



**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**Dormitory Automation System by Using Delphi**

**Graduation Project**

**COM 400**

**Student: Ünal TAŞDİZEN (20041238)**

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

**Nicosia – 2008**



## ACKNOWLEDGEMENT

*First of all, I would like give my special thanks to my supervisor Assist. Prof. Dr. İmanov ELBRUS. He helped and supported me to complete my project by any means of necessary. In addition to this he never doubted about me, he always believed in me that I will fulfill and succeed on my project. I am glad to that I did not disappoint him.*

*Furthermore, I want to give my special thanks and best regards to my parents. They were always kind and patient to me. I wouldn't be here without their endless support.*

*Finally, I want to give my special thanks to my friends whose are Cenk ÜNDAŞ, Ebru GÜLTEK and Nurten ÖZTÜRK. They are supported and helped me to complete my project. I am very happy to have such friends.*

ÜNAL TAŞDİZEN

## ABSTRACT

The aim of this program is to develop automation software which deals with large scale dormitory residential areas. As a programming language Delphi was used and as a database MySQL Database Server was used.

I chose Delphi as programming language because Delphi speeds Win32 development by combining Delphi's proven visual Rapid Application Development approach for accelerated Win32 development with support for Windows Vista, AJAX, and streamlined database connectivity. In the real world, developers need to be able to develop applications that run on multiple platforms, not just the latest and greatest. Most new machines come with Windows Vista, while existing machines will continue running Windows 2000 or XP. Developers must support this mixed-use environment, because they can't count on their organization or customers upgrading en masse. They have to meet the demand for critical new technologies and trends in marketplace by including support for these technologies in their applications, but they want to retain the flexibility of developing on the platform that is most productive for them.

The MySQL database has become the world's most popular open source database because of its consistent fast performance, high reliability and ease of use. It's used on every continent, even Antarctica and by individual Web developers as well as many of the world's largest and fastest-growing organizations such as Yahoo!, Alcatel-Lucent, Google, Nokia, YouTube, and Zappos.com. Not only is MySQL the world's most popular open source database, it's also become the database of choice for a new generation of applications built on the Linux, Apache, MySQL, PHP / Perl / Python, Delphi. MySQL runs on more than 20 platforms including Linux, Windows, OS/X, HP-UX, AIX, Netware, giving you the kind of flexibility that puts you in control.

# Table of Contents

ACKNOWLEDGEMENT.....	I
ABSTRACT.....	II
TABLE OF CONTENTS .....	III
INTRODUCTION .....	1

## CHAPTER ONE : BASIC CONCEPT OF DELPHI

1.1 Introduction to Delphi.....	2
1.2 What is Delphi? .....	2
1.2.1 Delphi Compilers.....	2
1.2.2 What kind of programming can you do with Delphi? .....	3
1.2.3 History of Delphi .....	4
1.2.4 Advantages & Disadvantages Delphi .....	6
1.3 Delphi 6 Editions .....	7
1.3.1 Delphi 6 Architect.....	7
1.3.2 Installation Delphi 6.....	8
1.4 A Tour of the Environment.....	10
1.4.1 Running Delphi for the First Time .....	10
1.4.2 The Delphi IDE.....	11
1.4.3 The Menus & Toolbar.....	12
1.4.4 The Component Palette.....	12
1.4.5 The Code Editor.....	13
1.4.6 The Object Inspector.....	14
1.4.7 The Object TreeView.....	15
1.4.8 Class Completion.....	16
1.4.9 Debugging applications .....	17
1.4.10 Exploring Databases .....	18
1.4.11 Templates and the Object Repository .....	19
1.5 Programming with Delphi .....	20
1.5.1 Starting a New Application.....	20
1.5.2 Setting Property Values .....	21
1.5.3 Adding objects to the form .....	22
1.5.4 Add a Table and a StatusBar to the Form.....	22
1.5.5 Connecting to a Database .....	24

## CHAPTER TWO : THE RAVE REPORTING

2.1 Project Tree.....	28
2.2 Design Tools .....	29
2.3 Reuse and Maintenance Tools .....	32
2.4 Standard Components .....	34
2.5 Drawing Components .....	35
2.6 Reporting Components .....	35
2.7 Barcode Components.....	39
2.8 Anchors.....	39
2.9 Code Based Reports.....	40



2.9.1	Simple Code Base Report.....	40
2.9.2	Tabular Code Based Report.....	41
2.9.3	Graphical Code Based Report.....	43
2.10	Visually Designed Reports .....	45
2.10.1	The Visual Designer .....	45
2.10.2	Interacting with the Project.....	48
2.11	Data Aware Reports.....	55
2.11.1	The Database Connection .....	55
2.11.2	The Driver Data View.....	55
2.11.3	Regions and Bands.....	58
2.11.4	Adding Fields.....	59
2.11.5	Adding the Report to Your Project.....	60

### CHAPTER THREE : USER MANUAL

3.1	Database Connection Screen .....	61
3.2	Main Menu.....	62
3.3	Buildings Menu.....	62
3.3.1	Building Organize Form .....	62
3.3.2	Floor Organize Form .....	64
3.3.3	Room Organize Form .....	66
3.4	Customers Menu .....	67
3.4.1	Customer Organize .....	67
3.5	Cafeteria Menu .....	70
3.5.1	Product Organize Form.....	70
3.5.2	Sales Form .....	72
3.6	Visitors Menu .....	73
3.6.1	Visitor Organize Form .....	73
3.7	About Menu .....	75
3.8	Exit Menu .....	75

<b>CONCLUSION</b> .....	76
-------------------------	----

<b>REFERENCES</b> .....	77
-------------------------	----

<b>APPENDIX</b> .....	78
-----------------------	----

# INTRODUCTION

Dormi v1.0 Dormitory Automation Software has a client – server architecture design. This means that there is no need to install any additional software or make configurations on client side. Just copy the Dormi v1.0 program anywhere in client's computer and execute it. Even server may not be placed in the same building or city. As long as the client has internet connection, it is possible to use Dormi v1.0 Dormitory Automation Software. As a database server Dormi v1.0 uses MySQL Server. There is no restriction about server version but it is recommended to use v4.1 or later. MySQL Database server is today's one of the most popular and powerful database server. More of it, it is free.

Dormi v1.0 is designed as simple as possible. You can't see fancy animations which drains CPU and reduces system performance or fully painted and rendered forms which allocates more memory and causes delay on execution. Also this kind of things distracts the user and it gets annoying on long term use. In addition to this Dormi v1.0 hasn't got any modal form restrictions. This means that you can work with more than one form at the same time and you can access your desktop any time you want. Main menu consumes very small space on desktop and user may open another program without closing the program.

Dormi v1.0 has related table structure. This means that information on different tables has relation with each other. In other words consider a customer is staying on building A and you updated building A's name as B, after update, if you check customer's information you will notice that building name will be shown as B.

# CHAPTER 1

## 1 BASIC CONCEPT OF DELPHI

### 1.1 Introduction to Delphi

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

### 1.2 What is Delphi?

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

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

For the purposes of this series I will be using Delphi 6. Delphi 6 provides all the tools you need to develop test and deploy Windows applications, including a large number of so-called reusable components.

Borland Delphi provides a cross platform solution when used with Borland Kylix - Borland's RAD tool for the Linux platform.

#### 1.2.1 Delphi Compilers

There are two types compiler for Delphi

- Turbo Delphi: Free industrial strength Delphi RAD (Rapid Application Development) environment and compiler for Windows. It comes with 200+ components and its own Visual Component Framework.

- Turbo Delphi for .NET: Free industrial strength Delphi application development environment and compiler for the Microsoft .NET platform.

### 1.2.2 What kind of programming can you do with Delphi?

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

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

Some projects to which Delphi is suited:

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

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



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

### **1.2.3 History of Delphi**

Delphi was one of the first of what came to be known as "RAD" tools, for Rapid Application Development, when released in 1995 for the 16-bit Windows 3.1. Delphi 2, released a year later, supported 32-bit Windows environments, and a C++ variant, C++ Builder, followed a few years after.

The chief architect behind Delphi, and its predecessor Turbo Pascal, was Anders Hejlsberg until he was headhunted in 1996 by Microsoft, where he worked on Visual J++ and subsequently became the chief designer of C Sharp programming language C# and a key participant in the creation of the Microsoft .NET Framework.

In 2001 a Linux version known as Kylix programming tool Kylix became available. However, due to low quality and subsequent lack of interest, Kylix was abandoned after version 3.

Support for Linux and Windows cross platform development (through Kylix and the CLX component library) was added in 2002 with the release of Delphi 6.

Delphi 8, released December 2003, was a .NET -only release that allowed developers to compile Delphi Object Pascal code into .NET Microsoft Intermediate Language MSIL. It was also significant in that it changed its IDE for the first time, from the multiple-floating-window-on-desktop style IDE to a look and feel similar to Microsoft's Visual Studio.NET.

Although Borland fulfilled one of the biggest requests from developers (.NET support), it was criticized both for making it available too late, when a lot of former Delphi developers had already moved to C#, and for focusing so much on backward compatibility that it was not very easy to write new code in Delphi. Delphi 8 also lacked significant high-level features of the c sharp, C# language, as well as many of the more appealing features of Microsoft's Visual Studio IDE. (There were also concerns about the future of Delphi Win32 development. Because Delphi 8 did not support Win32, Delphi 7.1 was included in the Delphi 8 package.)

The next version, Delphi 2005 (Delphi 9), included the Win32 and .NET development in a single IDE, reiterating Borland's commitment to Win32 developers. Delphi 2005 includes design-time manipulation of live data from a database. It also includes an improved IDE and added a "for ... in" statement (like C#'s for each) to the language. However, it was criticized by some for its bugs; both Delphi 8 and Delphi 2005 had stability problems when shipped, which were only partially resolved in service packs.

In late 2005, Delphi 2006 was released and federated development of C# and Delphi.NET, Delphi Win32 and C++ into a single IDE. It was much more stable than Delphi 8 or Delphi 2005 when shipped, and improved even more after the service packs and several hot fixes.

On February 8, 2006, Borland announced that it was looking for a buyer for its IDE and database line of products, which include Delphi, to concentrate on its Application Lifecycle Management ALM line. The news met with voluble optimism from the remaining Delphi users.

On September 6, 2006, The Developer Tools Group (the working name of the not yet spun off company) of Borland Software Corporation released single language versions of Borland Developer Studio, bringing back the popular "Turbo" moniker. The Turbo product set includes Turbo Delphi for Win32, Turbo Delphi for .NET, Turbo C++, and Turbo C#. Each version is available in two editions: "Explorer" a free downloadable version and "Professional" a relatively cheap (US\$399) version which opens access to

thousands of third-party components. Unlike earlier "Personal" editions of Delphi, new "Explorer" editions can be used for commercial development.

On November 14, 2006, Borland announced the cancellation of the sale of its Development tools; instead of that it would spin them off into an independent company named "CodeGear"

#### **1.2.4 Advantages & Disadvantages Delphi**

Delphi exhibits the following advantages:

- Rapid Application Development (RAD)
- Based on a well-designed language - high-level and strongly typed, with low-level escapes for experts
- A large community on Usenet and the World Wide Web (e.g. [news://newsgroups.borland.com](http://news://newsgroups.borland.com) and Borland's web access to Delphi)
- Can compile to a single executable, simplifying distribution and reducing DLL versioning issues
- Many VCL and third-party components (usually available with full source code) and tools (documentation, debug tools, etc.)
- Quick optimizing compiler and ability to use assembler code
- Multiple platform native code from the same source code
- High level of source compatibility between versions
- Cross Kylix - a third-party toolkit which allows you to compile native Kylix/Linux applications from inside the Windows Delphi IDE, hence easily enabling dual-platform development and deployment
- Cross FBC - a sister project to Cross Kylix, which enables you to cross-compile your Windows Delphi applications to multi-platform targets - supported by the Free Pascal compiler - without ever leaving the Delphi IDE
- Class helpers to bridge functionality available natively in the Delphi RTL, but not available in a new platform supported by Delphi
- The language's object orientation features only class- and interface-based Polymorphism in object-oriented programming polymorphism



Disadvantages:

- Limited cross-platform capability for Delphi itself. Compatibles provide more architecture/OS combinations
- Access to platform and third party libraries require header files to be translated to Pascal. This creates delays and introduces the possibilities of errors in translation.
- There are fewer published books on Delphi than on other popular programming languages such as C++ and C#
- A reluctance to break any code has lead to some convoluted language design choices, and orthogonally and predictability have suffered

### 1.3 Delphi 6 Editions

There are 3 editions in Delphi 6:

- Delphi Personal - makes learning to develop non-commercial Windows applications fast and fun. Delphi 6 Personal makes learning Windows development easy with drag-and-drop visual programming.
- Delphi Professional - adds the tools necessary to create applications with the latest Windows® ME/2000 look-and-feel. Dramatically enhance functionality with minimal code using the power and flexibility of SOAP and XML to easily integrate Web Services into client-side applications.
- Delphi Enterprise - includes additional tools, extensive options for Internet. Delphi 6 makes next-generation e-business development with Web Services a snap.

This Program will concentrate on the Enterprise edition.

#### 1.3.1 Delphi 6 Architect

Delphi 6 Architect is designed for professional enterprise developers who need to adapt quickly to changing business rules and manage sophisticated applications that synchronize with multiple database schemas. Delphi 2006 Architect includes an advanced ECO III framework that allows developers to rapidly deploy scalable external facing Web applications with executable state diagrams, object-relational mapping, and transparent persistence.



Delphi 6 Architect includes all of the capabilities of the Enterprise edition, and includes the complete ECO III framework, including new support for ECO State Machines powered by State Chart visual diagrams, and simultaneous persistence to multiple and mixed database servers.

- State Chart Diagrams
- Executable ECO State Machines
- Multi- and Mixed- ECO database support

### 1.3.2 Installation Delphi 6

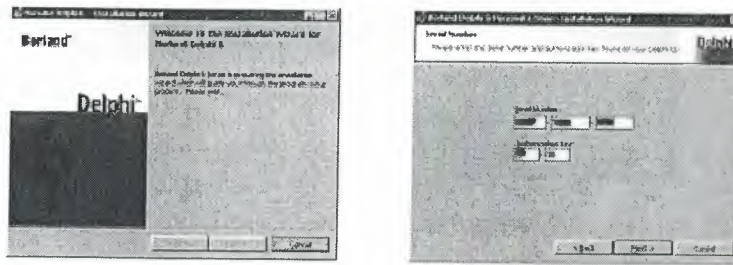
To install Delphi 6 Enterprise, run INSTALL.EXE (default location C:\Program Files\Borland Delphi) and follow the installation instructions.

We are prompted to select a product to install; you only have one choice "Delphi 6":



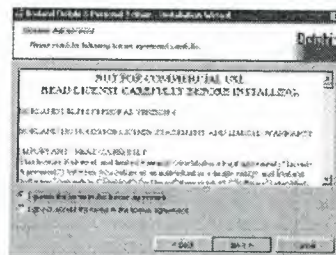
**Figure 1.1** The Select Page For Start Installation

While the setup runs, you'll need to enter your serial number and the authorization key (the two you got from inside a CdRom driver).



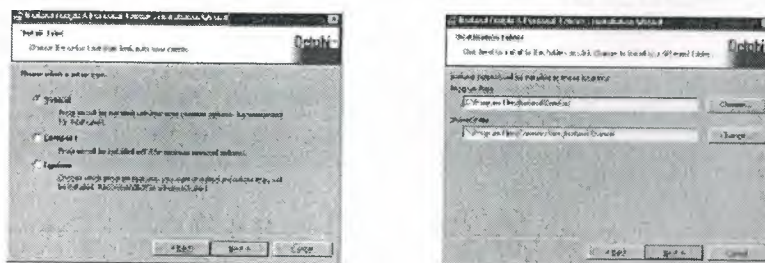
**Figure 1.2** Serial Number And Authorization Screen

Later, the License Agreement screen will popup:



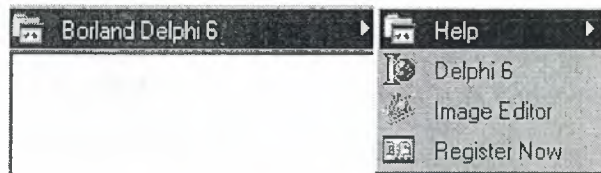
**Figure 1.3** License Agreement Screen

After that, you have to pick the Setup Type, choose Typical. This way Delphi 6 Enterprise will be installed with the most common options. The next screen prompts you to choose the Destination folder.



**Figure 1.4** SetUp Type and Destination Folder Screen

At the end of the installation process, the set-up program will create a sub menu in the Programs section of the Start menu, leading to the main Delphi 6 Enterprise program plus some additional tools.



**Figure 1.5 Start Menu**

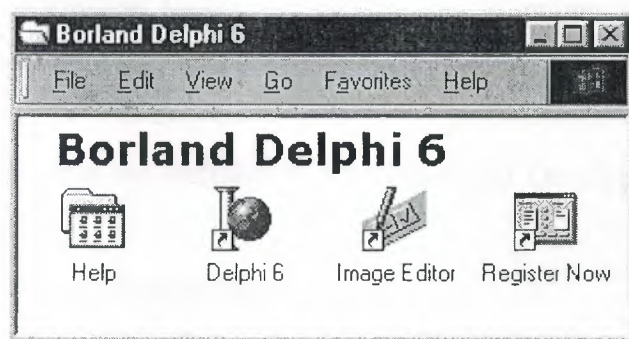
## 1.4 A Tour of the Environment

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

### 1.4.1 Running Delphi for the First Time

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

- Choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu
- Choose Run from the Windows Start menu and type Delphi32
- Double-click Delphi32.exe in the \$(DELPHI)\Bin folder. Where \$(DELPHI) is a folder where Delphi was installed. The default is C:\Program Files\Borland\Delphi6.
- Double-click the Delphi icon on the Desktop (if you've created a shortcut)



**Figure 1.6 Borland Delphi 6 Folder**



### 1.4.2 The Delphi IDE

As explained before, one of the ways to start Delphi is to choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu.

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

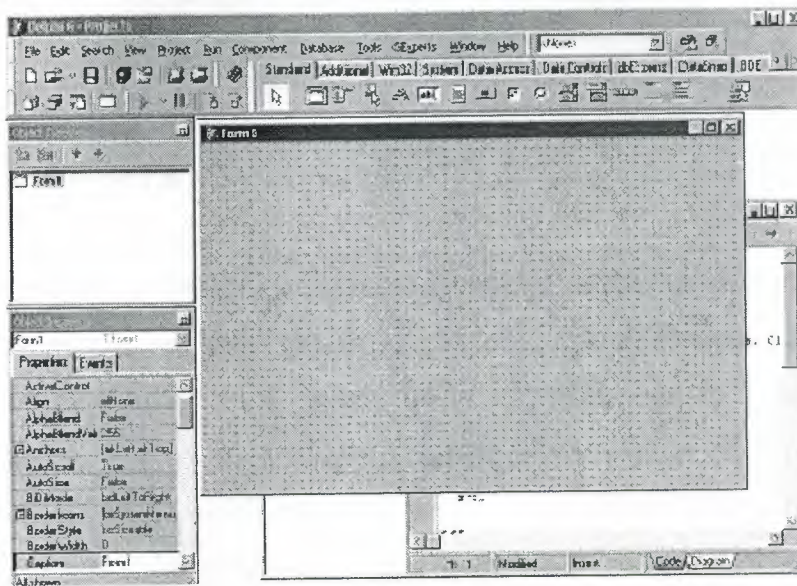


Figure 1.7 IDE

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

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

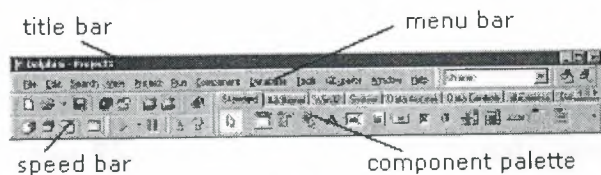
- A syntax sensitive program file editor
- A rapid optimizing compiler
- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools



- Image/Icon/Cursor creation / editing tools
- Version Control CASE tools

### 1.4.3 The Menus & Toolbar

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



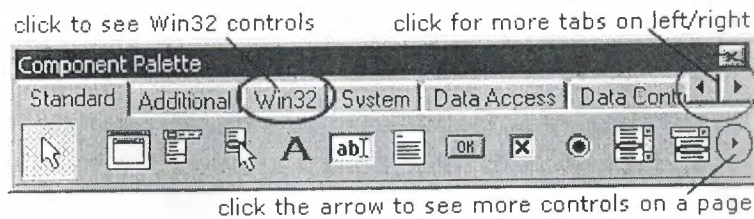
**Figure 1.8** Menu, Title, Speed Bar & Component Palette

The title bar of the main window contains the name of the current project (you'll see in some of the future chapters what exactly is a Delphi project). The menu bar includes a dozen drop-down menus - we'll explain many of the options in these menus later through this course. The toolbar provides a number of shortcuts to most frequently used operations and commands - such as running a project, or adding a new form to a project. To find out what particular button does, point your mouse "over" the button and wait for the tool tip. As you can see from the tool tip (for example, point to [Toggle Form/Unit]), many tool buttons have keyboard shortcuts ([F12]).

The menus and toolbars are freely customizable. I suggest you to leave the default arrangement while working through the chapters of this course.

### 1.4.4 The Component Palette

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



**Figure 1.9 Component Palates**

Depending on the version of Delphi (assumed Delphi 6 Personal through this course), you start with more than 85 components at your disposal - you can even add more components later (those that you create or from a third party component vendor).

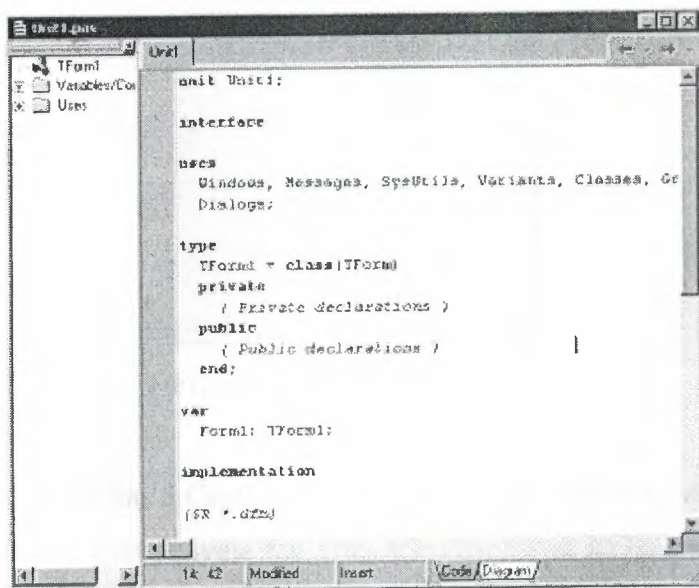
The components on the Component Palette are grouped according to the function they perform. Each page tab in the Component palette displays a group of icons representing the components you can use to design your application interface. For example, the Standard and Additional pages include controls such as an edit box, a button or a scroll box.

To see all components on a particular page (for example on the Win32 page) you simply click the tab name on the top of the palette. If a component palette lists more components that can be displayed on a page an arrow will appear on a far right side of the page allowing you to click it to scroll right. If a component palette has more tabs (pages) that can be displayed, more tabs can be displayed by clicking on the arrow buttons on the right-hand side.

#### **1.4.5 The Code Editor**

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

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



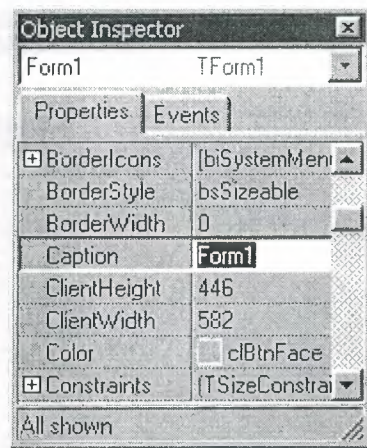
**Figure 1.10** Code Editor Window

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

### 1.4.6 The Object Inspector

Each component and each form has a set of properties – such as color, size, position, caption – that can be modified in the Delphi IDE or in your code, and a collection of events – such as a mouse click, keypress, or component activation – for which you can specify some additional behavior. The Object Inspector displays the properties and events (note the two tabs) for the selected component and allows you to change the property value or select the response to some event.





**Figure 1.11** Object Inspector

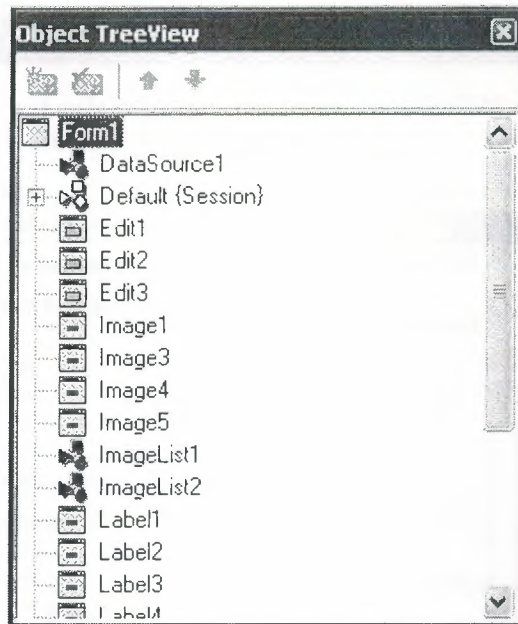
For example, each form has a Caption (the text that appears on its title bar). To change the captions of Form1 first activate the form by clicking on it. In the Object Inspector find the property Caption (in the left column), note that it has the 'Form1' value (in the right column). To change the captions of the form simply type the new text value, like 'My Form' (without the single quotes). When you press [Enter] the caption of the form will change to My Form.

Note that some properties can be changed more simply, the position of the form on the screen can be set by entering the value for the Left and Top properties - or the form can be simply dragged to the desired location.

#### 1.4.7 The Object TreeView

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





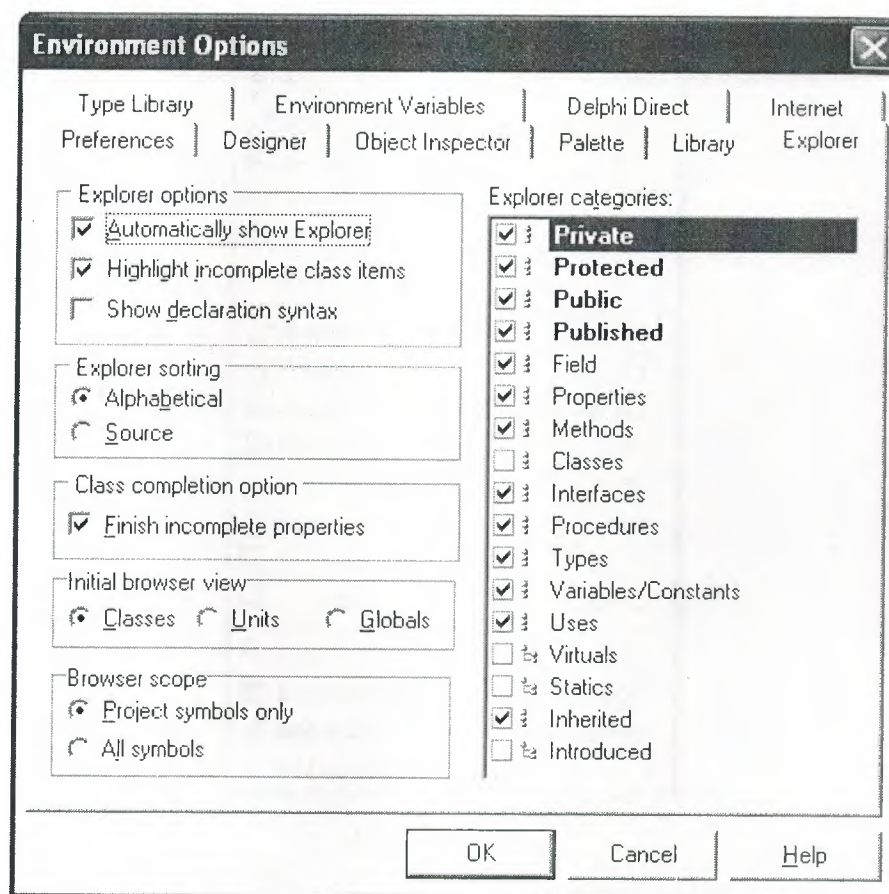
**Figure 1.12** Object Tree View

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

#### **1.4.8 Class Completion**

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press Ctrl+Shift+C, or right-click and select Complete Class at Cursor. Delphi automatically adds private read and write specifies to the declarations for any properties that require them, and then creates skeleton code for all the class's methods. You can also use Class Completion to fill in class declarations for methods you've already implemented.

