1.Execute then
   form1.Image1.Picture.LoadFromFile(form9.OpenPictureDialog1.FileName);
 end
 else
 begin
  beep;
  showmessage('YOU HAVE CANCELED WALLPAPERS BEFORE');
 end;
end;

procedure TForm9.SpeedButton1Click(Sender: TObject);
begin
 if form9.ColorDialog1.Execute then
 begin
  form1.Color:=form9.ColorDialog1.Color;
  form2.Color:=form9.ColorDialog1.Color;
  form3.Color:=form9.ColorDialog1.Color;
  form4.Color:=form9.ColorDialog1.Color;
  form5.Color:=form9.ColorDialog1.Color;
  form6.Color:=form9.ColorDialog1.Color;
  form7.Color:=form9.ColorDialog1.Color;
  form9.Color:=form9.ColorDialog1.Color;
  form10.Color:=form9.ColorDialog1.Color;
  form11.Color:=form9.ColorDialog1.Color;
  form12.Color:=form9.ColorDialog1.Color;
  form13.Color:=form9.ColorDialog1.Color;
  form14.Color:=form9.ColorDialog1.Color;
  form15.Color:=form9.ColorDialog1.Color;
  form16.Color:=form9.ColorDialog1.Color;
  form17.Color:=form9.ColorDialog1.Color;
  form18.Color:=form9.ColorDialog1.Color;
  form19.Color:=form9.ColorDialog1.Color;
  form20.Color:=form9.ColorDialog1.Color;
  form21.Color:=form9.ColorDialog1.Color;
  form22.Color:=form9.ColorDialog1.Color;
  form23.Color:=form9.ColorDialog1.Color;
  form24.Color:=form9.ColorDialog1.Color;
  form25.Color:=form9.ColorDialog1.Color;
```

```
    form26.Color:=form9.ColorDialog1.Color;
    form27.Color:=form9.ColorDialog1.Color;
    form28.Color:=form9.ColorDialog1.Color;
    form29.Color:=form9.ColorDialog1.Color;
    form30.Color:=form9.ColorDialog1.Color;
    form31.Color:=form9.ColorDialog1.Color;
    form32.Color:=form9.ColorDialog1.Color;
    form33.Color:=form9.ColorDialog1.Color;
    form34.Color:=form9.ColorDialog1.Color;
    form35.Color:=form9.ColorDialog1.Color;
    form36.Color:=form9.ColorDialog1.Color;
    form37.Color:=form9.ColorDialog1.Color;
    form38.Color:=form9.ColorDialog1.Color;
    form39.Color:=form9.ColorDialog1.Color;
    form40.Color:=form9.ColorDialog1.Color;
    form41.Color:=form9.ColorDialog1.Color;
  end;

end;

procedure TForm9.SpeedButton2Click(Sender: TObject);
begin
form1.color:=clBlack;
form2.color:=clBtnFace;
form3.color:=clBtnFace;
form4.color:=clBtnFace;
form5.color:=clBtnFace;
form6.color:=clBtnFace;
form7.color:=clBtnFace;
form9.color:=clBtnFace;
form10.color:=$004080FF;
form11.color:=$00C08080;
form12.color:=$00400040;
form13.color:=clGray;
form14.color:=clSilver;
form15.color:=$00404080;
form16.color:=clBtnFace;
form17.color:=clMoneyGreen;
form18.color:=$00400000;
form19.color:=clBlack;
form20.color:=clBtnFace;
form21.color:=$00404080;
form22.color:=clInactiveCaptionText;
form23.color:=clBtnFace;
form24.color:=clBtnFace;
form25.color:=clBtnFace;
form26.color:=clBtnFace;
form27.color:=clBtnFace;
form28.color:=clBtnFace;
form29.color:=clBtnFace;
```

```
form30.color:=clBtnFace;
form31.color:=clBtnFace;
form32.color:=clBtnFace;
form33.color:=clBtnFace;
form34.color:=clBtnFace;
form35.color:=clBtnFace;
form36.color:=clBtnFace;
form37.color:=clBtnFace;
form38.color:=clBtnFace;
form39.color:=$00D0A5A4;
form40.color:=clBtnFace;
form41.color:=clBtnFace;
end;

end.
```

FORM 10 CODES

```
unit Unit10;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Mask, Menus, DB, ADODB, Buttons, Grids,
  DBGrids, LbSpeedButton, ExtCtrls;

type
  TForm10 = class(TForm)
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Label8: TLabel;
```

```
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Edit4: TEdit;
    MaskEdit1: TMaskEdit;
    Memo1: TMemo;
    Edit5: TEdit;
    ComboBox3: TComboBox;
    ComboBox4: TComboBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Edit6: TEdit;
    DateTimePicker3: TDateTimePicker;
    MaskEdit2: TMaskEdit;
    Edit7: TEdit;
    Label18: TLabel;
    Memo2: TMemo;
    StatusBar1: TStatusBar;
    Label19: TLabel;
    Edit8: TEdit;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    DBGrid1: TDBGrid;
    SpeedButton1: TSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    Panel1: TPanel;
    ADOQuery2: TADOQuery;
    DataSource2: TDataSource;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;
```

```
implementation

uses Unit38;

{$R *.dfm}

procedure TForm10.FormCreate(Sender: TObject);
begin
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm10.SpeedButton1Click(Sender: TObject);
begin
 FORM38.SHOW;
 TA:=10;
end;

procedure TForm10.FormShow(Sender: TObject);
begin
form10.DateTimePicker1.Date:=date;
form10.DateTimePicker2.Date:=date;
//form10.DateTimePicker3.Date:=date;
form10.ADOQuery2.Close;
form10.ADOQuery2.SQL.Text:='select * from staff';
form10.ADOQuery2.Open;
end;

procedure TForm10.LbSpeedButton1Click(Sender: TObject);
begin
 form10.ADOQuery1.Close;
 form10.ADOQuery1.SQL.Text:='select * from staff where
Staff_name='+#39+form10.Edit2.Text+#39+' and
Staff_surname='+#39+form10.Edit3.Text+#39+' and
S_birthdate='+#39+datetostr(form10.DateTimePicker2.date)+#39;
 form10.ADOQuery1.Open;

 if form10.ADOQuery1.RecordCount = 0 then
 begin
  if (form10.Edit2.Text <> '') or (form10.Edit3.Text <> '') then
  begin
   form10.ADOQuery1.Close;
   form10.ADOQuery1.SQL.Text:='insert into staff
(Staff_name,Staff_surname,Staff_task,University,Grade_state,S_workstartdate,S_birthd
ate,S_TCidno,S_homephone,S_mobilphone,S_address,S_town,S_city,S_country,S_ema
il,S_web,S_leavingdate,S_note) values
('+#39+Form10.Edit2.Text+#39+','+#39+form10.Edit3.Text+#39+','+#39+form10.Com
boBox1.Text+#39+','+#39+form10.Edit4.Text+#39+','+#39+form10.ComboBox2.Text+
#39+','+#39+datetostr(form10.DateTimePicker1.date)+#39+','+#39+datetostr(form10.D
```

```
ateTimePicker2.date)+#39+','+#39+form10.Edit8.Text+#39+','+#39+form10.MaskEdit1
.Text+#39+','+#39+form10.MaskEdit2.Text+#39+','+#39+form10.Memo1.Text+#39+','
+#39+form10.Edit5.Text+#39+','+#39+form10.ComboBox3.Text+#39+','+#39+form10.
ComboBox4.Text+#39+','+#39+form10.Edit6.Text+#39+','+#39+form10.Edit7.Text+#3
9+','+#39+datetostr(form10.DateTimePicker3.date)+#39+','+#39+form10.Memo2.Text+
#39+')';
     form10.ADOQuery1.ExecSQL;
     form10.ADOQuery1.Close;
     form10.ADOQuery1.SQL.Text:='select * from staff';
     form10.ADOQuery1.Open;
     showmessage('RECORD SAVED');
     FORM10.LbSpeedButton4.Click;
   end
   else
    showmessage('CHECK THE FORM FOR EMPTY PLACE');
 end
 else
  showmessage('RECORD HAS RECORDED BEFORE');
end;

procedure TForm10.LbSpeedButton4Click(Sender: TObject);
begin
FORM10.Edit1.Clear;
FORM10.Edit2.Clear;
FORM10.Edit3.Clear;
FORM10.ComboBox1.Text:='Select One';
FORM10.Edit4.Clear;
FORM10.ComboBox2.Text:='Select One';
FORM10.DateTimePicker1.Date:=DATE;

dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'd/m/yyyy';

FORM10.DateTimePicker2.Date:=STRTODATE('09.09.9999');
FORM10.DateTimePicker3.Date:=STRTODATE('09.09.9999');
form10.Edit8.Clear;
form10.MaskEdit1.Clear;
form10.MaskEdit2.Clear;
form10.Memo1.Clear;
form10.Edit5.Clear;
FORM10.ComboBox3.Text:='Select One';
FORM10.ComboBox4.Text:='Select One';
form10.Edit6.Clear;
form10.Edit7.Clear;
form10.Memo2.clear;
form10.Edit2.SetFocus;

dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
```

```
shortdateformat := 'yyyy/m/d';

form10.ADOQuery2.Close;
form10.ADOQuery2.SQL.Text:='select * from staff';
form10.ADOQuery2.Open;


end;

procedure TForm10.Edit1Change(Sender: TObject);
begin
  form38.Hide;
  form10.ADOQuery1.Close;
  form10.ADOQuery1.SQL.Text:='select * from staff where
staff_id='+#39+form10.Edit1.Text+#39;
  form10.ADOQuery1.Open;
  if form10.ADOQuery1.RecordCount <> 0 then
  begin
    form10.Edit1.Text:=FORM10.ADOQuery1.Fields[0].Text;
    form10.Edit2.Text:=FORM10.ADOQuery1.Fields[1].Text;
    form10.Edit3.Text:=FORM10.ADOQuery1.Fields[2].Text;
    form10.ComboBox1.Text:=FORM10.ADOQuery1.Fields[3].Text;
    form10.Edit4.Text:=FORM10.ADOQuery1.Fields[4].Text;
    form10.ComboBox2.Text:=FORM10.ADOQuery1.Fields[5].Text;

    dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
    shortdateformat := 'd/m/yyyy';

    form10.DateTimePicker1.Date:=strtodate(FORM10.ADOQuery1.Fields[6].Text);
    form10.DateTimePicker2.Date:=strtodate(FORM10.ADOQuery1.Fields[7].Text);
    form10.Edit8.Text:=FORM10.ADOQuery1.Fields[8].Text;
    form10.MaskEdit1.Text:=FORM10.ADOQuery1.Fields[9].Text;
    form10.MaskEdit2.Text:=FORM10.ADOQuery1.Fields[10].Text;
    form10.Memo1.Text:=FORM10.ADOQuery1.Fields[11].Text;
    form10.Edit5.Text:=FORM10.ADOQuery1.Fields[12].Text;
    form10.ComboBox3.Text:=FORM10.ADOQuery1.Fields[13].Text;
    form10.ComboBox4.Text:=FORM10.ADOQuery1.Fields[14].Text;
    form10.Edit6.Text:=FORM10.ADOQuery1.Fields[15].Text;
    form10.Edit7.Text:=FORM10.ADOQuery1.Fields[16].Text;
    form10.DateTimePicker3.Date:=strtodate(FORM10.ADOQuery1.Fields[17].Text);
    form10.Memo2.Text:=FORM10.ADOQuery1.Fields[18].Text;

    dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
    shortdateformat := 'yyyy/m/d';
  end;

end;
```

```
procedure TForm10.LbSpeedButton2Click(Sender: TObject);
begin
  IF (FORM10.Edit1.Text <> '') AND (FORM10.Edit2.Text <> '') AND
(FORM10.Edit3.Text <> '') THEN
  BEGIN
    FORM10.ADOQuery2.Close;
    FORM10.ADOQuery2.SQL.Text:='UPDATE staff set Staff_name=
'+#39+form10.Edit2.Text+#39+', Staff_surname= '+#39+form10.Edit3.Text+#39+',
Staff_task='+#39+form10.ComboBox1.Text+#39+',
University='+#39+form10.Edit4.Text+#39+',
Grade_state='+#39+form10.ComboBox2.Text+#39+',
S_workstartdate='+#39+datetostr(form10.DateTimePicker1.date)+#39+',
S_birthdate='+#39+datetostr(form10.DateTimePicker2.date)+#39+',
S_TCidno='+#39+form10.Edit8.Text+#39+',
S_homephone='+#39+form10.MaskEdit1.Text+#39+',
S_mobilphone='+#39+form10.MaskEdit2.Text+#39+',
S_address='+#39+form10.Memo1.Text+#39+',
S_town='+#39+form10.Edit5.Text+#39+',
S_city='+#39+form10.ComboBox3.Text+#39+',
S_country='+#39+form10.ComboBox4.Text+#39+',
S_email='+#39+form10.Edit6.Text+#39+', S_web='+#39+form10.Edit7.Text+#39+',
S_leavingdate='+#39+datetostr(form10.DateTimePicker3.date)+#39+',
S_note='+#39+form10.Memo2.Text+#39+' WHERE
Staff_id='+#39+form10.Edit1.Text+#39;
    form10.ADOQuery2.ExecSQL;
    showmessage('RECORD UPDATED');
    FORM10.LbSpeedButton4.Click;
  END
  ELSE
    SHOWMESSAGE('PLEASE CHOOSE STAFF ID AND BE SURE'+#13+'TO
COMPLETE THE EMPTY PLACE');

end;

procedure TForm10.FormHide(Sender: TObject);
begin
FORM10.LbSpeedButton4.Click;
FORM10.ADOQuery1.Close;
FORM10.ADOQuery2.Close;
end;

procedure TForm10.FormClose(Sender: TObject; var Action: TCloseAction);
begin
FORM10.LbSpeedButton4.Click;
FORM10.ADOQuery1.Close;
FORM10.ADOQuery2.Close;
end;

end.
```

FORM 11 CODES

```
unit Unit11;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, Grids, DBGrids, DB, ADODB, ComCtrls, Buttons,
  StdCtrls, Menus;

type
  TForm11 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    ComboBox1: TComboBox;
    Label4: TLabel;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    StatusBar1: TStatusBar;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure DBGrid1KeyUp(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
```

```
  { Public declarations }
 end;

var
 Form11: TForm11;
 SS:WORD;
implementation

uses Unit12;

{$R *.dfm}

procedure TForm11.FormShow(Sender: TObject);
begin
form11.ADOQuery1.SQL.Text:='select * from vaccines';
form11.ADOQuery1.Open;
end;

procedure TForm11.SpeedButton2Click(Sender: TObject);
begin
 if (form11.Edit2.Text <> '') then
 begin
   form11.ADOQuery2.Close;
   form11.ADOQuery2.SQL.Text:='select * from vaccines where
vaccine_name='+#39+form11.Edit2.Text+#39;
   form11.ADOQuery2.Open;
   if form11.ADOQuery2.RecordCount = 0 then
   begin
     form11.ADOQuery2.Close;
     form11.ADOQuery2.SQL.Text:='insert into vaccines
(vaccine_name,vaccine_duration) values
('+#39+form11.Edit2.Text+#39+','+#39+form11.ComboBox1.Text+#39+')';
     form11.ADOQuery2.ExecSQL;
     showmessage('RECORD SAVED');
     FORM11.SpeedButton5.Click;
   END
   else
     showmessage('RECORD HAS SAVED BEFORE');
 END
 ELSE
   SHOWMESSAGE('BE SURE TO COMPLETE THE EMPTY PLACE');


end;

procedure TForm11.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM11.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
   FORM11.Edit1.Text:=FORM11.DBGrid1.Fields[0].Text;
```

```
  FORM11.Edit2.Text:=FORM11.DBGrid1.Fields[1].Text;
  FORM11.ComboBox1.Text:=FORM11.DBGrid1.Fields[2].Text;
 END
end;

procedure TForm11.DBGrid1KeyUp(Sender: TObject; var Key: Word;
 Shift: TShiftState);
begin
 IF  FORM11.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
  FORM11.Edit1.Text:=FORM11.DBGrid1.Fields[0].Text;
  FORM11.Edit2.Text:=FORM11.DBGrid1.Fields[1].Text;
  FORM11.ComboBox1.Text:=FORM11.DBGrid1.Fields[2].Text;
 END
end;

procedure TForm11.SpeedButton3Click(Sender: TObject);
begin
 IF  (FORM11.Edit2.Text <> '') OR ((FORM11.Edit1.Text <> '') AND
(FORM11.Edit2.Text <> ''))  THEN
 BEGIN
  FORM11.ADOQuery3.Close;
  FORM11.ADOQuery3.SQL.Text:='UPDATE vaccines set
vaccine_name='+#39+form11.Edit2.Text+#39+',
vaccine_duration='+#39+form11.ComboBox1.Text+#39+' where
vaccine_id='+#39+form11.Edit1.Text+#39;
  form11.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  FORM11.SpeedButton5.Click;
 end
 else
  showmessage('PLEASE BE SURE TO COMPLETE EMPTY PLACE');
end;

procedure TForm11.SpeedButton4Click(Sender: TObject);
begin
 IF  (FORM11.Edit1.Text <> '') THEN
 BEGIN
  SS:=MESSAGEDLG('ARE YOU SURE TO DELETE  " '+FORM11.Edit2.Text+' "
?',MTWARNING,[MBYES ,MBNO],0);
  IF SS = MRYES then
  begin
   FORM11.ADOQuery3.Close;
   FORM11.ADOQuery3.SQL.Text:='DELETE FROM vaccines where
vaccine_id='+#39+form11.Edit1.Text+#39;
   form11.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   FORM11.SpeedButton5.Click;
  end;
 end
```

```
 else
   showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm11.SpeedButton5Click(Sender: TObject);
begin
  FORM11.Edit1.Clear;
  FORM11.Edit2.Clear;
  FORM11.ComboBox1.Text:='Select One';
  form11.Edit2.SetFocus;
  FORM11.SpeedButton2.Enabled:=TRUE;
  FORM11.SpeedButton3.Enabled:=TRUE;
  form11.ADOQuery1.Close;
  form11.ADOQuery1.SQL.Text:='select * from vaccines';
  form11.ADOQuery1.Open;
end;

procedure TForm11.FormClose(Sender: TObject; var Action: TCloseAction);
begin
FORM11.SpeedButton5.Click;
form11.ADOQuery1.Close;
form11.ADOQuery2.Close;
form11.ADOQuery3.Close;
end;

procedure TForm11.FormHide(Sender: TObject);
begin
FORM11.SpeedButton5.Click;
form11.ADOQuery1.Close;
form11.ADOQuery2.Close;
form11.ADOQuery3.Close;
end;

end.
```

FORM 12 CODES

```
unit Unit12;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, Grids, DBGrids, ExtCtrls, Buttons, StdCtrls, Menus,
  DB, ADODB;

type
  TForm12 = class(TForm)
```

```
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Edit1: TEdit;
    Edit2: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    Label5: TLabel;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    StatusBar1: TStatusBar;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure SpeedButton3Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form12: TForm12;
  SS1:WORD;
implementation

{$R *.dfm}

procedure TForm12.FormShow(Sender: TObject);
begin
  form12.ADOQuery1.Close;
```

```
 form12.ADOQuery1.SQL.Text:='select * from drugs';
 form12.ADOQuery1.Open;
end;

procedure TForm12.SpeedButton2Click(Sender: TObject);
begin
 if (form12.Edit2.Text <> '') and (form12.ComboBox1.Text <> 'Select One') then
 begin
  form12.ADOQuery2.Close;
  form12.ADOQuery2.SQL.Text:='select * from drugs where
drug_name='+#39+form12.Edit2.Text+#39;
  form12.ADOQuery2.Open;
  if form12.ADOQuery2.RecordCount = 0 then
  begin
   form12.ADOQuery2.Close;
   form12.ADOQuery2.SQL.Text:='insert into drugs
(drug_name,drug_duration,drug_kind) values
('+#39+form12.Edit2.Text+#39+','+#39+form12.ComboBox1.Text+#39+','+#39+form1
2.ComboBox2.Text+#39+')';
   form12.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   Form12.SpeedButton5.Click;
  END
  ELSE
   SHOWMESSAGE('RECORD HAS SAVED BEFORE');
 END
 ELSE
  SHOWMESSAGE('BE SURE TO COMPLETE THE EMPTY PLACE');

end;

procedure TForm12.FormKeyPress(Sender: TObject; var Key: Char);
begin
 {IF KEY=VK_F2 THEN
 BEGIN
  if (form12.Edit2.Text <> '') and (form12.ComboBox1.Text <> 'Select One') then
  begin
   form12.ADOQuery2.Close;
   form12.ADOQuery2.SQL.Text:='select * from drugs where
drug_name='+#39+form12.Edit2.Text+#39;
   form12.ADOQuery2.Open;
   if form12.ADOQuery2.RecordCount = 0 then
    begin
     form12.ADOQuery2.Close;
     form12.ADOQuery2.SQL.Text:='insert into drugs
(drug_name,drug_duration,drug_kind) values
('+#39+form12.Edit2.Text+#39+','+#39+form12.ComboBox1.Text+#39+','+#39+form1
2.ComboBox2.Text+#39+')';
     form12.ADOQuery2.ExecSQL;
     showmessage('RECORD SAVED');
```

```
      FORM12.ADOQuery1.Close;
      FORM12.ADOQuery1.SQL.Text:='SELECT * FROM drugs';
      FORM12.ADOQuery1.Open;
     END
     ELSE
      SHOWMESSAGE('RECORD HAS SAVED BEFORE');
   END
   ELSE
    SHOWMESSAGE('BE SURE TO COMPLETE THE EMPTY PLACE');
 END;}
end;

procedure TForm12.SpeedButton3Click(Sender: TObject);
begin
 IF (FORM12.Edit1.Text <> '') AND (FORM12.Edit2.Text <> '')   THEN
 BEGIN
   FORM12.ADOQuery3.Close;
   FORM12.ADOQuery3.SQL.Text:='UPDATE drugs set
drug_name='+#39+form12.Edit2.Text+#39+',
drug_duration='+#39+form12.ComboBox1.Text+#39+',
drug_kind='+#39+form12.ComboBox2.Text+#39+'  where
drug_id='+#39+form12.Edit1.Text+#39;
   form12.ADOQuery3.ExecSQL;
   showmessage('RECORD UPDATED');
   Form12.SpeedButton5.Click;
 end
 else
   showmessage('PLEASE BE SURE TO COMPLETE EMPTY PLACE');
end;


procedure TForm12.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM12.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
   FORM12.Edit1.Text:=FORM12.DBGrid1.Fields[0].Text;
   FORM12.Edit2.Text:=FORM12.DBGrid1.Fields[1].Text;
   FORM12.ComboBox1.Text:=FORM12.DBGrid1.Fields[2].Text;
   FORM12.ComboBox2.Text:=FORM12.DBGrid1.Fields[3].Text;
 END
end;

procedure TForm12.SpeedButton4Click(Sender: TObject);
begin
 IF  (FORM12.Edit1.Text <> '') THEN
 BEGIN
   SS1:=MESSAGEDLG('ARE YOU SURE TO DELETE  " '+FORM12.Edit2.Text+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS1 = MRYES then
   BEGIN
```

```
    FORM12.ADOQuery3.Close;
    FORM12.ADOQuery3.SQL.Text:='DELETE FROM drugs where
drug_id='+#39+form12.Edit1.Text+#39;
    form12.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form12.SpeedButton5.Click;
   END;
  end
  else
   showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm12.SpeedButton5Click(Sender: TObject);
begin
  FORM12.Edit1.Clear;
  FORM12.Edit2.Clear;
  FORM12.ComboBox1.Text:='Select One';
  FORM12.ComboBox2.Text:='Select One';
  form12.Edit2.SetFocus;
  FORM12.SpeedButton2.Enabled:=TRUE;
  FORM12.SpeedButton3.Enabled:=TRUE;

  form12.ADOQuery1.Close;
  form12.ADOQuery1.SQL.Text:='select * from drugs';
  form12.ADOQuery1.Open;
end;

procedure TForm12.FormHide(Sender: TObject);
begin
  Form12.SpeedButton5.Click;
  form12.ADOQuery1.Close;
  form12.ADOQuery2.Close;
  form12.ADOQuery3.Close;
end;

procedure TForm12.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Form12.SpeedButton5.Click;
  form12.ADOQuery1.Close;
  form12.ADOQuery2.Close;
  form12.ADOQuery3.Close;
end;

end.
```

FORM 13 CODES

```
unit Unit13;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, Grids, DBGrids, ComCtrls, LbSpeedButton, Buttons,
  StdCtrls, Menus, DB, ADODB;

type
  TForm13 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    StatusBar1: TStatusBar;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form13: TForm13;
  SS13:WORD;
```

```
implementation

{$R *.dfm}

procedure TForm13.FormShow(Sender: TObject);
begin
  form13.ADOQuery1.Close;
  form13.ADOQuery1.SQL.Text:='select * from operations';
  form13.ADOQuery1.Open;
end;

procedure TForm13.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form13.LbSpeedButton4.Click;
  form13.ADOQuery1.Close;
  form13.ADOQuery2.Close;
  form13.ADOQuery3.Close;
end;

procedure TForm13.FormHide(Sender: TObject);
begin
  form13.LbSpeedButton4.Click;
  form13.ADOQuery1.Close;
  form13.ADOQuery2.Close;
  form13.ADOQuery3.Close;
end;

procedure TForm13.LbSpeedButton1Click(Sender: TObject);
begin
  if (form13.Edit2.Text <> '') then
  begin
    form13.ADOQuery2.Close;
    form13.ADOQuery2.SQL.Text:='select * from operations where
operation_name='+#39+form13.Edit2.Text+#39;
    form13.ADOQuery2.Open;
    if form13.ADOQuery2.RecordCount = 0 then
    begin
      form13.ADOQuery2.Close;
      form13.ADOQuery2.SQL.Text:='insert into operations (operation_name) values
('+#39+form13.Edit2.Text+#39+')';
      form13.ADOQuery2.ExecSQL;
      showmessage('RECORD SAVED');
      form13.LbSpeedButton4.Click;
    END
    ELSE
      SHOWMESSAGE('RECORD HAS SAVED BEFORE');
  END
  ELSE
    SHOWMESSAGE('BE SURE TO FILL THE OPERATION NAME');
end;
```

```
procedure TForm13.LbSpeedButton2Click(Sender: TObject);
begin
  IF (FORM13.Edit1.Text <> '') AND (FORM13.Edit2.Text <> '')  THEN
 BEGIN
  FORM13.ADOQuery3.Close;
  FORM13.ADOQuery3.SQL.Text:='UPDATE operations set
operation_name='+#39+form13.Edit2.Text+#39+' where
operation_id='+#39+form13.Edit1.Text+#39;
  form13.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  form13.LbSpeedButton4.Click;
 end
 else
  showmessage('PLEASE BE SURE TO COMPLETE EMPTY PLACE');
end;

procedure TForm13.LbSpeedButton3Click(Sender: TObject);
begin
 IF  (FORM13.Edit1.Text <> '') THEN
 BEGIN
  SS13:=MESSAGEDLG('ARE YOU SURE TO DELETE  ' '+FORM13.Edit2.Text+'
' ?',MTWARNING,[MBYES ,MBNO],0);
  IF SS13 = MRYES then
  BEGIN
   FORM13.ADOQuery3.Close;
   FORM13.ADOQuery3.SQL.Text:='DELETE FROM operations where
operation_id='+#39+form13.Edit1.Text+#39;
   form13.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   form13.LbSpeedButton4.Click;
  END;
 end
 else
  showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm13.LbSpeedButton4Click(Sender: TObject);
begin
 FORM13.Edit1.Clear;
 FORM13.Edit2.Clear;
 form13.Edit2.SetFocus;
 Form13.LbSpeedButton1.Enabled:=TRUE;
 Form13.LbSpeedButton2.Enabled:=TRUE;

 form13.ADOQuery1.Close;
 form13.ADOQuery1.SQL.Text:='select * from operations';
 form13.ADOQuery1.Open;
end;
```

```
procedure TForm13.DBGrid1CellClick(Column: TColumn);
begin
IF  FORM13.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
   FORM13.Edit1.Text:=FORM13.DBGrid1.Fields[0].Text;
   FORM13.Edit2.Text:=FORM13.DBGrid1.Fields[1].Text;
 END
end;

end.
```

FORM 14 CODES

```
unit Unit14;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, ExtCtrls, Grids, DBGrids, LbSpeedButton, Buttons,
  StdCtrls, Menus, DB, ADODB;

type
  TForm14 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    SpeedButton1: TSpeedButton;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    StatusBar1: TStatusBar;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
```

```
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form14: TForm14;
  SS14:WORD;
implementation

uses Unit38, Unit10, Unit12;

{$R *.dfm}

procedure TForm14.SpeedButton1Click(Sender: TObject);
begin
  form38.show;
  TA:=14;
end;

procedure TForm14.FormShow(Sender: TObject);
begin
  form14.ADOQuery1.Close;
  form14.ADOQuery1.SQL.Text:='select * from users';
  form14.ADOQuery1.Open;
end;

procedure TForm14.LbSpeedButton1Click(Sender: TObject);
begin
  if (form14.Edit1.Text <> '') and (form14.Edit2.Text <> '') and (form14.Edit3.Text <>
'') then
  begin
    form14.ADOQuery2.Close;
    form14.ADOQuery2.SQL.Text:='select * from users where
user_name='+#39+form14.Edit2.Text+#39;
    form14.ADOQuery2.Open;
    if form14.ADOQuery2.RecordCount = 0 then
    begin
```

```
    form14.ADOQuery2.Close;
    form14.ADOQuery2.SQL.Text:='insert into users
(user_name,password,staff_id,staff_state,staff_pozition) values
('+#39+form14.Edit2.Text+#39+','+#39+form14.Edit3.Text+#39+','+#39+form14.Edit1.
Text+#39+','+#39+form14.ComboBox1.Text+#39+','+#39+form14.ComboBox2.Text+#
39+')';
    form14.ADOQuery2.ExecSQL;
    showmessage('RECORD SAVED');
    Form14.LbSpeedButton4.Click;
   END
   ELSE
    SHOWMESSAGE('USER NAME IS USED');
 END
 ELSE if form14.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm14.LbSpeedButton2Click(Sender: TObject);
begin
 IF (FORM14.Edit1.Text <> '') AND (FORM14.Edit2.Text <> '') AND
(FORM14.Edit3.Text <> '')  THEN
 BEGIN
  form14.ADOQuery3.Close;
  form14.ADOQuery3.SQL.Text:='select * from users where
user_name='+#39+form14.Edit2.Text+#39;
  form14.ADOQuery3.Open;
  if form14.ADOQuery3.RecordCount = 0 then
  begin
   FORM14.ADOQuery3.Close;
   FORM14.ADOQuery3.SQL.Text:='UPDATE users set
user_name='+#39+form14.Edit2.Text+#39+',
password='+#39+form14.Edit3.Text+#39+', staff_id='+#39+form14.Edit1.Text+#39+',
staff_state='+#39+form14.ComboBox1.Text+#39+',
staff_pozition='+#39+form14.ComboBox2.Text+#39+'  where
staff_id='+#39+form14.Edit1.Text+#39;
    form14.ADOQuery3.ExecSQL;
    showmessage('RECORD UPDATED');
    Form14.LbSpeedButton4.Click;
   end
   else
    SHOWMESSAGE('USER NAME IS USED');
 end
 else
  showmessage('PLEASE BE SURE TO FILL EMPTY PLACE');
end;

procedure TForm14.LbSpeedButton3Click(Sender: TObject);
begin
```

```
 IF  (FORM14.Edit1.Text <> '') THEN
 BEGIN
  SS14:=MESSAGEDLG('ARE YOU SURE TO DELETE  '+FORM14.Edit2.Text+'
" ?',MTWARNING,[MBYES ,MBNO],0);
  IF SS14 = MRYES then
  BEGIN
   FORM14.ADOQuery3.Close;
   FORM14.ADOQuery3.SQL.Text:='DELETE FROM users where
user_name='+#39+form14.Edit2.Text+#39;
   form14.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   Form14.LbSpeedButton4.Click;
  END;
 end
 else
  showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm14.LbSpeedButton4Click(Sender: TObject);
begin
 FORM14.Edit1.Clear;
 FORM14.Edit2.Clear;
 FORM14.Edit3.Clear;
 FORM14.ComboBox1.Text:='Select One';
 FORM14.ComboBox2.Text:='Select One';
 form14.Edit2.SetFocus;
 Form14.LbSpeedButton1.Enabled:=TRUE;
 Form14.LbSpeedButton2.Enabled:=TRUE;

 form14.ADOQuery1.Close;
 form14.ADOQuery1.SQL.Text:='select * from users';
 form14.ADOQuery1.Open;
end;

procedure TForm14.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM14.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
  FORM14.Edit1.Text:=FORM14.DBGrid1.Fields[2].Text;
  FORM14.Edit2.Text:=FORM14.DBGrid1.Fields[0].Text;
  FORM14.Edit3.Text:=FORM14.DBGrid1.Fields[1].Text;
  FORM14.ComboBox1.Text:=FORM14.DBGrid1.Fields[3].Text;
  FORM14.ComboBox2.Text:=FORM14.DBGrid1.Fields[4].Text;
 END
end;

procedure TForm14.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 Form14.LbSpeedButton4.Click;
```

```
  form14.ADOQuery1.Close;
  form14.ADOQuery2.Close;
  form14.ADOQuery3.Close;
end;

procedure TForm14.FormHide(Sender: TObject);
begin
  Form14.LbSpeedButton4.Click;
  form14.ADOQuery1.Close;
  form14.ADOQuery2.Close;
  form14.ADOQuery3.Close;
end;

end.
```

FORM 15 CODES

```
unit Unit15;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, LbSpeedButton, StdCtrls, Buttons, Mask, Menus, ExtCtrls,
  ComCtrls, Grids, DBGrids, DB, ADODB;

type
  TForm15 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    MaskEdit1: TMaskEdit;
    MaskEdit2: TMaskEdit;
```

```
    MaskEdit3: TMaskEdit;
    MaskEdit4: TMaskEdit;
    Memo1: TMemo;
    ComboBox1: TComboBox;
    Edit4: TEdit;
    ComboBox2: TComboBox;
    Edit5: TEdit;
    Edit6: TEdit;
    Memo2: TMemo;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    ADOConnection1: TADOConnection;
    DBGrid1: TDBGrid;
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form15: TForm15;
  SS15:WORD;
implementation

uses Unit10;

{$R *.dfm}

procedure TForm15.FormShow(Sender: TObject);
begin
  FORM15.ADOQuery1.Close;
```

```
 FORM15.ADOQuery1.SQL.Text:='select
customer_id,Cname,Csurname,Mobilphone,homephone,workphone,fax,address,city,tow
n,country,email,web,c_note from customer';
 form15.ADOQuery1.Open;


end;

procedure TForm15.LbSpeedButton1Click(Sender: TObject);
begin
 if (form15.Edit2.Text <> '') or (form15.Edit3.Text <> '') then
 begin
   form15.ADOQuery2.Close;
   form15.ADOQuery2.SQL.Text:='select * from customer where
Cname='+#39+form15.Edit2.Text+#39+' and
Csurname='+#39+form15.Edit3.Text+#39+' and
mobilphone='+#39+form15.MaskEdit2.Text+#39;
   form15.ADOQuery2.Open;
   if form15.ADOQuery2.RecordCount = 0 then
   begin
     form15.ADOQuery2.Close;
     form15.ADOQuery2.SQL.Text:='insert into customer
(Cname,Csurname,homephone,mobilphone,workphone,fax,address,town,city,country,e
mail,web,C_note,recorddate,recordtime) values
('+#39+Form15.Edit2.Text+#39+','+#39+form15.Edit3.Text+#39+','+#39+form15.Mask
Edit1.Text+#39+','+#39+form15.MaskEdit2.Text+#39+','+#39+form15.MaskEdit3.Text
+#39+','+#39+form15.MaskEdit4.Text+#39+','+#39+form15.Memo1.Text+#39+','+#39
+form15.Edit4.Text+#39+','+#39+form15.ComboBox1.Text+#39+','+#39+form15.Com
boBox2.Text+#39+','+#39+form15.Edit5.Text+#39+','+#39+form15.Edit6.Text+#39+','
+#39+form15.Memo2.Text+#39+','+#39+datetostr(date)+#39+','+#39+timetostr(time)+
#39+')';
     form15.ADOQuery2.ExecSQL;
     form15.ADOQuery2.Close;
     form15.ADOQuery2.SQL.Text:='select * from customer';
     form15.ADOQuery2.Open;
     showmessage('RECORD SAVED');
     Form15.LbSpeedButton4.Click;
    END
    else
     SHOWMESSAGE('THE CUSTOMER SAVED BEFORE');
  END
  ELSE if form15.Edit2.Text = '' then
   showmessage('PLEASE FILL THE NAME')
  ELSE if form15.Edit3.Text = '' then
   showmessage('PLEASE FILL THE SURNAME')
  else
   SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');

end;
```

```
procedure TForm15.LbSpeedButton2Click(Sender: TObject);
begin
  IF (FORM15.Edit1.Text <> '') AND (FORM15.Edit2.Text <> '') AND
(FORM15.Edit3.Text <> '') THEN
 BEGIN
   FORM15.ADOQuery3.Close;
   FORM15.ADOQuery3.SQL.Text:='UPDATE customer set Cname=
'+#39+form15.Edit2.Text+#39+', Csurname= '+#39+form15.Edit3.Text+#39+',
homephone='+#39+form15.MaskEdit1.Text+#39+',
mobilphone='+#39+form15.MaskEdit2.Text+#39+ ',
workphone='+#39+form15.MaskEdit3.Text+#39+',
fax='+#39+form15.MaskEdit4.Text+#39+',
address='+#39+form15.Memo1.Text+#39+', town='+#39+form15.Edit4.Text+#39+',
city='+#39+form15.ComboBox1.Text+#39+',
country='+#39+form15.ComboBox2.Text+#39+',
email='+#39+form15.Edit5.Text+#39+', web='+#39+form15.Edit6.Text+#39+',
C_note='+#39+form15.Memo2.Text+#39+' WHERE
customer_id='+#39+form15.Edit1.Text+#39;
   form15.ADOQuery3.ExecSQL;
   showmessage('RECORD UPDATED');
   Form15.LbSpeedButton4.Click;
 END
 ELSE
   SHOWMESSAGE('PLEASE SELECT CUSTOMER AND BE SURE'+#13+'TO
FILL THE EMPTY PLACE');
end;

procedure TForm15.LbSpeedButton3Click(Sender: TObject);
begin
 IF (FORM15.Edit1.Text <> '') AND (FORM15.Edit2.Text <> '') AND
(FORM15.Edit3.Text <> '') then
 BEGIN
   SS15:=MESSAGEDLG('ARE YOU SURE TO DELETE  '' ID:
'+FORM15.Edit1.Text+' ; CUSTOMER: ' +FORM15.Edit2.Text+'
'+FORM15.Edit3.Text+' '' ?',MTWARNING,[MBYES ,MBNO],0);
   IF SS15 = MRYES then
   BEGIN
     FORM15.ADOQuery3.Close;
     form15.ADOQuery3.SQL.Text:='delete from customer where
customer_id='+#39+form15.Edit1.Text+#39;
     form15.ADOQuery3.ExecSQL;
     showmessage('RECORD DELETED');
     Form15.LbSpeedButton4.Click;
   END;
 end
 else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;
```

```
procedure TForm15.LbSpeedButton4Click(Sender: TObject);
begin
  form15.Edit1.Clear;
  form15.Edit2.Clear;
  form15.Edit3.Clear;
  form15.MaskEdit1.Clear;
  form15.MaskEdit2.Clear;
  form15.MaskEdit3.Clear;
  form15.MaskEdit4.Clear;
  form15.Memo1.Clear;
  form15.Edit4.Clear;
  form15.ComboBox1.Text:='Select One';
  form15.ComboBox2.Text:='Select One';
  form15.Edit5.Clear;
  form15.Edit6.Clear;
  form15.Memo2.Clear;
  FORM15.Edit2.SetFocus;
  Form15.LbSpeedButton1.Enabled:=TRUE;
  Form15.LbSpeedButton2.Enabled:=TRUE;

  FORM15.ADOQuery1.Close;
  FORM15.ADOQuery1.SQL.Text:='select
customer_id,Cname,Csurname,Mobilphone,homephone,workphone,fax,address,city,tow
n,country,email,web,c_note from customer';
  form15.ADOQuery1.Open;
end;

procedure TForm15.DBGrid1CellClick(Column: TColumn);
begin
    IF  FORM15.ADOQuery1.RecordCount <> 0 THEN
  BEGIN
    FORM15.Edit1.Text:=FORM15.DBGrid1.Fields[0].Text;
    FORM15.Edit2.Text:=FORM15.DBGrid1.Fields[1].Text;
    FORM15.Edit3.Text:=FORM15.DBGrid1.Fields[2].Text;
    form15.MaskEdit1.Text:=FORM15.DBGrid1.Fields[3].Text;
    form15.MaskEdit2.Text:=FORM15.DBGrid1.Fields[4].Text;
    form15.MaskEdit3.Text:=FORM15.DBGrid1.Fields[5].Text;
    form15.MaskEdit4.Text:=FORM15.DBGrid1.Fields[6].Text;
    FORM15.Memo1.Text:=FORM15.DBGrid1.Fields[7].Text;
    FORM15.Edit4.Text:=FORM15.DBGrid1.Fields[9].Text;
    FORM15.ComboBox1.Text:=FORM15.DBGrid1.Fields[8].Text;
    FORM15.ComboBox2.Text:=FORM15.DBGrid1.Fields[10].Text;
    FORM15.Edit5.Text:=FORM15.DBGrid1.Fields[11].Text;
    FORM15.Edit6.Text:=FORM15.DBGrid1.Fields[12].Text;
    FORM15.Memo2.Text:=FORM15.DBGrid1.Fields[13].Text;
  END;

end;

procedure TForm15.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
  Form15.LbSpeedButton4.Click;
  form15.ADOQuery1.Close;
  form15.ADOQuery2.Close;
  form15.ADOQuery3.Close;
end;

procedure TForm15.FormHide(Sender: TObject);
begin
  Form15.LbSpeedButton4.Click;
  form15.ADOQuery1.Close;
  form15.ADOQuery2.Close;
  form15.ADOQuery3.Close;
end;

end.
```

FORM 16 CODES

```
unit Unit16;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, Grids, DBGrids, ExtCtrls, StdCtrls, Buttons, Menus,
  DB, ADODB;

type
  TForm16 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Edit1: TEdit;
```

```
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    SpeedButton1: TSpeedButton;
    Memo1: TMemo;
    Memo2: TMemo;
    Memo3: TMemo;
    Edit7: TEdit;
    Label17: TLabel;
    Edit8: TEdit;
    Memo4: TMemo;
    ComboBox2: TComboBox;
    Edit9: TEdit;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    StatusBar1: TStatusBar;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure SpeedButton3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure DBGrid1CellClick(Column: TColumn);

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

var
  Form16: TForm16;
  SS16:WORD;
```

```
implementation

uses Unit10, Unit40, Unit15;

{$R *.dfm}


procedure TForm16.SpeedButton3Click(Sender: TObject);
begin
 if (form16.Edit2.Text <> '') and (form16.Edit3.Text <> '') and (form16.Edit5.Text <>
'')then
 begin
   form16.ADOQuery2.Close;
   form16.ADOQuery2.SQL.Text:='select * from animal where
animal_name='+#39+form16.Edit2.Text+#39+' and
animal_kind='+#39+form16.Edit3.Text+#39+' and
owner_no='+#39+form16.Edit5.Text+#39;
   form16.ADOQuery2.Open;
   if form16.ADOQuery2.RecordCount = 0 then
   begin
    form16.ADOQuery2.Close;
    form16.ADOQuery2.SQL.Text:='insert into animal
(animal_name,animal_kind,animal_race,owner_no,abirthdate,animal_sex,animal_color,
animal_weight,collar_no,earning_no,life_state,animal_mark,animal_alergy,acronic_me
dicine,A_note) values
('+#39+Form16.Edit2.Text+#39+','+#39+form16.Edit3.Text+#39+','+#39+form16.Edit4
.Text+#39+','+#39+form16.Edit5.Text+#39+','+#39+datetostr(form16.DateTimePicker1
.Date)+#39+','+#39+form16.ComboBox1.Text+#39+','+#39+form16.Edit6.Text+#39+','
+#39+form16.Edit7.Text+#39+','+#39+form16.Edit8.Text+#39+','+#39+form16.Edit9.
Text+#39+','+#39+form16.ComboBox2.Text+#39+','+#39+form16.Memo1.Text+#39+',
'+#39+form16.Memo2.Text+#39+','+#39+form16.Memo3.Text+#39+','+#39+form16.M
emo4.Text+#39+')';
    form16.ADOQuery2.ExecSQL;
    form16.ADOQuery2.Close;
    form16.ADOQuery2.SQL.Text:='select * from animal';
    form16.ADOQuery2.Open;
    showmessage('RECORD SAVED');
    Form16.SpeedButton6.Click;
   END
   else
    SHOWMESSAGE('THE ANIMAL SAVED BEFORE');
  END
  ELSE if form16.Edit2.Text = '' then
   showmessage('PLEASE FILL THE ANIMAL NAME')
  ELSE if form16.Edit5.Text = '' then
   showmessage('PLEASE CHOOSE THE OWNER NO')
  else
   SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
```

```
end;

procedure TForm16.FormCreate(Sender: TObject);
begin
form16.DateTimePicker1.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';

end;

procedure TForm16.FormShow(Sender: TObject);
begin
 FORM16.ADOQuery1.Close;
 FORM16.ADOQuery1.SQL.Text:='select
animal_id,animal_name,animal_kind,animal_race,owner_no,animal_sex,animal_color,a
nimal_weight,animal_mark,animal_alergy,acronic_medicine,collar_no,earning_no,life_
state,a_note from animal';
 form16.ADOQuery1.Open;
end;

procedure TForm16.SpeedButton4Click(Sender: TObject);
begin
  IF (FORM16.Edit1.Text <> '') AND (FORM16.Edit2.Text <> '') AND
(FORM16.Edit3.Text <> '') AND (FORM16.Edit5.Text <> '') THEN
 BEGIN
  FORM16.ADOQuery3.Close;
  FORM16.ADOQuery3.SQL.Text:='UPDATE animal set animal_name=
'+#39+form16.Edit2.Text+#39+', animal_kind= '+#39+form16.Edit3.Text+#39+',
animal_race= '+#39+form16.Edit4.Text+#39+',
abirthdate='+#39+datetostr(form16.DateTimePicker1.Date)+#39+',
animal_sex='+#39+form16.ComboBox1.Text+#39+', animal_color=
'+#39+form16.Edit6.Text+#39+', animal_weight= '+#39+form16.Edit7.Text+#39+',
collar_no= '+#39+form16.Edit8.Text+#39+', earning_no=
'+#39+form16.Edit9.Text+#39+', life_state='+#39+form16.ComboBox2.Text+#39+',
animal_mark='+#39+form16.Memo1.Text+#39+',
animal_alergy='+#39+form16.Memo2.Text+#39+',
acronic_medicine='+#39+form16.Memo3.Text+#39+',
a_note='+#39+form16.Memo4.Text+#39+' WHERE
animal_id='+#39+form16.Edit1.Text+#39;
  form16.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  Form16.SpeedButton6.Click;
 END
 ELSE
  SHOWMESSAGE('PLEASE SELECT CUSTOMER AND BE SURE'+#13+'TO
FILL THE EMPTY PLACE');
end;

procedure TForm16.SpeedButton5Click(Sender: TObject);
```

```
begin
 IF (FORM16.Edit1.Text <> '') AND (FORM16.Edit2.Text <> '') AND
(FORM16.Edit5.Text <> '') then
 BEGIN
   SS16:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ID:
'+FORM16.Edit1.Text+' ; ANIMAL: ' +FORM16.Edit2.Text+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS16 = MRYES then
   BEGIN
     FORM16.ADOQuery3.Close;
     form16.ADOQuery3.SQL.Text:='delete from animal where
animal_id='+#39+form16.Edit1.Text+#39;
     form16.ADOQuery3.ExecSQL;
     showmessage('RECORD DELETED');
     form16.SpeedButton6.Click;
   END;
 end
 else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm16.SpeedButton6Click(Sender: TObject);
begin
 form16.Edit1.Clear;
 form16.Edit2.Clear;
 form16.Edit3.Clear;
 form16.Edit4.Clear;
 form16.Edit5.Clear;
 form16.DateTimePicker1.Date:=date;
 form16.ComboBox1.Text:='Select One';
 form16.Edit6.Clear;
 form16.Edit7.Clear;
 form16.Edit8.Clear;
 form16.Edit9.Clear;
 form16.ComboBox2.Text:='Select One';
 form16.Memo1.Clear;
 form16.Memo2.Clear;
 form16.Memo3.Clear;
 form16.Memo4.Clear;
 FORM16.Edit2.SetFocus;
 FORM16.SpeedButton3.Enabled:=TRUE;
 FORM16.SpeedButton4.Enabled:=TRUE;

 FORM16.ADOQuery1.Close;
 FORM16.ADOQuery1.SQL.Text:='select
animal_id,animal_name,animal_kind,animal_race,owner_no,animal_sex,animal_color,a
nimal_weight,animal_mark,animal_alergy,acronic_medicine,collar_no,earning_no,life_
state,a_note from animal';
 form16.ADOQuery1.Open;
```

```
end;

procedure TForm16.SpeedButton1Click(Sender: TObject);
begin
FORM40.SHOW;
end;

procedure TForm16.FormHide(Sender: TObject);
begin
 form16.SpeedButton6.Click;
 form16.ADOQuery1.Close;
 form16.ADOQuery2.Close;
 form16.ADOQuery3.Close;
end;

procedure TForm16.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 form16.SpeedButton6.Click;
 form16.ADOQuery1.Close;
 form16.ADOQuery2.Close;
 form16.ADOQuery3.Close;
end;

procedure TForm16.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM16.ADOQuery1.RecordCount <> 0 THEN
 BEGIN
   FORM16.Edit1.Text:=FORM16.DBGrid1.Fields[0].Text;
   FORM16.Edit2.Text:=FORM16.DBGrid1.Fields[1].Text;
   FORM16.Edit3.Text:=FORM16.DBGrid1.Fields[2].Text;
   FORM16.Edit4.Text:=FORM16.DBGrid1.Fields[3].Text;
   FORM16.Edit5.Text:=FORM16.DBGrid1.Fields[4].Text;
   FORM16.ComboBox1.Text:=FORM16.DBGrid1.Fields[5].Text;
   FORM16.Edit6.Text:=FORM16.DBGrid1.Fields[6].Text;
   FORM16.Edit7.Text:=FORM16.DBGrid1.Fields[7].Text;
   FORM16.Memo1.Text:=FORM16.DBGrid1.Fields[8].Text;
   FORM16.Memo2.Text:=FORM16.DBGrid1.Fields[9].Text;
   FORM16.Memo3.Text:=FORM16.DBGrid1.Fields[10].Text;
   FORM16.Edit8.Text:=FORM16.DBGrid1.Fields[11].Text;
   FORM16.Edit9.Text:=FORM16.DBGrid1.Fields[12].Text;
   FORM16.ComboBox2.Text:=FORM16.DBGrid1.Fields[13].Text;
   FORM16.Memo4.Text:=FORM16.DBGrid1.Fields[14].Text;
  end;
end;

end.
```

FORM 17 CODES

```
unit Unit17;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, LbSpeedButton, ExtCtrls, ComCtrls, StdCtrls,
  Buttons, Menus, DB, ADODB;

type
  TForm17 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit1: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    SpeedButton1: TSpeedButton;
    Memo1: TMemo;
    SpeedButton2: TSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    DBGrid1: TDBGrid;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
```

```
  procedure FormCreate(Sender: TObject);
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure DBGrid1CellClick(Column: TColumn);
  procedure LbSpeedButton3Click(Sender: TObject);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form17: TForm17;
 SS17:WORD;
implementation

uses Unit10, Unit38, Unit39;

{$R *.dfm}

procedure TUAH();
begin
 form17.ComboBox1.Items.Clear;
 FORM17.ADOQuery4.Close;
 FORM17.ADOQuery4.SQL.Text:='select vaccine_name from vaccines';
 form17.ADOQuery4.Open;
 while not form17.ADOQuery4.Eof do
 begin
  form17.ComboBox1.Items.Add(form17.ADOQuery4['vaccine_name']);
  form17.ADOQuery4.Next;
 end;
END;

procedure TForm17.SpeedButton1Click(Sender: TObject);
begin
 form38.show;
 TA:=17;
end;

procedure TForm17.FormShow(Sender: TObject);
begin
 form17.ADOQuery1.Close;
 form17.ADOQuery1.SQL.Text:='select * from vaccinate';
 form17.ADOQuery1.Open;
 TUAH();
```

