

**NEAR EAST UNIVERSITY**



**Faculty of Engineering**

**Department of Computer Engineering**

**TIME TABLE DESIGN:  
DELPHI APPLICATION**

**Graduation Project  
COM-400**

**Student: Mamun Ali Khamis (20010702)**

**Supervisor: Ass.Prof.Dr. Adil Amirjanov**

**Nicosia- 2005**



## ACKNOWLEDGMENT

*"First, I would like to and foremost to thank Allah whom its accomplishment would not have been possible*

*Second, I would like to deeply thank my supervisor Ass. Prof. Dr ADEL AMIRCANOV for his invaluable advice and belief in my work and my self over the course of this graduation project*

*Third I am deeply indebted to my parents, brothers, and sisters for their love and their support. They have always encouraged me to pursue my interests and ambition throughout life.*

*Last but in no way least, I would also like to thank all of my friends especially Emad Dahdoh and Rami Aljundi they were always available for my assistance throughout this project."*

## ABSTRACT

The purpose of this project is design of faculty's time table, the main structure and elements of database for this design are clarified, the operation principles of each blocks of the design are modeled in Delphi Programming language .

The design allows us to make easily addition, updating, and searching for employees (instructors and assistants), courses, rooms, and days. All of these will explain in chapter three.

This project has been designed in a way that it works speedier than the normal record design to decrease the time.

## TABLE OF CONTENTS

ACKNOWLEDGMENT .....	i
ABSTRACT .....	ii
TABLE OF CONTENTS .....	iii
INTRODUCTION.....	v
<b>CHAPTER 1 .....</b>	<b>1</b>
<b>INTEGRATED DEVELOPMENT ENVIRONMENT (IDE).....</b>	<b>1</b>
1.1 INTRODUCTION .....	1
1.2 The Delphi IDE.....	1
1.3 A Quick Look at the Delphi IDE .....	2
1.4 The Delphi Workspace .....	3
1.5 The Delphi Main Menu and Toolbar. ....	3
1.6 Using the Component Palette.....	5
1.7 Placing Multiple Copies of a Component.....	5
1.8 About Delphi Forms .....	6
1.8.1 Main Window Forms .....	6
1.8.2 Creating the Main Window Form .....	6
1.8.3 Dialog Box Forms.....	8
1.8.4 Creating a Dialog Form .....	9
1.9 A Multiple-Form Application .....	9
1.9.1 Adding Units .....	10
1.9.2 Some Key Properties for Forms.....	10
1.10 The Object Inspector.....	11
1.10.1 The Component Selector.....	11
1.10.2 The Properties Page .....	12
1.10.3 The Events Page.....	13
1.11 Code Templates .....	13
1.12. Writing Code for the Window Menu .....	14
<b>CHAPTER 2.....</b>	<b>15</b>
<b>(DATABASE ACCESS) .....</b>	<b>15</b>
2.1. Microsoft Access Description.....	15
2.2. Starting Microsoft Access.....	16
2.3. Creating New, and Opening Existing Databases .....	17
2.4. Create a database using the Database Wizard.....	18
2.5. Create a database without using the Database Wizard .....	18
2.6. Tables .....	19
2.7. Create a Table from scratch in Design view .....	20

2.8. Primary Key .....	22
2.9. Switching Views .....	22
2.10. Entering Data .....	23
2.11. Manipulating Data .....	23
2.12. Advanced Table Features w/Microsoft Access.....	24
2.12.1. Assigning a field a specific set of characters .....	24
2.12.2. Formatting a field to look a specific way (HINT) .....	24
2.13. Relationships.....	25
<b>CHAPTER 3.....</b>	<b>28</b>
<b>TIME TABLE DESIGN .....</b>	<b>28</b>
3.1. Structure of Time Table Design:.....	28
3.2. Define Relationships Between Forms:.....	28
3.3. Define Relationships Between Tables: .....	29
3.4. Delphi database components: .....	30
3.5. Layout of the Application: .....	30
3.5.1. Main menu screen: .....	30
3.5.2. Add New Employee Screen: .....	31
3.5.3. Update Employee Record Screen: .....	33
3.5.4. Add New Course Screen: .....	34
3.5.5. Update Courses Screen: .....	34
3.5.6. Add New Room Screen: .....	35
3.5.7. Time Table Screen: .....	37
<b>CONCLUSION .....</b>	<b>40</b>
<b>REFERENCES.....</b>	<b>41</b>
<b>APPENDIX.....</b>	<b>42</b>

## INTRODUCTION

The aim of the project is the design of faculty's time table using Delphi programming. The intended audience for this project includes the following:

1. Explanation of Delphi's IDE.
2. Explanation of Access Database.
3. Screens and its consistence and some of codes.

In this project I used one of the programming languages that we are learned in our university-Delphi Programming language. In this language there are many things that we can use to create any kind of project. But in this project I used some standard components to create my project, also I used access database to create tables, we will see this later, regarding this program which basically divided into tow sections, Addition and Updating for all fields which consist of employees, courses, rooms , and time which are confirm the time table. My project allows to the users to know which time, room, and day that have a lecture for any instructor or assistant. These are the main ideas about my project.

## CHAPTER 1

### INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

#### 1.1 INTRODUCTION

One of the most difficult aspects of learning how to use a new programming environment is finding your way around: getting to know the basic menu structure, what all the options do, and how the environment works as a whole.

#### 1.2 The Delphi IDE

**Definition:** Integrated Development Environment. This is the user interface (GUI) where you can design, compile and debug your Delphi projects.

So, without further ado, take a look at Figure 1.1 and let's get on with it. If you have used Delphi before, you might find this chapter elementary.

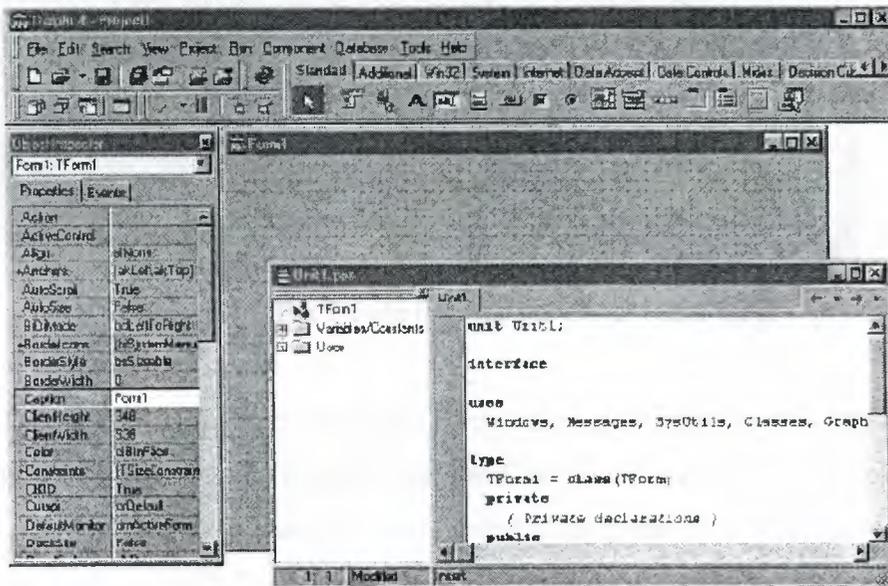


Figure 1.1 The Delphi IDE

1. The Delphi IDE consists of these main parts:
2. The main menu and toolbars
3. The Component palette
4. The Form Designer
5. The Code Editor

- 6. The Object Inspector 1
- 7. The Code Explorer 1
- 8. The Project Manage

### 1.3 A Quick Look at the Delphi IDE

This section contains a quick look at the Delphi integrated development environment (IDE). Because you are tackling Windows programming, I'll assume you are advanced enough to have figured out how to start Delphi. When you first start the program, you are presented with both a blank form and the IDE, as shown in Figure 1.2.[1]

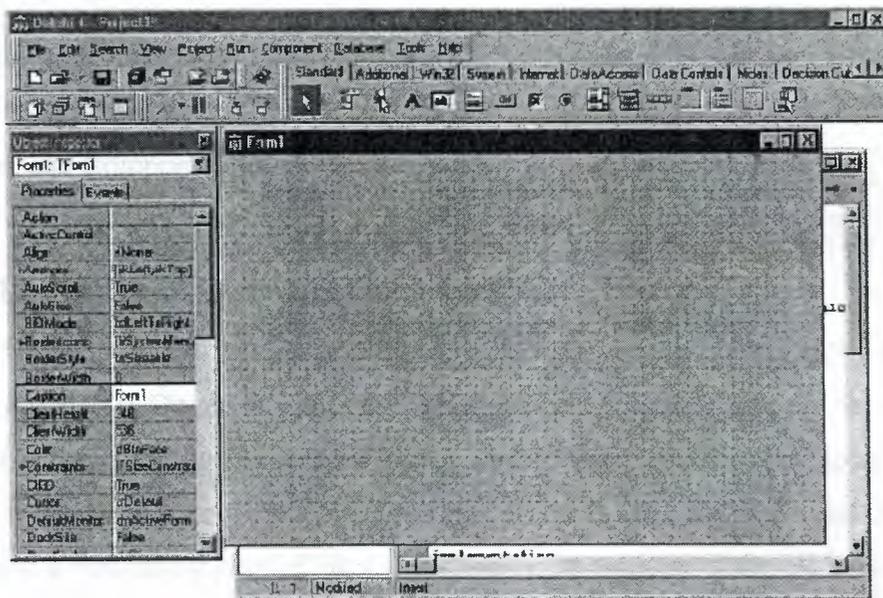


Figure 1.2 The Delphi IDE and the initial blank form

The Delphi IDE is divided into three parts. The top window can be considered the main window. It contains the toolbars and the Component palette. The Delphi toolbars give you one-click access to tasks such as opening, saving, and compiling projects.

The Component palette contains a wide array of components that you can drop onto your forms. (Components are text labels, edit controls, list boxes, buttons, and the like.) For convenience, the components are divided into groups. Go ahead and click on the tabs to explore the different components available to you. To place a component on your form, you simply click the component's button in the Component palette and then click

on your form where you want the component to appear. When you are done exploring, click on the tab labeled Standard, because you'll need it in a moment.

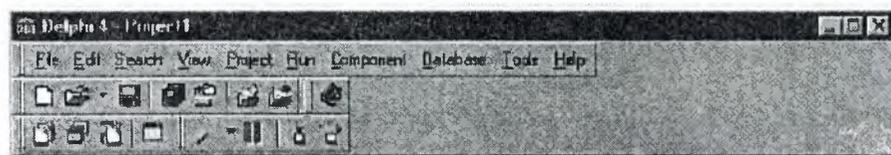
## **1.4 The Delphi Workspace**

The main part of the Delphi IDE is the workspace. The workspace initially displays the Form Designer. It should come as no surprise that the Form Designer enables you to create forms. In Delphi, a form represents a window in your program. The form might be the program's main window, a dialog box, or any other type of window. You use the Form Designer to place, move, and size components as part of the form creation process.

Hiding behind the Form Designer is the Code Editor. The Code Editor is where you type code when writing your programs. The Object Inspector, Form Designer, Code Editor, and Component palette work interactively as you build applications.

## **1.5 The Delphi Main Menu and Toolbar.**

The Delphi main menu has all the choices necessary to make Delphi work. Because programming in Delphi is a highly visual operation, you might not use the main menu as much as you might with other programming environments. Still, just about anything you need is available from the main menu if you prefer to work that way. The Delphi toolbars provide a convenient way of accomplishing often-repeated tasks. A button is easier to locate than a menu item, not to mention that it requires less mouse movement. The Delphi main window toolbars are illustrated in Figure 1.3.



**Figure 1.3** The Delphi main window

Delphi enables you to add buttons to the toolbar, remove buttons, and rearrange buttons however you see fit. To configure a toolbar, right-click on the toolbar to display the context menu. Choose Customize from the context menu. When you choose this menu item, the Customize dialog box is displayed.