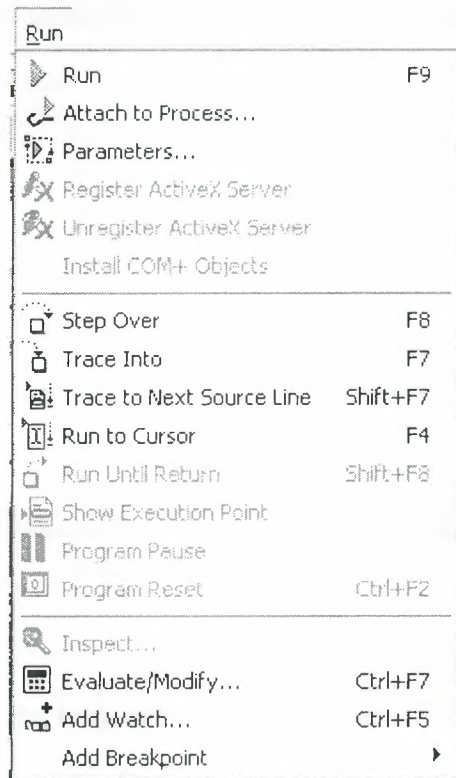
To configure Class Completion, choose Tools | Environment Options and click the Explorer tab.



**Fig.1.13** Class Completion

### 1.4.9 Debugging applications

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



**Figure1.14 Run**

To use the debugger, you must compile your program with debug information. Choose Project | Options, select the Compiler page, and check Debug Information. Then you can begin a debugging session by running the program from the IDE. To set debugger options, choose Tools | Debugger Options.

Many debugging windows are available, including Breakpoints, Call Stack, Watches, Local Variables, Threads, Modules, CPU, and Event Log. Display them by choosing View | Debug Windows. To learn how to combine debugging windows for more convenient use, see "Docking tool windows".

#### 1.4.10 Exploring Databases

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



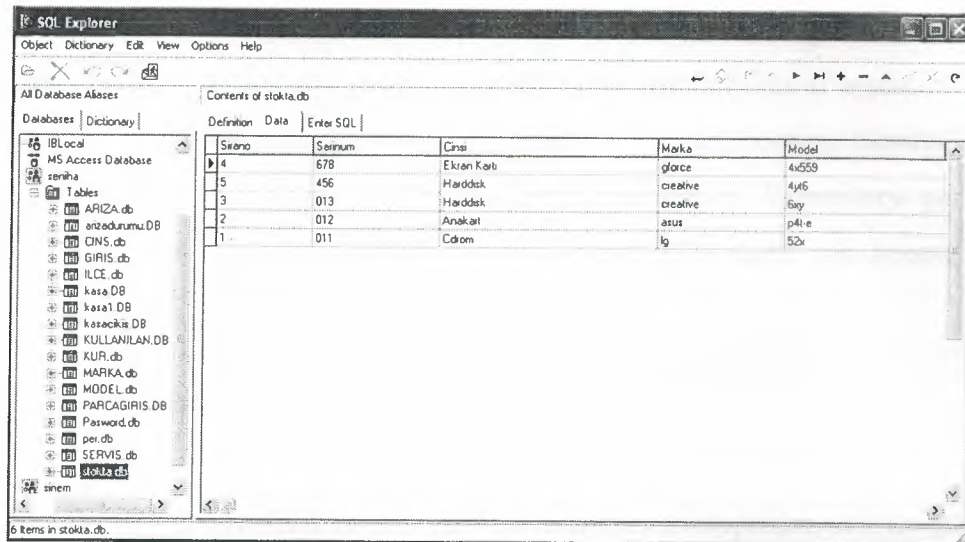


Figure 1.15 SQL Explorer

#### 1.4.11 Templates and the Object Repository

The Object Repository contains forms, dialog boxes, data modules, wizards, DLLs, sample applications, and other items that can simplify development. Choose File | New to display the New Items dialog when you begin a project. Check the Repository to see if it contains an object that resembles one you want to create.

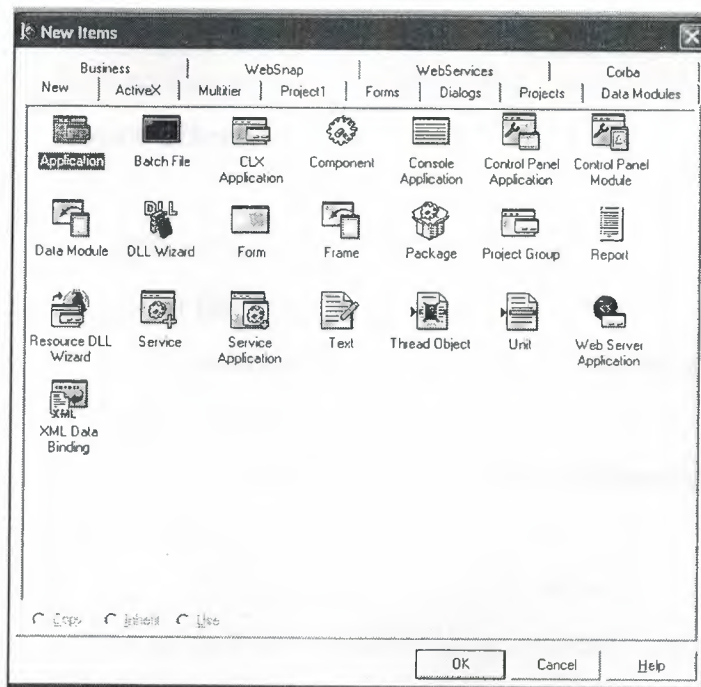


Figure 1.16 New Item

You can add your own objects to the Repository to facilitate reusing them and sharing them with other developers. Reusing objects lets you build families of applications with common user interfaces and functionality; building on an existing foundation also reduces development time and improves quality. The Object Repository provides a central location for tools that members of a development team can access over a network.

## **1.5 Programming with Delphi**

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

### **1.5.1 Starting a New Application**

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

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

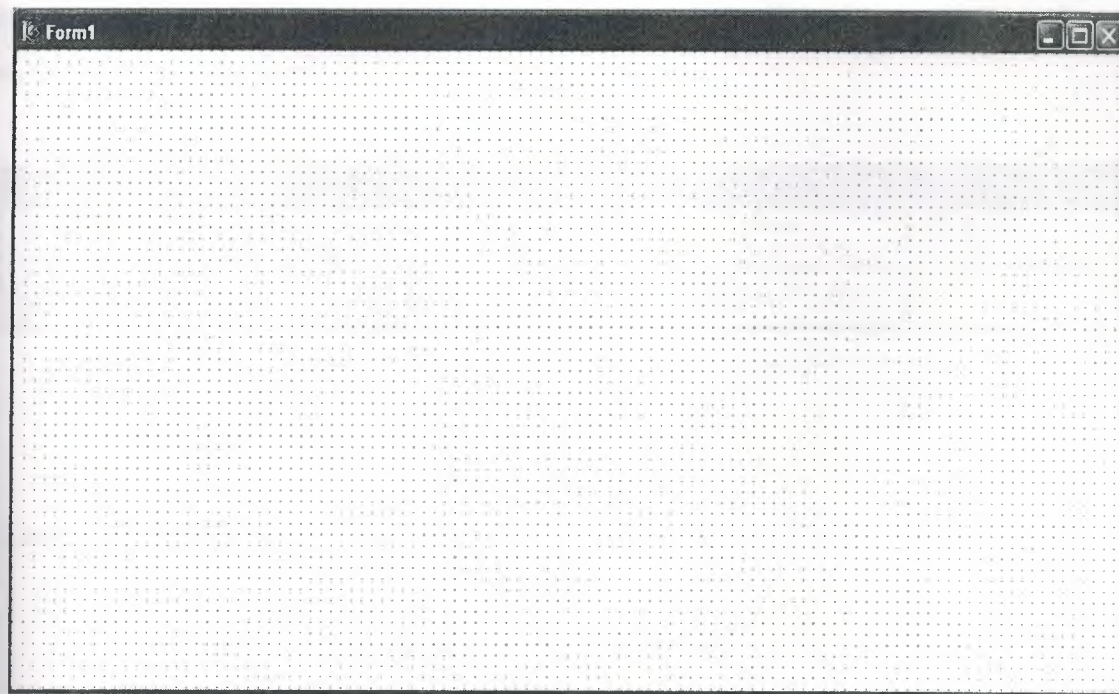
Each application is represented by a project. When you start Delphi, it opens a blank project by default. If another project is already open, choose File | New Application to create a new project. When you open a new project, Delphi automatically creates the following files.

- Project1.DPR : a source-code file associated with the project. This is called a project file.
  - Unit1.PAS : a source-code file associated with the main project form. This is called a unit file.
  - Unit1.DFM : a resource file that stores information about the main project form. This is called a form file.
3. Choose File | Save All to save your files to disk. When the Save dialog appears, navigate to your folder and save each file using its default name.

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

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

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



**Figure 1.17** Form Screen

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

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

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

### **1.5.2 Setting Property Values**

When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called design-time settings.

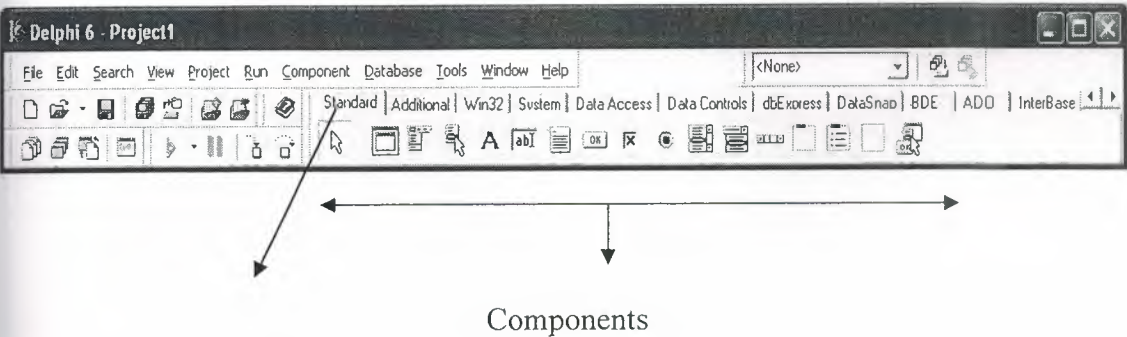
For Example; set the background color of Form1 to Aqua.



Find the form's Color property in the Object Inspector and click the drop-down list displayed to the right of the property. Choose clAqua from the list.

### 1.5.3 Adding objects to the form

The Component palette represents components by icons grouped onto tabbed pages. Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form.



Component palette tabs

Figure 1.18 Standard Bar

### 1.5.4 Add a Table and a StatusBar to the Form

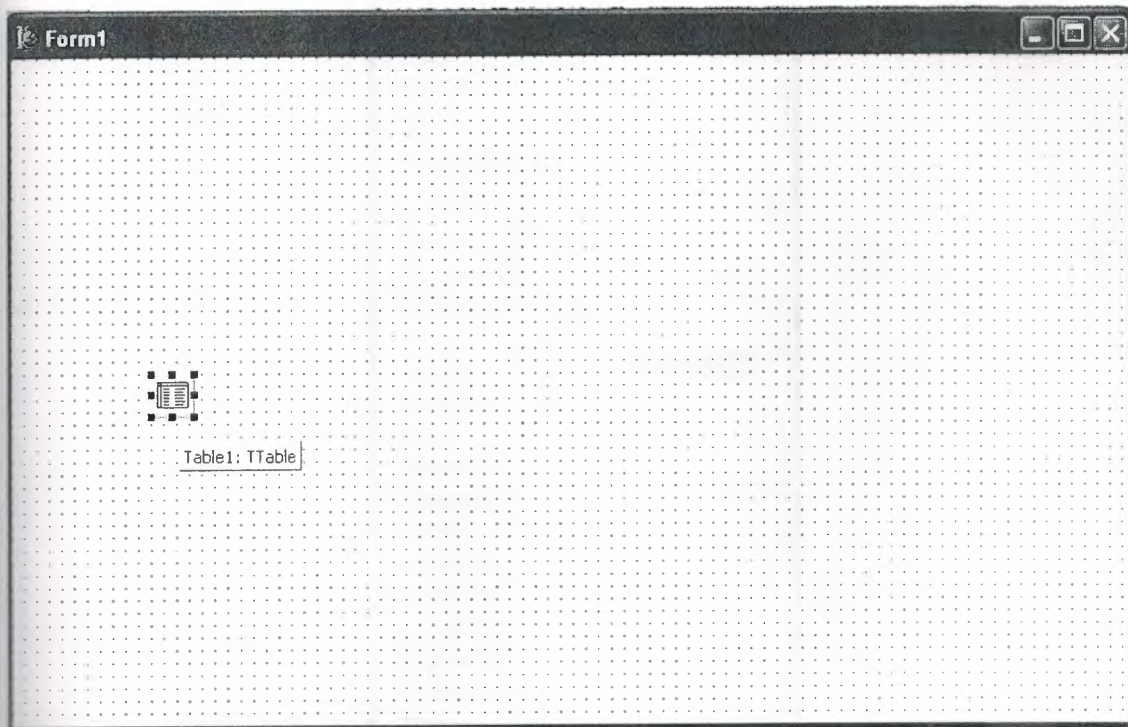
Drop a Table component onto the form. Click the BDE tab on the Component palette. To find the Table component, point at an icon on the palette for a moment; Delphi displays a Help hint showing the name of the component.



Figure 1.19 BDE Component palette

When you find the Table component, click it once to select it, and then click on the form to place the component. The Table component is non visual, so it doesn't matter

where you put it. Delphi names the object Table1 by default. (When you point to the component on the form, Delphi displays its name--Table1--and the type of object it is--Table.)

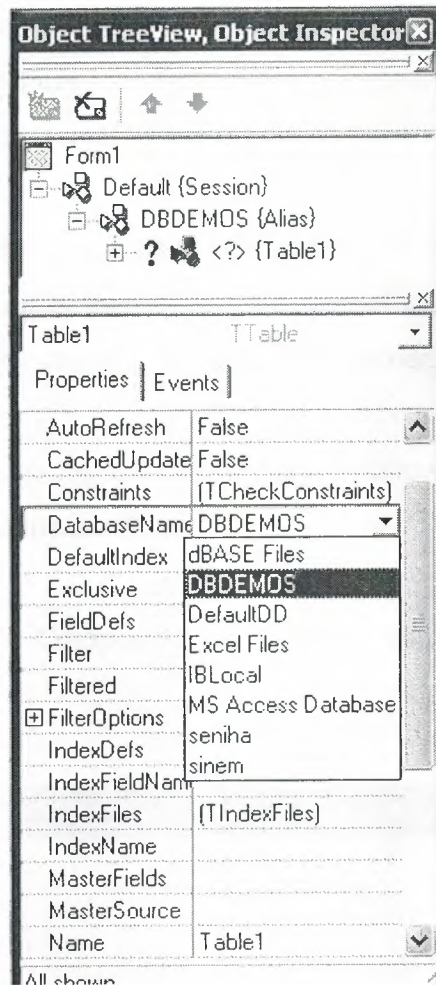


**Figure 1.20** Table in the Form

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

Set the DatabaseName property of Table1 to DBDEMOS. (DBDEMOS is an alias to the sample database that you're going to use.)

Select Table1 on the form, and then choose the DatabaseName property in the Object Inspector. Select DBDEMOS from the drop-down list.



**Figure 1.21** Select DatabaseName

Double-click the StatusBar component on the Win32 page of the Component palette. This adds a status bar to the bottom of the application.

Set the AutoHint property of the status bar to True. The easiest way to do this is to double-click on False next to AutoHint in the Object Inspector. (Setting AutoHint to True allows Help hints to appear in the status bar at runtime.)

### 1.5.5 Connecting to a Database

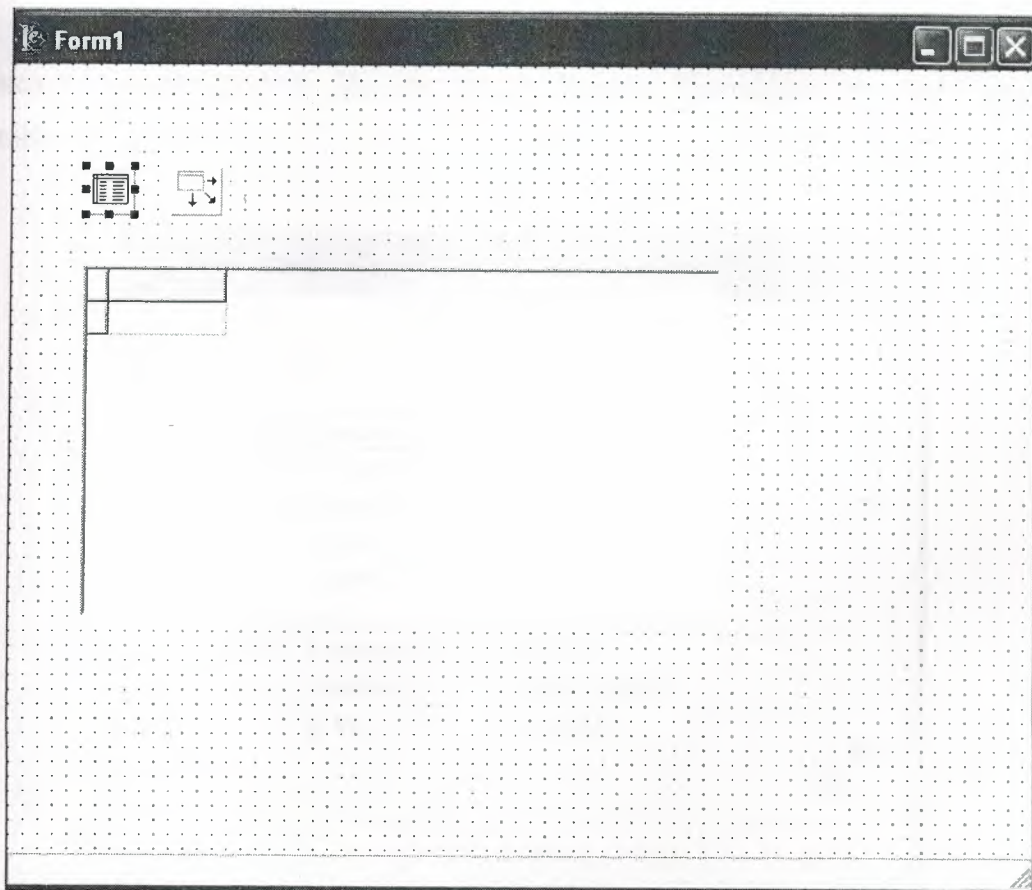
The next step is to add database controls and a DataSource to your form.



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

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

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



**Figure 1.22** DBGrid in the Form

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

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

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

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

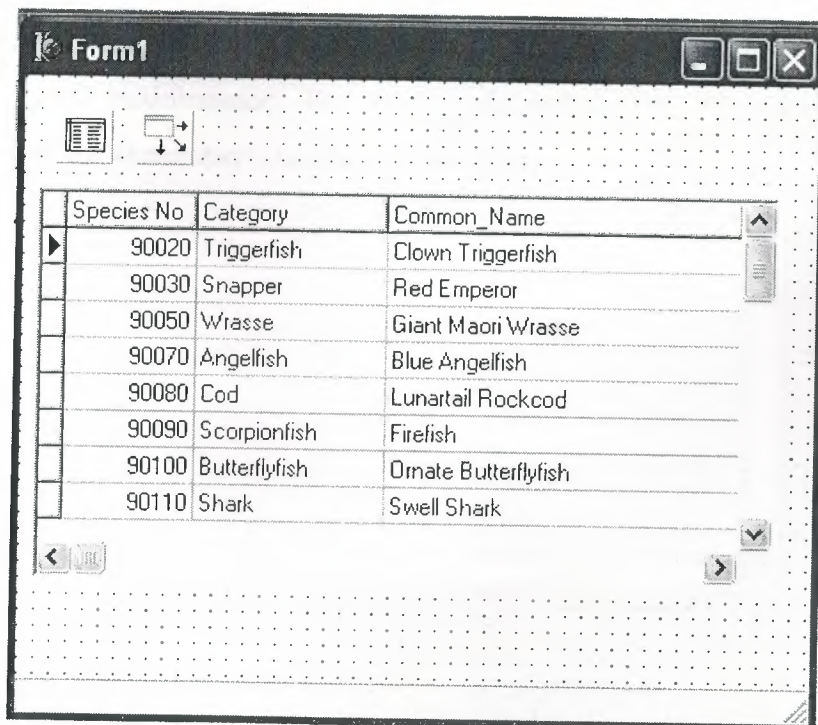


Figure 1.23 Show Table

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

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



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

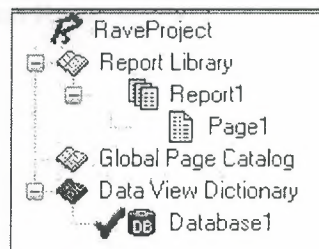


## CHAPTER 2

### 2 THE RAVE REPORTING

#### 2.1 Project Tree

The Project Tree provides an efficient way to visually manage all of the reports in your project. It quickly tells you the structure of your reporting project and the types of components contained on each page with icons that are the same as the component buttons. The Project Tree also visually shows parent-child relationships, the print order of component as well as the current selection (green check marks). You can select components by clicking on the component on the Page in the Visual Designer or on the Project Tree. Non-visual components appear only in the Project Tree in order not to clutter up your report design.



**Figure 2.1** Project Tree

There are three main sections in the Project Tree:

- The Report Library
- The Global Page Catalog
- The Data View Dictionary

Reports themselves can contain any number of page definitions. Global Pages are used to hold items that you want accessible to multiple reports. Data Views contain your field definitions and provide a link to the data in your application.

## 2.2 Design Tools

Rave is all about easy management. Besides making reporting easy and organized, Rave likes to keep itself organized and all according to what you want.

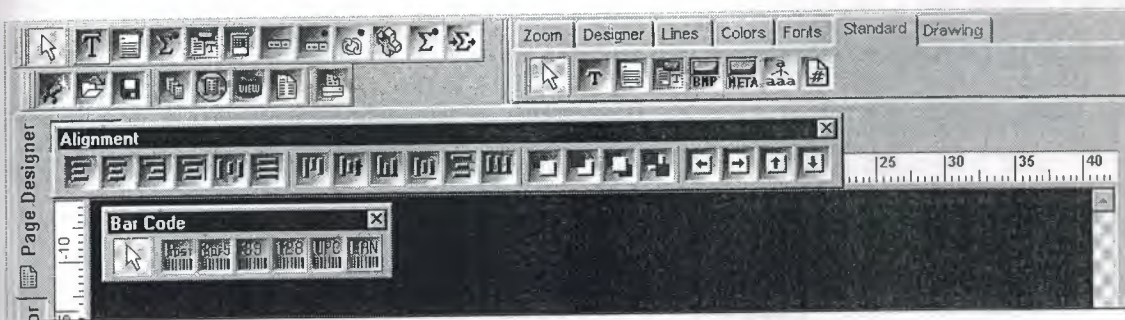


Figure 2.2 Toolbars

Since Rave is designed to be of ease to you there are three easy three ways for you to manage the many toolbars within Rave, which are:

- Tab-docking
- Normal docking
- Free-floating

Rave's many toolbars make it easy to design even the most complicated report. The toolbars include: Project, Designer, Zoom, Alignment, Color, Line, Font, Standard, Drawing, Report and Barcode component toolbars. Since it is possible to create and install new components, you may have other component toolbar buttons in your designer.



Figure 2.3 Project Toolbar

The Project toolbar provides quick access to project level functions such as New Project, Project Open, Project Save, New Report, New Global Page, New Data View, New Report Page or Execute Report.



**Figure 2.4** Designer Toolbar

The Designer toolbar allows you to change the characteristics of the Page in the Visual Designer. Characteristics such as whether the grid is being shown, snap to grid, draw grid on top, show band headers, show rulers, and show the waste area of the page. The last button brings up Rave's extensive Preferences dialog, which is described later.



**Figure 2.5** Zoom Toolbar

When you are working on a report with a complex design, you will find it much easier if you become familiar with the Zoom toolbar, which gives you quick access to Rave's extensive zooming capabilities. Select the zoom percent from a drop down list, type it in or use the Zoom Tool, Zoom In, Zoom Out, Zoom Selected, Zoom Page Width or Zoom Whole Page buttons.



**Figure 2.6** Alignment Toolbar

To help keep your report looking professional, Rave's Alignment toolbar provides access to a whole host of options to micro-manage the components on your page. The Left/Top, Center, Right/Bottom, Center In Parent, Space Equally, Equate Widths/Heights options offer the traditional alignment options. The Move Forward, Move Behind, Bring to Front and Send to Back order movement buttons allow you to change the print order of components and are visually backed up by the listing of the components in the Project Tree. Lastly, the buttons Tap Left, Tap Right, Tap Up and



Tap Down allow you to micro-adjust the position of components to the exact position you need.



Figure 2.7 Colors Toolbar

The Color toolbar allows you to quickly select the primary and secondary colors of your components. There are 8 color spots that you can use to store any custom colors that you will be reusing throughout the project. If the colors available aren't enough, you can double click on the custom color palettes and create a different color using Rave's Color Editor (shown at right). With the Color Editor, you can select from a wider variety of colors or create your own combination of Red, Green and Blue and even select a percent saturation for the current color.

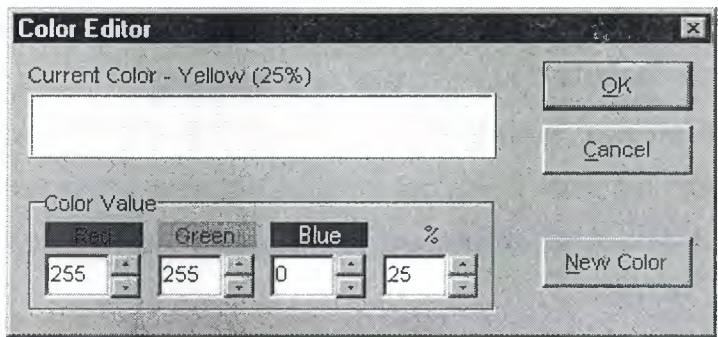


Figure 2.8 Colors Editor

The Line toolbar is a useful tool for changing the line/border thickness and style for components such as Line and Circle. Sizes are listed in points instead of pixels so that your lines will always be the same thickness on your reports no matter the resolution of the printer that you are using.



Figure 2.9 Line Toolbar

The Font toolbar provides quick access to a text component's font and alignment properties. It can also be useful for quickly viewing the font options for the currently selected text component(s).



**Figure 2.10** Fonts Toolbar

### 2.3 Reuse and Maintenance Tools

Reports often take a large part of the development time for an application. Many times, there are many similarities between the design of separate reports.

This is where Rave's Mirroring technology comes in. When a component is set to mirror another, it assumes the appearance and properties of the component it is mirroring. The two components can be on the same page, across pages within the same report or on a global page. This is the primary purpose of a global page. You can almost think of it like an Object Repository, a central location for you to store reporting items that you want accessible to more than one report. If the component is a container control like TRaveSection (similar to Delphi's TPanel), all child components are mirrored as well. When the original component changes, all mirroring components will also change. While the mirrored component cannot change its properties, you can add additional components if it is a container control.

Here are just a few examples of where Mirroring would be useful:

Your customer wants a standard page header and footer on every page of their 50 reports. Now imagine you have all the reports done and your customer wants to change the layout of the headers and footers.

**The Old Way** - You would need to open up all 50 report definitions and change them one at a time.



The Rave Way - You would mirror the standard header and footer on each report you create and then any changes would only have to be done in one location. Also, if the standard header included a large bitmap, your reporting project would only contain a single copy rather than the many copies that a traditional report designer would require. You have to replicate a pre-printed form. The problem is there are 6 different variations of this form with only minor differences between each.

The Old Way - Assuming a traditional report designer could even handle this type of report, you would create the first form, cut and paste it into the second, make the minor modifications, then repeat for the other 4 forms, ending up with 6 reports that would be hard to maintain and take up a lot more memory.

The Rave Way - You would first create the common items of the form on a separate page, then mirror those on each form and add the unique parts for each as needed. If anything ever needed to be changed in the common section of the form, you would only need to change it in one place and since you're sharing most of the form's content, the report definitions take up much less room.