```
end;



procedure TForm17.FormCreate(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 FORM17.DateTimePicker1.Date:=DATE;
 FORM17.DateTimePicker2.Date:=DATE;
end;

procedure TForm17.LbSpeedButton1Click(Sender: TObject);
begin

 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 if (form17.Edit1.Text <> '')  and (form17.ComboBox1.Text <> 'Select One') and
(form17.Edit4.Text <> '') AND (DATETOSTR(FORM17.DateTimePicker1.Date) <>
DATETOSTR(FORM17.DateTimePicker2.Date)) THEN
 begin
  form17.ADOQuery2.Close;
  form17.ADOQuery2.SQL.Text:='select * from vaccinate where
vaccine_serialno='+#39+form17.Edit2.Text+#39;
  form17.ADOQuery2.Open;
  if form17.ADOQuery2.RecordCount = 0 then
  begin
   form17.ADOQuery2.Close;
   form17.ADOQuery2.SQL.Text:='insert into vaccinate
(animal_id,vaccine_name,vaccinate_date,next_vaccinatedate,vaccine_serialno,vaccine_
producer,applied_staff,v_note) values
('+#39+form17.Edit1.Text+#39+','+#39+form17.ComboBox1.Text+#39+','+#39+dateto
str(form17.DateTimePicker1.Date)+#39+','+#39+datetostr(form17.DateTimePicker2.Da
te)+#39+','+#39+form17.Edit2.Text+#39+','+#39+form17.Edit3.Text+#39+','+#39+for
m17.Edit4.Text+#39+','+#39+form17.Memo1.Text+#39+')';
   form17.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form17.ADOQuery1.Close;
   form17.ADOQuery1.SQL.Text:='select * from vaccinate where
animal_id='+#39+form17.Edit1.Text+#39;
   form17.ADOQuery1.Open;
   TUAH();
  END
  ELSE
   SHOWMESSAGE('THE VACCINATE SAVED BEFORE');
 END
 ELSE if form17.Edit1.Text = '' then
```

```
   showmessage('PLEASE CHOOSE THE ANIMAL ID')
  ELSE if form17.Edit4.Text = '' then
   showmessage('PLEASE CHOOSE THE STAFF ID')
  else
   SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm17.SpeedButton2Click(Sender: TObject);
begin
 FORM39.SHOW;
 ANI:=17;
end;

procedure TForm17.Edit1Change(Sender: TObject);
begin
 form17.ADOQuery1.Close;
 form17.ADOQuery1.SQL.Text:='select * from vaccinate where
animal_id='+#39+form17.Edit1.Text+#39;
 form17.ADOQuery1.Open;
 TUAH();
end;

procedure TForm17.LbSpeedButton2Click(Sender: TObject);
begin

 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 IF (FORM17.Edit1.Text <> '') AND (FORM17.Edit2.Text <> '') AND
(FORM17.ComboBox1.Text <> 'Select One') THEN
 BEGIN
   FORM17.ADOQuery3.Close;
   FORM17.ADOQuery3.SQL.Text:='UPDATE vaccinate set
Animal_id='+#39+form17.Edit1.Text+#39+',
Vaccine_name='+#39+form17.ComboBox1.Text+#39+',
Vaccinate_date='+#39+datetostr(form17.DateTimePicker1.Date)+#39+',
Next_vaccinatedate='+#39+datetostr(form17.DateTimePicker2.Date)+#39+',
Vaccine_serialno='+#39+form17.Edit2.Text+#39+',
Vaccine_producer='+#39+form17.Edit3.Text+#39+',
Applied_staff='+#39+form17.Edit4.Text+#39+',
V_note='+#39+form17.Memo1.Text+#39+' where
Animal_id='+#39+form17.DBGrid1.Fields[0].Text+#39+' and
Vaccine_name='+#39+form17.DBGrid1.Fields[1].Text+#39+' and
Vaccinate_date='+#39+form17.DBGrid1.Fields[2].Text+#39+' and
Vaccine_serialno='+#39+form17.DBGrid1.Fields[4].Text+#39;
   form17.ADOQuery3.ExecSQL;
   showmessage('RECORD UPDATED');
   FORM17.LbSpeedButton4.Click;
 END
```

```
 ELSE
   SHOWMESSAGE('PLEASE CHOOSE VACCINATE FROM LIST');
end;

procedure TForm17.DBGrid1CellClick(Column: TColumn);
begin
  IF  FORM17.ADOQuery1.RecordCount <> 0 THEN
  BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM17.Edit1.Text:=FORM17.DBGrid1.Fields[0].Text;
   FORM17.ComboBox1.Text:=FORM17.DBGrid1.Fields[1].Text;

FORM17.DateTimePicker1.Date:=STRTODATE(FORM17.DBGrid1.Fields[2].Text);

FORM17.DateTimePicker2.Date:=STRTODATE(FORM17.DBGrid1.Fields[3].Text);
   FORM17.Edit2.Text:=FORM17.DBGrid1.Fields[4].Text;
   FORM17.Edit3.Text:=FORM17.DBGrid1.Fields[5].Text;
   FORM17.Edit4.Text:=FORM17.DBGrid1.Fields[6].Text;
   FORM17.Memo1.Text:=FORM17.DBGrid1.Fields[7].Text;
  END;

end;

procedure TForm17.LbSpeedButton3Click(Sender: TObject);
begin
  IF (FORM17.Edit1.Text <> '') AND (FORM17.combobox1.Text <> 'Select One')
AND (FORM17.Edit4.Text <> '') then
  BEGIN
   SS17:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM17.Edit1.Text+' ; VACCINE: ' +FORM17.COMBOBOX1.Text+'  ;
VACCINATE DATE: '+DATETOSTR(FORM17.DateTimePicker1.Date)+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS17 = MRYES then
   BEGIN
    FORM17.ADOQuery3.Close;
    form17.ADOQuery3.SQL.Text:='delete from vaccinate where
animal_id='+#39+form17.Edit1.Text+#39+' and
vaccine_name='+#39+form17.ComboBox1.Text+#39;
    //FORM17.ADOQuery3.SQL.Text:='DELETE FROM vaccinate where
Animal_id='+#39+form17.Edit1.Text+#39+' and
Vaccine_name='+#39+form17.ComboBox1.Text+#39+' and
Vaccinate_date='+#39+datetostr(form17.DateTimePicker1.Date)+#39+' and
Vaccine_serialno='+#39+form17.Edit2.Text+#39;
    form17.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form17.LbSpeedButton4.Click;
```

```
  END;
 end
 else
  showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm17.LbSpeedButton4Click(Sender: TObject);
begin
 FORM17.Edit1.Clear;
 FORM17.ComboBox1.Text:='Select One';
 form17.DateTimePicker1.Date:=date;
 form17.DateTimePicker2.Date:=date;
 FORM17.Edit2.Clear;
 FORM17.Edit3.Clear;
 FORM17.Edit4.Clear;
 FORM17.Memo1.Clear;
 form17.ComboBox1.SetFocus;
 Form17.LbSpeedButton1.Enabled:=TRUE;
 Form17.LbSpeedButton2.Enabled:=TRUE;

 form17.ADOQuery1.Close;
 form17.ADOQuery1.SQL.Text:='select * from vaccinate';
 form17.ADOQuery1.Open;
end;

procedure TForm17.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 form17.LbSpeedButton4.Click;
 form17.ADOQuery1.Close;
 form17.ADOQuery2.Close;
 form17.ADOQuery3.Close;
 form17.ADOQuery4.Close;
end;

procedure TForm17.FormHide(Sender: TObject);
begin
 form17.LbSpeedButton4.Click;
 form17.ADOQuery1.Close;
 form17.ADOQuery2.Close;
 form17.ADOQuery3.Close;
 form17.ADOQuery4.Close;
end;

end.
```

FORM 18 CODES

```
unit Unit18;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, ExtCtrls, ComCtrls, Buttons, StdCtrls, Menus,
  DB, ADODB;

type
  TForm18 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit1: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Edit2: TEdit;
    Memo1: TMemo;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    DBGrid1: TDBGrid;
    Label7: TLabel;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    Edit3: TEdit;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
```

```
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton6Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form18: TForm18;
  SS18:WORD;
implementation

uses Unit10, Unit39, Unit38;

{$R *.dfm}

PROCEDURE TUAHIN();
BEGIN
  form18.ComboBox1.Items.Clear;
  FORM18.ADOQuery4.Close;
  FORM18.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'INNER PARASITE'+#39;
  form18.ADOQuery4.Open;
  while not form18.ADOQuery4.Eof do
  begin
    form18.ComboBox1.Items.Add(form18.ADOQuery4['drug_name']);
    form18.ADOQuery4.Next;
  end;
END;

procedure TForm18.FormShow(Sender: TObject);
begin
  form18.ADOQuery1.Close;
  form18.ADOQuery1.SQL.Text:='select * from ipdrug';
  form18.ADOQuery1.Open;
  TUAHIN();
end;

procedure TForm18.SpeedButton3Click(Sender: TObject);
begin
   dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'yyyy/mm/dd';
```

```
 if (form18.Edit1.Text <> '')  and (form18.ComboBox1.Text <> 'Select One') and
(form18.Edit2.Text <> '') AND (DATETOSTR(FORM18.DateTimePicker1.Date) <>
DATETOSTR(FORM18.DateTimePicker2.Date)) THEN
 begin
  form18.ADOQuery2.Close;
  form18.ADOQuery2.SQL.Text:='select * from ipdrug where
animal_id='+#39+form18.Edit1.Text+#39+'and
ip_drugname='+#39+form18.ComboBox1.Text+#39+' and
ip_drugdate='+#39+datetostr(form18.DateTimePicker1.Date)+#39+' and
ip_nextdrugdate='+#39+datetostr(form18.DateTimePicker2.Date)+#39;
  form18.ADOQuery2.Open;
  if form18.ADOQuery2.RecordCount = 0 then
  begin
   form18.ADOQuery2.Close;
   form18.ADOQuery2.SQL.Text:='insert into ipdrug
(animal_id,ip_drugname,ip_drugdate,ip_nextdrugdate,applied_staff,ip_drugnote) values
('+#39+form18.Edit1.Text+#39+','+#39+form18.ComboBox1.Text+#39+','+#39+dateto
str(form18.DateTimePicker1.Date)+#39+','+#39+datetostr(form18.DateTimePicker2.Da
te)+#39+','+#39+form18.Edit2.Text+#39+','+#39+form18.Memo1.Text+#39+')';
   form18.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form18.ADOQuery1.Close;
   form18.ADOQuery1.SQL.Text:='select * from ipdrug where
animal_id='+#39+form18.Edit1.Text+#39;
   form18.ADOQuery1.Open;
   TUAHIN();
  END
  ELSE
   SHOWMESSAGE('THE INNER PARASITE APPLICATION SAVED BEFORE');
 END
 ELSE if form18.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE ANIMAL ID')
 ELSE if form18.Edit2.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm18.SpeedButton1Click(Sender: TObject);
begin
form39.show;
ANI:=18;
end;

procedure TForm18.SpeedButton2Click(Sender: TObject);
begin
 form38.show;
 TA:=18;
end;
```

```
procedure TForm18.SpeedButton4Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

  IF (FORM18.Edit1.Text <> '') AND (FORM18.Edit3.Text <> '') AND
(FORM18.Edit2.Text <> '') AND (FORM18.ComboBox1.Text <> 'Select One') THEN
  BEGIN
    FORM18.ADOQuery3.Close;
    FORM18.ADOQuery3.SQL.Text:='UPDATE ipdrug set
Animal_id='+#39+form18.Edit1.Text+#39+',
ip_drugname='+#39+form18.ComboBox1.Text+#39+',
ip_drugdate='+#39+datetostr(form18.DateTimePicker1.Date)+#39+',
ip_nextdrugdate='+#39+datetostr(form18.DateTimePicker2.Date)+#39+',
Applied_staff='+#39+form18.Edit2.Text+#39+',
ip_drugnote='+#39+form18.Memo1.Text+#39+' where
Ip_id='+#39+form18.Edit3.Text+#39;
    form18.ADOQuery3.ExecSQL;
    showmessage('RECORD UPDATED');
    FORM18.SpeedButton6.Click;
  END
  ELSE
    SHOWMESSAGE('PLEASE SELECT INNER PARASITE APPLICATION FROM
LIST');
end;

procedure TForm18.SpeedButton5Click(Sender: TObject);
begin
  IF (FORM18.Edit1.Text <> '') AND (FORM18.Edit3.Text <> '') then
  BEGIN
    SS18:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM18.Edit1.Text+' ; INNER DRUG: ' +FORM18.COMBOBOX1.Text+'  ; IP
DRUG DATE: '+DATETOSTR(FORM18.DateTimePicker1.Date)+' "
?',MTWARNING,[MBYES ,MBNO],0);
    IF SS18 = MRYES then
    BEGIN
      FORM18.ADOQuery3.Close;
      form18.ADOQuery3.SQL.Text:='delete from ipdrug where
Ip_id='+#39+form18.Edit3.Text+#39;
      form18.ADOQuery3.ExecSQL;
      showmessage('RECORD DELETED');
      Form18.SpeedButton6.Click;
    END;
  end
  else
    showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;
```

```
procedure TForm18.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM18.ADOQuery1.RecordCount <> 0 THEN
 BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM18.Edit3.Text:=FORM18.DBGrid1.Fields[0].Text;
   FORM18.Edit1.Text:=FORM18.DBGrid1.Fields[1].Text;
   FORM18.ComboBox1.Text:=FORM18.DBGrid1.Fields[2].Text;

FORM18.DateTimePicker1.Date:=STRTODATE(FORM18.DBGrid1.Fields[3].Text);

FORM18.DateTimePicker2.Date:=STRTODATE(FORM18.DBGrid1.Fields[4].Text);
   FORM18.Edit2.Text:=FORM18.DBGrid1.Fields[5].Text;
   FORM18.Memo1.Text:=FORM18.DBGrid1.Fields[6].Text;
 END;
end;

procedure TForm18.SpeedButton6Click(Sender: TObject);
begin
 FORM18.Edit3.Clear;
 FORM18.Edit1.Clear;
 FORM18.ComboBox1.Text:='Select One';
 form18.DateTimePicker1.Date:=date;
 form18.DateTimePicker2.Date:=date;
 FORM18.Edit2.Clear;
 FORM18.Memo1.Clear;
 form18.ComboBox1.SetFocus;
 Form18.SpeedButton3.Enabled:=TRUE;
 Form18.SpeedButton4.Enabled:=TRUE;

 form18.ADOQuery1.Close;
 form18.ADOQuery1.SQL.Text:='select * from ipdrug';
 form18.ADOQuery1.Open;
end;

procedure TForm18.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 FORM18.SpeedButton6.Click;
 FORM18.ADOQuery1.Close;
 FORM18.ADOQuery2.Close;
 FORM18.ADOQuery3.Close;
 FORM18.ADOQuery4.Close;
end;

procedure TForm18.FormHide(Sender: TObject);
```

```
begin
  FORM18.SpeedButton6.Click;
  FORM18.ADOQuery1.Close;
  FORM18.ADOQuery2.Close;
  FORM18.ADOQuery3.Close;
  FORM18.ADOQuery4.Close;
end;

procedure TForm18.FormCreate(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';
  FORM18.DateTimePicker1.Date:=DATE;
  FORM18.DateTimePicker2.Date:=DATE;
end;

procedure TForm18.Edit1Change(Sender: TObject);
begin
  form18.ADOQuery1.Close;
  form18.ADOQuery1.SQL.Text:='select * from ipdrug where
animal_id='+#39+form18.Edit1.Text+#39;
  form18.ADOQuery1.Open;
  TUAHIN();
end;

end.
```

FORM 19 CODES

```
unit Unit19;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Menus, Grids, DBGrids, LbSpeedButton,
  ExtCtrls, Buttons, DB, ADODB;

type
  TForm19 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
```

```
  StatusBar1: TStatusBar;
  Edit1: TEdit;
  ComboBox1: TComboBox;
  DateTimePicker1: TDateTimePicker;
  DateTimePicker2: TDateTimePicker;
  Edit2: TEdit;
  Memo1: TMemo;
  SpeedButton1: TSpeedButton;
  SpeedButton2: TSpeedButton;
  Panel1: TPanel;
  LbSpeedButton1: TLbSpeedButton;
  LbSpeedButton2: TLbSpeedButton;
  LbSpeedButton3: TLbSpeedButton;
  LbSpeedButton4: TLbSpeedButton;
  DBGrid1: TDBGrid;
  Label7: TLabel;
  Edit3: TEdit;
  ADOQuery1: TADOQuery;
  ADOQuery2: TADOQuery;
  ADOQuery3: TADOQuery;
  ADOQuery4: TADOQuery;
  DataSource1: TDataSource;
  DataSource2: TDataSource;
  DataSource3: TDataSource;
  DataSource4: TDataSource;
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure LbSpeedButton3Click(Sender: TObject);
  procedure DBGrid1CellClick(Column: TColumn);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form19: TForm19;
 SS19:WORD;
implementation

uses Unit10, Unit18, Unit39, Unit38;
```

```
{$R *.dfm}

PROCEDURE TUAHOUT();
BEGIN
 form19.ComboBox1.Items.Clear;
 FORM19.ADOQuery4.Close;
 FORM19.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'OUTER PARASITE'+#39;
 form19.ADOQuery4.Open;
 while not form19.ADOQuery4.Eof do
 begin
  form19.ComboBox1.Items.Add(form19.ADOQuery4['drug_name']);
  form19.ADOQuery4.Next;
 end;
END;


procedure TForm19.LbSpeedButton1Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

 if (form19.Edit1.Text <> '')  and (form19.ComboBox1.Text <> 'Select One') and
(form19.Edit2.Text <> '') AND (DATETOSTR(FORM19.DateTimePicker1.Date) <>
DATETOSTR(FORM19.DateTimePicker2.Date)) THEN
 begin
  form19.ADOQuery2.Close;
  form19.ADOQuery2.SQL.Text:='select * from opdrug where
animal_id='+#39+form19.Edit1.Text+#39+'and
op_drugname='+#39+form19.ComboBox1.Text+#39+' and
op_drugdate='+#39+datetostr(form19.DateTimePicker1.Date)+#39+' and
op_nextdrugdate='+#39+datetostr(form19.DateTimePicker2.Date)+#39;
  form19.ADOQuery2.Open;
  if form19.ADOQuery2.RecordCount = 0 then
  begin
   form19.ADOQuery2.Close;
   form19.ADOQuery2.SQL.Text:='insert into opdrug
(animal_id,op_drugname,op_drugdate,op_nextdrugdate,applied_staff,op_drugnote)
values
('+#39+form19.Edit1.Text+#39+','+#39+form19.ComboBox1.Text+#39+','+#39+dateto
str(form19.DateTimePicker1.Date)+#39+','+#39+datetostr(form19.DateTimePicker2.Da
te)+#39+','+#39+form19.Edit2.Text+#39+','+#39+form19.Memo1.Text+#39+')';
   form19.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form19.ADOQuery1.Close;
   form19.ADOQuery1.SQL.Text:='select * from opdrug where
animal_id='+#39+form19.Edit1.Text+#39;
   form19.ADOQuery1.Open;
   TUAHOUT();
```

```
  END
  ELSE
   SHOWMESSAGE('THE INNER PARASITE APPLICATION SAVED BEFORE');
 END
 ELSE if form19.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE ANIMAL ID')
 ELSE if form19.Edit2.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm19.FormShow(Sender: TObject);
begin
 form19.ADOQuery1.Close;
 form19.ADOQuery1.SQL.Text:='select * from opdrug';
 form19.ADOQuery1.Open;
 TUAHOUT();
end;

procedure TForm19.LbSpeedButton2Click(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 IF (FORM19.Edit1.Text <> '') AND (FORM19.Edit3.Text <> '') AND
(FORM19.Edit2.Text <> '') AND (FORM19.ComboBox1.Text <> 'Select One') THEN
 BEGIN
  FORM19.ADOQuery3.Close;
  FORM19.ADOQuery3.SQL.Text:='UPDATE opdrug set
Animal_id='+#39+form19.Edit1.Text+#39+',
op_drugname='+#39+form19.ComboBox1.Text+#39+',
op_drugdate='+#39+datetostr(form19.DateTimePicker1.Date)+#39+',
op_nextdrugdate='+#39+datetostr(form19.DateTimePicker2.Date)+#39+',
Applied_staff='+#39+form19.Edit2.Text+#39+',
op_drugnote='+#39+form19.Memo1.Text+#39+' where
Op_id='+#39+form19.Edit3.Text+#39;
  form19.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  Form19.LbSpeedButton4.Click;
 END
 ELSE
  SHOWMESSAGE('PLEASE SELECT OUTER PARASITE APPLICATION FROM
LIST');
end;

procedure TForm19.LbSpeedButton3Click(Sender: TObject);
begin
 IF (FORM19.Edit1.Text <> '') AND (FORM19.Edit3.Text <> '') then
```

```
 BEGIN
   SS19:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM19.Edit1.Text+' ; OUTER DRUG: ' +FORM19.COMBOBOX1.Text+'  ; OP
DRUG DATE: '+DATETOSTR(FORM19.DateTimePicker1.Date)+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS19 = MRYES then
   BEGIN
    FORM19.ADOQuery3.Close;
    form19.ADOQuery3.SQL.Text:='delete from opdrug where
Op_id='+#39+form19.Edit3.Text+#39;
    form19.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form19.LbSpeedButton4.Click;
   END;
 end
 else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm19.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM19.ADOQuery1.RecordCount <> 0 THEN
 BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM19.Edit3.Text:=FORM19.DBGrid1.Fields[0].Text;
   FORM19.Edit1.Text:=FORM19.DBGrid1.Fields[1].Text;
   FORM19.ComboBox1.Text:=FORM19.DBGrid1.Fields[2].Text;

FORM19.DateTimePicker1.Date:=STRTODATE(FORM19.DBGrid1.Fields[3].Text);

FORM19.DateTimePicker2.Date:=STRTODATE(FORM19.DBGrid1.Fields[4].Text);
   FORM19.Edit2.Text:=FORM19.DBGrid1.Fields[5].Text;
   FORM19.Memo1.Text:=FORM19.DBGrid1.Fields[6].Text;
 END;
end;

procedure TForm19.LbSpeedButton4Click(Sender: TObject);
begin
 FORM19.Edit3.Clear;
 FORM19.Edit1.Clear;
 FORM19.ComboBox1.Text:='Select One';
 form19.DateTimePicker1.Date:=date;
 form19.DateTimePicker2.Date:=date;
 FORM19.Edit2.Clear;
 FORM19.Memo1.Clear;
```

```
 form19.ComboBox1.SetFocus;
 Form19.LbSpeedButton1.Enabled:=TRUE;
 Form19.LbSpeedButton2.Enabled:=TRUE;

 form19.ADOQuery1.Close;
 form19.ADOQuery1.SQL.Text:='select * from opdrug';
 form19.ADOQuery1.Open;
end;

procedure TForm19.SpeedButton1Click(Sender: TObject);
begin
 form39.show;
 ANI:=19;
end;

procedure TForm19.SpeedButton2Click(Sender: TObject);
begin
 form38.show;
 TA:=19;
end;

procedure TForm19.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 FORM19.LbSpeedButton4.Click;
 FORM19.ADOQuery1.Close;
 FORM19.ADOQuery2.Close;
 FORM19.ADOQuery3.Close;
 FORM19.ADOQuery4.Close;
end;

procedure TForm19.FormHide(Sender: TObject);
begin
 FORM19.LbSpeedButton4.Click;
 FORM19.ADOQuery1.Close;
 FORM19.ADOQuery2.Close;
 FORM19.ADOQuery3.Close;
 FORM19.ADOQuery4.Close;
end;

procedure TForm19.FormCreate(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 FORM19.DateTimePicker1.Date:=DATE;
 FORM19.DateTimePicker2.Date:=DATE
end;

procedure TForm19.Edit1Change(Sender: TObject);
begin
```

```
 form19.ADOQuery1.Close;
 form19.ADOQuery1.SQL.Text:='select * from opdrug where
animal_id='+#39+form19.Edit1.Text+#39;
 form19.ADOQuery1.Open;
 TUAHOUT();
end;

end.
```

FORM 20 CODES

```
unit Unit20;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, Grids, DBGrids, ComCtrls, ExtCtrls, Buttons, StdCtrls, Menus,
 DB, ADODB;

type
 TForm20 = class(TForm)
   MainMenu1: TMainMenu;
   F1: TMenuItem;
   Label1: TLabel;
   Label2: TLabel;
   Label3: TLabel;
   Label4: TLabel;
   Label5: TLabel;
   Edit1: TEdit;
   ComboBox1: TComboBox;
   DateTimePicker1: TDateTimePicker;
   Edit2: TEdit;
   Memo1: TMemo;
   SpeedButton1: TSpeedButton;
   SpeedButton2: TSpeedButton;
   SpeedButton3: TSpeedButton;
   SpeedButton4: TSpeedButton;
   SpeedButton5: TSpeedButton;
   SpeedButton6: TSpeedButton;
   Panel1: TPanel;
   StatusBar1: TStatusBar;
   DBGrid1: TDBGrid;
   Label6: TLabel;
   Edit3: TEdit;
   ADOQuery1: TADOQuery;
   ADOQuery2: TADOQuery;
   ADOQuery3: TADOQuery;
   ADOQuery4: TADOQuery;
```

```
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    procedure SpeedButton3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form20: TForm20;
  SS20:WORD;
implementation

uses Unit10, Unit18, Unit39, Unit38;

{$R *.dfm}

PROCEDURE TUAHMED();
BEGIN
  form20.ComboBox1.Items.Clear;
  FORM20.ADOQuery4.Close;
  FORM20.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'GENERAL DRUG'+#39;
  form20.ADOQuery4.Open;
  while not form20.ADOQuery4.Eof do
  begin
    form20.ComboBox1.Items.Add(form20.ADOQuery4['drug_name']);
    form20.ADOQuery4.Next;
  end;
END;

procedure TForm20.SpeedButton3Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';
```

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

VETERINERIAN APPLICATION PROGRAM WITH DELPHI

Graduation Project
COM400

Student: Ahmet KAYABAŞ (20021329)

Supervisor: Mr. Elburus IMANOV

Lefkoşa-2007

TABLE OF CONTENT

ACKNOWLEDGMENT

When people start a new work they get excited.Because who do not know any thing about the future of work.When a time passed human becomes familiar for this work.Afterthat may be borred,maybe want to leave this work.That may be true maybe false.It changes from people to people.But I believe that the important thing in the life do not leave such who should embrace very tightly.When we get this it makes us happy.

In the life what is important for you.Business? Money? Science? Power? Family? Love? Humanity? or purpose of existence? In my opinion first of all aim of existence comes.Rest of all things involved in aim of existence.After that comes Love.The world exists of love.With love person gets power, gains working perseverence .

Well in this project I gained perseverence from ***Allah*** *and from my fiancee.I am happy to complete the task which I had given with blessing of* ***Allah*** *and also I am grateful to my fiancee and all the people in my life who have supported me, advised me. They all the time helped and encouraged me to follow my dreams and ambitions.*

For intellectual support, encouragement I want to thank to my supervisor ***Mr. Elburus Imanov*** *who made this project contributions.*

And thank ***my dearest parents*** *who supported me to continue beyond my undergraduate studies, and also many thanks to my dear familiy who brought me till such meaning days.*

To all my friends, especially ***M.Fethullah Akatay, Selman Kayabaş, Metin Yenigün , Kadir Bekiroğlu and My dear fiancee*** *for sharing wonderful moments, advice, and for making me feel at home and in life. And above, I thank God for giving me stamina and courage to achieve my objectives.*

AHMET KAYABAŞ

ABSTRACT

In the world not only human life is important.In the same time other entity lives with us.We are not alone on the earth.Animals share life with us.Ilnesses are not only for human.In the same time whole alive interested with illnesses.How Doctor is important for us like Veterinerian is important for animals.Todays Doctors use application program.Because of to keep knowledge of patient, to facility diagnosis of illness, to reach background of patient efficiently and easly.

Well Veterinerian application program is important like the program that is used human health.Also much more important then others.Because animal can not keep the illnesses knowledge.And also papers of the animal can lost.

This project has as its goal to develop software, processing information about activities of a veterinerian application software. Software developed in this project like not only for animal.In the same time for staff and for owner of the animal.All records keep in the other Database program.It acts easly and fast access.Veterinerian can keep all records in the program as concentment.

INTRODUCTION

Since human created by the powerful Allah, Human wonder everything.Well who tried to satisfy wonder.Such humanity came to nowadays as develop.Todays everyone says technology perfect developed.Yes that is right.By means of technology all process gained velocity.This development acts to spend time to the people.

Technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.The automation is also became a part of our lives. The people operate with automation systems in everywhere.

Veterinerian Application project which is my project.In this software veterinerian can keep animal knowledge, patient background knowledge of the animal, owner of the animal knowledge.With this software veterinerian will make record process easily and safetly.

In Software there are five types user.They can access to only their task process.In the same time in the program veterinerian can get obligation as daily.The software can be used at every animal clinic easly.

CHAPTER 1

DELPHI

1.1 INTRODUCTION TO DELPHI

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

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

It is a method for predicting future events.

It is a method for generating a quick consensus by a group.

It is the use of a survey to collect information.

It is the use of anonymity on the part of the participants.

It is the use of voting to reduce the need for long discussions.

It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing detailed critical examination and discussion, not at forcing a quick

compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective:

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

Additional opportunity has been added by the introduction of Computer Mediated Communication Systems (Hiltz and Turoff, 1978; Rice and Associates, 1984; Turoff, 1989; Turoff, 1991). These are computer systems that support group communications in either a synchronous (Group Decision Support Systems, Desanctis et. al., 1987) or an asynchronous manner (Computer Conferencing). Techniques that were developed and refined in the evolution of the Delphi Method (e.g. anonymity, voting) have been incorporated as basic facilities or tools in many of these computer based systems. As a result, any of these systems can be used to carry out some form of a Delphi process or Nominal Group Technique (Delbecq, et. al., 1975).

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

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

1.2 WHAT IS DELPHI?

Delphi is an object oriented, 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 What's more, 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 tools.

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!

1.3 WHAT KIND OF PROGRAMMING CAN YOU DO WITH DELPHI?

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

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

Some projects to which Delphi is suited:

- Simple, single user database applications

- Intermediate multi-user database applications

- Large scale multi-tier, multi-user database applications

- Internet applications

- Graphics Applications

- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools

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.

1.4 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 quitcapable ofthis.

1.5 SOME KNOWLEDGE ABOUT DELPHI

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

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

For the purposes of this series I will be using Delphi 7. There are more recent versions available (2005 and 2006) however Delphi 7 should be available inexpensively compared to the new versions which will set you back a lot of money. Delphi 7 will more than likely be available in a magazine for free.

1.5.2 Example: Try First Delphi Program

First thing is first, fire up your copy of Delphi and open the Project > Options menu. To compile a console application you need to change a setting on the Linker tab called 'Generate console application', check the box and click OK. Now select File > Close All if anything is already loaded. Then select File > New > Other > Console Application.

Notice the first line refers to the keyword program. You can rename this to HelloWorld. You can also remove the commented portion enclosed in curly brackets. The uses keyword allows you to list all units that you want to use in the program. At the moment just leave it as it is, SysUtils is all we need.

Your unit should now look like this:

Delphi Code:

```
program HelloWorld;

  {$APPTYPE CONSOLE}

 uses

  SysUtils;

 begin

 end.
```

Now what we have just done is written a program, it currently doesn't do a thing however. Hit the run button and see the result. Now wasn't that completely worthless.

Luckily this isn't the end of the article so we'll actually have a worthwhile program at the end of it. All we need to do is insert some code in the main procedure we have just made.

Every good programmer's first program was 'Hello World' and you'll be no exception. All we need to do is use the WriteLn procedure to write 'Hello World!' to the console, simple.Notice the semicolon at the end of the line, at the end of any statement you need to add a semicolon. Run the program and see the results…

Now I don't know about you but I saw hello world flash up and go away in a second, if you didn't write the program you wouldn't even know what it said. To solve this problem we need to tell the program to leave the console open until the user is ready to close it. We can use ReadLn for this which reads the users input from the console.

Delphi Code:

```
program HelloWorld;

 {$APPTYPE CONSOLE}

 uses

   SysUtils;

 begin

   WriteLn('Hello World!' + #13#10 + #13#10 +

     'Press RETURN to end...');

   ReadLn;

 end.
```

I have added a few extra things into the 'Hello World' string so the user knows what to do to end the program as it could be a bit confusing. '#13#10' is to insert a carriage

return as 13 and 10 are the ASCII codes for a carriage return followed by a new line feed. ASCII can be inserted in this way into strings.

1.5.2 Delphi Style

Coding style, the way you format your code and the way in which you present it on the page.At the end of the day who cares about my style, I can read it, and Delphi strips all the spaces out of it and doesn't care if I indent. Why waste my time?

Neatly present code which conforms to the accepted standards not only makes your code much easier for you to read and debug but also but any one else who might read your code to help you, or learn from you can do so with ease. After all which code is easier to follow, example 1 or 2?

Delphi Code:

```
// Example 1

procedure xyz();

var

x,y,z,a:integer;

begin

x:=1;y:=2;

for z:=x to y do begin

a:=power(z,y);

showmessage(inttostr(a));

end;
```

```
end;
```

Delphi Code:

```
// Example 2
procedure XYZ();
var
X,Y,Z,A: Integer;
begin
  X := 1;
  Y := 2;
  for Z := X to Y do
  begin
        A := Power(Z, Y);
        ShowMessage(IntToStr(A));
  end; // for end
end; // procedure end
```

Design patterns are frequently recurring structures and relationships in object-oriented design. Getting to know them can help you design better, more reusable code and also help you learn to design more complex systems.

Much of the ground-breaking work on design patterns was presented in the book Design Patterns: Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides. You might also have heard of the authors referred to as "the Gang of Four". If you haven't read this book before and you're designing objects, it's an excellent

primer to help structure your design. To get the most out of these examples, I recommend reading the book as well.

Another good source of pattern concepts is the book Object Models: Strategies, Patterns and Applications by Peter Coad. Coad's examples are more business oriented and he emphasises learning strategies to identify patterns in your own work.

1.6 HOW DELPHI HELPS YOU DEFINE PATTERNS

Delphi implements a fully object-oriented language with many practical refinements that simplify development.

The most important class attributes from a pattern perspective are the basic inheritance of classes; virtual and abstract methods; and use of protected and public scope. These give you the tools to create patterns that can be reused and extended, and let you isolate varying functionality from base attributes that are unchanging.

Delphi is a great example of an extensible application, through its component architecture, IDE interfaces and tool interfaces. These interfaces define many virtual and abstract constructors and operations.

1.6.1 Delphi Examples of Design Patterns

I should note from the outset, there may be alternative or better ways to implement these patterns and I welcome your suggestions on ways to improve the design. The following patterns from the book *Design Patterns* are discussed and illustrated in Delphi to give you a starting point for implementing your own Delphi patterns.

Pattern Name	*Definition*
Singleton	"Ensure a class has only one instance, and provide a global point of access to it."
Adapter	"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't

	otherwise because of incompatible interfaces."
Template Method	"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."
Builder	"Separate the construction of a complex object from its representation so that the same construction process can create different representations."
Abstract Factory	"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."
Factory Method	"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

Note: These definitions are taken from *Design Patterns*.

1.6.2 Pattern: Singleton

1.6.2.1 Definition

"Ensure a class has only one instance, and provide a global point of access to it."

This is one of the easiest patterns to implement.

1.6.2.2 Applications in Delphi

There are several examples of this sort of class in the Delphi VCL, such as TApplication, TScreen or TClipboard. The pattern is useful whenever you want a single global object in your application. Other uses might include a global exception handler, application security, or a single point of interface to another application.

1.6.2.3 Implementation Example

To implement a class of this type, override the constructor and destructor of the class to refer to a global (interface) variable of the class.

Abort the constructor if the variable is assigned, otherwise create the instance and assign the variable.

In the destructor, clear the variable if it refers to the instance being destroyed.

Note: To make the creation and destruction of the single instance automatic, include its creation in the initialization section of the unit. To destroy the instance, include its destruction in an ExitProc (Delphi 1) or in the finalization section of the unit (Delphi 2).

1.6.3 Pattern: Adapter

1.6.3.1 Definition

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

1.6.3.2 Applications in Delphi

A typical example of this is the wrapper Delphi generates when you import a VBX or OCX. Delphi generates a new class which translates the interface of the external control into a Pascal compatible interface. Another typical case is when you want to build a single interface to old and new systems.

Note Delphi does not allow class adaption through multiple inheritance in the way described in Design Patterns. Instead, the adapter needs to refer to a specific instance of the old class.

1.6.3.3 Implementation Example

The following example is a simple (read only) case of a new customer class, an adapter class and an old customer class. The adapter illustrates handling the year 2000 problem, translating an old customer record containing two digit years into a new date format. The client using this wrapper only knows about the new customer class. Translation between classes is handled by the use of virtual access methods for the properties. The old customer class and adapter class are hidden in the implementation of the unit.

1.6.4 Pattern: Template Method

1.6.4.1 Definition

"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

This pattern is essentially an extension of abstract methods to more complex algorithms.

1.6.4.2 Applications in Delphi

Abstraction is implemented in Delphi by abstract virtual methods. Abstract methods differ from virtual methods by the base class not providing any implementation. The descendant class is completely responsible for implementing an abstract method. Calling an abstract method that has not been overridden will result in a runtime error.

1.6.4.3 A typical example of abstraction is the TGraphic class.

TGraphic is an abstract class used to implement TBitmap, TIcon and TMetafile. Other developers have frequently used TGraphic as the basis for other graphics objects such as PCX, GIF, JPG representations. TGraphic defines abstract methods such as Draw, LoadFromFile and SaveToFile which are then overridden in the concrete classes. Other objects that use TGraphic, such as a TCanvas only know about the abstract Draw method, yet are used with the concrete class at runtime.

Many classes that use complex algorithms are likely to benefit from abstraction using the template method approach. Typical examples include data compression, encryption and advanced graphics processing.

1.6.4.4 Implementation Example

To implement template methods you need an abstract class and concrete classes for each alternate implementation. Define a public interface to an algorithm in an abstract base class. In that public method, implement the steps of the algorithm in calls to protected abstract methods of the class. In concrete classes derived from the base class, override each step of the algorithm with a concrete implementation specific to that class.

1.6.5 Pattern: Builder

1.6.5.1 Definition

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

A Builder seems similar in concept to the Abstract Factory. The difference as I see it is the Builder refers to single complex objects of different concrete classes but containing multiple parts, whereas the abstract factory lets you create whole families of concrete classes. For example, a builder might construct a house, cottage or office. You might employ a different builder for a brick house or a timber house, though you would give them both similar instructions about the size and shape of the house. On the other hand the factory generates parts and not the whole. It might produce a range of windows for buildings, or it might produce a quite different range of windows for cars.

1.6.5.2 Applications in Delphi

The functionality used in Delphi's VCL to create forms and components is similar in concept to the builder. Delphi creates forms using a common interface, through Application.CreateForm and through the TForm class constructor. TForm implements a

common constructor using the resource information (DFM file) to instantiate the components owned by the form. Many descendant classes reuse this same construction process to create different representations. Delphi also makes developer extensions easy. TForm's OnCreate event also adds a hook into the builder process to make the functionality easy to extend.

1.6.5.3 Implementation Example

The following example includes a class TAbstractFormBuilder and two concrete classes TRedFormBuilder and TBlueFormBuilder. For ease of development some common functionality of the concrete classes has been moved into the shared TAbstractFormBuilder class.

1.6.6 Pattern: Abstract Factory

1.6.6.1 Definition

"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."

The Factory Method pattern below is commonly used in this pattern.

1.6.6.2 Applications in Delphi

This pattern is ideal where you want to isolate your application from the implementation of the concrete classes. For example if you wanted to overlay Delphi's VCL with a common VCL layer for both 16 and 32 bit applications, you might start with the abstract factory as a base.

1.6.6.3 Implementation Example

The following example uses an abstract factory and two concrete factory classes to implement different styles of user interface components. TOAbstractFactory is a singleton class, since we usually want one factory to be used for the whole application.

At runtime, our client application instantiates the abstract factory with a concrete class and then uses the abstract interface. Parts of the client application that use the factory don't need to know which concrete class is actually in use.

1.6.7 Pattern: Factory Method

1.6.7.1 Definition

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

The Abstact Factory pattern can be viewed as a collection of Factory Methods.

1.6.7.2 Applications in Delphi

This pattern is useful when you want to encapsulate the construction of a class and isolate knowledge of the concrete class from the client application through an abstract interface.

One example of this might arise if you had an object oriented business application potentially interfacing to multiple target DBMS. The client application only wants to know about the business classes, not about their implementation-specific storage and retrieval.

1.6.7.3 Implementation Example

In the Abstract Factory example, each of the virtual widget constructor functions is a Factory Method. In their implementation we define a specific widget class to return.

1.7 KEY ELEMENTS OF DELPHI CLASS DEFINITIONS

1.7.1 Unit Structure

Delphi units (.PAS files) allow declaration of interface and implementation sections. The interface defines the part that is visible to other units using that unit. The keyword *uses* can be added to a unit's interface or implementation section to list the other units that your unit uses. This indicates to the compiler that your unit refers to parts of the used unit's interface. Parts of a unit declared in the implementation section are all private to that unit, i.e. never visible to any other unit. Types, functions and procedures declared in the interface of a unit must have a corresponding implementation, or be declared as external (e.g. a call to a function in a DLL).

1.7.2 Class Interfaces

Classes are defined as types in Delphi and may contain fields of standard data types or other objects, methods declared as functions or procedures, and properties. The type declaration of a class defines its interface and the scope of access to fields, methods and properties of the class. Class interfaces are usually defined in the interface of a unit to make them accessible to other modules using that unit. However they don't need to be. Sometimes a type declaration of a class may be used only within the implementation part of a unit.

1.7.3 Properties

Properties are a specialised interface to a field of a defined type, allowing access control through read and write methods. Properties are not virtual, you can replace a property with another property of the same name, but the parent class doesn't know about the new property. It is however possible to make the access methods of a property virtual.

1.7.4 Inheritance

Delphi's inheritance model is based on a single hierarchy. Every class inherits from TObject and can have only one parent.

A descendant class inherits all of the interface and functionality of its parent class, subject to the scope described below.

Multiple inheritance from more than one parent is not allowed directly. It can be implemented by using a container class to create instances one or more other classes and selectively expose parts of the contained classes.

Private, Protected, Public and Published ScopeScope refers to the visibility of methods and data defined in the interface of a class, i.e. what parts of the class are accessible to the rest of the application or to descendant classes.

The default scope is public, for instance the component instances you add to a form at design time. Public says "come and get me"; it makes the data or method visible to everything at runtime.

Published parts of a class are a specialized form of Public scope. They indicate special behaviour for classes derived from TPersistent. A persistent class can save and restore its published properties to persistent storage using Delphi's standard streaming methods. Published properties also interact with Delphi Object Inspector in the IDE. A class must descend from TPersistent in order to use Published. There's also not much point in publishing methods, since you can't store them, although Delphi's compiler doesn't stop you. Published also lets another application access details of the class through Delphi's runtime type information. This would be rarely used, except in Delphi's design time interaction with its VCL.

Encapsulation or information hiding is essential to object orientation, so Protected and Private scope let you narrow the access to parts of a class.

Protected parts are visible only to descendant classes, or to other classes defined in the same unit.

Private parts are visible only to the defining class, or to other classes defined in the same unit.

It's important to note that once something is given public or published scope, it cannot be hidden in descendant classes.

Static, Virtual and Dynamic Methods; Override and Inherited

Methods declared as virtual or dynamic let you change their behaviour using override in a descendant class. You're unlikely to see a virtual method in the private part of a class, since it could only be overridden in the same unit, although Delphi's compiler doesn't stop you from doing this.

Override indicates that your new method replaces the method of the same name from the parent class. The override must be declared with the same name and parameters as the original method.

When a method is overridden, a call to the parent class's method actually executes the override method in the real class of the object.

Static methods on the other hand have no virtual or override declaration. You can replace a method of a class in a descendant class by redeclaring another method, however this is not object oriented. If you reference your descendant class as the parent type and try to call the replaced method, the static method of the parent class is executed. So in most cases, it's a bad idea to replace a static method.

Virtual and dynamic methods can be used interchangeably. They differ only in their treatment by the compiler and runtime library. Delphi's help explains that dynamic methods have their implementation resolved at compile time and run slightly faster, whereas virtual methods are resolved at runtime, resulting in slightly slower access but a smaller compiled program. Virtual is usually the preferred declaration. Delphi's help suggests using dynamic when you have a base class with many descendants that may not override the method.

The inherited directive lets you refer back to a property or method as it was declared in the parent class. This is most often used in the implementation of an override method, to call the inherited method of the parent class and then supplement its behaviour.

1.7.5 Abstract Methods

Abstract is used in base classes to declare a method in the interface and defer its implementation to a descendant class. I.e. it defines an interface, but not the underlying

operation. Abstract must be used with the virtual or dynamic directive. Abstract methods are never implemented in the base class and must be implemented in descendant classes to be used. A runtime error occurs if you try to execute an abstract method that is not overridden. Calling inherited within the override implementation of an abstract method will also result in a runtime error, since there is no inherited behaviour.

1.7.6 Messages

Delphi's handling of Windows messages is a special case of virtual methods. Message handlers are implemented in classes that descend from TControl. I.e classes that have a handle and can receive messages. Message handlers are always virtual and can be declared in the private part of a class interface, yet still allow the inherited method to be called. Inherited in a message handler just uses the keyword inherited, there is no need to supply the name of the method to call.

1.7.7 Events

Events are also an important characteristic of Delphi, since they let you delegate extensible behaviour to instances of a class. Events are properties that refer to a method of another object. Events are not inherited in Delphi 1; Delphi 2 extends this behaviour to let you use inherited in an event. . Inherited in an event handler just uses the keyword inherited, there is no need to supply the name of the method to call.

Events are particularly important to component developers, since they provide a hook for the user of the component to modify its behaviour in a way that may not be foreseen at the time the component is written.

1.7.8 Constructors and Destructors

The constructor and destructor are two special types of methods. The constructor initializes a class instance (allocates memory initialized to 0) and returns a reference (pointer) to the object. The destructor deallocates memory used by the object (but not the memory of other objects created by the object).

Classes descended from TObject have a static constructor, Create, and a virtual destructor Destroy.

TComponent introduces a new public property, the Owner of the component and this must be initialized in the constructor. TComponent's constructor is declared virtual, i.e. it can be overridden in descendant classes.It is essential when you override a virtual constructor or destructor in a TComponent descendant to include a call to the inherited method.

1.8 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.

1.8.1 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.

1.8.2 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 a 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.

1.8.3 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.

1.8.4 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.

1.8.4.1 Standard Components

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox and Tedit, 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 Window's 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.

1.8.4.2 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.

1.8.4.3 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 can't 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.

1.8.4.4 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.

1.8.4.5 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.

1.8.4.6 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.

1.9 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.

1.9.1 Property-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 that 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.

1.9.2 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.

Property type	Object Inspector treatment
Simple	Numeric, character, and string properties appear in the Object Inspector as numbers, characters, and strings, respectively. The user can type and edit the value of the property directly.
Enumerated	Properties of enumerated types (including Boolean) display the value as defined in the source code. The user can cycle through the possible values by double-clicking the value column. There is also a drop-down list that shows all possible values of the enumerated type.
Set	Properties of set types appear in the Object Inspector looking like a set. By expanding the set, the user can treat each element of the set as a Boolean value: True if the element is included in the set or False if it's not included.
Object	Properties that are themselves objects often have their own property editors. However, if the object that is a property also has published properties, the Object Inspector allows the user to expand the list of object properties and edit them individually. Object properties must descend from TPersistent.
Array	Array properties must have their own property editors. The Object Inspector has no built-in support for editing array properties.

For more information on properties, refer to the "Component Writers Guide" which ships with Delphi.

1.9.3 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.

1.9.4 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.

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.

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;
```

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.

1.9.5 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.

1.9.6 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 code 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.

1.9.7 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 it's 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. This task is done automatically by Delphi's design-time environment when you drop a component from the Component Palette onto your form.

CHAPTER 2

DATABASE

Every thing around us has a particular identity. To identify anything system, actor or person in words we need a data or information. So this information is valuable and in this advanced era we can store it in database and access this data by the blink of eye.

For an instant if we go through the definitions of database we may find following definitions.

A database is a collection of related information.

A database is an organized body of related information.

2.1 DEMERITS OF ABSENCE OF DATABASE

A glance on the past will may help us to reveal the drawbacks in case of absence of database.

In the past when there wasn't proper system of database, Much paper work was need to do and to handle great deal of written paper documentation was giant among the problems itself.

In the huge networks to deal with equally bulky data, more workers are needed which affidavit cost much labor expanses.

The old criteria for saving data and making identification was much time consuming such as if we want to search the particular data of a person.

Before the Development of Computer database it was a great problem to search for some thing. Efforts to avoid the headache of search often results in new establishments of data.

Before the development of database it seemed very unsafe to keep the worthy information. In Some situation some big organization had to employee the special persons in order to secure the data.

Before the implementation of database any firm had to face the plenty of difficulties in order to maintain their Management. To hold the check on the expenses of the firm, the manager faced difficulties.

2.2 MERITS OF DATABASE

The modern era is known as the golden age computer sciences and technology. In a simple phrase we can express that the modern age is built on the foundation of database.

If we carefully watch our daily life we can examine that some how our daily life is being connected with database.

There are several benefits of database developments.

Now with the help of computerized database we can access data in a second.

By the development of the database we can make data more secure.

By the development of database we can reduce the cost.

2.3 DATABASE DESIGN

The design of a database has to do with the way data is stored and how that data is related. The design process is performed after you determine exactly what information needs to be stored and how it is to be retrieved.

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

Computerized library systems

Automated teller machines

Flight reservation systems

Computerized parts inventory systems

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query.

Database design is a complex subject. A properly designed database is a model of a business, Country Database or some other in the real world. Like their physical model counterparts, data models enable you to get answers about the facts that make up the objects being modeled. It's the questions that need answers that determine which facts need to be stored in the data model.

In the relational model, data is organized in tables that have the following characteristics: every record has the same number of facts, every field contains the same type of facts (Data) in each record, and there is only one entry for each fact. No two records are exactly the same.

The more carefully you design, the better the physical database meets users' needs. In the process of designing a complete system, you must consider user needs from a variety of viewpoints.

2.4 DATABASE MODELS

Various techniques are used to model data structures. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementation may be possible. An example of this is the relational model: in larger systems the physical implementation often has indexes which point to the data; this is similar to some aspects of common implementations of the network model. But in small relational database the data is often stored in a set of

files, one per table, in a flat, un-indexed structure. There is some confusion below and elsewhere in this article as to logical data model vs. its physical implementation.

2.4.1 Flat Model

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 might be used as a part of a system security database. Each row would have the specific password associated with a specific 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 the basis of the spreadsheet.

2.4.2 Network Model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.

2.4.3 Relational Model

The relational data model was introduced in an academic paper by E.F. Cod 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.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allow tables to be defined that allow duplicate rows an extension (or violation) of the relational model. In common English usage, a DBMS is

called relational if it supports relational operational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, not-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't 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 row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no nature key is suitable, an arbitrary key can be assigned (such as by given 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 can't 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).

2.4.3.1 Why we use a Relational Database Design

Maintaining a simple, so-called flat database consisting of a single table doesn't require much knowledge of database theory. On the other hand, most database worth maintaining are quite a bit more complicated than that. Real life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full-fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million

books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database.

2.5 RELATIONSHIPS BETWEEN TABLES

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many and many-to-many relationships.

2.5.2 One-To-One Relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their tracks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support in a table requires security, placing them in a separate table lets your application restrict to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to these fields.

2.5.3 One-To-Many Relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a Many-To-Many relationship as well.

2.6 DATA MODELING

In information system design, data modeling is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "Data Analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together), than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

2.6.1 Database Normalization

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMS lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case de-normalizations are sometimes used to improve performance, at the cost of reduced consistency.

2.6.2 Primary Key

In database design, a primary key is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions) and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a primary key is a candidate key chosen as the main method of uniquely identifying a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System Numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others. In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

2.6.3 Foreign Key

A foreign key (FK) is a field in a database record under one primary key that points to a key field of another database record in another table where the foreign key of one table refers to the primary key of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail needs not to include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book, or even the book itself. The ISBN is the primary key of the book, and it is used as a foreign key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

2.6.4 Compound Key

In database design, a compound key (also called a composite key) is a key that consists on 2 or more attributes.

No restriction is applied to the attribute regarding their (initial) ownership within the data model. This means that any one, none or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute value.

CHAPTER 3
MYSQL

3.1 INTRODUCTION TO MYSQL

This chapter provides a tutorial introduction to MySQL by showing how to use the mysql client program to create and use a simple database. mysql (sometimes referred to as the ``terminal monitor" or just ``monitor") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you place your queries in a file beforehand, then tell mysql to execute the contents of the file. Both ways of using mysql are covered here.

To see a list of options provided by mysql, invoke it with the --help option:

shell> mysql --help

This chapter assumes that mysql is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If you are the administrator, you will need to consult other sections of this manual.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an already-existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily left out. Consult the relevant sections of the manual for more information on the topics covered here.

3.2 WHAT IS MYSQL?

3.2.1 Definition

MySQL is an open source software relational database management system (RDBMS) which

uses a SQL (Structured Query Language)

SQL is the standard language used for interacting with databases.

3.3 WHY CHOOSE MYSQL?

There are many relational databases available to use, so why choose MySQL?

We are specifically interested in databases which PHP supports; these include Oracle, IBM's DB2 and Microsoft's SQL Server (all of which cost money).

The two main open source (free) alternatives to these are PostgreSQL and MySQL. PostgreSQL is arguably the better of the two, but MySQL is better supported on Windows, and is a popular choice among Web hosts that provide support for PHP.

