

**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**STOCK MANAGEMENT**

**Graduation Project**

**COM- 400**

**Student : Alp SOYDAN**

**Supervisor : Assist.Prof.Dr. İmanov ELBRUS**

**Nicosia – 2008**

## ACKNOWLEDGMENTS

*"First, I would like to thank my supervisor Dr. Elbus Imanov for his invaluable advice and belief in my work and myself over the course of this Graduation Project..*

*Second, I would like to express my gratitude to Near East University for the scholarship that made the work possible.*

*Third, I thank my family for their constant encouragement and support during the preparation of this project.*

*Fourth, I would like to thank Neu Computer Engineering Department academicians for their invaluable advice and support.*

*Finally, I would also like to thank my friend Evrim Kaki for his advice and support."*

## **TABLE OF CONTENTS**

<b>ACKNOWLEDGEMENT</b>	<b>i</b>
<b>TABLE OF CONTENTS</b>	<b>ii</b>
<b>ABSTRACT</b>	<b>iii</b>
<b>INTRODUCTION</b>	<b>1</b>
<b>CHAPTER ONE – DELPHI PROGRAMMING LANGUAGE</b>	<b>2</b>
1.1 Introduction	2
1.2 What is Delphi?	3
1.3 History Of Delphi	3
1.3.1 Pascal and Delphi's history	3
1.3.2 Delphi Versions	4
1.4 Delphi Programming Pheriphals	6
1.4.1 What is Project File(.DPR)?	6
1.4.2 Project Unit	8
1.4.3 Data types and variables	10
1.4.4 Procedures and functions	12
1.4.5 Classes and Objects	17
1.4.6 Libraries and Packages	21
<b>CHAPTER TWO – DATABASE CONCEPT OF DELPHI</b>	<b>23</b>
2.1 Borland Database Engine	23
2.1.1 What is BDE?	23
2.1.2 History of BDE	23
2.1.3 BDE Design Fundamentals	24
2.2 Paradox Database	25
2.2.1 Paradox Database Fundamentals	25
2.2.2 Paradox Table Field Types	27
2.2.3 Paradox 4 Table Specifications	31
2.2.4 Paradox 5 Table Specifications	31
2.2.5 Paradox 7 And Above Table Specifications	32
<b>CHAPTER THREE – USER'S MANUAL</b>	<b>33</b>
<b>CONCLUSION</b>	<b>39</b>
<b>REFERENCES</b>	<b>40</b>
<b>APPENDIX</b>	<b>41</b>

## **ABSTRACT**

Automation or management software is generally any software program that helps a business increase productivity or measure their productivity. The term covers a large variation of uses within the business environment, and can be categorized by using a small, medium and large matrix

The medium size, or SME, has a broader range of software applications, ranging from accounting, groupware, customer relationship management, human resources software, outsourcing relationship management, loan origination software, shopping cart software, field service software, and other productivity enhancing applications.

The main problem of a stock management system is tracking the data flow on time and calculating net profit or total net income up to that point and calculating the amount of money that will be assigned to investments for products. In my project there will be graphical aids for users that will make everything easier.

The aim of the project is to develop a stock management system that the users can save, edit, add stock or make any arrangements on orders or databases related with stock , customers and sales .

Stock Management simply enables companies to manage and control their inventory stock and it aides businesses in many areas. This ranges from assisting them in having complete control over the storage, quantity, and movement of their stock.

## INTRODUCTION

The purpose of the project is to design user friendly business software product. This business software is designed to help businesses improve while reducing costs. Since it is time saving and need minimal effort to run and follow the business with the use of such kind of software, I preferred to develop this for the user who would like to utilize or make use of this product which serves simplicity. For the successful execution of a project, effective planning is essential.

While planning what the necessary things and features should be in the content of the software and what functions should be fulfilled in the software, more and more functions seemed to be considered to design a proper and well working business software product. In order to develop a product which can serve its users, all the details should be considered in depth. In this software, all necessary things were thought in detail and the software which can also be suited to all business sizes was designed.

The aim of this project is to develop a simple Stock Management System for small companies. The project consists of introduction, three chapters and conclusion.

Chapter One describes general terms of Delphi Programming and specific details about Delphi Components and coding structures.

Chapter Two describes the main lines of Borland Delphi Databases and controls. It includes the Borland Database Engine Description and Paradox databases explanations in details.

Chapter Three is the User's Manual of the program that gives information about the system which is developed as Stock Management System.



# **CHAPTER ONE**

## **1.DELPHI PROGRAMMING LANGUAGE**

### **1.1 Introduction**

Object (or Delphi) Pascal, a set of object-oriented extensions to standard Pascal, is the language of Delphi. Delphi Pascal is a high-level, compiled, strongly typed language that supports structured and object-oriented design. Its benefits include easy-to-read code, quick compilation, and the use of multiple unit files for modular programming. Borland Delphi is a sophisticated Windows programming environment, suitable for beginners and professional programmers alike. Using Delphi you can easily create self-contained, user friendly, highly efficient Windows applications in a very short time - with a minimum of manual coding.

Delphi provides all the tools you need to develop, test and deploy Windows applications, including a large number of so-called reusable components.

Borland Delphi, in its latest version, provides a cross platform solution when used with Borland Kylix - Borland's RAD tool for the Linux platform. Borland Delphi (1/2/3/4/5) is a development tool for Microsoft Windows applications. Delphi is a powerful and easy to use tool for generating stand-alone graphical user interface (GUI) programs or 32-bit console applications (programs that have no GUI presence but instead run in what is commonly referred to as a "DOS box.")

When paired with Borland Kylix, Delphi 6 users can build single-source applications for both Windows and Linux, which opens new opportunities and increases the potential return on development investments. Use the Cross-platform CLX component library and visual designers to build high-performance portable applications for Windows that can be easily re-compiled on Linux. Delphi is the first programming language to shatter the barrier between high-level, easy-to-use rapid application development environments and low-level bits-and-bytes power tools. Delphi ships in a variety of configurations aimed at both departmental and enterprise needs. With Delphi, you can write Windows programs more quickly and more easily than was possible ever before.

Delphi can access many types of databases. Using forms and reports that you create, the BDE (Borland Database Engine) can access local databases, like Paradox and dBase, network SQL server databases, like InterBase, and SysBase, and any data source accessible through ODBC (open database connectivity).

## **1.2 What is Delphi?**

Borland Delphi is a high-level, compiled, strongly typed language that supports structured and object-oriented design. Delphi language is based on Object Pascal. Today, Delphi is much more than simply "Object Pascal language".

Borland Delphi is the first rapid application development environment for Windows that fully supports new and emerging Web Services. With Delphi, corporate or individual developers can create next-generation e-business applications quickly and easily.

## **1.3 History of Delphi**

This chapter gives information about history of Borland Delphi.

### **1.3.1 Pascal And Delphi's History**

The origin of Pascal owes much of its design to Algol - the first high-level language with a readable, structured, and systematically defined syntax. In the late sixties (1960s), several proposals for an evolutionary successor to Algol were developed. The most successful one was Pascal, defined by Prof. Niklaus Wirth. Wirth published the original definition of Pascal in 1971. It was implemented in 1973 with some modifications. Many of the features of Pascal came from earlier languages. The case statement, and value-result parameter passing came from Algol, and the records structures were similar to Cobol and PL 1. Besides cleaning up or leaving out some of Algol's more obscure features, Pascal added the capability to define new data types out of simpler existing ones. Pascal also supported dynamic data structures; i.e., data

structures which can grow and shrink while a program is running. The language was designed to be a teaching tool for students of programming classes.

In 1975, Wirth and Jensen produced the ultimate Pascal reference book "Pascal User Manual and Report". Wirth stopped its work on Pascal in 1977 to create a new language, Modula - the successor to Pascal.

With the release (November 1983) of Turbo Pascal 1.0, Borland started its journey into the world of development environments and tools. To create Turbo Pascal 1.0 Borland licensed the fast and inexpensive Pascal compiler core, written by Anders Hejlsberg. Turbo Pascal introduced an Integrated Development Environment (IDE) where you could edit the code, run the compiler, see the errors, and jump back to the lines containing those errors. Turbo Pascal compiler has been one of the best-selling series of compilers of all time, and made the language particularly popular on the PC platform.

In 1995 Borland revived its version of Pascal when it introduced the rapid application development environment named Delphi - turning Pascal into a visual programming language. The strategic decision was to make database tools and connectivity a central part of the new Pascal product.

After the release of Turbo Pascal 1, Anders joined the company as an employee and was the architect for all versions of the Turbo Pascal compiler and the first three versions of Delphi. As a chief architect at Borland, Hejlsberg secretly turned Turbo Pascal into an object-oriented application development language, complete with a truly visual environment and superb database-access features: Delphi.

### **1.3.2 Delphi Versions**

#### **Delphi-1(1995)**

Delphi, Borland's powerful Windows programming development tool first appeared in 1995. Delphi 1 extended the Borland Pascal language by providing object-orientated and form-based approach, extremely fast native code compiler, visual two-way tools and great database support, close integration with Windows and the component technology.



### **Delphi-2(1996)**

Delphi 2\* is the only Rapid Application Development tool that combines the performance of the world's fastest optimizing 32-bit native-code compiler, the productivity of visual component-based design, and the flexibility of scalable database architecture in a robust object-oriented environment.

Delphi 2, beside being developed for the Win32 platform (full Windows 95 support and integration), brought improved database grid, OLE automation and variant data type support, the long string data type and Visual Form Inheritance.

Delphi 2: "the Ease of VB with the Power of C++"

### **Delphi-3(1997)**

The most comprehensive set of visual, high-performance, client and server development tools for creating distributed enterprise and Web-enabled applications.

Delphi 3\* introduced new features and enhancements in the following areas: the code insight technology, DLL debugging, component templates, the DecisionCube and TeeChart components, the WebBroker technology, ActiveForms, component packages, and integration with COM through interfaces.

### **Delphi-4(1998)**

Delphi 4\* is a comprehensive set of professional and client/server development tools for building high productivity solutions for distributed computing. Delphi provides Java interoperability, high performance database drivers, CORBA development, and Microsoft BackOffice support. You've never had a more productive way to customize, manage, visualize and update data. With Delphi, you deliver robust applications to production, on time and on budget. Delphi 4 introduced docking, anchoring and constraining components. New features included the AppBrowser, dynamic arrays, method overloading, Windows 98 support, improved OLE and COM support as well as extended database support.

### **Delphi 5 (1999)**

Delphi 5\* introduced many new features and enhancements. Some, among many others, are: various desktop layouts, the concept of frames, parallel development, translation capabilities, enhanced integrated debugger, new Internet capabilities (XML), more database power (ADO support), etc.

### **Delphi 6(2000)**

Delphi 6 introduced new features and enhancements in IDE, Internet, XML, Compiler, COM/Active X and lastly database support areas. What's more, Delphi 6 added the support for cross-platform development – thus enabling the same code to be compiled with Delphi (under Windows) and Kylix (under Linux). More enhancements included , support for Web Services, the DBExpress engine, new components and classes.

### **Delphi 7(2001)**

For the 7th anniversary of Delphi, Borland prepared the most significant Delphi release: Delphi 7 continues to provide Visual Component Library (VCL) and Component Library for Cross-platform (CLX) development for Win32 (and Linux) as well as new features and continued framework, compiler, IDE, and design time enhancements.

## **1.4 Delphi Programming Pheriphals**

### **1.4.1 Using Project Files**

Since it is quite common for Delphi applications to share code or previously customized forms, Delphi organizes applications into what is called projects.

A project is made up of the visual interface along with the code that activates the interface. Each project can have multiple forms, allowing us to build applications that have multiple windows. The code that is needed for a form in our project is stored in a separate Unit file that Delphi automatically associates to the form. General code that we want to be shared by all the forms in our application is placed in unit files as well. Simply put, a Delphi project is a

collection of files that make up an application. What this means is that each project is made of one or more Form files (files with the .dfm extension) and one or more Unit files (.pas extension). We can also add resource files, and they are compiled into .RES files and linked when we compile the project.

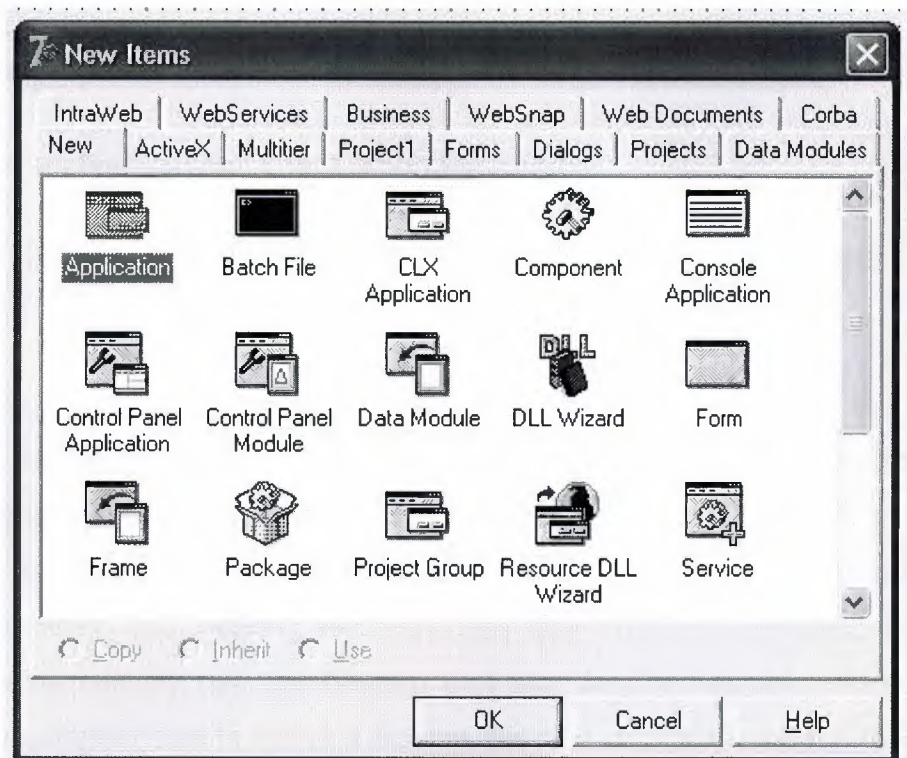


Figure 1.4.1.1 Delphi New Project View

Each project is made up of a single project file (.dpr). Project files contain directions for building an application. This is normally a set of simple routines which open the main form and any other forms that are set to be opened automatically and then starts the program by calling the *Initialize*, *CreateForm* and *Run* methods of the global Application object (which is actually a form of zero width and height, so it never actually appears on the screen).