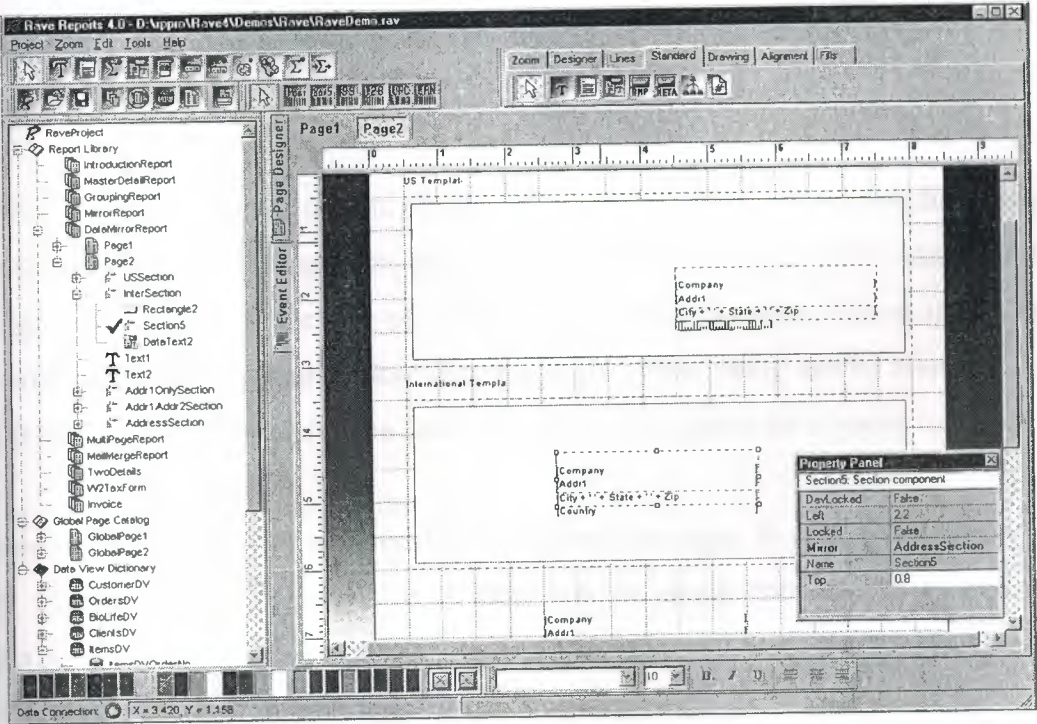


Figure 2.11 Mirror Report Example



Every text component has a FontMirror property which you can assign to a FontMaster component. This will allow you to change the fonts of many text controls from a single location. Imagine having Header, Body and Footer FontMaster components on a global page and changing the appearance of all of your reports with just a few mouse clicks.

Another important aspect of maintaining any large project is documentation. The Project and every Report, Page, Data View and Data Field component has a multi-line Description Property that can be used to comment the intended usage or other information. This can be useful if you are coming back to a project that you last worked on 6 months ago or especially if another programmer or your end user will be modifying reports that you created.

## 2.4 Standard Components



**Figure 2.12** Standard Tool Bar

**Text** - This component is used to display fixed text on your report for items such as column headers or report titles.

**Memo** - This component is used to display fixed text in a word wrapped fashion on your report. Using the MailMergeItems property and the Mail Merge Editor shown below, you can create a mail merge type of report where Rave will replace tokens in the memo text with a replacement string. Note that this replacement string can be edited with the Edit button, which will display the Data Text Editor for quite a bit of extra functionality.

**Section** - This component is a terrific component manager. It acts as a container for other components, in other words it help you to group components together. By properly using section components and mirroring, you can create reusable and maintainable reports in no time flat.

Bitmap - This component is used to display a bitmap (\*.bmp). Through the FileLink property you can reference a file on the hard disk.

MetaFile - This component is used to display a metafile (\*.wmf). Through the FileLink property you can reference a file on the hard disk.

FontMaster - This component is used to control the font characteristics of any text control through their FontMirror properties. See Reuse and Maintenance for more information.

## 2.5 Drawing Components

Line - Draws a diagonal line. (This may not seem like a unique feature but did you know that most Delphi reporting tools cannot create a diagonal line visually.)



**Figure 2.13** Drawing Tool Bar

HLine - Draws a horizontal line.

VLine - Draws a vertical line.

Rectangle - Draws a rectangle.

Square - Draws a square.

Ellipse - Draws an ellipse.

Circle - Draws a circle.

## 2.6 Reporting Components

Region - This component acts as a container for Band and DataBand components. To create a composite or sub-report, simply drop more than one region on a page and add the appropriate bands to each.

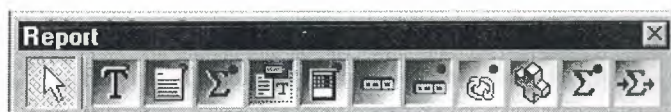


Figure 2.14 Report Tool Bar

Band - This component is primarily used to create header and footer bands in a banded style report. A Band component can only be created within a region and its purpose is controlled through the Band Style Editor shown below. The Band Style Editor displays a virtual layout of all of your bands for the given print locations of each band or data band. Note that you can create as many Bands as you like and a Band may print in multiple locations if the report design requires it. So for example, if you want a solid horizontal line to appear above and below a detail body, you could create a single band and set it to print on both the Body Header and Body Footer. You can also control the Print Occurrence for a Band, having it continue on a new page or column or any combination of occurrence settings. You can set a Band to group on specific fields and can create as many different types of group headers or footers as your report requires. Basically, with Rave's Band and DataBand components, you'll be able to create just about any banded style layout that you can imagine.

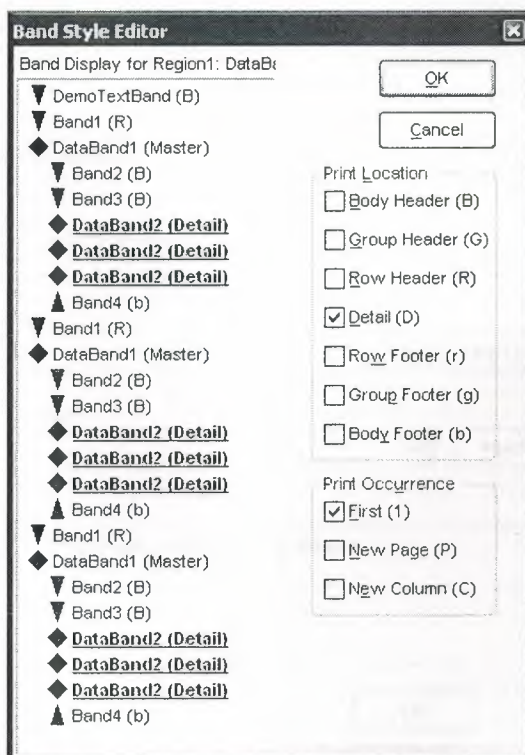


Figure 2.15 Band Style Editor



**DataBand** - The DataBand component is fairly similar to a band component except that it is tied to a particular DataView and iterates across the rows in the DataView. You can link DataBands together for Master-Detail to unlimited levels or multiple details on the same level. Some advanced features that are supported by a DataBand include KeepBodyTogether, KeepRowTogether, StartNewPage, MaxRows and Orphan/Widow control.

**DataText** - The DataText component is the primary means to output fields from your database. You can quickly select a specific DataView and DataField with Property Panel or use the Data Text Editor shown below to create any combination of string constants, data fields, report variables or project parameters. The & concatenation operator is the same as the + operator, except that it also inserts a space. Report Variables are items such as total pages or current date and time in a variety of formats. Project Parameters are custom variables that you create and initialize from your Delphi application. Project Parameters can be used for items such as user defined report titles, printing the current user name or other custom information.

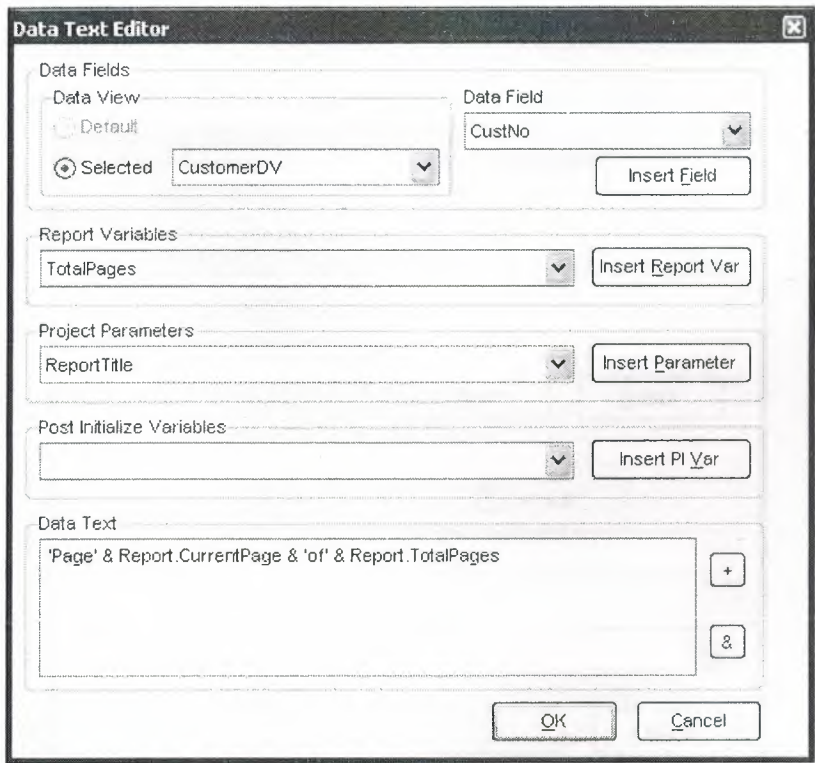


Figure 2.16 Data Text Editor

DataMemo - This component is very similar to the Memo component except that it retrieves data from a DataField. DataMemo component's print text data out in a word wrapped fashion and the DataField can be any text type, not just memo fields. It also has RTF and mail merge support.

CalcText - This component is used to perform simple operations such as Sum, Average, Count, Min and Max on a data field. You can set the value as a running total and place it in any type of band or anywhere on the page) you need it.

DataMirrorSection - The data mirror section component is similar to Rave's section component (found in the Standard Toolbar) with one major difference, it will dynamically mirror another section depending upon the value of a DataField. You configure the data mirror section using the Data Mirror Editor (shown below). This component is very useful for printing out data that has different formats depending upon the type of data. One example is an address field that could print a US format if the country field is "US" and an international format otherwise (using the Default option in the Data Mirror Editor). You could also print Boolean field values with your own custom bitmaps.



Figure 2.17 Data Mirror Editor

## 2.7 Barcode Components



Figure 2.18 Barcode Toolbar

PostNetBarCode - Prints a US PostNet bar code.

I2of5BarCode - Prints Interleaved 2 of 5 barcodes.

Code39BarCode - Prints standard and extended Code 39 barcodes.

Code128BarCode - Prints A, B and C Code 128 barcodes.

UPCBarCode - Prints UPC-12 barcodes.

EANBarCode - Prints EAN-13 barcodes.

## 2.8 Anchors

Anchors are a powerful way to create a report that dynamically adjusts to changing sizes. This allows you to create reports that can print well whether the user selects landscape or portrait, 8.5" by 11" or A4. There are 6 different anchor values for both the horizontal and vertical dimensions to allow you to control each component in exactly the manner that it needs. The Anchor Editor (shown at right) even shows you a helpful bitmap of how each anchor setting works.

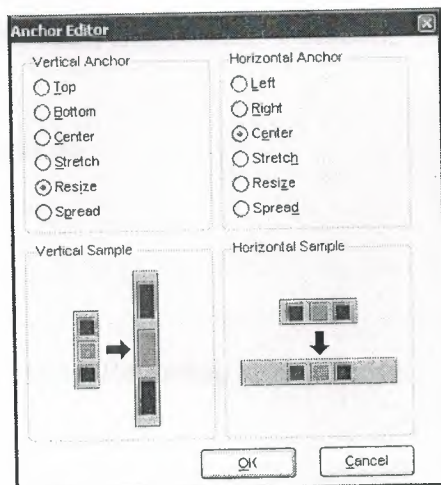


Figure 2.19 Anchor Editor



## 2.9 Code Based Reports

Lately Delphi has decided to include Rave Reports as the default reporting solution, replacing Quick Reports. Since they work in very different paradigms, many people were confused by the new environment. This is intended as an introduction for people who haven't worked with Rave yet, and would like to start.

Nowadays Delphi ships with Rave Reports 5.0.8. If you haven't already, download the update from the registered users page, since it fixes some important problems.

You can develop reports with Rave using two different ways: Code Based or with the Visual Designer.

With Code Based, you write reports using plain Delphi code. That provides a very flexible way displaying any kind of data, allowing any kind of complex layouts.

To write a code based report, just drop a TRvSystem component on the form and write the report on the OnPrint event handler. Sender is the report you are creating, and can be typecasted to TBaseReport. It contains all the methods you need to output information to that particular report.

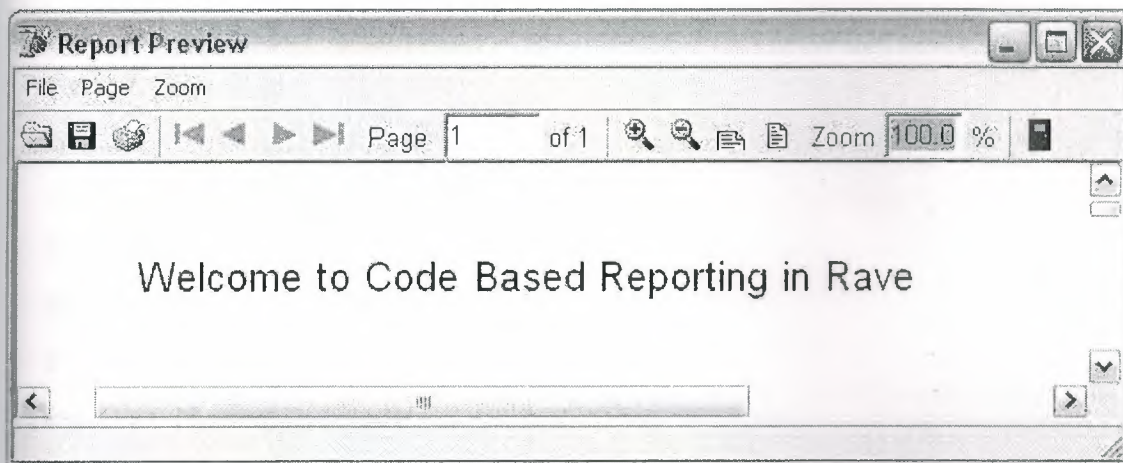
### 2.9.1 Simple Code Base Report

Here's a simple report using the code based mechanism:

```
procedure TFormMain.RvSystemPrint(Sender: TObject);
begin
  with Sender as TBaseReport do
  begin
    SetFont('Arial', 15);
    GotoXY(1,1);
    Print('Welcome to Code Based Reporting in Rave');
  end;
end;
```

To execute this report, call `RvSystem.Execute` method.

So, what does that simple code do? First, it calls `SetFont` to select the font and size of the text that will be printed from that point on. Then it positions the cursor on the coordinates (1,1). These coordinates are expressed using the units set in the `SystemPrinter.Units` property of the `RvSystem` object, and it defaults to Inches. You can set it to `unUser` and set a number relative to Inches in the `SystemPrinter.UnitsFactor` property. For example, if `UnitsFactor` was set to 0.5 then 1 unit would correspond to half an inch. Finally, the code calls the `Print` method to output the text. Here's the output:



**Figure 2.20** Report Preview

### 2.9.2 Tabular Code Based Report

Here's another example. It displays a list of the folders in the root of the current drive, along with a recursive count of number of files and folder, and total size of the files included in each folder.

```
procedure TFormMain.PrintTabularReport(Report: TBaseReport);
var
  FolderList : TStringList;
  i          : Integer;
  NumFiles   : Cardinal;
  NumFolders : Cardinal;
```

```

SizeFiles : Cardinal;
Root      : string;
begin
  with Report do
  begin
    SetFont('Arial', 15);
    NewLine;
    PrintCenter('List of Folders in the Drive Root', 4);
    NewLine;
    NewLine;
    ClearTabs;
    SetTab(0.2, pjLeft, 1.7, 0, 0, 0);
    SetTab(1.7, pjRight, 3.1, 0, 0, 0);
    SetTab(3.1, pjRight, 3.5, 0, 0, 0);
    SetTab(3.5, pjRight, 4.5, 0, 0, 0);
    SetFont('Arial', 10);
    Bold := True;
    PrintTab('Folder Name');
    PrintTab('Number of Files');
    PrintTab('Number of Folders');
    PrintTab('Size of Files');
    Bold := False;
    NewLine;
    FolderList := TStringList.Create;
    try
      Root := IncludeTrailingPathDelimiter(ExtractFileDrive(ParamStr(0)));
      EnumFolders(FolderList, Root);
      for i := 0 to FolderList.Count - 1 do
      begin
        PrintTab(FolderList[i]);
        GetFolderInfo(IncludeTrailingPathDelimiter(Root+FolderList[i]),
          NumFiles, NumFolders, SizeFiles);
        PrintTab(Format('%u',[NumFiles]));

```

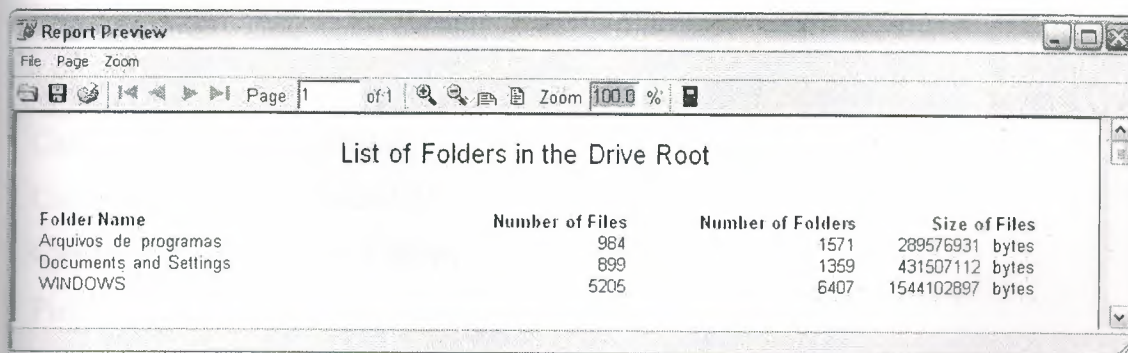


```

PrintTab(Format('%u',[NumFolders]));
PrintTab(Format('%u bytes',[SizeFiles]));
NewLine;
end;
finally
  FolderList.Free;
end;
end;
end;

```

Notice that a different approach has been taken: instead of specifying the coordinates of each text output, the printing was done using Lines and Columns as references. The line height depends on the size of the current font: each unit represents 1/72nds of an inch, so each line printed with a size 10 font will have, approximately, a height of 0.138 inches. Lines are advanced after calls to PrintLn or NewLine. Columns are defined using calls to the SetTabs method, and the PrintTab method will print the text in the current column and advance to the next one. Here's the output:



The screenshot shows a 'Report Preview' window with a menu bar (File, Page, Zoom) and a toolbar. The main content area displays a table titled 'List of Folders in the Drive Root'. The table has four columns: 'Folder Name', 'Number of Files', 'Number of Folders', and 'Size of Files'. The data rows are: 'Arquivos de programas' (964 files, 1571 folders, 289576931 bytes), 'Documents and Settings' (899 files, 1359 folders, 431507112 bytes), and 'WINDOWS' (5206 files, 6407 folders, 1544102897 bytes).

Folder Name	Number of Files	Number of Folders	Size of Files
Arquivos de programas	964	1571	289576931 bytes
Documents and Settings	899	1359	431507112 bytes
WINDOWS	5206	6407	1544102897 bytes

**Figure 2.21** Report Preview

### 2.9.3 Graphical Code Based Report

You can include shapes and images in your code based report, along with the text. The following example demonstrates that:

```

procedure TFormMain.PrintGraphicsReport(Report: TBaseReport);
var
  Bitmap : TBitmap;

```

```

begin
  with Report do
    begin
      Canvas.Brush.Color := clGray;
      Rectangle(0.3, 0.3, 4.7, 3.3);
      SetFont('Arial', 15);
      FontColor := clRed;
      PrintXY(0.5,0.5, 'Just look at all the graphics!');
      Bitmap := TBitmap.Create;
      try
        Bitmap.LoadFromFile('delphi.bmp');
        PrintBitmap(3.5,0.3,1,1, Bitmap);
        PrintBitmap(1,2,3,3, Bitmap);
        Canvas.Pen.Color := clBlue;
        Canvas.Brush.Bitmap := Bitmap;
        Ellipse(5,0.3,6,3.3);
        Ellipse(2,1,4,1.9);
      finally
        Bitmap.Free;
      end;
    end;
    Canvas.Pen.Color := clBlack;
    Canvas.Brush.Style := bsSolid;
    Canvas.Brush.Color := clYellow;
    Pie(0.7,0.7,1.7,1.7,1,1,2);
    Canvas.Brush.Color := clGreen;
    Pie(0.7,0.7,1.7,1.7,1,2,1,1);
  end;
end;

```

In this example the methods Rectangle, Ellipse and Pie have been used draw shapes with different fills. Bitmaps were outputted using PrintBitmap and as the brush of the ellipses. Here's the output:

**Graphics Report Example**

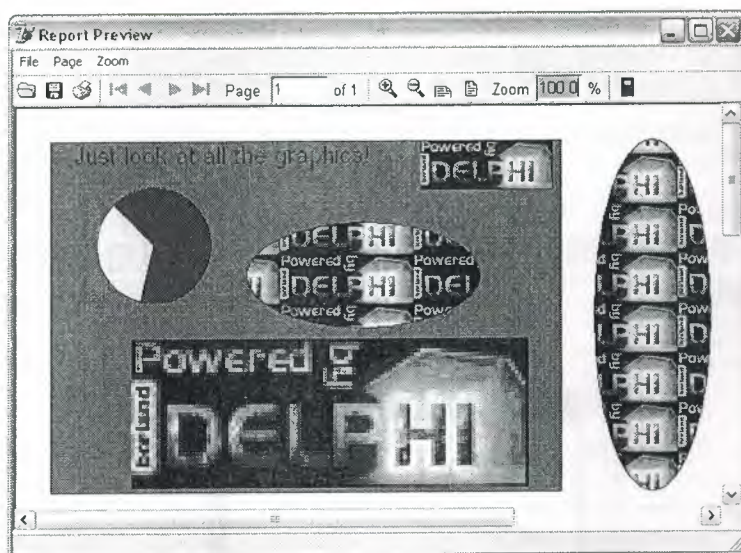


Figure 2.22 Report Preview

## 2.10 Visually Designed Reports

### 2.10.1 The Visual Designer

If you are used to work with Quick Reports, the default reporting engine included in the previous versions of Delphi, you created your reports using Delphi's own form designer, and they were save in the DFM, included as resources in your executable. Rave works a bit differently in this aspect: it has it's own report designer, and saves the report using it's own file format. This has some advantages, including the fact that your reports can be made "standalone", and be used or updated independently of your application, or even made available in a Intranet or in the Internet, using Nevrona's Rave Report Server. Of course, you can still have it saved in a form's DFM.

To get started with the Rave Visual Designer, drop a TRvProject in a form. This will be the link from your application to the reports you are developing. If you want, you can add a TRvSystem and link your RvProject to it, through it's Engine property. The RvSystem is the object responsible for the general configuration of the reports: the printer that is going to be used, the margins, the number of pages, and so on. To start a new project, double click the RvProject you added to the form, or select "Rave Visual Designer" from its context menu.



This is the interface that you will be working on:

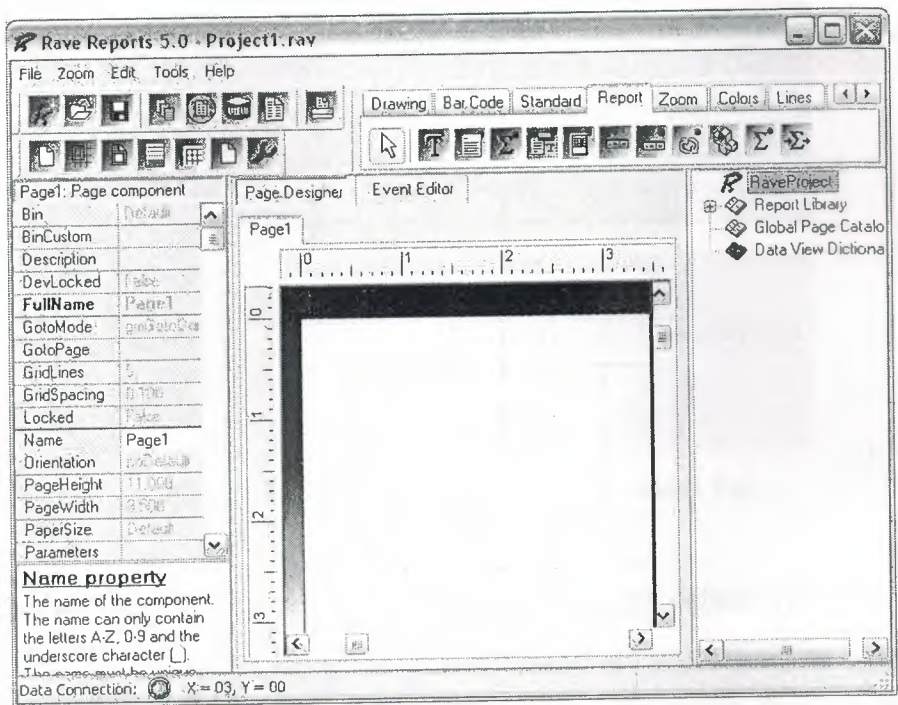


Figure 2.23 Rave Visual Designer

The interface is simple, and you might be familiar with some parts of it from Delphi's IDE. On the top there's the menu, the toolbar, and the component palette that contain the components that will be used in the reports. In the left there's the Object Inspector, which will be used to adjust the properties of the components of the report. In the middle there's the Page Designer or the Event Editor, and in the left there's the very useful Project Treeview. For a quick overview of the components in the palette, you can go to Nevrona's Visual Designer page.

A Rave Project File can have one or more reports. That way you can keep common items between them in a single location, called Global Pages. If you expand the Report Library node of the Project Treeview, you can see that right now you are working on Report1. Clicking on it, its properties will show on the Inspector. Let's change its name and call it SimpleReport. Next, go to the Standard tab on the Component Palette, and pick a Text component and add it to the page. Change its text property, and adjust its size and position. Here's how it looks like:

Anchor	(Top / Left)
Color	Black
DevLocked	False
DisplayOn	doParent
<b>Font</b>	<b>Arial 15</b>
FontJustify	piLeft
FontMirror	
Left	1,000
Locked	False
Mirror	
Name	Text1
Rotation	0
Tag	0
<b>Text</b>	<b>Welcome to Rave Reports Visual I</b>
Top	1,000
<b>Truncate</b>	<b>True</b>
Width	4,000

**Figure 2.24** Component Palette: Standard Tab

As you can see, the properties that were changed from the default values are shown in bold. In this case, I changed the Font, Text and Truncate properties. By default it does not highlight Name, Pos and Size changes. If you'd like to see them, right click the Inspector and uncheck "Exclude Name, Size and Pos changes" in the context menu.

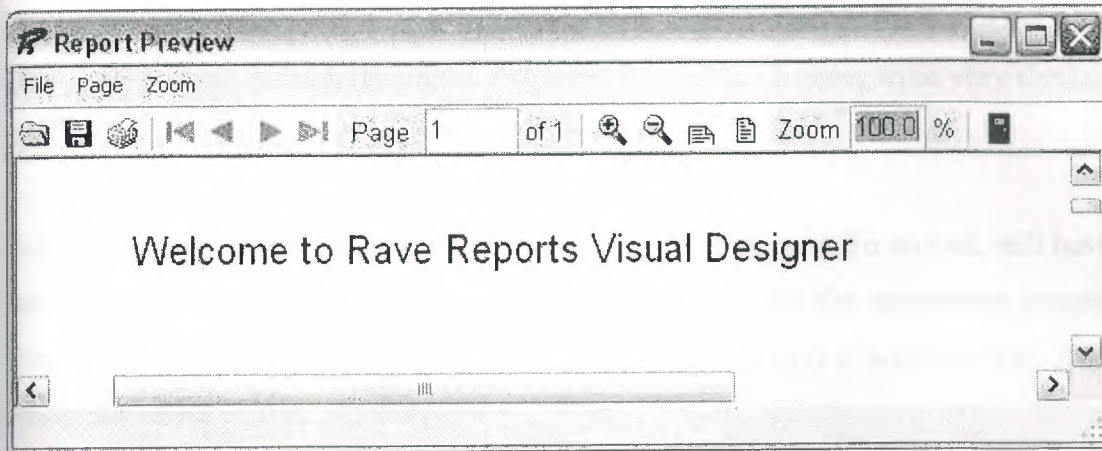
You might have also noticed that Rave does not have an auto size property. You can use the Truncate property to have that effect: if truncate is false, the design time size will have no effect.