Here are some of MySQL's advantages

- It's fast
- It's free to use, and commercial licenses are reasonable
- It's easy to use
- It is cross platform
- There is a wide community of technical support
- It's secure
- It supports large databases
- It is designed specifically for web base applications and hence works very well partnered with PHP

3.4 PREPARING THE WINDOWS MYSQL ENVIRONMENT

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL- Max server binaries. Here is a list of the different MySQL servers you can use:

mysqld	Compiled with full debugging and automatic memory allocation checking, symbolic links, InnoDB and DBD tables.
mysql-opt	Optimized binary with no support for transactional tables.
mysqld-nt	Optimized binary for NT with support for named pipes. You can run this version on Win98, but in this case no named pipes are created and you must have TCP/IP installed.
mysqld-max	Optimized binary with support for symbolic links, InnoDB and DBD tables.
mysqld-max-nt	Like mysqld-max, but compiled with support for named pipes.

All of the above binaries are optimized for the Pentium Pro processor but should work on any Intel processor >=i386

In the following circumstance, you will need to use the MySQL configuration file:

- The install/data directories are different than the default 'c:\mysql' and 'c:\mysql\data'.
- If you want to use one of these servers:
  - mysqld.exe
  - mysqld-max.exe
  - mysqld-max-nt.exe
- If you need to tune the server settings.

There are two configuration files with the same function: 'my.cnf' and 'my.ini' file, however, only one of these can/should be used. Both files are plain text. The 'my.cnf' file should be created in the root directory of drive C and the 'my.ini' file in the WinDir directory e.g.: C:\WINDOWS or C:\WINNT. If your PC uses a boot loader where the C drive isn't the boot drive, then your only option is to use the 'my.ini' file. Also note that if you use the WinMySQLAdmin tool, only the

'my.ini' file is used. The '\mysql\bin' directory contains a help file with instructions for using this tool.

Using Notepad, create the configuration file and edit the base section and keys:

[mysqld]

basedir = the_install_path # e.g. 'c:\mysql'

datadir = the_data_path # e.g. 'c:\mysql\data' or 'd:\mydata\data'

If the data directory is other than the default 'c:\mysql\data', you must cut the whole

'\data\mysql' directory and paste it on the your option new directory, e.g. 'd:\mydata\mysql'.

If you want to use the InnoDB transaction tables, you need to manually create two new directories to hold the InnoDB data and log files, e.g. 'c:\ibdata' and 'c:\iblogs'. You will also need to create some extra lines to the configuration file.

If you don't want to use, add the skip-innodb option to the configuration file.

Now you are ready to test starting the server.

3.5 STARTING THE SERVER FOR THE FIRST TIME

Testing from a DOS command prompt is the best thing to do because the server prints messages, so if something is wrong with your configuration, you will see a more accurate error message which will make it easier to identify and fix any problems.

Make sure you're in the right directory *(C:\>cd \mysql\bin),*

 # To install mysqld as a standalone program, enter:

C:\mysql\bin> mysqld-max --standalone

You should see the below print messages:

```
InnoDB: The first specified data file c:\ibdata\ibdata1 did not exist
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 200715200
InnoDB: Database physically writes the file full: wait
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

 # To install mysql as a service (Windows 2000), enter:

C:\mysql\bin> mysqld-nt --install

Now you can start and stop mysqld as follows:

C:\>NET START MySQL C:\>NET STOP MySQL

C:\>NET START MySQL

 # To start the MySQL Monitor, enter:

The MySql service is starting.

The MySQL service was started successfully.

C:\>cd \mysql

C:\mysql>bin\mysql

Welcome to the MySQL Monitor. Commands end with ; or \g. Your MySQL connection id

is 1 to server version 3.23.49-nt Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> (enter a command or enter 'QUIT' to quit)

mysql> QUIT Bye

C:\mysql>NET STOP MySQL The MySQL service is stopping.

The MySQL service was stopped successfully.

C:\mysql>

3.6 CONNECTING TO AND DISCONNECTING FROM THE SERVER

To connect to the server, you'll usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

shell> mysql -h host -u user -p

Enter password: ********

The ******** represents your password; enter it when mysql displays the Enter password:

prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

shell> mysql -h host -u user -p

Enter password: ********

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>

The prompt tells you that mysql is ready for you to enter commands.

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

shell> mysql

After you have connected successfully, you can disconnect any time by typing QUIT at the *mysql>*

prompt: mysql> QUIT Bye

You can also disconnect by pressing Control-D.

Most examples in the following sections assume you are connected to the server. They indicate this by the mysql> prompt.

3.7 ENTERING QUERIES

Make sure you are connected to the server, as discussed in the previous section. Doing so will not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how mysql works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown below following the mysql> prompt and hit the RETURN key:

mysql> SELECT VERSION( ), CURRENT_DATE;

version()	CURRENT_DATE
3.22.20a-log	1999-03-19

row in set (0.01 sec)

mysql>

This query illustrates several things about mysql:

A command normally consists of a SQL statement followed by a semicolon. (There are some exceptions where a semicolon is not needed. QUIT, mentioned earlier, is one of them. We'll get to others later.)

When you issue a command, mysql sends it to the server for execution and displays the results, then prints another mysql> to indicate that it is ready for another command.

Mysql displays query output as a table (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), mysql labels the column using the expression itself.

Mysql shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the ``rows in set'' line is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

mysql> SELECT VERSION(), CURRENT_DATE; mysql> select version(), current_date;
mysql> SELECT VERSION(), current_DATE;
*mysql> SELECT SIN(PI()/4), (4+1)*5;*

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

mysql> SELECT VERSION(); SELECT NOW();

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. mysql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, mysql accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

mysql> SELECT USER(),CURRENT_DATE;

USER()	CURRENT_DATE
joesmith@localhost	1999-03-18

In this example, notice how the prompt changes from mysql> to -> after you enter the first line of a multiple-line query. This is how mysql indicates that it hasn't seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you will always be aware of what mysql is waiting for.

If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing \c:

mysql> SELECT USER() \c mysql>

Here, too, notice the prompt. It switches back to mysql> after you type \c, providing feedback to indicate that mysql is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that mysql is in:

Prompt	Meaning
mysql>	Ready for new command.
->	Waiting for next line of multiple-line command.
'>	Waiting for next line, collecting a string that begins with a single quote (`').
">	Waiting for next line, collecting a string that begins with a double quote (`"').

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, mysql waits for more input:

mysql> SELECT USER()
->

If this happens to you (you think you've entered a statement but the only response is a -> prompt), most likely mysql is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and mysql will execute it:

mysql> SELECT USER()
-> ;
USER()
joesmith@localhost

The '> and "> prompts occur during string collection. In MySQL, you can write strings surrounded by either `'` or `"` characters (for example, 'hello' or "goodbye"), and mysql lets you enter strings that span multiple lines. When you see a '> or "> prompt, it means that you've entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. That's fine if you really are entering a multiple-line string, but how likely is that? Not very. More often, the '> and "> prompts indicate that you've inadvertantly left out a quote character. For example:

*mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;*
">

If you enter this SELECT statement, then hit RETURN and wait for the result, nothing will happen. Instead of wondering why this query takes so long, notice the clue provided by the "> prompt. It tells you that mysql expects to see the rest of an unterminated string. (Do you see the error in the statement? The string "Smith is missing the second quote.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type \c in this case, because mysql interprets it as part of the string that it is collecting! Instead, enter the closing quote character (so mysql knows you've finished the string), then type

*\c:mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;*
"> "\c mysql>

The prompt changes back to mysql>, indicating that mysql is ready for a new command.

It's important to know what the '> and "> prompts signify, because if you mistakenly enter an unterminated string, any further lines you type will appear to be ignored by mysql -- including a line containing QUIT! This can be quite confusing, especially if you don't know that you need to supply the terminating quote before you can cancel the current command.

CHAPTER 4

USER MANUAL

In this chapter I will try to explain the veterinerian application program that when it run.If a someone run the program; firstly splash form will be shown for 3000ms like below.

Figure 4.1

After 3000ms entry page (Secure Page) will be shown. (Figure 4.2)

Figure 4.2

On this page (Figure 4.2) the user must enter the user name and password.If user name and password found then the program check the user state for still working or has left.If still working; now the program check the user position for Admin,Veterinerian,Manager,User and Temporary.If the user has left then who can not enter the system; in the same time there is no user name and password program gives three trying chance to enter the system; when is thirth the program will be terminate.

If user is Admin then who can access everything to make on application program; If user is manager then who access everything to make exclusive of wrong password application; If user is veterinerian then who can not access process of user after that who can access; If user is a normal user then who can see some knowledge and can change the program settings; If the user is temporary then who can access only amusements, internet explorer, find folder and drug knowledge.

Then main page comes (Figure 4.3) it is shown below

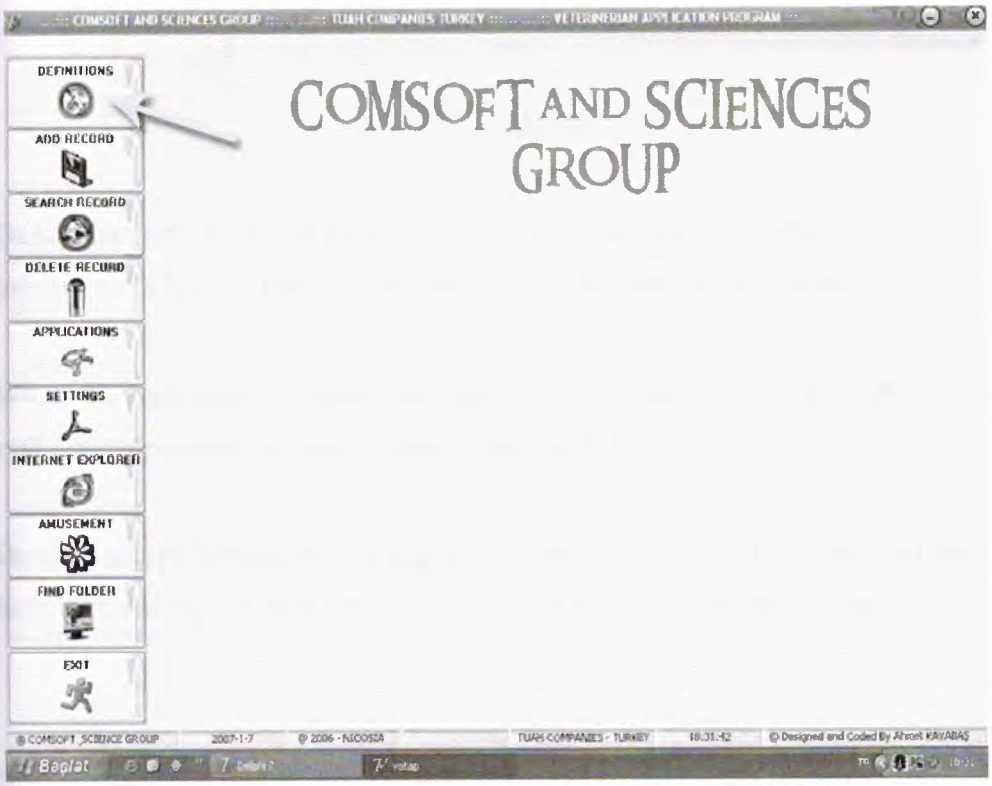

Figure 4.3

This is main page; other pages shown on it. There are ten button on this. User click one of them and access the page that wanted by the user.

When button of definition clicked definition selection page is shown like below figure

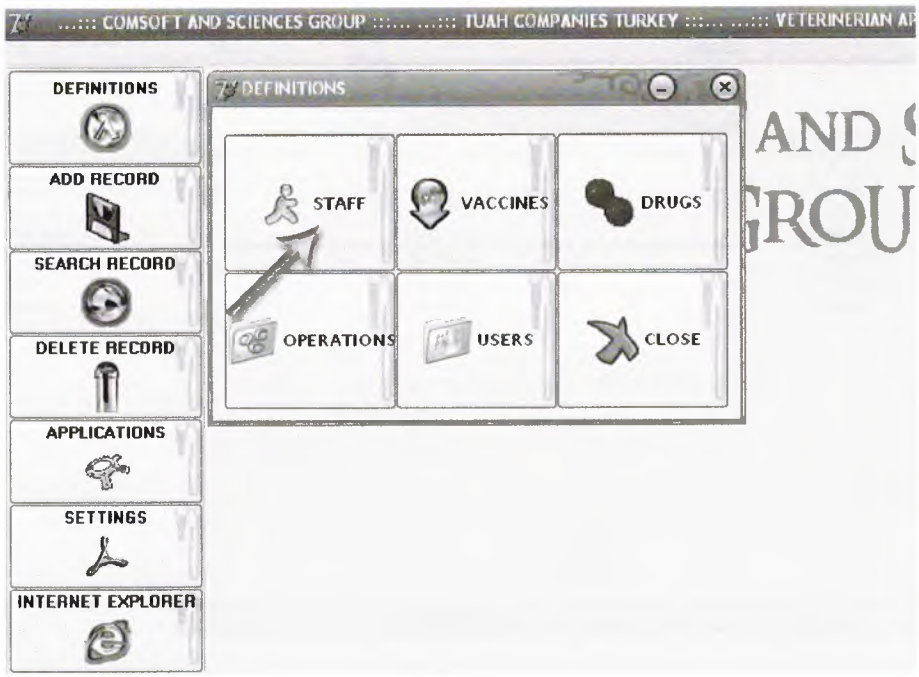

Figure 4.4

Definitions acts to create knowledge that is necessary for application process.User decide process and click the button to access the page for needed application.

When the staff button clicked; the page is shown that is figure 4.5. On this page user can make some process like save, update, delete and find.

On staff record form there is a magnifier button that acts that if there is a person who saved before;knowledge of that person is shown on form with all knowledge.

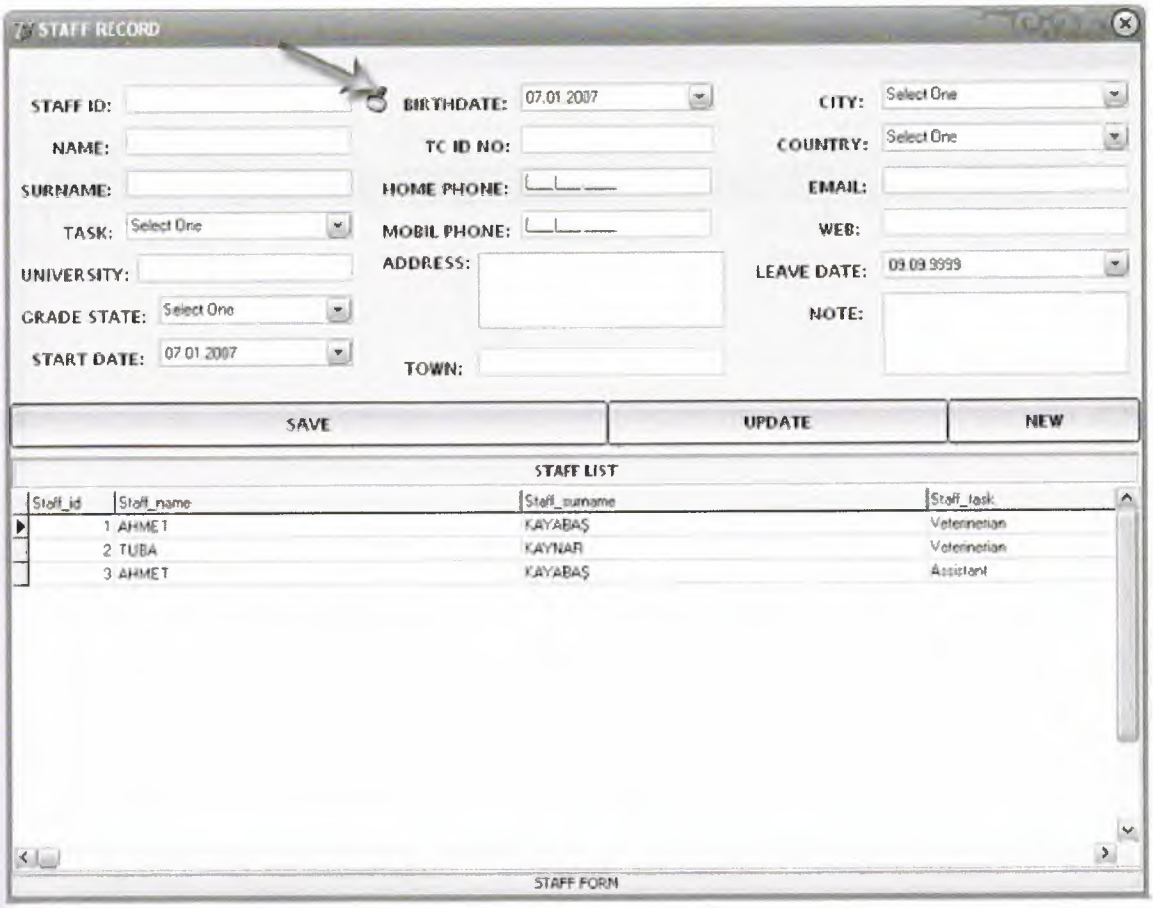

Figure 4.5

When the magnifier button clicked figure 4.6 is shown

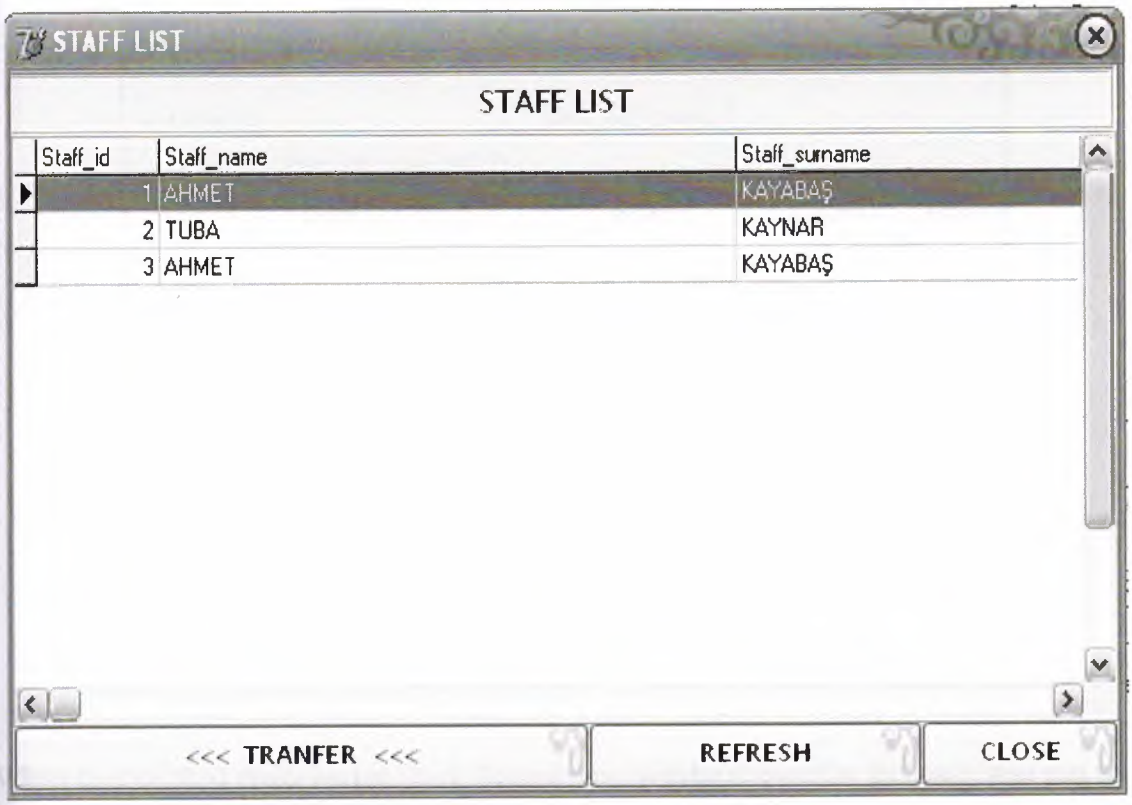

Figure 4.6

When the vaccines record clicked; the page is shown that is figure 4.5. On this page user can add new vaccine, can delete or update it.For process of vaccinates vaccine name called from here (Figure 4.7)

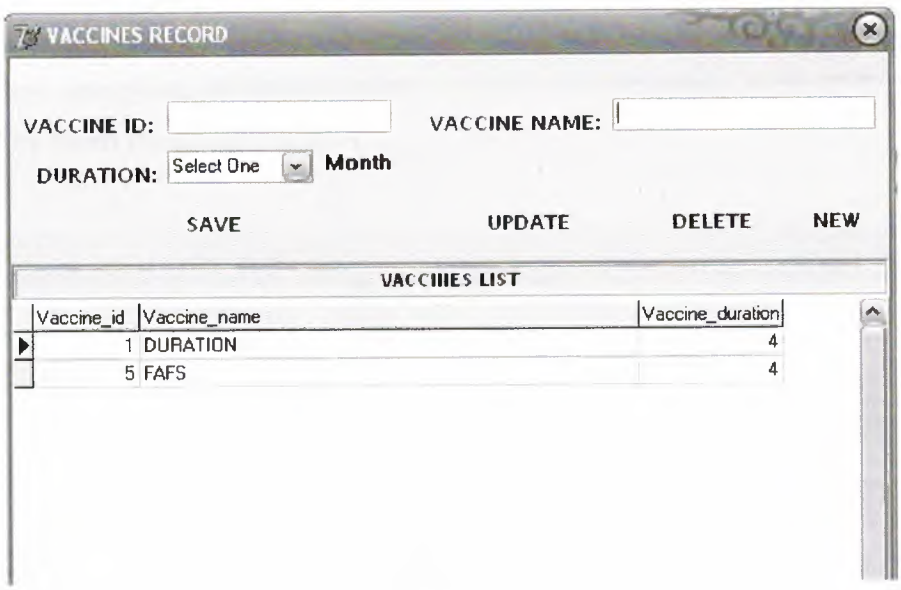

Figure 4.7

When user clicked drugs button Drug Record page will be shown.On this page user can add new drug, delete drug and update old drug record.For process of drug application drug name will absorb from here Figure 4.8

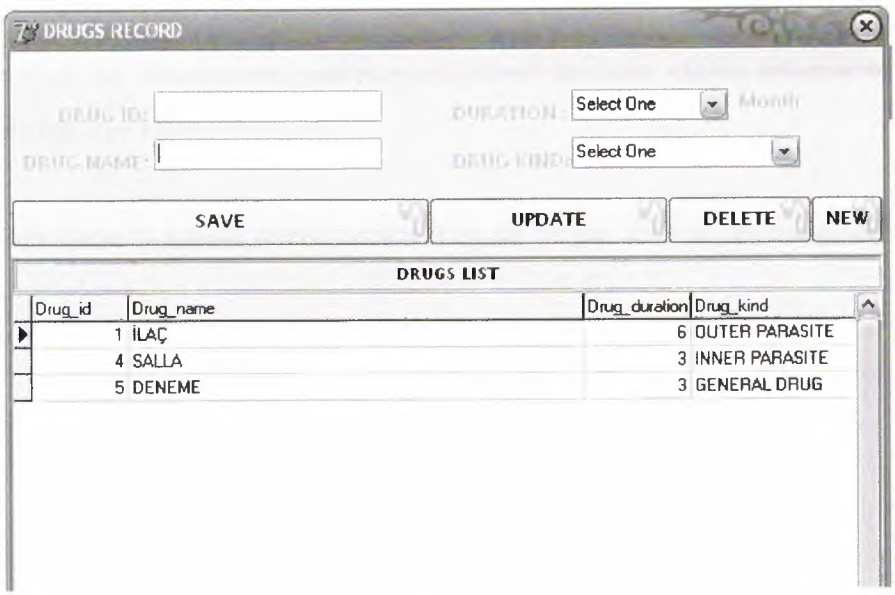

Figure 4.8

When operations button clicked Operation Record page will be shown.On this page user can add new operation, deleteand update operation.For operation application operation name will be taken from here Figure 4.9

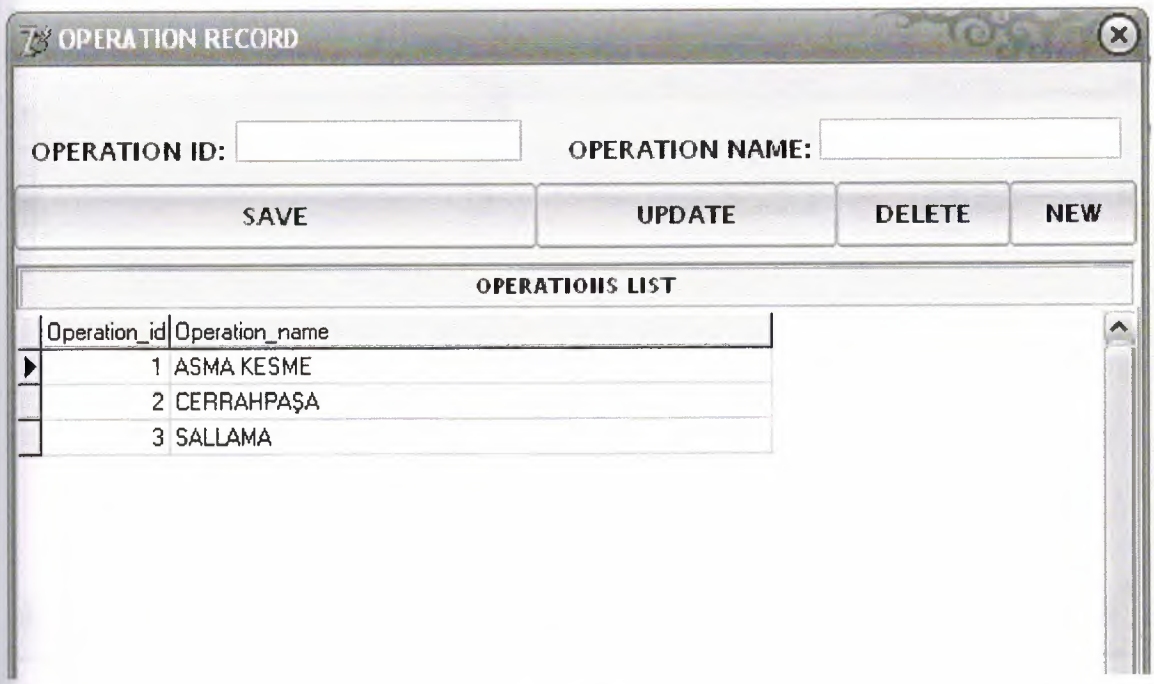

Figure 4.9

If the user click the user button; user page is opened to make adding,deleting and updating user knowledge like Figure 4.10.

On this form there is a mini arrow button.It is act to get staff to combine with users and staff.Because after when a knowledge is needed it stafisfied directly.

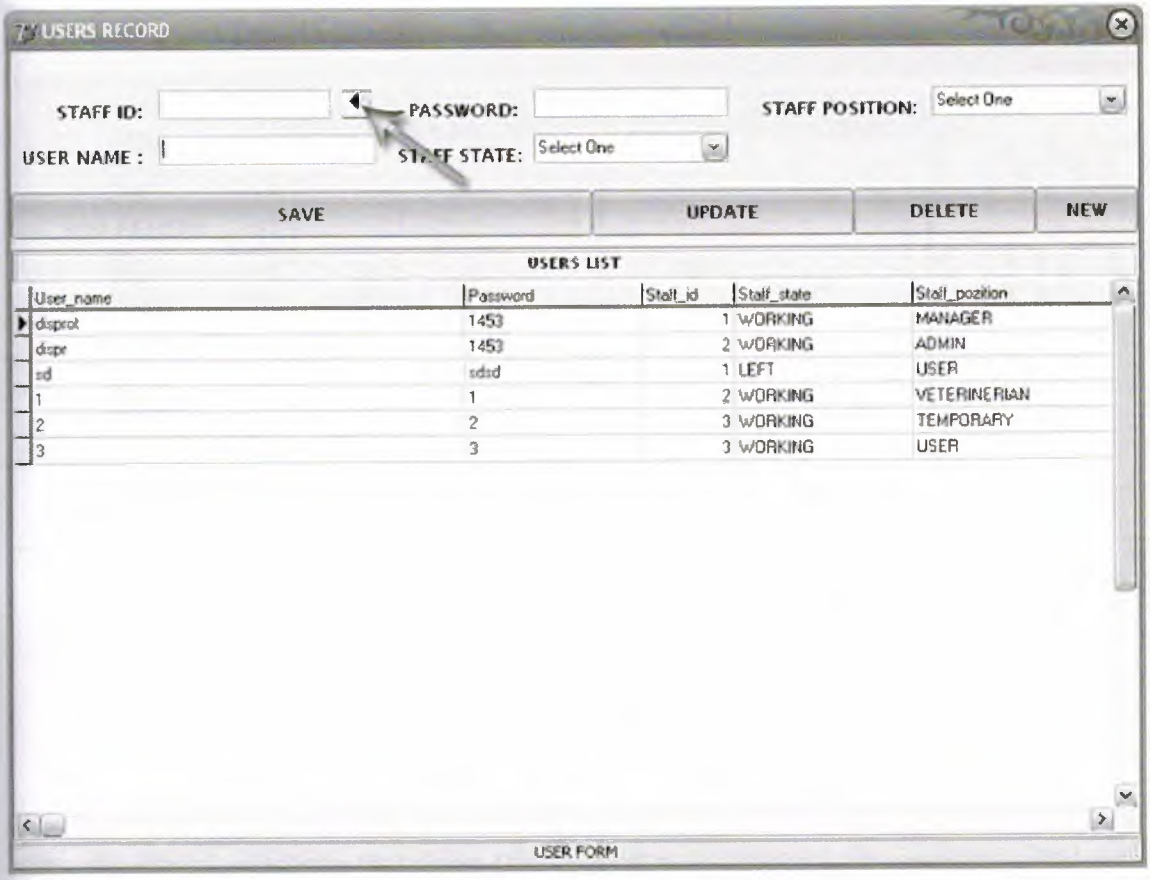

Figure 4.10

ADD Record button thet on main form acts to create knowledge that is necessary for continuity of program.Because Customer and Animal is defined here.User decide process and click the button to access the page for needed application.Figure 4.11 act transaction of this process from main menu.

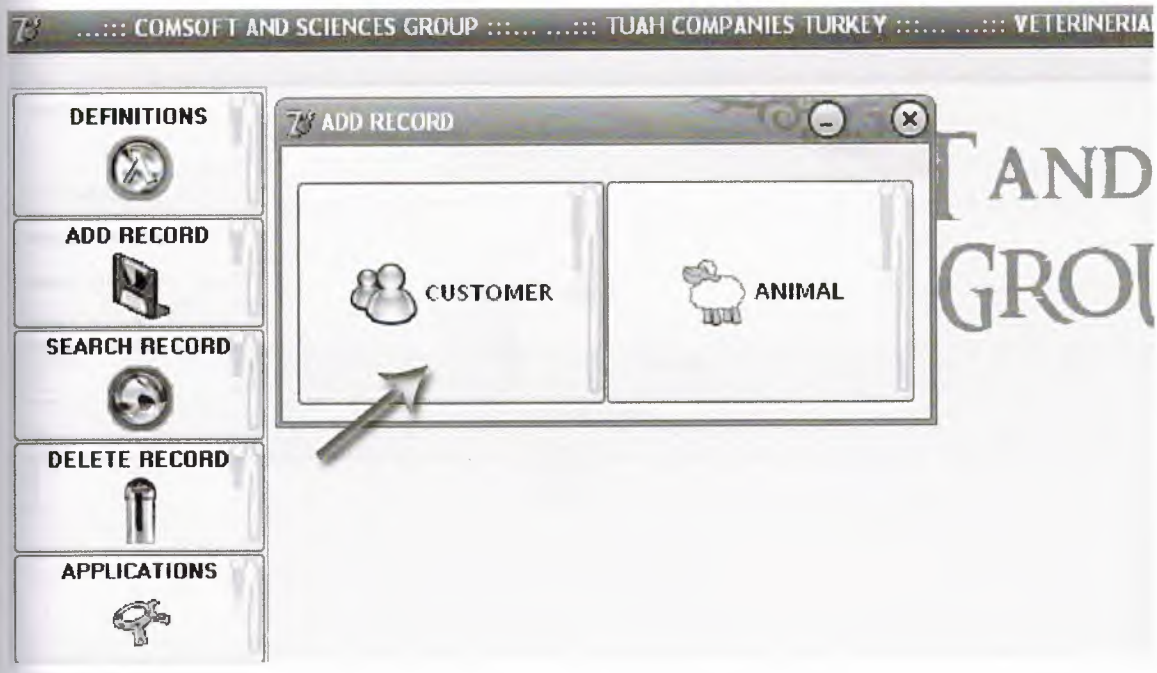

Figure 4.11

User can decide customer or animal.who if decide to continue for customer must click customer button.When he/she made this a new form is shown,Customer record form.With this form user can add an new customer or delete or update an old customer.Update or delete is needed.Well may be customer transferred to other city or transferred to other veterinerian.Figure 4.12 include a customer record page figure

The Program acts all of them easly.Interface is basic as shown.Every user can adapt easly to make operation.

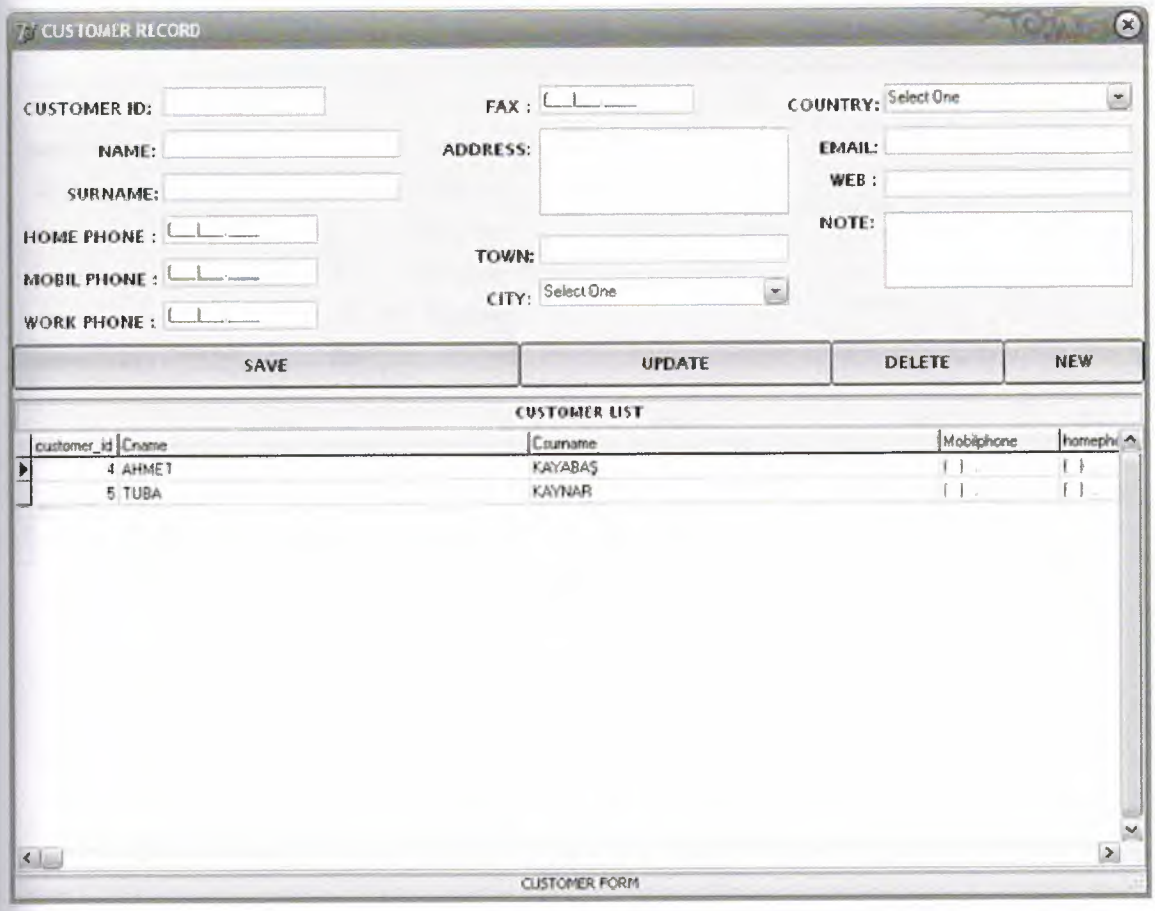

Figure 4.12

If user decided for animal must click animal button on Add Record Form (Figure 4.11). When he/she clicked animal button Animal Record Form will be displayed.With this form user can add an new animal or delete or update an old animal with their owner.Update or delete is needed.Well may be animal transferred to other veterinerian or may be died.

Figure 4.13 shows animal record form.

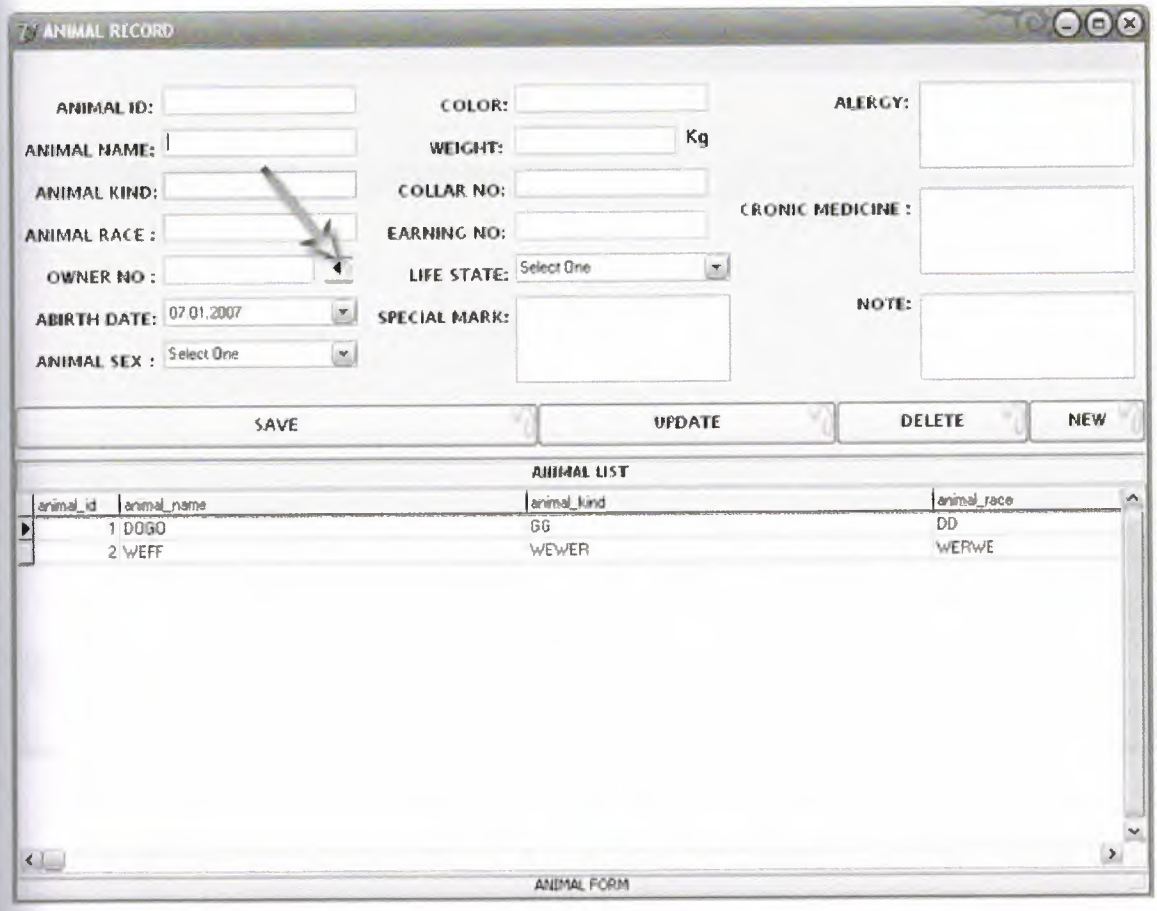

Figure 4.13

On this form (Figure 4.13) there is a mini arrow button.It find owner.Thats why initially customer must save then animal can save.Because as seen owner only called from other form directly.(Figure 4.14)

This Page (Figure 4.14) absorb the knowledge directly database through queries.When it opened datas comes onto dbgrid that on page.

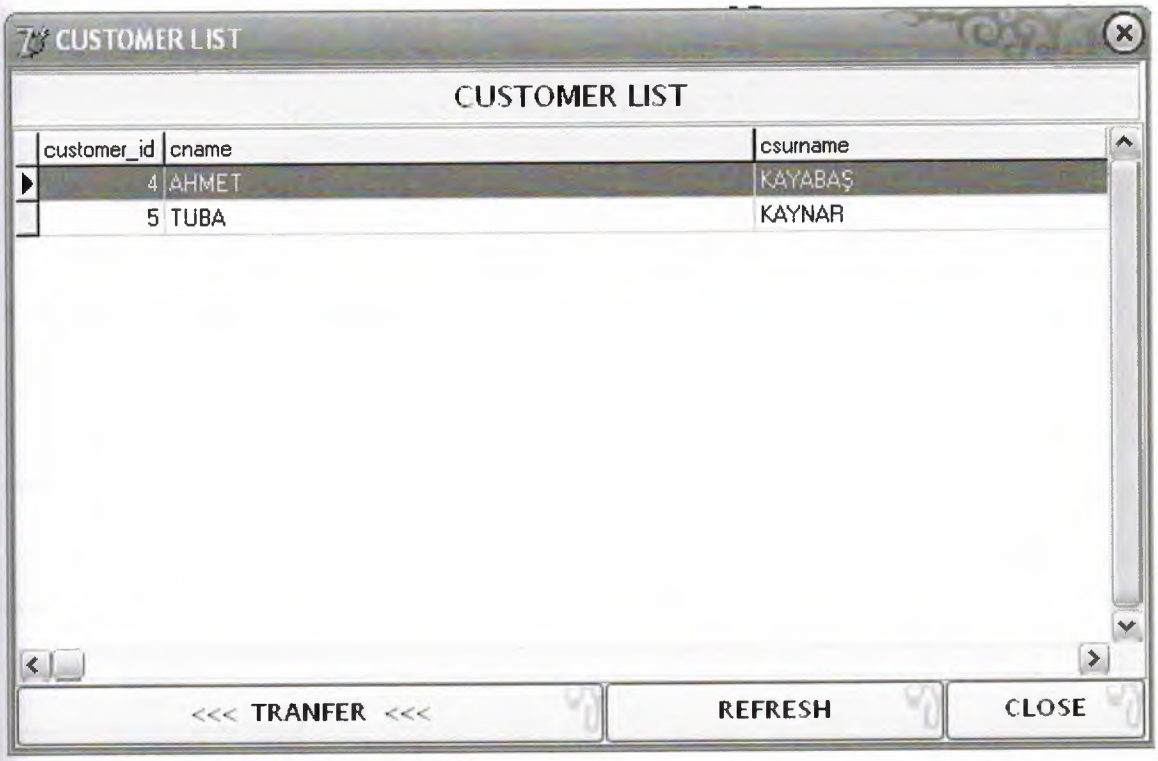

Figure 4.14

Search Record button that on main form acts to show knowledge.The knowledge stored in database.User can access data through this pages (Search pages).When Search Button clicked on main menu a new page will appear (Figure 4.15)

On this form (Figure 4.15) there are all states, applications.Well users can see, collect the datas easly.They must decide Only 'What do I need' then click button and access knowledge that needed by your own.

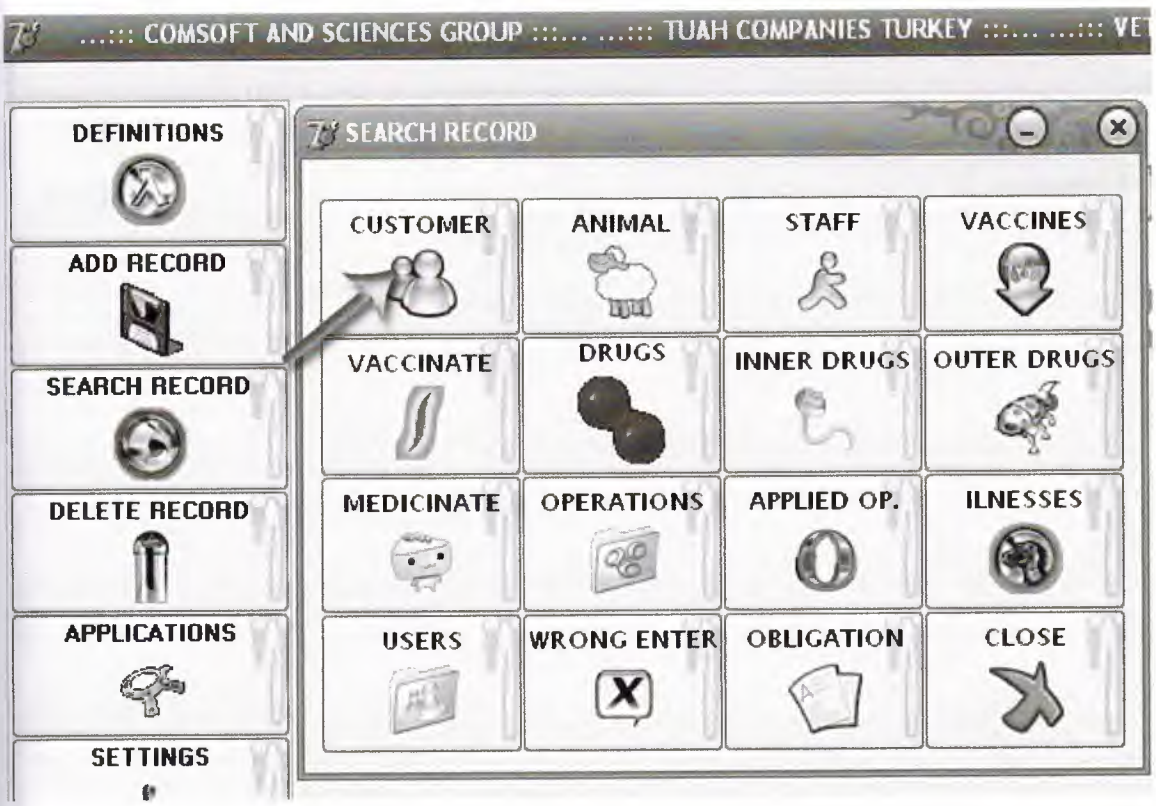

Figure 4.15

If user want to see customer knowledge, he/she must click customer button.Than customer search form will be displayed.Well easly got the data.Figure 4.16 has a customer search page image.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.16 has only one of them.All figure will append end of project as appendix.

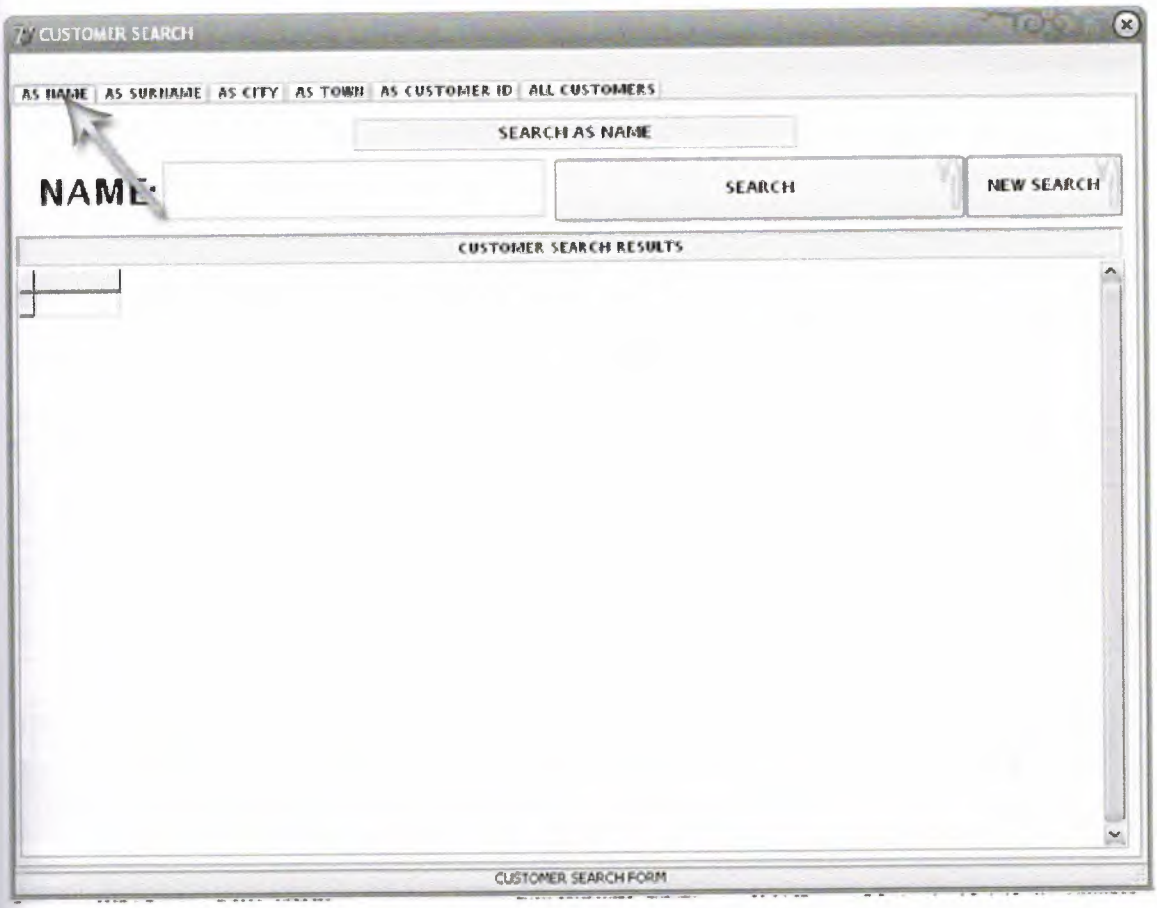

Figure 4.16

If user want to see animal knowledge, he/she must click animal button.Than animal search form will be displayed.Figure 4.17 shows an animal search page.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.17 has only one of them.

When user write character from keyboard the program will check the animals.

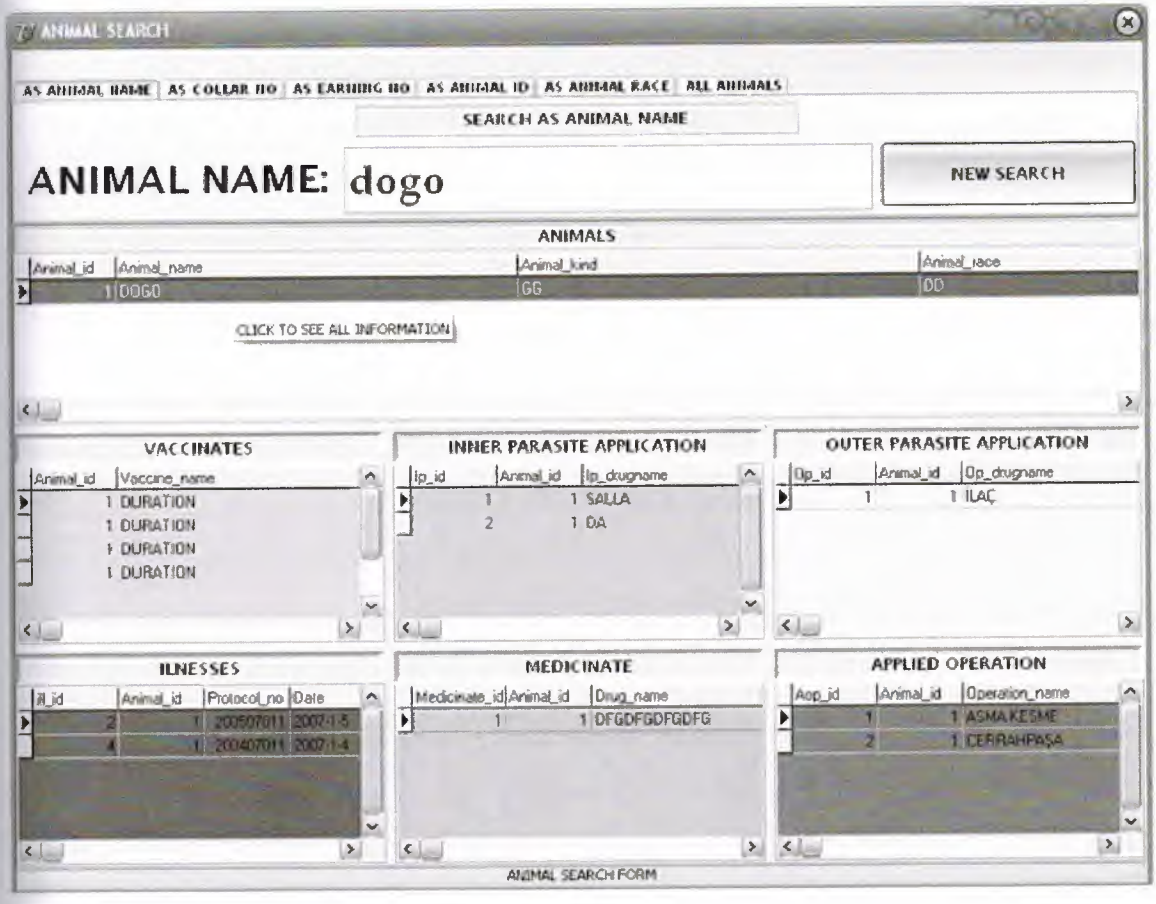

Figure 4.17

If user want to see staff knowledge, he/she must click staff button that is on main page.Than staff search form will be displayed.Figure 4.18 shows an staff search form.

As it seen there are seven criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.18 has only one of them.

When user write character from keyboard the program will check the staff name

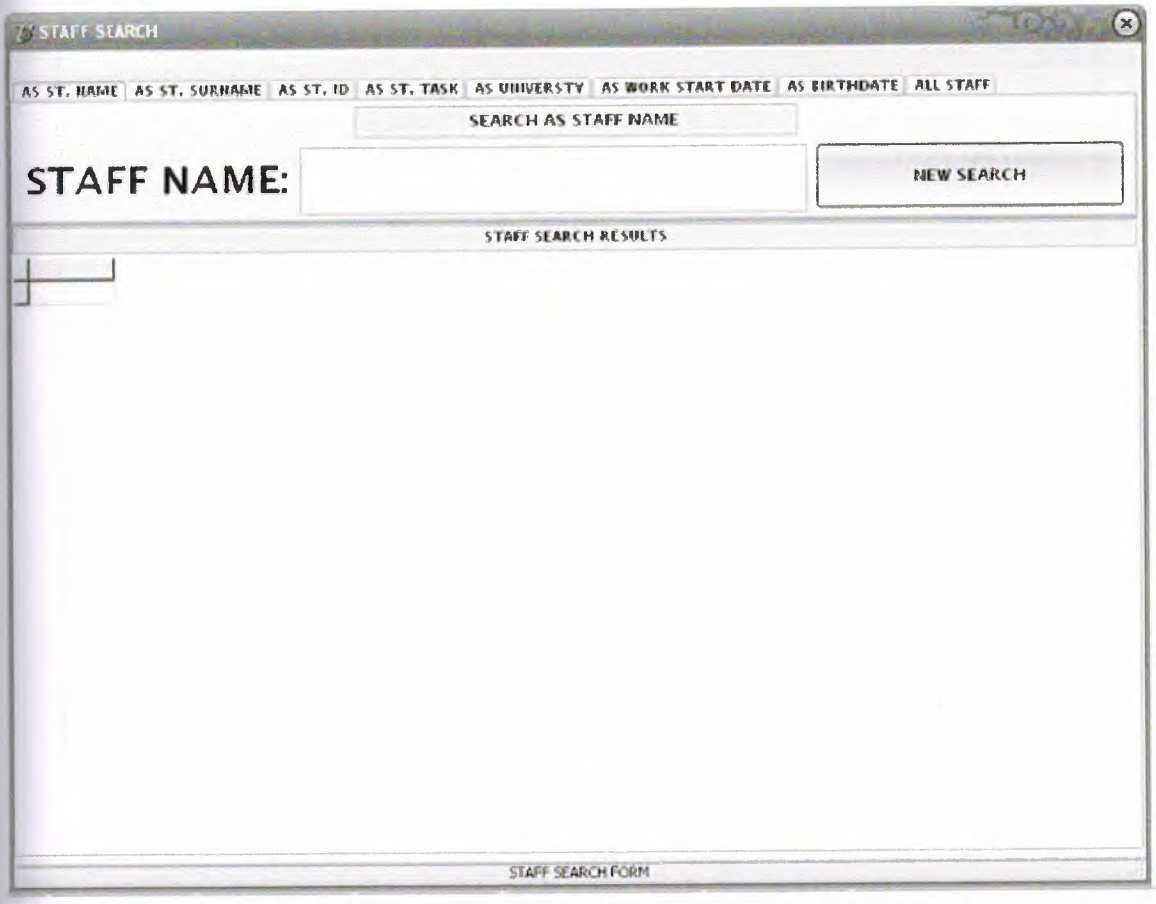

Figure 4.18

If user want to see vaccinate knowledge, he/she must click vaccinate button that is on main page.Than vaccinate search form will be displayed.Figure 4.19 shows an staff search form.

As it seen there are five criteria to make search.Well user can search for various situation. Every criteria has different page.Figure 4.19 has only one of them.

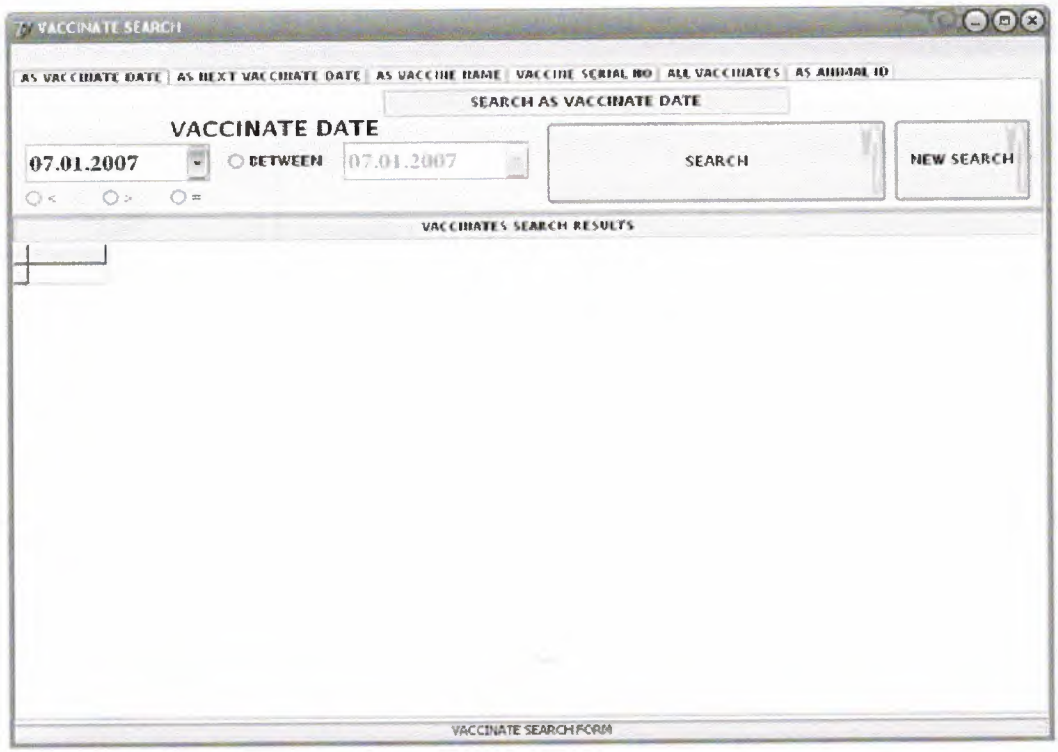

Figure 4.19

When User want to change the settings, he/she must click settings button that is on main page.Than setting page will be displayed.Figure 4.20 shows an settings form

As it seen there are two criteria to make search.Well user can change setting to various situation.User can change form color,can disable or enable skins, disable or enable picture, change skins and picture.

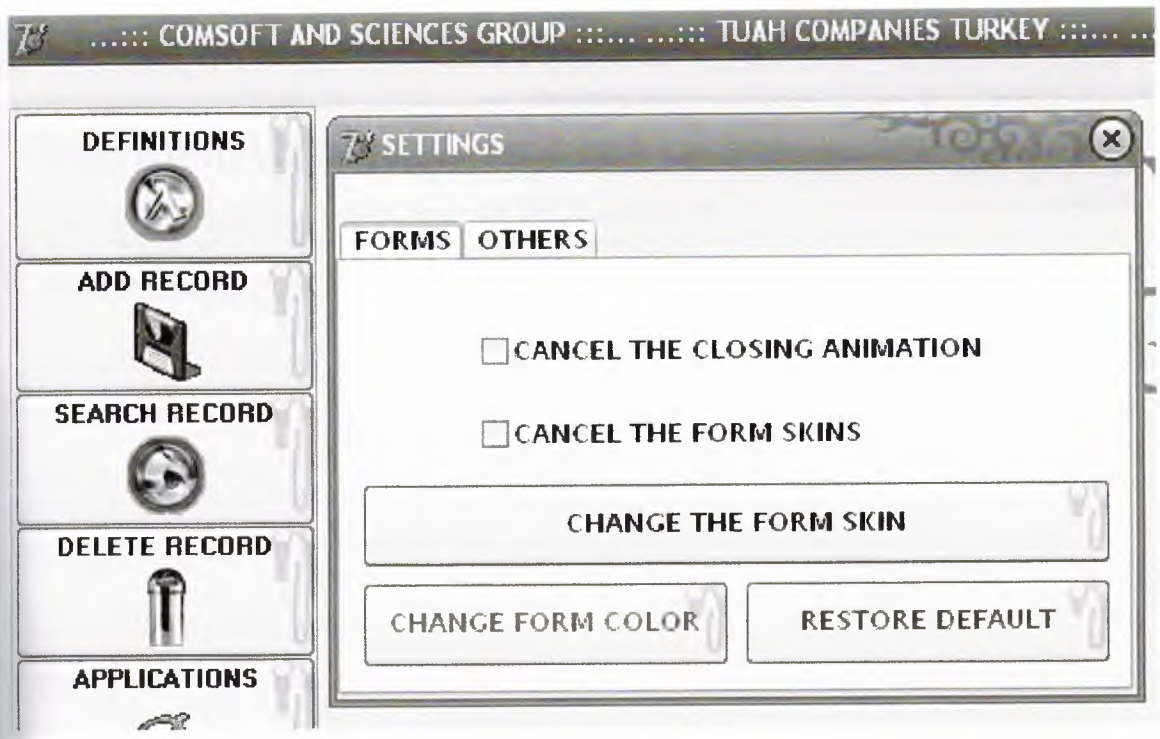

Figure 4.20

If User want to see 'What will I do today?','Which process will be made today ?', he/she must click obligation button that is on search record page.Than obligation page will be displayed.Figure 4.21 shows an settings form

As it seen there are three criteria to make search.Well user can learn to satisfy vaccinate process, inner parasite application process, outer parasite application process.

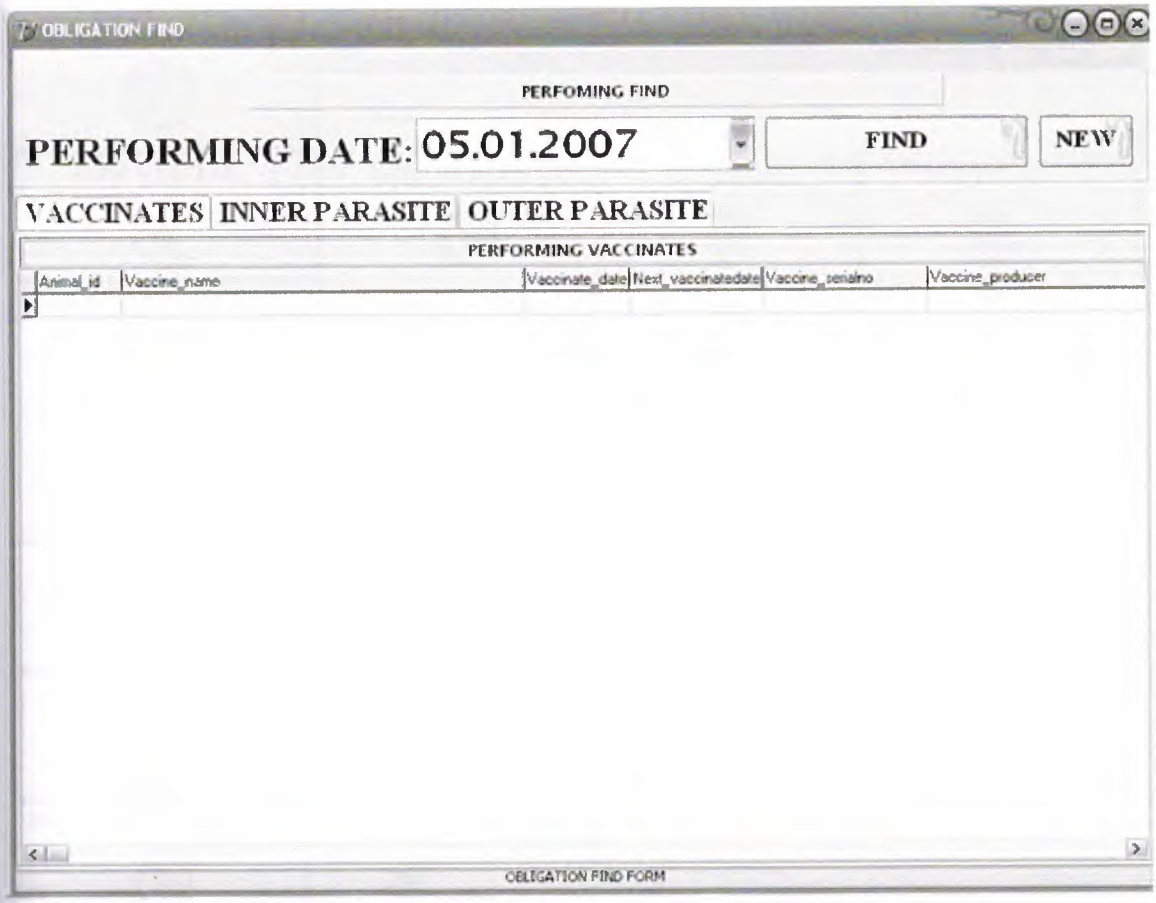

Figure 4.21

When User want to arrvive the amusement. He/she must click amusement button that is on main page.Than amusement selection page will be displayed.Figure 4.22 shows an amusement selection form

As it seen there are six selection object to fun.Well user can arrive various fun.

Figure 4.22

If User want to open a web page. He/she must click internet explorer button that is on main page.Than internet explorer page will be displayed.Figure 4.23 shows that.

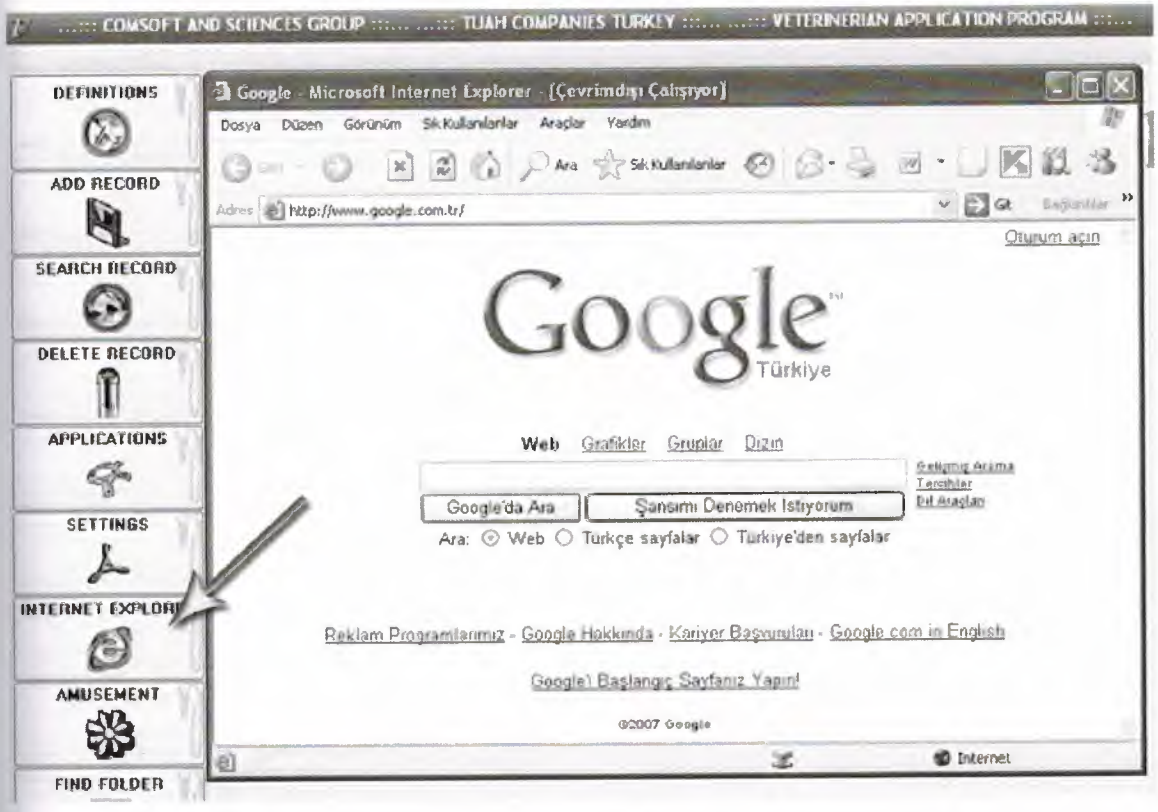

Figure 4.23

If User want toget an windows screen. He/she must click find folder button that is on main page.Than windows screen page will be displayed.Figure 4.24 shows that.In here can find folder, files.And also can process some operation about other application.

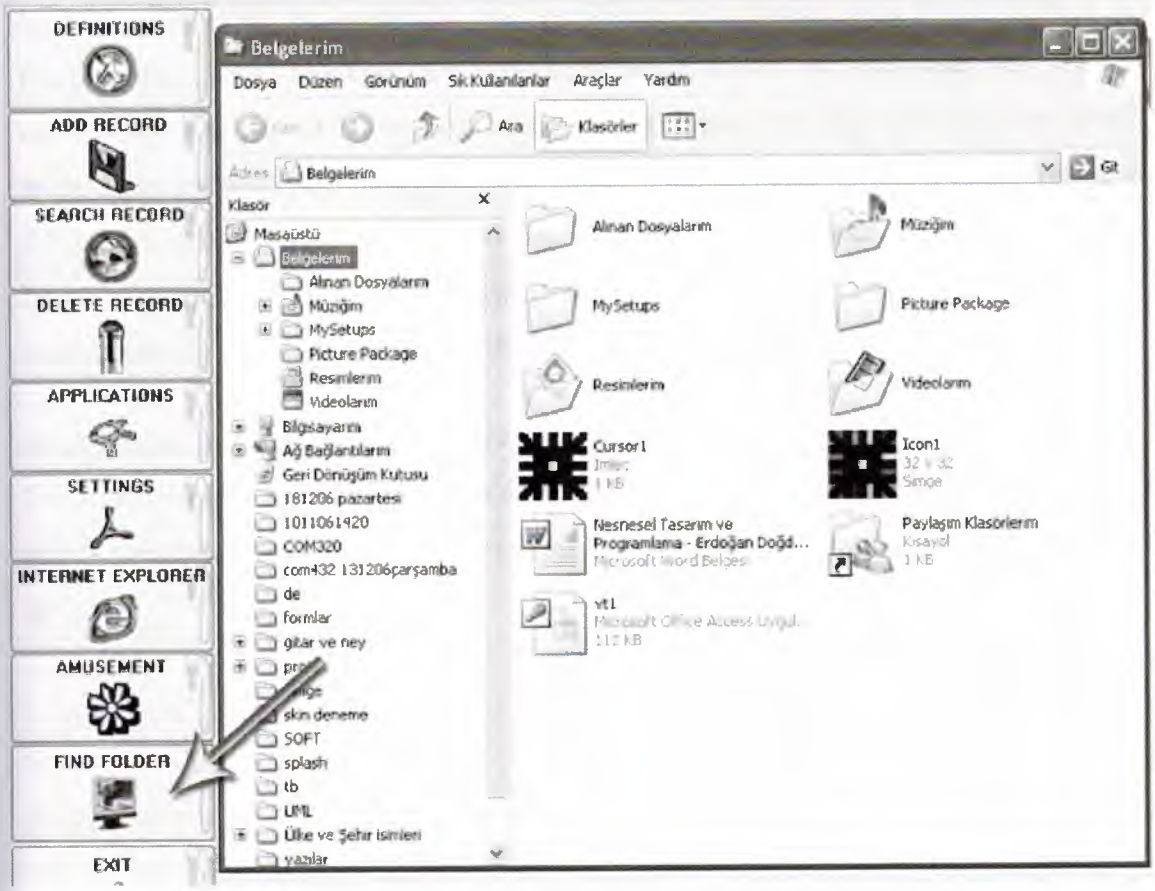
DEFINITIONS
ADD RECORD
SEARCH RECORD
DELETE RECORD
APPLICATIONS
SETTINGS
INTERNET EXPLORER
AMUSEMENT
FIND FOLDER
EXIT
Belgelerim
Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım
Ara
Klasörler
Adres
Belgelerim
Git
Klasör
Masaüstü
Belgelerim
Alınan Dosyalarım
Müziğim
MySetups
Picture Package
Resimlerim
Videolarım
Bilgisayarım
Ağ Bağlantılarım
Geri Dönüşüm Kutusu
181206 pazartesi
1011061420
COM320
com432 131206çarşamba
de
formlar
gitar ve ney
skin deneme
SOFT
splash
tb
UML
Ülke ve Şehir isimleri
yazılar
Alınan Dosyalarım
Müziğim
MySetups
Picture Package
Resimlerim
Videolarım
Cursor1
İmleç
1 KB
Icon1
32 x 32
Simge
Nesnesel Tasarım ve Programlama - Erdoğan Doğd...
Microsoft Word Belgesi
Paylaşım Klasörlerim
Kısayol
1 KB
vt1
Microsoft Office Access Uygul...
212 KB

Figure 4.24

CONCLUSION

MySQL and Delphi are powerful program.When I study with these two program,I get fun.Because these program are wonderful.Examination of the data for internal consistency and comparisons with externally available data indicates that the Delphi study appears reliable. However, the study was difficult to carry out owing to difficulties in obtaining answers from possible respondents. Thus, if a larger survey is to be undertaken to include all building components, it is recommended that committed respondents be obtained before devising the survey.

Veterinerian Application program for veterinerian and users act more facility.However Users adapt easly to the program and use it safetly.Nowadays in everywhere, in every job is combined with the computer.Well Veterinerian clinic will combine with this project.

APPENDIX

VETARINERIAN APPLICATION PROGRAM SOURCE CODE

FORM 1 CODES

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, Menus, ExtCtrls, WinSkinData, jpeg, StdCtrls,
  XPMan;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    StatusBar1: TStatusBar;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    Shape1: TShape;
    SkinData1: TSkinData;
    Label1: TLabel;
    Timer1: TTimer;
    Image1: TImage;
    XPManifest1: TXPManifest;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);

    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
```

```
  procedure SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton3MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton4MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton7MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton8MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton9MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton6MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  procedure SpeedButton10MouseMove(Sender: TObject; Shift: TShiftState;
    X, Y: Integer);
  procedure SpeedButton6Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit9, Unit41;

{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
//if (form1.Label1.Top <> 600) then//and (form1.Label1.Top > 1) then
//form1.Label1.Top:=form1.Label1.Top-1;
//if form1.Label1.Top <> 1 then
//form1.Label1.Top:=form1.Label1.Top+1;
FORM1.StatusBar1.Panels[5].Text:=TIMETOSTR(TIME);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Label1.Caption:='COMSOFT and SCIENCES'+#13+'                GROUP';
FORM1.StatusBar1.Panels[1].Text:=DATETOSTR(DATE);
FORM1.StatusBar1.Panels[5].Text:=TIMETOSTR(TIME);
```

```
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM2.SHOW;
end;

procedure TForm1.SpeedButton10Click(Sender: TObject);
begin
form41.CLOSE;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
FORM6.CLOSE;
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
 FORM3.SHOW;
end;



procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
form2.CLOSE;
form3.CLOSE;
form4.CLOSE;
form5.CLOSE;
form6.CLOSE;
form41.close;

end;

procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
```

```
form4.show;
end;

procedure TForm1.SpeedButton8Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
FORM5.SHOW;
end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM7.CLOSE;
form6.show;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM4.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM7.SHOW;
end;

procedure TForm1.SpeedButton9Click(Sender: TObject);
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
 if FileExists('C:\WINDOWS\explorer.exe') then
   winexec('C:\WINDOWS\explorer.exe',sw_shownormal);
end;

procedure TForm1.SpeedButton7Click(Sender: TObject);
```

```
begin
FORM9.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM4.CLOSE;
FORM6.CLOSE;
  if FileExists('C:\Program Files\Internet Explorer\iexplore.exe') then
    winexec('C:\Program Files\Internet Explorer\iexplore.exe',sw_shownormal);
end;

procedure TForm1.SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
FORM1.SpeedButton1.Hint:='     THIS ACTS TO DEFINE NEW
KNOWLEDGE'+#13+
                  '(STAFF, VACCINE, DRUGS, OPERATIONS, USERS)';

end;

procedure TForm1.SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton2.Hint:='USES TO SAVE NEW RECORD'+#13+
                  '   (CUSTOMER, ANIMAL)';
end;

procedure TForm1.SpeedButton3MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton3.Hint:='USE TO FIND RECORD'+#13+
                  '   (ALL CRITERIA)';
end;

procedure TForm1.SpeedButton4MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton4.Hint:='ACTS TO DELETE RECORD'+#13+
                  '    (ALL CRITERIA)';
end;

procedure TForm1.SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton5.Hint:='     USES TO SAVE NEW APPLICATION'+#13+
                  '(VACCINATE, INNER PARASITE, OUTER PARASITE)'+#13+
                  '(MEDICINATE, APPLIED OPERATIONS, ILNESSES )';
end;
```

```
procedure TForm1.SpeedButton7MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton7.Hint:='USES TO OPEN THE INTERNET EXPLORER';
end;

procedure TForm1.SpeedButton8MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton8.Hint:='USE TO HAVE FUN';
end;

procedure TForm1.SpeedButton9MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  FORM1.SpeedButton9.Hint:='USES TO SEE WINDOWS FILES OR FOLDERS';
end;

procedure TForm1.SpeedButton6MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
FORM1.SpeedButton6.Hint:='ACTS TO CHANGE THE PROGRAM SETTINGS';
end;

procedure TForm1.SpeedButton10MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
FORM1.SpeedButton10.Hint:='ACTS TO CLOSE THE PROGRAM';
end;

procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
FORM4.CLOSE;
FORM7.CLOSE;
FORM2.CLOSE;
FORM3.CLOSE;
FORM5.CLOSE;
FORM6.CLOSE;
FORM9.SHOW;
end;

end.
```