## 1.4.2 Project Unit

A program is constructed from source-code modules called units. Each unit is stored in its own file and compiled separately; compiled units are linked to create an application. Units allow you to

- divide large programs into modules that can be edited separately.
- create libraries that you can share among programs.
- distribute libraries to other developers without making the source code available.

In traditional Pascal programming, all source code, including the main program, is stored in .pas files. Borland tools use a project (.dpr) file to store the "main" program, while most other source code resides in unit (.pas) files. Each application--or project--consists of a single project file and one or more unit files. (Strictly speaking, you needn't explicitly use any units in a project, but all programs automatically use the System unit and the SysInit unit.) To build a project, the compiler needs either a source file or a compiled unit file for each unit.

Although you can look and edit the Project File, in most cases, you'll let Delphi maintain the DPR file. The main reason to view the project file is so we can see the units and forms that make up the project, and which form is specified as the application's main form.

Another reason to work with the project file is when we are creating a DLL rather than a stand-alone application or need some start-up code, such as a splash screen before the main form is created by Delphi.

Here is the default project file for a new application (containing one form:

"Form1"):

```
program Project1;  
uses
```



```

Forms,
Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}
begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1) ;
    Application.Run;
end.

```

The **program** identifies this unit as a program's main source unit. You can see that the unit name, Project1, follows the program keyword (Delphi gives the project a default name until you save the project with a more meaningful name). When we run a project file from the IDE, Delphi uses the name of the Project file for the name of the EXE file that it creates.

Delphi reads the **uses** clause of the project file to determine which units are part of a project.

The .dpr file is linked with the .pas file with the compile directive `{$R *.RES}` (in this case "\*" represents the root of the .pas filename rather than "any file"). This compiler directive tells Delphi to include this project's resource file. The project's resource file contains such items as the project's icon image.

The **begin..end** block is the main source-code block for the project.

Although **Initialize** is the first method called in the main project source code, it is not the first code that is executed in an application. The application first executes the **"initialization" section of all the units** used by the application.

The **Application.CreateForm** statement loads the form specified in its argument. Delphi adds an **Application.CreateForm** statement to the project file for each form you add to the project. This code's job is to first allocate memory for the form. The statements are listed in the order the forms are added to the project. This is the order that the forms will be created in memory at runtime. If

you want to change this order, do not edit the project source code. Use the Project|Options menu command.

The **Application.Run** statement starts your application. This instruction tells the predeclared object called Application to begin processing the events that occur during the run of a program.

Form objects are the basic building blocks of a Delphi application, the actual windows with which a user interacts when they run the application. Forms have their own properties, events, and methods with which you can control their appearance and behavior. A form is actually a Delphi component, but unlike other components, a form doesn't appear on the component palette.

We normally create a form object by starting a new application (File | New Application). This newly created form will be, by default, the application's main form - the first form created at runtime.

Note: To add an additional form to Delphi project, we select File|New Form. There are, of course, other ways to add a "new" form to a Delphi project.

### 1.4.3 Data Types And Variables

A type is essentially a name for a kind of data. When you declare a variable you must specify its type, which determines the set of values the variable can hold and the operations that can be performed on it. Every expression returns data of a particular type, as does every function. Most functions and procedures require parameters of specific types.

The Delphi language is a "strongly typed" language, which means that it distinguishes a variety of data types and does not always allow you to substitute one type for another. This is usually beneficial because it lets the compiler treat data intelligently and validate your code more thoroughly, preventing hard-to-diagnose runtime errors. When you need greater flexibility, however, there are mechanisms to circumvent strong typing. These include typecasting, pointers, variants, variant parts in records, and absolute addressing of variables.

There are several ways to categorize Delphi data types:

Some types are predefined (or built-in); the compiler recognizes these automatically, without the need for a declaration. Almost all of the types documented in this language reference are predefined. Other types are created by declaration; these include user-defined types and the types defined in the product libraries.

Types can be classified as either fundamental or generic. The range and format of a fundamental type is the same in all implementations of the Delphi language, regardless of the underlying CPU and operating system. The range and format of a generic type is platform-specific and could vary across different implementations. Most predefined types are fundamental, but a handful of integer, character, string, and pointer types are generic. It's a good idea to use generic types when possible, since they provide optimal performance and portability. However, changes in storage format from one implementation of a generic type to the next could cause compatibility problems--for example, if you are streaming content to a file as raw, binary data, without type and versioning information.

Types can be classified as simple, string, structured, pointer, procedural, or variant. In addition, type identifiers themselves can be regarded as belonging to a special "type" because they can be passed as parameters to certain functions (such as High, Low, and SizeOf).

The outline below shows the taxonomy of Delphi data types.

- simple
- ordinal
- integer
- character
- Boolean
- enumerated
- subrange
- real

string  
structured  
set  
array  
record  
file  
class  
class reference  
interface  
pointer  
procedural  
variant

### **Type Identifier**

The standard function `SizeOf` operates on all variables and type identifiers. It returns an integer representing the amount of memory (in bytes) required to store data of the specified type. For example, `SizeOf(Longint)` returns 4, since a Longint variable uses four bytes of memory.

Type declarations are illustrated in the sections that follow. For general information about type declarations, see *Declaring types*.

## **1.4.4 Procedures And Functions**

Procedures and functions, referred to collectively as routines, are self-contained statement blocks that can be called from different locations in a program. A function is a routine that returns a value when it executes. A procedure is a routine that does not return a value.

Function calls, because they return a value, can be used as expressions in assignments and operations. For example,

```
I := SomeFunction(X);
```



calls SomeFunction and assigns the result to I. Function calls cannot appear on the left side of an assignment statement.

Procedure calls--and, when extended syntax is enabled ({ $\$X+$ }), function calls--can be used as complete statements. For example,

```
DoSomething;
```

calls the DoSomething routine; if DoSomething is a function, its return value is discarded.

Procedures and functions can call themselves recursively.

When you declare a procedure or function, you specify its name, the number and type of parameters it takes, and, in the case of a function, the type of its return value; this part of the declaration is sometimes called the prototype, heading, or header. Then you write a block of code that executes whenever the procedure or function is called; this part is sometimes called the routine's body or block.

A procedure declaration has the form

```
procedure procedureName(parameterList); directives;  
    localDeclarations;  
    begin  
        statements  
    end;
```

where procedureName is any valid identifier, statements is a sequence of statements that execute when the procedure is called, and (parameterList), directives;, and localDeclarations; are optional.

Here is an example of a procedure declaration:

```
procedure NumString(N: Integer; var S: string);  
var  
  V: Integer;  
begin  
  V := Abs(N);  
  S := "";  
  repeat  
    S := Chr(V mod 10 + Ord('0')) + S;  
    V := V div 10;  
  until V = 0;  
  if N < 0 then S := '-' + S;  
end;
```

Given this declaration, you can call the NumString procedure like this:

```
NumString(17, MyString);
```

This procedure call assigns the value "17" to MyString (which must be a string variable).

Within a procedure's statement block, you can use variables and other identifiers declared in the localDeclarations part of the procedure. You can also use the parameter names from the parameter list (like N and S in the previous example); the parameter list defines a set of local variables, so don't try to

redeclare the parameter names in the localDeclarations section. Finally, you can use any identifiers within whose scope the procedure declaration falls.

Most procedure and function headers include a parameter list. For example, in the header

```
function Power(X: Real; Y: Integer): Real;
```

the parameter list is (X: Real; Y: Integer).

A parameter list is a sequence of parameter declarations separated by semicolons and enclosed in parentheses. Each declaration is a comma-delimited series of parameter names, followed in most cases by a colon and a type identifier, and in some cases by the = symbol and a default value. Parameter names must be valid identifiers. Any declaration can be preceded by var, const, or out (see Parameter semantics). Examples:

```
(X, Y: Real)
```

```
(var S: string; X: Integer)
```

```
(HWnd: Integer; Text, Caption: PChar; Flags: Integer)
```

```
(const P; I: Integer)
```

The parameter list specifies the number, order, and type of parameters that must be passed to the routine when it is called. If a routine does not take any parameters, omit the identifier list and the parentheses in its declaration:

```
procedure UpdateRecords;
```

```
begin
```

```
...
```

```
end;
```

Within the procedure or function body, the parameter names (X and Y in the first example) can be used as local variables. Do not redeclare the

parameter names in the local declarations section of the procedure or function body.

### **Calling Procedures:**

When you call a procedure or function, program control passes from the point where the call is made to the body of the routine. You can make the call using the routine's declared name (with or without qualifiers) or using a procedural variable that points to the routine. In either case, if the routine is declared with parameters, your call to it must pass parameters that correspond in order and type to the routine's parameter list. The parameters you pass to a routine are called actual parameters, while the parameters in the routine's declaration are called formal parameters.

When calling a routine, remember that expressions used to pass typed `const` and `value` parameters must be assignment-compatible with the corresponding formal parameters.

Expressions used to pass `var` and `out` parameters must be identically typed with the corresponding formal parameters, unless the formal parameters are untyped.

only assignable expressions can be used to pass `var` and `out` parameters.

if a routine's formal parameters are untyped, numerals and true constants with numeric values cannot be used as actual parameters.

When you call a routine that uses default parameter values, all actual parameters following the first accepted default must also use the default values; calls of the form `SomeFunction(,X)` are not legal.

You can omit parentheses when passing all and only the default parameters to a routine. For example, given the procedure

```
procedure DoSomething(X: Real = 1.0; I: Integer = 0; S: string = "");
```

the following calls are equivalent.

```
DoSomething();
```

```
DoSomething;
```



### 1.4.5 Classes and Objects

A class, or class type, defines a structure consisting of fields, methods, and properties. Instances of a class type are called objects. The fields, methods, and properties of a class are called its components or members.

A field is essentially a variable that is part of an object. Like the fields of a record, a class's fields represent data items that exist in each instance of the class.

A method is a procedure or function associated with a class. Most methods operate on objects--that is, instances of a class. Some methods (called class methods) operate on class types themselves.

A property is an interface to data associated with an object (often stored in a field). Properties have access specifiers, which determine how their data is read and modified. From other parts of a program--outside of the object itself--a property appears in most respects like a field.

Objects are dynamically allocated blocks of memory whose structure is determined by their class type. Each object has a unique copy of every field defined in the class, but all instances of a class share the same methods. Objects are created and destroyed by special methods called constructors and destructors.

A variable of a class type is actually a pointer that references an object. Hence more than one variable can refer to the same object. Like other pointers, class-type variables can hold the value nil. But you don't have to explicitly dereference a class-type variable to access the object it points to. For example, `SomeObject.Size := 100` assigns the value 100 to the Size property of the object referenced by `SomeObject`; you would not write this as `SomeObject^.Size := 100`.

## Class Types

A class type must be declared and given a name before it can be instantiated. (You cannot define a class type within a variable declaration.) Declare classes only in the outermost scope of a program or unit, not in a procedure or function declaration.

A class type declaration has the form

```
type className = class (ancestorClass)
    memberList
end;
```

where `className` is any valid identifier, `(ancestorClass)` is optional, and `memberList` declares members--that is, fields, methods, and properties--of the class. If you omit (

`ancestorClass)`, then the new class inherits directly from the predefined `TObject` class. If you include `(ancestorClass)` and `memberList` is empty, you can omit `end`. A class type declaration can also include a list of interfaces implemented by the class; see [Implementing interfaces](#).

Methods appear in a class declaration as function or procedure headings, with no body. Defining declarations for each method occur elsewhere in the program.

For example, here is the declaration of the `TMemoryStream` class from the `Classes` unit.

```
type
    TMemoryStream = class(TCustomMemoryStream)
```

```

private
    FCapacity: Longint;
    procedure SetCapacity(NewCapacity: Longint);
protected
    function Realloc(var NewCapacity: Longint): Pointer; virtual;
    property Capacity: Longint read FCapacity write SetCapacity;
public
    destructor Destroy; override;
    procedure Clear;
    procedure LoadFromStream(Stream: TStream);

    procedure LoadFromFile(const FileName: string);
    procedure SetSize(NewSize: Longint); override;
    function Write(const Buffer; Count: Longint): Longint; override;
end;

```

TMemoryStream descends from TStream (in the Classes unit), inheriting most of its members. But it defines--or redefines--several methods and properties, including its destructor method, Destroy. Its constructor, Create, is inherited without change from TObject, and so is not redeclared. Each member is declared as private, protected, or public (this class has no published members); for explanations of these terms, see Visibility of class members.

Given this declaration, you can create an instance of TMemoryStream as follows:

```

var stream: TMemoryStream;

stream := TMemoryStream.Create;

```

## Fields

A field is like a variable that belongs to an object. Fields can be of any type, including class types. (That is, fields can hold object references.) Fields are usually private.

To define a field member of a class, simply declare the field as you would a variable. All field declarations must occur before any property or method declarations. For example, the following declaration creates a class called TNumber whose only member, other than the methods it inherits from TObject, is an integer field called Int.

```
type TNumber = class
  Int: Integer;
end;
```

Fields are statically bound; that is, references to them are fixed at compile time. To see what this means, consider the following code.

```
type
  TAncestor = class
    Value: Integer;
  end;
  TDescendant = class(TAncestor)
    Value: string; // hides the inherited Value field
  end;
var
  MyObject: TAncestor;
begin
  MyObject := TDescendant.Create;
```



```
MyObject.Value := 'Hello!'; // error  
(MyObject as TDescendant).Value := 'Hello!'; // works!  
end;
```

Although MyObject holds an instance of TDescendant, it is declared as TAncestor. The compiler therefore interprets MyObject.Value as referring to the (integer) field declared in TAncestor. Both fields, however, exist in the TDescendant object; the inherited Value is hidden by the new one, and can be accessed through a typecast.

#### **1.4.6 Libraries And Packages :**

A dynamically loadable library is a dynamic-link library (DLL) on Windows or a shared object library file on Linux. It is a collection of routines that can be called by applications and by other DLLs or shared objects. Like units, dynamically loadable libraries contain sharable code or resources. But this type of library is a separately compiled executable that is linked at runtime to the programs that use it.