You can see the result of this simple report right on the designer: Press F9 or use File/Execute Report to run it. Now let's do it in our application. Save your project and return to Delphi. Change to ProjectFile property of RvProject to point to the file you just saved. To run the report, add a call to the Execute method of the RvProject object in a button click, for example.

RvProject.Execute will only work for now because we only have one report in this project. If we had multiple reports, we'd have to call SelectReport to choose one before calling Execute, or calling ExecuteReport directly.



Here's the output:



**Figure 2.25** Report Preview

Tip: If you Close and Open your project before executing, you won't need to recompile your application or restart it to see the changes you just made in the designer.

### **2.10.2 Interacting with the Project**

If you worked with Quick Reports, you might be used to manipulating the objects in runtime, changing their Position, Text and Visibility. After all, they were just TObjects! While this is possible with Rave, and I'll cover it in a later article, it's a little harder than it was with QR. But don't worry, Rave provides a different answer to this kind of problems.

#### **Parameters**

If you can use parameters in your reports. They can be defined using the parameters property of either the Project, a Report or a Page. Parameters can be defined in either of these places, they are just in multiple places for easier access.

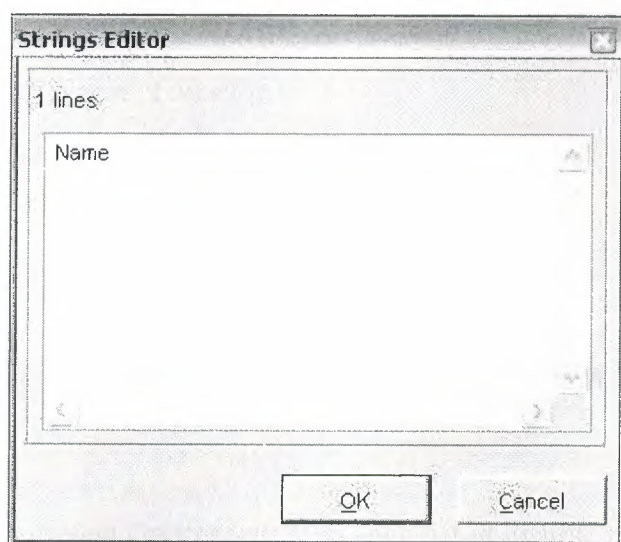
You can only select the Project and a Report through the Project Treeview. A page, however, can be selected using the Project Treeview or clicking on it's title above the page designer.

Among other uses, you can print parameters. So, for instance, if the title of your report can be user-defined, you could pass it from your application into the report as a parameter.



Let's add a new report to this project to see how parameters work. To do that, click the fourth button on the toolbar or choose File/New Report. Call it ParametrizedReport, changing its name through the object inspector. This report is going to be very similar to the first one, except the text is going to be user-defined.

Now we need to define the parameter that is going to be printed. To do that, still having the report as the selected object, open the property editor the the parameters property. There should be listed all parameters of this report, each on a separate line. Add a parameter called Name, like this:



**Figure 2.26** Strings Editor

Parameters can be printed using a DataText component, available in the Report tab of the component palette. Add a DataText to the page, and open the property editor of the DataField property. There you can choose which field is going to be printed, when working with DataAware reports. You can also choose Project Variables, Parameters and Post-Initialize Variables from there.

So choose the parameter added previously from the Parameters drop-down combo and press the Insert Parameter button. The data text expression is now Param.Name. Press

OK and try to execute the report, as before. Nothing is printed, since the parameter has not been set.

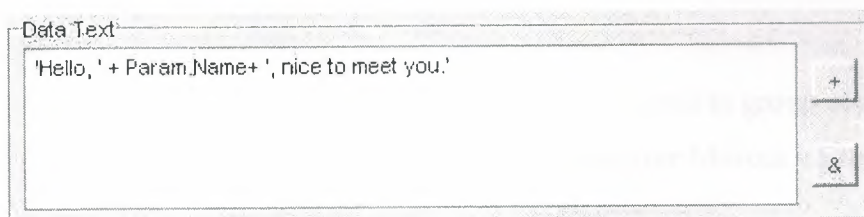
We need to set this parameter before printing. Don't forget to save your changes, and return to Delphi, adding a call to `SelectReport` before `Execute`, so we can see the right report. Before executing, though, we need to set the parameter we added. That is made using `RvProject`'s `SetParam` method. This is how my code looks like right now:

```
procedure TFormMain.btnExecuteClick(Sender: TObject);
begin
  RvProject.Open;
  RvProject.SelectReport('ParametrizedReport',False);
  RvProject.SetParam('Name','Leonel');
  RvProject.Execute;
  RvProject.Close;
end;
```

Now, when we execute the report, we are going to see the string we set as a parameter printed.

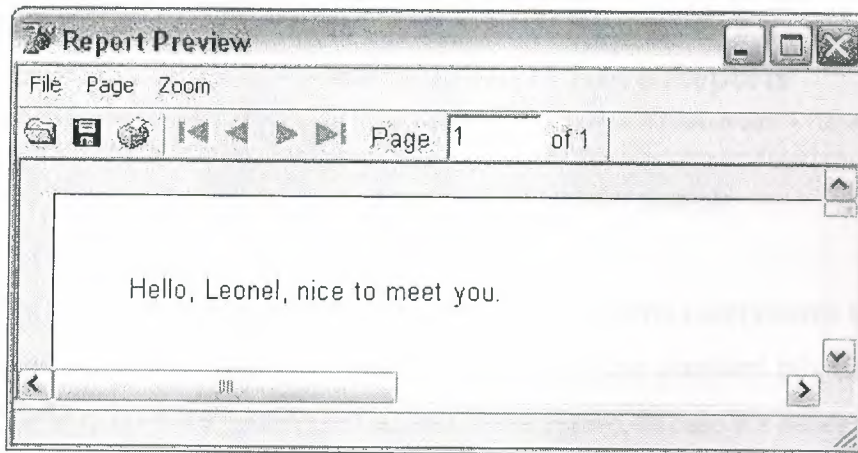
Tip: You can use `RvProject.GetReportList` to get a list of available projects, and add them to a `ComboBox`, or a `RadioGroup`, for example. That makes selecting the report easier.

But this is too simple. Let's change the expression that is going to be printed. Return to Rave Designer and open the property editor for the `DataText` we added. You can add any text you want, combining text, fields, parameters and variables. I changed it to this:



**Figure 2.27** Data Text Sample

Here's the result:



**Figure 2.28** Report Preview

### Post-Initialize Variables

Post-Initialize Variables, or simply PI Vars, are variables whose value is only known after the report has already been printed. It may sound strange, at first, but think about the number of pages of a report, for example. We can only know its value after the report is ready. Actually `TotalPages` is a report variable that acts like a PI var, and can easily be printed using `DataTexts` as we did with `Parameters`.

### Global Pages

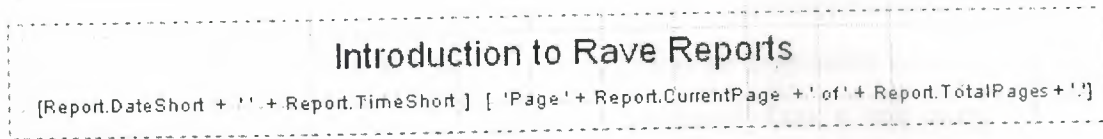
When you have parts of reports that are common to two or more reports, you can put these in a global page. Let's suppose we have a header with our company name, the date and time that report is being printed, the current page and the number of pages of that report. We want that header to be in every report. How can we do it?

First, add a global page to the project, using `File/New Global Page`, or the `Toolbar` shortcut. In that page, add a section component, available in the standard tab of the `component palette`.

Sections are logical groupings of components. They can be used to group component so they can be easily moved around the report or as containers for `Mirrors`, as we are doing right now.



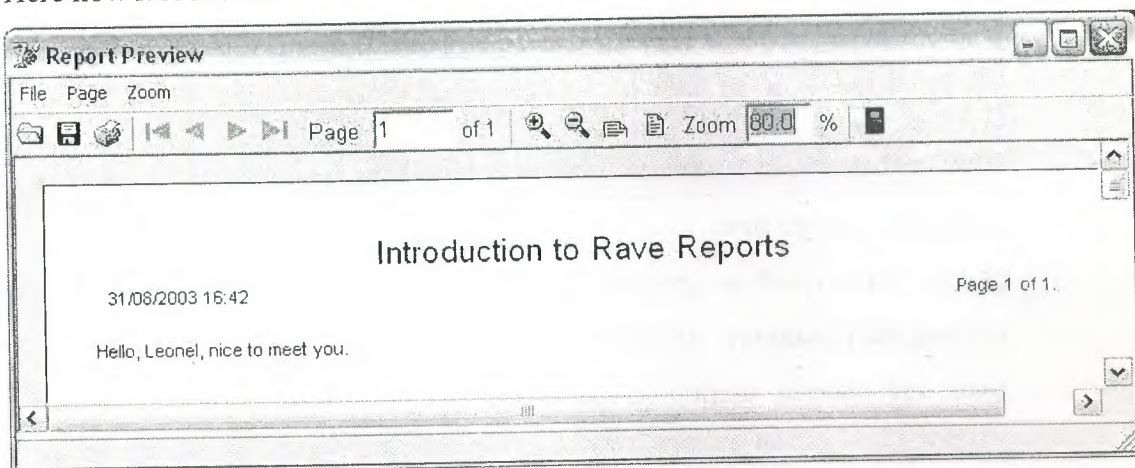
Inside that section we add what we want to be printed. In this case, a few DataTexts. My header looks like this:



**Figure 2.29** Header Sample

Hint: Instead of changing the font property of several components to the same font, link them to a FontMaster component, available in the standard tab, and set the font on it. That way is easier to change the font in the future, in case it's needed.

Now add another section to the Page1 of SimpleReport. Set its Mirror property to GlobalPage1.Section1. You will see a copy of the header you created in the global page. Do the same thing to ParametrizedReport. Now both reports share the same header. Here how it looks like:



**Figure 2.30** Report Preview

### Conditional Printing

Sometimes we need to print certain parts of a reporting depending of some conditions. Rave has a very powerful way of dealing with this. We can conditionally mirror sections depending on field values or parameters. Let's create a new Report, calling it a ConditionalReport.

Let's pretend that this new report is a trick one. The user can choose the header that is going to be printed, from two different kinds of headers. He can also choose for the report to be printed without a header. We are going to use a parameter to tell the report what kind of header is going to be printed, and a DataMirrorSection to select the proper header at runtime.

First, add a parameter to this new report called HeaderKind. Let's assume that it will have the values H0 (for no header), H1 (for the first header), H2 (for the second kind of header). Now add a new section to the global page (you can reach it through the Project Treeview), with the second kind of header layout. I created a header similar to the first one, changing the font title and adding a border around the values. It looks like this:

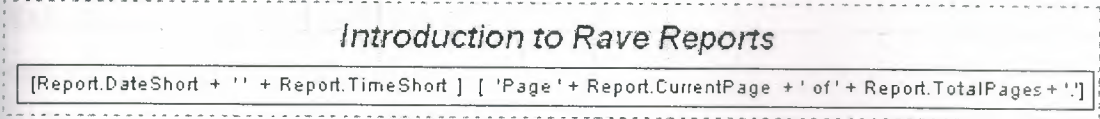


Figure 2.31 Header Sample

Now return to the Page1 of ConditionalReport, and add a DataMirrorSection, available at the Report tab of the component palette. Go to its DataField property editor, and set Param.HeaderKind as the expression. Now go to the DataMirrors property editor, and add two Data Mirrors: if the value is H1, it should point to the first header, H2, to the second. Since H0 does not match any mirrors, nothing will be printed. It should look like this:

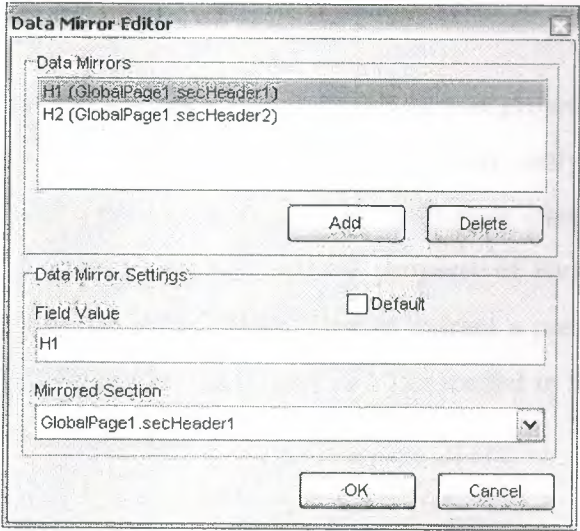


Figure 2.32 Data Mirror Editor

Notice that I gave more meaningful names to each of the sections earlier.

Hint: You can use the OnMirrorValue event of the DataMirrorSection to work on ranges of values.

Now return to Delphi and add the code to set the parameter according to the user's choice. I added a ComboBox with the options and my code looks like this:

```
procedure TFormMain.btnExecuteClick(Sender: TObject);
begin
  RvProject.Open;
  RvProject.SelectReport(cmbReports.Text,False);
  case cmbReports.ItemIndex of
    1: RvProject.SetParam('Name',edName.Text);
    2: RvProject.SetParam('HeaderKind',Format('H%d',[cmbHeaderKind.ItemIndex]));
  end;
  RvProject.Execute;
  RvProject.Close;
end;
```

Now the proper header will be printed according to the user's choice.

Embedding the Project in the Executable

When you deploy your application, you must include you project file. You can have it as a separated file, so you can update it in a easier way, only shipping a new one, without recompiling your application, or include it in your executable. It's easy to do that: open the property editor for the StoreRAV property of RvProject. There you can press Load to include the file in the DFM, Save to extract a previously saved file, and Clear to remove an embedded file. When there's a file loaded in this property, you don't need to ship the project file separately.



## 2.11 Data Aware Reports

### 2.11.1 The Database Connection

There are two ways to access data from inside a report: you can share the same connection established by your application, fetching records from Datasets that exists in your Forms or Datamodules, or you can configure a new connection on the report, allowing it to be independent of a particular application. For the first method you would use a Direct Data View, and a Driver Data View for the second. Data View is the analog of a DataSource/DataSet combination inside the report.

If you intend to deploy your application using Nevrona's Rave Report Server, you should use Driver Data Views.

### 2.11.2 The Driver Data View

Let's create a simple database report using a Driver Data View. Start the Rave Visual Designer, and start a new project. We need to define the database connection. To do this, choose File/New Database Object, or press the sixth button in the toolbar (the purple cube). The Data Connections window will appear:

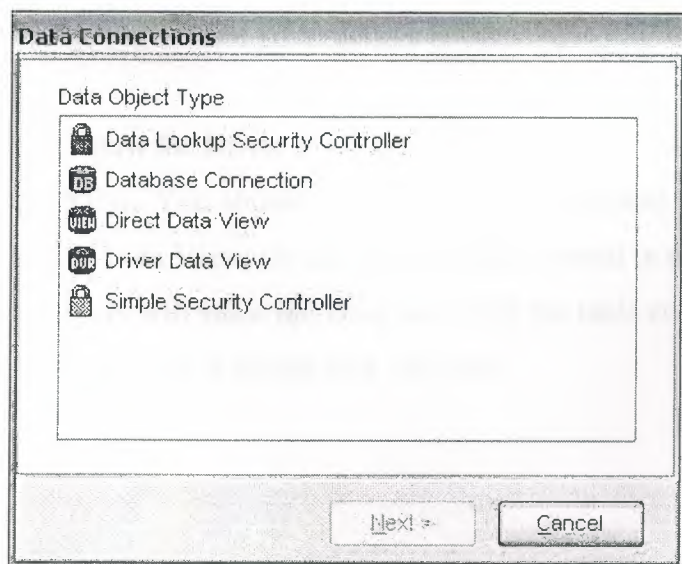
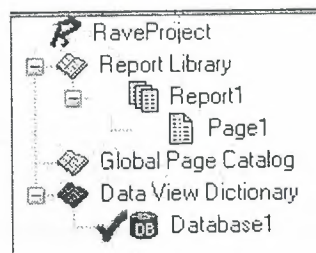


Figure 2.33 Data Connection Window

Choose Database Connection, and you will be asked which Data Link you are going to be using. There is a folder called DataLinks where Rave has been installed, containing some files with the .rvd extensions, responsible from the connection mechanism. By default, you can choose between BDE, DbExpress and ADO. I'll be using BDE for this example. Choose BDE, press Finish, and the Database Connection Parameters window will show up. Every Data Link has a different set of connection parameters available, similar to those available in the Delphi IDE. For now, just set Alias to DbDemos and press OK. Notice that a Database object has been added to the Project Treeview, under Data View Dictionary:



**Figure 2.34** Project Tree View

Notice that the settings you configured in the Database Connection Parameters, after the wizard, including username and password, if applicable, were saved in the AuthDesign property of the Database component. In the AuthRun property you can use different settings to be used at runtime, when your report has been deployed.

We are going to create now the Driver Data View. Click on New Data Object, and then choose Driver Data View. You should now choose the Database Connection that is going to be used by this Data View: choose the Database created in the previous step. A Query Advanced Designer will show up. Drag and Drop the table customer.db from the table list to the Layout window. It should look like this:

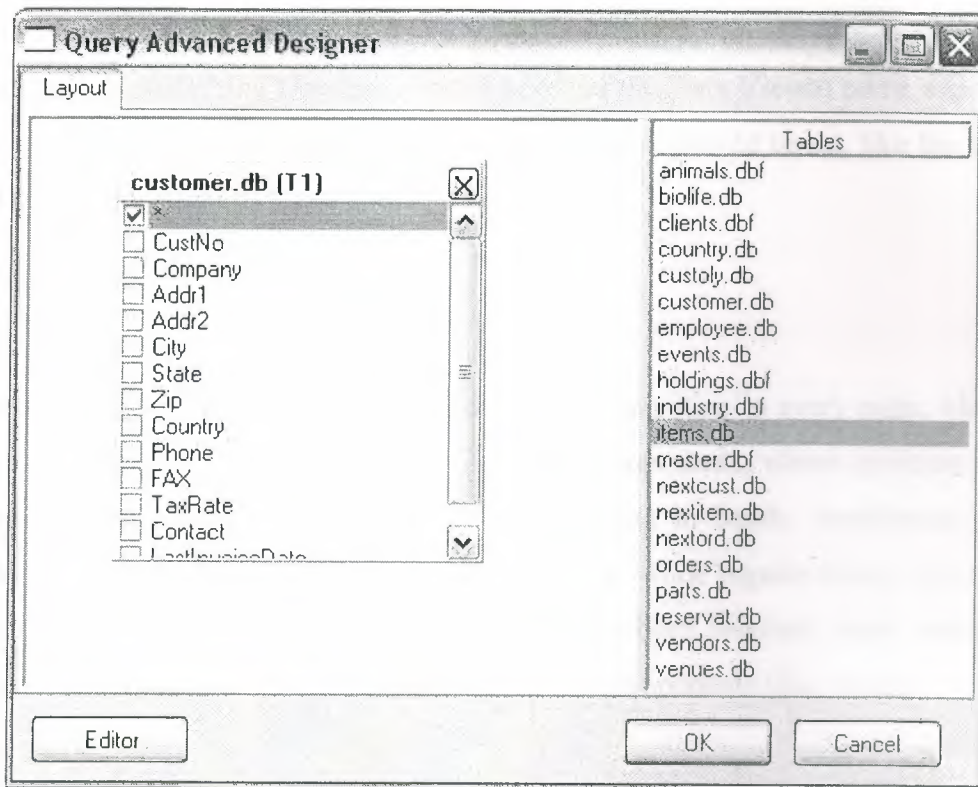


Figure 2.35 Query Advanced Designer Window

If you have more than one table, you should drag and drop fields that should be joined between tables. If you press the Editor button you can check the generated SQL, or type-in a more complex query. Let's keep the simple Customer Listing for now. Press OK and a DriverDataView will be added to the Project Treeview, below the Database components, having the selected fields as subitems:

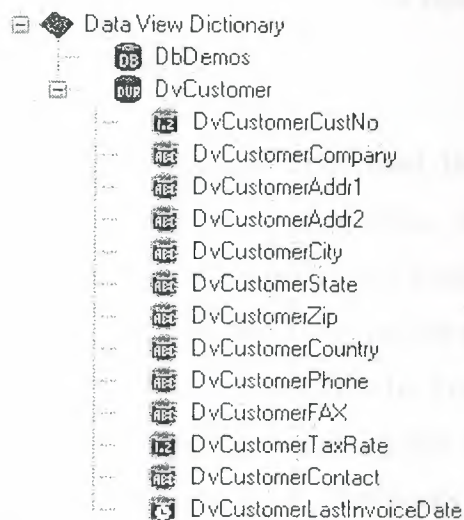


Figure 2.36 Project Tree View



Notice that I renamed the Database Connection and the Data View to more appropriate names. It's in the Treeview where properties of the fields should be set, like the Display Label (FullName property), and the DisplayFormat.

### 2.11.3 Regions and Bands

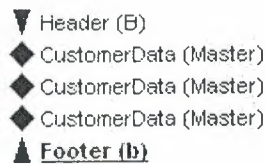
Report components that should be printed in a fixed position in every page, like fixed headers and footers can be put directly in page. Components, whose position will be dependent of previously printed items, should be put in bands. DataBands will be printed once for every record in the linked DataView, while regular Bands will only be printed once, regardless of how many records have been selected. Both can contain Data-Aware components (like DataText), or regular components (like Text).

Bands should be put inside Regions. Regions delimitate the width of the bands, and the maximum height that bands can use before starting a new page. One page can have many Regions, and one Region can contain many Bands.

Add a Region to the Page covering its whole area. Inside the region add a Band, to be used as the report header, a DataBand, to print the customer information, and another Band, the report footer.

If you wish to change the ordering of existing bands in a report, use the Move Forward and Move Behind buttons in the Alignment Toolbar.

Rename the bands to more meaningful names (I used Header, CustomerData and Footer). Set the DataView property of CustomerData to DvCustomer, and set CustomerData as the ControllerBand of the Header and Footer bands. You should also run the Band Style Editor, from the Object Inspector, and set the Print Location of those two bands to Body Header and Body Footer, respectively. You can have an idea on how the report is going to be printed observing the Band Display as you change the settings. It shows iterating bands repeated three times, and other bands only once:

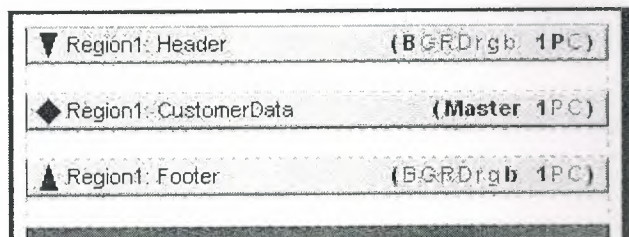


**Figure 2.37** Band Display

We also want the Header to be printed in other pages in case the listing spans more than one page: check the New Page option in the Print Occurrence groupbox, in that same dialog.

The Footer band will only print when DvCustomers has reached its end. If you want it printed in every page, regardless of that, just put the components directly on the page, below the region, and not in a Band.

In the editor, you can quickly identify the relationship between bands, their styles and their print occurrences:



**Figure 2.38** Editor Sample

#### 2.11.4 Adding Fields

It's not hard to add fields to a report. You can Ctrl+Drag the fields from the DataView, in the Project Treeview, to add DataText components to the report, and Alt+Drag them to add Text components containing the Fullname property. This allows you to quickly create the layout of the report. Now add some fields to CustomerData and their title to the Header. I added CustNo, Company, Phone, TaxRate and LastInvoiceDate.

Don't forget that you can use the tools on the Alignment Toolbar to align the components, even if they are in different bands.

I added a title to the Header band and a simple text to the Footer band, indicating that the listing has ended. Later on the series we are going to see how to use the CalcOp and CalcTotal components to be able to add totals, averages and other calculated values to the Footer.

### **2.11.5 Adding the Report to Your Project**

To add this report to your project you should use the same approach as seen in Part II: just use a RvProject in a Form or DataModule, link it to the report file, and call its Execute method. But there is one gotcha when using Driver Data Views: your application must load the appropriate driver. To do that, just add the unit RvDLBDE to your uses clause, if using BDE, RvDLDBX if using DbExpress, or RvDLADO if using ADO.

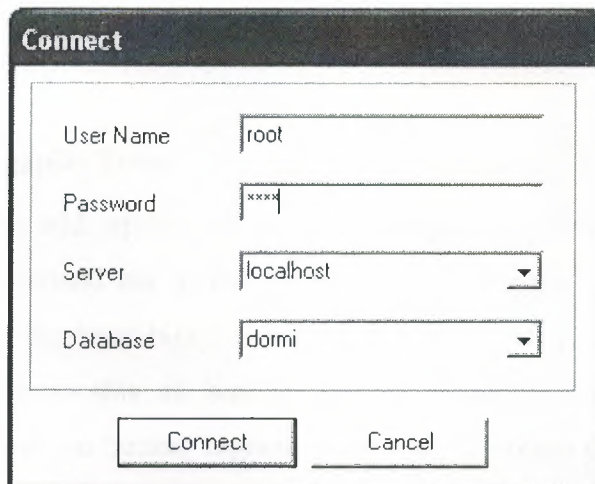


## CHAPTER 3

### 3 USER MANUAL

#### 3.1 Database Connection Screen

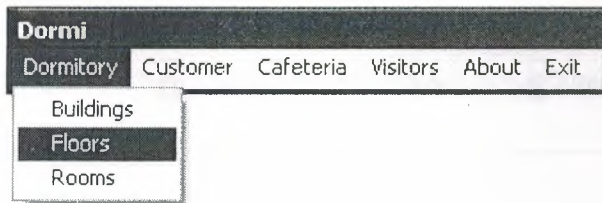
When user executes program, first database server connection screen appears. In this screen user enters user name, password, address of database server and database name which contains dormitory information's in it. In order to use the program user must have a valid user name and password. Also user must have appropriate privileges on database; such as view, add, update, delete.



The image shows a 'Connect' dialog box with a dark title bar. Inside, there are four input fields: 'User Name' with the text 'root', 'Password' with masked characters 'xxxx', 'Server' with a dropdown menu showing 'localhost', and 'Database' with a dropdown menu showing 'dormi'. At the bottom, there are two buttons: 'Connect' and 'Cancel'.

**Figure 3.1** Database Connection Screen

### 3.2 Main Menu



**Figure 3.2 Main Menu**

When the connection has been made and user granted to access database; “Main Menu” shows up. As you can see in the figure there are 6 main sections on main menu. 4 of them have sub menus (Dormitory, Customer, Cafeteria, Visitors) and two of them are accessed directly (About, Exit).

### 3.3 Buildings Menu

In this menu user can manage and organize the dormitory buildings, their floors and rooms.

#### 3.3.1 Building Organize Form

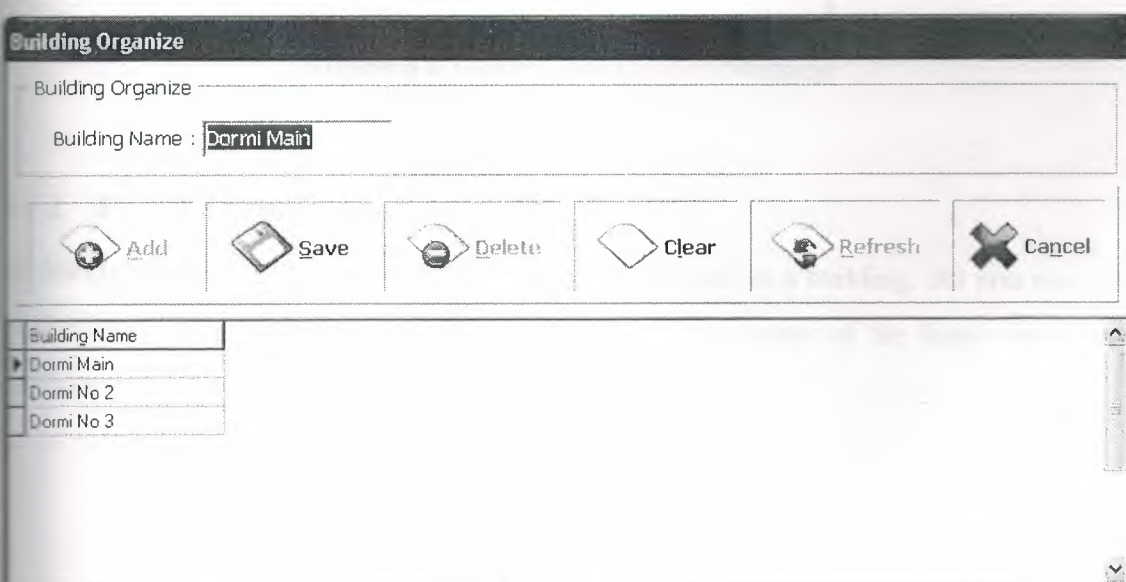
In this form user can add, update, or delete buildings. As you may see in the figure below, there is a command bar in the center of form. This is called standard bar. In order to increase understandability, simple icons are added according to buttons’ functions. In addition to this all buttons has keyboard shortcut key. If you press underlined latter shown on button, together with “ALT” button on keyboard, program will automatically consider that you clicked that button and will execute its function. As an example you entered a building name and you want to add it. No need to remove your hand from keyboard and reach to mouse than move it over the Add button and click it. Just simply press “ALT” and “A” buttons together on keyboard. That’s it. You have done it. Building has been added.