The Customize dialog box contains three tabs:

The first tab, Toolbars, shows you the toolbars available with a check mark next to toolbars that are currently visible. You can add or remove existing toolbars or reset the toolbars to their original default settings.[1]

The second tab, labeled Commands, shows all the available toolbar buttons. To add a button to the toolbar, just locate its description in the Commands list box and drag it to the place you want it to occupy on any toolbar. To remove a button from a toolbar, grab it and drag it off the toolbar. It's as simple as that. Figure 1.4 shows the act of adding a button to a toolbar. If you really make a mess of things, simply go back to the Toolbars page and click the Reset button. The toolbar will revert to its default settings.

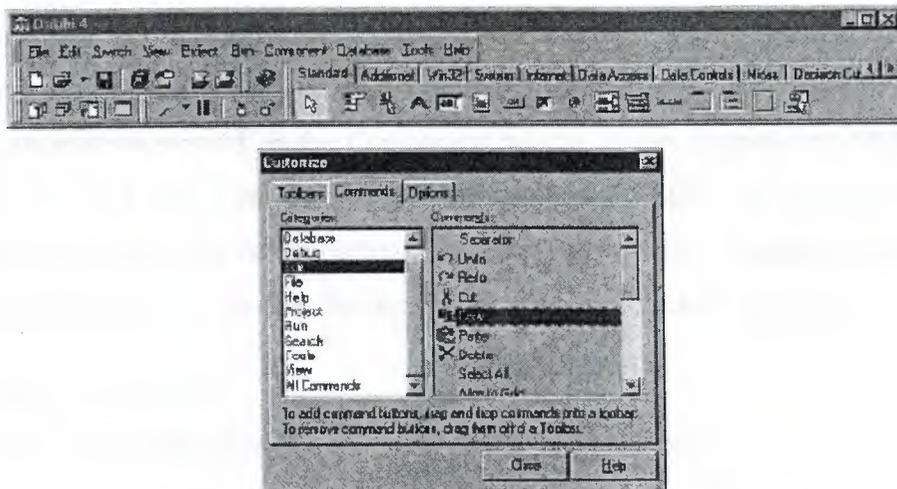


Figure 1.4 customizing the toolbar

The third tab, Options, contains options such as whether the tooltips are displayed and how they are displayed.

## **1.6 Using the Component Palette**

The Delphi Component palette is used to select a component or other control (such as an ActiveX control) in order to place that control on a form. The Component palette is a multipage window. Tabs are provided to enable you to navigate between pages.

Placing a component on a form is a two-step process. First, go to the Component palette and select the button representing the component you want to use.

Then click on the form to place the component on the form. The component appears with its upper-left corner placed where you clicked with the mouse.

## **1.7 Placing Multiple Copies of a Component**

So far you have placed only one component at a time on a form. You can easily place multiple components of the same type without selecting the component from the Component palette each time. To place multiple components on a form, press and hold the Shift key as you select the component from the Component palette. After you select the component, you can release the Shift key.

The component's button on the Component palette will appear pressed and will be highlighted with a blue border. Click on the form to place the first component. Notice that the button stays pressed on the Component palette. A new component will be placed each time you click the form. To stop placing components, click the selector button on the Component palette (the arrow button). The component button pops up to indicate that you are done placing components. Seeing is believing, so follow these steps:

1. Create a new project.
2. Press and hold the Shift key on the keyboard and click the Label component button in the Component palette.
3. Click three times on the form, moving the cursor each time to indicate where you want the new component placed. A new Label is placed on the form each time you click.
4. Click the arrow button on the Component palette to end the process and return to form design mode.

## **1.8 About Delphi Forms**

Before I continue with the discussion about the Delphi IDE, I need to spend some time explaining forms. You have seen several forms in action as you have worked through this chapter. You need some more background information on forms.

### **1.8.1 Main Window Forms**

Forms are the main building block of a Delphi application. Every GUI application has at least one form that serves as the main window. The main window form might be just a blank window, it might have controls on it, or it might have a bitmap displayed on it. In a typical Windows program, your main window would have a menu. It might also have decorations such as a toolbar or a status bar. Just about anything goes when creating the main window of your application. Each application is unique, and each has different requirements.

### **1.8.2 Creating the Main Window Form**

First you'll create the main window form. The main window for an MDI application must have the `FormStyle` property set to `fsMDIForm`. You also need to add a menu to the application, as well as File Open and File Save dialog boxes. Follow these steps:

1. Start Delphi and choose File | New Application from the main menu.
2. For the main form, change the Name property to `MainForm`.
3. Change the Caption property to `Picture Viewer`.
4. Change the Height to 450 and the Width to 575 (or other suitable values)
5. Change the `FormStyle` to `fsMDIForm`.

Now you've got the main part of the form done. Next you'll add a menu to the form, you will take the easy route to creating a menu. To do that, you can take advantage of a Delphi feature that enables you to import a predefined menu, as follows:

1. Click the Standard tab of the Component palette and click the MainMenu button.
2. Click on the form to place a MainMenu component on the form. It doesn't matter where you place the component because the icon representing the menu is just a placeholder and won't show on the form at runtime. This is how nonvisual components appear on a form.
3. Change the Name property to MainMenu.
4. Double-click the MainMenu component. The Menu Designer is displayed.
5. Place your cursor over the Menu Designer and click your right mouse button. Choose Insert from Template from the context menu. The Insert Template dialog box appears.
6. Choose MDI Frame Menu and click OK. The menu is displayed in the Menu Designer.
7. Click the system close box on the Menu Designer to close it.

Now you should be back to the main form. You can click on the top-level items to see the full menu. Don't click on any menu subitems at this point--you'll do that in a minute.

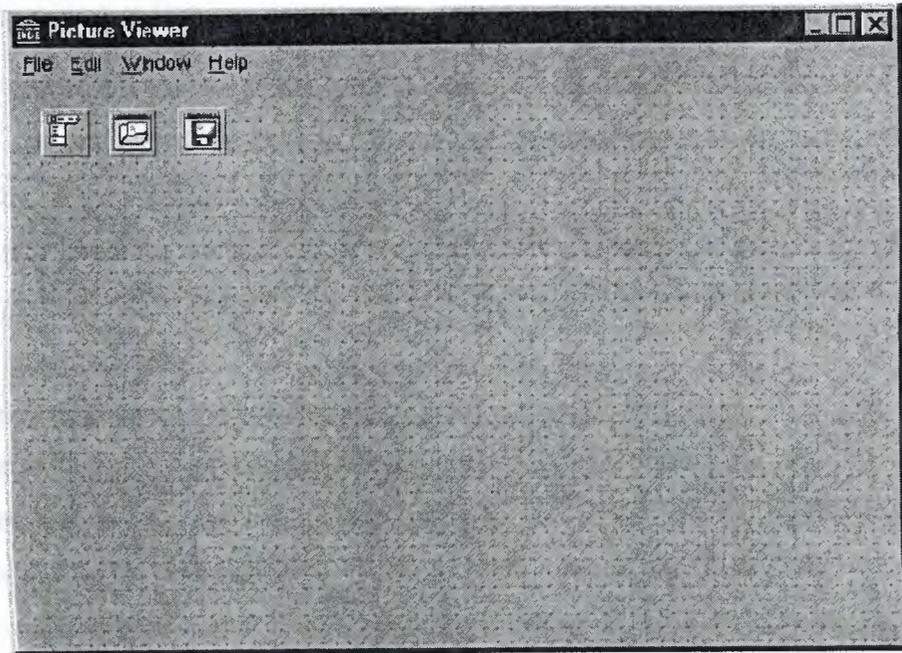
Now you need to prepare the File Open and File Save dialog boxes:

1. Click the Dialogs tab on the Component palette. Choose an Open Picture Dialog component and place it on the form.

The Open Picture Dialog component's icon can be placed anywhere on the form.

2. Change the Name property of the Open dialog box to Open Picture Dialog.
3. Change the Title property to open a Picture for Viewing.
4. Add a Save Picture Dialog component.
5. Change the Name property of the component to Save Picture Dialog and the Title property to Save a Picture.

Your form should now look like the one shown in Figure 1.5.



**Figure 1.5** Form designed

### **1.8.3 Dialog Box Forms**

Forms are also used for dialog boxes. In fact, to the user there is no difference between a Delphi form acting as a dialog box and a true dialog box. Dialog boxes usually have several traits that distinguish them from ordinary windows:

- Dialog boxes are not usually sizable. They usually perform a specific function, and sizing of the dialog box is neither useful nor desirable.
- Dialog boxes almost always have an OK button. Some dialog boxes have a button labeled Close that performs the same task. A simple dialog box such as an About dialog box typically has only the OK button. Dialog boxes can also have a Cancel button and a Help button.
- Dialog boxes typically have only the system close button on the title bar. They do not usually have minimize and maximize buttons.

## **13.4 Creating a Dialog Form**

First you'll add a button to the form that displays the about dialog box:

1. Bring the main form into view. Choose the Button component from the Component palette and drop a button on the form.
2. Arrange the two buttons that are now on the form to balance the look of the form.
3. Change the Name property of the new button to About Button and the Caption property to about.
4. Double-click the About Button you just created on the form. The Code Editor is displayed with the cursor placed in the event-handler function. Add this line of code at the cursor:

About Box Show Modal, You haven't actually created the About box yet, but when you do you'll name it About Box, so you know enough to type the code that will display the About box.

## **1.9 A Multiple-Form Application**

To illustrate how Delphi uses units, you can create an application with multiple forms. You'll create a simple application that displays a second form when you click a button:

1. Create a new project by choosing File | New Application from the main menu.
2. Change the Name property to MainForm and the Caption property to Multiple Forms Test Program.
3. Save the project. Save the unit as Main and the project as Multiple.
4. Now place a button on the form. Make the button's Name property ShowForm2 and the Caption property Show Form 2.
5. Choose File | New Form from the main menu to create a new form. At this point, the new form has a name of Form1 and is placed exactly over the main form. You want the new form to be smaller than the main form and more or less centered on the main form.
6. Size and position the new form so that it is about 50 percent of the size of the main form and centered on the main form. Use the title bar to move the new form. Size the form by dragging the lower-right corner.

7. Change the new form's Name property to SecondForm and the form's Caption property to A Second Form.
8. Choose File | Save from the main menu.
9. Choose a Label component and drop it on the new form.

### 1.9.1 Adding Units

Rather than having Delphi prompt you to add a unit to your uses list, you can add units yourself. You can manually type the unit name in the uses list for the form, or you can choose File | Use Unit from the main menu. When you choose the latter method, the Use Unit dialog box is displayed, as shown in Figure 1.6. The Use Unit dialog box shows a list of available units. Choose the unit you want to add and click OK. Delphi will add the unit to the current forms uses list.

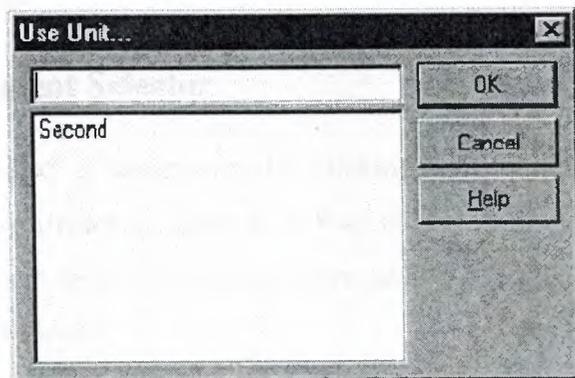


Figure 1.6 The Use Unit dialog box

### 1.9.2 Some Key Properties for Forms

The TForm class has a lot of properties. Some of these properties are obscure and rarely used others are widely used. I'll touch on the some widely used properties here.

- **Font** The Font property specifies the font that the form uses. The important issue to understand here is that the form's font is inherited by any components placed on the form. This also means that you can change the font used by all components at one time by changing just the form's font.

- **FormStyle** This property is usually set to fsNormal. If you want a form to always be on top, use the fsStayOnTop style. MDI forms should use the fsMDIForm style and MDI child forms should use the fsMDIChild style.