To distinguish them from stand-alone executables, on Windows files containing compiled DLLs are named with the .DLL extension. On Linux, files containing shared object files are named with a .so extension. Delphi programs can call DLLs or shared objects written in other languages, and applications written in other languages can call DLLs or shared objects written in Delphi.

#### **Calling Dynamically loadable libraries**

You can call operating system routines directly, but they are not linked to your application until runtime. This means that the library need not be present when you compile your program. It also means that there is no compile-time validation of attempts to import a routine.

Before you can call routines defined in a shared object, you must import them. This can be done in two ways: by declaring an external procedure or function, or by direct calls to the operating system. Whichever method you use, the routines are not linked to your application until runtime.

The Delphi language does not support importing of variables from shared libraries.

### **Static loading**

The simplest way to import a procedure or function is to declare it using the external directive. For example,

On Windows:

```
procedure DoSomething; external 'MYLIB.DLL';
```

On Linux:

```
procedure DoSomething; external 'mylib.so';
```

If you include this declaration in a program, MYLIB.DLL (Windows) or mylib.so (Linux) is loaded once, when the program starts. Throughout execution of the program, the identifier DoSomething always refers to the same entry point in the same shared library.

Declarations of imported routines can be placed directly in the program or unit where they are called. To simplify maintenance, however, you can collect external declarations into a separate "import unit" that also contains any constants and types required for interfacing with the library. Other modules that use the import unit can call any routines declared in it.

## 2.1 Borland Database Engine

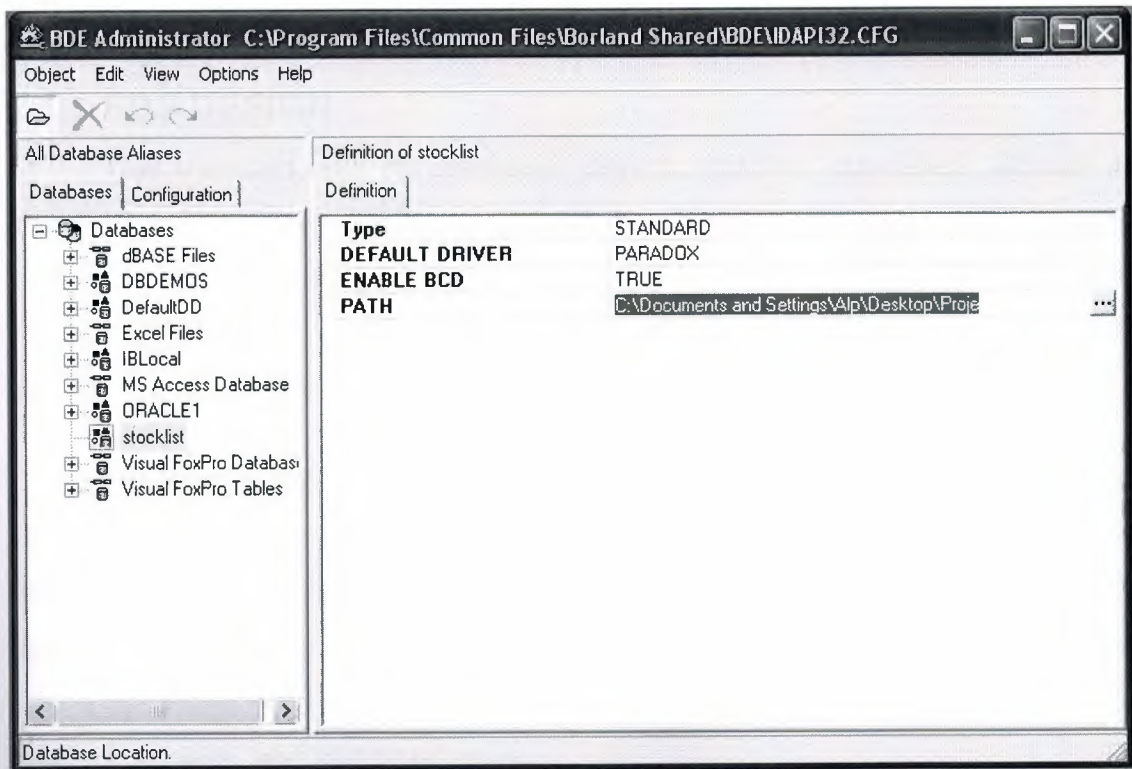


Figure 2.1.1 BDE Screen

### 2.1.1 What is BDE?

Borland Database Engine (BDE) is 32-bit Windows-based core database engine and connectivity software behind Borland Delphi, C++Builder, IntraBuilder, Paradox for Windows, and Visual dBASE for Windows.

### 2.1.2 History of BDE

Borland's Turbo Pascal included a "database" Toolbox, it was the beginning of the Borland compiler add-ons that facilitated database connectivity. Then came the Paradox Engine for Windows – PXENGWIN – which could be compiled into a program to facilitate connectivity to Paradox tables.

The first DLL-based connectivity engine was ODAPI (Open Database API). It represented Borland's attempt to centralise connectivity in its suite of applications which included the brand-new Paradox for Windows 4 and Quattro.



With version 4.5 / 5.0 of Paradox for Windows, this database engine was crystallised as IDAPI. In 2000, Borland introduced a new SQL driver architecture called dbExpress, which deprecated BDE SQL links technology.

### **2.1.3 BDE DESIGN**

The included set of database drivers enables consistent access to standard data sources: Paradox, dBASE, FoxPro, Access, and text databases. You can add Microsoft ODBC drivers as needed to the built-in ODBC socket. Optionally, Borland's SQL Links product provides access to a range of SQL servers, including Informix, DB2, InterBase, Oracle, and Sybase.

BDE is object-oriented in design. At runtime, application developers interact with BDE by creating various BDE objects. These runtime objects are then used to manipulate database entities, such as tables and queries. BDE's application program interface (API) provides direct C and C++ optimized access to the database engine, as well as BDE's built-in drivers for dBASE, Paradox, FoxPro, Access, and text databases.

The core database engine files consist of a set of DLLs that are fully re-entrant and thread-safe. Included with BDE are a set of supplemental tools and examples with sample code. BDE system is configured using the BDE Administrator (BDEADMIN.EXE).

Included with BDE is Borland's Local SQL, a subset of ANSI-92 SQL enhanced to support Paradox and dBASE (standard) naming conventions for tables and fields (called "columns" in SQL). Local SQL lets you use SQL to query "local" standard database tables that do not reside on a database server as well as "remote" SQL servers. Local SQL is also essential to make multi-table queries across both local standard tables and those on remote SQL servers.

The older name for the BDE API is the "Integrated Database Application Program Interface" or "IDAPI".



## **2.2 Paradox Database**

### **2.2.1 Paradox Database Fundamentals**

Paradox for Windows was a distinctly different product produced by a different team of programmers. Although key features of the DOS product, the QBE and the database engine, were ports keeping the DOS code, there was a major break in compatibility from PAL to ObjectPAL and in the shift to a GUI design metaphor for Forms and Reports. The ObjectPAL changes were controversial but forced since PAL was based on keystroke recording actions that had no equivalent in Windows. An object-based language based on ideas from Hypercard was used in place of keystroke recording. The Forms and Reports designers used device independent scaling including ability to work in zoomed mode for detailed layout. The mouse right-click was used for access to Forms and Reports properties, inspired by the Xerox Alto and Smalltalk, in a way now almost universal to Windows programs. The ObjectPAL was (like Hypercard) associated with the visual objects - also revealed by right click. Property inspection and layout tools could be "pinned up" to stay on screen, an idea borrowed from the NeXT and now fairly widely adopted in Windows.

For approximately the first year of development the object-oriented code was written in C aided by macros, until Turbo C++ was available at which point the remaining parts of the code were written in C++. The product manager up until shipping version 1.0 was Joe Duncan. The development and QA team totaled about 30 people.

Both Paradox for Windows and Quattro Pro for Windows, a closely related project, started development using beta versions of Windows 3.0, in the spring of 1990. Paradox/Windows ended up delayed about a year beyond its original plan, shipping in early 1993. The reasons were many, but not entirely surprising for a major rewrite, in OO language with new tools, shifting to a GUI paradigm, on what was essentially a first version operating system. Still it was a big problem for the company and Microsoft managed to ship Access a couple of months ahead of Paradox for Windows, a major marketing win to Microsoft.

In 1990 Borland also started work on an internal dBASE clone for both DOS and Windows, written in assembler, which was planned to ship in 1992. By early 1992 it became clear that Ashton-Tate was in difficulties on developing Windows versions of their products and so Borland switched plans, instead acquiring the company and anointing their internal project as the official successor. Part of the Ashton-Tate acquisition was the Interbase database and it was decided that Paradox/W should be able to work with Interbase as well as the Paradox engine and this led to the creation of an IDAPI engine based around Interbase.

The acquisition also shifted focus. Paradox had historically competed against dBASE in some markets, and Paradox/W originally was designed to improve the competitive position in the developer-oriented market. After dBASE was acquired this was no longer desirable and emphasis shifted towards an ease-of-use market. However the product could not be changed to match the emphasis (this occurred in later releases) at that late stage, making the product somewhat over complex for the entry level market. Access did a good job of addressing that same market and got there first, by Christmas 1992. Still, Paradox/W sold well for a while. Meanwhile, Borland was going through some serious problems caused by the Ashton-Tate acquisition. Many product lines were discontinued, corporate reorganization and consolidation was painful, and even worse the internal dBASE project at the center of the acquisition rationale was eventually cancelled for technical reasons leaving Borland with a collapse in revenues and a serious need to develop the missing dBASE for Windows in a hurry. Borland had lost the strength to fight the multiple marketing battles it needed for its range of products. Paradox was minimally marketed to the developers since the company decided it would hold out for a replacement of dBASE, which eventually came out in 1994, too late for the company.

Microsoft Access was sold for a fraction of the price of Paradox/Windows and bundled with Word, Excel and PowerPoint in Microsoft Office Professional. Furthermore, Access performance was good thanks to team contributions from FoxPro programmers. Despite solid follow-on versions with improvements to usability for entry-level users, Paradox faded from the market. It was included in

the sale of Borland products to Word Perfect, which were in turn resold as Word Perfect got into financial products, and at the current time of writing Paradox for Windows, Word Perfect and Quattro Pro for Windows are all owned by Corel and sold as part of their office suite. dBASE for Windows came out too late to be a significant player in the Windows market, most dBASE programmers by then had migrated to Microsoft FoxBase, a very similar database tool. Borland itself retained the Interbase/IDAPI server and focussed efforts on its Delphi tools which over the years gave it an influential but small part of the data-oriented developer market.

## **2.2.2 Paradox Table Field Types**

### **Alpha (A)**

Paradox 3.5, 4, 5, and 7 field type that can contain up to 255 letters and numbers. This field type was called Alphanumeric in versions of Paradox before version 5. It is similar to the Character field type in dBASE.

### **Autoincrement (+)**

Field type introduced in the Paradox 5 table format that adds one to the highest number in the table whenever a record is inserted. The starting range can from -2,147,483,647 to 2,147,483,647. Deleting a record does not change the field values of other records.

### **BCD (#)**

Paradox 5 and 7 field type which is provided only for compatibility with other applications that use BCD data. Paradox correctly interprets BCD data from other applications that use the BCD type. When Paradox performs calculations on BCD data, it converts the data to the numeric float type, then converts the result back to BCD. When this field type is fully supported, it will support up to 32 significant digits.

### **Binary (B)**

Paradox 1, 5, and 7 field type that can store binary data up to 256MB per field.



**Bytes (Y)**

Paradox 5 and 7 field type for storing binary data up to 255 bytes. Unlike binary fields, bytes fields are stored in the Paradox table (rather than in the separate .MB file), allowing for faster access.

**Date (D)**

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

**Formatted Memo (F)**

Paradox 1, 4.5, 5, and 7 field type is like a memo field except that you can format the text. You can alter and store the text attributes of typeface, style, color, and size. This rich text document has a variable-length up to 256MB per field.

**Graphic (G)**

Paradox 1, 5, and 7 field type can contain pictures in .BMP (up to 24 bit), .TIF (up to 256 color), .GIF (up to 256 color), .PCX, and .EPS file formats. Not all graphic variations are available. For example, currently you cannot store a 24-bit .TIF graphic. When you paste a graphic into a graphic field, Paradox converts the graphic into the .BMP format.

**Logical (L)**

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

**Memo (M)**

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

For Paradox tables, the file is divided into blocks of 512 characters. Each block is referenced by a sequential number, beginning at zero. Block 0 begins with a



4-byte number in hexadecimal format, in which the least significant byte comes first. This number specifies the number of the next available block. It is, in effect, a pointer to the end of the memo file. The remainder of Block 0 isn't used.

### **Money (\$)**

Paradox 3.5, 4, 5, and 7 field type, like number fields, can contain only numbers. They can hold positive or negative values. Paradox recognizes up to six decimal places when performing internal calculations on money fields. This field type was called Currency in previous versions of Paradox.

### **OLE (O)**

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

### **Number (N)**

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

dBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format. Use number fields when you need to perform precise calculations on the field data. Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

### **Short (S)**

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal).

## Time (T)

Paradox 5 and 7 field type that can contain time times of day, stored in milliseconds since midnight and limited to 24 hours.

This field type does not store duration which is the difference between two times. For example, if you need to store the duration of a song, use an Alpha field. Whenever you need to store time, make a distinction between clock time and duration. The Time field type is perfect for clock time. Duration can be stored in an Alpha field and manipulated with code.

## TimeStamp (@)

Paradox 5 field type comprised of both date and time values. Rules for this field type are the same as those for date fields and time fields.