The screenshot shows the 'Building Organize' form in its initial state. At the top, there is a title bar 'Building Organize' and a subtitle 'Building Organize'. Below this is a text field labeled 'Building Name :'. Underneath the text field is a horizontal toolbar with six buttons: 'Add' (with a plus icon), 'Edit' (with a pencil icon), 'Delete' (with a minus icon), 'Clear' (with a diamond icon), 'Refresh' (with a circular arrow icon), and 'Close' (with an 'X' icon). Below the toolbar is a list box titled 'Building Name' containing three items: 'Dormi Main', 'Dormi No 2', and 'Dormi No 3'. The 'Dormi Main' item is selected and highlighted.

**Figure 3.3** Building Organize Form

Also all database processes related with building can be done in single form. You don't have to open another form and enter building name then search it, after you find it enter new information and update then come back again previous form. No need to do this. Just simply select building that you want to update in list then click edit or press "ALT" and "E" together on your keyboard. Your form will transform its self to do update process like figure below. As you can see in the figure, only the buttons related with update is enabled. Rest of them is disabled.

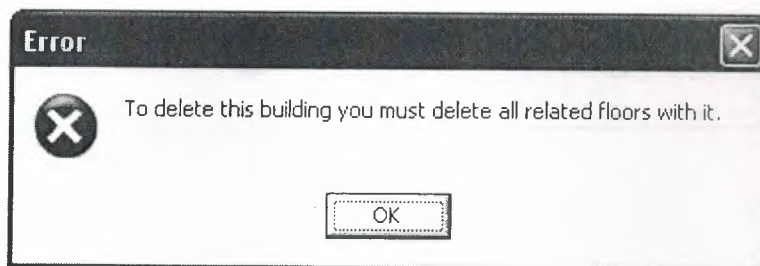


The screenshot shows the 'Building Organize' form in its edit mode. The title bar remains 'Building Organize'. The subtitle is still 'Building Organize'. The text field labeled 'Building Name :' now contains the text 'Dormi Main'. The toolbar has changed: the 'Add' button is disabled (grayed out), the 'Save' button (with a floppy disk icon) is now enabled, the 'Delete' button is disabled, the 'Clear' button is disabled, the 'Refresh' button is disabled, and the 'Cancel' button (with an 'X' icon) is now enabled. The list box titled 'Building Name' still contains the same three items: 'Dormi Main', 'Dormi No 2', and 'Dormi No 3'. The 'Dormi Main' item is still selected and highlighted.

**Figure 3.4** Building Organize Form in Edit Mode

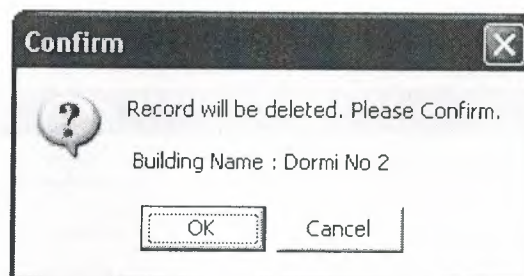


If you try to delete a building which has floors related with it you will get an error message as shown in figure below. To delete a building there must be no floors related with it.



**Figure 3.5** Delete Error Message

If building has no relation with floors table then you will get a confirmation as shown in figure below. If you confirm that process with clicking OK button record will be deleted permanently from database. If you don't confirm with clicking Cancel button no change will be done. Record will remain in table.



**Figure 3.6** Delete Confirmation Message

### 3.3.2 Floor Organize Form







In this form user can add, update, or delete floors related to a building. As you may see in the figure below, there is same command bar in the center of the form which was explained in previous form.

**Floor Organize**

Floor Organize

Building Name :

Floor Name :

 Add
  Edit
  Delete
  Clear
  Refresh
  Close

Building Name	Floor Name
Dormi Main	Floor 1
Dormi Main	Floor 2
Dormi Main	Floor 3
Dormi No 1	Floor 1
Dormi No 1	Floor 2
Dormi No 1	Floor 3
Dormi No 1	Floor 4

**Figure 3.7** Floor Organize Form







In the figure below form is transformed to edit mode. In addition to previous editing explanations, in this form you can change floor's building name. That means you can move a floor and all related information to another building with one click.

**Floor Organize**

Floor Organize

Building Name :

Floor Name :

 Add
  Save
  Delete
  Clear
  Refresh
  Cancel

Building Name	Floor Name
Dormi Main	Floor 1
Dormi Main	Floor 2
Dormi Main	Floor 3
Dormi No 1	Floor 1
Dormi No 1	Floor 2
Dormi No 1	Floor 3
Dormi No 1	Floor 4

**Figure 3.8** Floor Organize Form in Edit Mode

### 3.3.3 Room Organize Form

In this form user can add, update, or delete rooms related to a floor. As you may see in the figure below, there is same command bar in the center of the form which was explained previously. To add a room, user selects a building after that floors related with selected building is added to list and floor list becomes enabled. Then user selects floor name and enters a room name and defines room capacity. After all information has been entered to form, user clicks Add button or just simply presses “ALT” and “A” keys together on keyboard.

Building Name	Floor Name	Room Name	Room Capacity
Dormi Main	Floor 1	Room 101	1
Dormi Main	Floor 1	Room 102	1
Dormi Main	Floor 1	Room 103	1
Dormi Main	Floor 1	Room 104	1
Dormi Main	Floor 1	Room 105	1
Dormi Main	Floor 1	Room 106	1
Dormi Main	Floor 1	Room 107	1
Dormi Main	Floor 1	Room 108	1
Dormi Main	Floor 1	Room 109	1
Dormi Main	Floor 2	Room 201	1
Dormi Main	Floor 2	Room 202	1

Figure 3.9 Room Organize Form



In the figure below form is transformed to edit mode.

Room Organize

Room Organize

Building Name : 

Dormi Main

Floor Name : 

Floor 1

Room Name : 

Room 101

Room Capacity : 

1

Add

Save

Delete

Clear

Refresh

Cancel

Building Name	Floor Name	Room Name	Room Capacity
Dormi Main	Floor 1	Room 101	1
Dormi Main	Floor 1	Room 102	1
Dormi Main	Floor 1	Room 103	1
Dormi Main	Floor 1	Room 104	1
Dormi Main	Floor 1	Room 105	1
Dormi Main	Floor 1	Room 106	1
Dormi Main	Floor 1	Room 107	1
Dormi Main	Floor 1	Room 108	1
Dormi Main	Floor 1	Room 109	1
Dormi Main	Floor 2	Room 201	1
Dormi Main	Floor 2	Room 202	1

Figure 3.10 Room Organize Form in Edit Mode

### 3.4 Customers Menu

#### 3.4.1 Customer Organize

In this form user can add, update, or delete customers. Customer may not be related with a room. You can clear room information of customer and leave it in database for future use. As you may see in the figure below, there is same command bar in the center of the form which was explained previously. There is only one difference from previous command bar. Print button was added and Clear button removed from this bar.

To add a customer user must fill bold areas which are name, surname and gender. Other informations are optional. After the informations has been entered to form, user clicks Add button or just simply presses “ALT” and “A” keys together on keyboard.

**Customer Organize**

**Customer Informations**

Name :

Surname :

Gender :

Birth Date : 03.05.1983

Place Of Birth :

Blood Group :

Citizen Number :

Cell Phone :

Other Phone :

Contact Address :

**Customer's Father Informations**

Father's Name :

Father's Job :

Father's Phone :

Father's Address :

**Customer's Mother Informations**

Mother's Name :

Mother's Job :

Mother's Phone :

Mother's Address :

**Rent Informations**

Building Name :

Floor Name :

Room Name :

Rent Status :

Rent Duration :

Notes :

Add

Edit

Delete

Clear

Refresh

Print

Close

Name	Surname	Gender	Building	Floor	Room	Birthdate	Place Of Birth	Blood Group	Citizen Number	Cellphone	Other Phone	Fathers Name
Ismail	CELİK	M	Dormi M Floor 1	Room 1C	03.05.1983	Ankara	0 Rh (-)	43453478431	05335556677	05422223344	Ahmet	
Unal	TAŞDİZEN	M	Dormi M Floor 2	Room 2C	21.08.1982	KIRKLARELİ	B Rh (+)	45418514732	05355613171	05332221133	Adil	

Figure 3.11 Customer Organize Form

In the figure below form is transformed to edit mode.

**Customer Organize**

**Customer Informations**

Name : Unal

Surname : TAŞDİZEN

Gender : Male

Birth Date : 21.08.1982

Place Of Birth : KIRKLARELİ

Blood Group : B Rh (+)

Citizen Number : 45418514732

Cell Phone : 053355613171

Other Phone : 05332221133

Contact Address : Kocasinan Mah. Dutluk Sok. Kantürer Apt. No 10/2 Lüleburgaz / KIRKLARELİ

**Customer's Father Informations**

Father's Name : Adil

Father's Job : Retired

Father's Phone : 0532112233

Father's Address : Kocasinan Mah. Dutluk Sok. Kantürer Apt. No 10/2 Lüleburgaz /

**Customer's Mother Informations**

Mother's Name : Naciye

Mother's Job : House Wife

Mother's Phone : 05447778899

Mother's Address : Kocasinan Mah. Dutluk Sok. Kantürer Apt. No 10/2 Lüleburgaz /

**Rent Informations**

Building Name : Dormi Main

Floor Name : Floor 2

Room Name : Room 203

Rent Status : Staying

Rent Duration : 01.10.2007/31.01.2008

Notes : First Payment Done. Second Payment will be 15.12.2007

Add

Save

Delete

Clear

Refresh

Print

Cancel

Name	Surname	Gender	Building	Floor	Room	Birthdate	Place Of Birth	Blood Group	Citizen Number	Cellphone	Other Phone	Fathers Name
Ismail	CELİK	M	Dormi M Floor 1	Room 1C	03.05.1983	Ankara	0 Rh (-)	43453478431	05335556677	05422223344	Ahmet	
Unal	TAŞDİZEN	M	Dormi M Floor 2	Room 2C	21.08.1982	KIRKLARELİ	B Rh (+)	45418514732	05355613171	05332221133	Adil	

Figure 3.12 Customer Organize Form in Edit Mode

If user clicks Print button Output Options Screen appears. In this screen user decides what to do with output. There are three choices

- 1. Send To Disk
- 2. Send To Screen
- 3. Send To Printer

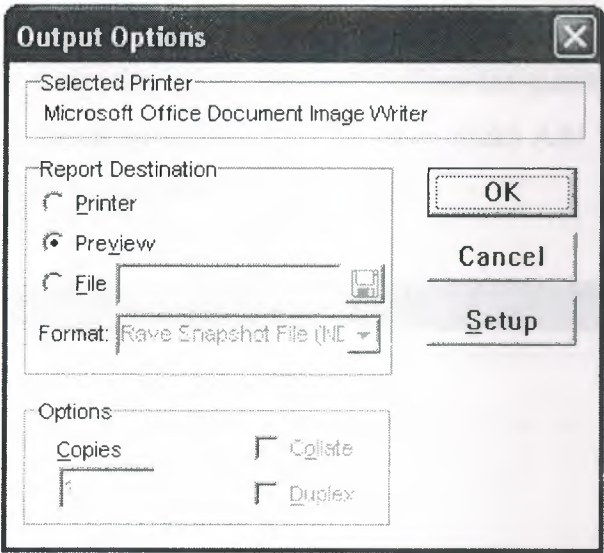


Figure 3.13 Output Options Window

If user selects Preview option sample of printing paper appears on the screen. With the help of this screen user has an idea when really has printed the list.

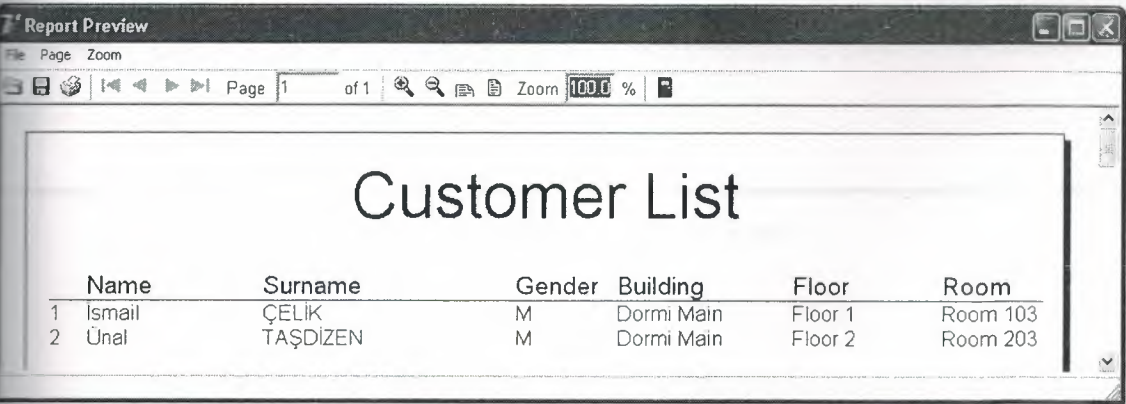


Figure 3.14 Print Preview of Customer List



## 5 Cafeteria Menu

This menu contains two forms which are Product Organize and Sale form.

### 5.1 Product Organize Form

To sell something in cafeteria user must enter some products to be sold. In this form user can add, update, or delete products. As you may see in the figure below, there is same command bar in the center of the form which was explained customer organize form.

To add a product user enters the information and defines the quantity of goods in stock. After the informations has been entered to form, user clicks Add button or just simply presses "ALT" and "A" keys together on keyboard.

**Product Organize**







Product Informations

Product Name :

Product Price :  (eg: 2,13)

Product Barcode :

Product Quantity :

 **Add**    **Edit**    **Delete**    **Refresh**    **Print**    **Close**

Product Name	Product Quantity	Product Price	Product Barcode
Coca Cola 1Lt	20	2	867544325
Coca Cola 1Lt Diet	7	2,25	862243556
Coca Cola 2Lt	10	3,5	876655432
Coca Cola 450ml	28	1,5	860506073
Fanta 1Lt	8	2	867754321
Fanta 2Lt	10	3,5	865546898
Fanta 330ml	5	1,5	865432567

**Figure 3.15** Product Organize Form

In the figure below form is transformed to edit mode.

Product Organize

Product Informations

Product Name :Coca Cola 1Lt

Product Price :2(eg: 2,13)

Product Barcode :867544325

Product Quantity :20

Add

Save

Delete

Refresh

Print

Cancel

Product Name	Product Quantity	Product Price	Product Barcode
Coca Cola 1Lt	20	2	867544325
Coca Cola 1Lt Diet	7	2,25	862243556
Coca Cola 2Lt	10	3,5	876655432
Coca Cola 450ml	28	1,5	860506073
Fanta 1Lt	8	2	867754321
Fanta 2Lt	10	3,5	865546898
Fanta 330ml	5	1,5	865432567

Figure 3.16 Product Organize Form in Edit Mode

If user selects Preview option after clicking Print button the figure below will be shown on screen. With the help of this screen user has an idea when really has printed the list.

Report Preview

FilePageZoom

Page 1 of 1

Zoom 100.0 %

Product Report

Product Name	Quantity	Price	Barcode
1 Coca Cola 1Lt	20	2,00	867544325
2 Coca Cola 1Lt Diet	7	2,25	862243556
3 Coca Cola 2Lt	10	3,50	876655432
4 Coca Cola 450ml	28	1,50	860506073
5 Fanta 1Lt	8	2,00	867754321
6 Fanta 2Lt	10	3,50	865546898
7 Fanta 330ml	5	1,50	865432567

Figure 3.17 Print Preview of Product Report

3.5.2 Sales Form

In this form user can make sale as long as the goods are in stock. As you may see in the figure below, there is same command bar in the center of the form which was explained customer organize form. Only one difference is Edit button was removed due to security issues.

To make a sale user selects a product form list and adjusts the sale quantity as desired. After the informations has been entered to form, user clicks Add button or just simply presses “ALT” and “A” keys together on keyboard. Sale time automatically assigned by system.

Sale Informations

Product Name : Coca Cola 450ml

In Stock : 25Barcode : 860506073

Sale Quantity : 3

Product Price : 1,5

Total : 4,5

Customer Informations

Building Name : Dormi Main

Floor Name : Floor 2

Room Name : Room 203

Customer Name : Ünal TAŞDİZEN

Show Sales : All

Add

Delete

Refresh

Print

Close

Customer Info	Product Info	Price	Quantity	Total	Sale Time
Not From Dormitory	Coca Cola 1Lt Diet	2,25	1	2,25	07.01.2008 18:48:06
Not From Dormitory	Fanta 330ml	1,5	2	3	04.01.2008 18:47:23
Dormi Main > Floor 1 > Room 103 > İsmail ÇELİK	Fanta 2Lt	3,5	3	10,5	04.01.2008 18:45:43
Dormi Main > Floor 2 > Room 203 > Ünal TAŞDİZEN	Coca Cola 450ml	1,5	3	4,5	04.01.2008 18:44:26
Not From Dormitory	Coca Cola 1Lt Diet	2,25	3	6,75	04.01.2008 18:43:37

Figure 3.18 Sales Form



If user selects Preview option after clicking Print button the figure below will be shown on screen. With the help of this screen user has an idea when really has printed the list.

Customer Info	Product Info	Quantity	Price	Total Price	Sale Time
1 Not From Dormitory	Coca Cola 1Lt Diet	1	2,25	2,25	07.01.2008 18:48:06
2 Not From Dormitory	Fanta 330ml	2	1,50	3,00	04.01.2008 18:47:23
3 Dormi Main > Floor 1 > Room 103 > Ismail ÇELİK	Fanta 2Lt	3	3,50	10,50	04.01.2008 18:45:43
4 Dormi Main > Floor 2 > Room 203 > Ünal TAŞDİZEN	Coca Cola 450ml	3	1,50	4,50	04.01.2008 18:44:26
5 Not From Dormitory	Coca Cola 1Lt Diet	3	2,25	6,75	04.01.2008 18:43:37

Figure 3.19 Print Preview of Sales Report

## 3.6 Visitors Menu

### 3.6.1 Visitor Organize Form

With the help of this form it is possible to track visitors entered and leaved the dormitory buildings. As you may see in the figure below, there is same command bar in the center of the form which was explained customer organize form.

To record a visit user must give the building, floor and room and visitor informations. Customer info is optional. After that user may adjust the entrance time as desired or may get the current time of system. After the informations has been entered to form, user clicks Add button or just simply presses "ALT" and "A" keys together on keyboard.

To end visit user simply selects related visit record and presses edit. In this mode only the exiting time field is enabled. User adjusts the exiting time as desired or gets the current system time and then presses the save button.

**Visitor Organize**

Visited Customer Informations

Building Name :

Floor Name :

Room Name :

Customer Name :


Show Visits :


Visitor Informations


Visitor(s) Info :


Entering Time :


Exiting Time :


 **Get Current Time**


 **Add**

 **Edit**

 **Delete**

 **Refresh**

 **Print**

 **Close**

Visited Customer Info	Visitor Info	Entering Time	Exiting Time
Dormi Main > Floor 2 > Room 203 > Ünal TAŞDİZEN	Tamer KUKLEN, Ali HARP	08.01.2008 21:06:05	08.01.2008 23:07:23
Dormi Main > Floor 1 > Room 101	5 Classmates	08.01.2008 19:07:30	08.01.2008 23:15:14
Dormi Main > Floor 3 > Room 301	Sevilay GÜNER	08.01.2008 19:07:30	08.01.2008 22:08:56
Dormi Main > Floor 2 > Room 203 > Ünal TAŞDİZEN	Ümit YÜCEL, Ufuk YILDIRIM	08.01.2008 19:03:06	08.01.2008 21:05:59

**Figure 3.20** Visitor Organize Form

If user selects Preview option after clicking Print button the figure below will be shown on screen. With the help of this screen user has an idea when really has printed the list.

**Report Preview**

Page Zoom

Page 1 of 1

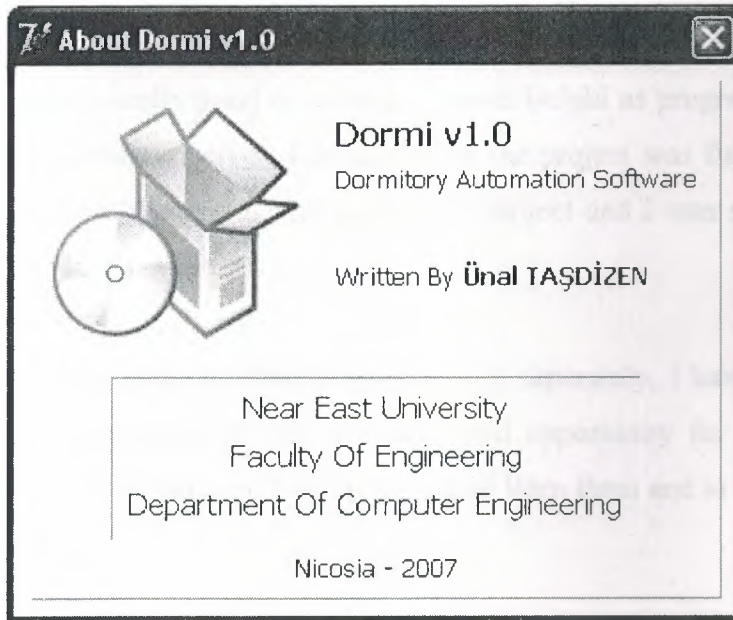
Zoom 100.0 %

Dormitory Visitor Report			
Visited Customer Info	Visitor Info	Entering Time	Exiting Time
1 Dormi Main > Floor 2 > Room 203 > Ünal TAŞDİZEN	Tamer KUKLEN, Ali HARP	08.01.2008 21:06:05	08.01.2008 23:07:23
2 Dormi Main > Floor 1 > Room 101	5 Classmates	08.01.2008 19:07:30	08.01.2008 23:15:14
3 Dormi Main > Floor 3 > Room 301	Sevilay GÜNER	08.01.2008 19:07:30	08.01.2008 22:08:56
4 Dormi Main > Floor 2 > Room 203 > Ünal TAŞDİZEN	Ümit YÜCEL, Ufuk YILDIRIM	08.01.2008 19:03:06	08.01.2008 21:05:59

**Figure 3.21** Print Preview of Dormitory Visitor Report

### 3.7 About Menu

This form gives some information about the program and the writer of it.



**Figure 3.22** About Form

### 3.8 Exit Menu

Once clicked, program immediately terminates it self where ever its or what ever it is doing.



## CONCLUSION

Since I am experienced on Delphi and SQL languages I did not had too much difficulties on this project. When I get stuck, mostly I used internet but also I searched some books. It was a really good decision to choose Delphi as programming language and MySQL as a database server. Because when the project was finished I achieved everything that I have planned before starting to project and I was satisfied with the result that I have got at the end.

Although I have experience on Delphi and MySQL separately, I haven't used Delphi and MySQL together before. It was a really good opportunity for me to use them together in such a project because I was planning to learn them and to do such a project for a very long time.

For future implementations I tried to make design and programming back bone as flexible as possible. Any additional modules can be added to automation without much effort. Printing templates are also stored external in order to make them updateable according to needs.

## REFERENCES

<http://www.codegear.com>

[http://www.scalabium.com/faq/dc\\_tips.htm](http://www.scalabium.com/faq/dc_tips.htm)

<http://www.nevrona.com/>

Delphi Programming Explorer, Jeff Dontemann – Jim Mischel ISBN 1-883-57725-X

Database Application Developers Book for Delphi (e Book)

Borland Delphi 6 for Windows (e Book)

Mastering Delphi 6 – Marco Cantu

# APPENDIX

## Program Codes

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, DB, DBAccess, MyDacVcl, MemDS, MyAccess, Menus,
  StdCtrls;

type
  TMainForm = class(TForm)
    MyConnection: TMyConnection;
    MyConnectDialog: TMyConnectDialog;
    MainMenu1: TMainMenu;
    DormitoryManagement1: TMenuItem;
    Buildings1: TMenuItem;
    Floors1: TMenuItem;
    Rooms1: TMenuItem;
    CustomerManagement1: TMenuItem;
    Organize1: TMenuItem;
    Cafeteria1: TMenuItem;
    Visitors1: TMenuItem;
    Organize2: TMenuItem;
    ProductOrganize1: TMenuItem;
    Sale1: TMenuItem;
    Exit1: TMenuItem;
    About1: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure Buildings1Click(Sender: TObject);
    procedure Floors1Click(Sender: TObject);
    procedure Rooms1Click(Sender: TObject);
    procedure Organize1Click(Sender: TObject);
    procedure Organize2Click(Sender: TObject);
    procedure ProductOrganize1Click(Sender: TObject);
    procedure Sale1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```



MainForm: TMainForm;

implementation

uses Buildings, Floors, Rooms, Customers, Visitors, Products, Sales, About;

{ \$R \*.dfm }

procedure TMainForm.FormCreate(Sender: TObject);

begin

    MainForm.Top := 0;

    MainForm.ClientHeight := 0;

    MyConnection.Connect;

end;

procedure TMainForm.Buildings1Click(Sender: TObject);

begin

    BuildingsForm.Show;

end;

procedure TMainForm.Floors1Click(Sender: TObject);

begin

    FloorsForm.Show;

end;

procedure TMainForm.Rooms1Click(Sender: TObject);

begin

    RoomsForm.Show;

end;

procedure TMainForm.Organize1Click(Sender: TObject);

begin

    CustomersForm.Show;

end;

procedure TMainForm.Organize2Click(Sender: TObject);

begin

    VisitorsForm.Show;

end;

procedure TMainForm.ProductOrganize1Click(Sender: TObject);

begin

    ProductsForm.Show;

end;

procedure TMainForm.Sale1Click(Sender: TObject);

begin

    SalesForm.Show;

end;

```
procedure TMainForm.Exit1Click(Sender: TObject);
begin
    Close;
end;
```

```
procedure TMainForm.About1Click(Sender: TObject);
begin
    AboutForm.Show;
end;
```

```
end.
```

```
unit Buildings;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, DB, MemDS, DBAccess, MyAccess, StdCtrls, Mask, DBCtrls,
ExtCtrls, Grids, DBGrids, Buttons;
```

```
type
```

```
TBuildingsForm = class(TForm)
```

```
    MyQuery: TMyQuery;
```

```
    DataSource: TDataSource;
```

```
    DBGrid: TDBGrid;
```

```
    CommandPanel: TPanel;
```

```
    AddBevel: TBevel;
```

```
    EditSaveBevel: TBevel;
```

```
    ClearBevel: TBevel;
```

```
    DeleteBevel: TBevel;
```

```
    CloseBevel: TBevel;
```

```
    AddBtn: TSpeedButton;
```

```
    SaveBtn: TSpeedButton;
```

```
    EditBtn: TSpeedButton;
```

```
    ClearBtn: TSpeedButton;
```

```
    DeleteBtn: TSpeedButton;
```

```
    BuildingGroup: TGroupBox;
```

```
    Label1: TLabel;
```

```
    BuildingName: TEdit;
```

```
    CloseBtn: TSpeedButton;
```

```
    RefreshBevel: TBevel;
```

```
    RefreshBtn: TSpeedButton;
```

```
    MyListQuery: TMyQuery;
```

```
    CancelBtn: TSpeedButton;
```

```
    procedure EditBtnClick(Sender: TObject);
```

```
    procedure SaveBtnClick(Sender: TObject);
```

```
    procedure ClearBtnClick(Sender: TObject);
```

```

procedure AddBtnClick(Sender: TObject);
procedure RefreshBtnClick(Sender: TObject);
procedure CloseBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure DBGridTitleClick(Column: TColumn);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure FormShow(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure CancelBtnClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  BuildingsForm: TBuildingsForm;
  SortingField, SortingMethod: String;
  PreviousColumnIndex : Integer = 0;

implementation

{$R *.dfm}

procedure TBuildingsForm.EditBtnClick(Sender: TObject);
begin
  BuildingName.Text := MyListQuery.Fields[1].AsString;
  BuildingName.SetFocus;
  BuildingName.SelectAll;
  EditBtn.Visible := False;
  CloseBtn.Visible := False;
  CancelBtn.Visible := True;
  SaveBtn.Visible := True;
  AddBtn.Enabled := False;
  DeleteBtn.Enabled := False;
  RefreshBtn.Enabled := False;
end;

procedure TBuildingsForm.SaveBtnClick(Sender: TObject);
var
  BuildingID: String;
begin
  BuildingID := MyListQuery.Fields[0].AsString;
  BuildingName.Text := Trim(BuildingName.Text);
  If (BuildingName.Text = "") Then
  Begin
    MessageDlg('You must enter a valid Building Name to save this record.', mtError,
      [mbOk], 0);
  End;
end;