## **1.10 The Object Inspector**

An integral part of the Delphi IDE is the Object Inspector. This window works with the Form Designer to aid in the creation of components.

The Object Inspector is where you set the design-time properties that affect how the component acts at runtime. The Object Inspector has three main areas:

- The Component Selector
- The Properties page
- The Events page

### **1.10.1 The Component Selector**

Normally, you select a component by clicking the component on a form. The Component Selector provides an alternative way of selecting a component to view or modify. The Component Selector is a drop-down combo box that is located at the top of the Object Inspector window.

The Component Selector displays the name of the component and the class from which it is derived. For example, a memo component named Memo would appear in the Component Selector as

Memo: TMemo

The class name does not show up in the drop-down list of components, it only appears in the top portion of the Component Selector. To select a component, click the drop-down button to reveal the list of components and then click the one you want to select.

After you select a component in the Component Selector, the component is selected on the form as well. The Properties and Events tabs change to display the properties and events for the selected component. Figure 1.7 shows the Object Inspector with the Component Selector list displayed.

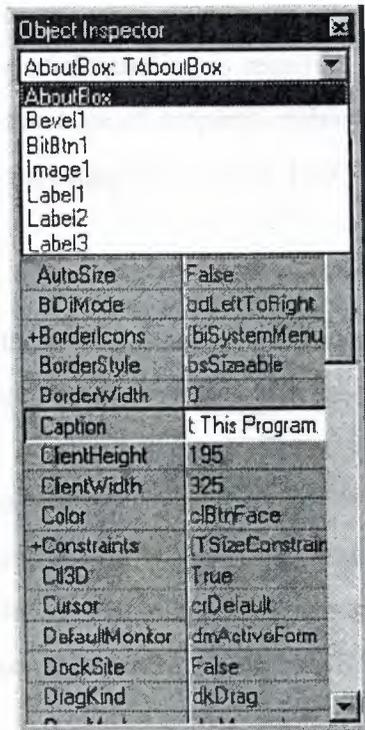


Figure 1.7 the component selector list

### 1.10.2 The Properties Page

The Properties page of the Object Inspector displays all the design-time properties for the currently selected control. The Properties page has two columns: The Property column is on the left side of the Properties page and shows the property name, the Value column is on the right side of the Properties page and is where you type or select the value for the property.

Properties can be integer values, enumerations, sets, other objects, strings, and other types.

The Object Inspector deals with each type of property according to the data type of the property. Delphi has several built-in property editors to handle data input for the property. For example, the Top property accepts an Integer value. Because the Integer type is a basic data type, no special handling is required, so the property editor is fairly basic. The property editor for this type of property enables you to type a value directly in the Value column for integer properties such as Top, Left, Width, and Height.

### **1.10.3 The Events Page**

The Events page lists all the events that the component is designed to handle. Using the Events page is pretty basic. Delphi creates an event-handling function for you with all the parameters needed to handle that event. The Code Editor is displayed and the cursor is placed in the event handler. All you have to do is start typing code. The name of the function is generated based on the Name property of the component and the event being handled. If, for example, you have a button named OKBtn and are handling the OnClick event, the function name generated would be OKBtnClick.

You can let Delphi generate the name of the event-handling function for you or you can provide the function name for Delphi to use. The Code Editor is displayed, and so is the event-handling function, complete with the name you supplied. After you create an event-handling function for a component, you can use that event handler for any component that handles the same event. Sometimes it's convenient to have several buttons use the same OnClick event.

### **1.11 Code Templates**

This feature lets you insert one of the predefined code templates, such as a complex statement with an inner begin...end block. Code templates must be activated manually, by pressing Ctrl+J to show a list of all of the templates. You can add custom code templates, so that you can build your own shortcuts for commonly used blocks of code.

For example, if you use the `MessageDlg` function often, you might want to add a template for it. To modify templates, go to the Source Options page of the Editor Options dialog box, select Pascal from the Source File Type list, and click the Edit Code Templates button. Doing so opens the new Delphi Code Templates dialog box. At this point, click the Add button, type in a new template name (for example, **mess**), type a description, and then add the following text to the template body in the Code memo control:

```
MessageDlg ('', mtInformation, [mbOK], 0);
```

Now, every time you need to create a message dialog box, you simply type **mess** and then press `Ctrl+J`, and you get the full text.

## **1.12. Writing Code for the Window Menu**

Now you can add code to the Window menu. This part is simple:

1. Switch back to the form by pressing `F12`. Choose `Window | Tile` from the form's menu.

2. You need to enter only a single line of code for the event handler. The finished event handler will look like this:

```
Procedure TMainForm.Tile1Click (Sender: TObject);  
Begin  
Tile;  
end;
```

3. Switch back to the form and repeat the process for `Window | Cascade`. The finished function looks like this:

```
Procedure TMainForm.Cascade1Click (Sender: TObject);  
Begin  
Cascade;  
end;
```

4. Repeat the steps for the `Window | Arrange All` menu item. The single line of code to add for the function body is the following:

```
ArrangeIcons.
```

## CHAPTER 2

### (DATABASE ACCESS)

#### 2.1. Microsoft Access Description

Microsoft Access is a powerful program to create and manage your databases. It has many built in features to assist you in constructing and viewing your information. Access is much more involved and is a more genuine database application than other programs such as Microsoft Works.

First of all you need to understand how Microsoft Access breaks down a database. Some keywords involved in this process are: *Database File, Table, Record, Field, and Data-type*. Here is the Hierarchy that Microsoft Access uses in breaking down a database

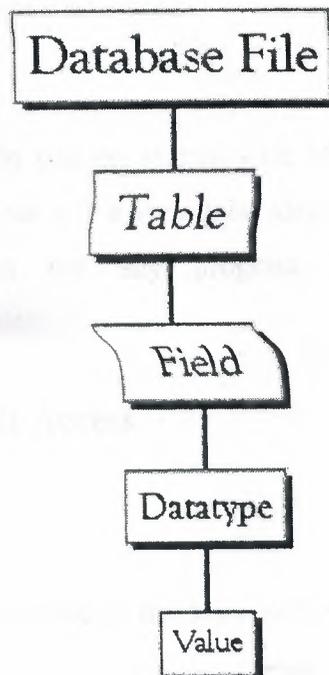


Figure 2.1. Microsoft Access uses in breaking down a database

- **Database File:** This is your main file that encompasses the entire database and that is saved to your hard-drive or floppy disk.

Example) StudentDatabase.mdb

- **Table:** A table is a collection of data about a specific topic. There can be multiple tables in a database.

Example #1) Students

Example #2) Teachers

- **Field:** Fields are the different categories within a Table. Tables usually contain multiple fields.

Example #1) Student Last Name

Example #2) Student First Name

- **Data types:** Data types are the properties of each field. A field only has 1 data type.

Field Name) Student Last Name

Data type) Text

This tutorial will help you get started with Microsoft Access and may solve some of your problems, but it is a very good idea to use the Help Files that come with Microsoft Access (or any program you use for that matter).

## 2.2. Starting Microsoft Access

- Two Ways

1. Double click on the Microsoft Access icon on the desktop.



Microsoft  
Access

2. Click on Start --> Programs --> Microsoft Access

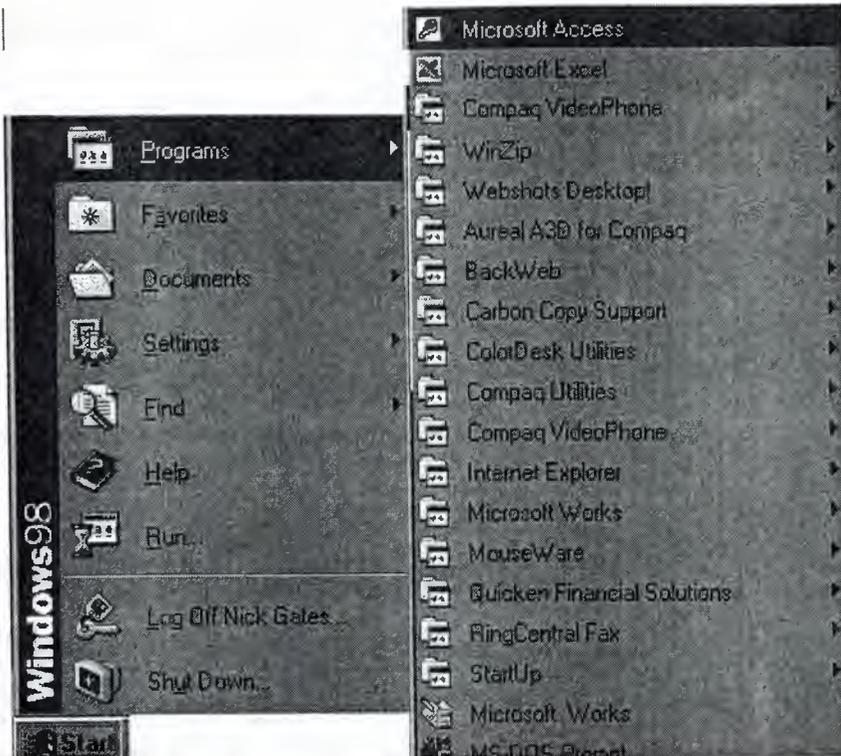


Figure 2.2 starting Microsoft access

### 2.3. Creating New, and Opening Existing Databases

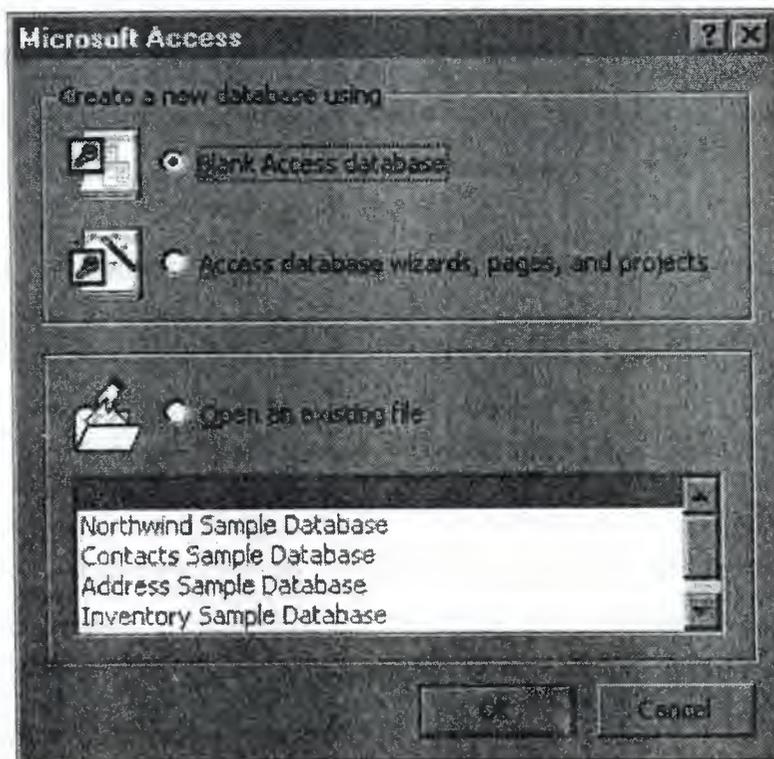


Figure 2.3 Creating New, and Opening Existing Databases

The above picture gives you the option to:

- Create a New Database from scratch.
- Use the wizard to create a New Database.
- Open an existing database

The white box gives you the most recent databases you have used. If you do not see the one you had created, choose the More Files option and hit OK. Otherwise choose the database you had previously used and click OK.