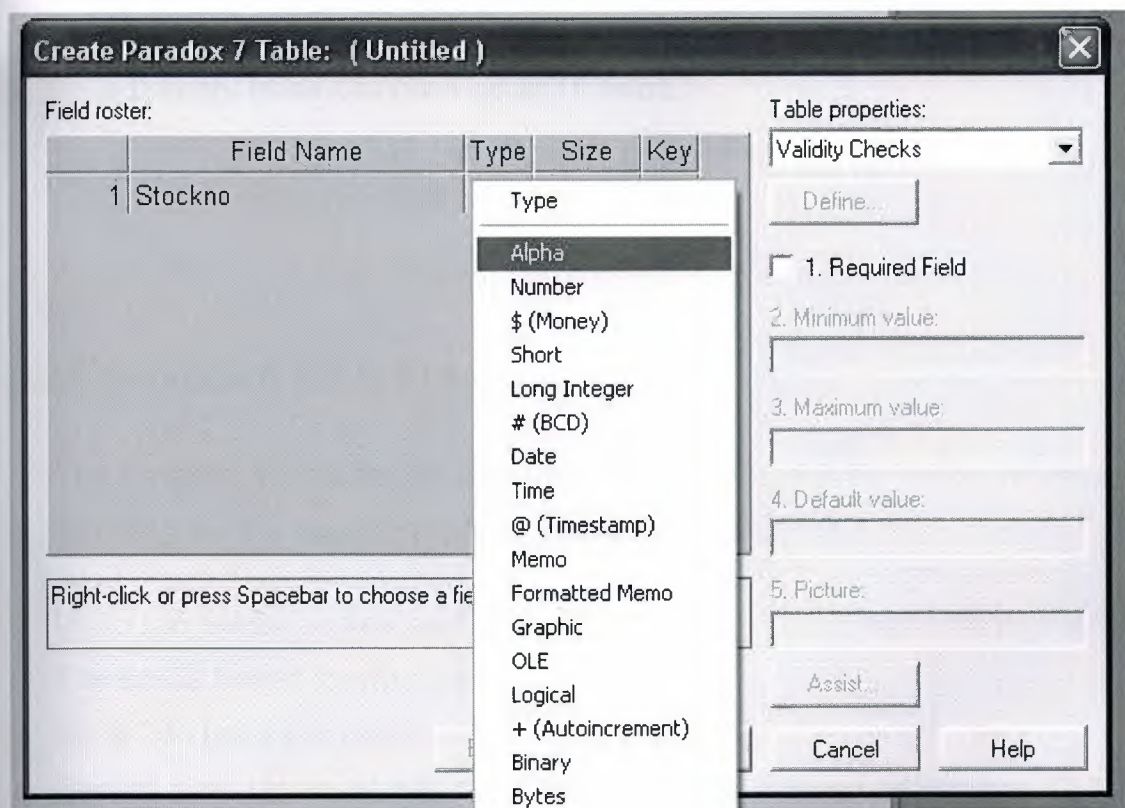


Figure 2.2.2.1 Field Types

### **2.2.3 Paradox 4 Table Structure**

The Paradox standard table format was introduced in Paradox for DOS version 4. Other products that use the standard format include Paradox for DOS version 4.5, ObjectVision 2.1, and Paradox for Windows versions 1.0 and 4.5.

Earlier versions of the Paradox table type are referred to as the Compatible table type. In the BDE Configuration Utility, the level option for the Paradox driver dictates what default table type is created by Paradox for Windows. Use 3 for Compatible tables, 4 for Standard tables (the default). Following are the specifications for standard Paradox tables:

- 256MB file size limit if the table is in Paradox format and using a 4 K block size
- Up to 255 fields per record.
- Up to 64 validity checks per table.
- A primary index can have up to 16 fields.
- Tables can have up to 127 secondary indexes.
- Up to two billion records per file.

### **2.2.4 Paradox 5 Table Structure**

The Paradox 5 table format was introduced in Paradox for Windows version 5. Following are the specifications for Paradox 5 tables.

- Up to two billion records per file.
- File size is limited to two gigabytes.
- Up to 255 fields per record.
- Record size: Up to 10,800 bytes per record for indexed tables and 32,750 bytes per record for nonindexed tables. When figuring out the size (the number of bytes or characters) of a table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.



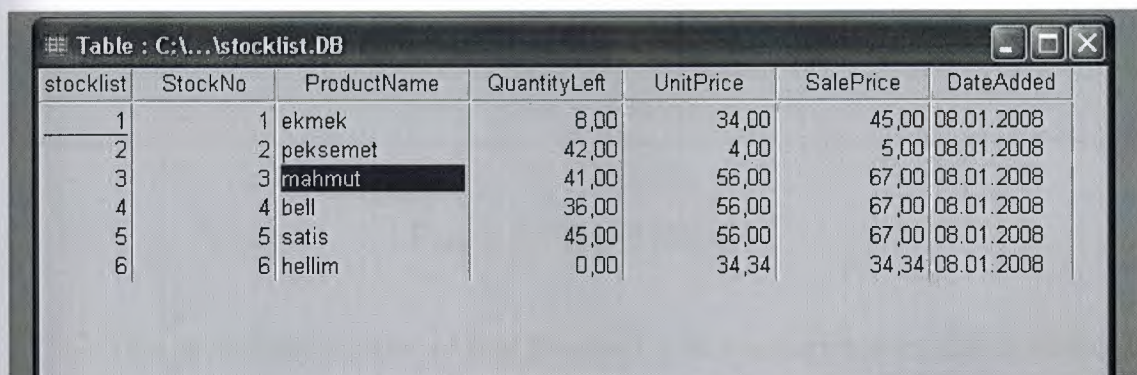
- Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.
- Up to 64 validity checks per table for Paradox for Windows tables.
- A primary index can have up to 16 fields.
- Tables can have up to 127 secondary indexes.
- Block size can be from 1K to 32K in steps of 1K. For example, 1024, 2048, 3072, 4096, 5120...32768.

## 2.2.5 Paradox 7 and Above Table Structure

The Paradox 7 table format was introduced in Paradox version 7 for Windows 95/NT. The Paradox 7 table format has all the same specifications as the Paradox 5 table format with two additions. Following are the specification additions for the Paradox 7 table format.

-Added descending secondary indexes.

-Added unique secondary indexes



stocklist	StockNo	ProductName	QuantityLeft	UnitPrice	SalePrice	DateAdded
1	1	ekmek	8,00	34,00	45,00	08.01.2008
2	2	peksemet	42,00	4,00	5,00	08.01.2008
3	3	mahmut	41,00	56,00	67,00	08.01.2008
4	4	bell	36,00	56,00	67,00	08.01.2008
5	5	satis	45,00	56,00	67,00	08.01.2008
6	6	hellim	0,00	34,34	34,34	08.01.2008

Figure 2.2.5.1 Paradox 7 Table



## CHAPTER 3

### USERS MANUAL

After executing the main program the following page welcomes us.(figure 3.1)

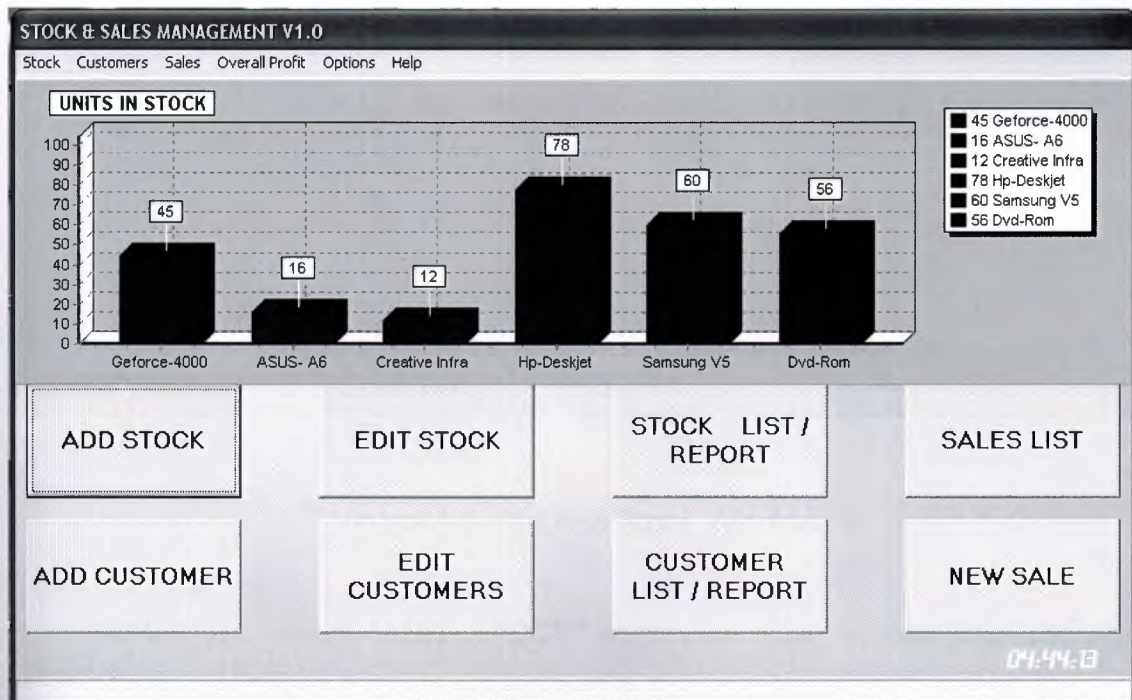


Figure 3.1 (Main Menu)

This is the main view of the program that we can select the operations that we want to proceed with buttons and a main menu in the top left position of the screen. There is a stock chart that will show you the initial number of products in units . This chart is a live chart that is active all through the usage of the program and updates with any changes in stock.

**ADD STOCK**

StockNo	ProductName	QuantityLeft	UnitPrice	SalePrice	DateAdded
7	Geforce-4000	45	56	67	10.01.2008
8	ASUS-A6	16	56	67	10.01.2008
9	Creative Infra	12	67	89	10.01.2008
10	Hp-Deskjet	78	45	67	10.01.2008
11	Samsung V5	60	45	45	10.01.2008
12	Dvd-Rom	56	78	89	10.01.2008

**Product Name :**   
**Quantity :**   
**Price :**   
**Sale Price :**   
**Date :**

Figure 3.2 The Add Stock Window

When you select "ADD STOCK" button on main screen then the add stock window in figure 3.2 will be opened automatically to add new stocks into our stocklist. After this you have to click "NEW" button to enter details of the new product , then you have to click "ADD" button to add product with database safety.

If you press the "CANCEL" button then the program will cancel the record and everything you wrote will be truncated automatically. When you press the "BACK" button you will be redirected to the main screen.

**7 EDIT STOCK**

**SELECT PRODUCT :**

StockNo	ProductName	QuantityLeft	UnitPrice	SalePrice	DateAdded
7	Geforce-4000	45	56	67	10.11.2008
8	ASUS- A6	16	56	67	10.01.2008
9	Creative Infra	12	67	89	10.01.2008
10	Hp-Deskjet	78	45	67	10.01.2008
11	Samsung V5	60	45	45	10.01.2008
12	Dvd-Rom	56	78	89	10.01.2008

**Stock No** :   
**Product Name** :   
**Quantity** :   
**Unit Price** :   
**Sale Price** :   
**Date** :

Figure 2.3 Edit Stock Window

In figure 2.3 we see our “EDIT STOCK” window after pressing the “EDIT STOCK” button on the main program view. In “EDIT STOCK” window you will be given the opportunity to select the products and edit their stock details which are already added to stock once a time ago.

When you press the edit button then you will be redirected to the product name input box to input the new values in order. Any time if you leave empty any boxes then you will be warned about the empty fields and “Empty Fields Detected” popup box will pop up to your screen and the record will not be saved automatically.

After Pressing back button the “EDIT STOCK” window will also close and you will be welcomed with main program screen again.



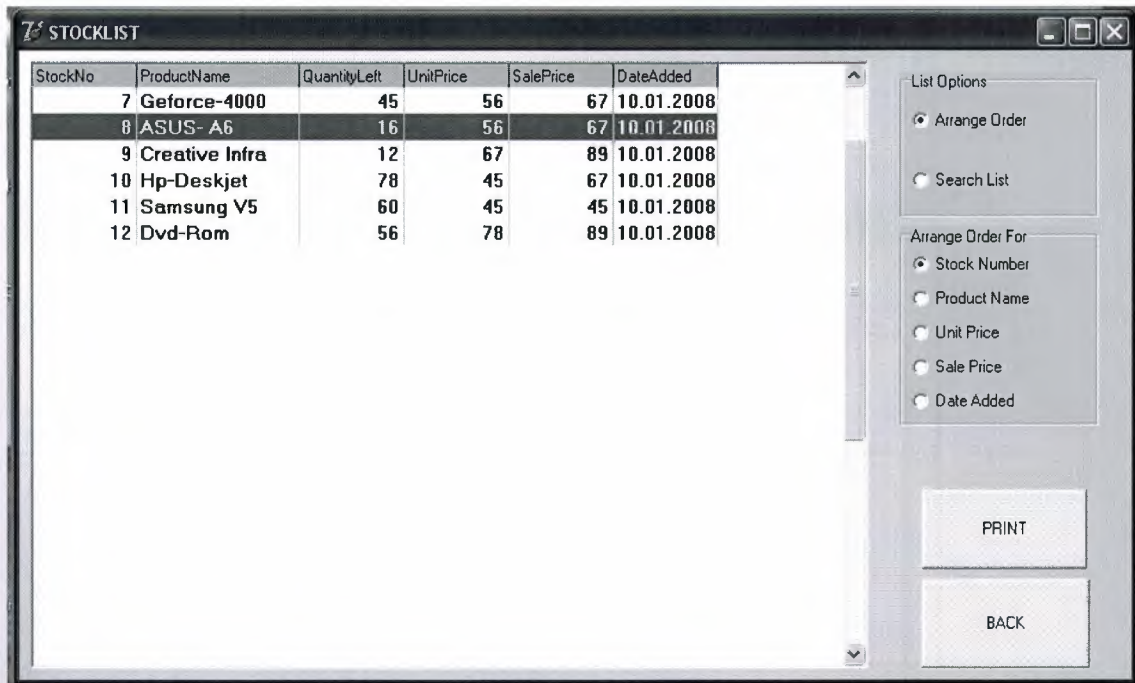


Figure 2.4 Stocklist Window

In figure 2.4 we can see the “STOCKLIST” window and this window opens when you press the “STOCK LIST/REPORT” window in the main program screen .

In this window you can view or list the products which are currently in stock . On list options screen you can arrange the list by “stock number”(numerical ordering) , “product name”(alphabetical ordering),”unit price”,”sale price”,”date added” credentials .