```



```

Exit;
End;
With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Update buildings Set building_name=:building_name Where
id=:building_id';
    Params[0].AsString := BuildingName.Text;
    Params[1].AsString := BuildingID;
    Execute;
End;
SaveBtn.Visible := False;
EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;
ClearBtn.Click;
RefreshBtn.Click;
end;

```

```

procedure TBuildingsForm.ClearBtnClick(Sender: TObject);
begin
    BuildingName.Text := "";
end;

```

```

procedure TBuildingsForm.AddBtnClick(Sender: TObject);
begin
    BuildingName.Text := Trim(BuildingName.Text);

    If (BuildingName.Text = "") Then
    Begin
        MessageDlg('You must enter a valid Building Name to save this record.', mtError,
[mbOk], 0);
        Exit;
    End;

```

```

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Insert Into buildings Set building_name=:building_name';
    Params[0].AsString := BuildingName.Text;
    Execute;
End;
ClearBtn.Click;
RefreshBtn.Click;

```

end;

```
procedure TBuildingsForm.RefreshBtnClick(Sender: TObject);
begin
  With MyListQuery Do
  Begin
    Close;
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select * From buildings Order By '+SortingField+SortingMethod;
    Open;
  End;
end;
```

```
procedure TBuildingsForm.CloseBtnClick(Sender: TObject);
begin
  BuildingsForm.Close;
end;
```

```
procedure TBuildingsForm.DeleteBtnClick(Sender: TObject);
var
  BuildingID, BuildingName: String;
begin
  BuildingID := MyListQuery.Fields[0].AsString;
  BuildingName := MyListQuery.Fields[1].AsString;
  With MyQuery Do
  Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select count(id) From floors Where building_id=:building_id';
    Params[0].AsString := BuildingID;
    Execute;
    If (MyQuery.Fields[0].AsInteger > 0 ) Then
    Begin
      MessageDlg('To delete this building you must delete all related floors with it.',
mtError, [mbOk], 0);
    End
  Else
  Begin
    If MessageDlg('Record will be deleted. Please Confirm. '#10#10' Building Name :
'+BuildingName, mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then
    Begin
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Delete From buildings Where id=:building_id';
      Params[0].AsString := BuildingID;
      Execute;
    End;
  End;
end;
```

```

End;
RefreshBtn.Click;
end;

procedure TBuildingsForm.DBGridTitleClick(Column: TColumn);
begin
  If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
    SortingMethod := "
  Else
    If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';
  SortingField := DBGrid.Columns[Column.Index].FieldName;
  Try
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
      DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
  Except
  End;
  Column.title.Font.Style := Column.title.Font.Style + [fsBold];
  PreviousColumnIndex := Column.Index;
  RefreshBtn.Click;
end;

procedure TBuildingsForm.DBGridMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
var
  pt: TGridcoord;
begin
  pt:= DBGrid.MouseCoord(x, y);
  If pt.y=0 Then
    DBGrid.Cursor:=crHandPoint
  Else
    DBGrid.Cursor:=crDefault;
end;

procedure TBuildingsForm.FormShow(Sender: TObject);
begin
  SaveBtn.Visible := False;
  CancelBtn.Visible := False;
  ClearBtn.Click;
  SortingField := 'building_name';
  SortingMethod := "";
  RefreshBtn.Click;
end;

procedure TBuildingsForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

```



```
procedure TBuildingsForm.CancelBtnClick(Sender: TObject);
begin
```

```
    SaveBtn.Visible := False;
    EditBtn.Visible := True;
    CancelBtn.Visible := False;
    CloseBtn.Visible := True;
    AddBtn.Enabled := True;
    DeleteBtn.Enabled := True;
    RefreshBtn.Enabled := True;
    DBGrid.Enabled := True;
    DBGrid.SetFocus;
    ClearBtn.Click;
```

```
end;
```

```
end.
```

```
unit Floors;
```

```
interface
```

```
uses
```

```
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, DB, MemDS, DBAccess, MyAccess, Buttons, ExtCtrls,
    Grids, DBGrids, ComCtrls;
```

```
type
```

```
TFloorsForm = class(TForm)
    FloorGroup: TGroupBox;
    Label1: TLabel;
    FloorName: TEdit;
    DBGrid: TDBGrid;
    CommandPanel: TPanel;
    EditSaveBevel: TBevel;
    SaveBtn: TSpeedButton;
    EditBtn: TSpeedButton;
    AddBevel: TBevel;
    ClearBevel: TBevel;
    DeleteBevel: TBevel;
    CloseBevel: TBevel;
    AddBtn: TSpeedButton;
    ClearBtn: TSpeedButton;
    DeleteBtn: TSpeedButton;
    CloseBtn: TSpeedButton;
    RefreshBevel: TBevel;
    RefreshBtn: TSpeedButton;
    MyQuery: TMyQuery;
    DataSource: TDataSource;
    Label2: TLabel;
```

```

BuildingsCmb: TComboBox;
MyListQuery: TMyQuery;
CancelBtn: TSpeedButton;
procedure RefreshBtnClick(Sender: TObject);
procedure ClearBtnClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure CloseBtnClick(Sender: TObject);
procedure AddBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure DBGridCellClick(Column: TColumn);
procedure DBGridTitleClick(Column: TColumn);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure EditBtnClick(Sender: TObject);
procedure SaveBtnClick(Sender: TObject);
procedure CancelBtnClick(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
  { Private declarations }
public
  { Public declarations }
end;

```

```

var
FloorsForm: TFloorsForm;
SortingField, SortingMethod: String;
PreviousColumnIndex : Integer = 0;

```

implementation

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

```

procedure TFloorsForm.FormShow(Sender: TObject);
begin
  SaveBtn.Visible := False;
  CancelBtn.Visible := False;
  ClearBtn.Click;
  SortingField := 'building_name, floor_name';
  SortingMethod := '';
  RefreshBtn.Click;
end;

```

```

procedure TFloorsForm.ClearBtnClick(Sender: TObject);
begin
  BuildingsCmb.ItemIndex := -1;
  FloorName.Text := '';
end;

```

```

procedure TFloorsForm.RefreshBtnClick(Sender: TObject);

```

```

begin
  BuildingsCmb.Clear;
  // FloorName.Text := "";
  With MyQuery Do
    Begin
      Close;
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Select * From buildings Order By building_name';
      Execute;
      While Not Eof Do
        Begin
          BuildingsCmb.Items.AddObject(FieldByName('building_name').AsString,
            TObject(FieldByName('id').AsInteger));
        Next;
      End;
    End;
  With MyListQuery Do
    Begin
      Close;
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Select * From buildings, floors Where buildings.id=floors.building_id
Order By '+SortingField+SortingMethod;
      Execute;
    End;
  end;

```

```

procedure TFloorsForm.CloseBtnClick(Sender: TObject);
begin
  FloorsForm.Close;
end;

```

```

procedure TFloorsForm.AddBtnClick(Sender: TObject);
Var
  BuildingID: String;
begin
  FloorName.Text := Trim(FloorName.Text);
  If (BuildingsCmb.ItemIndex < 0) Then
    Begin
      MessageDlg('You must select a Building Name to save this record.', mtError,
[mbOk], 0);
      Exit;
    End;
  If (FloorName.Text = "") Then
    Begin
      MessageDlg('You must enter a valid Floor Name to save this record.', mtError,
[mbOk], 0);
      Exit;
    End;
  end;

```



```

End;

BuildingID:=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Insert Into floors Set building_id=:building_id,
floor_name=:floor_name';
    Params[0].AsString := BuildingID;
    Params[1].AsString := FloorName.Text;
    Execute;
End;
ClearBtn.Click;
RefreshBtn.Click;
end;

procedure TFloorsForm.DeleteBtnClick(Sender: TObject);
var
    FloorID, FloorName: String;
begin
    With MyQuery Do
    Begin
        FloorID := MyListQuery.FieldByName('id_1').AsString;
        FloorName := MyListQuery.FieldByName('floor_name').AsString;
        SQL.Clear;
        Params.Clear;
        SQL.Text := 'Select count(id) From rooms Where floor_id=:floor_id';
        Params[0].AsString := FloorID;
        Execute;
        If (Fields[0].AsInteger > 0 ) Then
            Begin
                MessageDlg('To delete this building you must delete all related rooms with it.',
mtError, [mbOk], 0);
            End
        Else
            Begin
                If MessageDlg('Record will be deleted. Please Confirm. '#10#10' Floor Name :
'+FloorName, mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then
                    Begin
                        SQL.Clear;
                        Params.Clear;
                        SQL.Text := 'Delete From floors Where id=:floor_id';
                        Params[0].AsString := FloorID;
                        Execute;
                    End;
                End;
            End;
        End;
    End;
end;

```

```

RefreshBtn.Click;
end;

procedure TFloorsForm.EditBtnClick(Sender: TObject);
Var
  I: Integer;
begin
  // FloorName.Text := IntToStr(BuildingsCmb.Items.Count);
  For I := 0 To BuildingsCmb.Items.Count-1 Do
  Begin
    If BuildingsCmb.Items.Strings[I] =
MyListQuery.FieldByName('building_name').AsString
    Then BuildingsCmb.ItemIndex := I;
  End;
  DBGrid.SetFocus;
  DBGrid.Enabled := False;
  FloorName.Text := MyListQuery.FieldByName('floor_name').AsString;
  FloorName.SetFocus;
  FloorName.SelectAll;
  EditBtn.Visible := False;
  CloseBtn.Visible := False;
  CancelBtn.Visible := True;
  SaveBtn.Visible := True;
  AddBtn.Enabled := False;
  DeleteBtn.Enabled := False;
  RefreshBtn.Enabled := False;
end;

procedure TFloorsForm.SaveBtnClick(Sender: TObject);
Var
  BuildingID, FloorID: String;
begin
  FloorName.Text := Trim(FloorName.Text);
  If (BuildingsCmb.ItemIndex < 0) Then
  Begin
    MessageDlg('You must select a Building Name to save this record.', mtError,
[mbOk], 0);
    Exit;
  End;
  If (FloorName.Text = '') Then
  Begin
    MessageDlg('You must enter a valid Floor Name to save this record.', mtError,
[mbOk], 0);
    Exit;
  End;

  BuildingID :=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
  FloorID := MyListQuery.FieldByName('id_1').AsString;

```

```

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Update floors Set building_id=:building_id, floor_name=:floor_name
Where id=:floor_id';
    Params[0].AsString := BuildingID;
    Params[1].AsString := FloorName.Text;
    Params[2].AsString := FloorID;
    Execute;
End;
SaveBtn.Visible := False;
EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;
DBGrid.Enabled := True;
ClearBtn.Click;
RefreshBtn.Click;
end;

```

```

procedure TFloorsForm.CancelBtnClick(Sender: TObject);
begin
    SaveBtn.Visible := False;
    EditBtn.Visible := True;
    CancelBtn.Visible := False;
    CloseBtn.Visible := True;
    AddBtn.Enabled := True;
    DeleteBtn.Enabled := True;
    RefreshBtn.Enabled := True;
    DBGrid.Enabled := True;
    DBGrid.SetFocus;
    ClearBtn.Click;
end;

```

```

procedure TFloorsForm.DBGridCellClick(Column: TColumn);
begin
    // For Testing Purpose
end;

```

```

procedure TFloorsForm.DBGridTitleClick(Column: TColumn);
begin
    If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
        SortingMethod := "
    Else
        If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';

```



```

SortingField := DBGrid.Columns[Column.Index].FieldName;
Try
  DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
Except
End;
Column.title.Font.Style := Column.title.Font.Style + [fsBold];
PreviousColumnIndex := Column.Index;
RefreshBtn.Click;
end;

procedure TFloorsForm.DBGridMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
var
  pt: TGridCoord;
begin
  pt:= DBGrid.MouseCoord(x, y);
  If pt.y=0 Then
    DBGrid.Cursor:=crHandPoint
  Else
    DBGrid.Cursor:=crDefault;
end;

procedure TFloorsForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

end.

unit Rooms;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, MemDS, DBAccess, MyAccess, Buttons, ExtCtrls, Grids,
  DBGrids, StdCtrls, Spin;

type
  TRoomsForm = class(TForm)
    RoomGroup: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    RoomName: TEdit;
    BuildingsCmb: TComboBox;
    DBGrid: TDBGrid;
    CommandPanel: TPanel;

```

```

CloseBevel: TBevel;
CancelBtn: TSpeedButton;
CloseBtn: TSpeedButton;
EditSaveBevel: TBevel;
SaveBtn: TSpeedButton;
EditBtn: TSpeedButton;
AddBevel: TBevel;
ClearBevel: TBevel;
DeleteBevel: TBevel;
AddBtn: TSpeedButton;
ClearBtn: TSpeedButton;
DeleteBtn: TSpeedButton;
RefreshBevel: TBevel;
RefreshBtn: TSpeedButton;
MyQuery: TMyQuery;
DataSource: TDataSource;
MyListQuery: TMyQuery;
FloorCpt: TLabel;
FloorsCmb: TComboBox;
RoomCapacity: TSpinEdit;
Label3: TLabel;
procedure CloseBtnClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure ClearBtnClick(Sender: TObject);
procedure RefreshBtnClick(Sender: TObject);
procedure BuildingsCmbChange(Sender: TObject);
procedure AddBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure EditBtnClick(Sender: TObject);
procedure CancelBtnClick(Sender: TObject);
procedure SaveBtnClick(Sender: TObject);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure DBGridTitleClick(Column: TColumn);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    RoomsForm: TRoomsForm;
    SortingField, SortingMethod: String;
    PreviousColumnIndex : Integer = 0;

implementation

{$R *.dfm}

```

```

procedure TRoomsForm.FormShow(Sender: TObject);
begin
    SortingField := 'building_name, floor_name, room_name';
    SortingMethod := '';
    SaveBtn.Visible := False;
    CancelBtn.Visible := False;
    ClearBtn.Click;
    RefreshBtn.Click;
end;

```

```

procedure TRoomsForm.ClearBtnClick(Sender: TObject);
begin
    FloorsCmb.Enabled := False;
    FloorsCmb.ItemIndex := -1;
    BuildingsCmb.ItemIndex := -1;
    RoomName.Text := '';
    RoomCapacity.Text := '1';
end;

```

```

procedure TRoomsForm.RefreshBtnClick(Sender: TObject);
begin
    BuildingsCmb.Clear;
    FloorsCmb.Enabled := False;
    FloorsCmb.ItemIndex := -1;
    With MyQuery Do
    Begin
        Close;
        SQL.Clear;
        Params.Clear;
        SQL.Text := 'Select * From buildings Order By building_name';
        Execute;
        While Not Eof Do
        Begin
            BuildingsCmb.Items.AddObject(FieldByName('building_name').AsString,
                TObject(FieldByName('id').AsInteger));
        End;
    End;
    With MyListQuery Do
    Begin
        Close;
        SQL.Clear;
        Params.Clear;
        SQL.Text := 'Select * From buildings, floors, rooms Where '+
            'buildings.id=floors.building_id And floors.id=rooms.floor_id '+
            'Order By '+SortingField+SortingMethod;
        Execute;
    End;
end;

```



nd;

```
procedure TRoomsForm.BuildingsCmbChange(Sender: TObject);
begin
  FloorsCmb.Clear;
  With MyQuery Do
  Begin
    Close;
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select * From floors Where building_id=:building_id '+
      'Order By floor_name';
    Params[0].AsString :=
      IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
    Execute;
    While Not Eof Do
    Begin
      FloorsCmb.Items.AddObject(FieldByName('floor_name').AsString,
        TObject(FieldByName('id').AsInteger));
    End;
  End;
  FloorsCmb.Enabled := True;
end;
```

```
procedure TRoomsForm.CloseBtnClick(Sender: TObject);
begin
  RoomsForm.Close;
end;
```

```
procedure TRoomsForm.AddBtnClick(Sender: TObject);
Var
  BuildingID, FloorID: String;
begin
  RoomName.Text := Trim(RoomName.Text);
  If (BuildingsCmb.ItemIndex < 0) Then
  Begin
    MessageDlg('You must select a Building Name to save this record.', mtError,
      [mbOk], 0);
    Exit;
  End;
  If (FloorsCmb.ItemIndex < 0) Then
  Begin
    MessageDlg('You must select a Floor Name to save this record.', mtError, [mbOk],
      0);
    Exit;
  End;
  If (RoomName.Text = '') Then
  Begin
```

```

    MessageDlg('You must enter a valid Room Name to save this record.', mtError,
[mbOk], 0);
    Exit;
End;

```

```

BuildingID :=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
FloorID := IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]));

```

```

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select count(id) From rooms Where '+'
'floor_id=:floor_id And room_name=:room_name';
    Params[0].AsString := FloorID;
    Params[1].AsString := RoomName.Text;
    Execute;
    If (Fields[0].AsInteger > 0 ) Then
    Begin
        MessageDlg('This room already exist.', mtError, [mbOk], 0);
        Exit;
    End;
End;

```

```

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Insert Into rooms Set '+'
'floor_id=:floor_id, room_name=:room_name, room_capacity=:room_capacity';
    Params[0].AsString := FloorID;
    Params[1].AsString := RoomName.Text;
    Params[2].AsString := RoomCapacity.Text;
    Execute;
End;
RoomName.Text := '';
RoomCapacity.Text := '1';
MyListQuery.Execute;
end;

```

```

procedure TRoomsForm.DeleteBtnClick(Sender: TObject);
var
    RoomID, RoomName: String;
begin
    With MyQuery Do
    Begin
        RoomID := MyListQuery.FieldName('id_2').AsString;
        RoomName := MyListQuery.FieldName('room_name').AsString;

```

```

    If MessageDlg('Record will be deleted. Please Confirm. '#10#10' Room Name :
'+RoomName, mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then
    Begin
        SQL.Clear;
        Params.Clear;
        SQL.Text := 'Delete From rooms Where id=:room_id';
        Params[0].AsString := RoomID;
        Execute;
        MyListQuery.Execute;
    End;
End;

end;

procedure TRoomsForm.EditBtnClick(Sender: TObject);
Var
    I: Integer;
begin
    For I := 0 To BuildingsCmb.Items.Count-1 Do
    Begin
        If BuildingsCmb.Items.Strings[I] =
MyListQuery.FieldByName('building_name').AsString
        Then BuildingsCmb.ItemIndex := I;
    End;
    BuildingsCmb.OnChange(Sender);
    For I := 0 To FloorsCmb.Items.Count-1 Do
    Begin
        If FloorsCmb.Items.Strings[I] = MyListQuery.FieldByName('floor_name').AsString
        Then FloorsCmb.ItemIndex := I;
    End;

    DBGrid.SetFocus;
    DBGrid.Enabled := False;
    RoomName.Text := MyListQuery.FieldByName('room_name').AsString;
    RoomCapacity.Text := MyListQuery.FieldByName('room_capacity').AsString;
    RoomName.SetFocus;
    RoomName.SelectAll;
    EditBtn.Visible := False;
    CloseBtn.Visible := False;
    CancelBtn.Visible := True;
    SaveBtn.Visible := True;
    AddBtn.Enabled := False;
    DeleteBtn.Enabled := False;
    RefreshBtn.Enabled := False;
end;

procedure TRoomsForm.CancelBtnClick(Sender: TObject);
begin
    SaveBtn.Visible := False;

```



```

EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;
DBGrid.Enabled := True;
DBGrid.SetFocus;
ClearBtn.Click;
end;

procedure TRoomsForm.SaveBtnClick(Sender: TObject);
Var
  FloorID, RoomID: String;
begin
  RoomName.Text := Trim(RoomName.Text);
  If (BuildingsCmb.ItemIndex < 0) Then
    Begin
      MessageDlg('You must select a Building Name to save this record.', mtError,
[mbOk], 0);
      Exit;
    End;
  If (FloorsCmb.ItemIndex < 0) Then
    Begin
      MessageDlg('You must select a Floor Name to save this record.', mtError, [mbOk],
0);
      Exit;
    End;
  If (RoomName.Text = '') Then
    Begin
      MessageDlg('You must enter a valid Room Name to save this record.', mtError,
[mbOk], 0);
      Exit;
    End;

  FloorID := IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]));
  RoomID := MyListQuery.FieldByName('id_2').AsString;
  With MyQuery Do
    Begin
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Select count(id) From rooms Where '+'
'floor_id=:floor_id And room_name=:room_name And id!=:room_id';
      Params[0].AsString := FloorID;
      Params[1].AsString := RoomName.Text;
      Params[2].AsString := RoomID;
      Execute;
      If (Fields[0].AsInteger > 0 ) Then
        Begin

```

```

MessageDlg('This room already exist.', mtError, [mbOk], 0);
Exit;
End;
End;

```

With MyQuery Do

```

Begin
  SQL.Clear;
  Params.Clear;
  SQL.Text := 'Update rooms Set floor_id=:floor_id, room_name=:room_name, '+
'room_capacity=:room_capacity Where id=:room_id';
  Params[0].AsString := FloorID;
  Params[1].AsString := RoomName.Text;
  Params[2].AsString := RoomCapacity.Text;
  Params[3].AsString := RoomID;
  Execute;
End;
SaveBtn.Visible := False;
EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;
DBGrid.Enabled := True;
ClearBtn.Click;
MyListQuery.Execute;
end;

```

```

procedure TRoomsForm.DBGridTitleClick(Column: TColumn);
begin
  If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
    SortingMethod := "
  Else
    If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';
  SortingField := DBGrid.Columns[Column.Index].FieldName;
  Try
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
  Except
  End;
  Column.title.Font.Style := Column.title.Font.Style + [fsBold];
  PreviousColumnIndex := Column.Index;
  RefreshBtn.Click;
end;

```

```

procedure TRoomsForm.DBGridMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
var

```

```

pt: TGridcoord;
begin
pt:= DBGrid.MouseCoord(x, y);
If pt.y=0 Then
    DBGrid.Cursor:=crHandPoint
Else
    DBGrid.Cursor:=crDefault;
end;

procedure TRoomsForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

end.

unit Customers;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, ComCtrls, StdCtrls, DB, MemDS, DBAccess, MyAccess, Buttons,
    ExtCtrls, Grids, DBGrids, RpCon, RpConDS, RpDefine, RpRave;

type
    TCustomersForm = class(TForm)
        DBGrid: TDBGrid;
        CommandPanel: TPanel;
        CloseBevel: TBevel;
        CancelBtn: TSpeedButton;
        CloseBtn: TSpeedButton;
        EditSaveBevel: TBevel;
        SaveBtn: TSpeedButton;
        EditBtn: TSpeedButton;
        AddBevel: TBevel;
        ClearBevel: TBevel;
        DeleteBevel: TBevel;
        AddBtn: TSpeedButton;
        ClearBtn: TSpeedButton;
        DeleteBtn: TSpeedButton;
        RefreshBevel: TBevel;
        RefreshBtn: TSpeedButton;
        MyQuery: TMyQuery;
        DataSource: TDataSource;
        GroupBox1: TGroupBox;
        Label19: TLabel;
        FathersPhone: TEdit;
    end;

```



Label15: TLabel;  
 Label17: TLabel;  
 FathersJob: TEdit;  
 FathersName: TEdit;  
 Label3: TLabel;  
 GroupBox2: TGroupBox;  
 Label4: TLabel;  
 Label16: TLabel;  
 Label18: TLabel;  
 Label20: TLabel;  
 MothersName: TEdit;  
 MothersJob: TEdit;  
 MothersPhone: TEdit;  
 GroupBox3: TGroupBox;  
 Label9: TLabel;  
 Label22: TLabel;  
 Label23: TLabel;  
 Label24: TLabel;  
 Label25: TLabel;  
 PlaceOfBirth: TEdit;  
 Label26: TLabel;  
 Label27: TLabel;  
 CitizenNumber: TEdit;  
 CellPhone: TEdit;  
 Label28: TLabel;  
 Label29: TLabel;  
 OtherPhone: TEdit;  
 Label30: TLabel;  
 RentGroup: TGroupBox;  
 BuildingsCmb: TComboBox;  
 FloorsCmb: TComboBox;  
 RentDuration: TEdit;  
 Label5: TLabel;  
 Label6: TLabel;  
 RoomsCmb: TComboBox;  
 Label2: TLabel;  
 Label7: TLabel;  
 Label1: TLabel;  
 MyListQuery: TMyQuery;  
 GenderCmb: TComboBox;  
 BirthDatePicker: TDateTimePicker;  
 BloodGroupCmb: TComboBox;  
 ContactAddress: TMemo;  
 Notes: TMemo;  
 MothersAddress: TMemo;  
 FathersAddress: TMemo;  
 CustomerName: TEdit;  
 CutomerSurname: TEdit;  
 Label8: TLabel;

```

RentStatusCmb: TComboBox;
RvProject: TRvProject;
RvDataSetConnection1: TRvDataSetConnection;
PrintBtn: TSpeedButton;
Bevel1: TBevel;
procedure FormShow(Sender: TObject);
procedure ClearBtnClick(Sender: TObject);
procedure RefreshBtnClick(Sender: TObject);
procedure BuildingsCmbChange(Sender: TObject);
procedure FloorsCmbChange(Sender: TObject);
procedure CloseBtnClick(Sender: TObject);
procedure AddBtnClick(Sender: TObject);
procedure EditBtnClick(Sender: TObject);
procedure CancelBtnClick(Sender: TObject);
procedure SaveBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure PrintBtnClick(Sender: TObject);
procedure DBGridTitleClick(Column: TColumn);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);

```

```

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

```

```

var
  CustomersForm: TCustomersForm;
  SortingField, SortingMethod: String;
  PreviousColumnIndex : Integer = 0;

```

```

implementation

```

```

{$R *.dfm}

```

```

procedure TCustomersForm.FormShow(Sender: TObject);
begin
  SortingField := 'building_name, floor_name, room_name';
  SortingMethod := '';
  SaveBtn.Visible := False;
  CancelBtn.Visible := False;
  ClearBtn.Click;
  RefreshBtn.Click;
end;

```

```

procedure TCustomersForm.ClearBtnClick(Sender: TObject);
begin
  CustomerName.Text := '';

```

```

CustomerSurname.Text := "";
GenderCmb.ItemIndex := -1;
PlaceOfBirth.Text := "";
BloodGroupCmb.ItemIndex := -1;
CitizenNumber.Text := "";
CellPhone.Text := "";
OtherPhone.Text := "";
ContactAddress.Text := "";

```

```

FathersName.Text := "";
FathersJob.Text := "";
FathersPhone.Text := "";
FathersAddress.Text := "";

```

```

MothersName.Text := "";
MothersJob.Text := "";
MothersPhone.Text := "";
MothersAddress.Text := "";

```

```

BuildingsCmb.ItemIndex := -1;
FloorsCmb.Enabled := False;
FloorsCmb.ItemIndex := -1;
RoomsCmb.Enabled := False;
RoomsCmb.ItemIndex := -1;
RentStatusCmb.ItemIndex := -1;
RentDuration.Text := "";
Notes.Text := "";
end;

```

```

procedure TCustomersForm.RefreshBtnClick(Sender: TObject);
begin
  BuildingsCmb.Clear;
  With MyQuery Do
  Begin
    Close;
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select * From buildings Order By building_name';
    Execute;
    While Not Eof Do
    Begin
      BuildingsCmb.Items.AddObject(FieldByName('building_name').AsString,
        TObject(FieldByName('id').AsInteger));
    Next;
  End;
End;
FloorsCmb.Enabled := False;
FloorsCmb.ItemIndex := -1;
RoomsCmb.Enabled := False;

```



```

RoomsCmb.ItemIndex := -1;
With MyListQuery Do
Begin
  Close;
  SQL.Clear;
  Params.Clear;
  SQL.Text := 'Select customers.*,'+
  ' Coalesce(room_name,"") room_name,'+
  ' Coalesce(floor_name,"") floor_name,'+
  ' Coalesce(building_name,"") building_name From customers'+
  ' Left Join rooms On (customers.room_id=rooms.id)+'
  ' Left Join floors On (floors.id=rooms.floor_id)+'
  ' Left Join buildings On (buildings.id=floors.building_id)+'
  ' Order By '+SortingField+SortingMethod;
  Execute;
End;
end;

procedure TCustomersForm.BuildingsCmbChange(Sender: TObject);
begin
  RoomsCmb.Enabled := False;
  RoomsCmb.ItemIndex := -1;
  FloorsCmb.Clear;
  With MyQuery Do
  Begin
    Close;
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select * From floors Where building_id=:building_id '+
    'Order By floor_name';
    Params[0].AsString :=
    IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
    Execute;
    While Not Eof Do
    Begin
      FloorsCmb.Items.AddObject(FieldByName('floor_name').AsString,
      TObject(FieldByName('id').AsInteger));
    End;
  End;
  FloorsCmb.Enabled := True;
end;

procedure TCustomersForm.FloorsCmbChange(Sender: TObject);
begin
  RoomsCmb.Clear;
  With MyQuery Do
  Begin
    Close;