## 2.4. Create a database using the Database Wizard

1. When Microsoft Access first starts up, a dialog box is automatically displayed with options to create a new database or open an existing one. If this dialog box is displayed, click Access Database Wizards, pages, and projects and then click ok  
 if you have already opened a database or closed the dialog box that displays when Microsoft Access starts up, click *New Database* on the toolbar.
2. On the *Databases* tab, double-click the icon for the kind of database you want to create.
3. Specify a name and location for the database.
4. Click *Create* to start defining your new database

## 2.5. Create a database without using the Database Wizard

1. When Microsoft Access first starts up, a dialog box is automatically displayed with options to create a new database or open an existing one. If this dialog box is displayed, click *Blank Access Database*, and then click *OK*.

If you have already opened a database or closed the dialog box that displays when Microsoft Access starts up, click *New Database* on the toolbar, and then double-click the *Blank Database* icon on the *General* tab.

2. Specify a name and location for the database and click *Create*. (Below is the screen that shows up following this step).

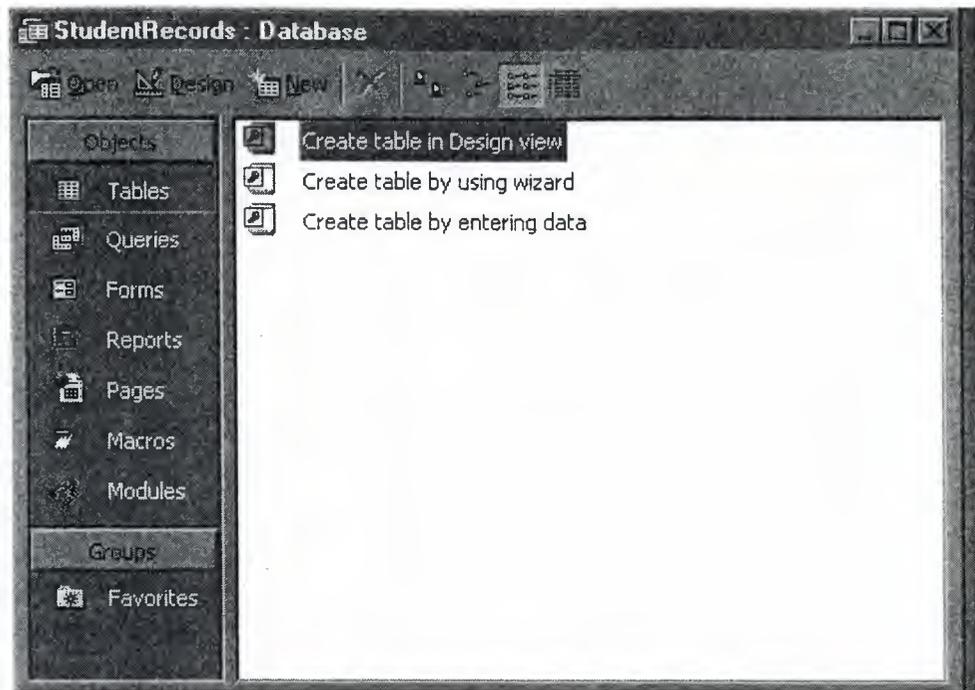


Figure 2.4 Create a database without using the Database Wizard

## 2.6. Tables

A table is a collection of data about a specific topic, such as students or contacts. Using a separate table for each topic means that you store that data only once, which makes your database more efficient, and reduces data-entry errors.

Tables organize data into columns (called *fields*) and rows (called *records*).

*Each field in the Student Records table contains the same type of information for every student, such as student's Social Security Number (Soc Sec #) This is an example of a COLUMN*

Student Records Table					
Soc Sec #	First Name	Last Name	BirthDate	Address	City
123456789	Todd	Jones	1/1/78	312 Wenona Rd	Bay City
315465866	Alan	Craig	2/8/80	123 N Union	Bay City
968585471	Stacy	Evans	3/8/81	RR 5 Box 880	Auburn
848131523	John	Anderson	4/5/80	83 Washington Dr.	Midland

*Each record in a Student Records table contains all of the information about one student, such as their First Name, Last Name, Birthday, Address, and City, etc. . This is an example of a ROW.*

## 2.7. Create a Table from scratch in Design view

1. If you haven't already done so, switch to the Database Window You can press F11 to switch to the Database window from any other window.

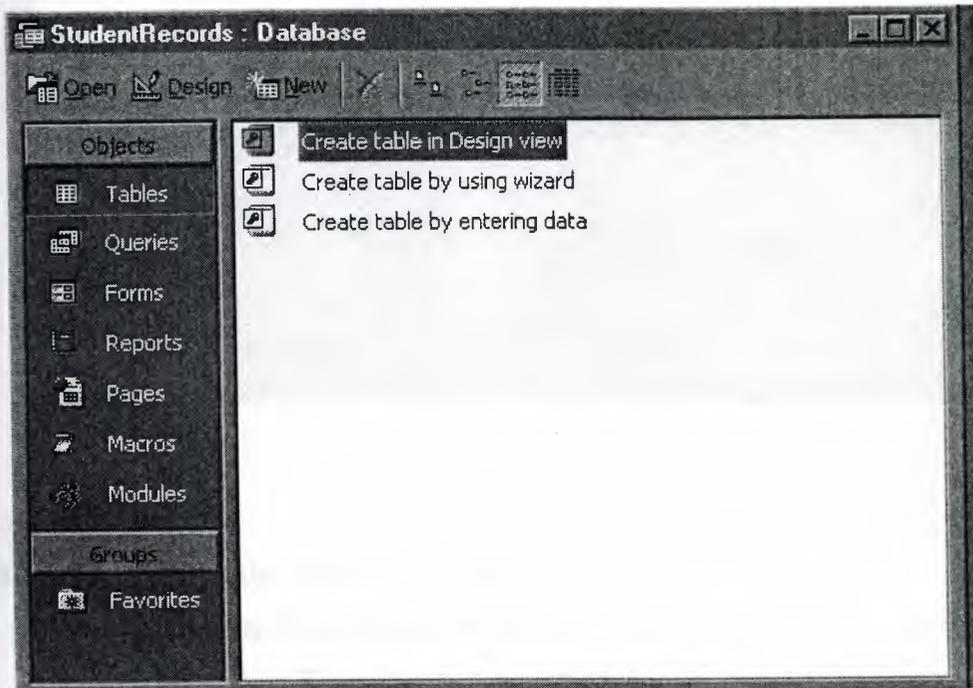
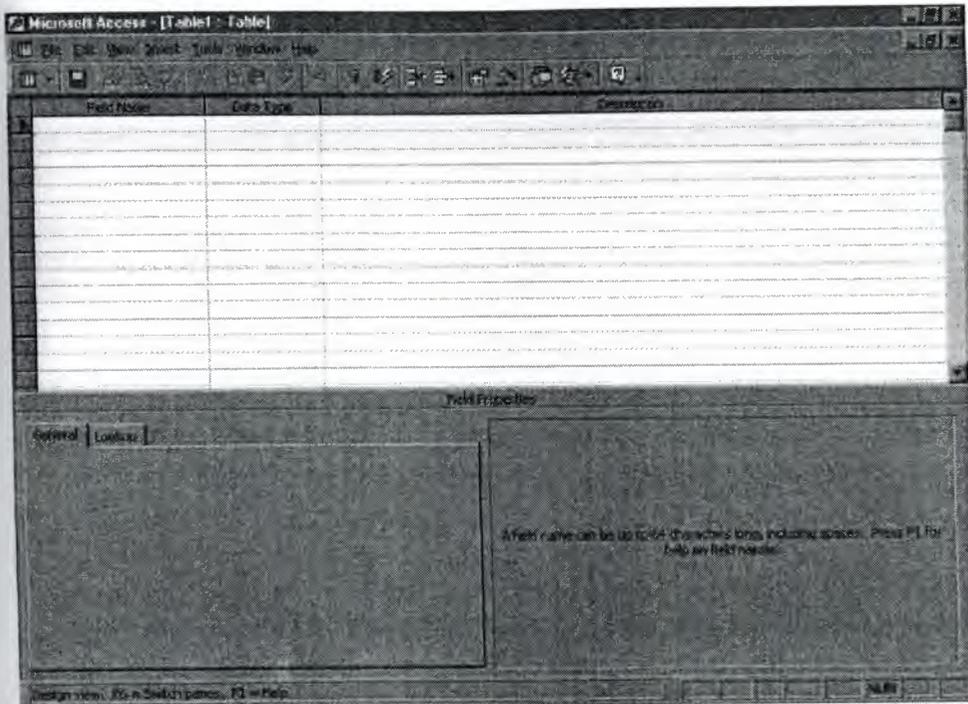


Figure 2.5 Create table in Design view

2. Double-Click on "Create table in Design view".  
(*DESIGN VIEW*)



3. Define each of the fields in your table.

- Under the Field Name column, enter the categories of your table.
- Under Data Type column, enter the type you want for you categories.

The attribute of a variable or field that determines what kind of data it can hold. For example, in a Microsoft Access database, the Text and Memo field data types allow the field to store either text or numbers, but the Number data type will allow the field to store numbers only. Number data type fields store numerical data that will be used in mathematical calculations. Use the Currency data type to display or calculate currency values. Other data types are Date/Time, Yes/No, Auto Number, and OLE object (Picture).

- Under the Description column, enter the text that describes what you field is. (This field is optional).

For our tutorial enter the following items:

Field Name	Data Type	Description
Soc Sec #	Text	Social Security Number. Uniquely identifies a student
First Name	Text	Student's First Name
Last Name	Text	Student's Last Name
BirthDate	Date/Time	Student's Birthdate
Address	Text	Students Address
City	Text	City student resides in
State	Text	State student resides in
Zip	Text	Zip Code student resides in
Phone	Text	Student's home phone number

## 2.8. Primary Key

- One or more fields (columns) whose value or values uniquely identify each record in a table. A primary key does not allow Null values and must always have a unique value. A primary key is used to relate a table to foreign keys in other tables.

You do not have to define a primary key, but it's usually a good idea. If you don't define a primary key, Microsoft Access asks you if you would like to create one when you save the table.

- For our tutorial, make the *Soc Sec #* field the primary key, meaning that *every* student has a social security number and no 2 are the same.
  - To do this, simply select the Soc Sec # field and select the primary key button
  - After you do this, Save the table

## 2.9. Switching Views

- To switch views from the datasheet (spreadsheet view) and the design view, simply click the button in the top-left hand corner of the Access program.

Datasheet View  
Design View



Displays the view, which allows you to enter raw data into your database table



Displays the view, which allows you to enter fields, data-types, and descriptions into your database table.

## 2.10. Entering Data

- Click on the Datasheet View and simply start "chugging" away by entering the data into each field. Before starting a new record, the *Soc Sec #* field must have something in it, because it is the Primary Key. If you did not set a Primary Key then it is OK.