FORM 2 CODES

```
unit Unit2;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm2 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton6: TSpeedButton;
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

uses Unit10, Unit11, Unit12, Unit13, Unit14;

{$R *.dfm}

procedure TForm2.SpeedButton6Click(Sender: TObject);
begin
  form2.hide;
end;

procedure TForm2.SpeedButton1Click(Sender: TObject);
begin
  FORM10.SHOW;
  form2.Hide;
end;

procedure TForm2.SpeedButton2Click(Sender: TObject);
begin
  form11.show;
  form2.Hide;
```

```
end;

procedure TForm2.SpeedButton3Click(Sender: TObject);
begin
  FORM12.SHOW;
  FORM2.Hide;
end;

procedure TForm2.SpeedButton4Click(Sender: TObject);
begin
  FORM13.SHOW;
  FORM2.HIDE;
end;

procedure TForm2.SpeedButton5Click(Sender: TObject);
begin
  FORM14.SHOW;
  FORM2.Hide;
end;

end.
```

FORM 3 CODES

```
unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm3 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    SpeedButton11: TSpeedButton;
    SpeedButton12: TSpeedButton;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    SpeedButton15: TSpeedButton;
```

```
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton16: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton15Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton16Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);


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

var
  Form3: TForm3;

implementation

uses Unit1, Unit23, Unit24, Unit25, Unit28, Unit27, Unit29, Unit30, Unit31,
  Unit32, Unit26, Unit33, Unit34, Unit35, Unit36, Unit37;

{$R *.dfm}


procedure TForm3.SpeedButton1Click(Sender: TObject);
begin
  FORM23.SHOW;
  FORM3.Hide;
end;

procedure TForm3.SpeedButton7Click(Sender: TObject);
begin
  FORM24.SHOW;
  FORM3.Hide;
end;
```

```
procedure TForm3.SpeedButton14Click(Sender: TObject);
begin
  FORM25.SHOW;
  FORM3.Hide;
end;

procedure TForm3.SpeedButton15Click(Sender: TObject);
begin
FORM27.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton2Click(Sender: TObject);
begin
FORM28.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton5Click(Sender: TObject);
begin
form29.show;
form3.Hide;
end;

procedure TForm3.SpeedButton9Click(Sender: TObject);
begin
FORM30.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton10Click(Sender: TObject);
begin
FORM31.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton3Click(Sender: TObject);
begin
FORM32.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton4Click(Sender: TObject);
begin
FORM26.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton8Click(Sender: TObject);
begin
```

```
FORM33.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton11Click(Sender: TObject);
begin
FORM34.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton6Click(Sender: TObject);
begin
FORM35.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton12Click(Sender: TObject);
begin
FORM36.SHOW;
FORM3.Hide;
end;

procedure TForm3.SpeedButton16Click(Sender: TObject);
begin
FORM3.Hide;
end;

procedure TForm3.SpeedButton13Click(Sender: TObject);
begin
FORM37.SHOW;
FORM3.Hide;
end;

end.
```

FORM 4 CODES

```
unit Unit4;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm4 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses Unit17, Unit18, Unit19, Unit20, Unit21, Unit22;

{$R *.dfm}

procedure TForm4.SpeedButton1Click(Sender: TObject);
begin
  FORM17.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton2Click(Sender: TObject);
begin
  FORM18.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton5Click(Sender: TObject);
begin
  FORM19.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton3Click(Sender: TObject);
begin
  FORM20.SHOW;
```

```
  FORM4.Hide;
end;

procedure TForm4.SpeedButton4Click(Sender: TObject);
begin
  FORM21.SHOW;
  FORM4.Hide;
end;

procedure TForm4.SpeedButton6Click(Sender: TObject);
begin
form22.show;
form4.Hide;
end;

procedure TForm4.SpeedButton7Click(Sender: TObject);
begin
FORM4.Hide;
end;

end.
```

FORM 5 CODES

```
unit Unit5;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm5 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
```

```
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form5: TForm5;

implementation

{$R *.dfm}

procedure TForm5.SpeedButton1Click(Sender: TObject);
begin
  if FileExists('C:\Program Files\Windows Media Player\wmplayer.exe') then
    winexec('C:\Program Files\Windows Media Player\wmplayer.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton2Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\sol.exe') then
    winexec('C:\WINDOWS\system32\sol.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton3Click(Sender: TObject);
begin
  if FileExists('C:\windows\system32\freecell.exe') then
    winexec('C:\windows\system32\freecell.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton4Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\winmine.exe') then
    winexec('C:\WINDOWS\system32\winmine.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton5Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\calc.exe') then
  winexec('C:\WINDOWS\system32\calc.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton8Click(Sender: TObject);
begin
```

```
  if FileExists('C:\WINDOWS\notepad.exe') then
    winexec('C:\WINDOWS\notepad.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton6Click(Sender: TObject);
begin
  if FileExists('C:\Program Files\MSN Messenger\msnmsgr.exe') then
    winexec('C:\Program Files\MSN Messenger\msnmsgr.exe',sw_shownormal)
  else if FileExists('C:\Program Files\Messenger\msmsgs.exe') then
    winexec('C:\Program Files\Messenger\msmsgs.exe',sw_shownormal);
end;

procedure TForm5.SpeedButton7Click(Sender: TObject);
begin
form5.Hide;
end;

procedure TForm5.SpeedButton9Click(Sender: TObject);
begin
  if FileExists('C:\WINDOWS\system32\mshearts.exe') then
    winexec('C:\WINDOWS\system32\mshearts.exe', sw_shownormal);
end;

end.
```

FORM 6 CODES

```
unit Unit6;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Buttons;

type
  TForm6 = class(TForm)
    SpeedButton1: TSpeedButton;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton11: TSpeedButton;
```

```
    SpeedButton12: TSpeedButton;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form6: TForm6;

implementation

uses Unit15, Unit11, Unit12, Unit13, Unit14, Unit16, Unit17, Unit18,
  Unit19, Unit21, Unit20, Unit22;

{$R *.dfm}

procedure TForm6.SpeedButton6Click(Sender: TObject);
begin
FORM6.CLOSE;
end;

procedure TForm6.SpeedButton1Click(Sender: TObject);
begin
  Form15.LbSpeedButton1.Enabled:=FALSE;
  Form15.LbSpeedButton2.Enabled:=FALSE;
  FORM15.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton5Click(Sender: TObject);
begin
  FORM11.SpeedButton2.Enabled:=FALSE;
  FORM11.SpeedButton3.Enabled:=FALSE;
  FORM11.SHOW;
```

```
  FORM6.Close;
end;

procedure TForm6.SpeedButton9Click(Sender: TObject);
begin
  FORM12.SpeedButton2.Enabled:=FALSE;
  FORM12.SpeedButton3.Enabled:=FALSE;
  FORM12.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton4Click(Sender: TObject);
begin
  Form13.LbSpeedButton1.Enabled:=FALSE;
  Form13.LbSpeedButton2.Enabled:=FALSE;
  FORM13.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton7Click(Sender: TObject);
begin
  Form14.LbSpeedButton1.Enabled:=FALSE;
  Form14.LbSpeedButton2.Enabled:=FALSE;
  FORM14.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton8Click(Sender: TObject);
begin
  FORM16.SpeedButton3.Enabled:=FALSE;
  FORM16.SpeedButton4.Enabled:=FALSE;
  FORM16.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton2Click(Sender: TObject);
begin
  Form17.LbSpeedButton1.Enabled:=FALSE;
  Form17.LbSpeedButton2.Enabled:=FALSE;
  FORM17.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton11Click(Sender: TObject);
begin
  Form18.SpeedButton3.Enabled:=FALSE;
  Form18.SpeedButton4.Enabled:=FALSE;
  FORM18.SHOW;
  FORM6.Close;
end;
```

```
procedure TForm6.SpeedButton14Click(Sender: TObject);
begin
  Form19.LbSpeedButton1.Enabled:=FALSE;
  Form19.LbSpeedButton2.Enabled:=FALSE;
  FORM19.SHOW;
  FORM6.CLOSE;
end;

procedure TForm6.SpeedButton12Click(Sender: TObject);
begin
  Form21.LbSpeedButton1.Enabled:=FALSE;
  Form21.LbSpeedButton2.Enabled:=FALSE;
  FORM21.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton3Click(Sender: TObject);
begin
  Form20.SpeedButton3.Enabled:=FALSE;
  Form20.SpeedButton4.Enabled:=FALSE;
  FORM20.SHOW;
  FORM6.Close;
end;

procedure TForm6.SpeedButton13Click(Sender: TObject);
begin
  Form22.SpeedButton3.Enabled:=FALSE;
  Form22.SpeedButton4.Enabled:=FALSE;
  FORM22.SHOW;
  FORM6.Close;
end;

end.
```

FORM 7 CODES

```
unit Unit7;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, Menus;

type
  TForm7 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
  SpeedButton1: TSpeedButton;
  SpeedButton2: TSpeedButton;
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form7: TForm7;

implementation

uses Unit15, Unit16;

{$R *.dfm}

procedure TForm7.SpeedButton1Click(Sender: TObject);
begin
 FORM15.SHOW;
 FORM7.HIDE;
end;

procedure TForm7.SpeedButton2Click(Sender: TObject);
begin
 FORM16.SHOW;
 FORM7.Hide;
end;

end.
```

FORM 8 CODES

```
unit Unit8;

interface

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

type
 TForm8 = class(TForm)
  Image1: TImage;
 private
  { Private declarations }
 public
```

```
    { Public declarations }
  end;

var
  Form8: TForm8;

implementation

{$R *.dfm}

end.
```

FORM 9 CODES

```
unit Unit9;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, Menus, StdCtrls, jpeg, ExtDlgs;

type
  TForm9 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    PageControl1: TPageControl;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    ColorDialog1: TColorDialog;
    FontDialog1: TFontDialog;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    SpeedButton4: TSpeedButton;
    CheckBox3: TCheckBox;
    CheckBox6: TCheckBox;
    SpeedButton5: TSpeedButton;
    OpenDialog1: TOpenDialog;
    OpenPictureDialog1: TOpenPictureDialog;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    procedure SpeedButton4Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
    procedure CheckBox6Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
  private
    { Private declarations }
```

```
 public
  { Public declarations }
 end;

var
 Form9: TForm9;

implementation

uses Unit1, Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit10, Unit11,
 Unit12, Unit13, Unit14, Unit15, Unit16, Unit17, Unit18, Unit19, Unit20,
 Unit21, Unit22, Unit23, Unit24, Unit25, Unit26, Unit27, Unit28, Unit29,
 Unit30, Unit31, Unit32, Unit33, Unit34, Unit35, Unit36, Unit37, Unit38,
 Unit39, Unit40, Unit41;

{$R *.dfm}

procedure TForm9.SpeedButton4Click(Sender: TObject);
begin
 if form9.CheckBox2.Checked <> true then
 begin
  form9.OpenDialog1.Filter:='Skin Files (skn)|*.skn';
  if form9.OpenDialog1.Execute then
  begin
   form1.SkinData1.LoadFromFile(form9.OpenDialog1.FileName);
   //form1.Label1.Caption:=form1.SkinData1.SkinFile;
  end;
 end
 else
 begin
  beep;
  showmessage('YOU HAVE CANCELED THE SKINS BEFORE');
 end;

end;

procedure TForm9.CheckBox2Click(Sender: TObject);
begin
 if form9.CheckBox2.Checked = true then
  form1.SkinData1.Active:=false;
 if form9.CheckBox2.Checked = false then
  form1.SkinData1.Active:=true;

end;

procedure TForm9.CheckBox6Click(Sender: TObject);
begin
 if form9.CheckBox6.Checked = true then
 begin
  form1.Image1.Visible:=true;
```

```
 end;

 if form9.CheckBox6.Checked = false then
 begin
  form1.Image1.Visible:=false;
 end;
end;

procedure TForm9.SpeedButton5Click(Sender: TObject);
begin
 if form9.CheckBox6.Checked = true then
 begin
  if form9.OpenPictureDialog1.Execute then
   form1.Image1.Picture.LoadFromFile(form9.OpenPictureDialog1.FileName);
 end
 else
 begin
  beep;
  showmessage('YOU HAVE CANCELED WALLPAPERS BEFORE');
 end;
end;

procedure TForm9.SpeedButton1Click(Sender: TObject);
begin
 if form9.ColorDialog1.Execute then
 begin
  form1.Color:=form9.ColorDialog1.Color;
  form2.Color:=form9.ColorDialog1.Color;
  form3.Color:=form9.ColorDialog1.Color;
  form4.Color:=form9.ColorDialog1.Color;
  form5.Color:=form9.ColorDialog1.Color;
  form6.Color:=form9.ColorDialog1.Color;
  form7.Color:=form9.ColorDialog1.Color;
  form9.Color:=form9.ColorDialog1.Color;
  form10.Color:=form9.ColorDialog1.Color;
  form11.Color:=form9.ColorDialog1.Color;
  form12.Color:=form9.ColorDialog1.Color;
  form13.Color:=form9.ColorDialog1.Color;
  form14.Color:=form9.ColorDialog1.Color;
  form15.Color:=form9.ColorDialog1.Color;
  form16.Color:=form9.ColorDialog1.Color;
  form17.Color:=form9.ColorDialog1.Color;
  form18.Color:=form9.ColorDialog1.Color;
  form19.Color:=form9.ColorDialog1.Color;
  form20.Color:=form9.ColorDialog1.Color;
  form21.Color:=form9.ColorDialog1.Color;
  form22.Color:=form9.ColorDialog1.Color;
  form23.Color:=form9.ColorDialog1.Color;
  form24.Color:=form9.ColorDialog1.Color;
  form25.Color:=form9.ColorDialog1.Color;
```

```
    form26.Color:=form9.ColorDialog1.Color;
    form27.Color:=form9.ColorDialog1.Color;
    form28.Color:=form9.ColorDialog1.Color;
    form29.Color:=form9.ColorDialog1.Color;
    form30.Color:=form9.ColorDialog1.Color;
    form31.Color:=form9.ColorDialog1.Color;
    form32.Color:=form9.ColorDialog1.Color;
    form33.Color:=form9.ColorDialog1.Color;
    form34.Color:=form9.ColorDialog1.Color;
    form35.Color:=form9.ColorDialog1.Color;
    form36.Color:=form9.ColorDialog1.Color;
    form37.Color:=form9.ColorDialog1.Color;
    form38.Color:=form9.ColorDialog1.Color;
    form39.Color:=form9.ColorDialog1.Color;
    form40.Color:=form9.ColorDialog1.Color;
    form41.Color:=form9.ColorDialog1.Color;
  end;

end;

procedure TForm9.SpeedButton2Click(Sender: TObject);
begin
form1.color:=clBlack;
form2.color:=clBtnFace;
form3.color:=clBtnFace;
form4.color:=clBtnFace;
form5.color:=clBtnFace;
form6.color:=clBtnFace;
form7.color:=clBtnFace;
form9.color:=clBtnFace;
form10.color:=$004080FF;
form11.color:=$00C08080;
form12.color:=$00400040;
form13.color:=clGray;
form14.color:=clSilver;
form15.color:=$00404080;
form16.color:=clBtnFace;
form17.color:=clMoneyGreen;
form18.color:=$00400000;
form19.color:=clBlack;
form20.color:=clBtnFace;
form21.color:=$00404080;
form22.color:=clInactiveCaptionText;
form23.color:=clBtnFace;
form24.color:=clBtnFace;
form25.color:=clBtnFace;
form26.color:=clBtnFace;
form27.color:=clBtnFace;
form28.color:=clBtnFace;
form29.color:=clBtnFace;
```

```
form30.color:=clBtnFace;
form31.color:=clBtnFace;
form32.color:=clBtnFace;
form33.color:=clBtnFace;
form34.color:=clBtnFace;
form35.color:=clBtnFace;
form36.color:=clBtnFace;
form37.color:=clBtnFace;
form38.color:=clBtnFace;
form39.color:=$00D0A5A4;
form40.color:=clBtnFace;
form41.color:=clBtnFace;
end;

end.
```

FORM 10 CODES

```
unit Unit10;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Mask, Menus, DB, ADODB, Buttons, Grids,
  DBGrids, LbSpeedButton, ExtCtrls;

type
  TForm10 = class(TForm)
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Label8: TLabel;
```

```
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Edit4: TEdit;
    MaskEdit1: TMaskEdit;
    Memo1: TMemo;
    Edit5: TEdit;
    ComboBox3: TComboBox;
    ComboBox4: TComboBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Edit6: TEdit;
    DateTimePicker3: TDateTimePicker;
    MaskEdit2: TMaskEdit;
    Edit7: TEdit;
    Label18: TLabel;
    Memo2: TMemo;
    StatusBar1: TStatusBar;
    Label19: TLabel;
    Edit8: TEdit;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    DBGrid1: TDBGrid;
    SpeedButton1: TSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    Panel1: TPanel;
    ADOQuery2: TADOQuery;
    DataSource2: TDataSource;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;
```

```
implementation

uses Unit38;

{$R *.dfm}

procedure TForm10.FormCreate(Sender: TObject);
begin
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm10.SpeedButton1Click(Sender: TObject);
begin
 FORM38.SHOW;
 TA:=10;
end;

procedure TForm10.FormShow(Sender: TObject);
begin
form10.DateTimePicker1.Date:=date;
form10.DateTimePicker2.Date:=date;
//form10.DateTimePicker3.Date:=date;
form10.ADOQuery2.Close;
form10.ADOQuery2.SQL.Text:='select * from staff';
form10.ADOQuery2.Open;
end;

procedure TForm10.LbSpeedButton1Click(Sender: TObject);
begin
 form10.ADOQuery1.Close;
 form10.ADOQuery1.SQL.Text:='select * from staff where
Staff_name='+#39+form10.Edit2.Text+#39+' and
Staff_surname='+#39+form10.Edit3.Text+#39+' and
S_birthdate='+#39+datetostr(form10.DateTimePicker2.date)+#39;
 form10.ADOQuery1.Open;

 if form10.ADOQuery1.RecordCount = 0 then
 begin
  if (form10.Edit2.Text <> '') or (form10.Edit3.Text <> '') then
  begin
   form10.ADOQuery1.Close;
   form10.ADOQuery1.SQL.Text:='insert into staff
(Staff_name,Staff_surname,Staff_task,University,Grade_state,S_workstartdate,S_birthd
ate,S_TCidno,S_homephone,S_mobilphone,S_address,S_town,S_city,S_country,S_ema
il,S_web,S_leavingdate,S_note) values
('+#39+Form10.Edit2.Text+#39+','+#39+form10.Edit3.Text+#39+','+#39+form10.Com
boBox1.Text+#39+','+#39+form10.Edit4.Text+#39+','+#39+form10.ComboBox2.Text+
#39+','+#39+datetostr(form10.DateTimePicker1.date)+#39+','+#39+datetostr(form10.D
```

```
ateTimePicker2.date)+#39+','+#39+form10.Edit8.Text+#39+','+#39+form10.MaskEdit1
.Text+#39+','+#39+form10.MaskEdit2.Text+#39+','+#39+form10.Memo1.Text+#39+','
+#39+form10.Edit5.Text+#39+','+#39+form10.ComboBox3.Text+#39+','+#39+form10.
ComboBox4.Text+#39+','+#39+form10.Edit6.Text+#39+','+#39+form10.Edit7.Text+#3
9+','+#39+datetostr(form10.DateTimePicker3.date)+#39+','+#39+form10.Memo2.Text+
#39+')';
     form10.ADOQuery1.ExecSQL;
     form10.ADOQuery1.Close;
     form10.ADOQuery1.SQL.Text:='select * from staff';
     form10.ADOQuery1.Open;
     showmessage('RECORD SAVED');
     FORM10.LbSpeedButton4.Click;
   end
   else
    showmessage('CHECK THE FORM FOR EMPTY PLACE');
 end
 else
  showmessage('RECORD HAS RECORDED BEFORE');
end;

procedure TForm10.LbSpeedButton4Click(Sender: TObject);
begin
FORM10.Edit1.Clear;
FORM10.Edit2.Clear;
FORM10.Edit3.Clear;
FORM10.ComboBox1.Text:='Select One';
FORM10.Edit4.Clear;
FORM10.ComboBox2.Text:='Select One';
FORM10.DateTimePicker1.Date:=DATE;

dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'd/m/yyyy';

FORM10.DateTimePicker2.Date:=STRTODATE('09.09.9999');
FORM10.DateTimePicker3.Date:=STRTODATE('09.09.9999');
form10.Edit8.Clear;
form10.MaskEdit1.Clear;
form10.MaskEdit2.Clear;
form10.Memo1.Clear;
form10.Edit5.Clear;
FORM10.ComboBox3.Text:='Select One';
FORM10.ComboBox4.Text:='Select One';
form10.Edit6.Clear;
form10.Edit7.Clear;
form10.Memo2.clear;
form10.Edit2.SetFocus;

dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
```

```
shortdateformat := 'yyyy/m/d';

form10.ADOQuery2.Close;
form10.ADOQuery2.SQL.Text:='select * from staff';
form10.ADOQuery2.Open;


end;

procedure TForm10.Edit1Change(Sender: TObject);
begin
  form38.Hide;
  form10.ADOQuery1.Close;
  form10.ADOQuery1.SQL.Text:='select * from staff where
staff_id='+#39+form10.Edit1.Text+#39;
  form10.ADOQuery1.Open;
  if form10.ADOQuery1.RecordCount <> 0 then
  begin
    form10.Edit1.Text:=FORM10.ADOQuery1.Fields[0].Text;
    form10.Edit2.Text:=FORM10.ADOQuery1.Fields[1].Text;
    form10.Edit3.Text:=FORM10.ADOQuery1.Fields[2].Text;
    form10.ComboBox1.Text:=FORM10.ADOQuery1.Fields[3].Text;
    form10.Edit4.Text:=FORM10.ADOQuery1.Fields[4].Text;
    form10.ComboBox2.Text:=FORM10.ADOQuery1.Fields[5].Text;

    dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
    shortdateformat := 'd/m/yyyy';

    form10.DateTimePicker1.Date:=strtodate(FORM10.ADOQuery1.Fields[6].Text);
    form10.DateTimePicker2.Date:=strtodate(FORM10.ADOQuery1.Fields[7].Text);
    form10.Edit8.Text:=FORM10.ADOQuery1.Fields[8].Text;
    form10.MaskEdit1.Text:=FORM10.ADOQuery1.Fields[9].Text;
    form10.MaskEdit2.Text:=FORM10.ADOQuery1.Fields[10].Text;
    form10.Memo1.Text:=FORM10.ADOQuery1.Fields[11].Text;
    form10.Edit5.Text:=FORM10.ADOQuery1.Fields[12].Text;
    form10.ComboBox3.Text:=FORM10.ADOQuery1.Fields[13].Text;
    form10.ComboBox4.Text:=FORM10.ADOQuery1.Fields[14].Text;
    form10.Edit6.Text:=FORM10.ADOQuery1.Fields[15].Text;
    form10.Edit7.Text:=FORM10.ADOQuery1.Fields[16].Text;
    form10.DateTimePicker3.Date:=strtodate(FORM10.ADOQuery1.Fields[17].Text);
    form10.Memo2.Text:=FORM10.ADOQuery1.Fields[18].Text;

    dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
    shortdateformat := 'yyyy/m/d';
  end;

end;
```

```
procedure TForm10.LbSpeedButton2Click(Sender: TObject);
begin
  IF (FORM10.Edit1.Text <> '') AND (FORM10.Edit2.Text <> '') AND
(FORM10.Edit3.Text <> '') THEN
  BEGIN
    FORM10.ADOQuery2.Close;
    FORM10.ADOQuery2.SQL.Text:='UPDATE staff set Staff_name=
'+#39+form10.Edit2.Text+#39+', Staff_surname= '+#39+form10.Edit3.Text+#39+',
Staff_task='+#39+form10.ComboBox1.Text+#39+',
University='+#39+form10.Edit4.Text+#39+',
Grade_state='+#39+form10.ComboBox2.Text+#39+',
S_workstartdate='+#39+datetostr(form10.DateTimePicker1.date)+#39+',
S_birthdate='+#39+datetostr(form10.DateTimePicker2.date)+#39+',
S_TCidno='+#39+form10.Edit8.Text+#39+',
S_homephone='+#39+form10.MaskEdit1.Text+#39+',
S_mobilphone='+#39+form10.MaskEdit2.Text+#39+',
S_address='+#39+form10.Memo1.Text+#39+',
S_town='+#39+form10.Edit5.Text+#39+',
S_city='+#39+form10.ComboBox3.Text+#39+',
S_country='+#39+form10.ComboBox4.Text+#39+',
S_email='+#39+form10.Edit6.Text+#39+', S_web='+#39+form10.Edit7.Text+#39+',
S_leavingdate='+#39+datetostr(form10.DateTimePicker3.date)+#39+',
S_note='+#39+form10.Memo2.Text+#39+' WHERE
Staff_id='+#39+form10.Edit1.Text+#39;
    form10.ADOQuery2.ExecSQL;
    showmessage('RECORD UPDATED');
    FORM10.LbSpeedButton4.Click;
  END
  ELSE
    SHOWMESSAGE('PLEASE CHOOSE STAFF ID AND BE SURE'+#13+'TO
COMPLETE THE EMPTY PLACE');

end;

procedure TForm10.FormHide(Sender: TObject);
begin
FORM10.LbSpeedButton4.Click;
FORM10.ADOQuery1.Close;
FORM10.ADOQuery2.Close;
end;

procedure TForm10.FormClose(Sender: TObject; var Action: TCloseAction);
begin
FORM10.LbSpeedButton4.Click;
FORM10.ADOQuery1.Close;
FORM10.ADOQuery2.Close;
end;

end.
```

FORM 11 CODES

```
unit Unit11;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, Grids, DBGrids, DB, ADODB, ComCtrls, Buttons,
  StdCtrls, Menus;

type
  TForm11 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    ComboBox1: TComboBox;
    Label4: TLabel;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    StatusBar1: TStatusBar;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure DBGrid1KeyUp(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
```

```
  { Public declarations }
 end;

var
 Form11: TForm11;
 SS:WORD;
implementation

uses Unit12;

{$R *.dfm}

procedure TForm11.FormShow(Sender: TObject);
begin
form11.ADOQuery1.SQL.Text:='select * from vaccines';
form11.ADOQuery1.Open;
end;

procedure TForm11.SpeedButton2Click(Sender: TObject);
begin
 if (form11.Edit2.Text <> '') then
 begin
   form11.ADOQuery2.Close;
   form11.ADOQuery2.SQL.Text:='select * from vaccines where
vaccine_name='+#39+form11.Edit2.Text+#39;
   form11.ADOQuery2.Open;
   if form11.ADOQuery2.RecordCount = 0 then
   begin
     form11.ADOQuery2.Close;
     form11.ADOQuery2.SQL.Text:='insert into vaccines
(vaccine_name,vaccine_duration) values
('+#39+form11.Edit2.Text+#39+','+#39+form11.ComboBox1.Text+#39+')';
     form11.ADOQuery2.ExecSQL;
     showmessage('RECORD SAVED');
     FORM11.SpeedButton5.Click;
   END
   else
     showmessage('RECORD HAS SAVED BEFORE');
 END
 ELSE
   SHOWMESSAGE('BE SURE TO COMPLETE THE EMPTY PLACE');


end;

procedure TForm11.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM11.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
   FORM11.Edit1.Text:=FORM11.DBGrid1.Fields[0].Text;
```

```
  FORM11.Edit2.Text:=FORM11.DBGrid1.Fields[1].Text;
  FORM11.ComboBox1.Text:=FORM11.DBGrid1.Fields[2].Text;
 END
end;

procedure TForm11.DBGrid1KeyUp(Sender: TObject; var Key: Word;
 Shift: TShiftState);
begin
 IF  FORM11.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
  FORM11.Edit1.Text:=FORM11.DBGrid1.Fields[0].Text;
  FORM11.Edit2.Text:=FORM11.DBGrid1.Fields[1].Text;
  FORM11.ComboBox1.Text:=FORM11.DBGrid1.Fields[2].Text;
 END
end;

procedure TForm11.SpeedButton3Click(Sender: TObject);
begin
 IF  (FORM11.Edit2.Text <> '') OR ((FORM11.Edit1.Text <> '') AND
(FORM11.Edit2.Text <> ''))  THEN
 BEGIN
  FORM11.ADOQuery3.Close;
  FORM11.ADOQuery3.SQL.Text:='UPDATE vaccines set
vaccine_name='+#39+form11.Edit2.Text+#39+',
vaccine_duration='+#39+form11.ComboBox1.Text+#39+' where
vaccine_id='+#39+form11.Edit1.Text+#39;
  form11.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  FORM11.SpeedButton5.Click;
 end
 else
  showmessage('PLEASE BE SURE TO COMPLETE EMPTY PLACE');
end;

procedure TForm11.SpeedButton4Click(Sender: TObject);
begin
 IF  (FORM11.Edit1.Text <> '') THEN
 BEGIN
  SS:=MESSAGEDLG('ARE YOU SURE TO DELETE  " '+FORM11.Edit2.Text+' "
?',MTWARNING,[MBYES ,MBNO],0);
  IF SS = MRYES then
  begin
   FORM11.ADOQuery3.Close;
   FORM11.ADOQuery3.SQL.Text:='DELETE FROM vaccines where
vaccine_id='+#39+form11.Edit1.Text+#39;
   form11.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   FORM11.SpeedButton5.Click;
  end;
 end
```

```
 else
   showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm11.SpeedButton5Click(Sender: TObject);
begin
  FORM11.Edit1.Clear;
  FORM11.Edit2.Clear;
  FORM11.ComboBox1.Text:='Select One';
  form11.Edit2.SetFocus;
  FORM11.SpeedButton2.Enabled:=TRUE;
  FORM11.SpeedButton3.Enabled:=TRUE;
  form11.ADOQuery1.Close;
  form11.ADOQuery1.SQL.Text:='select * from vaccines';
  form11.ADOQuery1.Open;
end;

procedure TForm11.FormClose(Sender: TObject; var Action: TCloseAction);
begin
FORM11.SpeedButton5.Click;
form11.ADOQuery1.Close;
form11.ADOQuery2.Close;
form11.ADOQuery3.Close;
end;

procedure TForm11.FormHide(Sender: TObject);
begin
FORM11.SpeedButton5.Click;
form11.ADOQuery1.Close;
form11.ADOQuery2.Close;
form11.ADOQuery3.Close;
end;

end.
```