```

```

SQL.Clear;
Params.Clear;
SQL.Text := 'Select * From rooms Where floor_id=:floor_id '+
'Order By room_name';
Params[0].AsString :=
IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]));
Execute;
While Not Eof Do
Begin
RoomsCmb.Items.AddObject(FieldByName('room_name').AsString,
TObject(FieldByName('id').AsInteger));
Next;
End;
End;
RoomsCmb.Enabled := True;
end;

```

```

procedure TCustomersForm.CloseBtnClick(Sender: TObject);
begin
CustomersForm.Close;
end;

```

```

procedure TCustomersForm.AddBtnClick(Sender: TObject);
Var
BuildingID, FloorID, RoomID: String;
begin
CustomerName.Text := Trim(CustomerName.Text);
CutomerSurname.Text := Trim(CutomerSurname.Text);

```

```

PlaceOfBirth.Text := Trim(PlaceOfBirth.Text);
CitizenNumber.Text := Trim(CitizenNumber.Text);
CellPhone.Text := Trim(CellPhone.Text);
OtherPhone.Text := Trim(OtherPhone.Text);
ContactAddress.Text := Trim(ContactAddress.Text);

```

```

FathersName.Text := Trim(FathersName.Text);
FathersJob.Text := Trim(FathersJob.Text);
FathersPhone.Text := Trim(FathersPhone.Text);
FathersAddress.Text := Trim(FathersAddress.Text);

```

```

MothersName.Text := Trim(MothersName.Text);
MothersJob.Text := Trim(MothersJob.Text);
MothersPhone.Text := Trim(MothersPhone.Text);
MothersAddress.Text := Trim(MothersAddress.Text);

```

```

RentDuration.Text := Trim(RentDuration.Text);
Notes.Text := Trim(Notes.Text);

```

```

If (CustomerName.Text = "") Then

```

```

Begin
  MessageDlg('You must enter a valid Customer Name to save this record.', mtError,
mbOk], 0);
  Exit;
End;
If (CutomerSurname.Text = "") Then
Begin
  MessageDlg('You must enter a valid Customer Surname to save this record.',
mtError, [mbOk], 0);
  Exit;
End;
If (GenderCmb.ItemIndex = -1) Then
Begin
  MessageDlg('You must select Customer Gender to save this record.', mtError,
[mbOk], 0);
  Exit;
End;

If (BuildingsCmb.ItemIndex > -1) Then
  BuildingID :=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]))
Else BuildingID := '0';
If (FloorsCmb.ItemIndex > -1) Then
  FloorID := IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]))
Else FloorID := '0';
If (RoomsCmb.ItemIndex > -1) Then
  RoomID := IntToStr(LongInt(RoomsCmb.Items.Objects[RoomsCmb.ItemIndex]))
Else RoomID := '0';

With MyQuery Do
Begin
  SQL.Clear;
  Params.Clear;
  SQL.Text := 'Select count(id) From customers Where '+
'customer_name=:customer_name And customer_surname=:customer_surname';
  Params[0].AsString := CustomerName.Text;
  Params[1].AsString := CutomerSurname.Text;
  Execute;
  If (Fields[0].AsInteger > 0 ) Then
  Begin
    MessageDlg('This customer already exist.', mtError, [mbOk], 0);
    Exit;
  End;
End;

With MyQuery Do
Begin
  SQL.Clear;
  Params.Clear;

```



```

SQL.Text := 'Insert Into customers Set '+
'customer_name=:customer_name, '+
'customer_surname=:customer_surname, '+
'customer_gender=:customer_gender, '+
'customer_birthdate=:customer_birthdate, '+
'customer_place_of_birth=:customer_place_of_birth, '+
'customer_blood_group=:customer_blood_group, '+
'customer_citizen_number=:customer_citizen_number, '+
'customer_cellphone=:customer_cellphone, '+
'customer_otherphone=:customer_otherphone, '+
'customer_contact_address=:customer_contact_address, '+
'fathers_name=:fathers_name, '+
'fathers_job=:fathers_job, '+
'fathers_phone=:fathers_phone, '+
'fathers_address=:fathers_address, '+
'mothers_name=:mothers_name, '+
'mothers_job=:mothers_job, '+
'mothers_phone=:mothers_phone, '+
'mothers_address=:mothers_address, '+
'room_id=:room_id, '+
'rent_status=:rent_status, '+
'rent_duration=:rent_duration, '+
'notes=:notes';
Params[0].AsString := CustomerName.Text;
Params[1].AsString := CustomerSurname.Text;
If GenderCmb.ItemIndex = 0 Then Params[2].AsString := 'M'
Else Params[2].AsString := 'F';
Params[3].AsString := FormatDateTime('yyyy-MM-dd', BirthDatePicker.Date);
Params[4].AsString := PlaceOfBirth.Text;
Params[5].AsString := BloodGroupCmb.Text;
Params[6].AsString := CitizenNumber.Text;
Params[7].AsString := CellPhone.Text;
Params[8].AsString := OtherPhone.Text;
Params[9].AsString := ContactAddress.Text;
Params[10].AsString := FathersName.Text;
Params[11].AsString := FathersJob.Text;
Params[12].AsString := FathersPhone.Text;
Params[13].AsString := FathersAddress.Text;
Params[14].AsString := MothersName.Text;
Params[15].AsString := MothersJob.Text;
Params[16].AsString := MothersPhone.Text;
Params[17].AsString := MothersAddress.Text;
Params[18].AsString := RoomID;
Params[19].AsString := RentStatusCmb.Text;
Params[20].AsString := RentDuration.Text;
Params[21].AsString := Notes.Text;
Execute;
End;
MyListQuery.Execute;

```

```
RefreshBtn.Click;  
end;
```

```
procedure TCustomersForm.EditBtnClick(Sender: TObject);
```

```
Var
```

```
I: Integer;
```

```
begin
```

```
For I := 0 To BuildingsCmb.Items.Count-1 Do
```

```
Begin
```

```
  If BuildingsCmb.Items.Strings[I] =
```

```
  MyListQuery.FieldByName('building_name').AsString
```

```
  Then
```

```
  Begin
```

```
    BuildingsCmb.ItemIndex := I;
```

```
    BuildingsCmb.OnChange(Sender);
```

```
  End;
```

```
End;
```

```
For I := 0 To FloorsCmb.Items.Count-1 Do
```

```
Begin
```

```
  If FloorsCmb.Items.Strings[I] = MyListQuery.FieldByName('floor_name').AsString
```

```
  Then
```

```
  Begin
```

```
    FloorsCmb.ItemIndex := I;
```

```
    FloorsCmb.OnChange(Sender);
```

```
  End;
```

```
End;
```

```
For I := 0 To RoomsCmb.Items.Count-1 Do
```

```
Begin
```

```
  If RoomsCmb.Items.Strings[I] = MyListQuery.FieldByName('room_name').AsString
```

```
  Then RoomsCmb.ItemIndex := I;
```

```
End;
```

```
For I := 0 To BloodGroupCmb.Items.Count-1 Do
```

```
Begin
```

```
  If BloodGroupCmb.Items.Strings[I] =
```

```
  MyListQuery.FieldByName('customer_blood_group').AsString
```

```
  Then BloodGroupCmb.ItemIndex := I;
```

```
End;
```

```
For I := 0 To RentStatusCmb.Items.Count-1 Do
```

```
Begin
```

```
  If RentStatusCmb.Items.Strings[I] =
```

```
  MyListQuery.FieldByName('rent_status').AsString
```

```
  Then RentStatusCmb.ItemIndex := I;
```

```
End;
```

```

DBGrid.SetFocus;
DBGrid.Enabled := False;

CustomerName.Text := MyListQuery.FieldByName('customer_name').AsString;
CustomerSurname.Text := MyListQuery.FieldByName('customer_surname').AsString;
If MyListQuery.FieldByName('customer_gender').AsString = 'M' Then
GenderCmb.ItemIndex := 0
Else GenderCmb.ItemIndex := 1;

BirthDatePicker.Date :=
StrToDate(MyListQuery.FieldByName('customer_birthdate').AsString);
PlaceOfBirth.Text :=
MyListQuery.FieldByName('customer_place_of_birth').AsString;

CitizenNumber.Text :=
MyListQuery.FieldByName('customer_citizen_number').AsString;
CellPhone.Text := MyListQuery.FieldByName('customer_cellphone').AsString;
OtherPhone.Text := MyListQuery.FieldByName('customer_otherphone').AsString;
ContactAddress.Text :=
MyListQuery.FieldByName('customer_contact_address').AsString;

FathersName.Text := MyListQuery.FieldByName('fathers_name').AsString;
FathersJob.Text := MyListQuery.FieldByName('fathers_job').AsString;
FathersPhone.Text := MyListQuery.FieldByName('fathers_phone').AsString;
FathersAddress.Text := MyListQuery.FieldByName('fathers_address').AsString;

MothersName.Text := MyListQuery.FieldByName('mothers_name').AsString;
MothersJob.Text := MyListQuery.FieldByName('mothers_job').AsString;
MothersPhone.Text := MyListQuery.FieldByName('mothers_phone').AsString;
MothersAddress.Text := MyListQuery.FieldByName('mothers_address').AsString;

RentDuration.Text := MyListQuery.FieldByName('rent_duration').AsString;
Notes.Text := MyListQuery.FieldByName('notes').AsString;

CustomerName.SetFocus;
CustomerName.SelectAll;
EditBtn.Visible := False;
CloseBtn.Visible := False;
CancelBtn.Visible := True;
SaveBtn.Visible := True;
AddBtn.Enabled := False;
DeleteBtn.Enabled := False;
RefreshBtn.Enabled := False;
end;

procedure TCustomersForm.CancelBtnClick(Sender: TObject);
begin
    SaveBtn.Visible := False;

```



```

EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;
DBGrid.Enabled := True;
DBGrid.SetFocus;
ClearBtn.Click;
end;

```

```

procedure TCustomersForm.SaveBtnClick(Sender: TObject);

```

```

Var

```

```

    BuildingID, FloorID, RoomID, CustomerID, BirthDate: String;

```

```

begin

```

```

    CustomerName.Text := Trim(CustomerName.Text);

```

```

    CustomerSurname.Text := Trim(CustomerSurname.Text);

```

```

    PlaceOfBirth.Text := Trim(PlaceOfBirth.Text);

```

```

    CitizenNumber.Text := Trim(CitizenNumber.Text);

```

```

    CellPhone.Text := Trim(CellPhone.Text);

```

```

    OtherPhone.Text := Trim(OtherPhone.Text);

```

```

    ContactAddress.Text := Trim(ContactAddress.Text);

```

```

    FathersName.Text := Trim(FathersName.Text);

```

```

    FathersJob.Text := Trim(FathersJob.Text);

```

```

    FathersPhone.Text := Trim(FathersPhone.Text);

```

```

    FathersAddress.Text := Trim(FathersAddress.Text);

```

```

    MothersName.Text := Trim(MothersName.Text);

```

```

    MothersJob.Text := Trim(MothersJob.Text);

```

```

    MothersPhone.Text := Trim(MothersPhone.Text);

```

```

    MothersAddress.Text := Trim(MothersAddress.Text);

```

```

    RentDuration.Text := Trim(RentDuration.Text);

```

```

    Notes.Text := Trim(Notes.Text);

```

```

    If (CustomerName.Text = "") Then

```

```

    Begin

```

```

        MessageDlg('You must enter a valid Customer Name to save this record.', mtError,
[mbOk], 0);

```

```

        Exit;

```

```

    End;

```

```

    If (CustomerSurname.Text = "") Then

```

```

    Begin

```

```

        MessageDlg('You must enter a valid Customer Surname to save this record.',
mtError, [mbOk], 0);

```

```

        Exit;

```

```

    End;

```

```

If (GenderCmb.ItemIndex = -1) Then
Begin
    MessageDlg('You must select Customer Gender to save this record.', mtError,
[mbOk], 0);
    Exit;
End;

If (BuildingsCmb.ItemIndex > -1) Then
    BuildingID :=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]))
Else BuildingID := '0';
If (FloorsCmb.ItemIndex > -1) Then
    FloorID := IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]))
Else FloorID := '0';
If (RoomsCmb.ItemIndex > -1) Then
    RoomID := IntToStr(LongInt(RoomsCmb.Items.Objects[RoomsCmb.ItemIndex]))
Else RoomID := '0';
CustomerID := MyListQuery.FieldByName('id').AsString;
With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select count(id) From customers Where '+
'customer_name=:customer_name And customer_surname=:customer_surname And
id!:=:customer_id';
    Params[0].AsString := CustomerName.Text;
    Params[1].AsString := CustomerSurname.Text;
    Params[2].AsString := CustomerID;
    Execute;
    If (Fields[0].AsInteger > 0 ) Then
    Begin
        MessageDlg('This customer already exist.', mtError, [mbOk], 0);
        Exit;
    End;
End;

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Update customers Set '+
'customer_name=:customer_name, '+
'customer_surname=:customer_surname, '+
'customer_gender=:customer_gender, '+
'customer_birthdate=:customer_birthdate, '+
'customer_place_of_birth=:customer_place_of_birth, '+
'customer_blood_group=:customer_blood_group, '+
'customer_citizen_number=:customer_citizen_number, '+
'customer_cellphone=:customer_cellphone, '+

```

```

'customer_otherphone=:customer_otherphone, '+
'customer_contact_address=:customer_contact_address, '+
'fathers_name=:fathers_name, '+
'fathers_job=:fathers_job, '+
'fathers_phone=:fathers_phone, '+
'fathers_address=:fathers_address, '+
'mothers_name=:mothers_name, '+
'mothers_job=:mothers_job, '+
'mothers_phone=:mothers_phone, '+
'mothers_address=:mothers_address, '+
'room_id=:room_id, '+
'rent_status=:rent_status, '+
'rent_duration=:rent_duration, '+
'notes=:notes '+
'Where id=:customer_id';
Params[0].AsString := CustomerName.Text;
Params[1].AsString := CustomerSurname.Text;
If GenderCmb.ItemIndex = 0 Then Params[2].AsString := 'M'
Else Params[2].AsString := 'F';
Params[3].AsString := FormatDateTime('yyyy-MM-dd', BirthDatePicker.Date);
Params[4].AsString := PlaceOfBirth.Text;
Params[5].AsString := BloodGroupCmb.Text;
Params[6].AsString := CitizenNumber.Text;
Params[7].AsString := CellPhone.Text;
Params[8].AsString := OtherPhone.Text;
Params[9].AsString := ContactAddress.Text;
Params[10].AsString := FathersName.Text;
Params[11].AsString := FathersJob.Text;
Params[12].AsString := FathersPhone.Text;
Params[13].AsString := FathersAddress.Text;
Params[14].AsString := MothersName.Text;
Params[15].AsString := MothersJob.Text;
Params[16].AsString := MothersPhone.Text;
Params[17].AsString := MothersAddress.Text;
Params[18].AsString := RoomID;
Params[19].AsString := RentStatusCmb.Text;
Params[20].AsString := RentDuration.Text;
Params[21].AsString := Notes.Text;
Params[22].AsString := CustomerID;
Notes.Text := DateToStr(BirthDatePicker.Date);
Execute;
End;
SaveBtn.Visible := False;
EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;

```



```

DBGrid.Enabled := True;
RefreshBtn.Click;
MyListQuery.Execute;
end;

```

```

procedure TCustomersForm.DeleteBtnClick(Sender: TObject);
var
  CustomerID, CustomerName: String;
begin
  With MyQuery Do
  Begin
    CustomerID := MyListQuery.FieldByName('id').AsString;
    CustomerName := MyListQuery.FieldByName('customer_name').AsString+' '+
      MyListQuery.FieldByName('customer_surname').AsString;
    If MessageDlg('Record will be deleted. Please Confirm. #10#10' Customer :
      CustomerName, mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then
    Begin
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Delete From customers Where id=:customer_id';
      Params[0].AsString := CustomerID;
      Execute;
      MyListQuery.Execute;
    End;
  End;
end;

```

```

procedure TCustomersForm.PrintBtnClick(Sender: TObject);
begin
  RvProject.Open;
  RvProject.Execute;
  RvProject.Close;
end;

```

```

procedure TCustomersForm.DBGridTitleClick(Column: TColumn);
begin
  If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
    SortingMethod := "
  Else
    If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';
  SortingField := DBGrid.Columns[Column.Index].FieldName;
  Try
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
      DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
  Except
  End;
end;

```

```

Column.title.Font.Style := Column.title.Font.Style + [fsBold];
PreviousColumnIndex := Column.Index;
RefreshBtn.Click;
end;

procedure TCustomersForm.DBGridMouseMove(Sender: TObject;
Shift: TShiftState; X, Y: Integer);
var
pt: TGridcoord;
begin
pt:= DBGrid.MouseCoord(x, y);
If pt.y=0 Then
    DBGrid.Cursor:=crHandPoint
Else
    DBGrid.Cursor:=crDefault;
end;

procedure TCustomersForm.FormClose(Sender: TObject;
var Action: TCloseAction);
begin
DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

end.

unit Products;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Spin, StdCtrls, ExtCtrls, Buttons, Grids, DBGrids, DB, MemDS,
DBAccess, MyAccess, Mask, RpCon, RpConDS, RpDefine, RpRave;

type
TProductsForm = class(TForm)
    DBGrid: TDBGrid;
    CommandPanel: TPanel;
    ClearBtn: TSpeedButton;
    PrintClearBevel: TBevel;
    CloseBevel: TBevel;
    CancelBtn: TSpeedButton;
    CloseBtn: TSpeedButton;
    EditSaveBevel: TBevel;
    SaveBtn: TSpeedButton;
    EditBtn: TSpeedButton;
    AddBevel: TBevel;
    DeleteBevel: TBevel;

```

```

AddBtn: TSpeedButton;
DeleteBtn: TSpeedButton;
RefreshBevel: TBevel;
RefreshBtn: TSpeedButton;
PrintBtn: TSpeedButton;
GroupBox1: TGroupBox;
Label1: TLabel;
ProductName: TEdit;
Label2: TLabel;
ProductQuantity: TSpinEdit;
Label3: TLabel;
Label4: TLabel;
ProductBarcode: TEdit;
MyQuery: TMyQuery;
MyListQuery: TMyQuery;
DataSource: TDataSource;
Label5: TLabel;
ProductPrice: TEdit;
RvProjectProduct: TRvProject;
RvDataSetConnectionProduct: TRvDataSetConnection;
procedure FormShow(Sender: TObject);
procedure ClearBtnClick(Sender: TObject);
procedure RefreshBtnClick(Sender: TObject);
procedure AddBtnClick(Sender: TObject);
procedure EditBtnClick(Sender: TObject);
procedure CancelBtnClick(Sender: TObject);
procedure SaveBtnClick(Sender: TObject);
procedure CloseBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure PrintBtnClick(Sender: TObject);
procedure DBGridTitleClick(Column: TColumn);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    ProductsForm: TProductsForm;
    SortingField, SortingMethod: String;
    PreviousColumnIndex : Integer = 0;

implementation

{$R *.dfm}

```



```

procedure TProductsForm.FormShow(Sender: TObject);
begin
  SortingField := 'product_name';
  SortingMethod := '';
  SaveBtn.Visible := False;
  CancelBtn.Visible := False;
  ClearBtn.Visible := False;
  ClearBtn.Click;
  RefreshBtn.Click;
end;

```

```

procedure TProductsForm.ClearBtnClick(Sender: TObject);
begin
  ProductName.Text := '';
  ProductPrice.Text := '0,00';
  ProductBarcode.Text := '';
  ProductQuantity.Text := '0';
end;

```

```

procedure TProductsForm.RefreshBtnClick(Sender: TObject);
begin
  With MyListQuery Do
    Begin
      Close;
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Select * From products'+
        ' Order By '+SortingField+SortingMethod;
      Execute;
    End;
end;

```

```

procedure TProductsForm.CloseBtnClick(Sender: TObject);
begin
  ProductsForm.Close;
end;

```

```

procedure TProductsForm.AddBtnClick(Sender: TObject);
Var
  Product_Price: Double;
begin
  ProductName.Text := Trim(ProductName.Text);
  ProductPrice.Text := Trim(ProductPrice.Text);
  ProductBarcode.Text := Trim(ProductBarcode.Text);

  If (ProductName.Text = '') Then
    Begin
      MessageDlg('You must enter a valid Product Name to save this record.', mtError,
        [mbOk], 0);
    End;
  End;
end;

```

```

Exit;
End;
Try
    Product_Price := StrToFloat(ProductPrice.Text);
Except
    On EConvertError Do
    Begin
        MessageDlg('You must enter a numeric Product Price to save this record.', mtError,
[mbOk], 0);
        Exit;
    End;
End;
If (Product_Price < 0) Then
Begin
    MessageDlg('You must enter a positive Product Price to save this record.', mtError,
[mbOk], 0);
    Exit;
End;
With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Insert Into products Set '+
'product_name=:product_name, '+
'product_quantity=:product_quantity, '+
'product_price=:product_price, '+
'product_barcode=:product_barcode';
    Params[0].AsString := ProductName.Text;
    Params[1].AsString := ProductQuantity.Text;
    Params[2].AsFloat := Product_Price;
    Params[3].AsString := ProductBarcode.Text;
    Execute;
End;
MyListQuery.Execute;
ClearBtn.Click;
RefreshBtn.Click;
end;

```

```

procedure TProductsForm.EditBtnClick(Sender: TObject);

```

```

begin

```

```

    DBGrid.SetFocus;
    DBGrid.Enabled := False;

```

```

    ProductName.Text := MyListQuery.FieldByName('product_name').AsString;
    ProductQuantity.Text := MyListQuery.FieldByName('product_quantity').AsString;
    ProductPrice.Text := MyListQuery.FieldByName('product_price').AsString;
    ProductBarcode.Text := MyListQuery.FieldByName('product_barcode').AsString;

```

```

    ProductName.SetFocus;

```

```

ProductName.SelectAll;
EditBtn.Visible := False;
CloseBtn.Visible := False;
CancelBtn.Visible := True;
SaveBtn.Visible := True;
AddBtn.Enabled := False;
DeleteBtn.Enabled := False;
RefreshBtn.Enabled := False;
end;

```

```

procedure TProductsForm.CancelBtnClick(Sender: TObject);
begin
    SaveBtn.Visible := False;
    EditBtn.Visible := True;
    CancelBtn.Visible := False;
    CloseBtn.Visible := True;
    AddBtn.Enabled := True;
    DeleteBtn.Enabled := True;
    RefreshBtn.Enabled := True;
    DBGrid.Enabled := True;
    DBGrid.SetFocus;
    ClearBtn.Click;
end;

```

```

procedure TProductsForm.SaveBtnClick(Sender: TObject);
Var
    Product_Price: Double;
begin
    ProductName.Text := Trim(ProductName.Text);
    ProductPrice.Text := Trim(ProductPrice.Text);
    ProductBarcode.Text := Trim(ProductBarcode.Text);

    If (ProductName.Text = "") Then
        Begin
            MessageDlg('You must enter a valid Product Name to save this record.', mtError,
[mbOk], 0);
            Exit;
        End;
    Try
        Product_Price := StrToFloat(ProductPrice.Text);
    Except
        On EConvertError Do
            Begin
                MessageDlg('You must enter a numeric Product Price to save this record.', mtError,
[mbOk], 0);
                Exit;
            End;
        End;
    End;
    If (Product_Price < 0) Then

```



```

Begin
    MessageDlg('You must enter a positive Product Price to save this record.', mtError,
[mbOk], 0);
    Exit;
End;

```

With MyQuery Do

```

Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Update products Set '+
'product_name=:product_name, '+
'product_quantity=:product_quantity, '+
'product_price=:product_price, '+
'product_barcode=:product_barcode '+
'Where id=:product_id';
    Params[0].AsString := ProductName.Text;
    Params[1].AsString := ProductQuantity.Text;
    Params[2].AsFloat := Product_Price;
    Params[3].AsString := ProductBarcode.Text;
    Params[4].AsString := MyListQuery.FieldName('id').AsString;
    Execute;
End;
SaveBtn.Visible := False;
EditBtn.Visible := True;
CancelBtn.Visible := False;
CloseBtn.Visible := True;
AddBtn.Enabled := True;
DeleteBtn.Enabled := True;
RefreshBtn.Enabled := True;
DBGrid.Enabled := True;
MyListQuery.Execute;
ClearBtn.Click;
RefreshBtn.Click;
end;

```

procedure TProductsForm.DeleteBtnClick(Sender: TObject);

```

Var
    ProductInfo: String;
begin
    ProductInfo := MyListQuery.FieldName('Product_name').AsString;
    With MyQuery Do
    Begin
        If MessageDlg('Record will be deleted. Please Confirm. '#10#10' Product :
'+ProductInfo, mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then
        Begin
            SQL.Clear;
            Params.Clear;
            SQL.Text := 'Delete From products Where id=:product_id';

```

```

Params[0].AsString := MyListQuery.FieldByName('id').AsString;
Execute;
MyListQuery.Execute;
End;
End;
end;

procedure TProductsForm.PrintBtnClick(Sender: TObject);
begin
  RvProjectProduct.Open;
  RvProjectProduct.Execute;
  RvProjectProduct.Close;
end;

procedure TProductsForm.DBGridTitleClick(Column: TColumn);
begin
  If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
    SortingMethod := "
  Else
    If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';
  SortingField := DBGrid.Columns[Column.Index].FieldName;
  Try
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
  Except
  End;
  Column.title.Font.Style := Column.title.Font.Style + [fsBold];
  PreviousColumnIndex := Column.Index;
  RefreshBtn.Click;
end;

procedure TProductsForm.DBGridMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
var
  pt: TGridCoord;
begin
  pt:= DBGrid.MouseCoord(x, y);
  If pt.y=0 Then
    DBGrid.Cursor:=crHandPoint
  Else
    DBGrid.Cursor:=crDefault;
end;

procedure TProductsForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
  DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

```

end.

unit Sales;

interface

uses  
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Spin, ExtCtrls, Buttons, Grids, DBGrids, DB, MemDS,  
DBAccess, MyAccess, DateUtils, RpCon, RpConDS, RpDefine, RpRave;

type

```
TSalesForm = class(TForm)
  DBGrid: TDBGrid;
  CommandPanel: TPanel;
  ClearBtn: TSpeedButton;
  PrintClearBevel: TBevel;
  CloseBevel: TBevel;
  CloseBtn: TSpeedButton;
  AddBevel: TBevel;
  DeleteBevel: TBevel;
  AddBtn: TSpeedButton;
  DeleteBtn: TSpeedButton;
  RefreshBevel: TBevel;
  RefreshBtn: TSpeedButton;
  PrintBtn: TSpeedButton;
  GroupBox1: TGroupBox;
  Label1: TLabel;
  ProductNameCmb: TComboBox;
  CustomerInfoGroup: TGroupBox;
  BuildingsLbl: TLabel;
  FloorsLbl: TLabel;
  RoomsLbl: TLabel;
  CustomersLbl: TLabel;
  BuildingsCmb: TComboBox;
  FloorsCmb: TComboBox;
  RoomsCmb: TComboBox;
  CustomersCmb: TComboBox;
  Label5: TLabel;
  SaleQuantity: TSpinEdit;
  ShowVisitsLbl: TLabel;
  ShowSalesCmb: TComboBox;
  Bevel3: TBevel;
  MyQuery: TMyQuery;
  MyListQuery: TMyQuery;
  DataSource: TDataSource;
  Label6: TLabel;
  SaleTotal: TLabel;
```



```

ProductPrice: TLabel;
Label3: TLabel;
ProductQuantity: TLabel;
Label2: TLabel;
ProductBarcode: TLabel;
Label4: TLabel;
RvProjectSales: TRvProject;
RvDataSetConnectionSales: TRvDataSetConnection;
procedure FormShow(Sender: TObject);
procedure ClearBtnClick(Sender: TObject);
procedure RefreshBtnClick(Sender: TObject);
procedure BuildingsCmbChange(Sender: TObject);
procedure FloorsCmbChange(Sender: TObject);
procedure RoomsCmbChange(Sender: TObject);
procedure CloseBtnClick(Sender: TObject);
procedure ProductNameCmbChange(Sender: TObject);
procedure SaleQuantityChange(Sender: TObject);
procedure AddBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure ShowSalesCmbChange(Sender: TObject);
procedure PrintBtnClick(Sender: TObject);
procedure DBGridTitleClick(Column: TColumn);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  SalesForm: TSalesForm;
  SortingField, SortingMethod, SaleDateRange: String;
  PreviousColumnIndex : Integer = 0;

implementation

{$R *.dfm}

procedure TSalesForm.FormShow(Sender: TObject);
begin
  SortingField := 'sale_time';
  SortingMethod := ' Desc';
  SaleDateRange := ' sale_time Between "' + FormatDateTime('yyyy-MM-dd 00:00:00',
Date)+
  "' And "' + FormatDateTime('yyyy-MM-dd 23:59:59', Date) + '"';
  ClearBtn.Visible := False;
  ShowSalesCmb.ItemIndex := 0;