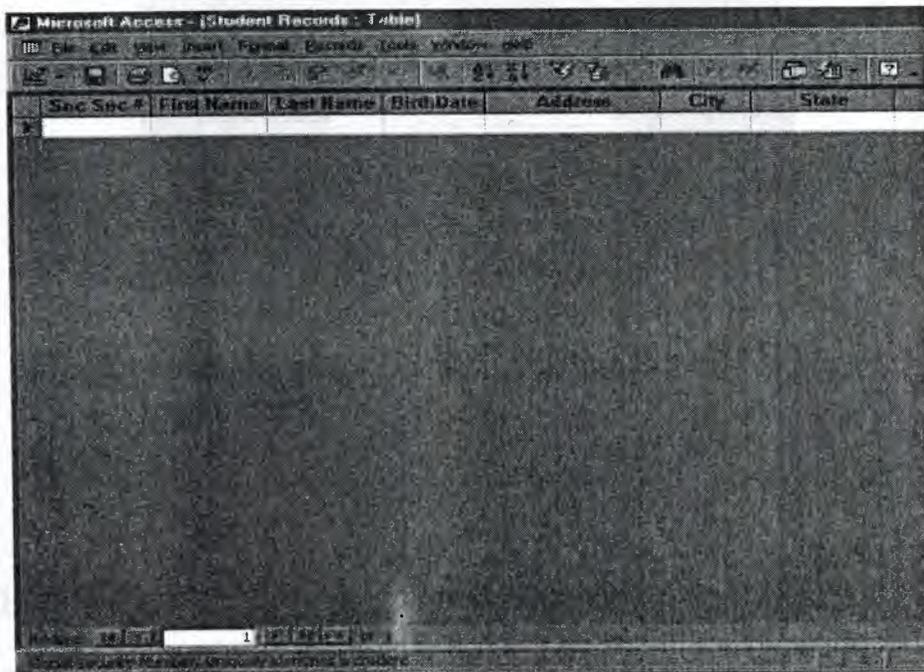


Figure 2.6 Entering Data

## 2.11. Manipulating Data

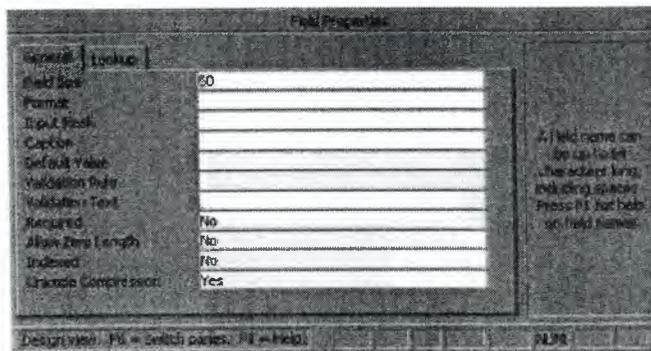
- **Adding a new row**
  - Simply drop down to a new line and enter the information

- **Updating a record**
  - Simply select the record and field you want to update, and change its data with what you want
- **Deleting a record**
  - Simply select the entire row and hit the Delete Key on the keyboard

## 2.12. Advanced Table Features w/Microsoft Access

### 2.12.1. Assigning a field a specific set of characters

- Example) Making a Social Security Number only allows 9 characters.
  1. Switch to Design View.
  2. Select the field you want to alter.
  3. At the bottom select the General Tab.



**Figure 2.7** Assigning a field a specific set of characters.

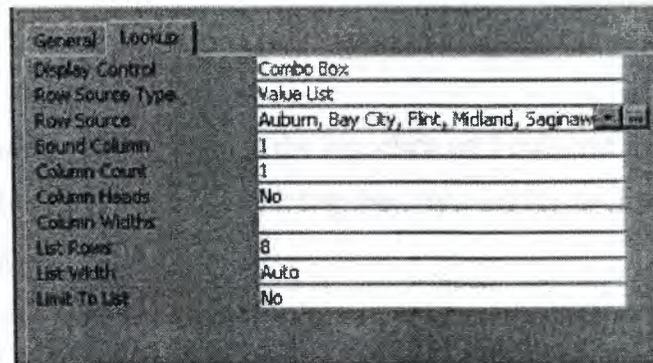
4. Select *Field Size*
5. Enter the number of characters you want this field to have

### 2.12.2. Formatting a field to look a specific way (HINT)

You do not need to assign a field a specific set of characters if you do this)

- Example) Formatting Phone Number w/ Area Code (xxx) xxx-xxxx
  1. Switch to Design View
  2. Select the field you want to format
  3. At the bottom select the General Tab

4. Select *Input Mask Box* and click on the ... button at the right.
5. Click on Next
6. Leave! (999) 000-0000 the way it is. This is a default.
7. Click Next
8. Select which option you want it to look like
9. Click Next
10. Click Finish



7. Select in the datasheet view and you should see the change when you go to the city field.

Address	City
	Auburn
	Bay City
	Flint
	Midland
	Saginaw

## 2.13. Relationships

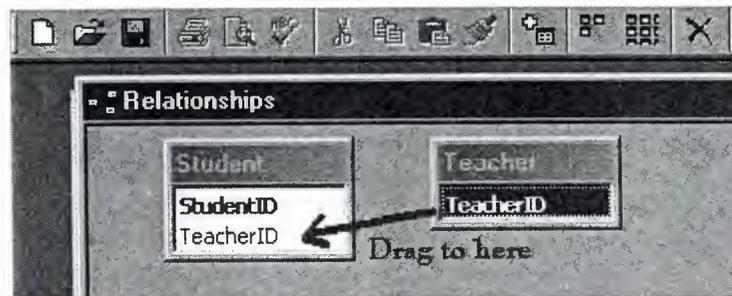
After you've set up multiple tables in your Microsoft Access database, you need a way of telling Access how to bring that information back together again. The first step in this process is to define relationships between your tables. After you've done that, you can create queries, forms, and reports to display information from several tables At once.

A relationship works by matching data in key fields - usually a field with the same name in both tables. In most cases, these matching fields are the primary key from one table, which provides a unique identifier for each record, and a foreign key in the other table. For example, teachers can be associated with the students they're responsible for by creating a relationship between the teacher's table and the student's table using the Teacher ID fields.

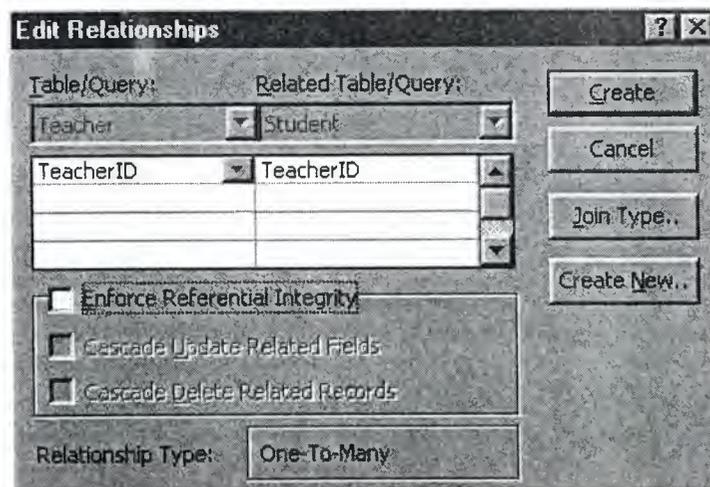
Having met the criteria above, follow these steps for creating relationships between tables.

1. In the database window view, at the top, click on Tools ---> Relationships
2. Select the Tables you want to link together, by clicking on them and selecting the Add Button

Drag the primary key of the Parent table (Teacher in this case), and drop it into the same field in the Child table (Student in this case.)



Select *Enforce Referential Integrity*.



**Figure 2.8** Edit Relationships



## **CHAPTER 3**

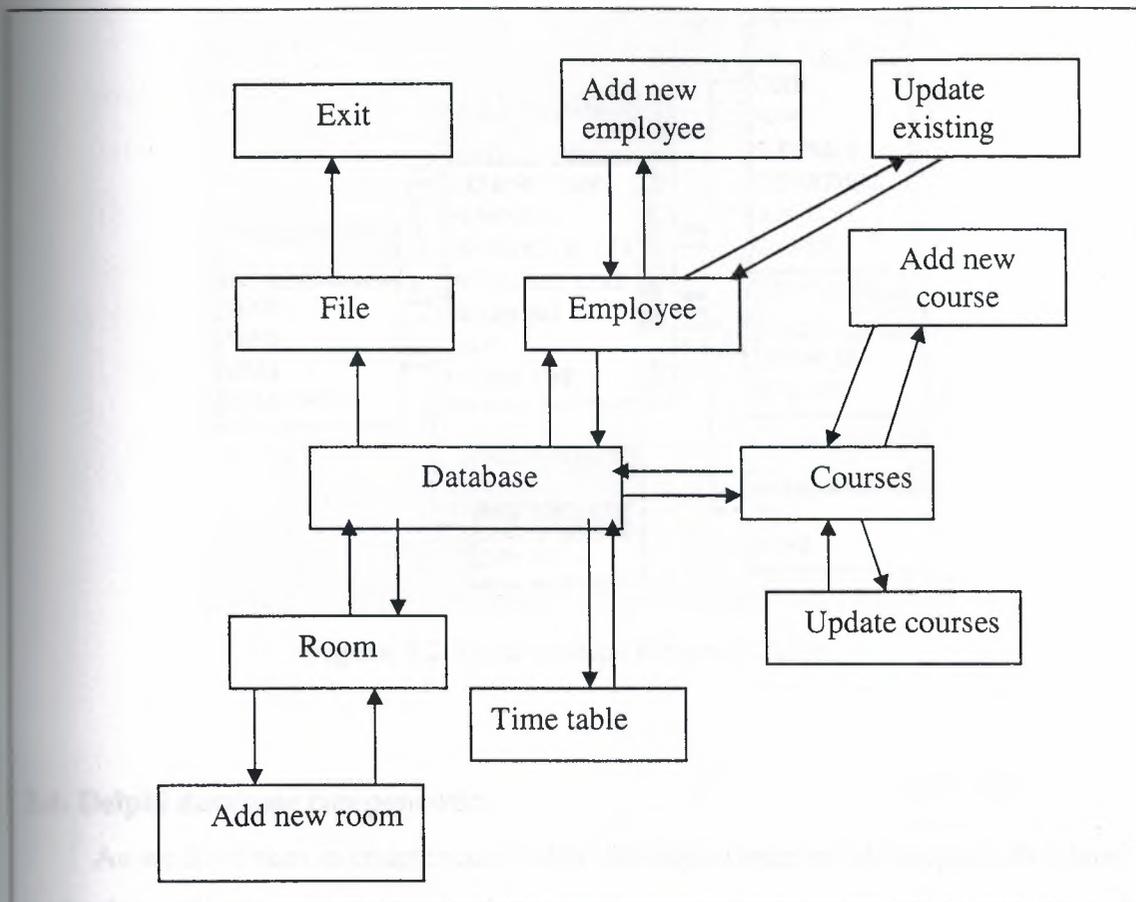
### **TIME TABLE DESIGN**

#### **3.1. Structure of Time Table Design:**

First thing as we know Delphi's support for database applications is one of the key feathers of the programming environment. Many programmers spend most of their time writing data-access code, which needs to be the most robust portion of a database application. You can create very complex database applications, starting from a blank form or one generated by Delphi's database from wizard. On computer, permanent data including database data is always stored in files. There are several techniques you can use to accomplish this storage. Delphi can use both approaches, or more precisely, you always refer to a database with its name, which is a sort of a nickname of a database but this reference can be to a database file or to a directory containing files with tables.

#### **3.2. Define Relationships Between Forms:**

As shown bellow in figure 3.1 we can see and understand the relationships and the procedures between forms, each form is related with the other forms through the database which is confirm a time table form at the end, later on we will see all the explanations of each form and its consistence.



**Figure 3.1** Relationships Between forms.

### 3.3. Define Relationships Between Tables:

When we create a relationship, the related fields don't have the same names. However, related field must have the same data type unless the primary key field is an AutoNumber field. We can match an AutoNumber field with a number field only if the fieldsize property of both of the matching fields is the same. Here we can see the relationships between the tables of this project as shown below:

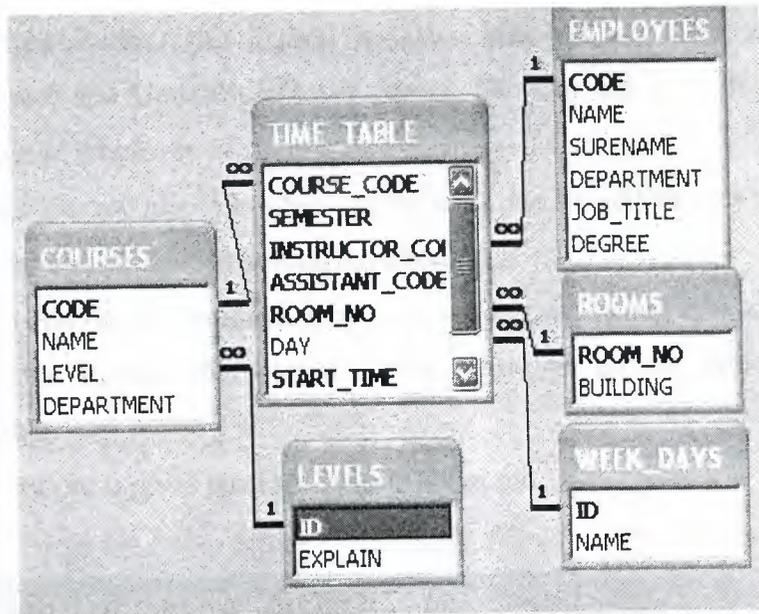


Figure 3.2 Relationships Between Tables.