Also you can print the selected list on a report sheet by pressing the “PRINT” button. After pressing the “PRINT” button there will be print prier , printer setup and print immediately selection which will pop up on the screen.Then you can select the preview option to view your current stock list.(figure 2.5)



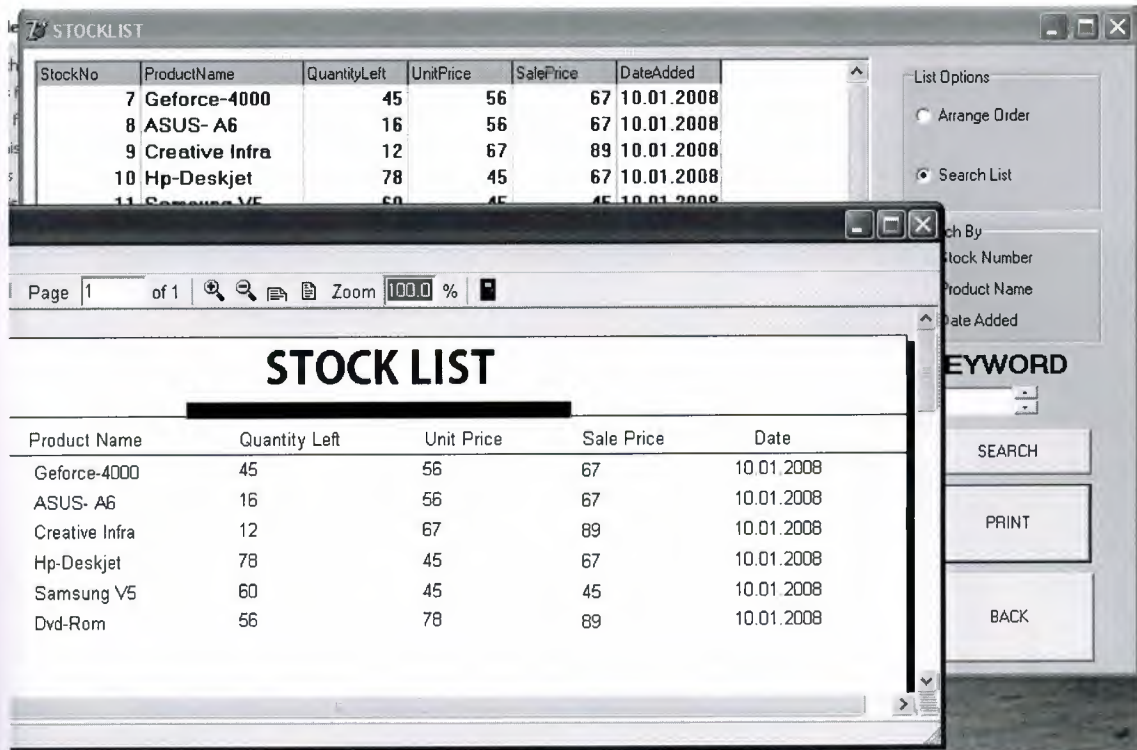


Figure 2.5 The Print Preview Window

The print Preview Screen Welcomes us with the list of products in details.

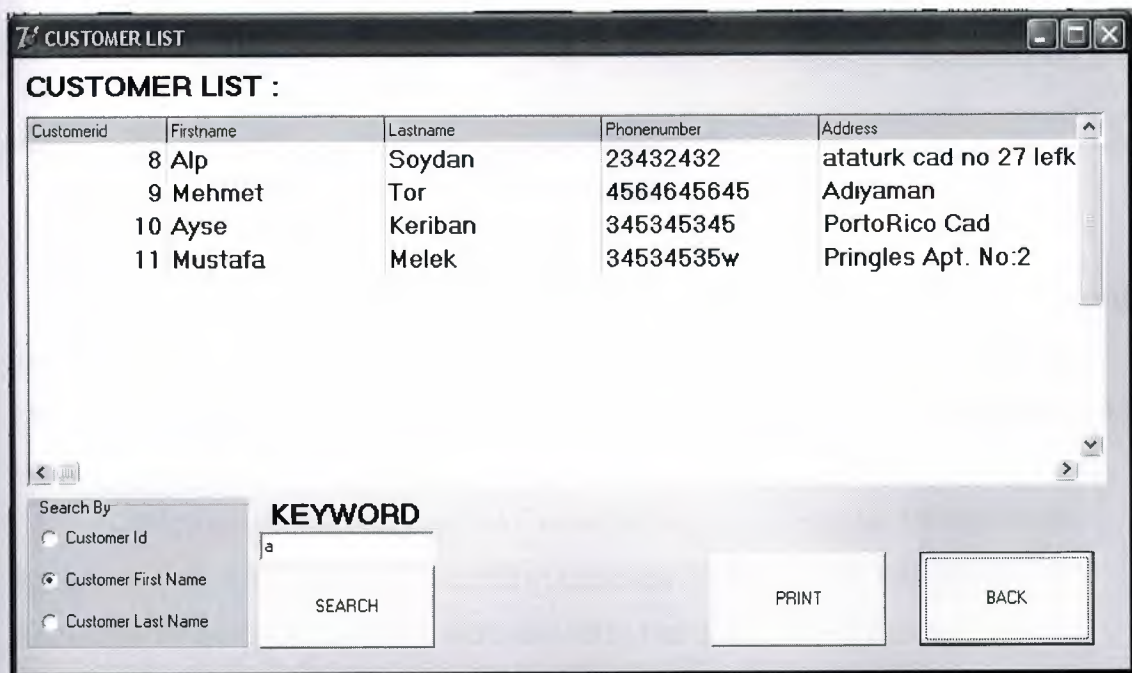


Figure 2.6 The Customer List Window

In figure 2.6 you can see the customer list screen which has same properties with stock list screen like printing the list , searching for any customer and printing the report for any customer.

The screenshot shows a software window titled "7 SALES...". It is divided into several sections:

- SELECT CUSTOMER:** A table with columns: CustomerId, Firstname, Lastname, Phonenumber, and Address. It lists five customers. Below the table is a "FAST NAME SEARCH" field and a "FIND" button.
- CUSTOMER INFORMATION:** Fields for "Customer Id", "First Name", and "Last Name". The "First Name" field contains "Alp" and the "Last Name" field contains "Soydan". There is a "Confirm Customer" checkbox.
- SELECT PRODUCT:** A table with columns: StockNo, ProductName, QuantityLeft, UnitPrice, SalePrice, and DateAdded. It lists four products. Below the table is a "FAST NAME SEARCH" field and a "FIND" button.
- ADDED PRODUCTS:** A table with columns: Product Name, Quantity, and Price. It lists three products: Hp-Deskjet (6), ASUS-A6 (5), and Hp-Deskjet (2).
- ADD PRODUCT:** A button to add new products.
- QUANTITY:** A field with a value of "0" and a spinner control.
- Confirm Products:** A checkbox.
- Sub ToTal:** A field showing "871".
- VAT%:** A field showing "15 %".
- DISCOUNT%:** A field showing "0 %".
- GRAND TOTAL:** A field showing "1001,6 YTL".
- Buttons:** "NEW SALE", "CANCEL SALE", "BACK", "PRINT REPORT", and "APPROVE SALE".

Figure 2.7 The Sales Window

When you select sales in main program window then you will be directed to sales window automatically.(Figure 2.7) In sales window you can fast name search for any customer, confirm the customer and you can add products to the shopping list with their names , quantities and prices in order .

The program will ask you VAT rate but its initially set as 15% and will apply it to Sub Total when you confirm product. Also you can make any discounts on sales if you wish with selecting the discount rate on the right bottom side of the screen.The program can print/report the current sale after checking it out.

## CONCLUSION

Although there are many types and forms of business software, the overall purpose stays the same, which is to help administration and maintain the control of the business. Buying a business software is an investment so it is advised that you must think carefully about the programs and packages that you purchase for your business; make sure that the business software that you buy is best suited to your business. It can be hard to decide what software to invest your time and money in but this decision will help your business in the long run, for example business software will smooth the transaction process as ordering and arranged data is automatically stored in your database then the products can be carefully watched, all with minimal effort.

Delphi gives us the chance to create any software for any businesses. it has many components or design tools that makes the job easier and faster than any other development platforms. I also used several components and tools to design a stock management software. Business software that was developed and designed by me allows you to control your business in a user friendly way without having to spend your entire time doing so. It allows you to be more productive, which will in turn generate better business.

Business software can help all business sizes. That's why; the software which I've designed can also be developed for bigger business if your business has larger scale. In other words, it is possible to redesign according to aim and size of the business to make it suit your business. This is the suitability of my business software project.

## REFERENCES

- [1] Mastering Borland Delphi 2005 (Mastering) by Marco Cantu' (Paperback - Aug 19, 2005)
- [2] Inside Delphi 2006 (Wordware Delphi Developer's Library) by Ivan Hladni (Paperback - Nov 25, 2005)
- [3] Introducing Delphi Programming: Theory through Practice by John Barrow, Linda Miller, Katherine Malan, and Helene Gelderblom
- [4] [www.google.com](http://www.google.com)
- [5] [www.altavista.com](http://www.altavista.com)



# APPENDIX

## PROGRAM CODE

### FORM 1

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Menus, DB, DBTables, StdCtrls, TeEngine, Series, ExtCtrls,  
TeeProcs, Chart, DbChart, jpeg, TeeFunci, ComCtrls, Mask, DBCtrls, Grids,  
DBGrids;

type

TForm1 = class(TForm)  
    MainMenu1: TMainMenu;  
    Stock1: TMenuItem;  
    ADDStock1: TMenuItem;  
    EditStock1: TMenuItem;  
    Exit1: TMenuItem;  
    Options1: TMenuItem;  
    Extras1: TMenuItem;  
    OverallProfit1: TMenuItem;  
    MonthlyProfit1: TMenuItem;  
    Help1: TMenuItem;  
    ProfitChart1: TMenuItem;  
    SellingChart1: TMenuItem;  
    Exit2: TMenuItem;  
    NewSale1: TMenuItem;  
    Help2: TMenuItem;  
    Howtos1: TMenuItem;

```
LockProgram1: TMenuItem;
DataSource1: TDataSource;
Image1: TImage;
Button1: TButton;
Button2: TButton;
Button3: TButton;
Button4: TButton;
Button5: TButton;
Button6: TButton;
Button7: TButton;
Button8: TButton;
StatusBar1: TStatusBar;
Timer1: TTimer;
Label1: TLabel;
Query1: TQuery;
Chart1: TChart;
Series1: TBarSeries;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
EditCustomers1: TMenuItem;
ListCustomers1: TMenuItem;
Stocklist1: TMenuItem;
Customer1: TMenuItem;
Sales1: TMenuItem;
DeleteAllData1: TMenuItem;
DeleteAllData2: TMenuItem;
DeleteAllData3: TMenuItem;
Query3: TQuery;
Query4: TQuery;
Query5: TQuery;
Edit2: TEdit;
Query2: TQuery;
procedure ADDStock1Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
```

```

procedure Select1Click(Sender: TObject);
procedure EditStock1Click(Sender: TObject);
procedure Exit2Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure SALES1Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Timer2Timer(Sender: TObject);
procedure Howtos1Click(Sender: TObject);
procedure NewSale1Click(Sender: TObject);
procedure EditCustomers1Click(Sender: TObject);
procedure ListCustomers1Click(Sender: TObject);
procedure ProfitChart1Click(Sender: TObject);
procedure SellingChart1Click(Sender: TObject);
procedure DeleteAllData1Click(Sender: TObject);
procedure DeleteAllData2Click(Sender: TObject);
procedure DeleteAllData3Click(Sender: TObject);
procedure MonthlyProfit1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;

```

implementation

uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit8, Unit9;

{ \$R \*.dfm }

```
procedure TForm1.ADDStock1Click(Sender: TObject);
begin
  form2.show;
  form2.Query1.Refresh;
  form1.enabled:=false;
end;
```

```
procedure TForm1.Exit1Click(Sender: TObject);
begin
  form5.Show;
  form1.enabled:=false;
end;
```

```
procedure TForm1.Select1Click(Sender: TObject);
begin
  form3.show;
end;
```

```
procedure TForm1.EditStock1Click(Sender: TObject);
begin
  form3.Show;
  form3.Query1.Refresh;
  form1.enabled:=false;
end;
```

```
procedure TForm1.Exit2Click(Sender: TObject);
var
  a:word;
```



```

begin
a:=application.MessageBox('Are you Sure?','Close Program',36);
if(a=IDYES) then
begin
form1.close;
end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
form2.show;
form2.Query1.Refresh;
form1.enabled:=False;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
form3.Show;
form3.Query1.Refresh;
form1.enabled:=false;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
form4.show;
form4.query1.Refresh;
form1.enabled:=false;
end;

procedure TForm1.SALES1Click(Sender: TObject);
begin
form4.show;
form4.Query1.refresh;

end;

```

```
procedure TForm1.Button4Click(Sender: TObject);
begin
form5.show;
form1.enabled:=false;
end;
```

```
procedure TForm1.Button5Click(Sender: TObject);
begin
form6.show;
form1.enabled:=false;
end;
```

```
procedure TForm1.Button6Click(Sender: TObject);
begin
form7.show;
form7.Query1.Refresh;
form1.enabled:=false;
end;
```

```
procedure TForm1.Button7Click(Sender: TObject);
begin
form8.show;
form8.Query1.Refresh;
form1.enabled:=false;
end;
```

```
procedure TForm1.Button8Click(Sender: TObject);
begin
form9.show;
form9.Query1.Close;
form9.Query1.Open;
form9.Query2.Close;
form9.Query2.Open;
```

```
form9.Query1.Refresh;
form9.Query2.Refresh;
form9.button7.enabled:=true;
form1.enabled:=false;
end;
```

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
query1.Open;
label1.Visible:=true;
label1.Caption:=timetostr(time);
while not query1.eof do begin
series1.AddBar(strtoint(dbedit1.text),dbedit2.text,clblue);
query1.next;
end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
label1.Visible:=false;
edit2.text:='0';
end;
```

```
procedure TForm1.Timer2Timer(Sender: TObject);
begin
form1.Visible:=false;
end;
```

```
procedure TForm1.Howtos1Click(Sender: TObject);
begin
application.MessageBox('Stock & Sales Management V1.0 , Programmed by
Alp Soydan','About',32);
end;
```

```
procedure TForm1.NewSale1Click(Sender: TObject);  
begin  
form6.show;  
form1.enabled:=false;  
end;
```

```
procedure TForm1.EditCustomers1Click(Sender: TObject);  
begin  
form7.show;  
form7.Query1.Refresh;  
form1.enabled:=false;  
end;
```

```
procedure TForm1.ListCustomers1Click(Sender: TObject);  
begin  
form8.show;  
form8.Query1.Refresh;  
form1.enabled:=false;  
end;
```