FORM 12 CODES

```
unit Unit12;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, Grids, DBGrids, ExtCtrls, Buttons, StdCtrls, Menus,
  DB, ADODB;

type
  TForm12 = class(TForm)
```

```
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Edit1: TEdit;
    Edit2: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    Label5: TLabel;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    StatusBar1: TStatusBar;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure SpeedButton3Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form12: TForm12;
  SS1:WORD;
implementation

{$R *.dfm}

procedure TForm12.FormShow(Sender: TObject);
begin
  form12.ADOQuery1.Close;
```

```
 form12.ADOQuery1.SQL.Text:='select * from drugs';
 form12.ADOQuery1.Open;
end;

procedure TForm12.SpeedButton2Click(Sender: TObject);
begin
 if (form12.Edit2.Text <> '') and (form12.ComboBox1.Text <> 'Select One') then
 begin
  form12.ADOQuery2.Close;
  form12.ADOQuery2.SQL.Text:='select * from drugs where
drug_name='+#39+form12.Edit2.Text+#39;
  form12.ADOQuery2.Open;
  if form12.ADOQuery2.RecordCount = 0 then
  begin
   form12.ADOQuery2.Close;
   form12.ADOQuery2.SQL.Text:='insert into drugs
(drug_name,drug_duration,drug_kind) values
('+#39+form12.Edit2.Text+#39+','+#39+form12.ComboBox1.Text+#39+','+#39+form1
2.ComboBox2.Text+#39+')';
   form12.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   Form12.SpeedButton5.Click;
  END
  ELSE
   SHOWMESSAGE('RECORD HAS SAVED BEFORE');
 END
 ELSE
  SHOWMESSAGE('BE SURE TO COMPLETE THE EMPTY PLACE');

end;

procedure TForm12.FormKeyPress(Sender: TObject; var Key: Char);
begin
 {IF KEY=VK_F2 THEN
 BEGIN
  if (form12.Edit2.Text <> '') and (form12.ComboBox1.Text <> 'Select One') then
  begin
   form12.ADOQuery2.Close;
   form12.ADOQuery2.SQL.Text:='select * from drugs where
drug_name='+#39+form12.Edit2.Text+#39;
   form12.ADOQuery2.Open;
   if form12.ADOQuery2.RecordCount = 0 then
    begin
     form12.ADOQuery2.Close;
     form12.ADOQuery2.SQL.Text:='insert into drugs
(drug_name,drug_duration,drug_kind) values
('+#39+form12.Edit2.Text+#39+','+#39+form12.ComboBox1.Text+#39+','+#39+form1
2.ComboBox2.Text+#39+')';
     form12.ADOQuery2.ExecSQL;
     showmessage('RECORD SAVED');
```

```
      FORM12.ADOQuery1.Close;
      FORM12.ADOQuery1.SQL.Text:='SELECT * FROM drugs';
      FORM12.ADOQuery1.Open;
     END
     ELSE
      SHOWMESSAGE('RECORD HAS SAVED BEFORE');
   END
   ELSE
    SHOWMESSAGE('BE SURE TO COMPLETE THE EMPTY PLACE');
 END;}
end;

procedure TForm12.SpeedButton3Click(Sender: TObject);
begin
 IF (FORM12.Edit1.Text <> '') AND (FORM12.Edit2.Text <> '')   THEN
 BEGIN
   FORM12.ADOQuery3.Close;
   FORM12.ADOQuery3.SQL.Text:='UPDATE drugs set
drug_name='+#39+form12.Edit2.Text+#39+',
drug_duration='+#39+form12.ComboBox1.Text+#39+',
drug_kind='+#39+form12.ComboBox2.Text+#39+'  where
drug_id='+#39+form12.Edit1.Text+#39;
   form12.ADOQuery3.ExecSQL;
   showmessage('RECORD UPDATED');
   Form12.SpeedButton5.Click;
 end
 else
   showmessage('PLEASE BE SURE TO COMPLETE EMPTY PLACE');
end;


procedure TForm12.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM12.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
   FORM12.Edit1.Text:=FORM12.DBGrid1.Fields[0].Text;
   FORM12.Edit2.Text:=FORM12.DBGrid1.Fields[1].Text;
   FORM12.ComboBox1.Text:=FORM12.DBGrid1.Fields[2].Text;
   FORM12.ComboBox2.Text:=FORM12.DBGrid1.Fields[3].Text;
 END
end;

procedure TForm12.SpeedButton4Click(Sender: TObject);
begin
 IF  (FORM12.Edit1.Text <> '') THEN
 BEGIN
   SS1:=MESSAGEDLG('ARE YOU SURE TO DELETE  " '+FORM12.Edit2.Text+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS1 = MRYES then
   BEGIN
```

```
    FORM12.ADOQuery3.Close;
    FORM12.ADOQuery3.SQL.Text:='DELETE FROM drugs where
drug_id='+#39+form12.Edit1.Text+#39;
    form12.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form12.SpeedButton5.Click;
   END;
  end
  else
   showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm12.SpeedButton5Click(Sender: TObject);
begin
  FORM12.Edit1.Clear;
  FORM12.Edit2.Clear;
  FORM12.ComboBox1.Text:='Select One';
  FORM12.ComboBox2.Text:='Select One';
  form12.Edit2.SetFocus;
  FORM12.SpeedButton2.Enabled:=TRUE;
  FORM12.SpeedButton3.Enabled:=TRUE;

  form12.ADOQuery1.Close;
  form12.ADOQuery1.SQL.Text:='select * from drugs';
  form12.ADOQuery1.Open;
end;

procedure TForm12.FormHide(Sender: TObject);
begin
  Form12.SpeedButton5.Click;
  form12.ADOQuery1.Close;
  form12.ADOQuery2.Close;
  form12.ADOQuery3.Close;
end;

procedure TForm12.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Form12.SpeedButton5.Click;
  form12.ADOQuery1.Close;
  form12.ADOQuery2.Close;
  form12.ADOQuery3.Close;
end;

end.
```

FORM 13 CODES

```
unit Unit13;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, Grids, DBGrids, ComCtrls, LbSpeedButton, Buttons,
  StdCtrls, Menus, DB, ADODB;

type
  TForm13 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    StatusBar1: TStatusBar;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form13: TForm13;
  SS13:WORD;
```

```
implementation

{$R *.dfm}

procedure TForm13.FormShow(Sender: TObject);
begin
  form13.ADOQuery1.Close;
  form13.ADOQuery1.SQL.Text:='select * from operations';
  form13.ADOQuery1.Open;
end;

procedure TForm13.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form13.LbSpeedButton4.Click;
  form13.ADOQuery1.Close;
  form13.ADOQuery2.Close;
  form13.ADOQuery3.Close;
end;

procedure TForm13.FormHide(Sender: TObject);
begin
  form13.LbSpeedButton4.Click;
  form13.ADOQuery1.Close;
  form13.ADOQuery2.Close;
  form13.ADOQuery3.Close;
end;

procedure TForm13.LbSpeedButton1Click(Sender: TObject);
begin
  if (form13.Edit2.Text <> '') then
  begin
    form13.ADOQuery2.Close;
    form13.ADOQuery2.SQL.Text:='select * from operations where
operation_name='+#39+form13.Edit2.Text+#39;
    form13.ADOQuery2.Open;
    if form13.ADOQuery2.RecordCount = 0 then
    begin
      form13.ADOQuery2.Close;
      form13.ADOQuery2.SQL.Text:='insert into operations (operation_name) values
('+#39+form13.Edit2.Text+#39+')';
      form13.ADOQuery2.ExecSQL;
      showmessage('RECORD SAVED');
      form13.LbSpeedButton4.Click;
    END
    ELSE
      SHOWMESSAGE('RECORD HAS SAVED BEFORE');
  END
  ELSE
    SHOWMESSAGE('BE SURE TO FILL THE OPERATION NAME');
end;
```

```
procedure TForm13.LbSpeedButton2Click(Sender: TObject);
begin
  IF (FORM13.Edit1.Text <> '') AND (FORM13.Edit2.Text <> '')  THEN
 BEGIN
  FORM13.ADOQuery3.Close;
  FORM13.ADOQuery3.SQL.Text:='UPDATE operations set
operation_name='+#39+form13.Edit2.Text+#39+' where
operation_id='+#39+form13.Edit1.Text+#39;
  form13.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  form13.LbSpeedButton4.Click;
 end
 else
  showmessage('PLEASE BE SURE TO COMPLETE EMPTY PLACE');
end;

procedure TForm13.LbSpeedButton3Click(Sender: TObject);
begin
 IF  (FORM13.Edit1.Text <> '') THEN
 BEGIN
  SS13:=MESSAGEDLG('ARE YOU SURE TO DELETE  ' '+FORM13.Edit2.Text+'
' ?',MTWARNING,[MBYES ,MBNO],0);
  IF SS13 = MRYES then
  BEGIN
   FORM13.ADOQuery3.Close;
   FORM13.ADOQuery3.SQL.Text:='DELETE FROM operations where
operation_id='+#39+form13.Edit1.Text+#39;
   form13.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   form13.LbSpeedButton4.Click;
  END;
 end
 else
  showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm13.LbSpeedButton4Click(Sender: TObject);
begin
 FORM13.Edit1.Clear;
 FORM13.Edit2.Clear;
 form13.Edit2.SetFocus;
 Form13.LbSpeedButton1.Enabled:=TRUE;
 Form13.LbSpeedButton2.Enabled:=TRUE;

 form13.ADOQuery1.Close;
 form13.ADOQuery1.SQL.Text:='select * from operations';
 form13.ADOQuery1.Open;
end;
```

```
procedure TForm13.DBGrid1CellClick(Column: TColumn);
begin
IF  FORM13.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
   FORM13.Edit1.Text:=FORM13.DBGrid1.Fields[0].Text;
   FORM13.Edit2.Text:=FORM13.DBGrid1.Fields[1].Text;
 END
end;

end.
```

FORM 14 CODES

```
unit Unit14;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, ExtCtrls, Grids, DBGrids, LbSpeedButton, Buttons,
  StdCtrls, Menus, DB, ADODB;

type
  TForm14 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    SpeedButton1: TSpeedButton;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    StatusBar1: TStatusBar;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
```

```
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form14: TForm14;
  SS14:WORD;
implementation

uses Unit38, Unit10, Unit12;

{$R *.dfm}

procedure TForm14.SpeedButton1Click(Sender: TObject);
begin
  form38.show;
  TA:=14;
end;

procedure TForm14.FormShow(Sender: TObject);
begin
  form14.ADOQuery1.Close;
  form14.ADOQuery1.SQL.Text:='select * from users';
  form14.ADOQuery1.Open;
end;

procedure TForm14.LbSpeedButton1Click(Sender: TObject);
begin
  if (form14.Edit1.Text <> '') and (form14.Edit2.Text <> '') and (form14.Edit3.Text <>
'') then
  begin
    form14.ADOQuery2.Close;
    form14.ADOQuery2.SQL.Text:='select * from users where
user_name='+#39+form14.Edit2.Text+#39;
    form14.ADOQuery2.Open;
    if form14.ADOQuery2.RecordCount = 0 then
    begin
```

```
    form14.ADOQuery2.Close;
    form14.ADOQuery2.SQL.Text:='insert into users
(user_name,password,staff_id,staff_state,staff_pozition) values
('+#39+form14.Edit2.Text+#39+','+#39+form14.Edit3.Text+#39+','+#39+form14.Edit1.
Text+#39+','+#39+form14.ComboBox1.Text+#39+','+#39+form14.ComboBox2.Text+#
39+')';
    form14.ADOQuery2.ExecSQL;
    showmessage('RECORD SAVED');
    Form14.LbSpeedButton4.Click;
   END
   ELSE
    SHOWMESSAGE('USER NAME IS USED');
 END
 ELSE if form14.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm14.LbSpeedButton2Click(Sender: TObject);
begin
 IF (FORM14.Edit1.Text <> '') AND (FORM14.Edit2.Text <> '') AND
(FORM14.Edit3.Text <> '')  THEN
 BEGIN
  form14.ADOQuery3.Close;
  form14.ADOQuery3.SQL.Text:='select * from users where
user_name='+#39+form14.Edit2.Text+#39;
  form14.ADOQuery3.Open;
  if form14.ADOQuery3.RecordCount = 0 then
  begin
   FORM14.ADOQuery3.Close;
   FORM14.ADOQuery3.SQL.Text:='UPDATE users set
user_name='+#39+form14.Edit2.Text+#39+',
password='+#39+form14.Edit3.Text+#39+', staff_id='+#39+form14.Edit1.Text+#39+',
staff_state='+#39+form14.ComboBox1.Text+#39+',
staff_pozition='+#39+form14.ComboBox2.Text+#39+'  where
staff_id='+#39+form14.Edit1.Text+#39;
    form14.ADOQuery3.ExecSQL;
    showmessage('RECORD UPDATED');
    Form14.LbSpeedButton4.Click;
   end
   else
    SHOWMESSAGE('USER NAME IS USED');
 end
 else
  showmessage('PLEASE BE SURE TO FILL EMPTY PLACE');
end;

procedure TForm14.LbSpeedButton3Click(Sender: TObject);
begin
```

```
 IF  (FORM14.Edit1.Text <> '') THEN
 BEGIN
  SS14:=MESSAGEDLG('ARE YOU SURE TO DELETE  '+FORM14.Edit2.Text+'
" ?',MTWARNING,[MBYES ,MBNO],0);
  IF SS14 = MRYES then
  BEGIN
   FORM14.ADOQuery3.Close;
   FORM14.ADOQuery3.SQL.Text:='DELETE FROM users where
user_name='+#39+form14.Edit2.Text+#39;
   form14.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   Form14.LbSpeedButton4.Click;
  END;
 end
 else
  showmessage('PLEASE BE SURE TO CHOOSE DATA THAT YOU WILL
DELETE');
end;

procedure TForm14.LbSpeedButton4Click(Sender: TObject);
begin
 FORM14.Edit1.Clear;
 FORM14.Edit2.Clear;
 FORM14.Edit3.Clear;
 FORM14.ComboBox1.Text:='Select One';
 FORM14.ComboBox2.Text:='Select One';
 form14.Edit2.SetFocus;
 Form14.LbSpeedButton1.Enabled:=TRUE;
 Form14.LbSpeedButton2.Enabled:=TRUE;

 form14.ADOQuery1.Close;
 form14.ADOQuery1.SQL.Text:='select * from users';
 form14.ADOQuery1.Open;
end;

procedure TForm14.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM14.DBGrid1.Fields[0].IsNull = false THEN
 BEGIN
  FORM14.Edit1.Text:=FORM14.DBGrid1.Fields[2].Text;
  FORM14.Edit2.Text:=FORM14.DBGrid1.Fields[0].Text;
  FORM14.Edit3.Text:=FORM14.DBGrid1.Fields[1].Text;
  FORM14.ComboBox1.Text:=FORM14.DBGrid1.Fields[3].Text;
  FORM14.ComboBox2.Text:=FORM14.DBGrid1.Fields[4].Text;
 END
end;

procedure TForm14.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 Form14.LbSpeedButton4.Click;
```

```
  form14.ADOQuery1.Close;
  form14.ADOQuery2.Close;
  form14.ADOQuery3.Close;
end;

procedure TForm14.FormHide(Sender: TObject);
begin
  Form14.LbSpeedButton4.Click;
  form14.ADOQuery1.Close;
  form14.ADOQuery2.Close;
  form14.ADOQuery3.Close;
end;

end.
```

FORM 15 CODES

```
unit Unit15;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, LbSpeedButton, StdCtrls, Buttons, Mask, Menus, ExtCtrls,
  ComCtrls, Grids, DBGrids, DB, ADODB;

type
  TForm15 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    MaskEdit1: TMaskEdit;
    MaskEdit2: TMaskEdit;
```

```
    MaskEdit3: TMaskEdit;
    MaskEdit4: TMaskEdit;
    Memo1: TMemo;
    ComboBox1: TComboBox;
    Edit4: TEdit;
    ComboBox2: TComboBox;
    Edit5: TEdit;
    Edit6: TEdit;
    Memo2: TMemo;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    ADOConnection1: TADOConnection;
    DBGrid1: TDBGrid;
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form15: TForm15;
  SS15:WORD;
implementation

uses Unit10;

{$R *.dfm}

procedure TForm15.FormShow(Sender: TObject);
begin
  FORM15.ADOQuery1.Close;
```

```
 FORM15.ADOQuery1.SQL.Text:='select
customer_id,Cname,Csurname,Mobilphone,homephone,workphone,fax,address,city,tow
n,country,email,web,c_note from customer';
 form15.ADOQuery1.Open;


end;

procedure TForm15.LbSpeedButton1Click(Sender: TObject);
begin
 if (form15.Edit2.Text <> '') or (form15.Edit3.Text <> '') then
 begin
   form15.ADOQuery2.Close;
   form15.ADOQuery2.SQL.Text:='select * from customer where
Cname='+#39+form15.Edit2.Text+#39+' and
Csurname='+#39+form15.Edit3.Text+#39+' and
mobilphone='+#39+form15.MaskEdit2.Text+#39;
   form15.ADOQuery2.Open;
   if form15.ADOQuery2.RecordCount = 0 then
   begin
     form15.ADOQuery2.Close;
     form15.ADOQuery2.SQL.Text:='insert into customer
(Cname,Csurname,homephone,mobilphone,workphone,fax,address,town,city,country,e
mail,web,C_note,recorddate,recordtime) values
('+#39+Form15.Edit2.Text+#39+','+#39+form15.Edit3.Text+#39+','+#39+form15.Mask
Edit1.Text+#39+','+#39+form15.MaskEdit2.Text+#39+','+#39+form15.MaskEdit3.Text
+#39+','+#39+form15.MaskEdit4.Text+#39+','+#39+form15.Memo1.Text+#39+','+#39
+form15.Edit4.Text+#39+','+#39+form15.ComboBox1.Text+#39+','+#39+form15.Com
boBox2.Text+#39+','+#39+form15.Edit5.Text+#39+','+#39+form15.Edit6.Text+#39+','
+#39+form15.Memo2.Text+#39+','+#39+datetostr(date)+#39+','+#39+timetostr(time)+
#39+')';
     form15.ADOQuery2.ExecSQL;
     form15.ADOQuery2.Close;
     form15.ADOQuery2.SQL.Text:='select * from customer';
     form15.ADOQuery2.Open;
     showmessage('RECORD SAVED');
     Form15.LbSpeedButton4.Click;
    END
    else
     SHOWMESSAGE('THE CUSTOMER SAVED BEFORE');
  END
  ELSE if form15.Edit2.Text = '' then
   showmessage('PLEASE FILL THE NAME')
  ELSE if form15.Edit3.Text = '' then
   showmessage('PLEASE FILL THE SURNAME')
  else
   SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');

end;
```

```
procedure TForm15.LbSpeedButton2Click(Sender: TObject);
begin
  IF (FORM15.Edit1.Text <> '') AND (FORM15.Edit2.Text <> '') AND
(FORM15.Edit3.Text <> '') THEN
 BEGIN
   FORM15.ADOQuery3.Close;
   FORM15.ADOQuery3.SQL.Text:='UPDATE customer set Cname=
'+#39+form15.Edit2.Text+#39+', Csurname= '+#39+form15.Edit3.Text+#39+',
homephone='+#39+form15.MaskEdit1.Text+#39+',
mobilphone='+#39+form15.MaskEdit2.Text+#39+ ',
workphone='+#39+form15.MaskEdit3.Text+#39+',
fax='+#39+form15.MaskEdit4.Text+#39+',
address='+#39+form15.Memo1.Text+#39+', town='+#39+form15.Edit4.Text+#39+',
city='+#39+form15.ComboBox1.Text+#39+',
country='+#39+form15.ComboBox2.Text+#39+',
email='+#39+form15.Edit5.Text+#39+', web='+#39+form15.Edit6.Text+#39+',
C_note='+#39+form15.Memo2.Text+#39+' WHERE
customer_id='+#39+form15.Edit1.Text+#39;
   form15.ADOQuery3.ExecSQL;
   showmessage('RECORD UPDATED');
   Form15.LbSpeedButton4.Click;
 END
 ELSE
   SHOWMESSAGE('PLEASE SELECT CUSTOMER AND BE SURE'+#13+'TO
FILL THE EMPTY PLACE');
end;

procedure TForm15.LbSpeedButton3Click(Sender: TObject);
begin
 IF (FORM15.Edit1.Text <> '') AND (FORM15.Edit2.Text <> '') AND
(FORM15.Edit3.Text <> '') then
 BEGIN
   SS15:=MESSAGEDLG('ARE YOU SURE TO DELETE  '' ID:
'+FORM15.Edit1.Text+' ; CUSTOMER: ' +FORM15.Edit2.Text+'
'+FORM15.Edit3.Text+' '' ?',MTWARNING,[MBYES ,MBNO],0);
   IF SS15 = MRYES then
   BEGIN
     FORM15.ADOQuery3.Close;
     form15.ADOQuery3.SQL.Text:='delete from customer where
customer_id='+#39+form15.Edit1.Text+#39;
     form15.ADOQuery3.ExecSQL;
     showmessage('RECORD DELETED');
     Form15.LbSpeedButton4.Click;
   END;
 end
 else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;
```

```
procedure TForm15.LbSpeedButton4Click(Sender: TObject);
begin
  form15.Edit1.Clear;
  form15.Edit2.Clear;
  form15.Edit3.Clear;
  form15.MaskEdit1.Clear;
  form15.MaskEdit2.Clear;
  form15.MaskEdit3.Clear;
  form15.MaskEdit4.Clear;
  form15.Memo1.Clear;
  form15.Edit4.Clear;
  form15.ComboBox1.Text:='Select One';
  form15.ComboBox2.Text:='Select One';
  form15.Edit5.Clear;
  form15.Edit6.Clear;
  form15.Memo2.Clear;
  FORM15.Edit2.SetFocus;
  Form15.LbSpeedButton1.Enabled:=TRUE;
  Form15.LbSpeedButton2.Enabled:=TRUE;

  FORM15.ADOQuery1.Close;
  FORM15.ADOQuery1.SQL.Text:='select
customer_id,Cname,Csurname,Mobilphone,homephone,workphone,fax,address,city,tow
n,country,email,web,c_note from customer';
  form15.ADOQuery1.Open;
end;

procedure TForm15.DBGrid1CellClick(Column: TColumn);
begin
    IF  FORM15.ADOQuery1.RecordCount <> 0 THEN
  BEGIN
    FORM15.Edit1.Text:=FORM15.DBGrid1.Fields[0].Text;
    FORM15.Edit2.Text:=FORM15.DBGrid1.Fields[1].Text;
    FORM15.Edit3.Text:=FORM15.DBGrid1.Fields[2].Text;
    form15.MaskEdit1.Text:=FORM15.DBGrid1.Fields[3].Text;
    form15.MaskEdit2.Text:=FORM15.DBGrid1.Fields[4].Text;
    form15.MaskEdit3.Text:=FORM15.DBGrid1.Fields[5].Text;
    form15.MaskEdit4.Text:=FORM15.DBGrid1.Fields[6].Text;
    FORM15.Memo1.Text:=FORM15.DBGrid1.Fields[7].Text;
    FORM15.Edit4.Text:=FORM15.DBGrid1.Fields[9].Text;
    FORM15.ComboBox1.Text:=FORM15.DBGrid1.Fields[8].Text;
    FORM15.ComboBox2.Text:=FORM15.DBGrid1.Fields[10].Text;
    FORM15.Edit5.Text:=FORM15.DBGrid1.Fields[11].Text;
    FORM15.Edit6.Text:=FORM15.DBGrid1.Fields[12].Text;
    FORM15.Memo2.Text:=FORM15.DBGrid1.Fields[13].Text;
  END;

end;

procedure TForm15.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
  Form15.LbSpeedButton4.Click;
  form15.ADOQuery1.Close;
  form15.ADOQuery2.Close;
  form15.ADOQuery3.Close;
end;

procedure TForm15.FormHide(Sender: TObject);
begin
  Form15.LbSpeedButton4.Click;
  form15.ADOQuery1.Close;
  form15.ADOQuery2.Close;
  form15.ADOQuery3.Close;
end;

end.
```

FORM 16 CODES

```
unit Unit16;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, Grids, DBGrids, ExtCtrls, StdCtrls, Buttons, Menus,
  DB, ADODB;

type
  TForm16 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Edit1: TEdit;
```

```
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    SpeedButton1: TSpeedButton;
    Memo1: TMemo;
    Memo2: TMemo;
    Memo3: TMemo;
    Edit7: TEdit;
    Label17: TLabel;
    Edit8: TEdit;
    Memo4: TMemo;
    ComboBox2: TComboBox;
    Edit9: TEdit;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    StatusBar1: TStatusBar;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    procedure SpeedButton3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure DBGrid1CellClick(Column: TColumn);

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

var
  Form16: TForm16;
  SS16:WORD;
```

```
implementation

uses Unit10, Unit40, Unit15;

{$R *.dfm}


procedure TForm16.SpeedButton3Click(Sender: TObject);
begin
 if (form16.Edit2.Text <> '') and (form16.Edit3.Text <> '') and (form16.Edit5.Text <>
'')then
 begin
   form16.ADOQuery2.Close;
   form16.ADOQuery2.SQL.Text:='select * from animal where
animal_name='+#39+form16.Edit2.Text+#39+' and
animal_kind='+#39+form16.Edit3.Text+#39+' and
owner_no='+#39+form16.Edit5.Text+#39;
   form16.ADOQuery2.Open;
   if form16.ADOQuery2.RecordCount = 0 then
   begin
    form16.ADOQuery2.Close;
    form16.ADOQuery2.SQL.Text:='insert into animal
(animal_name,animal_kind,animal_race,owner_no,abirthdate,animal_sex,animal_color,
animal_weight,collar_no,earning_no,life_state,animal_mark,animal_alergy,acronic_me
dicine,A_note) values
('+#39+Form16.Edit2.Text+#39+','+#39+form16.Edit3.Text+#39+','+#39+form16.Edit4
.Text+#39+','+#39+form16.Edit5.Text+#39+','+#39+datetostr(form16.DateTimePicker1
.Date)+#39+','+#39+form16.ComboBox1.Text+#39+','+#39+form16.Edit6.Text+#39+','
+#39+form16.Edit7.Text+#39+','+#39+form16.Edit8.Text+#39+','+#39+form16.Edit9.
Text+#39+','+#39+form16.ComboBox2.Text+#39+','+#39+form16.Memo1.Text+#39+',
'+#39+form16.Memo2.Text+#39+','+#39+form16.Memo3.Text+#39+','+#39+form16.M
emo4.Text+#39+')';
    form16.ADOQuery2.ExecSQL;
    form16.ADOQuery2.Close;
    form16.ADOQuery2.SQL.Text:='select * from animal';
    form16.ADOQuery2.Open;
    showmessage('RECORD SAVED');
    Form16.SpeedButton6.Click;
   END
   else
    SHOWMESSAGE('THE ANIMAL SAVED BEFORE');
  END
  ELSE if form16.Edit2.Text = '' then
   showmessage('PLEASE FILL THE ANIMAL NAME')
  ELSE if form16.Edit5.Text = '' then
   showmessage('PLEASE CHOOSE THE OWNER NO')
  else
   SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
```

```
end;

procedure TForm16.FormCreate(Sender: TObject);
begin
form16.DateTimePicker1.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';

end;

procedure TForm16.FormShow(Sender: TObject);
begin
 FORM16.ADOQuery1.Close;
 FORM16.ADOQuery1.SQL.Text:='select
animal_id,animal_name,animal_kind,animal_race,owner_no,animal_sex,animal_color,a
nimal_weight,animal_mark,animal_alergy,acronic_medicine,collar_no,earning_no,life_
state,a_note from animal';
 form16.ADOQuery1.Open;
end;

procedure TForm16.SpeedButton4Click(Sender: TObject);
begin
  IF (FORM16.Edit1.Text <> '') AND (FORM16.Edit2.Text <> '') AND
(FORM16.Edit3.Text <> '') AND (FORM16.Edit5.Text <> '') THEN
 BEGIN
  FORM16.ADOQuery3.Close;
  FORM16.ADOQuery3.SQL.Text:='UPDATE animal set animal_name=
'+#39+form16.Edit2.Text+#39+', animal_kind= '+#39+form16.Edit3.Text+#39+',
animal_race= '+#39+form16.Edit4.Text+#39+',
abirthdate='+#39+datetostr(form16.DateTimePicker1.Date)+#39+',
animal_sex='+#39+form16.ComboBox1.Text+#39+', animal_color=
'+#39+form16.Edit6.Text+#39+', animal_weight= '+#39+form16.Edit7.Text+#39+',
collar_no= '+#39+form16.Edit8.Text+#39+', earning_no=
'+#39+form16.Edit9.Text+#39+', life_state='+#39+form16.ComboBox2.Text+#39+',
animal_mark='+#39+form16.Memo1.Text+#39+',
animal_alergy='+#39+form16.Memo2.Text+#39+',
acronic_medicine='+#39+form16.Memo3.Text+#39+',
a_note='+#39+form16.Memo4.Text+#39+' WHERE
animal_id='+#39+form16.Edit1.Text+#39;
  form16.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  Form16.SpeedButton6.Click;
 END
 ELSE
  SHOWMESSAGE('PLEASE SELECT CUSTOMER AND BE SURE'+#13+'TO
FILL THE EMPTY PLACE');
end;

procedure TForm16.SpeedButton5Click(Sender: TObject);
```

```
begin
 IF (FORM16.Edit1.Text <> '') AND (FORM16.Edit2.Text <> '') AND
(FORM16.Edit5.Text <> '') then
 BEGIN
   SS16:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ID:
'+FORM16.Edit1.Text+' ; ANIMAL: ' +FORM16.Edit2.Text+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS16 = MRYES then
   BEGIN
     FORM16.ADOQuery3.Close;
     form16.ADOQuery3.SQL.Text:='delete from animal where
animal_id='+#39+form16.Edit1.Text+#39;
     form16.ADOQuery3.ExecSQL;
     showmessage('RECORD DELETED');
     form16.SpeedButton6.Click;
   END;
 end
 else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm16.SpeedButton6Click(Sender: TObject);
begin
 form16.Edit1.Clear;
 form16.Edit2.Clear;
 form16.Edit3.Clear;
 form16.Edit4.Clear;
 form16.Edit5.Clear;
 form16.DateTimePicker1.Date:=date;
 form16.ComboBox1.Text:='Select One';
 form16.Edit6.Clear;
 form16.Edit7.Clear;
 form16.Edit8.Clear;
 form16.Edit9.Clear;
 form16.ComboBox2.Text:='Select One';
 form16.Memo1.Clear;
 form16.Memo2.Clear;
 form16.Memo3.Clear;
 form16.Memo4.Clear;
 FORM16.Edit2.SetFocus;
 FORM16.SpeedButton3.Enabled:=TRUE;
 FORM16.SpeedButton4.Enabled:=TRUE;

 FORM16.ADOQuery1.Close;
 FORM16.ADOQuery1.SQL.Text:='select
animal_id,animal_name,animal_kind,animal_race,owner_no,animal_sex,animal_color,a
nimal_weight,animal_mark,animal_alergy,acronic_medicine,collar_no,earning_no,life_
state,a_note from animal';
 form16.ADOQuery1.Open;
```

```
end;

procedure TForm16.SpeedButton1Click(Sender: TObject);
begin
FORM40.SHOW;
end;

procedure TForm16.FormHide(Sender: TObject);
begin
 form16.SpeedButton6.Click;
 form16.ADOQuery1.Close;
 form16.ADOQuery2.Close;
 form16.ADOQuery3.Close;
end;

procedure TForm16.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 form16.SpeedButton6.Click;
 form16.ADOQuery1.Close;
 form16.ADOQuery2.Close;
 form16.ADOQuery3.Close;
end;

procedure TForm16.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM16.ADOQuery1.RecordCount <> 0 THEN
 BEGIN
   FORM16.Edit1.Text:=FORM16.DBGrid1.Fields[0].Text;
   FORM16.Edit2.Text:=FORM16.DBGrid1.Fields[1].Text;
   FORM16.Edit3.Text:=FORM16.DBGrid1.Fields[2].Text;
   FORM16.Edit4.Text:=FORM16.DBGrid1.Fields[3].Text;
   FORM16.Edit5.Text:=FORM16.DBGrid1.Fields[4].Text;
   FORM16.ComboBox1.Text:=FORM16.DBGrid1.Fields[5].Text;
   FORM16.Edit6.Text:=FORM16.DBGrid1.Fields[6].Text;
   FORM16.Edit7.Text:=FORM16.DBGrid1.Fields[7].Text;
   FORM16.Memo1.Text:=FORM16.DBGrid1.Fields[8].Text;
   FORM16.Memo2.Text:=FORM16.DBGrid1.Fields[9].Text;
   FORM16.Memo3.Text:=FORM16.DBGrid1.Fields[10].Text;
   FORM16.Edit8.Text:=FORM16.DBGrid1.Fields[11].Text;
   FORM16.Edit9.Text:=FORM16.DBGrid1.Fields[12].Text;
   FORM16.ComboBox2.Text:=FORM16.DBGrid1.Fields[13].Text;
   FORM16.Memo4.Text:=FORM16.DBGrid1.Fields[14].Text;
  end;
end;

end.
```

FORM 17 CODES

```
unit Unit17;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, LbSpeedButton, ExtCtrls, ComCtrls, StdCtrls,
  Buttons, Menus, DB, ADODB;

type
  TForm17 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit1: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    SpeedButton1: TSpeedButton;
    Memo1: TMemo;
    SpeedButton2: TSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    DBGrid1: TDBGrid;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
```

```
  procedure FormCreate(Sender: TObject);
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure DBGrid1CellClick(Column: TColumn);
  procedure LbSpeedButton3Click(Sender: TObject);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form17: TForm17;
 SS17:WORD;
implementation

uses Unit10, Unit38, Unit39;

{$R *.dfm}

procedure TUAH();
begin
 form17.ComboBox1.Items.Clear;
 FORM17.ADOQuery4.Close;
 FORM17.ADOQuery4.SQL.Text:='select vaccine_name from vaccines';
 form17.ADOQuery4.Open;
 while not form17.ADOQuery4.Eof do
 begin
  form17.ComboBox1.Items.Add(form17.ADOQuery4['vaccine_name']);
  form17.ADOQuery4.Next;
 end;
END;

procedure TForm17.SpeedButton1Click(Sender: TObject);
begin
 form38.show;
 TA:=17;
end;

procedure TForm17.FormShow(Sender: TObject);
begin
 form17.ADOQuery1.Close;
 form17.ADOQuery1.SQL.Text:='select * from vaccinate';
 form17.ADOQuery1.Open;
 TUAH();
```

```
end;



procedure TForm17.FormCreate(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 FORM17.DateTimePicker1.Date:=DATE;
 FORM17.DateTimePicker2.Date:=DATE;
end;

procedure TForm17.LbSpeedButton1Click(Sender: TObject);
begin

 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 if (form17.Edit1.Text <> '')  and (form17.ComboBox1.Text <> 'Select One') and
(form17.Edit4.Text <> '') AND (DATETOSTR(FORM17.DateTimePicker1.Date) <>
DATETOSTR(FORM17.DateTimePicker2.Date)) THEN
 begin
  form17.ADOQuery2.Close;
  form17.ADOQuery2.SQL.Text:='select * from vaccinate where
vaccine_serialno='+#39+form17.Edit2.Text+#39;
  form17.ADOQuery2.Open;
  if form17.ADOQuery2.RecordCount = 0 then
  begin
   form17.ADOQuery2.Close;
   form17.ADOQuery2.SQL.Text:='insert into vaccinate
(animal_id,vaccine_name,vaccinate_date,next_vaccinatedate,vaccine_serialno,vaccine_
producer,applied_staff,v_note) values
('+#39+form17.Edit1.Text+#39+','+#39+form17.ComboBox1.Text+#39+','+#39+dateto
str(form17.DateTimePicker1.Date)+#39+','+#39+datetostr(form17.DateTimePicker2.Da
te)+#39+','+#39+form17.Edit2.Text+#39+','+#39+form17.Edit3.Text+#39+','+#39+for
m17.Edit4.Text+#39+','+#39+form17.Memo1.Text+#39+')';
   form17.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form17.ADOQuery1.Close;
   form17.ADOQuery1.SQL.Text:='select * from vaccinate where
animal_id='+#39+form17.Edit1.Text+#39;
   form17.ADOQuery1.Open;
   TUAH();
  END
  ELSE
   SHOWMESSAGE('THE VACCINATE SAVED BEFORE');
 END
 ELSE if form17.Edit1.Text = '' then
```

```
   showmessage('PLEASE CHOOSE THE ANIMAL ID')
  ELSE if form17.Edit4.Text = '' then
   showmessage('PLEASE CHOOSE THE STAFF ID')
  else
   SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm17.SpeedButton2Click(Sender: TObject);
begin
 FORM39.SHOW;
 ANI:=17;
end;

procedure TForm17.Edit1Change(Sender: TObject);
begin
 form17.ADOQuery1.Close;
 form17.ADOQuery1.SQL.Text:='select * from vaccinate where
animal_id='+#39+form17.Edit1.Text+#39;
 form17.ADOQuery1.Open;
 TUAH();
end;

procedure TForm17.LbSpeedButton2Click(Sender: TObject);
begin

 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 IF (FORM17.Edit1.Text <> '') AND (FORM17.Edit2.Text <> '') AND
(FORM17.ComboBox1.Text <> 'Select One') THEN
 BEGIN
   FORM17.ADOQuery3.Close;
   FORM17.ADOQuery3.SQL.Text:='UPDATE vaccinate set
Animal_id='+#39+form17.Edit1.Text+#39+',
Vaccine_name='+#39+form17.ComboBox1.Text+#39+',
Vaccinate_date='+#39+datetostr(form17.DateTimePicker1.Date)+#39+',
Next_vaccinatedate='+#39+datetostr(form17.DateTimePicker2.Date)+#39+',
Vaccine_serialno='+#39+form17.Edit2.Text+#39+',
Vaccine_producer='+#39+form17.Edit3.Text+#39+',
Applied_staff='+#39+form17.Edit4.Text+#39+',
V_note='+#39+form17.Memo1.Text+#39+' where
Animal_id='+#39+form17.DBGrid1.Fields[0].Text+#39+' and
Vaccine_name='+#39+form17.DBGrid1.Fields[1].Text+#39+' and
Vaccinate_date='+#39+form17.DBGrid1.Fields[2].Text+#39+' and
Vaccine_serialno='+#39+form17.DBGrid1.Fields[4].Text+#39;
   form17.ADOQuery3.ExecSQL;
   showmessage('RECORD UPDATED');
   FORM17.LbSpeedButton4.Click;
 END
```

```
 ELSE
   SHOWMESSAGE('PLEASE CHOOSE VACCINATE FROM LIST');
end;

procedure TForm17.DBGrid1CellClick(Column: TColumn);
begin
  IF  FORM17.ADOQuery1.RecordCount <> 0 THEN
  BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM17.Edit1.Text:=FORM17.DBGrid1.Fields[0].Text;
   FORM17.ComboBox1.Text:=FORM17.DBGrid1.Fields[1].Text;

FORM17.DateTimePicker1.Date:=STRTODATE(FORM17.DBGrid1.Fields[2].Text);

FORM17.DateTimePicker2.Date:=STRTODATE(FORM17.DBGrid1.Fields[3].Text);
   FORM17.Edit2.Text:=FORM17.DBGrid1.Fields[4].Text;
   FORM17.Edit3.Text:=FORM17.DBGrid1.Fields[5].Text;
   FORM17.Edit4.Text:=FORM17.DBGrid1.Fields[6].Text;
   FORM17.Memo1.Text:=FORM17.DBGrid1.Fields[7].Text;
  END;

end;

procedure TForm17.LbSpeedButton3Click(Sender: TObject);
begin
  IF (FORM17.Edit1.Text <> '') AND (FORM17.combobox1.Text <> 'Select One')
AND (FORM17.Edit4.Text <> '') then
  BEGIN
   SS17:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM17.Edit1.Text+' ; VACCINE: ' +FORM17.COMBOBOX1.Text+'  ;
VACCINATE DATE: '+DATETOSTR(FORM17.DateTimePicker1.Date)+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS17 = MRYES then
   BEGIN
    FORM17.ADOQuery3.Close;
    form17.ADOQuery3.SQL.Text:='delete from vaccinate where
animal_id='+#39+form17.Edit1.Text+#39+' and
vaccine_name='+#39+form17.ComboBox1.Text+#39;
    //FORM17.ADOQuery3.SQL.Text:='DELETE FROM vaccinate where
Animal_id='+#39+form17.Edit1.Text+#39+' and
Vaccine_name='+#39+form17.ComboBox1.Text+#39+' and
Vaccinate_date='+#39+datetostr(form17.DateTimePicker1.Date)+#39+' and
Vaccine_serialno='+#39+form17.Edit2.Text+#39;
    form17.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form17.LbSpeedButton4.Click;
```

```
  END;
 end
 else
  showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm17.LbSpeedButton4Click(Sender: TObject);
begin
 FORM17.Edit1.Clear;
 FORM17.ComboBox1.Text:='Select One';
 form17.DateTimePicker1.Date:=date;
 form17.DateTimePicker2.Date:=date;
 FORM17.Edit2.Clear;
 FORM17.Edit3.Clear;
 FORM17.Edit4.Clear;
 FORM17.Memo1.Clear;
 form17.ComboBox1.SetFocus;
 Form17.LbSpeedButton1.Enabled:=TRUE;
 Form17.LbSpeedButton2.Enabled:=TRUE;

 form17.ADOQuery1.Close;
 form17.ADOQuery1.SQL.Text:='select * from vaccinate';
 form17.ADOQuery1.Open;
end;

procedure TForm17.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 form17.LbSpeedButton4.Click;
 form17.ADOQuery1.Close;
 form17.ADOQuery2.Close;
 form17.ADOQuery3.Close;
 form17.ADOQuery4.Close;
end;

procedure TForm17.FormHide(Sender: TObject);
begin
 form17.LbSpeedButton4.Click;
 form17.ADOQuery1.Close;
 form17.ADOQuery2.Close;
 form17.ADOQuery3.Close;
 form17.ADOQuery4.Close;
end;

end.
```

FORM 18 CODES

```
unit Unit18;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, ExtCtrls, ComCtrls, Buttons, StdCtrls, Menus,
  DB, ADODB;

type
  TForm18 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit1: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Edit2: TEdit;
    Memo1: TMemo;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    DBGrid1: TDBGrid;
    Label7: TLabel;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    Edit3: TEdit;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
```

```
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton6Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form18: TForm18;
  SS18:WORD;
implementation

uses Unit10, Unit39, Unit38;

{$R *.dfm}

PROCEDURE TUAHIN();
BEGIN
  form18.ComboBox1.Items.Clear;
  FORM18.ADOQuery4.Close;
  FORM18.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'INNER PARASITE'+#39;
  form18.ADOQuery4.Open;
  while not form18.ADOQuery4.Eof do
  begin
    form18.ComboBox1.Items.Add(form18.ADOQuery4['drug_name']);
    form18.ADOQuery4.Next;
  end;
END;

procedure TForm18.FormShow(Sender: TObject);
begin
  form18.ADOQuery1.Close;
  form18.ADOQuery1.SQL.Text:='select * from ipdrug';
  form18.ADOQuery1.Open;
  TUAHIN();
end;

procedure TForm18.SpeedButton3Click(Sender: TObject);
begin
   dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'yyyy/mm/dd';
```

```
 if (form18.Edit1.Text <> '')  and (form18.ComboBox1.Text <> 'Select One') and
(form18.Edit2.Text <> '') AND (DATETOSTR(FORM18.DateTimePicker1.Date) <>
DATETOSTR(FORM18.DateTimePicker2.Date)) THEN
 begin
  form18.ADOQuery2.Close;
  form18.ADOQuery2.SQL.Text:='select * from ipdrug where
animal_id='+#39+form18.Edit1.Text+#39+'and
ip_drugname='+#39+form18.ComboBox1.Text+#39+' and
ip_drugdate='+#39+datetostr(form18.DateTimePicker1.Date)+#39+' and
ip_nextdrugdate='+#39+datetostr(form18.DateTimePicker2.Date)+#39;
  form18.ADOQuery2.Open;
  if form18.ADOQuery2.RecordCount = 0 then
  begin
   form18.ADOQuery2.Close;
   form18.ADOQuery2.SQL.Text:='insert into ipdrug
(animal_id,ip_drugname,ip_drugdate,ip_nextdrugdate,applied_staff,ip_drugnote) values
('+#39+form18.Edit1.Text+#39+','+#39+form18.ComboBox1.Text+#39+','+#39+dateto
str(form18.DateTimePicker1.Date)+#39+','+#39+datetostr(form18.DateTimePicker2.Da
te)+#39+','+#39+form18.Edit2.Text+#39+','+#39+form18.Memo1.Text+#39+')';
   form18.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form18.ADOQuery1.Close;
   form18.ADOQuery1.SQL.Text:='select * from ipdrug where
animal_id='+#39+form18.Edit1.Text+#39;
   form18.ADOQuery1.Open;
   TUAHIN();
  END
  ELSE
   SHOWMESSAGE('THE INNER PARASITE APPLICATION SAVED BEFORE');
 END
 ELSE if form18.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE ANIMAL ID')
 ELSE if form18.Edit2.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm18.SpeedButton1Click(Sender: TObject);
begin
form39.show;
ANI:=18;
end;

procedure TForm18.SpeedButton2Click(Sender: TObject);
begin
 form38.show;
 TA:=18;
end;
```

```
procedure TForm18.SpeedButton4Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

  IF (FORM18.Edit1.Text <> '') AND (FORM18.Edit3.Text <> '') AND
(FORM18.Edit2.Text <> '') AND (FORM18.ComboBox1.Text <> 'Select One') THEN
  BEGIN
    FORM18.ADOQuery3.Close;
    FORM18.ADOQuery3.SQL.Text:='UPDATE ipdrug set
Animal_id='+#39+form18.Edit1.Text+#39+',
ip_drugname='+#39+form18.ComboBox1.Text+#39+',
ip_drugdate='+#39+datetostr(form18.DateTimePicker1.Date)+#39+',
ip_nextdrugdate='+#39+datetostr(form18.DateTimePicker2.Date)+#39+',
Applied_staff='+#39+form18.Edit2.Text+#39+',
ip_drugnote='+#39+form18.Memo1.Text+#39+' where
Ip_id='+#39+form18.Edit3.Text+#39;
    form18.ADOQuery3.ExecSQL;
    showmessage('RECORD UPDATED');
    FORM18.SpeedButton6.Click;
  END
  ELSE
    SHOWMESSAGE('PLEASE SELECT INNER PARASITE APPLICATION FROM
LIST');
end;

procedure TForm18.SpeedButton5Click(Sender: TObject);
begin
  IF (FORM18.Edit1.Text <> '') AND (FORM18.Edit3.Text <> '') then
  BEGIN
    SS18:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM18.Edit1.Text+' ; INNER DRUG: ' +FORM18.COMBOBOX1.Text+'  ; IP
DRUG DATE: '+DATETOSTR(FORM18.DateTimePicker1.Date)+' "
?',MTWARNING,[MBYES ,MBNO],0);
    IF SS18 = MRYES then
    BEGIN
      FORM18.ADOQuery3.Close;
      form18.ADOQuery3.SQL.Text:='delete from ipdrug where
Ip_id='+#39+form18.Edit3.Text+#39;
      form18.ADOQuery3.ExecSQL;
      showmessage('RECORD DELETED');
      Form18.SpeedButton6.Click;
    END;
  end
  else
    showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;
```

```
procedure TForm18.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM18.ADOQuery1.RecordCount <> 0 THEN
 BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM18.Edit3.Text:=FORM18.DBGrid1.Fields[0].Text;
   FORM18.Edit1.Text:=FORM18.DBGrid1.Fields[1].Text;
   FORM18.ComboBox1.Text:=FORM18.DBGrid1.Fields[2].Text;

FORM18.DateTimePicker1.Date:=STRTODATE(FORM18.DBGrid1.Fields[3].Text);

FORM18.DateTimePicker2.Date:=STRTODATE(FORM18.DBGrid1.Fields[4].Text);
   FORM18.Edit2.Text:=FORM18.DBGrid1.Fields[5].Text;
   FORM18.Memo1.Text:=FORM18.DBGrid1.Fields[6].Text;
 END;
end;

procedure TForm18.SpeedButton6Click(Sender: TObject);
begin
 FORM18.Edit3.Clear;
 FORM18.Edit1.Clear;
 FORM18.ComboBox1.Text:='Select One';
 form18.DateTimePicker1.Date:=date;
 form18.DateTimePicker2.Date:=date;
 FORM18.Edit2.Clear;
 FORM18.Memo1.Clear;
 form18.ComboBox1.SetFocus;
 Form18.SpeedButton3.Enabled:=TRUE;
 Form18.SpeedButton4.Enabled:=TRUE;

 form18.ADOQuery1.Close;
 form18.ADOQuery1.SQL.Text:='select * from ipdrug';
 form18.ADOQuery1.Open;
end;

procedure TForm18.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 FORM18.SpeedButton6.Click;
 FORM18.ADOQuery1.Close;
 FORM18.ADOQuery2.Close;
 FORM18.ADOQuery3.Close;
 FORM18.ADOQuery4.Close;
end;

procedure TForm18.FormHide(Sender: TObject);
```

```
begin
  FORM18.SpeedButton6.Click;
  FORM18.ADOQuery1.Close;
  FORM18.ADOQuery2.Close;
  FORM18.ADOQuery3.Close;
  FORM18.ADOQuery4.Close;
end;

procedure TForm18.FormCreate(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';
  FORM18.DateTimePicker1.Date:=DATE;
  FORM18.DateTimePicker2.Date:=DATE;
end;

procedure TForm18.Edit1Change(Sender: TObject);
begin
  form18.ADOQuery1.Close;
  form18.ADOQuery1.SQL.Text:='select * from ipdrug where
animal_id='+#39+form18.Edit1.Text+#39;
  form18.ADOQuery1.Open;
  TUAHIN();
end;

end.
```