```

```
RefreshBtn.Click;  
end;
```

```
procedure TSalesForm.ClearBtnClick(Sender: TObject);  
begin  
  ProductNameCmb.ItemIndex := -1;  
  SaleQuantity.Text := '1';  
  ProductPrice.Caption := " ";  
  ProductQuantity.Caption := " ";  
  ProductBarcode.Caption := " ";  
  SaleTotal.Caption := " ";  
  BuildingsCmb.ItemIndex := -1;  
  FloorsCmb.Enabled := False;  
  FloorsCmb.ItemIndex := -1;  
  RoomsCmb.Enabled := False;  
  RoomsCmb.ItemIndex := -1;  
  CustomersCmb.Enabled := False;  
  CustomersCmb.ItemIndex := -1;  
end;
```

```
procedure TSalesForm.RefreshBtnClick(Sender: TObject);  
begin  
  ProductNameCmb.Clear;  
  With MyQuery Do  
  Begin  
    Close;  
    SQL.Clear;  
    Params.Clear;  
    SQL.Text := 'Select * From products Order By product_name';  
    Execute;  
    While Not Eof Do  
    Begin  
      ProductNameCmb.Items.AddObject(FieldByName('product_name').AsString,  
                                      TObject(FieldByName('id').AsInteger));  
    Next;  
  End;  
End;  
BuildingsCmb.Clear;  
With MyQuery Do  
Begin  
  Close;  
  SQL.Clear;  
  Params.Clear;  
  SQL.Text := 'Select * From buildings Order By building_name';  
  Execute;  
  While Not Eof Do  
  Begin  
    BuildingsCmb.Items.AddObject(FieldByName('building_name').AsString,  
                                  TObject(FieldByName('id').AsInteger));  
  End;  
End;
```

```

Next;
End;
End;
FloorsCmb.Enabled := False;
FloorsCmb.ItemIndex := -1;
RoomsCmb.Enabled := False;
RoomsCmb.ItemIndex := -1;
CustomersCmb.Enabled := False;
CustomersCmb.ItemIndex := -1;

```

```

With MyListQuery Do
Begin
Close;
SQL.Clear;
Params.Clear;
SQL.Text := 'Select * From sales Where'+SaleDateRange+
' Order By '+SortingField+SortingMethod;
Execute;
End;

```

```

ClearBtn.Click;
end;

```

```

procedure TSalesForm.ProductNameCmbChange(Sender: TObject);
begin
With MyQuery Do
Begin
Close;
SQL.Clear;
Params.Clear;
SQL.Text := 'Select * From products Where id=:product_id';
Params[0].AsString :=
IntToStr(LongInt(ProductNameCmb.Items.Objects[ProductNameCmb.ItemIndex]));
Execute;
ProductPrice.Caption := FieldByName('product_price').AsString;
ProductQuantity.Caption := FieldByName('product_quantity').AsString;
SaleQuantity.MaxValue := FieldByName('product_quantity').AsInteger;
If StrToInt(SaleQuantity.Text) > FieldByName('product_quantity').AsInteger Then
SaleQuantity.Text := FieldByName('product_quantity').AsString;
ProductBarcode.Caption := FieldByName('product_barcode').AsString;
End;
SaleTotal.Caption :=
FloatToStr((StrToInt(SaleQuantity.Text)*StrToFloat(ProductPrice.Caption)));
end;

```

```

procedure TSalesForm.BuildingsCmbChange(Sender: TObject);
begin
RoomsCmb.Enabled := False;
RoomsCmb.ItemIndex := -1;

```



```

FloorsCmb.Clear;
With MyQuery Do
Begin
  Close;
  SQL.Clear;
  Params.Clear;
  SQL.Text := 'Select * From floors Where building_id=:building_id '+
  'Order By floor_name';
  Params[0].AsString :=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
  Execute;
  While Not Eof Do
  Begin
    FloorsCmb.Items.AddObject(FieldByName('floor_name').AsString,
      TObject(FieldByName('id').AsInteger));

    Next;
  End;
End;
FloorsCmb.Enabled := True;
end;

```

```

procedure TSalesForm.FloorsCmbChange(Sender: TObject);
begin
  RoomsCmb.Clear;
  With MyQuery Do
  Begin
    Close;
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Select * From rooms Where floor_id=:floor_id '+
    'Order By room_name';
    Params[0].AsString :=
IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]));
    Execute;
    While Not Eof Do
    Begin
      RoomsCmb.Items.AddObject(FieldByName('room_name').AsString,
        TObject(FieldByName('id').AsInteger));

      Next;
    End;
  End;
  RoomsCmb.Enabled := True;
end;

```

```

procedure TSalesForm.RoomsCmbChange(Sender: TObject);
begin
  CustomersCmb.Clear;
  With MyQuery Do
  Begin

```

```

Close;
SQL.Clear;
Params.Clear;
SQL.Text := 'Select * From customers Where room_id=:room_id '+
'Order By customer_name';
Params[0].AsString :=
IntToStr(LongInt(RoomsCmb.Items.Objects[RoomsCmb.ItemIndex]));
Execute;
While Not Eof Do
Begin
CustomersCmb.Items.AddObject(FieldByName('customer_name').AsString+'
'+FieldByName('customer_surname').AsString,
TObject(FieldByName('id').AsInteger));

Next;
End;
End;
CustomersCmb.Enabled := True;
end;

procedure TSalesForm.SaleQuantityChange(Sender: TObject);
begin
If ProductNameCmb.ItemIndex > -1 Then
SaleTotal.Caption :=
FloatToStr((StrToInt(SaleQuantity.Text)*StrToFloat(ProductPrice.Caption)));
end;

procedure TSalesForm.CloseBtnClick(Sender: TObject);
begin
SalesForm.Close;
end;

procedure TSalesForm.AddBtnClick(Sender: TObject);
begin
If (ProductNameCmb.ItemIndex = -1) Then
Begin
MessageDlg('You must select a Product Name to save this record.', mtError, [mbOk],
0);
Exit;
End;
If (BuildingsCmb.ItemIndex <> -1) Then
Begin
If (FloorsCmb.ItemIndex = -1) Then
Begin
MessageDlg('You must select a Floor Name to save this record.', mtError, [mbOk],
0);
Exit;
End;
If (RoomsCmb.ItemIndex = -1) Then
Begin

```

```

    MessageDlg('You must select a Room Name to save this record.', mtError, [mbOk],
0);
    Exit;
End;
If (CustomersCmb.ItemIndex = -1) Then
Begin
    MessageDlg('You must select a Customer Name to save this record.', mtError,
[mbOk], 0);
    Exit;
End;
End;

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Insert Into sales Set '+
'customer_info=:customer_info, '+
'product_info=:product_info, '+
'product_price=:product_price, '+
'sale_quantity=:sale_quantity, '+
'total_price=:total_price, '+
'sale_time=:sale_time';
    If BuildingsCmb.Text <> " " Then
        Params[0].AsString := BuildingsCmb.Text+' > '+FloorsCmb.Text+' >
'+RoomsCmb.Text
    Else
        Params[0].AsString := 'Not From Dormitory';
    If CustomersCmb.ItemIndex > -1 Then Params[0].AsString := Params[0].AsString+'
> '+CustomersCmb.Text;
    Params[1].AsString := ProductNameCmb.Text;
    Params[2].AsFloat := StrToFloat(ProductPrice.Caption);
    Params[3].AsString := SaleQuantity.Text;
    Params[4].AsFloat := StrToFloat(SaleTotal.Caption);
    Params[5].AsString := FormatDateTime('yyyy-MM-dd', Date)+FormatDateTime('
hh:mm:ss', Time);
    Execute;
End;
With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Update products Set '+
'product_quantity=product_quantity-:sale_quantity '+
'Where id=:product_id';
    Params[0].AsString := SaleQuantity.Text;
    Params[1].AsString :=
IntToStr(LongInt(ProductNameCmb.Items.Objects[ProductNameCmb.ItemIndex]));
    Execute;

```



```

End;
RefreshBtn.Click;
end;

procedure TSalesForm.DeleteBtnClick(Sender: TObject);
Var
  SaleID, SaleInfo, SaleQuantity: String;
begin
  With MyQuery Do
  Begin
    SaleID := MyListQuery.FieldName('id').AsString;
    SaleInfo := MyListQuery.FieldName('sale_time').AsString+' - '+
      MyListQuery.FieldName('product_info').AsString;
    SaleQuantity := MyListQuery.FieldName('sale_quantity').AsString;
    If MessageDlg('Record will be deleted. Please Confirm.#10#10'Sale Info :
'+SaleInfo+
      #10#10'You should manually increase this product's quantity by '+SaleQuantity,
    mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then
    Begin
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Delete From sales Where id=:sale_id';
      Params[0].AsString := SaleID;
      Execute;
      MyListQuery.Execute;
    End;
  End;
end;

procedure TSalesForm.ShowSalesCmbChange(Sender: TObject);
begin
  Case ShowSalesCmb.ItemIndex Of
    0: SaleDateRange := ' sale_time Between '"+FormatDateTime('yyyy-MM-dd
00:00:00', Date)+'
    '" And '"+FormatDateTime('yyyy-MM-dd 23:59:59', Date)+'"';
    1: SaleDateRange := ' sale_time Between '"+FormatDateTime('yyyy-MM-dd
00:00:00', Yesterday)+'
    '" And '"+FormatDateTime('yyyy-MM-dd 23:59:59', Yesterday)+'"';
    2: SaleDateRange := ' sale_time Between '"+FormatDateTime('yyyy-MM-01
00:00:00', Date)+'
    '" And '"+FormatDateTime('yyyy-MM-31 23:59:59', Date)+'"';
    3: SaleDateRange := ' sale_time Between '"+FormatDateTime('yyyy-01-01 00:00:00',
Date)+'
    '" And '"+FormatDateTime('yyyy-12-31 23:59:59', Date)+'"';
    4: SaleDateRange := ' sale_time Between "0000-01-01 00:00:00" And "9999-12-31
23:59:59"';
  end;
  RefreshBtn.Click;
end;

```

```

procedure TSalesForm.PrintBtnClick(Sender: TObject);
begin
    RvProjectSales.Open;
    RvProjectSales.Execute;
    RvProjectSales.Close;
end;

```

```

procedure TSalesForm.DBGridTitleClick(Column: TColumn);
begin
    If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
        SortingMethod := "
    Else
        If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';
        SortingField := DBGrid.Columns[Column.Index].FieldName;
    Try
        DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
            DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
    Except
    End;
    Column.title.Font.Style := Column.title.Font.Style + [fsBold];
    PreviousColumnIndex := Column.Index;
    RefreshBtn.Click;
end;

```

```

procedure TSalesForm.DBGridMouseMove(Sender: TObject; Shift: TShiftState;
    X, Y: Integer);
var
    pt: TGridcoord;
begin
    pt:= DBGrid.MouseCoord(x, y);
    If pt.y=0 Then
        DBGrid.Cursor:=crHandPoint
    Else
        DBGrid.Cursor:=crDefault;
end;

```

```

procedure TSalesForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
        DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

```

end.

unit Visitors;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Buttons, ComCtrls, StdCtrls, DB, MemDS, DBAccess, MyAccess,  
Grids, DBGrids, ExtCtrls, DateUtils, RpCon, RpConDS, RpDefine, RpRave;

type

```
TVisitorsForm = class(TForm)
  CustomerInfoGroup: TGroupBox;
  BuildingsLbl: TLabel;
  FloorsLbl: TLabel;
  RoomsLbl: TLabel;
  CustomersLbl: TLabel;
  BuildingsCmb: TComboBox;
  FloorsCmb: TComboBox;
  RoomsCmb: TComboBox;
  CustomersCmb: TComboBox;
  GroupBox1: TGroupBox;
  VisitorLbl: TLabel;
  EnteringLbl: TLabel;
  EnteringTimePicker: TDateTimePicker;
  GetCurrentTimeBtn: TSpeedButton;
  ExitingLbl: TLabel;
  ExitingTimePicker: TDateTimePicker;
  Bevel1: TBevel;
  VisitorInfo: TMemo;
  CommandPanel: TPanel;
  PrintClearBevel: TBevel;
  CloseBevel: TBevel;
  CancelBtn: TSpeedButton;
  CloseBtn: TSpeedButton;
  EditSaveBevel: TBevel;
  SaveBtn: TSpeedButton;
  EditBtn: TSpeedButton;
  AddBevel: TBevel;
  DeleteBevel: TBevel;
  AddBtn: TSpeedButton;
  DeleteBtn: TSpeedButton;
  RefreshBevel: TBevel;
  RefreshBtn: TSpeedButton;
  PrintBtn: TSpeedButton;
  DBGrid: TDBGrid;
  MyQuery: TMyQuery;
  MyListQuery: TMyQuery;
  DataSource: TDataSource;
  ShowVisitsCmb: TComboBox;
  ShowVisitsLbl: TLabel;
  Bevel3: TBevel;
  ClearBtn: TSpeedButton;
  RvProjectVisitor: TRvProject;
```



```

RvDataSetConnectionVisitor: TRvDataSetConnection;
procedure FormShow(Sender: TObject);
procedure ClearBtnClick(Sender: TObject);
procedure RefreshBtnClick(Sender: TObject);
procedure BuildingsCmbChange(Sender: TObject);
procedure FloorsCmbChange(Sender: TObject);
procedure CloseBtnClick(Sender: TObject);
procedure RoomsCmbChange(Sender: TObject);
procedure GetCurrentTimeBtnClick(Sender: TObject);
procedure AddBtnClick(Sender: TObject);
procedure EditBtnClick(Sender: TObject);
procedure CancelBtnClick(Sender: TObject);
procedure SaveBtnClick(Sender: TObject);
procedure DeleteBtnClick(Sender: TObject);
procedure ShowVisitsCmbChange(Sender: TObject);
procedure PrintBtnClick(Sender: TObject);
procedure DBGridTitleClick(Column: TColumn);
procedure DBGridMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  VisitorsForm: TVisitorsForm;
  SortingField, SortingMethod, VisitDateRange: String;
  PreviousColumnIndex : Integer = 0;

implementation

{$R *.dfm}

procedure TVisitorsForm.FormShow(Sender: TObject);
begin
  SortingField := 'entering_time';
  SortingMethod := ' Desc';
  VisitDateRange := ' entering_time Between "' + FormatDateTime('yyyy-MM-dd
00:00:00', Date) +
  '" And "' + FormatDateTime('yyyy-MM-dd 23:59:59', Date) + '"';
  SaveBtn.Visible := False;
  CancelBtn.Visible := False;
  ClearBtn.Visible := False;
  ClearBtn.Click;
  RefreshBtn.Click;
end;

```

```

procedure TVisitorsForm.ClearBtnClick(Sender: TObject);
begin
  BuildingsCmb.ItemIndex := -1;
  FloorsCmb.Enabled := False;
  FloorsCmb.ItemIndex := -1;
  RoomsCmb.Enabled := False;
  RoomsCmb.ItemIndex := -1;
  CustomersCmb.Enabled := False;
  CustomersCmb.ItemIndex := -1;
  ShowVisitsCmb.ItemIndex := 0;
  VisitorInfo.Text := '';
  EnteringTimePicker.Date := Date;
  EnteringTimePicker.Time := Time;
  ExitingLbl.Enabled := False;
  ExitingTimePicker.Time := StrToTime('00:00:00');
  ExitingTimePicker.Enabled := False;
end;

```

```

procedure TVisitorsForm.RefreshBtnClick(Sender: TObject);
begin
  BuildingsCmb.Clear;
  With MyQuery Do
    Begin
      Close;
      SQL.Clear;
      Params.Clear;
      SQL.Text := 'Select * From buildings Order By building_name';
      Execute;
      While Not Eof Do
        Begin
          BuildingsCmb.Items.AddObject(FieldByName('building_name').AsString,
            TObject(FieldByName('id').AsInteger));
          Next;
        End;
      End;
      FloorsCmb.Enabled := False;
      FloorsCmb.ItemIndex := -1;
      RoomsCmb.Enabled := False;
      RoomsCmb.ItemIndex := -1;
      CustomersCmb.Enabled := False;
      CustomersCmb.ItemIndex := -1;

      With MyListQuery Do
        Begin
          Close;
          SQL.Clear;
          Params.Clear;
          SQL.Text := 'Select * From visitors Where'+VisitDateRange+
            ' Order By '+SortingField+SortingMethod;

```

```

    Execute;
End;
end;

procedure TVisitorsForm.BuildingsCmbChange(Sender: TObject);
begin
    RoomsCmb.Enabled := False;
    RoomsCmb.ItemIndex := -1;
    FloorsCmb.Clear;
    With MyQuery Do
    Begin
        Close;
        SQL.Clear;
        Params.Clear;
        SQL.Text := 'Select * From floors Where building_id=:building_id '+'
        'Order By floor_name';
        Params[0].AsString :=
IntToStr(LongInt(BuildingsCmb.Items.Objects[BuildingsCmb.ItemIndex]));
        Execute;
        While Not Eof Do
        Begin
            FloorsCmb.Items.AddObject(FieldByName('floor_name').AsString,
                TObject(FieldByName('id').AsInteger));

            Next;
        End;
    End;
    FloorsCmb.Enabled := True;
end;

```

```

procedure TVisitorsForm.FloorsCmbChange(Sender: TObject);
begin
    RoomsCmb.Clear;
    With MyQuery Do
    Begin
        Close;
        SQL.Clear;
        Params.Clear;
        SQL.Text := 'Select * From rooms Where floor_id=:floor_id '+'
        'Order By room_name';
        Params[0].AsString :=
IntToStr(LongInt(FloorsCmb.Items.Objects[FloorsCmb.ItemIndex]));
        Execute;
        While Not Eof Do
        Begin
            RoomsCmb.Items.AddObject(FieldByName('room_name').AsString,
                TObject(FieldByName('id').AsInteger));

            Next;
        End;
    End;
End;

```



```

RoomsCmb.Enabled := True;
end;

procedure TVisitorsForm.RoomsCmbChange(Sender: TObject);
begin
    CustomersCmb.Clear;
    With MyQuery Do
        Begin
            Close;
            SQL.Clear;
            Params.Clear;
            SQL.Text := 'Select * From customers Where room_id=:room_id '+'
            'Order By customer_name';
            Params[0].AsString :=
            IntToStr(LongInt(RoomsCmb.Items.Objects[RoomsCmb.ItemIndex]));
            Execute;
            While Not Eof Do
                Begin
                    CustomersCmb.Items.AddObject(FieldByName('customer_name').AsString+'
                    '+FieldByName('customer_surname').AsString,
                    TObject(FieldByName('id').AsInteger));
                Next;
            End;
        End;
    CustomersCmb.Enabled := True;
end;

procedure TVisitorsForm.CloseBtnClick(Sender: TObject);
begin
    VisitorsForm.Close;
end;

procedure TVisitorsForm.GetCurrentTimeBtnClick(Sender: TObject);
begin
    If EnteringTimePicker.Enabled Then EnteringTimePicker.Time := Time
    Else ExitingTimePicker.Time := Time;
end;

procedure TVisitorsForm.AddBtnClick(Sender: TObject);
begin
    VisitorInfo.Text := Trim(VisitorInfo.Text);
    If (BuildingsCmb.ItemIndex = -1) Then
        Begin
            MessageDlg('You must select a Building Name to save this record.', mtError,
            [mbOk], 0);
            Exit;
        End;
    If (FloorsCmb.ItemIndex = -1) Then
        Begin

```

```

    MsgBox('You must select a Floor Name to save this record.', mtError, [mbOk],
0);
    Exit;
End;
If (RoomsCmb.ItemIndex = -1) Then
Begin
    MsgBox('You must select a Room Name to save this record.', mtError, [mbOk],
0);
    Exit;
End;
If (VisitorInfo.Text = "") Then
Begin
    MsgBox('You must enter valid Visitor(s) Information to save this record.',
mtError, [mbOk], 0);
    Exit;
End;

With MyQuery Do
Begin
    SQL.Clear;
    Params.Clear;
    SQL.Text := 'Insert Into visitors Set '+
'visited_customer_info=:visited_customer_info, '+
'visitor_info=:visitor_info, '+
'entering_time=:entering_time';

    Params[0].AsString := BuildingsCmb.Text+' > '+FloorsCmb.Text+' >
'+RoomsCmb.Text;
    If CustomersCmb.ItemIndex > -1 Then Params[0].AsString := Params[0].AsString+'
> '+CustomersCmb.Text;
    Params[1].AsString := VisitorInfo.Text;
    Params[2].AsString := FormatDateTime('yyyy-MM-dd hh:mm:ss',
EnteringTimePicker.Date);
    Execute;
End;
VisitorInfo.Text := "";
RefreshBtn.Click;
end;

procedure TVisitorsForm.EditBtnClick(Sender: TObject);
Var
    I: Integer;
begin
    DBGrid.SetFocus;
    DBGrid.Enabled := False;

    VisitorInfo.Text := MyListQuery.FieldName('visitor_info').AsString;
    EnteringTimePicker.Time :=
StrToDate(MyListQuery.FieldName('entering_time').AsString);

```

```
ExitingTimePicker.Date := Date;  
ExitingTimePicker.Time := Time;
```

```
BuildingsLbl.Enabled := False;  
BuildingsCmb.Enabled := False;  
FloorsLbl.Enabled := False;  
FloorsCmb.Enabled := False;  
RoomsLbl.Enabled := False;  
RoomsCmb.Enabled := False;  
CustomersLbl.Enabled := False;  
CustomersCmb.Enabled := False;  
ShowVisitsLbl.Enabled := False;  
ShowVisitsCmb.Enabled := False;  
VisitorLbl.Enabled := False;  
VisitorInfo.Enabled := False;  
EnteringLbl.Enabled := False;  
EnteringTimePicker.Enabled := False;  
ExitingLbl.Enabled := True;  
ExitingTimePicker.Enabled := True;
```

```
EditBtn.Visible := False;  
CloseBtn.Visible := False;  
CancelBtn.Visible := True;  
SaveBtn.Visible := True;  
AddBtn.Enabled := False;  
DeleteBtn.Enabled := False;  
RefreshBtn.Enabled := False;
```

```
end;
```

```
procedure TVisitorsForm.CancelBtnClick(Sender: TObject);  
begin
```

```
BuildingsLbl.Enabled := True;  
BuildingsCmb.Enabled := True;  
FloorsLbl.Enabled := True;  
FloorsCmb.Enabled := True;  
RoomsLbl.Enabled := True;  
RoomsCmb.Enabled := True;  
CustomersLbl.Enabled := True;  
CustomersCmb.Enabled := True;  
ShowVisitsLbl.Enabled := True;  
ShowVisitsCmb.Enabled := True;  
VisitorLbl.Enabled := True;  
VisitorInfo.Enabled := True;  
ExitingLbl.Enabled := False;  
ExitingTimePicker.Enabled := False;  
EnteringLbl.Enabled := True;
```



```
EnteringTimePicker.Enabled := True;
```

```
SaveBtn.Visible := False;  
EditBtn.Visible := True;  
CancelBtn.Visible := False;  
CloseBtn.Visible := True;  
AddBtn.Enabled := True;  
DeleteBtn.Enabled := True;  
RefreshBtn.Enabled := True;  
DBGrid.Enabled := True;  
DBGrid.SetFocus;  
ClearBtn.Click;  
end;
```

```
procedure TVisitorsForm.SaveBtnClick(Sender: TObject);  
begin  
  With MyQuery Do  
  Begin  
    SQL.Clear;  
    Params.Clear;  
    SQL.Text := 'Update visitors Set exiting_time=:exiting_time Where id=:visit_id';  
    Params[0].AsString := FormatDateTime('yyyy-MM-dd hh:mm:ss',  
    ExitingTimePicker.Date);  
    Params[1].AsString := MyListQuery.FieldName('id').AsString;  
    Execute;  
  End;  
  CancelBtn.Click;  
  RefreshBtn.Click;  
  //MyListQuery.Execute;  
end;
```

```
procedure TVisitorsForm.DeleteBtnClick(Sender: TObject);  
Var  
  VisitID, VisitInfo: String;  
begin  
  With MyQuery Do  
  Begin  
    VisitID := MyListQuery.FieldName('id').AsString;  
    VisitInfo := MyListQuery.FieldName('entering_time').AsString+' - '+'  
      MyListQuery.FieldName('visitor_info').AsString;  
    If MessageDlg('Record will be deleted. Please Confirm. #10#10' Visit Info :  
    '+VisitInfo, mtConfirmation, [mbOK, mbCancel], 0) = mrOK Then  
    Begin  
      SQL.Clear;  
      Params.Clear;  
      SQL.Text := 'Delete From visitors Where id=:visit_id';  
      Params[0].AsString := VisitID;  
      Execute;  
      MyListQuery.Execute;
```

```

End;
End;
end;

procedure TVisitorsForm.ShowVisitsCmbChange(Sender: TObject);
begin
  Case ShowVisitsCmb.ItemIndex Of
    0: VisitDateRange := ' entering_time Between "' + FormatDateTime('yyyy-MM-dd
00:00:00', Date) +
    "" And "' + FormatDateTime('yyyy-MM-dd 23:59:59', Date) + ""';
    1: VisitDateRange := ' entering_time Between "' + FormatDateTime('yyyy-MM-dd
00:00:00', Yesterday) +
    "" And "' + FormatDateTime('yyyy-MM-dd 23:59:59', Yesterday) + ""';
    2: VisitDateRange := ' entering_time Between "' + FormatDateTime('yyyy-MM-01
00:00:00', Date) +
    "" And "' + FormatDateTime('yyyy-MM-31 23:59:59', Date) + ""';
    3: VisitDateRange := ' entering_time Between "' + FormatDateTime('yyyy-01-01
00:00:00', Date) +
    "" And "' + FormatDateTime('yyyy-12-31 23:59:59', Date) + ""';
    4: VisitDateRange := ' entering_time Between "0000-01-01 00:00:00" And "9999-12-
31 23:59:59"";
  end;
  RefreshBtn.Click;
end;

procedure TVisitorsForm.PrintBtnClick(Sender: TObject);
begin
  RvProjectVisitor.Open;
  RvProjectVisitor.Execute;
  RvProjectVisitor.Close;
end;

procedure TVisitorsForm.DBGridTitleClick(Column: TColumn);
begin
  If SortingField <> DBGrid.Columns[Column.Index].FieldName Then
    SortingMethod := "
  Else
    If SortingMethod <> " Then SortingMethod := " Else SortingMethod := ' Desc';
  SortingField := DBGrid.Columns[Column.Index].FieldName;
  Try
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
  Except
  End;
  Column.title.Font.Style := Column.title.Font.Style + [fsBold];
  PreviousColumnIndex := Column.Index;
  RefreshBtn.Click;
end;

```

```

procedure TVisitorsForm.DBGridMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);

```

```

var
  pt: TGridCoord;
begin
  pt:= DBGrid.MouseCoord(x, y);
  If pt.y=0 Then
    DBGrid.Cursor:=crHandPoint
  Else
    DBGrid.Cursor:=crDefault;
end;

```

```

procedure TVisitorsForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  DBGrid.Columns[PreviousColumnIndex].title.Font.Style :=
    DBGrid.Columns[PreviousColumnIndex].title.Font.Style - [fsBold];
end;

```

```

end.

```

```

unit About;

```

```

interface

```

```

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

```

```

type

```

```

  TAboutForm = class(TForm)

```

```

    Image1: TImage;

```

```

    Label1: TLabel;

```

```

    Label2: TLabel;

```

```

    Label3: TLabel;

```

```

    Label4: TLabel;

```

```

    Bevel1: TBevel;

```

```

    Label5: TLabel;

```

```

    Label6: TLabel;

```

```

    Bevel2: TBevel;

```

```

    Label7: TLabel;

```

```

    Label8: TLabel;

```

```

  private

```

```

    { Private declarations }

```

```

  public

```

```

    { Public declarations }

```

```

  end;

```

```

var

```