### 3.4. Delphi database components:

As we have seen in chapter one Delphi includes a number of components related to database. The data access page of the component palette contains components used to interact to database. To access database in Delphi you generally need a data source, identified by data source component. The data source component, however, does not indicate the data directly, it refers to a data set component this can be tables (as in this project) or some other custom data set. As soon as you have placed a table component on the form, you can use the data sets property of the data source component to refer to it. for this property, the object inspector lists available data set of the current form or of other forms connected with the current one(using the file >used form command).[2]

### 3.5. Layout of the Application:

#### 3.5.1. Main menu screen:

It consists of six buttons. Each button has a specific mission, and these missions will be explaining as follow:

1. File Button: we can use this button to exit from the program.

2. Employee Button: this button has two sub buttons these are Add new employee, and Update employee record. We can use it to do whatever we want about employee's informations.
3. Courses Button: also it has two sub buttons, one for adding new course and the other for update the course's information.
4. Rooms Button: this button has one sub button used to add new room.
5. Timetable Button: this button allows us to get all the information about timetable.
6. Help button: it gives you information about me.

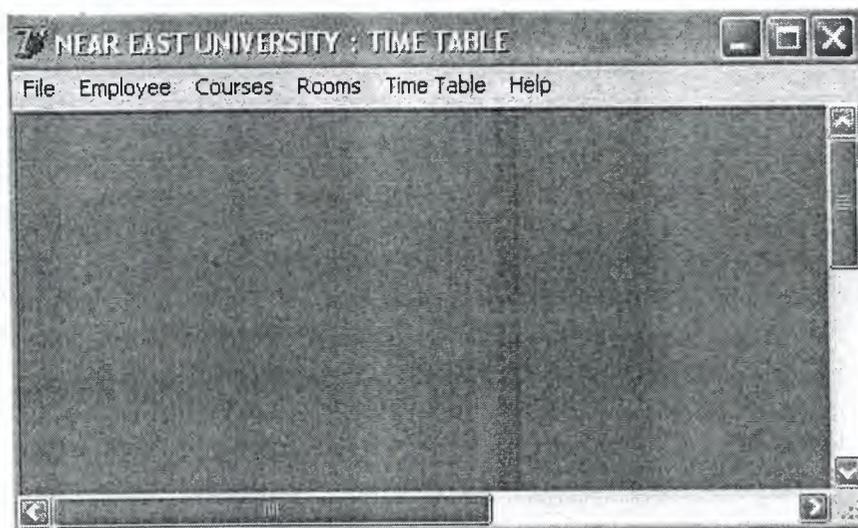


Figure 3.3 Main Menu Screen.

### 3.5.2. Add New Employee Screen:

This screen allows us to enter information about employees such as code, name, surname, department, job title, and degree. As we see in the figure3.4 bellow there are three DBEdit buttons and the other are DBComboBox to select the data. Also in this screen we have three BitBtn and their functions are shown as:

1. NEW Button: it's used to enter a new data.
2. SAVE Button: it's used to keep and save the information of employee to a file. When we click the save button without fill all the fields we will get an error message telling us that "please fill all fields", also if we insert a code which have been saved before we will get another error message telling us that " record can not be saved" We can create this procedure as follows :

```

Procedure TAddEmpForm.BitBtn1Click (Sender: TObject);
Begin
  If (DBEdit1.Text="") or (DBComboBox1.Text="") or
(DBComboBox2.Text="") or (DBComboBox3.Text="") or (DBEdit1.Text=")
then
  Begin
    MessageDlg (' Please Fill All Fields!', mtError,[mbOK],0);
  End
  Else
  Begin
    Try
      ADODataset1.Post;
    Except on Error: Exception do
      Begin
        MessageDlg ('Record Can not be saved!'+#13+Error.Message,
mtError,[mbOK],0);
        ADODataset1.Delete;
        BitBtn3Click (Sender);
      End;
    End;
    BitBtn3Click (sender);
  End;
End;
End.

```

3. CLOSE Button: it's used to exit from the screen.

Figure 3.4 Add New Employee Screen.

### 3.5.3. Update Employee Record Screen:

This form has three sections (search, update, and time table) and their functions as follow:

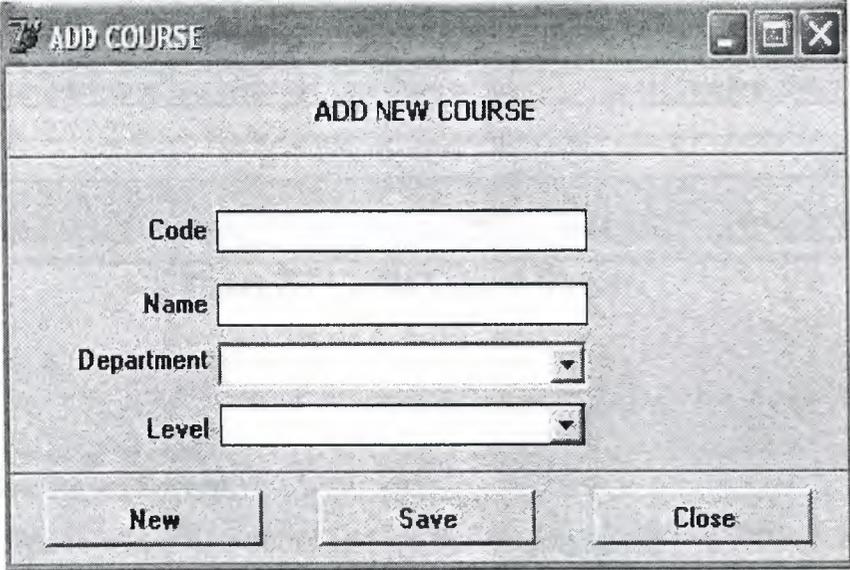
1. Searching Part: it has two Edit buttons which allow us to search by code or name of employee so we can call all the informations by these two buttons.
2. Updating part: this part allows us to adjust all the informations about employees which we entered it before such as code, name, department, and degree and save it again.
3. Time Table Part: since we insert the code or the name of employee we directly got the time table of that employee as shown bellow in figure 3.5.

COURSE_CODE	SEMESTER	ROOM_NO	START_TIME	END_TIME
COM 442	SPRING	DR4	11:10	13:30
COM 442	SPRING	DR2	13:30	15:30
COM 411	SPRING	R1	15:30	17:20
COM 411	SPRING	DR4	11:10	12:40

Figure 3.5 Update Employee Record.

### 3.5.4. Add New Course Screen:

This screen allows us to enter information about courses such as code, name, department, and level. As we see in the figure 3.6 below there are two DBEdit buttons and the other are DBComboBox to select the data. Also in this screen we have three BitBtn to create a new application and save the data in a file and close to exit from the screen.



The screenshot shows a Windows-style dialog box titled "ADD COURSE". The main content area is titled "ADD NEW COURSE". It contains four input fields: "Code" (a text box), "Name" (a text box), "Department" (a dropdown menu), and "Level" (a dropdown menu). At the bottom of the dialog, there are three buttons: "New", "Save", and "Close".

Figure 3.6 Add New Course Screen.

### 3.5.5. Update Courses Screen:

Here we follow the same procedure of update employee screen but here there is some different information, this screen used to correct or adjust some of courses information which saved before by using search button and save it again in a file.

The screenshot shows a window titled "UPDATE COURSES". It features a search section with "Code" and "Name" text boxes and a "Search" button. Below this is a form with "Code", "Name", "Department" (dropdown), and "Level" (dropdown) fields. At the bottom, there is a table with columns: SEMESTER, INSTRUCTOR\_CODE, ASSISTANT\_CODE, ROOM\_NO, DAY, START\_TIME, and END\_TIME. Below the table are "Save" and "Close" buttons.

Figure 3.7 Update Courses Screen.

### 3.5.6. Add New Room Screen:

Its design to add a new room and it's building by using two DBEdit buttons one for room and the second for building, in this screen we have three BitBtn and their functions are shown as:

1. NEW Button: it's used to enter a new data but to add it we have to fill the room field and building field other wise we will get an error message. We can see this procedure as:

```

Procedure TADDROOM.BitBtn1Click (Sender: TObject);
Begin
  ADODataSet1.Open;
  ADODataSet1.Insert;
  DBEdit1.SetFocus;
End;

```

```

Procedure TADDROOM.BitBtn2Click (Sender: TObject);
Begin
  if (DBEdit1.Text="") OR (DBEdit2.Text="") then
  Begin
    MessageDlg (' Please Fill All Fields!',mtError,[mbOK],0);
  End
  Else
  Begin
    Try

```

```

ADODataset1.Post;
Except on Error: Exception do
Begin
    MessageDlg ('Record Can not be saved!'+#13+Error.Message,
mtError,[mbOk],0);
    ADODataset1.Delete;
    BitBtn1Click (Sender);
End;
End;
BitBtn1Click (sender);
End;
End;

```

```

Procedure TADDDROOM.BitBtn3Click (Sender: TObject);
Begin
CLOSE;
End;

```

```

Procedure TADDDROOM.FormShow (Sender: TObject);
Begin
    BitBtn1Click (SENDER);
End;
End.

```

2. SAVE Button: it's used to keep and save the information of room to a file.
3. CLOSE Button: it's used to exit from the screen.

The image shows a Windows-style dialog box titled "ADD ROOM". Inside the dialog, the text "ADD NEW ROOM" is centered at the top. Below this, there are two labels, "Room" and "Building", each followed by a rectangular text input field. At the bottom of the dialog, there are three buttons arranged horizontally: "New", "Save", and "Close". The dialog box has standard window controls (minimize, maximize, close) in the top right corner.

**Figure 3.8** Add New Room Screen.

### 3.5.7. Time Table Screen:

We can say that this screen is our aim because it gives us the whole informations which are related with each other as we have seen it before between tables and also with screens. This screen has two parts:

1. Searching part: it consists of five ComboBox for Course, Instructor, Assistant, room, and day. By choose one of them or more we can get the time table of that searching which have been choosed. For example if we select instructor and we click search button we will get all the courses which are giving by instructor and it's name, semester, assistant, day, start time and end time because all the fields are related with each other. We can write this procedure as:

```

Procedure TTimeTableForm.BitBtn1Click (Sender: TObject);
Begin
ADODataset2.Close;
ADODataset2.CommandText:='SELECT * FROM TIME_TABLE WHERE
COURSE_CODE<>'+' ' ;

If ComboBox1.Text<>" then
ADODataset2.CommandText:=ADODataset2.CommandText+' AND
COURSE_CODE="'+ComboBox1.Text+'";
If ComboBox2.Text<>" then
Begin
ADODataset1.Close;
ADODataset1.CommandText:='SELECT * FROM EMPLOYEES ORDER BY
NAME ASC';
ADODataset1.Open;
ADODataset1.RecNo:=ComboBox2.ItemIndex+1;
ADODataset2.CommandText:=ADODataset2.CommandText+' AND
INSTRUCTOR_CODE="'+ADODataset1.FieldByName
('CODE').ASSTRING+'";
End;
If ComboBox3.Text<>" then
Begin
ADODataset1.Close;
ADODataset1.CommandText:='SELECT * FROM EMPLOYEES ORDER BY
NAME ASC';
ADODataset1.Open;
ADODataset1.RecNo:=ComboBox3.ItemIndex+1;
ADODataset2.CommandText:=ADODataset2.CommandText+' AND
ASSISTANT_CODE="'+ADODataset1.FieldByName ('CODE').ASSTRING+'";
End;
If ComboBox4.Text<>" then
ADODataset2.CommandText:=ADODataset2.CommandText+' AND
ROOM_NO="'+ComboBox4.Text+'";
If ComboBox5.Text<>" then

```

```

Begin
ADODataset1.Close;
ADODataset1.CommandText:='SELECT * FROM WEEK_DAYS ORDER BY
ID ASC';
ADODataset1.Open;
ADODataset1.RecNo:=ComboBox5.ItemIndex+1;
ADODataset2.CommandText:=ADODataset2.CommandText+' AND DAY
='+ADODataset1.FieldName ('ID').ASSTRING;
End;
ADODataset2.Open;
End;

```

```

Procedure TTimeTableForm.BitBtn2Click (Sender: TObject);

```

```

Begin
Try
ADODataset2.Edit;
ADODataset2.Post;
MessageDlg ('Time Table has been saved successfully.' mtInformation, [mbOK],
0);
// ResetForm ();
Except on Error: Exception do
Begin
MessageDlg ('Record Can not be saved!' +#13+Error.Message, mtError, [mbOK],
0);
End;
End;
End;

```

```

Procedure TTimeTableForm.BitBtn4Click (Sender: TObject);

```

```

Begin
ComboBox1.Text:="";
ComboBox2.Text:="";
ComboBox3.Text:="";
ComboBox4.Text:="";
ComboBox5.Text:="";
ADODataset2.Close;
End;
End.