```
procedure TForm1.ProfitChart1Click(Sender: TObject);  
begin  
form9.show;  
form9.Query1.Close;  
form9.Query1.Open;  
form9.Query2.Close;  
form9.Query2.Open;  
form9.Query1.Refresh;  
form9.Query2.Refresh;  
form9.button7.enabled:=true;  
form1.enabled:=false;  
end;
```

```
procedure TForm1.SellingChart1Click(Sender: TObject);
```



```
begin
form4.show;
form4.query1.Refresh;
form1.enabled:=false;
end;
```

```
procedure TForm1.DeleteAllData1Click(Sender: TObject);
var b: integer;
var i: integer;
var a:word;
begin
a:=application.MessageBox('Are you Sure?','Clear Stock',36);
if(a=IDYES) then
begin
b:=query3.RecordCount;
query3.First;
for i:=1 to b do begin
query3.Delete;
query3.next;
end;
query3.Close;
query3.Open;
form1.series1.clear;
form1.Query1.close;
form1.query1.open;
end;
end;

procedure TForm1.DeleteAllData2Click(Sender: TObject);
var b: integer;
var i: integer;
var a:word;
begin
a:=application.MessageBox('Are you Sure?','Delete All Customers',36);
if(a=IDYES) then
```

```

begin
b:=query4.RecordCount;
query4.First;
for i:=1 to b do begin
query4.Delete;
query4.next;
end;
query4.Close;
query4.Open;
form1.series1.clear;
form1.Query1.close;
form1.query1.open;
end;
end;

```

```

procedure TForm1.DeleteAllData3Click(Sender: TObject);
var b: integer;
var i: integer;
var a:word;
begin
a:=application.MessageBox('Are you Sure?','Clear All Sales',36);
if(a=IDYES) then
begin
b:=query5.RecordCount;
query5.First;
for i:=1 to b do begin
query5.Delete;
query5.next;
end;
query5.Close;
query5.Open;
form1.series1.clear;
form1.Query1.close;
form1.query1.open;

```

```

end;
end;

procedure TForm1.MonthlyProfit1Click(Sender: TObject);
begin
query2.Close;
query2.sql.Clear;
query2.sql.Text:=('select sum(netprofit) from sales where
datedone='+#39+datetostr(date)+#39);
query2.Open;
edit2.Text:=query2.Fields[0].AsString;
showmessage('TodaysProfit Up to Now :'+ ' '+edit2.Text+' '+'YTL');
query2.Close;
end;

end.

```

## FORM 2

```
unit Unit2;
```

```
interface
```

```
uses
```

```

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

```

```
type
```

```

TForm2 = class(TForm)
    DBGrid1: TDBGrid;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;

```



```
Query1: TQuery;
DataSource1: TDataSource;
Timer1: TTimer;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
DBEdit5: TDBEdit;
Button1: TButton;
DBEdit6: TDBEdit;
Label7: TLabel;
Button2: TButton;
Button3: TButton;
Image1: TImage;
Button4: TButton;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure DBGrid1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
procedure DBGrid1CellClick(Column: TColumn);
procedure FormCreate(Sender: TObject);
procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
procedure Button1KeyPress(Sender: TObject; var Key: Char);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
```



var

Form2: TForm2;

implementation

uses Unit1;

{ \$R \*.dfm }

procedure TForm2.Button1Click(Sender: TObject);

begin

dbedit2.enabled:=false;

dbedit3.enabled:=false;

dbedit4.enabled:=false;

dbedit5.enabled:=false;

button2.enabled:=true;

button1.enabled:=false;

dbgrid1.enabled:=true;

button3.Enabled:=true;

button4.Enabled:=false;

if dbedit2.text<>'' then begin

if dbedit3.text<>'' then begin

if dbedit4.text<>'' then begin

if dbedit5.text<>'' then begin

query1.Post;

end else begin

query1.Cancel;

query1.Close;

query1.Open;

application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT  
SAVED', 'Warning', 32);

end;

end else begin

```

query1.Cancel;
query1.Close;
query1.Open;
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
end;
end else begin
query1.Cancel;
query1.Close;
query1.Open;
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
end;
end else begin
query1.Cancel;
query1.Close;
query1.Open;
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
end;
end;

```

```

procedure TForm2.Button2Click(Sender: TObject);
begin
dbedit2.enabled:=true;
dbedit3.enabled:=true;
dbedit4.enabled:=true;
dbedit5.enabled:=true;
button1.Enabled:=true;
button2.enabled:=false;
button3.Enabled:=false;
dbedit2.Text:="";
dbedit3.Text:="";

```

```

dbedit4.Text:="";
dbedit5.Text:="";
dbedit2.SetFocus;
button4.enabled:=true;
query1.insert;
DBEdit6.Text:= datetostr(date);
dbgrid1.Enabled:=false;
dbedit2.SetFocus;
end;

```

```

procedure TForm2.DBGrid1MouseDown(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
dbedit2.enabled:=false;
dbedit3.enabled:=false;
dbedit4.enabled:=false;
dbedit5.enabled:=false;
button1.Enabled:=false;

end;

```

```

procedure TForm2.DBGrid1CellClick(Column: TColumn);
begin
dbedit2.enabled:=false;
dbedit3.enabled:=false;
dbedit4.enabled:=false;
dbedit5.enabled:=false;
button1.Enabled:=false;
end;

```

```

procedure TForm2.FormCreate(Sender: TObject);
begin
form2.BorderIcons:= BorderIcons - [biMaximize];

```

```
dbedit2.enabled:=false;  
dbedit3.enabled:=false;  
dbedit4.enabled:=false;  
dbedit5.enabled:=false;  
button1.Enabled:=false;
```

```
end;
```

```
procedure TForm2.DBEdit1KeyPress(Sender: TObject; var Key: Char);
```

```
begin  
if(key=#13) then dbedit2.SetFocus;  
end;
```

```
procedure TForm2.DBEdit2KeyPress(Sender: TObject; var Key: Char);
```

```
begin  
if(key=#13)then dbedit3.SetFocus;  
end;
```

```
procedure TForm2.DBEdit3KeyPress(Sender: TObject; var Key: Char);
```

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

```
procedure TForm2.DBEdit4KeyPress(Sender: TObject; var Key: Char);
```

```
begin  
if(key=#13) then dbedit5.SetFocus;  
end;
```

```
procedure TForm2.DBEdit5KeyPress(Sender: TObject; var Key: Char);
```

```
begin  
if(key=#13) then button1.SetFocus;  
end;
```



```
procedure TForm2.Button1KeyPress(Sender: TObject; var Key: Char);  
begin  
if(key=#13) then button2.SetFocus;  
end;
```

```
procedure TForm2.Button3Click(Sender: TObject);  
begin  
form2.Close;  
form1.enabled:=true;  
form1.series1.clear;  
form1.Query1.close;  
form1.query1.open;  
form1.SetFocus;
```

```
end;
```

```
procedure TForm2.Button4Click(Sender: TObject);  
begin  
query1.close;  
query1.Open;  
button1.Enabled:=false;  
button2.Enabled:=true;  
button4.enabled:=false;  
button3.Enabled:=true;  
dbedit2.enabled:=false;  
dbedit3.enabled:=false;  
dbedit4.enabled:=false;  
dbedit5.enabled:=false;  
dbedit6.enabled:=false;
```

```
end;
```

```
end.
```

## FORM 3

unit Unit3;

interface

uses

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

type

```
TForm3 = class(TForm)
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;
    DBEdit5: TDBEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    DataSource1: TDataSource;
    DBEdit6: TDBEdit;
    Label6: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Image1: TImage;
    Button4: TButton;
    Label7: TLabel;
```

```

procedure Button2Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure DBEdit1Chnge(Sender: TObject);
procedure DBEdit1Click(Sender: TObject);
procedure DBEdit2Click(Sender: TObject);
procedure DBEdit3Click(Sender: TObject);
procedure DBEdit4Click(Sender: TObject);
procedure DBEdit5Click(Sender: TObject);
procedure DBEdit6Click(Sender: TObject);
procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
procedure DBGrid1CellClick(Column: TColumn);
procedure Button4Click(Sender: TObject);

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

var
    Form3: TForm3;

implementation

uses Unit1;

{$R *.dfm}

```

```

procedure TForm3.Button2Click(Sender: TObject);
var
a:word;
b:string;
begin
b:=dbedit2.text;
a:=application.MessageBox('Are you sure?','Warning',36);
if(a=IDYES) then
begin
query1.Delete;
end;
end;

procedure TForm3.Button1Click(Sender: TObject);
begin
query1.edit;
button2.Enabled:=false;
button3.Enabled:=true;
dbedit1.enabled:=true;
dbedit2.enabled:=true;
dbedit3.enabled:=true;
dbedit4.enabled:=true;
dbedit5.enabled:=true;;
dbedit2.SetFocus;
dbgrid1.Enabled:=false;
button1.Enabled:=false;
button4.Enabled:=false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
button3.Enabled:=false;
button2.Enabled:=true;
button1.enabled:=true;
dbgrid1.enabled:=true;

```



```

button4.Enabled:=true;
dbedit1.Enabled:=false;
dbedit2.enabled:=false;
dbedit3.enabled:=false;
dbedit4.enabled:=false;
dbedit5.enabled:=false;
dbedit6.enabled:=false;
if dbedit2.text<>" then begin
if dbedit3.text<>" then begin
if dbedit4.text<>" then begin
if dbedit5.text<>" then begin
query1.Post;
end else begin
query1.Cancel;
query1.Close;
query1.Open;
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
end;
end else begin
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
query1.Cancel;
query1.Close;
query1.Open;
end;
end else begin
query1.Cancel;
query1.Close;
query1.Open;
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
end;
end else begin

```

```

query1.Cancel;
query1.Close;
query1.Open;
application.MessageBox('EMPTY FIELDS DETECTED : RECORD NOT
SAVED','Warning',32);
end;
end;

```

```

procedure TForm3.FormCreate(Sender: TObject);
begin
button1.Enabled:=false;
button2.Enabled:=false;
form3.Refresh;
form3.Query1.Open;
button3.enabled:=false;
dbedit1.enabled:=false;
dbedit2.enabled:=false;
dbedit3.enabled:=false;
dbedit4.enabled:=false;
dbedit5.enabled:=false;
dbedit6.enabled:=false;
end;

```

```

procedure TForm3.DBEdit1Chnge(Sender: TObject);
begin
button1.enabled:=false;
end;

```

```

procedure TForm3.DBEdit1Click(Sender: TObject);
begin
button1.enabled:=false;
end;

```

```

procedure TForm3.DBEdit2Click(Sender: TObject);

```

```
begin
button1.enabled:=false;
end;
```

```
procedure TForm3.DBEdit3Click(Sender: TObject);
begin
button1.enabled:=false;
end;
```

```
procedure TForm3.DBEdit4Click(Sender: TObject);
begin
button1.enabled:=false;
end;
```

```
procedure TForm3.DBEdit5Click(Sender: TObject);
begin
button1.enabled:=false;
end;
```

```
procedure TForm3.DBEdit6Click(Sender: TObject);
begin
button1.enabled:=false;
end;
```

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

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

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

```
procedure TForm3.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit5.SetFocus;
end;
```

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

```
procedure TForm3.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then button3.SetFocus;
end;
```

```
procedure TForm3.DBGrid1CellClick(Column: TColumn);
begin
button1.enabled:=true;
button2.enabled:=true;
end;
```

```
procedure TForm3.Button4Click(Sender: TObject);
begin
form1.enabled:=true;
form1.series1.clear;
form1.Query1.close;
form1.query1.open;
form3.close;
form1.SetFocus;
```



end;

end.

## FORM 4

unit Unit4;

interface

uses

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

type

TForm4 = class(TForm)  
    DataSource1: TDataSource;  
    Query1: TQuery;  
    Timer1: TTimer;  
    RadioGroup1: TRadioGroup;  
    Edit1: TEdit;  
    Label1: TLabel;  
    Button1: TButton;  
    DBGrid1: TDBGrid;  
    Image1: TImage;  
    Button2: TButton;  
    UpDown1: TUpDown;  
    RadioGroup2: TRadioGroup;  
    Edit2: TEdit;  
    UpDown2: TUpDown;  
    Edit3: TEdit;  
    UpDown3: TUpDown;  
    Edit4: TEdit;

```

    UpDown4: TUpDown;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure RadioGroup2Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form4: TForm4;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm4.Button1Click(Sender: TObject);
begin
    form4.Close;
    form1.enabled:=true;
    form1.setfocus;
end;

procedure TForm4.FormCreate(Sender: TObject);
begin
    edit1.text:="";
    radiogroup1.itemindex:=0;
    edit1.Visible:=true;
    edit2.Visible:=false;

```

```

edit3.Visible:=False;
edit4.Visible:=false;
updown1.Visible:=true;
updown2.visible:=false;
updown3.Visible:=false;
updown4.Visible:=false;
end;

```

```

procedure TForm4.Button2Click(Sender: TObject);
begin
if(radiogroup1.itemindex=0) then
begin
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from sales where saleid='+#39+edit1.text+#39);
query1.Open;
end;
if(radiogroup1.itemindex=1) then
begin
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from sales where customerid='+#39+(edit1.text)+#39);
query1.Open;
end;
if(radiogroup1.itemindex=2) then
begin
edit1.Text:=edit2.text+'.'+edit3.Text+'.'+edit4.text;
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from sales where datedone='+#39+(edit1.text)+#39);
query1.Open;
end;
end;
procedure TForm4.RadioGroup1Click(Sender: TObject);

```

```

begin
edit1.Text:='1';
if radiogroup1.Itemindex=0 then begin
edit1.Visible:=true;
updown1.visible:=true;
edit2.Visible:=false;
edit3.Visible:=false;
edit4.visible:=false;
updown2.Visible:=false;
updown3.Visible:=false;
updown4.visible:=false;
radiogroup2.visible:=true;
end;
if radiogroup1.itemindex=1 then begin
edit1.Visible:=true;
updown1.visible:=true;
edit2.Visible:=false;
edit3.Visible:=false;
edit4.visible:=false;
updown2.Visible:=false;
updown3.Visible:=false;
updown4.visible:=false;
radiogroup2.Visible:=true;
end;
if radiogroup1.ItemIndex=2 then begin
edit1.Visible:=false;
updown1.visible:=false;
edit2.visible:=true;
edit3.visible:=true;
edit4.visible:=true;
updown2.visible:=true;
updown3.visible:=true;
updown4.visible:=true;
radiogroup2.Visible:=false;

```