FORM 19 CODES

```
unit Unit19;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Menus, Grids, DBGrids, LbSpeedButton,
  ExtCtrls, Buttons, DB, ADODB;

type
  TForm19 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
```

```
  StatusBar1: TStatusBar;
  Edit1: TEdit;
  ComboBox1: TComboBox;
  DateTimePicker1: TDateTimePicker;
  DateTimePicker2: TDateTimePicker;
  Edit2: TEdit;
  Memo1: TMemo;
  SpeedButton1: TSpeedButton;
  SpeedButton2: TSpeedButton;
  Panel1: TPanel;
  LbSpeedButton1: TLbSpeedButton;
  LbSpeedButton2: TLbSpeedButton;
  LbSpeedButton3: TLbSpeedButton;
  LbSpeedButton4: TLbSpeedButton;
  DBGrid1: TDBGrid;
  Label7: TLabel;
  Edit3: TEdit;
  ADOQuery1: TADOQuery;
  ADOQuery2: TADOQuery;
  ADOQuery3: TADOQuery;
  ADOQuery4: TADOQuery;
  DataSource1: TDataSource;
  DataSource2: TDataSource;
  DataSource3: TDataSource;
  DataSource4: TDataSource;
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure LbSpeedButton3Click(Sender: TObject);
  procedure DBGrid1CellClick(Column: TColumn);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form19: TForm19;
 SS19:WORD;
implementation

uses Unit10, Unit18, Unit39, Unit38;
```

```
{$R *.dfm}

PROCEDURE TUAHOUT();
BEGIN
 form19.ComboBox1.Items.Clear;
 FORM19.ADOQuery4.Close;
 FORM19.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'OUTER PARASITE'+#39;
 form19.ADOQuery4.Open;
 while not form19.ADOQuery4.Eof do
 begin
  form19.ComboBox1.Items.Add(form19.ADOQuery4['drug_name']);
  form19.ADOQuery4.Next;
 end;
END;


procedure TForm19.LbSpeedButton1Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

 if (form19.Edit1.Text <> '')  and (form19.ComboBox1.Text <> 'Select One') and
(form19.Edit2.Text <> '') AND (DATETOSTR(FORM19.DateTimePicker1.Date) <>
DATETOSTR(FORM19.DateTimePicker2.Date)) THEN
 begin
  form19.ADOQuery2.Close;
  form19.ADOQuery2.SQL.Text:='select * from opdrug where
animal_id='+#39+form19.Edit1.Text+#39+'and
op_drugname='+#39+form19.ComboBox1.Text+#39+' and
op_drugdate='+#39+datetostr(form19.DateTimePicker1.Date)+#39+' and
op_nextdrugdate='+#39+datetostr(form19.DateTimePicker2.Date)+#39;
  form19.ADOQuery2.Open;
  if form19.ADOQuery2.RecordCount = 0 then
  begin
   form19.ADOQuery2.Close;
   form19.ADOQuery2.SQL.Text:='insert into opdrug
(animal_id,op_drugname,op_drugdate,op_nextdrugdate,applied_staff,op_drugnote)
values
('+#39+form19.Edit1.Text+#39+','+#39+form19.ComboBox1.Text+#39+','+#39+dateto
str(form19.DateTimePicker1.Date)+#39+','+#39+datetostr(form19.DateTimePicker2.Da
te)+#39+','+#39+form19.Edit2.Text+#39+','+#39+form19.Memo1.Text+#39+')';
   form19.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form19.ADOQuery1.Close;
   form19.ADOQuery1.SQL.Text:='select * from opdrug where
animal_id='+#39+form19.Edit1.Text+#39;
   form19.ADOQuery1.Open;
   TUAHOUT();
```

```
  END
  ELSE
   SHOWMESSAGE('THE INNER PARASITE APPLICATION SAVED BEFORE');
 END
 ELSE if form19.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE ANIMAL ID')
 ELSE if form19.Edit2.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm19.FormShow(Sender: TObject);
begin
 form19.ADOQuery1.Close;
 form19.ADOQuery1.SQL.Text:='select * from opdrug';
 form19.ADOQuery1.Open;
 TUAHOUT();
end;

procedure TForm19.LbSpeedButton2Click(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 IF (FORM19.Edit1.Text <> '') AND (FORM19.Edit3.Text <> '') AND
(FORM19.Edit2.Text <> '') AND (FORM19.ComboBox1.Text <> 'Select One') THEN
 BEGIN
  FORM19.ADOQuery3.Close;
  FORM19.ADOQuery3.SQL.Text:='UPDATE opdrug set
Animal_id='+#39+form19.Edit1.Text+#39+',
op_drugname='+#39+form19.ComboBox1.Text+#39+',
op_drugdate='+#39+datetostr(form19.DateTimePicker1.Date)+#39+',
op_nextdrugdate='+#39+datetostr(form19.DateTimePicker2.Date)+#39+',
Applied_staff='+#39+form19.Edit2.Text+#39+',
op_drugnote='+#39+form19.Memo1.Text+#39+' where
Op_id='+#39+form19.Edit3.Text+#39;
  form19.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  Form19.LbSpeedButton4.Click;
 END
 ELSE
  SHOWMESSAGE('PLEASE SELECT OUTER PARASITE APPLICATION FROM
LIST');
end;

procedure TForm19.LbSpeedButton3Click(Sender: TObject);
begin
 IF (FORM19.Edit1.Text <> '') AND (FORM19.Edit3.Text <> '') then
```

```
 BEGIN
   SS19:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM19.Edit1.Text+' ; OUTER DRUG: ' +FORM19.COMBOBOX1.Text+'  ; OP
DRUG DATE: '+DATETOSTR(FORM19.DateTimePicker1.Date)+' "
?',MTWARNING,[MBYES ,MBNO],0);
   IF SS19 = MRYES then
   BEGIN
    FORM19.ADOQuery3.Close;
    form19.ADOQuery3.SQL.Text:='delete from opdrug where
Op_id='+#39+form19.Edit3.Text+#39;
    form19.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form19.LbSpeedButton4.Click;
   END;
 end
 else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm19.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM19.ADOQuery1.RecordCount <> 0 THEN
 BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM19.Edit3.Text:=FORM19.DBGrid1.Fields[0].Text;
   FORM19.Edit1.Text:=FORM19.DBGrid1.Fields[1].Text;
   FORM19.ComboBox1.Text:=FORM19.DBGrid1.Fields[2].Text;

FORM19.DateTimePicker1.Date:=STRTODATE(FORM19.DBGrid1.Fields[3].Text);

FORM19.DateTimePicker2.Date:=STRTODATE(FORM19.DBGrid1.Fields[4].Text);
   FORM19.Edit2.Text:=FORM19.DBGrid1.Fields[5].Text;
   FORM19.Memo1.Text:=FORM19.DBGrid1.Fields[6].Text;
 END;
end;

procedure TForm19.LbSpeedButton4Click(Sender: TObject);
begin
 FORM19.Edit3.Clear;
 FORM19.Edit1.Clear;
 FORM19.ComboBox1.Text:='Select One';
 form19.DateTimePicker1.Date:=date;
 form19.DateTimePicker2.Date:=date;
 FORM19.Edit2.Clear;
 FORM19.Memo1.Clear;
```

```
 form19.ComboBox1.SetFocus;
 Form19.LbSpeedButton1.Enabled:=TRUE;
 Form19.LbSpeedButton2.Enabled:=TRUE;

 form19.ADOQuery1.Close;
 form19.ADOQuery1.SQL.Text:='select * from opdrug';
 form19.ADOQuery1.Open;
end;

procedure TForm19.SpeedButton1Click(Sender: TObject);
begin
 form39.show;
 ANI:=19;
end;

procedure TForm19.SpeedButton2Click(Sender: TObject);
begin
 form38.show;
 TA:=19;
end;

procedure TForm19.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 FORM19.LbSpeedButton4.Click;
 FORM19.ADOQuery1.Close;
 FORM19.ADOQuery2.Close;
 FORM19.ADOQuery3.Close;
 FORM19.ADOQuery4.Close;
end;

procedure TForm19.FormHide(Sender: TObject);
begin
 FORM19.LbSpeedButton4.Click;
 FORM19.ADOQuery1.Close;
 FORM19.ADOQuery2.Close;
 FORM19.ADOQuery3.Close;
 FORM19.ADOQuery4.Close;
end;

procedure TForm19.FormCreate(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 FORM19.DateTimePicker1.Date:=DATE;
 FORM19.DateTimePicker2.Date:=DATE
end;

procedure TForm19.Edit1Change(Sender: TObject);
begin
```

```
 form19.ADOQuery1.Close;
 form19.ADOQuery1.SQL.Text:='select * from opdrug where
animal_id='+#39+form19.Edit1.Text+#39;
 form19.ADOQuery1.Open;
 TUAHOUT();
end;

end.
```

FORM 20 CODES

```
unit Unit20;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, Grids, DBGrids, ComCtrls, ExtCtrls, Buttons, StdCtrls, Menus,
 DB, ADODB;

type
 TForm20 = class(TForm)
   MainMenu1: TMainMenu;
   F1: TMenuItem;
   Label1: TLabel;
   Label2: TLabel;
   Label3: TLabel;
   Label4: TLabel;
   Label5: TLabel;
   Edit1: TEdit;
   ComboBox1: TComboBox;
   DateTimePicker1: TDateTimePicker;
   Edit2: TEdit;
   Memo1: TMemo;
   SpeedButton1: TSpeedButton;
   SpeedButton2: TSpeedButton;
   SpeedButton3: TSpeedButton;
   SpeedButton4: TSpeedButton;
   SpeedButton5: TSpeedButton;
   SpeedButton6: TSpeedButton;
   Panel1: TPanel;
   StatusBar1: TStatusBar;
   DBGrid1: TDBGrid;
   Label6: TLabel;
   Edit3: TEdit;
   ADOQuery1: TADOQuery;
   ADOQuery2: TADOQuery;
   ADOQuery3: TADOQuery;
   ADOQuery4: TADOQuery;
```

```
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    procedure SpeedButton3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form20: TForm20;
  SS20:WORD;
implementation

uses Unit10, Unit18, Unit39, Unit38;

{$R *.dfm}

PROCEDURE TUAHMED();
BEGIN
  form20.ComboBox1.Items.Clear;
  FORM20.ADOQuery4.Close;
  FORM20.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'GENERAL DRUG'+#39;
  form20.ADOQuery4.Open;
  while not form20.ADOQuery4.Eof do
  begin
    form20.ComboBox1.Items.Add(form20.ADOQuery4['drug_name']);
    form20.ADOQuery4.Next;
  end;
END;

procedure TForm20.SpeedButton3Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';
```

```
 if (form20.Edit1.Text <> '')  and (form20.ComboBox1.Text <> 'Select One') and
(form20.Edit2.Text <> '') THEN
 begin
  form20.ADOQuery2.Close;
  form20.ADOQuery2.SQL.Text:='select * from medicinate where
animal_id='+#39+form20.Edit1.Text+#39+'and
drug_name='+#39+form20.ComboBox1.Text+#39+' and
medicinate_date='+#39+datetostr(form20.DateTimePicker1.Date)+#39;
  form20.ADOQuery2.Open;
  if form20.ADOQuery2.RecordCount = 0 then
  begin
   form20.ADOQuery2.Close;
   form20.ADOQuery2.SQL.Text:='insert into medicinate
(animal_id,drug_name,medicinate_date,applied_staff,M_note) values
('+#39+form20.Edit1.Text+#39+','+#39+form20.ComboBox1.Text+#39+','+#39+dateto
str(form20.DateTimePicker1.Date)+#39+','+#39+form20.Edit2.Text+#39+','+#39+form
20.Memo1.Text+#39+')';
   form20.ADOQuery2.ExecSQL;
   showmessage('RECORD SAVED');
   form20.ADOQuery1.Close;
   form20.ADOQuery1.SQL.Text:='select * from medicinate where
animal_id='+#39+form20.Edit1.Text+#39;
   form20.ADOQuery1.Open;
   TUAHMED();
  END
  ELSE
   SHOWMESSAGE('THE MEDICINATE APPLICATION SAVED BEFORE');
 END
 ELSE if form20.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE ANIMAL ID')
 ELSE if form20.Edit2.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm20.FormCreate(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 FORM20.DateTimePicker1.Date:=DATE;
end;

procedure TForm20.SpeedButton4Click(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
```

```
 IF (FORM20.Edit1.Text <> '') AND (FORM20.Edit3.Text <> '') AND
(FORM20.Edit2.Text <> '') AND (FORM20.ComboBox1.Text <> 'Select One') THEN
 BEGIN
  FORM20.ADOQuery3.Close;
  FORM20.ADOQuery3.SQL.Text:='UPDATE medicinate set
Animal_id='+#39+form20.Edit1.Text+#39+',
drug_name='+#39+form20.ComboBox1.Text+#39+',
medicinate_date='+#39+datetostr(form20.DateTimePicker1.Date)+#39+',
Applied_staff='+#39+form20.Edit2.Text+#39+',
M_note='+#39+form20.Memo1.Text+#39+' where
medicinate_id='+#39+form20.Edit3.Text+#39;
  form20.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  Form20.SpeedButton6.Click;
 END
 ELSE
  SHOWMESSAGE('PLEASE SELECT MEDICINATE APPLICATION FROM
LIST');
end;

procedure TForm20.SpeedButton5Click(Sender: TObject);
begin
  IF (FORM20.Edit1.Text <> '') AND (FORM20.Edit3.Text <> '') then
 BEGIN
  SS20:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM20.Edit1.Text+' ; MEDICINE: '
+FORM20.COMBOBOX1.Text+#13+'MEDICINATE DATE:
'+DATETOSTR(FORM20.DateTimePicker1.Date)+' " ?',MTWARNING,[MBYES
,MBNO],0);
  IF SS20 = MRYES then
  BEGIN
    FORM20.ADOQuery3.Close;
    form20.ADOQuery3.SQL.Text:='delete from medicinate where
medicinate_id='+#39+form20.Edit3.Text+#39;
    form20.ADOQuery3.ExecSQL;
    showmessage('RECORD DELETED');
    Form20.SpeedButton6.Click;
  END;
 end
 else
  showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm20.SpeedButton6Click(Sender: TObject);
begin
 FORM20.Edit3.Clear;
 FORM20.Edit1.Clear;
 FORM20.ComboBox1.Text:='Select One';
```

```
 form20.DateTimePicker1.Date:=date;
 FORM20.Edit2.Clear;
 FORM20.Memo1.Clear;
 form20.ComboBox1.SetFocus;
 Form20.SpeedButton3.Enabled:=TRUE;
 Form20.SpeedButton4.Enabled:=TRUE;

 form20.ADOQuery1.Close;
 form20.ADOQuery1.SQL.Text:='select * from medicinate';
 form20.ADOQuery1.Open;
 TUAHMED();
end;

procedure TForm20.DBGrid1CellClick(Column: TColumn);
begin
  IF  FORM20.ADOQuery1.RecordCount <> 0 THEN
  BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM20.Edit3.Text:=FORM20.DBGrid1.Fields[0].Text;
   FORM20.Edit1.Text:=FORM20.DBGrid1.Fields[1].Text;
   FORM20.ComboBox1.Text:=FORM20.DBGrid1.Fields[2].Text;

FORM20.DateTimePicker1.Date:=STRTODATE(FORM20.DBGrid1.Fields[3].Text);
   FORM20.Edit2.Text:=FORM20.DBGrid1.Fields[4].Text;
   FORM20.Memo1.Text:=FORM20.DBGrid1.Fields[5].Text;
  END;
end;

procedure TForm20.SpeedButton1Click(Sender: TObject);
begin
 FORM39.SHOW;
 ANI:=20;
end;

procedure TForm20.SpeedButton2Click(Sender: TObject);
begin
 FORM38.SHOW;
 TA:=20;
end;

procedure TForm20.FormShow(Sender: TObject);
begin
 form20.ADOQuery1.Close;
 form20.ADOQuery1.SQL.Text:='select * from medicinate';
 form20.ADOQuery1.Open;
 TUAHMED();
```

```
end;

procedure TForm20.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 Form20.SpeedButton6.Click;
 FORM20.ADOQuery1.Close;
 FORM20.ADOQuery2.Close;
 FORM20.ADOQuery3.Close;
 FORM20.ADOQuery4.Close;
end;

procedure TForm20.FormHide(Sender: TObject);
begin
 Form20.SpeedButton6.Click;
 FORM20.ADOQuery1.Close;
 FORM20.ADOQuery2.Close;
 FORM20.ADOQuery3.Close;
 FORM20.ADOQuery4.Close;
end;

procedure TForm20.Edit1Change(Sender: TObject);
begin
 form20.ADOQuery1.Close;
 form20.ADOQuery1.SQL.Text:='select * from medicinate where
animal_id='+#39+form20.Edit1.Text+#39;
 form20.ADOQuery1.Open;
 TUAHMED();
end;

end.
```

FORM 21 CODES

```
unit Unit21;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, Grids, DBGrids, ExtCtrls, ComCtrls, LbSpeedButton, Buttons,
 StdCtrls, Menus, DB, ADODB;

type
 TForm21 = class(TForm)
  MainMenu1: TMainMenu;
  F1: TMenuItem;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
```

```
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit1: TEdit;
    ComboBox1: TComboBox;
    DateTimePicker1: TDateTimePicker;
    ComboBox2: TComboBox;
    Edit2: TEdit;
    Memo1: TMemo;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    Edit3: TEdit;
    Label7: TLabel;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    ADOQuery5: TADOQuery;
    DataSource5: TDataSource;
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form21: TForm21;
```

```
  SS21:WORD;
implementation

uses Unit10, Unit20, Unit39, Unit38;

{$R *.dfm}

PROCEDURE TUAHAPPDRUG();
BEGIN
  form21.ComboBox2.Items.Clear;
  FORM21.ADOQuery4.Close;
  FORM21.ADOQuery4.SQL.Text:='select drug_name from drugs where
drug_kind='+#39+'GENERAL DRUG'+#39;
  form21.ADOQuery4.Open;
  while not form21.ADOQuery4.Eof do
  begin
    form21.ComboBox2.Items.Add(form21.ADOQuery4['drug_name']);
    form21.ADOQuery4.Next;
  end;
END;

PROCEDURE TUAHAPPOPR();
BEGIN
  form21.ComboBox1.Items.Clear;
  FORM21.ADOQuery5.Close;
  FORM21.ADOQuery5.SQL.Text:='select operation_name from operations';
  form21.ADOQuery5.Open;
  while not form21.ADOQuery5.Eof do
  begin
    form21.ComboBox1.Items.Add(form21.ADOQuery5['operation_name']);
    form21.ADOQuery5.Next;
  end;
END;



procedure TForm21.FormCreate(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';
  FORM21.DateTimePicker1.Date:=DATE;
end;

procedure TForm21.FormShow(Sender: TObject);
begin
  form21.ADOQuery1.Close;
  form21.ADOQuery1.SQL.Text:='select * from appliedoperation';
  form21.ADOQuery1.Open;
  TUAHAPPOPR();
```

```
  TUAHAPPDRUG();
end;

procedure TForm21.LbSpeedButton1Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

  if (form21.Edit1.Text <> '')  and (form21.ComboBox1.Text <> 'Select One') and
(form21.Edit2.Text <> '') THEN
  begin
    form21.ADOQuery2.Close;
    form21.ADOQuery2.SQL.Text:='select * from appliedoperation where
animal_id='+#39+form21.Edit1.Text+#39+'and
operation_name='+#39+form21.ComboBox1.Text+#39+' and
operation_date='+#39+datetostr(form21.DateTimePicker1.Date)+#39;
    form21.ADOQuery2.Open;
    if form21.ADOQuery2.RecordCount = 0 then
    begin
      form21.ADOQuery2.Close;
      form21.ADOQuery2.SQL.Text:='insert into appliedoperation
(animal_id,operation_name,operation_date,drug_name,applied_staff,O_note) values
('+#39+form21.Edit1.Text+#39+','+#39+form21.ComboBox1.Text+#39+','+#39+dateto
str(form21.DateTimePicker1.Date)+#39+','+#39+form21.ComboBox2.Text+#39+','+#3
9+form21.Edit2.Text+#39+','+#39+form21.Memo1.Text+#39+')';
      form21.ADOQuery2.ExecSQL;
      showmessage('RECORD SAVED');
      form21.ADOQuery1.Close;
      form21.ADOQuery1.SQL.Text:='select * from appliedoperation where
animal_id='+#39+form21.Edit1.Text+#39;
      form21.ADOQuery1.Open;
      TUAHAPPOPR();
      TUAHAPPDRUG();
    END
    ELSE
      SHOWMESSAGE('THE OPERATION SAVED BEFORE');
  END
  ELSE if form21.Edit1.Text = '' then
    showmessage('PLEASE CHOOSE THE ANIMAL ID')
  ELSE if form21.Edit2.Text = '' then
    showmessage('PLEASE CHOOSE THE STAFF ID')
  else
    SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');
end;

procedure TForm21.LbSpeedButton2Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
```

```
 shortdateformat := 'yyyy/mm/dd';

 IF (FORM21.Edit1.Text <> '') AND (FORM21.Edit3.Text <> '') AND
(FORM21.Edit2.Text <> '') AND (FORM21.ComboBox1.Text <> 'Select One') THEN
 BEGIN
  FORM21.ADOQuery3.Close;
  FORM21.ADOQuery3.SQL.Text:='UPDATE appliedoperation set
Animal_id='+#39+form21.Edit1.Text+#39+',
operation_name='+#39+form21.ComboBox1.Text+#39+',
operation_date='+#39+datetostr(form21.DateTimePicker1.Date)+#39+',
drug_name='+#39+form21.ComboBox2.Text+#39+',
Applied_staff='+#39+form21.Edit2.Text+#39+',
O_note='+#39+form21.Memo1.Text+#39+' where
aop_id='+#39+form21.Edit3.Text+#39;
  form21.ADOQuery3.ExecSQL;
  showmessage('RECORD UPDATED');
  Form21.LbSpeedButton4.Click;
 END
 ELSE
  SHOWMESSAGE('PLEASE SELECT OPERATION FROM LIST');
end;

procedure TForm21.LbSpeedButton3Click(Sender: TObject);
begin
 IF (FORM21.Edit1.Text <> '') AND (FORM21.Edit3.Text <> '') then
 BEGIN
  SS21:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM21.Edit1.Text+' ; OPERATION NAME: '
+FORM21.COMBOBOX1.Text+#13+'OPERATION DATE:
'+DATETOSTR(FORM21.DateTimePicker1.Date)+' " ?',MTWARNING,[MBYES
,MBNO],0);
  IF SS21 = MRYES then
  BEGIN
   FORM21.ADOQuery3.Close;
   form21.ADOQuery3.SQL.Text:='delete from appliedoperation where
aop_id='+#39+form21.Edit3.Text+#39;
   form21.ADOQuery3.ExecSQL;
   showmessage('RECORD DELETED');
   Form21.LbSpeedButton4.Click;
  END;
 end
 else
  showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm21.LbSpeedButton4Click(Sender: TObject);
begin
 FORM21.Edit3.Clear;
 FORM21.Edit1.Clear;
```

```
 FORM21.ComboBox1.Text:='Select One';
 form21.DateTimePicker1.Date:=date;
 FORM21.ComboBox2.Text:='Select One';
 FORM21.Edit2.Clear;
 FORM21.Memo1.Clear;
 form21.ComboBox1.SetFocus;
 Form21.LbSpeedButton1.Enabled:=TRUE;
 Form21.LbSpeedButton2.Enabled:=TRUE;

 form21.ADOQuery1.Close;
 form21.ADOQuery1.SQL.Text:='select * from appliedoperation';
 form21.ADOQuery1.Open;
 TUAHAPPOPR();
 TUAHAPPDRUG();
end;

procedure TForm21.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM21.ADOQuery1.RecordCount <> 0 THEN
 BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM21.Edit3.Text:=FORM21.DBGrid1.Fields[0].Text;
   FORM21.Edit1.Text:=FORM21.DBGrid1.Fields[1].Text;
   FORM21.ComboBox1.Text:=FORM21.DBGrid1.Fields[2].Text;

FORM21.DateTimePicker1.Date:=STRTODATE(FORM21.DBGrid1.Fields[3].Text);
   FORM21.ComboBox2.Text:=FORM21.DBGrid1.Fields[4].Text;
   FORM21.Edit2.Text:=FORM21.DBGrid1.Fields[5].Text;
   FORM21.Memo1.Text:=FORM21.DBGrid1.Fields[6].Text;
 END;
end;

procedure TForm21.SpeedButton2Click(Sender: TObject);
begin
 form39.show;
 ANI:=21;
end;

procedure TForm21.SpeedButton1Click(Sender: TObject);
begin
 form38.show;
 TA:=21;
end;

procedure TForm21.FormClose(Sender: TObject; var Action: TCloseAction);
begin
```

```
  Form21.LbSpeedButton4.Click;
  FORM21.ADOQuery1.Close;
  FORM21.ADOQuery2.Close;
  FORM21.ADOQuery3.Close;
  FORM21.ADOQuery4.Close;
  FORM21.ADOQuery5.Close;
end;

procedure TForm21.FormHide(Sender: TObject);
begin
  Form21.LbSpeedButton4.Click;
  FORM21.ADOQuery1.Close;
  FORM21.ADOQuery2.Close;
  FORM21.ADOQuery3.Close;
  FORM21.ADOQuery4.Close;
  FORM21.ADOQuery5.Close;
end;

procedure TForm21.Edit1Change(Sender: TObject);
begin
  form21.ADOQuery1.Close;
  form21.ADOQuery1.SQL.Text:='select * from appliedoperation where
animal_id='+#39+form21.Edit1.Text+#39;
  form21.ADOQuery1.Open;
  TUAHAPPOPR();
  TUAHAPPDRUG();
end;

end.
```

FORM 22 CODES

```
unit Unit22;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls, ComCtrls, Menus, ToolWin, LbSpeedButton,
  dxCore, dxButton, Grids, DBGrids, ExtCtrls, DB, ADODB;

type
  TForm22 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
```

```
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    DateTimePicker1: TDateTimePicker;
    Edit3: TEdit;
    Memo1: TMemo;
    Memo2: TMemo;
    Edit4: TEdit;
    Memo3: TMemo;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    StatusBar1: TStatusBar;
    SpeedButton7: TSpeedButton;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    SpeedButton8: TSpeedButton;
    Edit5: TEdit;
    Label9: TLabel;
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  Form22: TForm22;
  SS22:WORD;
  pn,yn,tbm:integer;
  spn,spn1,spn2,yspn,tamdt,prcdt1,prcdt2,prcdt3,prcdt4:string;
implementation

uses Unit10, Unit39, Unit38;

{$R *.dfm}

procedure TForm22.SpeedButton2Click(Sender: TObject);
begin
  if form22.SpeedButton2.Caption='r' then
  begin
  form22.Edit2.Clear;
  form22.Edit2.Enabled:=false;
  form22.Edit2.ReadOnly:=true;;
  form22.SpeedButton2.Caption:='b';
  //tbm:=0;
  end
  else if form22.SpeedButton2.Caption='b' then
  begin
  form22.Edit2.Clear;
  form22.Edit2.Enabled:=true;
  form22.Edit2.ReadOnly:=false;
  form22.SpeedButton2.Caption:='r';
  //tbm:=1;
  end;
  FORM22.SpeedButton3.Enabled:=FALSE;

end;

procedure TForm22.SpeedButton6Click(Sender: TObject);
begin
  FORM22.Edit5.Clear;
  FORM22.Edit1.Clear;
  FORM22.Edit2.Clear;
  form22.DateTimePicker1.Date:=date;
  FORM22.Edit3.Clear;
  FORM22.Edit4.Clear;
  FORM22.Memo1.Clear;
  FORM22.Memo2.Clear;
  FORM22.Memo3.Clear;
  form22.DateTimePicker1.SetFocus;
  Form22.SpeedButton3.Enabled:=TRUE;
```

```
 Form22.SpeedButton4.Enabled:=TRUE;

 form22.ADOQuery1.Close;
 form22.ADOQuery1.SQL.Text:='select * from illnesses';
 form22.ADOQuery1.Open;
 form22.Edit2.Enabled:=false;
 form22.Edit2.ReadOnly:=true;;
 form22.SpeedButton2.Caption:='b';
 FORM22.SpeedButton3.Enabled:=TRUE;
end;

procedure TForm22.SpeedButton8Click(Sender: TObject);
begin
 if form22.SpeedButton2.Caption='b' then
 begin
   dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'yyyy/mm/dd';

   form22.ADOQuery4.Close;
   form22.ADOQuery4.SQL.Text:='select max(protocol_no) from illnesses where
date='+#39+datetostr(FORM22.DateTimePicker1.Date)+#39;
   form22.ADOQuery4.Open;
   if form22.ADOQuery4['max(protocol_no)'] <> null then
   begin
     pn:=form22.ADOQuery4['max(protocol_no)'];
     spn:=inttostr(pn);
   //showmessage('maxprotocol:='+spn);
     spn1:=copy(spn,1,8);
   //showmessage('ilk tarih kýsmý:='+spn1);
     spn2:=copy(spn,9,length(spn));
   //showmessage('numara kýsmý:='+spn2);
     yn:=strtoint(spn2)+1;
   //showmessage('yeni numara kýsmý:='+inttostr(yn));
     yspn:=spn1+inttostr(yn);
   //showmessage('yeni protocol no:='+yspn);
   end
   else
   begin

     dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
     shortdateformat := 'dd/mm/yyyy';

     tamdt:=datetostr(FORM22.DateTimePicker1.Date);
   //showmessage('bugunun tarihi:='+tamdt);
     prcdt1:=copy(tamdt,1,2);
   //showmessage('tarihin günü:='+prcdt1);
     prcdt2:=copy(tamdt,4,2);
   //showmessage('tarihin ayý:='+prcdt2);
```

```
    prcdt3:=copy(tamdt,7,2);
   //showmessage('yýlýn ilk parçasý:='+prcdt3);
    prcdt4:=copy(tamdt,9,2);
   //showmessage('yýlýn son parçasý:='+prcdt4);
    yspn:=prcdt3+prcdt1+prcdt4+prcdt2+'1';
   //showmessage('yeni protocol no:='+yspn);
   end;
   FORM22.Edit2.Text:=yspn;
 end
 else
   showmessage('PLEASE DISABLE THE PROTOCOL NO BUTTON');
end;

procedure TForm22.FormCreate(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 FORM22.DateTimePicker1.Date:=DATE;
end;

procedure TForm22.SpeedButton3Click(Sender: TObject);
begin
 dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';

 if (form22.Edit1.Text <> '')  and (form22.Edit2.Text <> '' ) and (form22.Edit4.Text <>
'') THEN
 begin
   form22.ADOQuery2.Close;
   form22.ADOQuery2.SQL.Text:='select * from illnesses where
animal_id='+#39+form22.Edit1.Text+#39+'and
illness='+#39+form22.Edit3.Text+#39+' and
date='+#39+datetostr(form22.DateTimePicker1.Date)+#39;
   form22.ADOQuery2.Open;
   if form22.ADOQuery2.RecordCount = 0 then
   begin
     form22.ADOQuery2.Close;
     form22.ADOQuery2.SQL.Text:='insert into illnesses
(animal_id,protocol_no,date,illness,applied_staff,treatment,laboratory_result,i_note)
values
('+#39+form22.Edit1.Text+#39+','+#39+form22.Edit2.Text+#39+','+#39+datetostr(for
m22.DateTimePicker1.Date)+#39+','+#39+form22.Edit3.Text+#39+','+#39+form22.Edi
t4.Text+#39+','+#39+form22.Memo1.Text+#39+','+#39+form22.Memo2.Text+#39+','+
#39+form22.Memo3.Text+#39+')';
     form22.ADOQuery2.ExecSQL;
     showmessage('RECORD SAVED');
     Form22.SpeedButton6.Click;
```

```
  END
  ELSE
    SHOWMESSAGE('THE ILLNESS SAVED BEFORE');
 END
 ELSE if form22.Edit1.Text = '' then
  showmessage('PLEASE CHOOSE THE ANIMAL ID')
 ELSE if form22.Edit2.Text = '' then
  showmessage('PLEASE FILL THE PROTOCOL NO')
 ELSE if form22.Edit4.Text = '' then
  showmessage('PLEASE CHOOSE THE STAFF ID')
 else
  SHOWMESSAGE('BE SURE TO FILL THE EMPTY PLACE');


 {dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
 shortdateformat := 'yyyy/mm/dd';
 if form22.Edit2.Text <> '' then
 begin
 form22.ADOQuery2.Close;
 form22.ADOQuery2.SQL.Text:='insert into illnesses (protocol_no,date) values
('+#39+FORM22.Edit2.Text+#39+','+#39+datetostr(form22.DateTimePicker1.Date)+#3
9+')';
 form22.ADOQuery2.ExecSQL;
 form22.ADOQuery1.Close;
 form22.ADOQuery1.SQL.Text:='select protocol_no from illnesses';
 form22.ADOQuery1.Open;
 end;
 form22.Edit2.Clear;}
end;

procedure TForm22.Edit2Change(Sender: TObject);
begin
 //if tbm = 1 then
 if form22.SpeedButton2.Caption = 'r' then
 begin
 form22.ADOQuery1.Close;
 form22.ADOQuery1.SQL.Text:='select * from illnesses where protocol_no
like'+#39+'%'+form22.Edit2.Text+'%'+#39;
 form22.ADOQuery1.Open;
 end;
end;

procedure TForm22.Edit1Change(Sender: TObject);
begin
 form22.ADOQuery1.Close;
 form22.ADOQuery1.SQL.Text:='select * from illnesses where
animal_id='+#39+form22.Edit1.Text+#39;
 form22.ADOQuery1.Open;
end;
```

```
procedure TForm22.FormShow(Sender: TObject);
begin
  form22.ADOQuery1.Close;
  form22.ADOQuery1.SQL.Text:='select * from illnesses';
  form22.ADOQuery1.Open;
end;

procedure TForm22.SpeedButton4Click(Sender: TObject);
begin
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

  IF (FORM22.Edit1.Text <> '') AND (FORM22.Edit2.Text <> '') AND
(FORM22.Edit5.Text <> '')  AND (FORM22.Edit4.Text <> '') THEN
  BEGIN
    FORM22.ADOQuery3.Close;
    FORM22.ADOQuery3.SQL.Text:='UPDATE illnesses set
Animal_id='+#39+form22.Edit1.Text+#39+',
protocol_no='+#39+form22.Edit2.Text+#39+',
date='+#39+datetostr(form22.DateTimePicker1.Date)+#39+',
illness='+#39+form22.Edit3.Text+#39+',
Applied_staff='+#39+form22.Edit4.Text+#39+',
treatment='+#39+form22.Memo1.Text+#39+',
laboratory_result='+#39+form22.Memo2.Text+#39+',
i_note='+#39+form22.Memo3.Text+#39+' where ill_id='+#39+form22.Edit5.Text+#39;
    form22.ADOQuery3.ExecSQL;
    showmessage('RECORD UPDATED');
    Form22.SpeedButton6.Click;
  END
  ELSE
    SHOWMESSAGE('PLEASE SELECT ILLNESS FROM LIST');
end;

procedure TForm22.SpeedButton5Click(Sender: TObject);
begin
  IF (FORM22.Edit1.Text <> '')  AND (FORM22.Edit2.Text <> '') AND
(FORM22.Edit5.Text <> '') AND (FORM22.Edit4.Text <> '') then
  BEGIN
    SS22:=MESSAGEDLG('ARE YOU SURE TO DELETE  " ANIMAL ID:
'+FORM22.Edit1.Text+' ; ILLNESS: ' +FORM22.EDIT3.Text+'  ; PROTOCOL NO:
'+FORM22.Edit2.Text+' " ?',MTWARNING,[MBYES ,MBNO],0);
    IF SS22 = MRYES then
    BEGIN
      FORM22.ADOQuery3.Close;
      form22.ADOQuery3.SQL.Text:='delete from illnesses where
ill_id='+#39+form22.Edit5.Text+#39;
      form22.ADOQuery3.ExecSQL;
      showmessage('RECORD DELETED');
```

```
    Form22.SpeedButton6.Click;
   END;
  end
  else
   showmessage('PLEASE BE SURE TO SELECT DATA THAT YOU WILL
DELETE');
end;

procedure TForm22.SpeedButton1Click(Sender: TObject);
begin
 FORM39.SHOW;
 ANI:=22;
end;

procedure TForm22.SpeedButton7Click(Sender: TObject);
begin
 FORM38.SHOW;
 TA:=22;
end;

procedure TForm22.DBGrid1CellClick(Column: TColumn);
begin
 IF  FORM22.ADOQuery1.RecordCount <> 0 THEN
 BEGIN

   dateseparator := '.';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
   shortdateformat := 'dd/mm/yyyy';

   FORM22.Edit5.Text:=FORM22.DBGrid1.Fields[0].Text;
   FORM22.Edit1.Text:=FORM22.DBGrid1.Fields[1].Text;
   FORM22.Edit2.Text:=FORM22.DBGrid1.Fields[2].Text;

FORM22.DateTimePicker1.Date:=STRTODATE(FORM22.DBGrid1.Fields[3].Text);
   FORM22.Edit3.Text:=FORM22.DBGrid1.Fields[4].Text;
   FORM22.Edit4.Text:=FORM22.DBGrid1.Fields[7].Text;
   FORM22.Memo1.Text:=FORM22.DBGrid1.Fields[5].Text;
   FORM22.Memo2.Text:=FORM22.DBGrid1.Fields[6].Text;
   FORM22.Memo3.Text:=FORM22.DBGrid1.Fields[8].Text;
 END;
end;

procedure TForm22.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 Form22.SpeedButton6.Click;
 FORM22.ADOQuery1.Close;
 FORM22.ADOQuery2.Close;
 FORM22.ADOQuery3.Close;
 FORM22.ADOQuery4.Close;
end;
```

```
procedure TForm22.FormHide(Sender: TObject);
begin
  Form22.SpeedButton6.Click;
  FORM22.ADOQuery1.Close;
  FORM22.ADOQuery2.Close;
  FORM22.ADOQuery3.Close;
  FORM22.ADOQuery4.Close;
end;

end.
```

FORM 23 CODES

```
unit Unit23;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, ComCtrls, Buttons, StdCtrls, Grids, DBGrids, ExtCtrls,
  LbSpeedButton, DB, ADODB;

type
  TForm23 = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet5: TTabSheet;
    Edit1: TEdit;
    ComboBox1: TComboBox;
    Edit4: TEdit;
    Edit5: TEdit;
    Label1: TLabel;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    SpeedButton1: TSpeedButton;
    StatusBar1: TStatusBar;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    Panel2: TPanel;
    Edit2: TEdit;
    Panel3: TPanel;
    Label2: TLabel;
    Panel4: TPanel;
    DBGrid2: TDBGrid;
    LbSpeedButton1: TLbSpeedButton;
```

```
    LbSpeedButton2: TLbSpeedButton;
    SpeedButton2: TSpeedButton;
    Panel5: TPanel;
    Label3: TLabel;
    SpeedButton4: TSpeedButton;
    Panel6: TPanel;
    DBGrid3: TDBGrid;
    Panel7: TPanel;
    Panel8: TPanel;
    Label4: TLabel;
    LbSpeedButton3: TLbSpeedButton;
    LbSpeedButton4: TLbSpeedButton;
    DBGrid4: TDBGrid;
    Panel9: TPanel;
    Panel10: TPanel;
    Label5: TLabel;
    SpeedButton6: TSpeedButton;
    DBGrid5: TDBGrid;
    TabSheet7: TTabSheet;
    Panel13: TPanel;
    DBGrid7: TDBGrid;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    procedure CheckBox1Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure LbSpeedButton1Click(Sender: TObject);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure TabSheet1Show(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Edit5Change(Sender: TObject);
    procedure TabSheet7Show(Sender: TObject);
    procedure TabSheet5Show(Sender: TObject);
    procedure TabSheet4Show(Sender: TObject);
    procedure TabSheet3Show(Sender: TObject);
    procedure TabSheet2Show(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure LbSpeedButton3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  Form23: TForm23;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm23.CheckBox1Click(Sender: TObject);
begin
{  if form23.CheckBox1.Checked=true then
  begin
    form23.Label3.Enabled:=false;
    form23.ComboBox1.Enabled:=false;
    form23.RadioButton5.Enabled:=true;
    form23.RadioButton6.Enabled:=true;
    FORM23.Edit3.Enabled:=TRUE;
  end
  else if form23.CheckBox1.Checked=false then
  begin
    form23.Label3.Enabled:=true;
    form23.ComboBox1.Enabled:=true;
    form23.RadioButton5.Enabled:=false;
    form23.RadioButton6.Enabled:=false;
    form23.RadioButton5.Checked:=false;
    form23.RadioButton6.Checked:=false;
    FORM23.Edit3.Enabled:=FALSE;
    form23.Edit3.Clear;
  end;}
end;

procedure TForm23.Edit1Change(Sender: TObject);
begin
form23.ADOQuery1.Close;
form23.ADOQuery1.SQL.Text:='select * from customer where cname
like'+#39+'%'+form23.Edit1.Text+'%'+#39;
form23.ADOQuery1.Open;
end;

procedure TForm23.SpeedButton1Click(Sender: TObject);
begin
  form23.ADOQuery1.Close;
  form23.ADOQuery1.SQL.Text:='select * from customer where
cname='+#39+form23.Edit1.Text+#39;
  form23.ADOQuery1.Open;
end;

procedure TForm23.SpeedButton2Click(Sender: TObject);
begin
FORM23.Edit1.Clear;
```

```
form23.ADOQuery1.Close;
end;

procedure TForm23.Edit2Change(Sender: TObject);
begin
form23.ADOQuery1.Close;
form23.ADOQuery1.SQL.Text:='select * from customer where csurname
like'+#39+'%'+form23.Edit2.Text+'%'+#39;
form23.ADOQuery1.Open;
end;

procedure TForm23.LbSpeedButton1Click(Sender: TObject);
begin
 form23.ADOQuery1.Close;
 form23.ADOQuery1.SQL.Text:='select * from customer where
csurname='+#39+form23.Edit2.Text+#39;
 form23.ADOQuery1.Open;
end;

procedure TForm23.LbSpeedButton2Click(Sender: TObject);
begin
FORM23.Edit2.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.TabSheet1Show(Sender: TObject);
begin
form23.Edit1.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.ComboBox1Change(Sender: TObject);
begin
form23.ADOQuery1.Close;
form23.ADOQuery1.SQL.Text:='select * from customer where city
like'+#39+'%'+form23.ComboBox1.Text+'%'+#39;
form23.ADOQuery1.Open;
end;

procedure TForm23.Edit4Change(Sender: TObject);
begin
 form23.ADOQuery1.Close;
 form23.ADOQuery1.SQL.Text:='select * from customer where town
like'+#39+'%'+form23.Edit4.Text+'%'+#39;
 form23.ADOQuery1.Open;
end;

procedure TForm23.Edit5Change(Sender: TObject);
begin
 form23.ADOQuery1.Close;
```

```
 form23.ADOQuery1.SQL.Text:='select * from customer where customer_id
like'+#39+'%'+form23.Edit5.Text+'%'+#39;
 form23.ADOQuery1.Open;
end;

procedure TForm23.TabSheet7Show(Sender: TObject);
begin
form23.ADOQuery1.Close;
form23.ADOQuery1.SQL.Text:='select * from customer';
form23.ADOQuery1.Open;
end;

procedure TForm23.TabSheet5Show(Sender: TObject);
begin
form23.Edit5.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.TabSheet4Show(Sender: TObject);
begin
form23.Edit4.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.TabSheet3Show(Sender: TObject);
begin
form23.ComboBox1.Text:='Select One';
form23.ADOQuery1.Close;
end;

procedure TForm23.TabSheet2Show(Sender: TObject);
begin
form23.Edit2.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.SpeedButton4Click(Sender: TObject);
begin
form23.ComboBox1.Text:='Select One';
form23.ADOQuery1.Close;
end;

procedure TForm23.LbSpeedButton4Click(Sender: TObject);
begin
form23.Edit4.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.SpeedButton6Click(Sender: TObject);
begin
```

```
form23.Edit5.Clear;
form23.ADOQuery1.Close;
end;

procedure TForm23.LbSpeedButton3Click(Sender: TObject);
begin
ADOQuery1.Close;
ADOQuery1.SQL.Text:='select * from customer where
Form23.Edit4.Text+#39;
form23.ADOQuery1.Open;
end;

end.
```

FORM 24 CODES

```
unit Unit24;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, LbSpeedButton, StdCtrls, Buttons, ExtCtrls, Grids, DBGrids,
  ComCtrls, Menus, DB, ADODB;

type
  TForm24 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    Panel1: TPanel;
    Panel2: TPanel;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet5: TTabSheet;
    TabSheet6: TTabSheet;
    TabSheet7: TTabSheet;
    Panel3: TPanel;
    Panel4: TPanel;
    Panel5: TPanel;
    Panel6: TPanel;
    Panel7: TPanel;
    Panel8: TPanel;
    Label2: TLabel;
    LbSpeedButton2: TLbSpeedButton;
```

```
    LbSpeedButton4: TLbSpeedButton;
    LbSpeedButton6: TLbSpeedButton;
    Label3: TLabel;
    Edit3: TEdit;
    SpeedButton4: TSpeedButton;
    Label4: TLabel;
    Edit4: TEdit;
    Label5: TLabel;
    Edit5: TEdit;
    SpeedButton6: TSpeedButton;
    Label6: TLabel;
    Edit6: TEdit;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    ADOQuery2: TADOQuery;
    DataSource2: TDataSource;
    Edit2: TEdit;
    DBGrid3: TDBGrid;
    Panel9: TPanel;
    DBGrid4: TDBGrid;
    Panel10: TPanel;
    Panel11: TPanel;
    DBGrid5: TDBGrid;
    DBGrid6: TDBGrid;
    DBGrid7: TDBGrid;
    Panel12: TPanel;
    Panel13: TPanel;
    ADOQuery3: TADOQuery;
    ADOQuery4: TADOQuery;
    ADOQuery5: TADOQuery;
    ADOQuery6: TADOQuery;
    ADOQuery7: TADOQuery;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    DataSource5: TDataSource;
    DataSource6: TDataSource;
    DataSource7: TDataSource;
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Edit5Change(Sender: TObject);
    procedure Edit6Change(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure LbSpeedButton2Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure LbSpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure LbSpeedButton6Click(Sender: TObject);
    procedure TabSheet2Show(Sender: TObject);
```

```
    procedure TabSheet3Show(Sender: TObject);
    procedure TabSheet4Show(Sender: TObject);
    procedure TabSheet5Show(Sender: TObject);
    procedure TabSheet6Show(Sender: TObject);
    procedure TabSheet7Show(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormHide(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form24: TForm24;

implementation

uses Unit10, Unit23;

{$R *.dfm}

procedure TForm24.Edit1Change(Sender: TObject);
begin
  form24.ADOQuery1.Close;
  form24.ADOQuery1.SQL.Text:='select * from animal where cname
like'+#39+'%'+form23.Edit1.Text+'%'+#39;
  form24.ADOQuery1.Open;
end;

procedure TForm24.Edit2Change(Sender: TObject);
begin
  form24.ADOQuery1.Close;
  form24.ADOQuery1.SQL.Text:='select * from animal where animal_name
like'+#39+'%'+form24.Edit2.Text+'%'+#39;
  form24.ADOQuery1.Open;
end;

procedure TForm24.Edit3Change(Sender: TObject);
begin
  form24.ADOQuery1.Close;
  form24.ADOQuery1.SQL.Text:='select * from animal where collar_no
like'+#39+'%'+form24.Edit3.Text+'%'+#39;
  form24.ADOQuery1.Open;
end;

procedure TForm24.Edit4Change(Sender: TObject);
begin
  form24.ADOQuery1.Close;
```

```
 form24.ADOQuery1.SQL.Text:='select * from animal where earning_no
like'+#39+'%'+form24.Edit4.Text+'%'+#39;
 form24.ADOQuery1.Open;
end;

procedure TForm24.Edit5Change(Sender: TObject);
begin
 form24.ADOQuery1.Close;
 form24.ADOQuery1.SQL.Text:='select * from animal where animal_id
like'+#39+'%'+form24.Edit5.Text+'%'+#39;
 form24.ADOQuery1.Open;
end;

procedure TForm24.Edit6Change(Sender: TObject);
begin
 form24.ADOQuery1.Close;
 form24.ADOQuery1.SQL.Text:='select * from animal where animal_race
like'+#39+'%'+form24.Edit6.Text+'%'+#39;
 form24.ADOQuery1.Open;
end;

procedure TForm24.DBGrid1CellClick(Column: TColumn);
begin
 if (form24.ADOQuery1.IsEmpty <> true) then
 begin
   FORM24.ADOQuery2.Close;
   FORM24.ADOQuery2.SQL.Text:='select * from vaccinate where
animal_id='+#39+form24.DBGrid1.Fields[0].Text+#39;
   form24.ADOQuery2.Open;
   FORM24.ADOQuery3.Close;
   FORM24.ADOQuery3.SQL.Text:='select * from ipdrug where
animal_id='+#39+form24.DBGrid1.Fields[0].Text+#39;
   form24.ADOQuery3.Open;
   FORM24.ADOQuery4.Close;
   FORM24.ADOQuery4.SQL.Text:='select * from opdrug where
animal_id='+#39+form24.DBGrid1.Fields[0].Text+#39;
   form24.ADOQuery4.Open;
   FORM24.ADOQuery5.Close;
   FORM24.ADOQuery5.SQL.Text:='select * from illnesses where
animal_id='+#39+form24.DBGrid1.Fields[0].Text+#39;
   form24.ADOQuery5.Open;
   FORM24.ADOQuery6.Close;
   FORM24.ADOQuery6.SQL.Text:='select * from medicinate where
animal_id='+#39+form24.DBGrid1.Fields[0].Text+#39;
   form24.ADOQuery6.Open;
   FORM24.ADOQuery7.Close;
   FORM24.ADOQuery7.SQL.Text:='select * from appliedoperation where
animal_id='+#39+form24.DBGrid1.Fields[0].Text+#39;
   form24.ADOQuery7.Open;
 end;
```

```
end;

procedure TForm24.LbSpeedButton2Click(Sender: TObject);
begin
  FORM24.Edit2.Clear;
  form24.ADOQuery1.Close;
  form24.ADOQuery2.Close;
  form24.ADOQuery3.Close;
  form24.ADOQuery4.Close;
  form24.ADOQuery5.Close;
  form24.ADOQuery6.Close;
  form24.ADOQuery7.Close;
end;

procedure TForm24.SpeedButton4Click(Sender: TObject);
begin
  FORM24.Edit3.Clear;
  form24.ADOQuery1.Close;
  form24.ADOQuery2.Close;
  form24.ADOQuery3.Close;
  form24.ADOQuery4.Close;
  form24.ADOQuery5.Close;
  form24.ADOQuery6.Close;
  form24.ADOQuery7.Close;
end;

procedure TForm24.LbSpeedButton4Click(Sender: TObject);
begin
  FORM24.Edit4.Clear;
  form24.ADOQuery1.Close;
  form24.ADOQuery2.Close;
  form24.ADOQuery3.Close;
  form24.ADOQuery4.Close;
  form24.ADOQuery5.Close;
  form24.ADOQuery6.Close;
  form24.ADOQuery7.Close;
end;

procedure TForm24.SpeedButton6Click(Sender: TObject);
begin
  FORM24.Edit5.Clear;
  form24.ADOQuery1.Close;
  form24.ADOQuery2.Close;
  form24.ADOQuery3.Close;
  form24.ADOQuery4.Close;
  form24.ADOQuery5.Close;
  form24.ADOQuery6.Close;
  form24.ADOQuery7.Close;
end;
```

```
procedure TForm24.LbSpeedButton6Click(Sender: TObject);
begin
  FORM24.Edit6.Clear;
  form24.ADOQuery1.Close;
  form24.ADOQuery2.Close;
  form24.ADOQuery3.Close;
  form24.ADOQuery4.Close;
  form24.ADOQuery5.Close;
  form24.ADOQuery6.Close;
  form24.ADOQuery7.Close;
end;

procedure TForm24.TabSheet2Show(Sender: TObject);
begin
  Form24.LbSpeedButton2.Click;
end;

procedure TForm24.TabSheet3Show(Sender: TObject);
begin
  Form24.SpeedButton4.Click;
end;

procedure TForm24.TabSheet4Show(Sender: TObject);
begin
  Form24.LbSpeedButton4.Click;
end;

procedure TForm24.TabSheet5Show(Sender: TObject);
begin
  Form24.SpeedButton6.Click;
end;

procedure TForm24.TabSheet6Show(Sender: TObject);
begin
  Form24.LbSpeedButton6.Click;
end;

procedure TForm24.TabSheet7Show(Sender: TObject);
begin
  form24.ADOQuery1.Close;
  form24.ADOQuery1.SQL.Text:='select * from animal' ;
  form24.ADOQuery1.Open;
  form24.ADOQuery2.Close;
  form24.ADOQuery3.Close;
  form24.ADOQuery4.Close;
  form24.ADOQuery5.Close;
  form24.ADOQuery6.Close;
  form24.ADOQuery7.Close;
end;
```

```
procedure TForm24.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  FORM24.Edit2.Clear;
  FORM24.Edit3.Clear;
  FORM24.Edit4.Clear;
  FORM24.Edit5.Clear;
  FORM24.Edit6.Clear;
  FORM24.ADOQuery1.Close;
  FORM24.ADOQuery2.Close;
  FORM24.ADOQuery3.Close;
  FORM24.ADOQuery4.Close;
  FORM24.ADOQuery5.Close;
  FORM24.ADOQuery6.Close;
  FORM24.ADOQuery7.Close;
end;

procedure TForm24.FormHide(Sender: TObject);
begin
  FORM24.Edit2.Clear;
  FORM24.Edit3.Clear;
  FORM24.Edit4.Clear;
  FORM24.Edit5.Clear;
  FORM24.Edit6.Clear;
  FORM24.ADOQuery1.Close;
  FORM24.ADOQuery2.Close;
  FORM24.ADOQuery3.Close;
  FORM24.ADOQuery4.Close;
  FORM24.ADOQuery5.Close;
  FORM24.ADOQuery6.Close;
  FORM24.ADOQuery7.Close;
end;

end.
```