```

2. Updating part: since we get all the informations about any section of searching this screen allows us to correct or adjust some of informations as we want .This part have nine of DBGrid buttons(course code, course name, semester, assistant, day, room, start time, and end time ) which confirm a time table.

Also this screen has three BitBtn and their functions are shown as:

1. RESET Button: it's used to reset and clear the data which appear
2. SAVE Button: it's used to keep and save the informations to a file.
3. CLOSE Button: it's used to exit from the screen.

The screenshot shows a window titled 'TIME TABLE' with a sub-header 'COURSES TIME TABLE'. It contains several search filters: 'Course', 'Instructor' (set to 'ADEL AMIRCANOV'), 'Assistant', 'Room', and 'Day'. A 'Search' button is located to the right of these filters. Below the filters is a table with the following data:

COURSE	COURSE NAME	SEMESTER	INSTRUCTOR	DAY	ROOM	START TIME	END TI
COM 442	OBJECT ORIENTED PR	SPRING	ADEL AMIRCANOV	Monday	DR4	11:10	13:30
COM 442	OBJECT ORIENTED PR	SPRING	ADEL AMIRCANOV	Wendsday	DR2	13:30	15:30
COM 411	SOFTWARE ENGINEER	SPRING	ADEL AMIRCANOV	Tuesday	R1	15:30	17:20
COM 411	SOFTWARE ENGINEER	SPRING	ADEL AMIRCANOV	Friday	DR4	11:10	12:40

At the bottom of the window, there are three buttons: 'Reset', 'Save', and 'Close'.

Figure 3.9 Time Table Screen.

## CONCLUSION

In the graduation project the description of tables and screens are given, the structure of the time table design (instructors, assistants, rooms, semesters, days, start time, and end time) is presented.

In this project I learned a lot of things that in the first time and even through not all of things I wanted to do in this project but this is mainly because of the lack of time and knowledge in programming with Delphi programming. But we can say the access database support is very extensive and complete. I have very high hopes on expanding the capability of this program in near future and from there I will take off in mastering Delphi to design any project. I will try to take a lot of experience which is very important tool that I will need to take any obstacles being faced in the future.

## REFERENCES

### APPENDIX

#### Books

[1]- Borland Delphi in 21 days

[2]- Jeff Duntemann, Jim Mischel, and Don Taylor, " Delphi Programming Explorer"  
the coriolis grope Inc 1995.

[3]- MarcoCantu, " Mastering Delphi ", SYBEX, second edition.

#### Websites

- [www.sybex.com](http://www.sybex.com)
- [www.marcocantu.com](http://www.marcocantu.com)
- [www.kdtool.net](http://www.kdtool.net)

## APPENDIX

**1.Main menu**

```
unit MainUnit;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Menus, ComCtrls, DB, ADODB, ExtCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
  MainMenu1: TMainMenu;
```

```
  File1: TMenuItem;
```

```
  Exit1: TMenuItem;
```

```
  Employee1: TMenuItem;
```

```
  Add1: TMenuItem;
```

```
  Update1: TMenuItem;
```

```
  Courses1: TMenuItem;
```

```
  AddNewCourse1: TMenuItem;
```

```
  Update2: TMenuItem;
```

```
  Rooms1: TMenuItem;
```

```
  AddNewRoom1: TMenuItem;
```

```
  imeTable1: TMenuItem;
```

```
  Help1: TMenuItem;
```

```
  About1: TMenuItem;
```

```
  StatusBar1: TStatusBar;
```

```
  Connection1: TADOConnection;
```

```
  Image1: TImage;
```

```
  procedure Exit1Click(Sender: TObject);
```

```
  procedure Add1Click(Sender: TObject);
```

```
  procedure Update1Click(Sender: TObject);
```

```
  procedure AddNewCourse1Click(Sender: TObject);
```

```
  procedure Update2Click(Sender: TObject);
```

```
  procedure imeTable1Click(Sender: TObject);
```

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

```
  procedure AddNewRoom1Click(Sender: TObject);
```

```
private
```

```
  { Private declarations }
```

```
public
```

```
  { Public declarations }
```

```
end;
```

```
var
```

```
  MainForm: TMainForm;
```

```
implementation
```

```

uses AddEmpUnit, UpdateEmpUnit, AddCourseUnit, UpdateCoursesUnit,
  TimeTableUnit, ADDROOMUnit;

{$R *.dfm}

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

procedure TMainForm.Add1Click(Sender: TObject);
begin
  Application.CreateForm(TAddEmpForm, AddEmpForm);
end;

procedure TMainForm.Update1Click(Sender: TObject);
begin
  Application.CreateForm(TUpdateEmpForm, UpdateEmpForm);
end;

procedure TMainForm.AddNewCourse1Click(Sender: TObject);
begin
  Application.CreateForm(TAddCourseForm, AddCourseForm);
end;

procedure TMainForm.Update2Click(Sender: TObject);
begin
  Application.CreateForm(TUpdateCoursesForm, UpdateCoursesForm);
end;

procedure TMainForm.timeTable1Click(Sender: TObject);
begin
  Application.CreateForm(TTimeTableForm, TimeTableForm);
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
  TxtF : TextFile;
  S : String;
begin
  AssignFile(TxtF, ExtractFilePath(Application.ExeName)+'DatabasePath.txt');
  Reset(TxtF);
  ReadLn(TxtF, S);
  Connection1.ConnectionString:='Provider=Microsoft.Jet.OLEDB.4.0;Data
Source='+S+';Persist Security Info=False';
  Connection1.Open;
  CloseFile(TxtF);
end;

```

```

procedure TMainForm.AddNewRoom1Click(Sender: TObject);
begin
  Application.CreateForm(TADDROOM, ADDROOM);
  ADDROOM.Show;
end;

end.

```

## 2.ADD NEW EMPLOYEE

```

unit AddEmpUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, DBCtrls, Mask, DB, ADODB;

type
  TAddEmpForm = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBComboBox1: TDBComboBox;
    DBComboBox2: TDBComboBox;
    DBComboBox3: TDBComboBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    ADODataset1: TADODataset;
    BitBtn3: TBitBtn;
    ADODataset1CODE: TWideStringField;
    ADODataset1NAME: TWideStringField;
    ADODataset1SURENAME: TWideStringField;
    ADODataset1DEPARTMENT: TWideStringField;
    ADODataset1JOB_TITLE: TWideStringField;
    ADODataset1DEGREE: TWideStringField;
    DataSource1: TDataSource;
    procedure BitBtn2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormShow(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  end;

```

```

    procedure BitBtn1Click(Sender: TObject);
    procedure Panel2Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    AddEmpForm: TAddEmpForm;

implementation

uses MainUnit;

{$R *.dfm}

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

procedure TAddEmpForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Action:=cafree;
end;

procedure TAddEmpForm.FormShow(Sender: TObject);
begin
    width:=450;
    Height:=350;
    BitBtn3Click(sender);
end;

procedure TAddEmpForm.BitBtn3Click(Sender: TObject);
begin
    DataSource1.AutoEdit:=true;
    ADODataset1.Insert;
    DBEdit1.SetFocus;
end;

procedure TAddEmpForm.BitBtn1Click(Sender: TObject);
begin
    if (DBEdit1.Text='') or (DBComboBox1.Text='') or (DBComboBox2.Text='') or
(DBComboBox3.Text='') or (DBEdit1.Text='') then
        begin
            MessageDlg(' Please Fill All Fields !',mtError,[mbOK],0);
        end
    else
        begin

```

```

try
  ADODataset1.Post;
except on Error:Exception do
begin
  MessageDlg('Record Can not be saved
'+#13+Error.Message,mtError,[mbOk],0);
  ADODataset1.Delete;
  BitBtn3Click(Sender);
end;
end;
BitBtn3Click(sender);
end;
end;

procedure TAddEmpForm.Panel2Click(Sender: TObject);
begin

end;

end.

```

### 3.ADD ROOM

```

unit ADDROOMUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Mask, DBCtrls, ExtCtrls, DB, ADODB;

type
  TADDROOM = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    ADODataset1: TADODataset;
    DataSource1: TDataSource;
    ADODataset1ROOM_NO: TWideStringField;
    ADODataset1BUILDING: TWideStringField;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);

```

```

procedure BitBtn3Click(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure Panel2Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  ADDRROOM: TADDRROOM;

implementation

uses MainUnit;

{$R *.dfm}

procedure TADDRROOM.BitBtn1Click(Sender: TObject);
begin
  ADODataset1.Open;
  ADODataset1.Insert;
  DBEdit1.SetFocus;
end;

procedure TADDRROOM.BitBtn2Click(Sender: TObject);
begin
  if (DBEdit1.Text='') OR (DBEdit2.Text='') then
  begin
    MessageDlg(' Please Fill All Fields !',mtError,[mbOK],0);
  end
  else
  begin
    try
      ADODataset1.Post;
    except on Error:Exception do
      begin
        MessageDlg('Record Can not be saved
!'+#13+Error.Message,mtError,[mbOk],0);
        ADODataset1.Delete;
        BitBtn1Click(Sender);
      end;
    end;
    BitBtn1Click(sender);
  end;
end;

procedure TADDRROOM.BitBtn3Click(Sender: TObject);
begin
  CLOSE;

```

```

end;

procedure TADDDROOM.FormShow(Sender: TObject);
begin
  BitBtn1Click(SENDER);
end;

procedure TADDDROOM.Panel2Click(Sender: TObject);
begin

end;

end.