```

end;
end;

procedure TForm4.RadioGroup2Click(Sender: TObject);
begin
if (radiogroup2.Itemindex=0) then begin
updown1.Increment:=1;
end;
if(radiogroup2.itemindex=1) then begin
updown1.Increment:=10;
end;
if(radiogroup2.itemindex=2) then begin
updown1.increment:=100;

end;
end;
end.

```

## FORM 5

```
unit Unit5;
```

```
interface
```

```
uses
```

```

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls, Grids, DBGrids, DB, DBTables, jpeg, RpCon,
RpConDS, RpDefine, RpRave, Spin, ComCtrls;

```

```
type
```

```

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

```

```

RadioGroup1: TRadioGroup;
Button1: TButton;
Image1: TImage;
Button2: TButton;
Edit1: TEdit;
Label1: TLabel;
RadioGroup2: TRadioGroup;
Button3: TButton;
RvProject1: TRvProject;
RvDataSetConnection1: TRvDataSetConnection;
RadioGroup3: TRadioGroup;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
UpDown1: TUpDown;
UpDown2: TUpDown;
UpDown3: TUpDown;
Edit5: TEdit;
UpDown4: TUpDown;
procedure RadioGroup1Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure RadioGroup3Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure RadioGroup2Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form5: TForm5;

```

implementation

uses Unit1;

{ \$R \*.dfm }

```
procedure TForm5.RadioGroup1Click(Sender: TObject);
begin
  if(radiogroup1.itemindex=0) then
  begin
    query1.close;
    query1.SQL.clear;
    query1.sql.text:=('select * from stocklist order by stockno');
    query1.Open;
  end;
  if(radiogroup1.itemindex=1) then
  begin
    query1.close;
    query1.SQL.clear;
    query1.sql.text:=('select * from stocklist order by productname');
    query1.Open;
  end;
  if(radiogroup1.itemindex=2) then
  begin
    query1.close;
    query1.SQL.clear;
    query1.sql.text:=('select * from stocklist order by unitprice');
    query1.Open;
  end;
  if(radiogroup1.itemindex=3) then
  begin
    query1.close;
    query1.SQL.clear;
```

```

query1.sql.text:=('select * from stocklist order by saleprice');
query1.Open;
end;
if(radiogroup1.itemindex=4) then
begin
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist order by saleprice');
query1.Open;
end;
if(radiogroup1.itemindex=5) then
begin
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist order by dateadded');
query1.Open;
end;
end;
procedure TForm5.Button1Click(Sender: TObject);
begin
form5.close;
form1.enabled:=true;
form1.setfocus;
end;

procedure TForm5.Button2Click(Sender: TObject);
begin
if(radiogroup2.itemindex=0) then
begin
edit1.Text:=edit5.text;
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist where stockno='+#39+(edit1.text)+#39);
query1.Open;

```



```

end;
if(radiogroup2.itemindex=1) then
begin
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist where ProductName
like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
if(radiogroup2.itemindex=2) then
begin
edit1.Text:=edit2.text+'.'+edit3.Text+'.'+edit4.text;
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist where
dateadded='+#39+(edit1.text)+#39);
query1.Open;
end;
end;

```

```

procedure TForm5.Button3Click(Sender: TObject);
begin
rvproject1.Execute;
end;

```

```

procedure TForm5.RadioGroup3Click(Sender: TObject);
begin
edit5.Visible:=false;
updown4.visible:=false;
radiogroup2.ItemIndex:=-1;
if(radiogroup3.itemindex=0) then
begin
radiogroup1.Visible:=true;
radiogroup2.Visible:=false;

```

```

radiogroup1.Enabled:=true;
radiogroup2.Enabled:=false;
edit1.Visible:=False;
edit2.Visible:=false;
edit3.visible:=false;
edit4.visible:=false;
updown1.Visible:=false;
updown2.Visible:=false;
updown3.Visible:=false;
button2.visible:=false;
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist order by stockno');
query1.Open;
end;
if(radiogroup3.itemindex=1) then
begin
radiogroup1.Enabled:=false;
radiogroup1.Visible:=false;
radiogroup2.Visible:=true;
radiogroup2.Enabled:=true;
edit1.Visible:=true;
button2.visible:=true;
query1.close;
query1.SQL.clear;
query1.sql.text:=('select * from stocklist order by stockno');
query1.Open;
end;
end;

procedure TForm5.FormCreate(Sender: TObject);
begin
edit1.Text:='';
edit2.Visible:=false;

```

```

edit3.Visible:=false;
edit4.Visible:=false;
edit5.Visible:=false;
updown1.Visible:=false;
updown2.Visible:=false;
updown3.Visible:=false;
radiogroup2.Visible:=false;
edit1.Visible:=false;
button2.Visible:=false;
updown1.Min:=01;
updown1.Max:=31;
updown2.min:=01;
updown2.Max:=12;
updown3.Min:=2008;
updown3.max:=2100;
updown4.Min:=01;
updown4.visible:=false;
end;

```

```

procedure TForm5.RadioGroup2Click(Sender: TObject);
begin
if radiogroup2.Itemindex=2 then begin
edit1.Visible:=false;
updown1.visible:=true;
updown2.visible:=true;
updown3.visible:=true;
edit2.Visible:=true;
edit3.Visible:=true;
edit4.Visible:=true;
edit5.Visible:=false;
end;
if radiogroup2.itemindex=0 then begin
edit1.Visible:=false;
updown1.visible:=false;

```

```

updown2.visible:=false;
updown3.visible:=false;
edit2.Visible:=false;
edit3.Visible:=false;
edit4.Visible:=false;
edit5.visible:=true;
updown4.visible:=true;
end;
if radiogroup2.itemindex=1 then begin
edit1.Visible:=true;
edit1.Text:="";
edit1.SetFocus;
updown1.visible:=false;
updown2.visible:=false;
updown3.visible:=false;
updown3.visible:=false;
updown4.visible:=false;
edit2.Visible:=false;
edit3.Visible:=false;
edit4.Visible:=false;
edit5.Visible:=false;
end;
end;

end.

```

## FORM 6

```
unit Unit6;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```



Dialogs, Grids, DBGrids, StdCtrls, Mask, DBCtrls, DB, DBTables, jpeg,  
ExtCtrls;

type

```
TForm6 = class(TForm)
  DataSource1: TDataSource;
  Query1: TQuery;
  DBEdit2: TDBEdit;
  DBEdit3: TDBEdit;
  DBEdit4: TDBEdit;
  DBEdit5: TDBEdit;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  DBGrid1: TDBGrid;
  Label6: TLabel;
  Button1: TButton;
  Button2: TButton;
  Button3: TButton;
  Button4: TButton;
  Image1: TImage;
  procedure Button2Click(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure Button4Click(Sender: TObject);
  procedure Button3Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
```

private

```
{ Private declarations }
```

public

```
{ Public declarations }
```

end;

var

Form6: TForm6;

implementation

uses Unit2, Unit1;

{\$R \*.dfm}

procedure TForm6.Button2Click(Sender: TObject);

begin

dbedit2.enabled:=true;

dbedit3.enabled:=true;

dbedit4.enabled:=true;

dbedit5.enabled:=true;

button1.Enabled:=true;

button2.enabled:=false;

button3.Enabled:=false;

dbedit2.Text:="";

dbedit3.Text:="";

dbedit4.Text:="";

dbedit5.Text:="";

dbedit2.SetFocus;

button4.enabled:=true;

query1.insert;

dbgrid1.Enabled:=false;

end;

procedure TForm6.Button1Click(Sender: TObject);

begin

dbedit2.enabled:=false;

dbedit3.enabled:=false;

dbedit4.enabled:=false;

```
dbedit5.enabled:=false;  
button2.enabled:=true;  
button1.enabled:=false;  
dbgrid1.enabled:=true;  
button3.Enabled:=true;  
button4.Enabled:=false;  
query1.Post;  
form2.Refresh;  
end;
```

```
procedure TForm6.Button4Click(Sender: TObject);  
begin  
query1.close;  
query1.Open;  
button1.Enabled:=false;  
button2.Enabled:=true;  
button4.enabled:=false;  
button3.Enabled:=true;  
dbedit2.enabled:=false;  
dbedit3.enabled:=false;  
dbedit4.enabled:=false;  
dbedit5.enabled:=false;  
button1.Enabled:=false;  
end;
```

```
procedure TForm6.Button3Click(Sender: TObject);  
begin  
dbedit2.enabled:=false;  
dbedit3.enabled:=false;  
dbedit4.enabled:=false;  
dbedit5.enabled:=false;  
button1.Enabled:=false;  
form6.hide;
```

```
form1.enabled:=true;
```

```
form1.setfocus;
```

```
end;
```

```
procedure TForm6.FormCreate(Sender: TObject);
```

```
begin
```

```
button4.Enabled:=false;
```

```
form6.BorderIcons:= BorderIcons - [biMaximize];
```

```
dbedit2.enabled:=false;
```

```
dbedit3.enabled:=false;
```

```
dbedit4.enabled:=false;
```

```
dbedit5.enabled:=false;
```

```
button1.Enabled:=false;
```

```
end;
```

```
end.
```

## **FORM 7**

```
unit Unit7;
```

```
interface
```

```
uses
```

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

```
type
```

```
TForm7 = class(TForm)
```

```
Label1: TLabel;
```

```
Button1: TButton;
```

```
Button2: TButton;
```



```

Button3: TButton;
Button4: TButton;
DataSource1: TDataSource;
DBGrid1: TDBGrid;
Query1: TQuery;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Image1: TImage;
Label2: TLabel;
DBEdit1: TDBEdit;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure DBGrid1CellClick(Column: TColumn);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form7: TForm7;

implementation

uses Unit1;

```

```
{ $R *.dfm }
```

```
procedure TForm7.Button1Click(Sender: TObject);  
begin  
  query1.edit;  
  button2.Enabled:=false;  
  button3.Enabled:=true;  
  dbedit1.enabled:=true;  
  dbedit2.enabled:=true;  
  dbedit3.enabled:=true;  
  dbedit4.enabled:=true;  
  dbedit5.enabled:=true;  
  dbedit2.SetFocus;  
  dbgrid1.Enabled:=false;  
  button1.Enabled:=false;  
end;
```

```
procedure TForm7.Button2Click(Sender: TObject);  
var  
  a:word;  
  b:string;  
begin  
  b:=dbedit2.text;  
  a:=application.MessageBox('Are you sure?','Warning',36);  
  if(a=IDYES) then  
  begin  
    query1.Delete;  
  end;  
end;
```

```
procedure TForm7.Button4Click(Sender: TObject);  
begin
```

```
form7.Close;  
form7.Query1.cancel;  
form7.Query1.Close;  
form7.Query1.Open;  
form7.button1.enabled:=false;  
form7.button2.Enabled:=false;  
form7.button3.enabled:=false;  
form7.dbedit1.Enabled:=false;  
form7.dbedit2.Enabled:=false;  
form7.dbedit3.Enabled:=false;  
form7.dbedit4.Enabled:=false;  
form7.dbedit5.Enabled:=false;  
form7.DBGrid1.Enabled:=true;  
form1.enabled:=true;  
form1.setfocus;  
end;
```

```
procedure TForm7.Button3Click(Sender: TObject);  
begin  
  query1.Post;  
  button3.Enabled:=false;  
  button2.Enabled:=true;  
  button1.enabled:=true;  
  dbgrid1.enabled:=true;  
end;
```

```
procedure TForm7.FormCreate(Sender: TObject);  
begin  
  form7.Refresh;  
  button1.Enabled:=false;  
  button2.Enabled:=false;  
  button3.enabled:=false;  
  dbedit1.enabled:=false;  
  dbedit2.enabled:=false;
```

```

dbedit3.enabled:=false;
dbedit4.enabled:=false;
dbedit5.enabled:=false;
end;

procedure TForm7.DBGrid1CellClick(Column: TColumn);
begin
button2.Enabled:=true;
button1.enabled:=true;
end;

end.

```

## FORM 8

```
unit Unit8;
```

```
interface
```

```
uses
```

```

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, DB, DBTables, Grids, DBGrids, ExtCtrls, jpeg, ComCtrls,
RpCon, RpConDS, RpConBDE, RpDefine, RpRave;

```

```
type
```

```

TForm8 = class(TForm)
  RadioGroup1: TRadioGroup;
  DBGrid1: TDBGrid;
  Label1: TLabel;
  DataSource1: TDataSource;
  Query1: TQuery;
  Button1: TButton;
  Button2: TButton;
  Edit1: TEdit;

```

```

Image1: TImage;
Label2: TLabel;
Button3: TButton;
Edit2: TEdit;
UpDown1: TUpDown;
RvProject1: TRvProject;
RvQueryConnection1: TRvQueryConnection;
procedure Button1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure RadioGroup1Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form8: TForm8;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm8.Button1Click(Sender: TObject);
begin
    if(radiogroup1.itemindex=0) then
    begin
        edit1.text:=edit2.text;
        query1.close;
        query1.SQL.clear;
    end;
end;

```



```

query1.sql.add('select * from customer where
customerid='+#39+(edit1.text)+#39);
query1.Open;
end;
if(radiogroup1.itemindex=1) then
begin
query1.close;
query1.SQL.clear;
query1.sql.add('select * from customer where firstname
like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.itemindex=2) then
begin
query1.close;
query1.SQL.clear;
query1.sql.add('select * from customer where lastname
like'+#39+(edit1.text)+'%'+#39);
query1.Open;
end;
end;
procedure TForm8.FormCreate(Sender: TObject);
begin
edit1.text:="";
updown1.Min:=1;
updown1.Max:=1000;
end;

procedure TForm8.Button2Click(Sender: TObject);
begin
form8.Close;
form1.enabled:=true;
form1.setfocus;
end;

```

```

procedure TForm8.RadioGroup1Click(Sender: TObject);
begin
  if radiogroup1.Itemindex=0 then begin
    edit1.Visible:=false;
    updown1.visible:=true;
    edit2.Visible:=true;
    edit1.text:="";
  end;
  if radiogroup1.itemindex=1 then begin
    edit1.Visible:=true;
    updown1.visible:=false;
    edit2.Visible:=false;

    end;
  if radiogroup1.itemindex=1 then begin
    edit1.Visible:=true;
    edit1.Text:="";
    edit1.SetFocus;
    updown1.visible:=false;
    edit2.Visible:=false;
  end;
end;

procedure TForm8.Button3Click(Sender: TObject);
begin
  rvproject1.Execute;
end;

end.