FORM 25 CODES

```
unit Unit25;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, ComCtrls, Grids, DBGrids, Menus, Buttons,
  LbSpeedButton, DB, ADODB;

type
  TForm25 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
```

```
PageControl1: TPageControl;
TabSheet1: TTabSheet;
TabSheet2: TTabSheet;
TabSheet3: TTabSheet;
TabSheet4: TTabSheet;
TabSheet5: TTabSheet;
TabSheet6: TTabSheet;
TabSheet7: TTabSheet;
TabSheet9: TTabSheet;
DBGrid1: TDBGrid;
StatusBar1: TStatusBar;
Panel1: TPanel;
Label1: TLabel;
Edit1: TEdit;
Panel13: TPanel;
LbSpeedButton2: TLbSpeedButton;
LbSpeedButton4: TLbSpeedButton;
LbSpeedButton6: TLbSpeedButton;
LbSpeedButton8: TLbSpeedButton;
Label2: TLabel;
Edit2: TEdit;
SpeedButton2: TSpeedButton;
Label3: TLabel;
Edit3: TEdit;
Label4: TLabel;
SpeedButton3: TSpeedButton;
Label5: TLabel;
Edit5: TEdit;
Label6: TLabel;
DateTimePicker1: TDateTimePicker;
SpeedButton6: TSpeedButton;
Label7: TLabel;
DateTimePicker2: TDateTimePicker;
Panel2: TPanel;
Panel3: TPanel;
Panel4: TPanel;
Panel5: TPanel;
Panel6: TPanel;
Panel7: TPanel;
Panel9: TPanel;
ComboBox1: TComboBox;
ADOQuery1: TADOQuery;
DataSource1: TDataSource;
procedure CheckBox1Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure Edit3Change(Sender: TObject);
procedure ComboBox1Change(Sender: TObject);
procedure Edit5Change(Sender: TObject);
procedure TabSheet6Show(Sender: TObject);
```

```
  procedure DateTimePicker1Change(Sender: TObject);
  procedure DateTimePicker2Change(Sender: TObject);
  procedure TabSheet7Show(Sender: TObject);
  procedure TabSheet9Show(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure SpeedButton3Click(Sender: TObject);
  procedure LbSpeedButton6Click(Sender: TObject);
  procedure SpeedButton6Click(Sender: TObject);
  procedure LbSpeedButton8Click(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure TabSheet3Show(Sender: TObject);
  procedure TabSheet4Show(Sender: TObject);
  procedure TabSheet5Show(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form25: TForm25;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm25.CheckBox1Click(Sender: TObject);
begin
{if form25.CheckBox1.Checked=true then
 begin
  form25.Label8.Enabled:=false;
  form25.ComboBox1.Enabled:=false;
  form25.RadioButton11.Enabled:=true;
  form25.RadioButton12.Enabled:=true;
  FORM25.Edit6.Enabled:=TRUE;
 end
 else if form25.CheckBox1.Checked=false then
 begin
  form25.Label8.Enabled:=true;
  form25.ComboBox1.Enabled:=true;
  form25.RadioButton11.Enabled:=false;
  form25.RadioButton12.Enabled:=false;
  form25.RadioButton11.Checked:=false;
```

```
  form25.RadioButton12.Checked:=false;
  FORM25.Edit6.Enabled:=FALSE;
  form25.Edit6.Clear;
 end;}
end;

procedure TForm25.Edit1Change(Sender: TObject);
begin
 form25.ADOQuery1.Close;
 form25.ADOQuery1.SQL.Text:='select * from staff where staff_name
like'+#39+'%'+form25.Edit1.Text+'%'+#39;
 form25.ADOQuery1.Open;
end;

procedure TForm25.Edit2Change(Sender: TObject);
begin
 form25.ADOQuery1.Close;
 form25.ADOQuery1.SQL.Text:='select * from staff where staff_surname
like'+#39+'%'+form25.Edit2.Text+'%'+#39;
 form25.ADOQuery1.Open;
end;

procedure TForm25.Edit3Change(Sender: TObject);
begin
 form25.ADOQuery1.Close;
 form25.ADOQuery1.SQL.Text:='select * from staff where staff_id
like'+#39+'%'+form25.Edit3.Text+'%'+#39;
 form25.ADOQuery1.Open;
end;

procedure TForm25.ComboBox1Change(Sender: TObject);
begin
 form25.ADOQuery1.Close;
 form25.ADOQuery1.SQL.Text:='select * from staff where staff_task
like'+#39+'%'+form25.ComboBox1.Text+'%'+#39;
 form25.ADOQuery1.Open;
end;

procedure TForm25.Edit5Change(Sender: TObject);
begin
 form25.ADOQuery1.Close;
 form25.ADOQuery1.SQL.Text:='select * from staff where university
like'+#39+'%'+form25.Edit5.Text+'%'+#39;
 form25.ADOQuery1.Open;
end;

procedure TForm25.TabSheet6Show(Sender: TObject);
begin
 Form25.SpeedButton6.Click;
```

```
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';

end;

procedure TForm25.DateTimePicker1Change(Sender: TObject);
begin
  form25.ADOQuery1.Close;
  form25.ADOQuery1.SQL.Text:='select * from staff where s_workstartdate
like'+#39+'%'+datetostr(form25.DateTimePicker1.Date)+'%'+#39;
  form25.ADOQuery1.Open;
end;

procedure TForm25.DateTimePicker2Change(Sender: TObject);
begin
  form25.ADOQuery1.Close;
  form25.ADOQuery1.SQL.Text:='select * from staff where s_birthdate
like'+#39+'%'+datetostr(form25.DateTimePicker2.Date)+'%'+#39;
  form25.ADOQuery1.Open;
end;

procedure TForm25.TabSheet7Show(Sender: TObject);
begin
  Form25.LbSpeedButton8.Click;
  dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
  shortdateformat := 'yyyy/mm/dd';
end;

procedure TForm25.TabSheet9Show(Sender: TObject);
begin
  form25.ADOQuery1.Close;
  form25.ADOQuery1.SQL.Text:='select * from staff';
  form25.ADOQuery1.Open;
end;

procedure TForm25.LbSpeedButton2Click(Sender: TObject);
begin
  FORM25.Edit1.Clear;
  form25.ADOQuery1.Close;
end;

procedure TForm25.SpeedButton2Click(Sender: TObject);
begin
  FORM25.Edit2.Clear;
  form25.ADOQuery1.Close;
end;

procedure TForm25.LbSpeedButton4Click(Sender: TObject);
```

```
begin
  FORM25.Edit3.Clear;
  form25.ADOQuery1.Close;
end;

procedure TForm25.SpeedButton3Click(Sender: TObject);
begin
  FORM25.ComboBox1.Text:='Select One';
  form25.ADOQuery1.Close;
end;

procedure TForm25.LbSpeedButton6Click(Sender: TObject);
begin
  FORM25.Edit5.Clear;
  form25.ADOQuery1.Close;
end;

procedure TForm25.SpeedButton6Click(Sender: TObject);
begin
  form25.DateTimePicker1.Date:=date;
  form25.ADOQuery1.Close;
end;

procedure TForm25.LbSpeedButton8Click(Sender: TObject);
begin
  form25.DateTimePicker2.Date:=date;
  form25.ADOQuery1.Close;
end;

procedure TForm25.TabSheet1Show(Sender: TObject);
begin
Form25.LbSpeedButton2.Click;
end;

procedure TForm25.TabSheet2Show(Sender: TObject);
begin
Form25.SpeedButton2.Click;
end;

procedure TForm25.TabSheet3Show(Sender: TObject);
begin
Form25.LbSpeedButton4.Click;
end;

procedure TForm25.TabSheet4Show(Sender: TObject);
begin
  Form25.SpeedButton3.Click;
end;

procedure TForm25.TabSheet5Show(Sender: TObject);
```

```
begin
  Form25.LbSpeedButton6.Click;
end;

procedure TForm25.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form25.Edit1.Clear;
  form25.Edit2.Clear;
  form25.Edit3.Clear;
  form25.ComboBox1.Text:='Select One';
  form25.Edit5.Clear;
  form25.DateTimePicker1.Date:=date;
  form25.DateTimePicker2.Date:=date;
  form25.ADOQuery1.Close;
end;

procedure TForm25.FormHide(Sender: TObject);
begin
  form25.Edit1.Clear;
  form25.Edit2.Clear;
  form25.Edit3.Clear;
  form25.ComboBox1.Text:='Select One';
  form25.Edit5.Clear;
  form25.DateTimePicker1.Date:=date;
  form25.DateTimePicker2.Date:=date;
  form25.ADOQuery1.Close;
end;

end.
```

FORM 26 CODES

```
unit Unit26;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls, Grids, DBGrids, ExtCtrls, ComCtrls, Menus,
  DB, ADODB;

type
  TForm26 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    Panel1: TPanel;
```

```
  DBGrid1: TDBGrid;
  TabSheet3: TTabSheet;
  Label1: TLabel;
  Edit1: TEdit;
  SpeedButton2: TSpeedButton;
  Panel5: TPanel;
  Panel2: TPanel;
  Label2: TLabel;
  Edit2: TEdit;
  SpeedButton4: TSpeedButton;
  Panel3: TPanel;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  procedure Edit1Change(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure TabSheet3Show(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure SpeedButton4Click(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure FormHide(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 Form26: TForm26;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm26.Edit1Change(Sender: TObject);
begin
  form26.ADOQuery1.Close;
  form26.ADOQuery1.SQL.Text:='select * from vaccines where vaccine_name
like'+#39+'%'+form26.Edit1.Text+'%'+#39;
  form26.ADOQuery1.Open;
end;

procedure TForm26.Edit2Change(Sender: TObject);
begin
  form26.ADOQuery1.Close;
  form26.ADOQuery1.SQL.Text:='select * from vaccines where vaccine_id
like'+#39+'%'+form26.Edit2.Text+'%'+#39;
```

```
 form26.ADOQuery1.Open;
end;

procedure TForm26.TabSheet3Show(Sender: TObject);
begin
 form26.ADOQuery1.Close;
 form26.ADOQuery1.SQL.Text:='select * from vaccines';
 form26.ADOQuery1.Open;
end;

procedure TForm26.SpeedButton2Click(Sender: TObject);
begin
 FORM26.Edit1.Clear;
 form26.ADOQuery1.Close;
end;

procedure TForm26.SpeedButton4Click(Sender: TObject);
begin
 FORM26.Edit2.Clear;
 form26.ADOQuery1.Close;
end;

procedure TForm26.TabSheet1Show(Sender: TObject);
begin
Form26.SpeedButton2.Click;
end;

procedure TForm26.TabSheet2Show(Sender: TObject);
begin
Form26.SpeedButton4.Click;
end;

procedure TForm26.FormHide(Sender: TObject);
begin
  form26.Edit1.Clear;
  form26.Edit2.Clear;
  form26.ADOQuery1.Close;
end;

procedure TForm26.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form26.Edit1.Clear;
  form26.Edit2.Clear;
  form26.ADOQuery1.Close;
end;

end.
```

FORM 27 CODES

```
unit Unit27;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, ComCtrls, StdCtrls, ExtCtrls, Grids, DBGrids, Menus,
  LbSpeedButton, DB, ADODB;

type
  TForm27 = class(TForm)
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet7: TTabSheet;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    Label1: TLabel;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    Panel5: TPanel;
    Label2: TLabel;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    DateTimePicker3: TDateTimePicker;
    DateTimePicker4: TDateTimePicker;
    RadioButton5: TRadioButton;
    RadioButton6: TRadioButton;
    RadioButton7: TRadioButton;
    RadioButton8: TRadioButton;
    Panel2: TPanel;
    Panel3: TPanel;
    Label3: TLabel;
    Edit1: TEdit;
    SpeedButton4: TSpeedButton;
    Panel4: TPanel;
    Label4: TLabel;
```

```
  LbSpeedButton4: TLbSpeedButton;
  Panel7: TPanel;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  Edit2: TEdit;
  procedure LbSpeedButton3Click(Sender: TObject);
  procedure RadioButton4Click(Sender: TObject);
  procedure RadioButton3Click(Sender: TObject);
  procedure RadioButton2Click(Sender: TObject);
  procedure RadioButton1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure RadioButton6Click(Sender: TObject);
  procedure RadioButton5Click(Sender: TObject);
  procedure RadioButton7Click(Sender: TObject);
  procedure RadioButton8Click(Sender: TObject);
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure TabSheet7Show(Sender: TObject);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure SpeedButton4Click(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure TabSheet3Show(Sender: TObject);
  procedure TabSheet4Show(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form27: TForm27;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm27.LbSpeedButton3Click(Sender: TObject);
begin
SHOWMESSAGE('SELAMUN ALEYKÜM');
end;
```

```
procedure TForm27.RadioButton4Click(Sender: TObject);
begin
if form27.RadioButton4.Checked = true then
  form27.DateTimePicker2.Enabled:=true
else
  form27.DateTimePicker2.Enabled:=false;
end;

procedure TForm27.RadioButton3Click(Sender: TObject);
begin
  form27.DateTimePicker2.Date:=date;
  form27.DateTimePicker2.Enabled:=false;
end;

procedure TForm27.RadioButton2Click(Sender: TObject);
begin
  form27.DateTimePicker2.Date:=date;
  form27.DateTimePicker2.Enabled:=false;
end;

procedure TForm27.RadioButton1Click(Sender: TObject);
begin
  form27.DateTimePicker2.Date:=date;
  form27.DateTimePicker2.Enabled:=false;
end;

procedure TForm27.FormCreate(Sender: TObject);
begin
form27.DateTimePicker1.Date:=date;
form27.DateTimePicker2.Date:=date;
form27.DateTimePicker3.Date:=date;
form27.DateTimePicker4.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm27.SpeedButton1Click(Sender: TObject);
begin
  if (form27.RadioButton1.Checked = true) then
  begin
    form27.ADOQuery1.Close;
    form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_drugdate <
'+#39+datetostr(form27.DateTimePicker1.Date)+#39;
    form27.ADOQuery1.Open;
  end
  else if (form27.RadioButton2.Checked = true) then
  begin
    form27.ADOQuery1.Close;
```

```
  form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_drugdate >
'+#39+datetostr(form27.DateTimePicker1.Date)+#39;
  form27.ADOQuery1.Open;
 end
 else if (form27.RadioButton3.Checked = true) then
 begin
  form27.ADOQuery1.Close;
  form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_drugdate =
'+#39+datetostr(form27.DateTimePicker1.Date)+#39;
  form27.ADOQuery1.Open;
 end
 else if (form27.RadioButton4.Checked = true) then
 begin
  form27.ADOQuery1.Close;
  form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_drugdate between
'+#39+datetostr(form27.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form27.DateTimePicker2.Date)+#39;
  form27.ADOQuery1.Open;
 end
 else
  showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm27.SpeedButton2Click(Sender: TObject);
begin
 FORM27.DateTimePicker1.Date:=DATE;
 FORM27.DateTimePicker2.Date:=DATE;
 FORM27.DateTimePicker2.Enabled:=FALSE;
 FORM27.RadioButton1.Checked:=FALSE;
 FORM27.RadioButton2.Checked:=FALSE;
 FORM27.RadioButton3.Checked:=FALSE;
 FORM27.RadioButton4.Checked:=FALSE;
 FORM27.ADOQuery1.Close;
end;

procedure TForm27.RadioButton6Click(Sender: TObject);
begin
if form27.RadioButton6.Checked = true then
 form27.DateTimePicker4.Enabled:=true
else
 form27.DateTimePicker4.Enabled:=false;
end;

procedure TForm27.RadioButton5Click(Sender: TObject);
begin
 form27.DateTimePicker4.Date:=date;
 form27.DateTimePicker4.Enabled:=false;
end;

procedure TForm27.RadioButton7Click(Sender: TObject);
```

```
begin
  form27.DateTimePicker4.Date:=date;
  form27.DateTimePicker4.Enabled:=false;
end;

procedure TForm27.RadioButton8Click(Sender: TObject);
begin
  form27.DateTimePicker4.Date:=date;
  form27.DateTimePicker4.Enabled:=false;
end;

procedure TForm27.LbSpeedButton1Click(Sender: TObject);
begin
if (form27.RadioButton5.Checked = true) then
  begin
    form27.ADOQuery1.Close;
    form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_nextdrugdate <
'+#39+datetostr(form27.DateTimePicker3.Date)+#39;
    form27.ADOQuery1.Open;
  end
  else if (form27.RadioButton7.Checked = true) then
  begin
    form27.ADOQuery1.Close;
    form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_nextdrugdate >
'+#39+datetostr(form27.DateTimePicker3.Date)+#39;
    form27.ADOQuery1.Open;
  end
  else if (form27.RadioButton8.Checked = true) then
  begin
    form27.ADOQuery1.Close;
    form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_nextdrugdate =
'+#39+datetostr(form27.DateTimePicker3.Date)+#39;
    form27.ADOQuery1.Open;
  end
  else if (form27.RadioButton6.Checked = true) then
  begin
    form27.ADOQuery1.Close;
    form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_nextdrugdate
 between '+#39+datetostr(form27.DateTimePicker3.Date)+#39+' and
'+#39+datetostr(form27.DateTimePicker3.Date)+#39;
    form27.ADOQuery1.Open;
  end
  else
    showmessage('PLEASE SELECT SEARCH CRITERIA');
 end;

 procedure TForm27.LbSpeedButton2Click(Sender: TObject);
 begin
  FORM27.DateTimePicker3.Date:=DATE;
  FORM27.DateTimePicker4.Date:=DATE;
```

```
 FORM27.DateTimePicker4.Enabled:=FALSE;
 FORM27.RadioButton5.Checked:=FALSE;
 FORM27.RadioButton6.Checked:=FALSE;
 FORM27.RadioButton7.Checked:=FALSE;
 FORM27.RadioButton8.Checked:=FALSE;
 FORM27.ADOQuery1.Close;
end;

procedure TForm27.Edit1Change(Sender: TObject);
begin
 form27.ADOQuery1.Close;
 form27.ADOQuery1.SQL.Text:='select * from ipdrug where animal_id
like'+#39+'%'+form27.Edit1.Text+'%'+#39;
 form27.ADOQuery1.Open;
end;

procedure TForm27.Edit2Change(Sender: TObject);
begin
 form27.ADOQuery1.Close;
 form27.ADOQuery1.SQL.Text:='select * from ipdrug where ip_drugname
like'+#39+'%'+form27.Edit2.Text+'%'+#39;
 form27.ADOQuery1.Open;
end;

procedure TForm27.TabSheet7Show(Sender: TObject);
begin
 form27.ADOQuery1.Close;
 form27.ADOQuery1.SQL.Text:='select * from ipdrug';
 form27.ADOQuery1.Open;
end;

procedure TForm27.LbSpeedButton4Click(Sender: TObject);
begin
form27.Edit2.Clear;
form27.ADOQuery1.Close;
end;

procedure TForm27.SpeedButton4Click(Sender: TObject);
begin
form27.Edit1.Clear;
form27.ADOQuery1.Close;
end;

procedure TForm27.TabSheet1Show(Sender: TObject);
begin
 Form27.SpeedButton2.Click;
end;

procedure TForm27.TabSheet2Show(Sender: TObject);
begin
```

```
Form27.LbSpeedButton2.Click;
end;

procedure TForm27.TabSheet3Show(Sender: TObject);
begin
Form27.SpeedButton4.Click;
end;

procedure TForm27.TabSheet4Show(Sender: TObject);
begin
Form27.LbSpeedButton4.Click;
end;

procedure TForm27.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form27.Edit1.Clear;
  form27.Edit2.Clear;
  form27.ADOQuery1.Close;
  form27.RadioButton1.Checked:=false;
  form27.RadioButton2.Checked:=false;
  form27.RadioButton3.Checked:=false;
  form27.RadioButton4.Checked:=false;
  form27.RadioButton5.Checked:=false;
  form27.RadioButton6.Checked:=false;
  form27.RadioButton7.Checked:=false;
  form27.RadioButton8.Checked:=false;
  form27.DateTimePicker2.Enabled:=false;
  form27.DateTimePicker4.Enabled:=false;
end;

procedure TForm27.FormHide(Sender: TObject);
begin
  form27.Edit1.Clear;
  form27.Edit2.Clear;
  form27.ADOQuery1.Close;
  form27.RadioButton1.Checked:=false;
  form27.RadioButton2.Checked:=false;
  form27.RadioButton3.Checked:=false;
  form27.RadioButton4.Checked:=false;
  form27.RadioButton5.Checked:=false;
  form27.RadioButton6.Checked:=false;
  form27.RadioButton7.Checked:=false;
  form27.RadioButton8.Checked:=false;
  form27.DateTimePicker2.Enabled:=false;
  form27.DateTimePicker4.Enabled:=false;
end;

end.
```

FORM 28 CODES

```
unit Unit28;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Grids, DBGrids, StdCtrls, LbSpeedButton, ExtCtrls,
  ComCtrls, Buttons, DB, ADODB;

type
  TForm28 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Panel2: TPanel;
    TabSheet2: TTabSheet;
    Label2: TLabel;
    LbSpeedButton1: TLbSpeedButton;
    LbSpeedButton2: TLbSpeedButton;
    Panel5: TPanel;
    DateTimePicker3: TDateTimePicker;
    DateTimePicker4: TDateTimePicker;
    RadioButton5: TRadioButton;
    RadioButton6: TRadioButton;
    RadioButton7: TRadioButton;
    RadioButton8: TRadioButton;
    TabSheet4: TTabSheet;
    Label4: TLabel;
    LbSpeedButton4: TLbSpeedButton;
    Panel4: TPanel;
    TabSheet5: TTabSheet;
    Label5: TLabel;
    SpeedButton6: TSpeedButton;
    Panel6: TPanel;
    Edit2: TEdit;
    TabSheet7: TTabSheet;
    Panel7: TPanel;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
```

```
  MainMenu1: TMainMenu;
  F1: TMenuItem;
  TabSheet3: TTabSheet;
  Label3: TLabel;
  SpeedButton4: TSpeedButton;
  Panel3: TPanel;
  Edit1: TEdit;
  Edit3: TEdit;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  procedure RadioButton1Click(Sender: TObject);
  procedure RadioButton2Click(Sender: TObject);
  procedure RadioButton3Click(Sender: TObject);
  procedure RadioButton4Click(Sender: TObject);
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure RadioButton6Click(Sender: TObject);
  procedure RadioButton5Click(Sender: TObject);
  procedure RadioButton7Click(Sender: TObject);
  procedure RadioButton8Click(Sender: TObject);
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure Edit3Change(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure TabSheet7Show(Sender: TObject);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure SpeedButton6Click(Sender: TObject);
  procedure SpeedButton4Click(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure TabSheet4Show(Sender: TObject);
  procedure TabSheet5Show(Sender: TObject);
  procedure TabSheet3Show(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormCreate(Sender: TObject);
  procedure FormHide(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form28: TForm28;

implementation

uses Unit10;
```

```
{$R *.dfm}

procedure TForm28.RadioButton1Click(Sender: TObject);
begin
  form28.DateTimePicker2.Date:=date;
  form28.DateTimePicker2.Enabled:=false;
end;

procedure TForm28.RadioButton2Click(Sender: TObject);
begin
  form28.DateTimePicker2.Date:=date;
  form28.DateTimePicker2.Enabled:=false;
end;

procedure TForm28.RadioButton3Click(Sender: TObject);
begin
  form28.DateTimePicker2.Date:=date;
  form28.DateTimePicker2.Enabled:=false;
end;

procedure TForm28.RadioButton4Click(Sender: TObject);
begin
if form28.RadioButton4.Checked = true then
  form28.DateTimePicker2.Enabled:=true
else
  form28.DateTimePicker2.Enabled:=false;
end;

procedure TForm28.SpeedButton1Click(Sender: TObject);
begin
if (form28.RadioButton1.Checked = true) then
  begin
    form28.ADOQuery1.Close;
    form28.ADOQuery1.SQL.Text:='select * from vaccinate where vaccinate_date <
'+#39+datetostr(form28.DateTimePicker1.Date)+#39;
    form28.ADOQuery1.Open;
  end
  else if (form28.RadioButton2.Checked = true) then
  begin
    form28.ADOQuery1.Close;
    form28.ADOQuery1.SQL.Text:='select * from vaccinate where vaccinate_date >
'+#39+datetostr(form28.DateTimePicker1.Date)+#39;
    form28.ADOQuery1.Open;
  end
  else if (form28.RadioButton3.Checked = true) then
  begin
    form28.ADOQuery1.Close;
    form28.ADOQuery1.SQL.Text:='select * from vaccinate where vaccinate_date =
'+#39+datetostr(form28.DateTimePicker1.Date)+#39;
    form28.ADOQuery1.Open;
```

```
 end
 else if (form28.RadioButton4.Checked = true) then
 begin
   form28.ADOQuery1.Close;
   form28.ADOQuery1.SQL.Text:='select * from vaccinate where vaccinate_date
between '+#39+datetostr(form28.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form28.DateTimePicker2.Date)+#39;
   form28.ADOQuery1.Open;
 end
 else
   showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm28.SpeedButton2Click(Sender: TObject);
begin
 FORM28.DateTimePicker1.Date:=DATE;
 FORM28.DateTimePicker2.Date:=DATE;
 FORM28.DateTimePicker2.Enabled:=FALSE;
 FORM28.RadioButton1.Checked:=FALSE;
 FORM28.RadioButton2.Checked:=FALSE;
 FORM28.RadioButton3.Checked:=FALSE;
 FORM28.RadioButton4.Checked:=FALSE;
 FORM28.ADOQuery1.Close;
end;

procedure TForm28.RadioButton6Click(Sender: TObject);
begin
if form28.RadioButton6.Checked = true then
 form28.DateTimePicker4.Enabled:=true
else
 form28.DateTimePicker4.Enabled:=false;
end;

procedure TForm28.RadioButton5Click(Sender: TObject);
begin
 form28.DateTimePicker4.Date:=date;
 form28.DateTimePicker4.Enabled:=false;
end;

procedure TForm28.RadioButton7Click(Sender: TObject);
begin
 form28.DateTimePicker4.Date:=date;
 form28.DateTimePicker4.Enabled:=false;
end;

procedure TForm28.RadioButton8Click(Sender: TObject);
begin
 form28.DateTimePicker4.Date:=date;
 form28.DateTimePicker4.Enabled:=false;
end;
```

```
procedure TForm28.LbSpeedButton1Click(Sender: TObject);
begin
if (form28.RadioButton5.Checked = true) then
 begin
  form28.ADOQuery1.Close;
  form28.ADOQuery1.SQL.Text:='select * from vaccinate where next_vaccinatedate <
'+#39+datetostr(form28.DateTimePicker3.Date)+#39;
  form28.ADOQuery1.Open;
 end
 else if (form28.RadioButton7.Checked = true) then
 begin
  form28.ADOQuery1.Close;
  form28.ADOQuery1.SQL.Text:='select * from vaccinate where next_vaccinatedate >
'+#39+datetostr(form28.DateTimePicker3.Date)+#39;
  form28.ADOQuery1.Open;
 end
 else if (form28.RadioButton8.Checked = true) then
 begin
  form28.ADOQuery1.Close;
  form28.ADOQuery1.SQL.Text:='select * from vaccinate where next_vaccinatedate =
'+#39+datetostr(form28.DateTimePicker3.Date)+#39;
  form28.ADOQuery1.Open;
 end
 else if (form28.RadioButton6.Checked = true) then
 begin
  form28.ADOQuery1.Close;
  form28.ADOQuery1.SQL.Text:='select * from vaccinate where next_vaccinatedate
between '+#39+datetostr(form28.DateTimePicker3.Date)+#39+' and
'+#39+datetostr(form28.DateTimePicker3.Date)+#39;
  form28.ADOQuery1.Open;
 end
 else
  showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm28.LbSpeedButton2Click(Sender: TObject);
begin
 FORM28.DateTimePicker3.Date:=DATE;
 FORM28.DateTimePicker4.Date:=DATE;
 FORM28.DateTimePicker4.Enabled:=FALSE;
 FORM28.RadioButton5.Checked:=FALSE;
 FORM28.RadioButton6.Checked:=FALSE;
 FORM28.RadioButton7.Checked:=FALSE;
 FORM28.RadioButton8.Checked:=FALSE;
 FORM28.ADOQuery1.Close;
end;

procedure TForm28.Edit3Change(Sender: TObject);
begin
```

```
 form28.ADOQuery1.Close;
 form28.ADOQuery1.SQL.Text:='select * from vaccinate where vaccine_name
like'+#39+'%'+form28.Edit3.Text+'%'+#39;
 form28.ADOQuery1.Open;
end;

procedure TForm28.Edit2Change(Sender: TObject);
begin
 form28.ADOQuery1.Close;
 form28.ADOQuery1.SQL.Text:='select * from vaccinate where vaccine_serialno
like'+#39+'%'+form28.Edit2.Text+'%'+#39;
 form28.ADOQuery1.Open;
end;

procedure TForm28.Edit1Change(Sender: TObject);
begin
 form28.ADOQuery1.Close;
 form28.ADOQuery1.SQL.Text:='select * from vaccinate where animal_id
like'+#39+'%'+form28.Edit1.Text+'%'+#39;
 form28.ADOQuery1.Open;
end;

procedure TForm28.TabSheet7Show(Sender: TObject);
begin
 form28.ADOQuery1.Close;
 form28.ADOQuery1.SQL.Text:='select * from vaccinate';
 form28.ADOQuery1.Open;
end;

procedure TForm28.LbSpeedButton4Click(Sender: TObject);
begin
form28.Edit3.Clear;
form28.ADOQuery1.Close;
end;

procedure TForm28.SpeedButton6Click(Sender: TObject);
begin
form28.Edit2.Clear;
form28.ADOQuery1.Close;
end;

procedure TForm28.SpeedButton4Click(Sender: TObject);
begin
form28.Edit1.Clear;
form28.ADOQuery1.Close;
end;

procedure TForm28.TabSheet1Show(Sender: TObject);
begin
  Form28.SpeedButton2.Click;
```

```
end;

procedure TForm28.TabSheet2Show(Sender: TObject);
begin
Form28.LbSpeedButton2.Click;
end;

procedure TForm28.TabSheet4Show(Sender: TObject);
begin
Form28.LbSpeedButton4.Click;
end;

procedure TForm28.TabSheet5Show(Sender: TObject);
begin
Form28.SpeedButton6.Click;
end;

procedure TForm28.TabSheet3Show(Sender: TObject);
begin
Form28.SpeedButton4.Click;
end;

procedure TForm28.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form28.Edit1.Clear;
  form28.Edit2.Clear;
  form28.Edit3.Clear;
  form28.ADOQuery1.Close;
  form28.RadioButton1.Checked:=false;
  form28.RadioButton2.Checked:=false;
  form28.RadioButton3.Checked:=false;
  form28.RadioButton4.Checked:=false;
  form28.RadioButton5.Checked:=false;
  form28.RadioButton6.Checked:=false;
  form28.RadioButton7.Checked:=false;
  form28.RadioButton8.Checked:=false;
  form28.DateTimePicker2.Enabled:=false;
  form28.DateTimePicker4.Enabled:=false;
end;

procedure TForm28.FormCreate(Sender: TObject);
begin
form28.DateTimePicker1.Date:=date;
form28.DateTimePicker2.Date:=date;
form28.DateTimePicker3.Date:=date;
form28.DateTimePicker4.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';
end;
```

```
procedure TForm28.FormHide(Sender: TObject);
begin
  form28.Edit1.Clear;
  form28.Edit2.Clear;
  form28.Edit3.Clear;
  form28.ADOQuery1.Close;
  form28.RadioButton1.Checked:=false;
  form28.RadioButton2.Checked:=false;
  form28.RadioButton3.Checked:=false;
  form28.RadioButton4.Checked:=false;
  form28.RadioButton5.Checked:=false;
  form28.RadioButton6.Checked:=false;
  form28.RadioButton7.Checked:=false;
  form28.RadioButton8.Checked:=false;
  form28.DateTimePicker2.Enabled:=false;
  form28.DateTimePicker4.Enabled:=false;
end;

end.
```

FORM 29 CODES

```
unit Unit29;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Grids, DBGrids, StdCtrls, LbSpeedButton, ExtCtrls,
  ComCtrls, Buttons, DB, ADODB;

type
  TForm29 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Panel2: TPanel;
    TabSheet2: TTabSheet;
    Label2: TLabel;
```

```
LbSpeedButton1: TLbSpeedButton;
LbSpeedButton2: TLbSpeedButton;
Panel5: TPanel;
DateTimePicker3: TDateTimePicker;
DateTimePicker4: TDateTimePicker;
RadioButton5: TRadioButton;
RadioButton6: TRadioButton;
RadioButton7: TRadioButton;
RadioButton8: TRadioButton;
TabSheet3: TTabSheet;
Label3: TLabel;
SpeedButton4: TSpeedButton;
Panel3: TPanel;
Edit1: TEdit;
TabSheet4: TTabSheet;
Label4: TLabel;
LbSpeedButton4: TLbSpeedButton;
Panel4: TPanel;
TabSheet7: TTabSheet;
Panel7: TPanel;
DBGrid1: TDBGrid;
Panel1: TPanel;
MainMenu1: TMainMenu;
F1: TMenuItem;
Edit2: TEdit;
ADOQuery1: TADOQuery;
DataSource1: TDataSource;
procedure RadioButton1Click(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure RadioButton3Click(Sender: TObject);
procedure RadioButton4Click(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure SpeedButton2Click(Sender: TObject);
procedure RadioButton5Click(Sender: TObject);
procedure RadioButton7Click(Sender: TObject);
procedure RadioButton8Click(Sender: TObject);
procedure RadioButton6Click(Sender: TObject);
procedure LbSpeedButton1Click(Sender: TObject);
procedure LbSpeedButton2Click(Sender: TObject);
procedure TabSheet1Show(Sender: TObject);
procedure TabSheet2Show(Sender: TObject);
procedure SpeedButton4Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure LbSpeedButton4Click(Sender: TObject);
procedure TabSheet3Show(Sender: TObject);
procedure TabSheet4Show(Sender: TObject);
procedure TabSheet7Show(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
```

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

var
 Form29: TForm29;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm29.RadioButton1Click(Sender: TObject);
begin
 form29.DateTimePicker2.Date:=date;
 form29.DateTimePicker2.Enabled:=false;
end;

procedure TForm29.RadioButton2Click(Sender: TObject);
begin
 form29.DateTimePicker2.Date:=date;
 form29.DateTimePicker2.Enabled:=false;
end;

procedure TForm29.RadioButton3Click(Sender: TObject);
begin
 form29.DateTimePicker2.Date:=date;
 form29.DateTimePicker2.Enabled:=false;
end;

procedure TForm29.RadioButton4Click(Sender: TObject);
begin
if form29.RadioButton4.Checked = true then
 form29.DateTimePicker2.Enabled:=true
else
 form29.DateTimePicker2.Enabled:=false;
end;

procedure TForm29.SpeedButton1Click(Sender: TObject);
begin
if (form29.RadioButton1.Checked = true) then
 begin
  form29.ADOQuery1.Close;
  form29.ADOQuery1.SQL.Text:='select * from opdrug where op_drugdate <
'+#39+datetostr(form29.DateTimePicker1.Date)+#39;
  form29.ADOQuery1.Open;
```

```
 end
 else if (form29.RadioButton2.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op_drugdate >
'+#39+datetostr(form29.DateTimePicker1.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else if (form29.RadioButton3.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op_drugdate =
'+#39+datetostr(form29.DateTimePicker1.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else if (form29.RadioButton4.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op_drugdate between
'+#39+datetostr(form29.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form29.DateTimePicker2.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else
   showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm29.SpeedButton2Click(Sender: TObject);
begin
  FORM29.DateTimePicker1.Date:=DATE;
  FORM29.DateTimePicker2.Date:=DATE;
  FORM29.DateTimePicker2.Enabled:=FALSE;
  FORM29.RadioButton1.Checked:=FALSE;
  FORM29.RadioButton2.Checked:=FALSE;
  FORM29.RadioButton3.Checked:=FALSE;
  FORM29.RadioButton4.Checked:=FALSE;
  FORM29.ADOQuery1.Close;
end;

procedure TForm29.RadioButton5Click(Sender: TObject);
begin
  form29.DateTimePicker4.Date:=date;
  form29.DateTimePicker4.Enabled:=false;
end;

procedure TForm29.RadioButton7Click(Sender: TObject);
begin
  form29.DateTimePicker4.Date:=date;
  form29.DateTimePicker4.Enabled:=false;
end;
```

```
procedure TForm29.RadioButton8Click(Sender: TObject);
begin
 form29.DateTimePicker4.Date:=date;
 form29.DateTimePicker4.Enabled:=false;
end;

procedure TForm29.RadioButton6Click(Sender: TObject);
begin
if form29.RadioButton6.Checked = true then
 form29.DateTimePicker4.Enabled:=true
else
 form29.DateTimePicker4.Enabled:=false;
end;

procedure TForm29.LbSpeedButton1Click(Sender: TObject);
begin
if (form29.RadioButton5.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op_nextdrugdate <
'+#39+datetostr(form29.DateTimePicker3.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else if (form29.RadioButton7.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op_nextdrugdate >
'+#39+datetostr(form29.DateTimePicker3.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else if (form29.RadioButton8.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op  nextdrugdate =
'+#39+datetostr(form29.DateTimePicker3.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else if (form29.RadioButton6.Checked = true) then
 begin
   form29.ADOQuery1.Close;
   form29.ADOQuery1.SQL.Text:='select * from opdrug where op_nextdrugdate
between '+#39+datetostr(form29.DateTimePicker3.Date)+#39+' and
'+#39+datetostr(form29.DateTimePicker3.Date)+#39;
   form29.ADOQuery1.Open;
 end
 else
   showmessage('PLEASE SELECT SEARCH CRITERIA');
end;
```

```
procedure TForm29.LbSpeedButton2Click(Sender: TObject);
begin
 FORM29.DateTimePicker3.Date:=DATE;
 FORM29.DateTimePicker4.Date:=DATE;
 FORM29.DateTimePicker4.Enabled:=FALSE;
 FORM29.RadioButton5.Checked:=FALSE;
 FORM29.RadioButton6.Checked:=FALSE;
 FORM29.RadioButton7.Checked:=FALSE;
 FORM29.RadioButton8.Checked:=FALSE;
 FORM29.ADOQuery1.Close;
end;

procedure TForm29.TabSheet1Show(Sender: TObject);
begin
Form29.SpeedButton2.Click;
end;

procedure TForm29.TabSheet2Show(Sender: TObject);
begin
Form29.LbSpeedButton2.Click;
end;

procedure TForm29.SpeedButton4Click(Sender: TObject);
begin
form29.Edit1.Clear;
form29.ADOQuery1.Close;
end;

procedure TForm29.Edit1Change(Sender: TObject);
begin
 form29.ADOQuery1.Close;
 form29.ADOQuery1.SQL.Text:='select * from opdrug where animal_id
like'+#39+'%'+form29.Edit1.Text+'%'+#39;
 form29.ADOQuery1.Open;
end;

procedure TForm29.Edit2Change(Sender: TObject);
begin
 form29.ADOQuery1.Close;
 form29.ADOQuery1.SQL.Text:='select * from opdrug where op_drugname
like'+#39+'%'+form29.Edit2.Text+'%'+#39;
 form29.ADOQuery1.Open;
end;

procedure TForm29.LbSpeedButton4Click(Sender: TObject);
begin
form29.Edit2.Clear;
form29.ADOQuery1.Close;
end;
```

```
procedure TForm29.TabSheet3Show(Sender: TObject);
begin
Form29.SpeedButton4.Click;
end;

procedure TForm29.TabSheet4Show(Sender: TObject);
begin
Form29.LbSpeedButton4.Click;
end;

procedure TForm29.TabSheet7Show(Sender: TObject);
begin
  form29.ADOQuery1.Close;
  form29.ADOQuery1.SQL.Text:='select * from opdrug';
  form29.ADOQuery1.Open;
end;

procedure TForm29.FormCreate(Sender: TObject);
begin
form29.DateTimePicker1.Date:=date;
form29.DateTimePicker2.Date:=date;
form29.DateTimePicker3.Date:=date;
form29.DateTimePicker4.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarýný MySql database sisteminin
anlayacaðý þekle dönüþtürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm29.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form29.Edit1.Clear;
  form29.Edit2.Clear;
  form29.ADOQuery1.Close;
  form29.RadioButton1.Checked:=false;
  form29.RadioButton2.Checked:=false;
  form29.RadioButton3.Checked:=false;
  form29.RadioButton4.Checked:=false;
  form29.RadioButton5.Checked:=false;
  form29.RadioButton6.Checked:=false;
  form29.RadioButton7.Checked:=false;
  form29.RadioButton8.Checked:=false;
  form29.DateTimePicker2.Enabled:=false;
  form29.DateTimePicker4.Enabled:=false;
end;

procedure TForm29.FormHide(Sender: TObject);
begin
  form29.Edit1.Clear;
  form29.Edit2.Clear;
  form29.ADOQuery1.Close;
```

```
 form29.RadioButton1.Checked:=false;
 form29.RadioButton2.Checked:=false;
 form29.RadioButton3.Checked:=false;
 form29.RadioButton4.Checked:=false;
 form29.RadioButton5.Checked:=false;
 form29.RadioButton6.Checked:=false;
 form29.RadioButton7.Checked:=false;
 form29.RadioButton8.Checked:=false;
 form29.DateTimePicker2.Enabled:=false;
 form29.DateTimePicker4.Enabled:=false;
end;

end.
```

FORM 30 CODES

```
unit Unit30;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Menus, Grids, DBGrids, ExtCtrls, Buttons, ComCtrls,
  DB, ADODB;

type
  TForm30 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton2: TSpeedButton;
    Edit1: TEdit;
    Panel5: TPanel;
    TabSheet2: TTabSheet;
    Label2: TLabel;
    SpeedButton4: TSpeedButton;
    Panel2: TPanel;
    Edit2: TEdit;
    TabSheet3: TTabSheet;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    TabSheet4: TTabSheet;
    Panel3: TPanel;
    Panel4: TPanel;
    Label3: TLabel;
    ComboBox1: TComboBox;
```

```
  SpeedButton6: TSpeedButton;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  procedure Edit1Change(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure ComboBox1Change(Sender: TObject);
  procedure TabSheet4Show(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure SpeedButton4Click(Sender: TObject);
  procedure SpeedButton6Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure TabSheet3Show(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 Form30: TForm30;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm30.Edit1Change(Sender: TObject);
begin
  form30.ADOQuery1.Close;
  form30.ADOQuery1.SQL.Text:='select * from drugs where drug_name
like'+#39+'%'+form30.Edit1.Text+'%'+#39;
  form30.ADOQuery1.Open;
end;

procedure TForm30.Edit2Change(Sender: TObject);
begin
  form30.ADOQuery1.Close;
  form30.ADOQuery1.SQL.Text:='select * from drugs where drug_id
like'+#39+'%'+form30.Edit2.Text+'%'+#39;
  form30.ADOQuery1.Open;
 end;

 procedure TForm30.ComboBox1Change(Sender: TObject);
 begin
   form30.ADOQuery1.Close;
```

```
 form30.ADOQuery1.SQL.Text:='select * from drugs where drug_kind
like'+#39+'%'+form30.ComboBox1.Text+'%'+#39;
 form30.ADOQuery1.Open;
end;

procedure TForm30.TabSheet4Show(Sender: TObject);
begin
 form30.ADOQuery1.Close;
 form30.ADOQuery1.SQL.Text:='select * from drugs';
 form30.ADOQuery1.Open;
end;

procedure TForm30.SpeedButton2Click(Sender: TObject);
begin
form30.Edit1.Clear;
form30.ADOQuery1.Close;
end;

procedure TForm30.SpeedButton4Click(Sender: TObject);
begin
form30.Edit2.Clear;
form30.ADOQuery1.Close;
end;

procedure TForm30.SpeedButton6Click(Sender: TObject);
begin
form30.ComboBox1.Text:='Select One';
form30.ADOQuery1.Close;
end;

procedure TForm30.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form30.Edit1.Clear;
  form30.Edit2.Clear;
  form30.ComboBox1.Text:='Select One';
  form30.ADOQuery1.Close;
end;

procedure TForm30.FormHide(Sender: TObject);
begin
  form30.Edit1.Clear;
  form30.Edit2.Clear;
  form30.ComboBox1.Text:='Select One';
  form30.ADOQuery1.Close;
end;

procedure TForm30.TabSheet1Show(Sender: TObject);
begin
Form30.SpeedButton2.Click;
end;
```

```
procedure TForm30.TabSheet2Show(Sender: TObject);
begin
Form30.SpeedButton4.Click;
end;

procedure TForm30.TabSheet3Show(Sender: TObject);
begin
Form30.SpeedButton6.Click;
end;

end.
```

FORM 31 CODES

```
unit Unit31;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Grids, DBGrids, ExtCtrls, StdCtrls, Buttons, ComCtrls,
  DB, ADODB;

type
  TForm31 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton2: TSpeedButton;
    Edit1: TEdit;
    Panel5: TPanel;
    TabSheet2: TTabSheet;
    Label2: TLabel;
    SpeedButton4: TSpeedButton;
    Panel2: TPanel;
    Edit2: TEdit;
    TabSheet3: TTabSheet;
    Panel3: TPanel;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    procedure Edit2Change(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure TabSheet3Show(Sender: TObject);
```

```
  procedure SpeedButton2Click(Sender: TObject);
  procedure SpeedButton4Click(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure FormHide(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form31: TForm31;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm31.Edit2Change(Sender: TObject);
begin
 form31.ADOQuery1.Close;
 form31.ADOQuery1.SQL.Text:='select * from operations where operation_id
like'+#39+'%'+form31.Edit2.Text+'%'+#39;
 form31.ADOQuery1.Open;
end;

procedure TForm31.Edit1Change(Sender: TObject);
begin
 form31.ADOQuery1.Close;
 form31.ADOQuery1.SQL.Text:='select * from operations where operation_name
like'+#39+'%'+form31.Edit1.Text+'%'+#39;
 form31.ADOQuery1.Open;
end;

procedure TForm31.TabSheet3Show(Sender: TObject);
begin
 form31.ADOQuery1.Close;
 form31.ADOQuery1.SQL.Text:='select * from operations';
 form31.ADOQuery1.Open;
end;

procedure TForm31.SpeedButton2Click(Sender: TObject);
begin
form31.Edit1.Clear;
form31.ADOQuery1.Close;
end;
```

```
procedure TForm31.SpeedButton4Click(Sender: TObject);
begin
form31.Edit2.Clear;
form31.ADOQuery1.Close;
end;

procedure TForm31.TabSheet1Show(Sender: TObject);
begin
Form31.SpeedButton2.Click;
end;

procedure TForm31.TabSheet2Show(Sender: TObject);
begin
Form31.SpeedButton4.Click;
end;

procedure TForm31.FormHide(Sender: TObject);
begin
  form31.Edit1.Clear;
  form31.Edit2.Clear;
  form31.ADOQuery1.Close;
end;

procedure TForm31.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form31.Edit1.Clear;
  form31.Edit2.Clear;
  form31.ADOQuery1.Close;
end;

end.
```

FORM 32 CODES

```
unit Unit32;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Grids, DBGrids, StdCtrls, LbSpeedButton, ExtCtrls,
  ComCtrls, Buttons, DB, ADODB;

type
  TForm32 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
```

```
SpeedButton1: TSpeedButton;
SpeedButton2: TSpeedButton;
RadioButton1: TRadioButton;
RadioButton2: TRadioButton;
RadioButton3: TRadioButton;
RadioButton4: TRadioButton;
DateTimePicker1: TDateTimePicker;
DateTimePicker2: TDateTimePicker;
Panel2: TPanel;
TabSheet3: TTabSheet;
Label3: TLabel;
SpeedButton4: TSpeedButton;
Panel3: TPanel;
Edit1: TEdit;
TabSheet4: TTabSheet;
Label4: TLabel;
LbSpeedButton4: TLbSpeedButton;
Panel4: TPanel;
TabSheet7: TTabSheet;
Panel7: TPanel;
DBGrid1: TDBGrid;
Panel1: TPanel;
MainMenu1: TMainMenu;
F1: TMenuItem;
Edit2: TEdit;
ADOQuery1: TADOQuery;
DataSource1: TDataSource;
procedure RadioButton1Click(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure RadioButton3Click(Sender: TObject);
procedure RadioButton4Click(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure SpeedButton2Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure TabSheet7Show(Sender: TObject);
procedure SpeedButton4Click(Sender: TObject);
procedure LbSpeedButton4Click(Sender: TObject);
procedure TabSheet1Show(Sender: TObject);
procedure TabSheet3Show(Sender: TObject);
procedure TabSheet4Show(Sender: TObject);
procedure FormHide(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

```
var
  Form32: TForm32;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm32.RadioButton1Click(Sender: TObject);
begin
  form32.DateTimePicker2.Date:=date;
  form32.DateTimePicker2.Enabled:=false;
end;

procedure TForm32.RadioButton2Click(Sender: TObject);
begin
  form32.DateTimePicker2.Date:=date;
  form32.DateTimePicker2.Enabled:=false;
end;

procedure TForm32.RadioButton3Click(Sender: TObject);
begin
  form32.DateTimePicker2.Date:=date;
  form32.DateTimePicker2.Enabled:=false;
end;

procedure TForm32.RadioButton4Click(Sender: TObject);
begin
if form32.RadioButton4.Checked = true then
  form32.DateTimePicker2.Enabled:=true
else
  form32.DateTimePicker2.Enabled:=false;
end;

procedure TForm32.SpeedButton1Click(Sender: TObject);
begin
if (form32.RadioButton1.Checked = true) then
  begin
    form32.ADOQuery1.Close;
    form32.ADOQuery1.SQL.Text:='select * from medicinate where medicinate_date <
'+#39+datetostr(form32.DateTimePicker1.Date)+#39;
    form32.ADOQuery1.Open;
  end
  else if (form32.RadioButton2.Checked = true) then
  begin
    form32.ADOQuery1.Close;
    form32.ADOQuery1.SQL.Text:='select * from medicinate where medicinate_date >
'+#39+datetostr(form32.DateTimePicker1.Date)+#39;
    form32.ADOQuery1.Open;
```

```
 end
 else if (form32.RadioButton3.Checked = true) then
 begin
  form32.ADOQuery1.Close;
  form32.ADOQuery1.SQL.Text:='select * from medicinate where medicinate_date =
'+#39+datetostr(form32.DateTimePicker1.Date)+#39;
  form32.ADOQuery1.Open;
 end
 else if (form32.RadioButton4.Checked = true) then
 begin
  form32.ADOQuery1.Close;
  form32.ADOQuery1.SQL.Text:='select * from medicinate where medicinate_date
between '+#39+datetostr(form32.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form32.DateTimePicker2.Date)+#39;
  form32.ADOQuery1.Open;
 end
 else
  showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm32.SpeedButton2Click(Sender: TObject);
begin
 FORM32.DateTimePicker1.Date:=DATE;
 FORM32.DateTimePicker2.Date:=DATE;
 FORM32.DateTimePicker2.Enabled:=FALSE;
 FORM32.RadioButton1.Checked:=FALSE;
 FORM32.RadioButton2.Checked:=FALSE;
 FORM32.RadioButton3.Checked:=FALSE;
 FORM32.RadioButton4.Checked:=FALSE;
 FORM32.ADOQuery1.Close;
end;

procedure TForm32.Edit1Change(Sender: TObject);
begin
 form32.ADOQuery1.Close;
 form32.ADOQuery1.SQL.Text:='select * from medicinate where animal_id
like'+#39+'%'+form32.Edit1.Text+'%'+#39;
 form32.ADOQuery1.Open;
end;

procedure TForm32.Edit2Change(Sender: TObject);
begin
 form32.ADOQuery1.Close;
 form32.ADOQuery1.SQL.Text:='select * from medicinate where drug_name
like'+#39+'%'+form32.Edit2.Text+'%'+#39;
 form32.ADOQuery1.Open;
end;

procedure TForm32.TabSheet7Show(Sender: TObject);
begin
```

```
 form32.ADOQuery1.Close;
 form32.ADOQuery1.SQL.Text:='select * from medicinate';
 form32.ADOQuery1.Open;
end;

procedure TForm32.SpeedButton4Click(Sender: TObject);
begin
form32.Edit1.Clear;
form32.ADOQuery1.Close;
end;

procedure TForm32.LbSpeedButton4Click(Sender: TObject);
begin
form32.Edit2.Clear;
form32.ADOQuery1.Close;
end;

procedure TForm32.TabSheet1Show(Sender: TObject);
begin
Form32.SpeedButton2.Click;
end;

procedure TForm32.TabSheet3Show(Sender: TObject);
begin
Form32.SpeedButton4.Click;
end;

procedure TForm32.TabSheet4Show(Sender: TObject);
begin
Form32.LbSpeedButton4.Click;
end;

procedure TForm32.FormHide(Sender: TObject);
begin
  form32.Edit1.Clear;
  form32.Edit2.Clear;
  form32.ADOQuery1.Close;
  form32.RadioButton1.Checked:=false;
  form32.RadioButton2.Checked:=false;
  form32.RadioButton3.Checked:=false;
  form32.RadioButton4.Checked:=false;
  form32.DateTimePicker2.Enabled:=false;
end;

procedure TForm32.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form32.Edit1.Clear;
  form32.Edit2.Clear;
  form32.ADOQuery1.Close;
  form32.RadioButton1.Checked:=false;
```

```
 form32.RadioButton2.Checked:=false;
 form32.RadioButton3.Checked:=false;
 form32.RadioButton4.Checked:=false;
 form32.DateTimePicker2.Enabled:=false;
end;

procedure TForm32.FormCreate(Sender: TObject);
begin
form32.DateTimePicker1.Date:=date;
form32.DateTimePicker2.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarini MySql database sisteminin
anlayacagi sekle dönüstürdüm...
shortdateformat := 'yyyy/m/d';
end;

end.
```