```

## FORM 9

unit Unit9;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, DB, DBTables, StdCtrls, DBCtrls, dbcgrids, Grids, DBGrids, Mask,  
jpeg, ExtCtrls, ComCtrls, RpRenderPreview, RpDefine, RpRender,  
RpRenderCanvas, RpRenderPrinter, RpCon, RpConDS, RpRave;

type

TForm9 = class(TForm)  
    Label2: TLabel;  
    Query1: TQuery;  
    DataSource1: TDataSource;  
    DBGrid1: TDBGrid;  
    Label3: TLabel;  
    DBGrid2: TDBGrid;  
    Query2: TQuery;  
    DataSource2: TDataSource;  
    DBEdit1: TDBEdit;  
    DBEdit2: TDBEdit;  
    DBEdit3: TDBEdit;  
    Button1: TButton;  
    ListBox1: TListBox;  
    Label4: TLabel;  
    Label5: TLabel;  
    Edit1: TEdit;  
    Label6: TLabel;  
    Edit2: TEdit;  
    Button2: TButton;  
    Button3: TButton;

ListBox2: TListBox;  
Label7: TLabel;  
Label8: TLabel;  
Edit3: TEdit;  
Label9: TLabel;  
Label10: TLabel;  
Label11: TLabel;  
Label12: TLabel;  
Label13: TLabel;  
DBEdit4: TDBEdit;  
Label14: TLabel;  
Button4: TButton;  
CheckBox1: TCheckBox;  
DBEdit5: TDBEdit;  
ListBox3: TListBox;  
Edit4: TEdit;  
Label15: TLabel;  
Label16: TLabel;  
Edit5: TEdit;  
Label17: TLabel;  
Edit6: TEdit;  
Label18: TLabel;  
Edit7: TEdit;  
Label19: TLabel;  
Label20: TLabel;  
Button5: TButton;  
Button6: TButton;  
CheckBox2: TCheckBox;  
Button7: TButton;  
Image1: TImage;  
Image2: TImage;  
DBEdit6: TDBEdit;  
Edit8: TEdit;  
Edit9: TEdit;

```

UpDown1: TUpDown;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
Query3: TQuery;
DataSource3: TDataSource;
Label1: TLabel;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
DBEdit16: TDBEdit;
DBEdit17: TDBEdit;
DBEdit18: TDBEdit;
Query4: TQuery;
DataSource4: TDataSource;
DBEdit19: TDBEdit;
DBEdit20: TDBEdit;
RvProject1: TRvProject;
Button8: TButton;
RvDataSetConnection2: TRvDataSetConnection;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit21: TDBEdit;
DBEdit22: TDBEdit;
Query5: TQuery;
procedure Button2Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure CheckBox1Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Button5Click(Sender: TObject);

```



```

procedure CheckBox2Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure UpDown1Click(Sender: TObject; Button: TUDBtnType);
procedure DBGrid2CellClick(Column: TColumn);
procedure Button8Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

```

```

var
  Form9: TForm9;

```

```

implementation

```

```

uses Unit1;

```

```

{$R *.dfm}

```

```

procedure TForm9.Button2Click(Sender: TObject);
begin
  query1.close;
  query1.SQL.clear;
  query1.sql.add('select * from customer where firstname
  like'+#39+(edit1.text)+'%'+#39);
  query1.Open;
end;

```

```

procedure TForm9.Button1Click(Sender: TObject);
var a: extended;
begin
  form9.query4.Insert;
  dbedit11.text:=dbedit1.text;

```

```

dbedit12.text:=dbedit2.Text;
dbedit13.text:=dbedit3.Text;
dbedit14.text:=query2.Fields[1].asString;
dbedit15.text:=edit3.Text;
dbedit16.Text:=dbedit5.text;
dbedit22.Text:=datetostr(date);
dbedit21.Text:=timetostr(time);
button1.Enabled:=false;
edit8.Text:=floattostr((strtofloat(dbedit6.Text)*strtofloat(edit3.text))+strtofloat(edit
8.Text));
checkbox2.Enabled:=false;
query2.edit;
listbox1.Items.Add(query2.Fields[1].asString);
listbox2.Items.Add(edit3.text);
dbedit4.Text:= floattostr(strtofloat(dbedit4.text) - strtofloat(edit3.Text));
a:=strtofloat(dbedit5.text)*strtofloat(edit3.text);
listbox3.items.add(floattostr(a)+' '+'YTL');
edit4.Text:=floattostr(strtofloat(edit4.text)+a);
edit3.Text:='0';
form9.query4.Post;

```

```

if listbox1.Items.count>=1 then begin
checkbox1.Enabled:=true;

```

```

end;

```

```

end;

```

```

procedure TForm9.Button4Click(Sender: TObject);
begin
query4.Insert;
dbedit11.text:=dbedit1.text;
dbedit12.text:=dbedit2.Text;

```

```
dbedit13.text:=dbedit3.Text;
dbedit17.Text:=edit4.text;
dbedit18.text:=edit5.Text;
dbedit19.Text:=edit6.text;
dbedit20.text:=edit7.text;
query4.Post;
query2.applyupdates;
button8.Enabled:=true;
button7.enabled:=true;
dbedit10.Text:=floattostr((strtofloat(dbedit10.Text)-
(strtofloat(edit4.Text)*0.01*strtofloat(edit6.Text))));
query3.Post;
edit3.Text:='0';
edit8.text:='0';
edit2.enabled:=false;
dbgrid1.enabled:=false;
dbgrid2.enabled:=false;
button3.enabled:=false;
button1.enabled:=false;
button5.enabled:=false;
checkbox1.Enabled:=false;
edit3.enabled:=false;
edit5.enabled:=false;
edit6.Enabled:=false;
button4.enabled:=false;
button7.SetFocus;
dbedit1.enabled:=false;
dbedit2.enabled:=false;
dbedit3.enabled:=false;
dbgrid1.Refresh;
dbgrid2.Refresh;
form9.query2.close;
form9.query2.Open;
form9.query3.close;
```

```
form9.query3.open;  
form9.query4.Close;  
form9.query4.Open;  
form9.query1.Close;  
form9.query1.Open;  
query2.Close;  
query2.Open;  
end;
```

```
procedure TForm9.FormCreate(Sender: TObject);  
begin  
    button8.Enabled:=false;  
    edit8.text:='0';  
    edit1.enabled:=false;  
    edit2.enabled:=false;  
    edit1.Text:="";  
    edit2.Text:="";  
    edit7.Text:='0';  
    edit6.Text:='0';  
    edit3.Text:='0';  
    edit3.enabled:=false;  
    edit2.Enabled:=false;  
    button1.enabled:=false;  
    button2.enabled:=false;  
    button3.enabled:=false;  
    button5.Enabled:=false;  
    listbox1.Clear;  
    listbox2.Clear;  
    listbox3.clear;  
    dbgrid1.enabled:=false;  
    dbgrid2.enabled:=false;  
    edit5.Text:='15';  
    button4.Enabled:=false;  
    checkbox1.Enabled:=false;
```

```

checkbox2.enabled:=false;
edit4.text:='0';
dbedit1.Enabled:=false;
dbedit2.enabled:=false;
dbedit3.Enabled:=false;
edit9.Text:='0';
end;
procedure TForm9.CheckBox1Click(Sender: TObject);
var b: extended;
var c: extended;
begin
b:=strtofloat(edit4.Text)* (strtofloat(edit5.Text)*0.01);
c:=strtofloat(edit4.text)* (strtofloat(edit6.Text)*0.01);
edit7.Text:=floattostr( strtofloat(edit4.text)+ b - c);
edit9.Text:=floattostr(strtofload(edit4.Text)-strtload(edit8.text));

if (checkbox1.Checked=true) then begin
edit3.enabled:=false;
button1.enabled:=false;
listbox1.Enabled:=false;
listbox2.Enabled:=false;
listbox3.enabled:=false;
dbgrid2.Enabled:=false;
dbgrid2.Refresh;
dbedit8.text:=edit7.text;
dbedit9.text:=floattostr(b);
dbedit10.Text:=edit9.Text;
button4.enabled:=true;
edit5.enabled:=false;
edit6.enabled:=false;
button5.enabled:=true;
end;
begin
if (checkbox1.Checked=false) then begin

```



```

edit3.enabled:=true;
dbgrid2.Enabled:=true;
dbgrid2.Refresh;
dbedit11.Text:=query1.fields[0].AsString;
dbedit12.Text:=query1.Fields[1].AsString;
dbedit13.Text:=query1.Fields[2].asString;
listbox1.Enabled:=true;
listbox2.Enabled:=true;
listbox3.enabled:=true;
button4.enabled:=false;
edit5.enabled:=true;
edit6.Enabled:=true;
end;
end;
end;

```

```

procedure TForm9.Button3Click(Sender: TObject);
begin
query2.close;
query2.SQL.clear;
query2.sql.add('select * from stocklist where productname
like'+#39+(edit2.text)+'%'+#39);
query2.Open;
end;

```

```

procedure TForm9.Button6Click(Sender: TObject);
begin
form1.enabled:=true;
form1.setfocus;
form1.series1.clear;
form1.Query1.close;
form1.query1.open;
form9.query3.Cancel;
form9.edit3.Text:='0';

```

```

form9.edit8.text:='0';
form9.edit9.text:='0';
form9.query2.cancelupdates;
form9.query1.Cancel;
form9.Query4.Close;
form9.query4.open;
form9.edit4.Text:='0';
form9.edit3.enabled:=false;
form9.edit2.enabled:=false;
form9.button3.Enabled:=false;
form9.dbgrid2.enabled:=false;
form9.button1.enabled:=false;
form9.edit7.Text:='0';
form9.listbox1.Clear;
form9.listbox2.Clear;
form9.listbox3.Clear;
form9.checkbox1.checked:=false;
form9.checkbox1.Enabled:=false;
form9.checkbox2.enabled:=false;
form9.checkbox2.checked:=false;
form9.button5.Enabled:=false;
form9.dbedit1.Enabled:=false;
form9.dbedit2.Enabled:=false;
form9.DBEdit3.enabled:=false;
form9.dbgrid1.Enabled:=false;
form9.dbgrid1.Refresh;
form9.close;
end;

procedure TForm9.Button5Click(Sender: TObject);
begin
button7.enabled:=true;
query3.Cancel;
edit3.Text:='0';

```

```

edit8.text:='0';
edit9.text:='0';
query2.cancelupdates;
query1.Cancel;
edit4.Text:='0';
edit2.enabled:=false;
button3.Enabled:=false;
button1.enabled:=false;
edit7.Text:='0';
listbox1.Clear;
listbox2.Clear;
listbox3.Clear;
checkbox1.checked:=false;
checkbox1.Enabled:=false;
checkbox2.enabled:=false;
checkbox2.checked:=false;
button5.Enabled:=false;
edit3.enabled:=false;
dbedit1.enabled:=false;
dbedit2.enabled:=false;
dbedit3.enabled:=false;
dbgrid2.enabled:=false;
dbgrid1.enabled:=false;
dbgrid2.refresh;
dbgrid1.refresh;
end;

```

```

procedure TForm9.CheckBox2Click(Sender: TObject);
begin
if checkbox2.Checked=true then begin
dbedit7.text:=dbedit1.text;
edit1.enabled:=false;
dbgrid1.enabled:=false;
button3.enabled:=true;

```

```

edit3.Enabled:=true;
button1.enabled:=false;
edit2.enabled:=true;
dbgrid2.enabled:=true;
dbgrid2.Refresh;
dbgrid1.Refresh;
dbedit1.Enabled:=false;
dbedit2.Enabled:=false;
dbedit3.enabled:=false;
dbedit7.text:=dbedit1.text;
end else begin
dbgrid2.enabled:=false;
dbgrid2.refresh;
dbgrid1.Enabled:=true;
dbgrid1.Refresh;
dbedit1.Enabled:=true;
dbedit2.Enabled:=true;
dbedit3.enabled:=true;
dbedit7.text:=dbedit1.text;
end;
end;

```

```

procedure TForm9.Button7Click(Sender: TObject);
var i: integer;
var b: integer;
begin
button8.Enabled:=false;
button7.enabled:=false;
listbox1.clear;
listbox2.clear;
listbox3.Clear;
query3.Insert;
edit3.Text:='0';
edit4.Text:='0';

```

```

edit6.Text:='0';
edit7.Text:='0';
edit1.enabled:=true;
button2.enabled:=true;
dbgrid1.enabled:=true;
dbedit7.Text:='';
dbedit8.Text:='';
dbedit9.Text:='';
dbedit10.text:='';
dbedit11.text:='';
dbedit12.text:='';
dbedit13.text:='';
dbedit14.Text:='';
dbedit15.Text:='';
dbedit16.text:='';
dbedit17.Text:='';
dbedit18.text:='';
dbedit19.text:='';
dbedit20.text:='';
checkbox1.checked:=false;
checkbox2.Checked:=false;
checkbox2.enabled:=true;
dbgrid1.Refresh;
dbedit1.enabled:=true;
dbedit2.enabled:=true;
dbedit3.enabled:=true;
updown1.Enabled:=false;
b:=query4.RecordCount;
form9.query4.first;
for i:=1 to b do begin
form9.query4.Delete;
form9.query4.next;
end;
form9.query4.Close;

```



```
form9.query4.Open;
form9.query4.Refresh;
end;
```

```
procedure TForm9.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var b: integer;
begin
b:=strtoint(dbedit4.Text);
updown1.Min:=0;
updown1.Max:=b;
if strtoint(edit3.Text)>0 then begin
button1.Enabled:=true;
end;
end;
```

```
procedure TForm9.DBGrid2CellClick(Column: TColumn);
begin
dbedit11.text:=dbedit1.text;
dbedit12.text:=dbedit2.Text;
dbedit13.text:=dbedit3.Text;
if strtoint(dbedit4.Text)=0 then begin
updown1.Enabled:=false;
button1.Enabled:=false;
end else begin
updown1.Enabled:=true;
button1.enabled:=true;
edit3.Text:='1';
end;
end;
```

```
procedure TForm9.Button8Click(Sender: TObject);
begin
rvproject1.Execute;
end;
```

```
end.
```