FORM 33 CODES

```
unit Unit33;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Grids, DBGrids, StdCtrls, LbSpeedButton, ExtCtrls,
  ComCtrls, Buttons, DB, ADODB;

type
  TForm33 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    DateTimePicker1: TDateTimePicker;
    DateTimePicker2: TDateTimePicker;
    Panel2: TPanel;
    TabSheet3: TTabSheet;
    Label3: TLabel;
    SpeedButton4: TSpeedButton;
    Panel3: TPanel;
    Edit1: TEdit;
    TabSheet4: TTabSheet;
```

```
  Label4: TLabel;
  LbSpeedButton4: TLbSpeedButton;
  Panel4: TPanel;
  TabSheet7: TTabSheet;
  DBGrid1: TDBGrid;
  Panel1: TPanel;
  MainMenu1: TMainMenu;
  F1: TMenuItem;
  TabSheet2: TTabSheet;
  Panel7: TPanel;
  Panel5: TPanel;
  Label2: TLabel;
  Edit2: TEdit;
  LbSpeedButton2: TLbSpeedButton;
  Edit3: TEdit;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure Edit3Change(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure LbSpeedButton4Click(Sender: TObject);
  procedure SpeedButton4Click(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
  procedure TabSheet3Show(Sender: TObject);
  procedure TabSheet4Show(Sender: TObject);
  procedure TabSheet7Show(Sender: TObject);
  procedure TabSheet2Show(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
  procedure RadioButton1Click(Sender: TObject);
  procedure RadioButton2Click(Sender: TObject);
  procedure RadioButton3Click(Sender: TObject);
  procedure RadioButton4Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form33: TForm33;

implementation

uses Unit10;
```

```
{$R *.dfm}

procedure TForm33.SpeedButton1Click(Sender: TObject);
begin
if (form33.RadioButton1.Checked = true) then
 begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where
operation_date < '+#39+datetostr(form33.DateTimePicker1.Date)+#39;
  form33.ADOQuery1.Open;
 end
 else if (form33.RadioButton2.Checked = true) then
 begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where
operation_date > '+#39+datetostr(form33.DateTimePicker1.Date)+#39;
  form33.ADOQuery1.Open;
 end
 else if (form33.RadioButton3.Checked = true) then
 begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where
operation_date = '+#39+datetostr(form33.DateTimePicker1.Date)+#39;
  form33.ADOQuery1.Open;
 end
 else if (form33.RadioButton4.Checked = true) then
 begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where
operation_date between '+#39+datetostr(form33.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form33.DateTimePicker2.Date)+#39;
  form33.ADOQuery1.Open;
 end
 else
  showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm33.SpeedButton2Click(Sender: TObject);
begin
 FORM33.DateTimePicker1.Date:=DATE;
 FORM33.DateTimePicker2.Date:=DATE;
 FORM33.DateTimePicker2.Enabled:=FALSE;
 FORM33.RadioButton1.Checked:=FALSE;
 FORM33.RadioButton2.Checked:=FALSE;
 FORM33.RadioButton3.Checked:=FALSE;
 FORM33.RadioButton4.Checked:=FALSE;
 FORM33.ADOQuery1.Close;
end;

procedure TForm33.Edit1Change(Sender: TObject);
```

```
begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where animal_id
like'+#39+'%'+form33.Edit1.Text+'%'+#39;
  form33.ADOQuery1.Open;
end;

procedure TForm33.Edit3Change(Sender: TObject);
begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where drug_name
like'+#39+'%'+form33.Edit3.Text+'%'+#39;
  form33.ADOQuery1.Open;
end;

procedure TForm33.Edit2Change(Sender: TObject);
begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation where
operation_name like'+#39+'%'+form33.Edit2.Text+'%'+#39;
  form33.ADOQuery1.Open;
end;

procedure TForm33.LbSpeedButton2Click(Sender: TObject);
begin
form33.Edit2.Clear;
form33.ADOQuery1.Close;
end;

procedure TForm33.LbSpeedButton4Click(Sender: TObject);
begin
form33.Edit3.Clear;
form33.ADOQuery1.Close;
end;

procedure TForm33.SpeedButton4Click(Sender: TObject);
begin
form33.Edit1.Clear;
form33.ADOQuery1.Close;
end;

procedure TForm33.TabSheet1Show(Sender: TObject);
begin
Form33.SpeedButton2.Click;
end;

procedure TForm33.TabSheet3Show(Sender: TObject);
begin
Form33.SpeedButton4.Click;
end;
```

```
procedure TForm33.TabSheet4Show(Sender: TObject);
begin
Form33.LbSpeedButton4.Click;
end;

procedure TForm33.TabSheet7Show(Sender: TObject);
begin
Form33.LbSpeedButton2.Click;
end;

procedure TForm33.TabSheet2Show(Sender: TObject);
begin
  form33.ADOQuery1.Close;
  form33.ADOQuery1.SQL.Text:='select * from appliedoperation';
  form33.ADOQuery1.Open;
end;

procedure TForm33.FormCreate(Sender: TObject);
begin
form33.DateTimePicker1.Date:=date;
form33.DateTimePicker2.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarını MySql database sisteminin
anlayacağı şekle dönüştürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm33.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form33.Edit1.Clear;
  form33.Edit2.Clear;
  form33.ADOQuery1.Close;
  form33.RadioButton1.Checked:=false;
  form33.RadioButton2.Checked:=false;
  form33.RadioButton3.Checked:=false;
  form33.RadioButton4.Checked:=false;
  form33.DateTimePicker2.Enabled:=false;
end;

procedure TForm33.FormHide(Sender: TObject);
begin
  form33.Edit1.Clear;
  form33.Edit2.Clear;
  form33.ADOQuery1.Close;
  form33.RadioButton1.Checked:=false;
  form33.RadioButton2.Checked:=false;
  form33.RadioButton3.Checked:=false;
  form33.RadioButton4.Checked:=false;
  form33.DateTimePicker2.Enabled:=false;
end;
```

```
procedure TForm33.RadioButton1Click(Sender: TObject);
begin
 form33.DateTimePicker2.Date:=date;
 form33.DateTimePicker2.Enabled:=false;
end;

procedure TForm33.RadioButton2Click(Sender: TObject);
begin
 form33.DateTimePicker2.Date:=date;
 form33.DateTimePicker2.Enabled:=false;
end;

procedure TForm33.RadioButton3Click(Sender: TObject);
begin
 form33.DateTimePicker2.Date:=date;
 form33.DateTimePicker2.Enabled:=false;
end;

procedure TForm33.RadioButton4Click(Sender: TObject);
begin
if form33.RadioButton4.Checked = true then
 form33.DateTimePicker2.Enabled:=true
else
 form33.DateTimePicker2.Enabled:=false;
end;

end.
```

FORM 34 CODES

```
unit Unit34;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Menus, Grids, DBGrids, LbSpeedButton, ExtCtrls,
  ComCtrls, Buttons, DB, ADODB;

type
  TForm34 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    RadioButton1: TRadioButton;
```

```
RadioButton2: TRadioButton;
RadioButton3: TRadioButton;
RadioButton4: TRadioButton;
DateTimePicker1: TDateTimePicker;
DateTimePicker2: TDateTimePicker;
Panel2: TPanel;
TabSheet3: TTabSheet;
Label3: TLabel;
SpeedButton4: TSpeedButton;
Panel3: TPanel;
Edit1: TEdit;
TabSheet4: TTabSheet;
Label4: TLabel;
LbSpeedButton4: TLbSpeedButton;
Panel4: TPanel;
TabSheet7: TTabSheet;
Label2: TLabel;
LbSpeedButton2: TLbSpeedButton;
Panel5: TPanel;
Edit2: TEdit;
TabSheet2: TTabSheet;
Panel7: TPanel;
DBGrid1: TDBGrid;
Panel1: TPanel;
MainMenu1: TMainMenu;
F1: TMenuItem;
Memo1: TMemo;
ADOQuery1: TADOQuery;
DataSource1: TDataSource;
procedure RadioButton1Click(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure RadioButton3Click(Sender: TObject);
procedure RadioButton4Click(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure SpeedButton2Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Memo1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure TabSheet2Show(Sender: TObject);
procedure SpeedButton4Click(Sender: TObject);
procedure LbSpeedButton4Click(Sender: TObject);
procedure LbSpeedButton2Click(Sender: TObject);
procedure TabSheet1Show(Sender: TObject);
procedure TabSheet3Show(Sender: TObject);
procedure TabSheet4Show(Sender: TObject);
procedure TabSheet7Show(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormHide(Sender: TObject);
private
```

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

var
 Form34: TForm34;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm34.RadioButton1Click(Sender: TObject);
begin
 form34.DateTimePicker2.Date:=date;
 form34.DateTimePicker2.Enabled:=false;
end;

procedure TForm34.RadioButton2Click(Sender: TObject);
begin
 form34.DateTimePicker2.Date:=date;
 form34.DateTimePicker2.Enabled:=false;
end;

procedure TForm34.RadioButton3Click(Sender: TObject);
begin
 form34.DateTimePicker2.Date:=date;
 form34.DateTimePicker2.Enabled:=false;
end;

procedure TForm34.RadioButton4Click(Sender: TObject);
begin
if form34.RadioButton4.Checked = true then
 form34.DateTimePicker2.Enabled:=true
else
 form34.DateTimePicker2.Enabled:=false;
end;

procedure TForm34.SpeedButton1Click(Sender: TObject);
begin
if (form34.RadioButton1.Checked = true) then
 begin
  form34.ADOQuery1.Close;
  form34.ADOQuery1.SQL.Text:='select * from illnesses where date <
'+#39+datetostr(form34.DateTimePicker1.Date)+#39;
  form34.ADOQuery1.Open;
 end
 else if (form34.RadioButton2.Checked = true) then
```

```
 begin
   form34.ADOQuery1.Close;
   form34.ADOQuery1.SQL.Text:='select * from illnesses where date >
'+#39+datetostr(form34.DateTimePicker1.Date)+#39;
   form34.ADOQuery1.Open;
 end
 else if (form34.RadioButton3.Checked = true) then
 begin
   form34.ADOQuery1.Close;
   form34.ADOQuery1.SQL.Text:='select * from illnesses where date =
'+#39+datetostr(form34.DateTimePicker1.Date)+#39;
   form34.ADOQuery1.Open;
 end
 else if (form34.RadioButton4.Checked = true) then
 begin
   form34.ADOQuery1.Close;
   form34.ADOQuery1.SQL.Text:='select * from illnesses where date between
'+#39+datetostr(form34.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form34.DateTimePicker2.Date)+#39;
   form34.ADOQuery1.Open;
 end
 else
   showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm34.SpeedButton2Click(Sender: TObject);
begin
 FORM34.DateTimePicker1.Date:=DATE;
 FORM34.DateTimePicker2.Date:=DATE;
 FORM34.DateTimePicker2.Enabled:=FALSE;
 FORM34.RadioButton1.Checked:=FALSE;
 FORM34.RadioButton2.Checked:=FALSE;
 FORM34.RadioButton3.Checked:=FALSE;
 FORM34.RadioButton4.Checked:=FALSE;
 FORM34.ADOQuery1.Close;
end;

procedure TForm34.Edit1Change(Sender: TObject);
begin
 form34.ADOQuery1.Close;
 form34.ADOQuery1.SQL.Text:='select * from illnesses where animal_id
like'+#39+'%'+form34.Edit1.Text+'%'+#39;
 form34.ADOQuery1.Open;
end;

procedure TForm34.Memo1Change(Sender: TObject);
begin
 form34.ADOQuery1.Close;
 form34.ADOQuery1.SQL.Text:='select * from illnesses where illness
like'+#39+'%'+form34.memo1.Text+'%'+#39;
```

```
 form34.ADOQuery1.Open;
end;

procedure TForm34.Edit2Change(Sender: TObject);
begin
 form34.ADOQuery1.Close;
 form34.ADOQuery1.SQL.Text:='select * from illnesses where protocol_no
like'+#39+'%'+form34.Edit2.Text+'%'+#39;
 form34.ADOQuery1.Open;
end;

procedure TForm34.TabSheet2Show(Sender: TObject);
begin
 form34.ADOQuery1.Close;
 form34.ADOQuery1.SQL.Text:='select * from illnesses';
 form34.ADOQuery1.Open;
end;

procedure TForm34.SpeedButton4Click(Sender: TObject);
begin
form34.Edit1.Clear;
form34.ADOQuery1.Close;
end;

procedure TForm34.LbSpeedButton4Click(Sender: TObject);
begin
form34.Memo1.Clear;;
form34.ADOQuery1.Close;
end;

procedure TForm34.LbSpeedButton2Click(Sender: TObject);
begin
form34.Edit2.Clear;
form34.ADOQuery1.Close;
end;

procedure TForm34.TabSheet1Show(Sender: TObject);
begin
Form34.SpeedButton2.Click;
end;

procedure TForm34.TabSheet3Show(Sender: TObject);
begin
Form34.SpeedButton4.Click;
end;

procedure TForm34.TabSheet4Show(Sender: TObject);
begin
Form34.LbSpeedButton4.Click;
end;
```

```
procedure TForm34.TabSheet7Show(Sender: TObject);
begin
Form34.LbSpeedButton2.Click;
end;

procedure TForm34.FormCreate(Sender: TObject);
begin
form34.DateTimePicker1.Date:=date;
form34.DateTimePicker2.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarını MySql database sisteminin
anlayacağı şekle dönüştürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm34.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form34.Edit1.Clear;
  form34.Edit2.Clear;
  form34.Memo1.Clear;
  form34.ADOQuery1.Close;
  form34.RadioButton1.Checked:=false;
  form34.RadioButton2.Checked:=false;
  form34.RadioButton3.Checked:=false;
  form34.RadioButton4.Checked:=false;
  form34.DateTimePicker2.Enabled:=false;
end;

procedure TForm34.FormHide(Sender: TObject);
begin
  form34.Edit1.Clear;
  form34.Edit2.Clear;
  form34.Memo1.Clear;
  form34.ADOQuery1.Close;
  form34.RadioButton1.Checked:=false;
  form34.RadioButton2.Checked:=false;
  form34.RadioButton3.Checked:=false;
  form34.RadioButton4.Checked:=false;
  form34.DateTimePicker2.Enabled:=false;
end;

end.
```

FORM 35 CODES

```
unit Unit35;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Menus, Grids, DBGrids, ExtCtrls, Buttons, ComCtrls,
  DB, ADODB;

type
  TForm35 = class(TForm)
    StatusBar1: TStatusBar;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    Label1: TLabel;
    SpeedButton2: TSpeedButton;
    Edit1: TEdit;
    Panel5: TPanel;
    TabSheet2: TTabSheet;
    Label2: TLabel;
    SpeedButton4: TSpeedButton;
    Panel2: TPanel;
    Edit2: TEdit;
    TabSheet3: TTabSheet;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    TabSheet4: TTabSheet;
    Panel3: TPanel;
    Panel4: TPanel;
    Label3: TLabel;
    ComboBox1: TComboBox;
    SpeedButton6: TSpeedButton;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
    procedure TabSheet4Show(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure TabSheet1Show(Sender: TObject);
    procedure TabSheet2Show(Sender: TObject);
    procedure TabSheet3Show(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form35: TForm35;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm35.Edit1Change(Sender: TObject);
begin
  form35.ADOQuery1.Close;
  form35.ADOQuery1.SQL.Text:='select * from users where user_name
like'+#39+'%'+form35.Edit1.Text+'%'+#39;
  form35.ADOQuery1.Open;
end;

procedure TForm35.Edit2Change(Sender: TObject);
begin
  form35.ADOQuery1.Close;
  form35.ADOQuery1.SQL.Text:='select * from users where staff_id
like'+#39+'%'+form35.Edit2.Text+'%'+#39;
  form35.ADOQuery1.Open;
end;

procedure TForm35.ComboBox1Change(Sender: TObject);
begin
  form35.ADOQuery1.Close;
  form35.ADOQuery1.SQL.Text:='select * from users where staff_state
like'+#39+'%'+form35.ComboBox1.Text+'%'+#39;
  form35.ADOQuery1.Open;
end;

procedure TForm35.TabSheet4Show(Sender: TObject);
begin
  form35.ADOQuery1.Close;
  form35.ADOQuery1.SQL.Text:='select * from users';
  form35.ADOQuery1.Open;
end;

procedure TForm35.SpeedButton2Click(Sender: TObject);
begin
form35.Edit1.Clear;
form35.ADOQuery1.Close;
end;

procedure TForm35.SpeedButton4Click(Sender: TObject);
begin
form35.Edit2.Clear;
form35.ADOQuery1.Close;
```

```
end;

procedure TForm35.SpeedButton6Click(Sender: TObject);
begin
form35.ComboBox1.Text:='Select One';
form35.ADOQuery1.Close;
end;

procedure TForm35.TabSheet1Show(Sender: TObject);
begin
Form35.SpeedButton2.Click;
end;

procedure TForm35.TabSheet2Show(Sender: TObject);
begin
Form35.SpeedButton4.Click;
end;

procedure TForm35.TabSheet3Show(Sender: TObject);
begin
Form35.SpeedButton6.Click;
end;

procedure TForm35.FormHide(Sender: TObject);
begin
  form35.Edit1.Clear;
  form35.Edit2.Clear;
  form35.ComboBox1.Text:='Select One';
  form35.ADOQuery1.Close;
end;

procedure TForm35.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  form35.Edit1.Clear;
  form35.Edit2.Clear;
  form35.ComboBox1.Text:='Select One';
  form35.ADOQuery1.Close;
end;

end.
```

FORM 36 CODES

```
unit Unit36;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```

```
 Dialogs, StdCtrls, Mask, Menus, Grids, DBGrids, LbSpeedButton, ExtCtrls,
 ComCtrls, Buttons, DB, ADODB;

type
 TForm36 = class(TForm)
  StatusBar1: TStatusBar;
  PageControl1: TPageControl;
  TabSheet1: TTabSheet;
  Label1: TLabel;
  RadioButton1: TRadioButton;
  RadioButton2: TRadioButton;
  RadioButton3: TRadioButton;
  RadioButton4: TRadioButton;
  DateTimePicker1: TDateTimePicker;
  DateTimePicker2: TDateTimePicker;
  Panel2: TPanel;
  TabSheet7: TTabSheet;
  Panel7: TPanel;
  DBGrid1: TDBGrid;
  Panel1: TPanel;
  MainMenu1: TMainMenu;
  F1: TMenuItem;
  LbSpeedButton1: TLbSpeedButton;
  LbSpeedButton2: TLbSpeedButton;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  procedure RadioButton1Click(Sender: TObject);
  procedure RadioButton2Click(Sender: TObject);
  procedure RadioButton3Click(Sender: TObject);
  procedure RadioButton4Click(Sender: TObject);
  procedure TabSheet7Show(Sender: TObject);
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure TabSheet1Show(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form36: TForm36;

implementation

uses Unit10;
```

```
{$R *.dfm}

procedure TForm36.RadioButton1Click(Sender: TObject);
begin
  form36.DateTimePicker2.Date:=date;
  form36.DateTimePicker2.Enabled:=false;
end;

procedure TForm36.RadioButton2Click(Sender: TObject);
begin
  form36.DateTimePicker2.Date:=date;
  form36.DateTimePicker2.Enabled:=false;
end;

procedure TForm36.RadioButton3Click(Sender: TObject);
begin
  form36.DateTimePicker2.Date:=date;
  form36.DateTimePicker2.Enabled:=false;
end;

procedure TForm36.RadioButton4Click(Sender: TObject);
begin
if form36.RadioButton4.Checked = true then
  form36.DateTimePicker2.Enabled:=true
else
  form36.DateTimePicker2.Enabled:=false;
end;

procedure TForm36.TabSheet7Show(Sender: TObject);
begin
  form36.ADOQuery1.Close;
  form36.ADOQuery1.SQL.Text:='select * from wrongpasswords';
  form36.ADOQuery1.Open;
end;

procedure TForm36.LbSpeedButton1Click(Sender: TObject);
begin
if (form36.RadioButton1.Checked = true) then
  begin
    form36.ADOQuery1.Close;
    form36.ADOQuery1.SQL.Text:='select * from wrongpasswords where try_date <
'+#39+datetostr(form36.DateTimePicker1.Date)+#39;
    form36.ADOQuery1.Open;
  end
  else if (form36.RadioButton2.Checked = true) then
  begin
    form36.ADOQuery1.Close;
    form36.ADOQuery1.SQL.Text:='select * from wrongpasswords where try_date >
'+#39+datetostr(form36.DateTimePicker1.Date)+#39;
    form36.ADOQuery1.Open;
```

```
 end
 else if (form36.RadioButton3.Checked = true) then
 begin
  form36.ADOQuery1.Close;
  form36.ADOQuery1.SQL.Text:='select * from wrongpasswords where try_date =
'+#39+datetostr(form36.DateTimePicker1.Date)+#39;
  form36.ADOQuery1.Open;
 end
 else if (form36.RadioButton4.Checked = true) then
 begin
  form36.ADOQuery1.Close;
  form36.ADOQuery1.SQL.Text:='select * from wrongpasswords where try_date
between '+#39+datetostr(form36.DateTimePicker1.Date)+#39+' and
'+#39+datetostr(form36.DateTimePicker2.Date)+#39;
  form36.ADOQuery1.Open;
 end
 else
  showmessage('PLEASE SELECT SEARCH CRITERIA');
end;

procedure TForm36.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 form36.ADOQuery1.Close;
 form36.RadioButton1.Checked:=false;
 form36.RadioButton2.Checked:=false;
 form36.RadioButton3.Checked:=false;
 form36.RadioButton4.Checked:=false;
 form36.DateTimePicker2.Enabled:=false;
end;

procedure TForm36.FormHide(Sender: TObject);
begin
 form36.ADOQuery1.Close;
 form36.RadioButton1.Checked:=false;
 form36.RadioButton2.Checked:=false;
 form36.RadioButton3.Checked:=false;
 form36.RadioButton4.Checked:=false;
 form36.DateTimePicker2.Enabled:=false;
end;

procedure TForm36.LbSpeedButton2Click(Sender: TObject);
begin
 FORM36.DateTimePicker1.Date:=DATE;
 FORM36.DateTimePicker2.Date:=DATE;
 FORM36.DateTimePicker2.Enabled:=FALSE;
 FORM36.RadioButton1.Checked:=FALSE;
 FORM36.RadioButton2.Checked:=FALSE;
 FORM36.RadioButton3.Checked:=FALSE;
 FORM36.RadioButton4.Checked:=FALSE;
 FORM36.ADOQuery1.Close;
```

```
end;

procedure TForm36.FormCreate(Sender: TObject);
begin
form36.DateTimePicker1.Date:=date;
form36.DateTimePicker2.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarını MySql database sisteminin
anlayacağı şekle dönüştürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm36.TabSheet1Show(Sender: TObject);
begin
Form36.LbSpeedButton2.Click;
end;

end.
```

FORM 37 CODES

```
unit Unit37;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls, ComCtrls, Grids, DBGrids, ExtCtrls, Menus,
  DB, ADODB;

type
  TForm37 = class(TForm)
    PageControl1: TPageControl;
    MainMenu1: TMainMenu;
    F1: TMenuItem;
    StatusBar1: TStatusBar;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    Panel2: TPanel;
    Panel3: TPanel;
    DBGrid2: TDBGrid;
    DBGrid3: TDBGrid;
    Panel4: TPanel;
    DateTimePicker1: TDateTimePicker;
    Label1: TLabel;
    SpeedButton2: TSpeedButton;
    Panel5: TPanel;
```

```
  SpeedButton1: TSpeedButton;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  ADOQuery2: TADOQuery;
  DataSource2: TDataSource;
  ADOQuery3: TADOQuery;
  DataSource3: TDataSource;
  procedure SpeedButton1Click(Sender: TObject);
  procedure SpeedButton2Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormHide(Sender: TObject);
  procedure FormCreate(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form37: TForm37;

implementation

uses Unit10;

{$R *.dfm}

procedure TForm37.SpeedButton1Click(Sender: TObject);
begin
   form37.ADOQuery1.Close;
   form37.ADOQuery1.SQL.Text:='select * from vaccinate where next_vaccinatedate =
'+#39+datetostr(form37.DateTimePicker1.Date)+#39;
   form37.ADOQuery1.Open;
   form37.ADOQuery2.Close;
   form37.ADOQuery2.SQL.Text:='select * from ipdrug where ip_nextdrugdate =
'+#39+datetostr(form37.DateTimePicker1.Date)+#39;
   form37.ADOQuery2.Open;
   form37.ADOQuery3.Close;
   form37.ADOQuery3.SQL.Text:='select * from opdrug where op_nextdrugdate =
'+#39+datetostr(form37.DateTimePicker1.Date)+#39;
   form37.ADOQuery3.Open;
end;

procedure TForm37.SpeedButton2Click(Sender: TObject);
begin
 form37.DateTimePicker1.Date:=date;
 form37.ADOQuery1.Close;
 form37.ADOQuery2.Close;
 form37.ADOQuery3.Close;
end;
```

```
procedure TForm37.FormClose(Sender: TObject; var Action: TCloseAction);
begin
Form37.SpeedButton2.Click;
end;

procedure TForm37.FormHide(Sender: TObject);
begin
Form37.SpeedButton2.Click;
end;

procedure TForm37.FormCreate(Sender: TObject);
begin
form37.DateTimePicker1.Date:=date;
dateseparator := '-';    // Burada tarih'in ayraçlarını MySql database sisteminin
anlayacağı şekle dönüştürdüm...
shortdateformat := 'yyyy/m/d';
end;

end.
```

FORM 38 CODES

```
unit Unit38;

interface

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

type
  TForm38 = class(TForm)
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
```

```
 end;

var
 Form38: TForm38;
 TA:integer;
implementation

uses Unit10, Unit14, Unit17, Unit18, Unit19, Unit20, Unit21, Unit22;

{$R *.dfm}

procedure TForm38.FormShow(Sender: TObject);
begin
 FORM38.ADOQuery1.SQL.Text:='select
Staff_id,Staff_name,Staff_surname,S_birthdate from staff';
 form38.ADOQuery1.Open;
end;

procedure TForm38.SpeedButton1Click(Sender: TObject);
begin
 if form38.ADOQuery1.RecordCount <> 0 then
 begin
  if TA = 10 then
   form10.edit1.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 14 then
   form14.edit1.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 17 then
   form17.edit4.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 18 then
   form18.edit2.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 19 then
   form19.edit2.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 20 then
   form20.edit2.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 21 then
   form21.edit2.text:=form38.DBGrid1.Fields[0].Text
  else if TA = 22 then
   form22.edit4.text:=form38.DBGrid1.Fields[0].Text;
  form38.Hide;
  TA:=0;
 end
 else
  showmessage('THERE IS NO RECORD IN DATABASE');
end;

procedure TForm38.SpeedButton2Click(Sender: TObject);
begin
 form38.ADOQuery1.Close;
 FORM38.ADOQuery1.SQL.Text:='select
Staff_id,Staff_name,Staff_surname,S_birthdate from staff';
```

```
 form38.ADOQuery1.Open;
end;

procedure TForm38.SpeedButton3Click(Sender: TObject);
begin
form38.ADOQuery1.Close;
FORM38.Hide;
end;

end.
```

FORM 39 CODES

```
unit Unit39;

interface

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

type
  TForm39 = class(TForm)
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    SpeedButton1: TSpeedButton;
    Panel2: TPanel;
    Panel3: TPanel;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    Panel4: TPanel;
    ADOQuery1: TADOQuery;
    ADOQuery2: TADOQuery;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure DBGrid2CellClick(Column: TColumn);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure SpeedButton3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
 Form39: TForm39;
 ANI:INTEGER;
implementation

uses Unit10, Unit17, Unit18, Unit19, Unit20, Unit21, Unit22;

{$R *.dfm}

procedure TForm39.FormShow(Sender: TObject);
begin
 FORM39.ADOQuery1.Close;
 FORM39.ADOQuery1.SQL.Text:='select customer_id,Cname,Csurname,Mobilphone
from customer';
 form39.ADOQuery1.Open;
end;

procedure TForm39.DBGrid2CellClick(Column: TColumn);
begin
 if form39.DBGrid2.Fields[0].IsNull = false then
 begin
  form39.ADOQuery2.Close;
  form39.ADOQuery2.SQL.Text:='select Animal_id,Animal_name from animal where
owner_no='+#39+form39.DBGrid2.Fields[0].Text+#39;
  form39.ADOQuery2.Open;
 end
 else
  showmessage('THERE IS NO CUSTOMER IN DATABASE');

end;

procedure TForm39.DBGrid1CellClick(Column: TColumn);
begin
  if form39.DBGrid1.Fields[0].IsNull = false then
   FORM39.Panel4.Caption:=FORM39.DBGrid1.Fields[0].Text
  ELSE
   SHOWMESSAGE('PLEASE SELECT A CUSTOMER FROM CUSTOMER LIST');

end;

procedure TForm39.SpeedButton1Click(Sender: TObject);
begin
 IF FORM39.Panel4.Caption <> '' THEN
 BEGIN
  IF ANI=17 THEN
   FORM17.EDIT1.TEXT:=FORM39.Panel4.Caption
  ELSE IF ANI=18 THEN
   FORM18.EDIT1.TEXT:=FORM39.Panel4.Caption
  ELSE IF ANI=19 THEN
```

```
    FORM19.EDIT1.TEXT:=FORM39.Panel4.Caption
   ELSE IF ANI=20 THEN
    FORM20.EDIT1.TEXT:=FORM39.Panel4.Caption
   ELSE IF ANI=21 THEN
    FORM21.EDIT1.TEXT:=FORM39.Panel4.Caption
   ELSE IF ANI=22 THEN
    FORM22.EDIT1.TEXT:=FORM39.Panel4.Caption;
   FORM39.Hide;
   ANI:=0;
  END
  ELSE
   SHOWMESSAGE('SELECT AN ANIMAL FROM ANIMAL LIST');
end;

procedure TForm39.SpeedButton2Click(Sender: TObject);
begin
 FORM39.ADOQuery1.Close;
 FORM39.ADOQuery1.SQL.Text:='select customer_id,Cname,Csurname,Mobilphone
from customer';
 form39.ADOQuery1.Open;
 FORM39.ADOQuery2.Close;
 FORM39.Panel4.Caption:='';
end;

procedure TForm39.FormHide(Sender: TObject);
begin
 FORM39.ADOQuery1.Close;
 FORM39.ADOQuery2.Close;
 FORM39.Panel4.Caption:='';
end;

procedure TForm39.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 FORM39.ADOQuery1.Close;
 FORM39.ADOQuery2.Close;
 FORM39.Panel4.Caption:='';
end;

procedure TForm39.SpeedButton3Click(Sender: TObject);
begin
 FORM39.Close;
end;

end.
```

FORM 40 CODES

```
unit Unit40;

interface

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

type
  TForm40 = class(TForm)
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    ADOConnection1: TADOConnection;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form40: TForm40;

implementation

uses Unit16;

{$R *.dfm}

procedure TForm40.SpeedButton1Click(Sender: TObject);
begin
  if form40.ADOQuery1.RecordCount <> 0 then
   begin
     FORM16.EDIT5.TEXT:=FORM40.DBGrid1.Fields[0].Text;
     form40.Hide;
   END
   ELSE
     SHOWMESSAGE('THERE IS NO RECORD IN DATABASE');
 end;
```

```
procedure TForm40.FormShow(Sender: TObject);
begin
 FORM40.ADOQuery1.Close;
 FORM40.ADOQuery1.SQL.Text:='SELECT customer_id,cname,csurname FROM
customer';
 form40.ADOQuery1.Open;
end;

procedure TForm40.SpeedButton2Click(Sender: TObject);
begin
 FORM40.ADOQuery1.Close;
 FORM40.ADOQuery1.SQL.Text:='SELECT customer_id,cname,csurname FROM
customer';
 form40.ADOQuery1.Open;
end;

procedure TForm40.SpeedButton3Click(Sender: TObject);
begin
 form40.ADOQuery1.Close;
 FORM40.Hide;
end;

end.
```

FORM 41 CODES

```
unit Unit41;

interface

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

type
  TForm41 = class(TForm)
    Image1: TImage;
    Edit1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Edit2: TEdit;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    Panel5: TPanel;
    Panel6: TPanel;
    Panel7: TPanel;
    Panel8: TPanel;
    Panel9: TPanel;
```

```
  Panel10: TPanel;
  LbSpeedButton1: TLbSpeedButton;
  LbSpeedButton2: TLbSpeedButton;
  Timer1: TTimer;
  ADOQuery1: TADOQuery;
  DataSource1: TDataSource;
  ADOQuery2: TADOQuery;
  DataSource2: TDataSource;
  procedure Timer1Timer(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure LbSpeedButton1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure LbSpeedButton2Click(Sender: TObject);
  procedure FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure Panel2Click(Sender: TObject);
  procedure Panel3Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form41: TForm41;
 renk,num:integer;

implementation

uses Unit10, Unit3, Unit2, Unit1, Unit4, Unit6, Unit7, Unit9;

{$R *.dfm}

procedure TForm41.Timer1Timer(Sender: TObject);
begin
 renk:=renk+1;
 if renk = 1 then
 begin
  form41.Panel2.Color:=clblack;
  form41.Panel2.Font.Color:=clwhite;
 end
 else if renk = 2 then
 begin
  form41.Panel3.Color:=clblack;
  form41.Panel3.Font.Color:=clwhite;
 end
 else if renk = 3 then
 begin
  form41.Panel4.Color:=clblack;
```

```
 form41.Panel4.Font.Color:=clwhite;
end
else if renk = 4 then
begin
 form41.Panel5.Color:=clblack;
 form41.Panel5.Font.Color:=clwhite;
end
else if renk = 5 then
begin
 form41.Panel6.Color:=clblack;
 form41.Panel6.Font.Color:=clwhite;
end
else if renk = 6 then
begin
 form41.Panel7.Color:=clblack;
 form41.Panel7.Font.Color:=clwhite;
end
else if renk = 7 then
begin
 form41.Panel8.Color:=clblack;
 form41.Panel8.Font.Color:=clwhite;
end
else if renk = 8 then
begin
 form41.Panel9.Color:=clblack;
 form41.Panel9.Font.Color:=clwhite;
end
else if renk = 9 then
begin
 form41.Panel10.Color:=clblack;
 form41.Panel10.Font.Color:=clwhite;
end
else if renk = 10 then
begin
 form41.Panel10.Color:=clwhite;
 form41.Panel10.Font.Color:=clblack;
end
else if renk = 11 then
begin
 form41.Panel9.Color:=clwhite;
 form41.Panel9.Font.Color:=clblack;
end
else if renk = 12 then
begin
 form41.Panel8.Color:=clwhite;
 form41.Panel8.Font.Color:=clblack;
end
else if renk = 13 then
begin
 form41.Panel7.Color:=clwhite;
```

```
 form41.Panel7.Font.Color:=clblack;
end
else if renk = 14 then
begin
 form41.Panel6.Color:=clwhite;
 form41.Panel6.Font.Color:=clblack;
end
else if renk = 15 then
begin
 form41.Panel5.Color:=clwhite;
 form41.Panel5.Font.Color:=clblack;
end
else if renk = 16 then
begin
 form41.Panel4.Color:=clwhite;
 form41.Panel4.Font.Color:=clblack;
end
else if renk = 17 then
begin
 form41.Panel3.Color:=clwhite;
 form41.Panel3.Font.Color:=clblack;
end
else if renk = 18 then
begin
 form41.Panel2.Color:=clwhite;
 form41.Panel2.Font.Color:=clblack;
end
else if renk = 19 then
  renk:=0;
end;

procedure TForm41.FormShow(Sender: TObject);
begin
renk:=0;
num:=0;
end;

procedure TForm41.LbSpeedButton1Click(Sender: TObject);
begin
if (form41.Edit1.Text <> '') and (form41.Edit2.Text <> '') then
begin
 if num <> 3 then    //Her Butona basılışında sorguluyor. num 3 ise çıkışı sağlıyor.
 begin          //If kullanmamın nedeni butona basılıyor olmasından. For döngüsünde
daima dögüye giriyor ve direkt programı sonlandırıyor...
   FORM41.ADOQuery1.Close;
   form41.ADOQuery1.SQL.Text:='select * from users  where
user_name='+#39+form41.Edit1.Text+#39+' and
password='+#39+form41.Edit2.Text+#39;
   form41.ADOQuery1.Open;      //Kullanıcı adı ve şifre sorgulaması yapılıyor..
```

```
  if (form41.ADOQuery1['user_name'] = Null) and (form41.ADOQuery1['password'] =
Null) or (form41.ADOQuery1['user_name'] = Null) or
(form41.ADOQuery1['password'] = Null) then
  begin
   showmessage('WRONG USER NAME OR PASSWORD.PLEASE TRY AGAIN');
   num:=num+1;    //yanlış girişlerin sayısını hesaplıyor. her seferinde 1 arttırıyor.
  END;
  if(form41.ADOQuery1['user_name'] <> Null) and (form41.ADOQuery1['password']
<> Null) then
  begin
   if (form41.ADOQuery1['staff_state'] = 'WORKING') then
   begin
    if (form41.ADOQuery1['staff_pozition'] = 'MANAGER') then
    begin
     form3.SpeedButton12.Enabled:=false;
     //form6.SpeedButton15.Enabled:=false;
    end
    else if (form41.ADOQuery1['staff_pozition'] = 'VETERINERIAN') then
    begin
     form2.SpeedButton5.Enabled:=false;
     form3.SpeedButton6.Enabled:=false;
     form3.SpeedButton12.Enabled:=false;
     form6.SpeedButton7.Enabled:=false;
     //form6.SpeedButton15.Enabled:=false;
     form2.SpeedButton1.Enabled:=false;
     form3.SpeedButton14.Enabled:=false;
    end
    else if (form41.ADOQuery1['staff_pozition'] = 'USER') then
    begin
     form2.SpeedButton5.Enabled:=false;
     form3.SpeedButton6.Enabled:=false;
     form1.SpeedButton2.Enabled:=false;
     //form3.SpeedButton7.Enabled:=false;
     form3.SpeedButton12.Enabled:=false;
     //form6.SpeedButton7.Enabled:=false;
     //form6.SpeedButton15.Enabled:=false;
     form2.SpeedButton1.Enabled:=false;
     form3.SpeedButton14.Enabled:=false;
     form7.SpeedButton1.Enabled:=false;
     form7.SpeedButton2.Enabled:=false;
     form1.SpeedButton4.Enabled:=false;
     form3.SpeedButton1.Enabled:=false;
     form3.SpeedButton14.Enabled:=false;
     form1.SpeedButton6.Enabled:=false;
    end
    else if (form41.ADOQuery1['staff_pozition'] = 'TEMPORARY') then
    begin
     form1.SpeedButton1.Enabled:=false;
     form1.SpeedButton2.Enabled:=false;
     form1.SpeedButton4.Enabled:=false;
```

```
        form1.SpeedButton5.Enabled:=false;
        form1.SpeedButton6.Enabled:=false;
        form3.SpeedButton1.Enabled:=false;
        form3.SpeedButton7.Enabled:=false;
        form3.SpeedButton14.Enabled:=false;
        form3.SpeedButton2.Enabled:=false;
        form3.SpeedButton15.Enabled:=false;
        form3.SpeedButton5.Enabled:=false;
        form3.SpeedButton3.Enabled:=false;
        form3.SpeedButton10.Enabled:=false;
        form3.SpeedButton8.Enabled:=false;
        form3.SpeedButton11.Enabled:=false;
        form3.SpeedButton6.Enabled:=false;
        form3.SpeedButton12.Enabled:=false;
        form3.SpeedButton13.Enabled:=false;
       end;
       FORM1.SHOW;
       FORM41.Hide;
     end
     else if (form41.ADOQuery1['staff_state'] = 'LEFT') then
     begin
       showmessage('YOU CAN NOT ENTER SYSTEM.YOUR ACCOUNT HAS
BLOCKED');
       FORM41.ADOQuery2.Close;
       FORM41.ADOQuery2.SQL.Text:='insert into wrongpasswords
(wuser_name,w_password,try_date,try_time) values
('+#39+form41.Edit1.Text+#39+','+#39+form41.Edit2.Text+#39+','+#39+datetostr(date
)+#39+','+#39+timetostr(time)+#39+')';
       form41.ADOQuery2.ExecSQL;
       form41.Edit1.Clear;
       form41.Edit2.Clear;
      end;
    end;
  end;
  if num=3 then
  BEGIN
    FORM41.ADOQuery2.Close;
    FORM41.ADOQuery2.SQL.Text:='insert into wrongpasswords
(wuser_name,w_password,try_date,try_time) values
('+#39+form41.Edit1.Text+#39+','+#39+form41.Edit2.Text+#39+','+#39+datetostr(date
)+#39+','+#39+timetostr(time)+#39+')';
    form41.ADOQuery2.ExecSQL;
    showmessage('YOU TRIED THREE TIMES TO ENTER SYSTEM.'+#13+'THE
 APPLICATION PROGRAM WILL BE TERMINATE');
    form41.Edit1.Clear;
    form41.Edit2.Clear;
    FORM41.Close;
  END;
 end
 else
```

```
 showmessage('FILL THE USER NAME AND PASSWORD');
end;

procedure TForm41.FormCreate(Sender: TObject);
begin
dateseparator := '-';    // Burada tarih'in ayraçlarını MySql database sisteminin
anlayacağı şekle dönüştürdüm...
shortdateformat := 'yyyy/m/d';
end;

procedure TForm41.FormClose(Sender: TObject; var Action: TCloseAction);
begin
if form9.CheckBox1.Checked <> true then
AnimateWindow(Form1.Handle, 1000, AW_HOR_POSITIVE or
AW_HOR_NEGATIVE or AW_Hide);
end;

procedure TForm41.LbSpeedButton2Click(Sender: TObject);
begin
form41.Close;
end;

procedure TForm41.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if key=13 then
  begin
  Form41.LbSpeedButton1.Click;
  end;
  if key = 27 then
    form41.Close;

end;

procedure TForm41.Panel2Click(Sender: TObject);
begin
form41.Timer1.Enabled:=true;
end;

procedure TForm41.Panel3Click(Sender: TObject);
begin
form41.Timer1.Enabled:=false;
end;

end.
```

VETAP PROJECT CODES

```
program vetap;
```

```
uses
  Forms,
  windows,
  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},
  Unit20 in 'Unit20.pas' {Form20},
  Unit21 in 'Unit21.pas' {Form21},
  Unit22 in 'Unit22.pas' {Form22},
  Unit23 in 'Unit23.pas' {Form23},
  Unit24 in 'Unit24.pas' {Form24},
  Unit25 in 'Unit25.pas' {Form25},
  Unit26 in 'Unit26.pas' {Form26},
  Unit27 in 'Unit27.pas' {Form27},
  Unit28 in 'Unit28.pas' {Form28},
  Unit29 in 'Unit29.pas' {Form29},
  Unit30 in 'Unit30.pas' {Form30},
  Unit31 in 'Unit31.pas' {Form31},
  Unit32 in 'Unit32.pas' {Form32},
  Unit33 in 'Unit33.pas' {Form33},
  Unit34 in 'Unit34.pas' {Form34},
  Unit35 in 'Unit35.pas' {Form35},
  Unit36 in 'Unit36.pas' {Form36},
  Unit37 in 'Unit37.pas' {Form37},
  Unit38 in 'Unit38.pas' {Form38},
  Unit39 in 'Unit39.pas' {Form39},
  Unit40 in 'Unit40.pas' {Form40},
  Unit41 in 'Unit41.pas' {Form41};

  {$R *.res}

  begin
    Application.Initialize;
```

```
Application.CreateForm(TForm41, Form41);
Application.CreateForm(TForm1, Form1);
Application.CreateForm(TForm2, Form2);
Application.CreateForm(TForm3, Form3);
Application.CreateForm(TForm4, Form4);
Application.CreateForm(TForm5, Form5);
Application.CreateForm(TForm6, Form6);
Application.CreateForm(TForm7, Form7);
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);
Application.CreateForm(TForm19, Form19);
Application.CreateForm(TForm20, Form20);
Application.CreateForm(TForm21, Form21);
Application.CreateForm(TForm22, Form22);
Application.CreateForm(TForm23, Form23);
Application.CreateForm(TForm24, Form24);
Application.CreateForm(TForm25, Form25);
Application.CreateForm(TForm26, Form26);
Application.CreateForm(TForm27, Form27);
Application.CreateForm(TForm28, Form28);
Application.CreateForm(TForm29, Form29);
Application.CreateForm(TForm30, Form30);
Application.CreateForm(TForm31, Form31);
Application.CreateForm(TForm32, Form32);
Application.CreateForm(TForm33, Form33);
Application.CreateForm(TForm34, Form34);
Application.CreateForm(TForm35, Form35);
Application.CreateForm(TForm36, Form36);
Application.CreateForm(TForm37, Form37);
Application.CreateForm(TForm38, Form38);
Application.CreateForm(TForm39, Form39);
Application.CreateForm(TForm40, Form40);
form8:=Tform8.Create(Application);
form8.Show;
form8.Update;
sleep(3000);

form8.Hide;
form8.Free;
//Application.CreateForm(TForm8, Form8);
 Application.Run;
end.
```

DATABASE CREATION CODES

```
# Host: localhost
# Database: vetap
# Table: 'animal'
#
CREATE TABLE `animal` (
  `Animal_id` int(12) NOT NULL auto_increment,
  `Animal_name` varchar(50) NOT NULL default '',
  `Animal_kind` varchar(50) NOT NULL default '',
  `Animal_race` varchar(50) NOT NULL default '',
  `owner_no` varchar(12) NOT NULL default '',
  `ABirthdate` date NOT NULL default '0000-00-00',
  `Animal_sex` varchar(6) NOT NULL default '',
  `Animal_color` varchar(20) NOT NULL default '',
  `Animal_weight` varchar(10) NOT NULL default '',
  `Animal_mark` varchar(100) NOT NULL default '',
  `Animal_alergy` varchar(100) NOT NULL default '',
  `ACronic_medicine` varchar(100) NOT NULL default '',
  `Collar_no` varchar(15) NOT NULL default '',
  `Earning_no` varchar(15) NOT NULL default '',
  `A_note` varchar(100) NOT NULL default '',
  `Life_state` varchar(10) NOT NULL default '',
  PRIMARY KEY  (`Animal_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'appliedoperation'
#
CREATE TABLE `appliedoperation` (
  `Aop_id` int(12) NOT NULL auto_increment,
```

```
 `Animal_id` int(12) NOT NULL default '0',
 `Operation_name` varchar(50) NOT NULL default '',
 `Operation_date` date NOT NULL default '0000-00-00',
 `Drug_name` varchar(50) NOT NULL default '',
 `Applied_staff` int(12) NOT NULL default '0',
 `O_note` varchar(100) NOT NULL default '',
 PRIMARY KEY  (`Aop_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'customer'
#
CREATE TABLE `customer` (
  `Customer_id` int(12) NOT NULL auto_increment,
  `Cname` varchar(50) NOT NULL default '',
  `Csurname` varchar(50) NOT NULL default '',
  `Homephone` varchar(14) NOT NULL default '',
  `Mobilphone` varchar(14) NOT NULL default '',
  `Workphone` varchar(14) NOT NULL default '',
  `Fax` varchar(14) NOT NULL default '',
  `Address` varchar(100) NOT NULL default '',
  `City` varchar(50) NOT NULL default '',
  `Town` varchar(50) NOT NULL default '',
  `Country` varchar(50) NOT NULL default '',
  `Email` varchar(50) NOT NULL default '',
  `Web` varchar(50) NOT NULL default '',
  `Recorddate` date NOT NULL default '0000-00-00',
  `Recordtime` time NOT NULL default '00:00:00',
  `C_note` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`Customer_id`)
) TYPE=MyISAM;
```

```
# Host: localhost
# Database: vetap
# Table: 'drugs'
#
CREATE TABLE `drugs` (
  `Drug_id` int(12) NOT NULL auto_increment,
  `Drug_name` varchar(50) NOT NULL default '',
  `Drug_duration` int(2) NOT NULL default '0',
  `Drug_kind` varchar(20) NOT NULL default '',
  PRIMARY KEY  (`Drug_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'illnesses'
#
CREATE TABLE `illnesses` (
  `ill_id` int(12) NOT NULL auto_increment,
  `Animal_id` int(12) NOT NULL default '0',
  `Protocol_no` int(14) NOT NULL default '0',
  `Date` date NOT NULL default '0000-00-00',
  `Illness` varchar(50) NOT NULL default '',
  `Treatment` varchar(100) NOT NULL default '',
  `Laboratory_result` varchar(100) NOT NULL default '',
  `Applied_staff` int(12) NOT NULL default '0',
  `i_note` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`ill_id`)
) TYPE=MyISAM;
```

```
# Host: localhost
# Database: vetap
# Table: 'illnesses'
#
CREATE TABLE `illnesses` (
  `ill_id` int(12) NOT NULL auto_increment,
  `Animal_id` int(12) NOT NULL default '0',
  `Protocol_no` int(14) NOT NULL default '0',
  `Date` date NOT NULL default '0000-00-00',
  `Illness` varchar(50) NOT NULL default '',
  `Treatment` varchar(100) NOT NULL default '',
  `Laboratory_result` varchar(100) NOT NULL default '',
  `Applied_staff` int(12) NOT NULL default '0',
  `i_note` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`ill_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'ipdrug'
#
CREATE TABLE `ipdrug` (
  `Ip_id` int(12) NOT NULL auto_increment,
  `Animal_id` int(12) NOT NULL default '0',
  `Ip_drugname` varchar(50) NOT NULL default '',
  `Ip_drugdate` date NOT NULL default '0000-00-00',
  `Ip_nextdrugdate` date NOT NULL default '0000-00-00',
  `Applied_staff` int(12) NOT NULL default '0',
  `Ip_drugnote` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`Ip_id`)
```

```
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'medicinate'
#
CREATE TABLE `medicinate` (
  `Medicinate_id` int(12) NOT NULL auto_increment,
  `Animal_id` int(12) NOT NULL default '0',
  `Drug_name` varchar(50) NOT NULL default '',
  `Medicinate_date` date NOT NULL default '0000-00-00',
  `Applied_staff` int(12) NOT NULL default '0',
  `M_note` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`Medicinate_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'opdrug'
#
CREATE TABLE `opdrug` (
  `Op_id` int(12) NOT NULL auto_increment,
  `Animal_id` int(12) NOT NULL default '0',
  `Op_drugname` varchar(50) NOT NULL default '',
  `Op_drugdate` date NOT NULL default '0000-00-00',
  `Op_nextdrugdate` date NOT NULL default '0000-00-00',
  `Applied_staff` int(12) NOT NULL default '0',
  `Op_drugnote` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`Op_id`)
) TYPE=MyISAM;
```

```
# Host: localhost
# Database: vetap
# Table: 'operations'
#
CREATE TABLE `operations` (
  `Operation_id` int(12) NOT NULL auto_increment,
  `Operation_name` varchar(50) NOT NULL default '',
  PRIMARY KEY  (`Operation_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'staff'
#
CREATE TABLE `staff` (
  `Staff_id` int(12) NOT NULL auto_increment,
  `Staff_name` varchar(50) NOT NULL default '',
  `Staff_surname` varchar(50) NOT NULL default '',
  `Staff_task` varchar(50) NOT NULL default '',
  `University` varchar(100) NOT NULL default '',
  `Grade_state` varchar(50) NOT NULL default '',
  `S_workstartdate` date NOT NULL default '0000-00-00',
  `S_birthdate` date NOT NULL default '0000-00-00',
  `S_TCidno` varchar(15) NOT NULL default '',
  `S_homephone` varchar(14) NOT NULL default '',
  `S_mobilphone` varchar(14) NOT NULL default '',
  `S_address` varchar(100) NOT NULL default '',
  `S_city` varchar(50) NOT NULL default '',
  `S_town` varchar(50) NOT NULL default '',
  `S_country` varchar(50) NOT NULL default '',
```

```
 `S_email` varchar(50) NOT NULL default '',
 `S_web` varchar(50) NOT NULL default '',
 `S_leavingdate` date NOT NULL default '0000-00-00',
 `S_note` varchar(100) NOT NULL default '',
 PRIMARY KEY (`Staff_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'users'
#
CREATE TABLE `users` (
 `User_name` varchar(50) NOT NULL default '',
 `Password` varchar(20) NOT NULL default '',
 `Staff_id` int(12) NOT NULL default '0',
 `Staff_state` varchar(20) NOT NULL default '',
 `Staff_pozition` varchar(50) NOT NULL default '',
 PRIMARY KEY (`User_name`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'vaccinate'
#
CREATE TABLE `vaccinate` (
 `Animal_id` int(12) NOT NULL default '0',
 `Vaccine_name` varchar(50) NOT NULL default '',
 `Vaccinate_date` date NOT NULL default '0000-00-00',
 `Next_vaccinatedate` date NOT NULL default '0000-00-00',
 `Vaccine_serialno` varchar(20) NOT NULL default '',
 `Vaccine_producer` varchar(50) NOT NULL default '',
 `Applied_staff` int(12) NOT NULL default '0',
```

```
 `V_note` varchar(100) NOT NULL default '',
 PRIMARY KEY  (`Animal_id`,`Vaccine_name`,`Vaccinate_date`,`Vaccine_serialno`)
) TYPE=MyISAM;

# Host: localhost
# Database: vetap
# Table: 'vaccines'
#
CREATE TABLE `vaccines` (
 `Vaccine_id` int(12) NOT NULL auto_increment,
 `Vaccine_name` varchar(50) NOT NULL default '',
 `Vaccine_duration` int(2) NOT NULL default '0',
 PRIMARY KEY  (`Vaccine_id`)
) TYPE=MyISAM;


# Host: localhost
# Database: vetap
# Table: 'wrongpasswords'
#
CREATE TABLE `wrongpasswords` (
 `Wuser_name` varchar(50) NOT NULL default '',
 `W_password` varchar(50) NOT NULL default '',
 `Try_date` date NOT NULL default '0000-00-00',
 `Try_time` time NOT NULL default '00:00:00'
) TYPE=MyISAM;
```