```

#### 4.TIME TABLE UNIT

```

unit TimeTableUnit;

interface

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

type
  TTimeTableForm = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    ComboBox3: TComboBox;
    ComboBox4: TComboBox;
    ComboBox5: TComboBox;
    BitBtn1: TBitBtn;
    ADODataset1: TADODataset;
    BitBtn4: TBitBtn;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    ADODataset2: TADODataset;

```

```

ADODataset3: TADODataset;
ADODataset4: TADODataset;
ADODataset6: TADODataset;
ADODataset2COURSE_CODE: TWideStringField;
ADODataset2SEMESTER: TWideStringField;
ADODataset2INSTRUCTOR_CODE: TWideStringField;
ADODataset2ASSISTANT_CODE: TWideStringField;
ADODataset2ROOM_NO: TWideStringField;
ADODataset2DAY: TIntegerField;
ADODataset3CODE: TWideStringField;
ADODataset3NAME: TWideStringField;
ADODataset3LEVEL: TIntegerField;
ADODataset3DEPARTMENT: TWideStringField;
ADODataset4CODE: TWideStringField;
ADODataset4NAME: TWideStringField;
ADODataset4SURENAME: TWideStringField;
ADODataset4DEPARTMENT: TWideStringField;
ADODataset4JOB_TITLE: TWideStringField;
ADODataset4DEGREE: TWideStringField;
ADODataset6ID: TAutoIncField;
ADODataset6NAME: TWideStringField;
ADODataset2INSTNAME: TStringField;
ADODataset2ASSNAME: TStringField;
ADODataset2DAYSTR: TStringField;
ADODataset2COURSENAME: TStringField;
ADODataset2CCODE: TStringField;
ADODataset5: TADODataset;
ADODataset2ROOMNO: TStringField;
ADODataset2START_TIME: TWideStringField;
ADODataset2END_TIME: TWideStringField;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormShow(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  TimeTableForm: TTimeTableForm;

implementation

uses MainUnit;

{$R *.dfm}

```

```

procedure TTimeTableForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action:=cafree;
end;

procedure TTimeTableForm.FormShow(Sender: TObject);
begin
  ADODataset1.Close;
  ADODataset1.CommandText:='SELECT CODE FROM COURSES';
  ADODataset1.Open;
  ADODataset1.First;
  While not ADODataset1.Eof do
  begin
    ComboBox1.Items.Add(ADODataset1.FieldByName('CODE').AsString);
    ADODataset1.Next;
  end;

  ADODataset1.Close;
  ADODataset1.CommandText:='SELECT NAME FROM EMPLOYEES ORDER
BY NAME ASC';
  ADODataset1.Open;
  ADODataset1.First;
  While not ADODataset1.Eof do
  begin
    ComboBox2.Items.Add(ADODataset1.FieldByName('NAME').AsString);
    ADODataset1.Next;
  end;

  ADODataset1.Close;
  ADODataset1.CommandText:='SELECT NAME FROM EMPLOYEES ORDER
BY NAME ASC';
  ADODataset1.Open;
  ADODataset1.First;
  While not ADODataset1.Eof do
  begin
    ComboBox3.Items.Add(ADODataset1.FieldByName('NAME').AsString);
    ADODataset1.Next;
  end;

  ADODataset1.Close;
  ADODataset1.CommandText:='SELECT ROOM_NO FROM ROOMS';
  ADODataset1.Open;
  ADODataset1.First;
  While not ADODataset1.Eof do
  begin
    ComboBox4.Items.Add(ADODataset1.FieldByName('ROOM_NO').AsString);
    ADODataset1.Next;
  end;

```

```

ADODataSet1.Close;
ADODataSet1.CommandText:='SELECT NAME FROM WEEK_DAYS ORDER
BY ID ASC';
ADODataSet1.Open;
ADODataSet1.First;
While not ADODataSet1.Eof do
begin
  ComboBox5.Items.Add(ADODataSet1.FieldByName('NAME').AsString);
  ADODataSet1.Next;
end;
end;

procedure TTimeTableForm.BitBtn3Click(Sender: TObject);
begin
  Close;
end;

procedure TTimeTableForm.BitBtn1Click(Sender: TObject);
begin
  ADODataSet2.Close;
  ADODataSet2.CommandText:='SELECT * FROM TIME_TABLE WHERE
COURSE_CODE<>"" ';

  If ComboBox1.Text<>" then
    ADODataSet2.CommandText:=ADODataSet2.CommandText+' AND
COURSE_CODE="'+ComboBox1.Text+'";
  If ComboBox2.Text<>" then
    Begin
      ADODataSet1.Close;
      ADODataSet1.CommandText:='SELECT * FROM EMPLOYEES ORDER BY
NAME ASC';
      ADODataSet1.Open;
      ADODataSet1.RecNo:=ComboBox2.ItemIndex+1;
      ADODataSet2.CommandText:=ADODataSet2.CommandText+' AND
INSTRUCTOR_CODE="'+ADODataSet1.FieldByName('CODE').ASSTRING+'";
    end;
  If ComboBox3.Text<>" then
    Begin
      ADODataSet1.Close;
      ADODataSet1.CommandText:='SELECT * FROM EMPLOYEES ORDER BY
NAME ASC';
      ADODataSet1.Open;
      ADODataSet1.RecNo:=ComboBox3.ItemIndex+1;
      ADODataSet2.CommandText:=ADODataSet2.CommandText+' AND
ASSISTANT_CODE="'+ADODataSet1.FieldByName('CODE').ASSTRING+'";
    end;
  If ComboBox4.Text<>" then
    ADODataSet2.CommandText:=ADODataSet2.CommandText+' AND
ROOM_NO="'+ComboBox4.Text+'";

```

```

If ComboBox5.Text<>" then
Begin
  ADODataset1.Close;
  ADODataset1.CommandText:='SELECT * FROM WEEK_DAYS ORDER BY ID
ASC';
  ADODataset1.Open;
  ADODataset1.RecNo:=ComboBox5.ItemIndex+1;
  ADODataset2.CommandText:=ADODataset2.CommandText+' AND DAY
='+ADODataset1.FieldByName('ID').ASSTRING;
end;
  ADODataset2.Open;
end;

```

```

procedure TTimeTableForm.BitBtn2Click(Sender: TObject);
begin
  try
    ADODataset2.Edit;
    ADODataset2.Post;
    MessageDlg('Time Table has been saved successfully.',mtInformation,[mbOk],0);
  //  ResetForm();
  except on Error:Exception do
    begin
      MessageDlg('Record Can not be saved
!'+#13+Error.Message,mtError,[mbOk],0);
    end;
  end;
end;

```

```

procedure TTimeTableForm.BitBtn4Click(Sender: TObject);
begin
  ComboBox1.Text:="";
  ComboBox2.Text:="";
  ComboBox3.Text:="";
  ComboBox4.Text:="";
  ComboBox5.Text:="";
  ADODataset2.Close;
end;

end.

```

## 5.UPDATE COURSE

```
unit UpdateCoursesUnit;
```

```
interface
```

```
uses
```

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

```

type
  TUpdateCoursesForm = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    BitBtn1: TBitBtn;
    Panel2: TPanel;
    Panel3: TPanel;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Panel4: TPanel;
    DBGrid1: TDBGrid;
    ADODataset1: TADODataset;
    ADODataset2: TADODataset;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    Label3: TLabel;
    Label4: TLabel;
    DBComboBox2: TDBComboBox;
    Label5: TLabel;
    Label6: TLabel;
    DBLookupComboBox1: TDBLookupComboBox;
    ADODataset3: TADODataset;
    DataSource3: TDataSource;
    ADODataset2COURSE_CODE: TWideStringField;
    ADODataset2SEMESTER: TWideStringField;
    ADODataset2INSTRUCTOR_CODE: TWideStringField;
    ADODataset2ASSISTANT_CODE: TWideStringField;
    ADODataset2ROOM_NO: TWideStringField;
    ADODataset2DAY: TIntegerField;
    ADODataset2START_TIME: TWideStringField;
    ADODataset2END_TIME: TWideStringField;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure BitBtn2Click(Sender: TObject);
    procedure ResetForm();
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  UpdateCoursesForm: TUpdateCoursesForm;

```

implementation

uses MainUnit;

{ \$R \*.dfm }

```

procedure TUpdateCoursesForm.BitBtn1Click(Sender: TObject);
begin
  if (Edit1.Text<>") and (Edit2.Text<>") then
  begin
    ADODataSet1.Close;
    ADODataSet1.CommandText:='SELECT * FROM COURSES WHERE
CODE="'+Edit1.Text+'" AND NAME="'+Edit2.Text+'";
    ADODataSet1.Open;
    ADODataSet1.Edit;
    ADODataSet2.Close;
    ADODataSet2.CommandText:='SELECT * FROM TIME_TABLE WHERE
COURSE_CODE="'+Edit1.Text+'";
    ADODataSet2.Open;
  end
  else
  if (Edit1.Text<>") then
  begin
    ADODataSet1.Close;
    ADODataSet1.CommandText:='SELECT * FROM COURSES WHERE
CODE="'+Edit1.Text+'";
    ADODataSet1.Open;
    ADODataSet1.Edit;
    ADODataSet2.Close;
    ADODataSet2.CommandText:='SELECT * FROM TIME_TABLE WHERE
COURSE_CODE="'+Edit1.Text+'";
    ADODataSet2.Open;
  end
  else
  if (Edit2.Text<>") then
  begin
    ADODataSet1.Close;
    ADODataSet1.CommandText:='SELECT * FROM COURSES WHERE
NAME="'+Edit2.Text+'";
    ADODataSet1.Open;
    ADODataSet1.Edit;
    Edit1.Text:=ADODataSet1.FieldByName('CODE').AsString;
    ADODataSet2.Close;
    ADODataSet2.CommandText:='SELECT * FROM TIME_TABLE WHERE
COURSE_CODE="'+Edit1.Text+'";
    ADODataSet2.Open;
  end;
end;

```

```

procedure TUpdateCoursesForm.BitBtn3Click(Sender: TObject);
begin
  Close;
end;

```

```

procedure TUpdateCoursesForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action:=cafree;
end;

```

```

procedure TUpdateCoursesForm.BitBtn2Click(Sender: TObject);
begin
  try
    ADODataset1.Post;
    ResetForm();
  except on Error:Exception do
    begin
      MessageDlg('Record Can not be saved
!'+#13+Error.Message,mtError,[mbOk],0);
    end;
  end;
end;

```

```

procedure TUpdateCoursesForm.ResetForm();
begin
  Edit1.Text:="";
  Edit2.Text:="";
  ADODataset1.Close;
  ADODataset2.Close;
  Edit1.SetFocus;
end;

```

end.

## 6.UPDATE EMPLOYEE RECORD

```

unit UpdateEmpUnit;

```

```

interface

```

```

uses

```

```

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

```

```

type

```

```

  TUpdateEmpForm = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;

```

```

Panel3: TPanel;
Edit1: TEdit;
Label1: TLabel;
Edit2: TEdit;
Label2: TLabel;
BitBtn1: TBitBtn;
Label6: TLabel;
Label5: TLabel;
Label4: TLabel;
Label3: TLabel;
Label7: TLabel;
Label8: TLabel;
DBEdit3: TDBEdit;
DBEdit2: TDBEdit;
DBEdit1: TDBEdit;
DBComboBox3: TDBComboBox;
DBComboBox2: TDBComboBox;
DBComboBox1: TDBComboBox;
Panel4: TPanel;
DBGrid1: TDBGrid;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
ADODataset1: TADODataset;
ADODataset2: TADODataset;
DataSource1: TDataSource;
DataSource2: TDataSource;
procedure BitBtn3Click(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure ResetForm();
private
  { Private declarations }
public
  { Public declarations }
end;

var
  UpdateEmpForm: TUpdateEmpForm;

implementation

uses MainUnit;

{$R *.dfm}

procedure TUpdateEmpForm.BitBtn3Click(Sender: TObject);
begin
  Close;
end;

```

```

procedure TUpdateEmpForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action:=cafree;
end;

procedure TUpdateEmpForm.BitBtn1Click(Sender: TObject);
begin
  if (Edit1.Text<>") and (Edit2.Text<>") then
  begin
    ADODataset1.Close;
    ADODataset1.CommandText:='SELECT * FROM EMPLOYEES WHERE
CODE="'+Edit1.Text+"' AND NAME="'+Edit2.Text+"'";
    ADODataset1.Open;
    ADODataset1.Edit;
    ADODataset2.Close;
    ADODataset2.CommandText:='SELECT * FROM TIME_TABLE WHERE
INSTRUCTOR_CODE="'+Edit1.Text+"' OR ASSISTANT_CODE="'+Edit1.Text+"'";
    ADODataset2.Open;
  end
  else
  if (Edit1.Text<>") then
  begin
    ADODataset1.Close;
    ADODataset1.CommandText:='SELECT * FROM EMPLOYEES WHERE
CODE="'+Edit1.Text+"'";
    ADODataset1.Open;
    ADODataset1.Edit;
    ADODataset2.Close;
    ADODataset2.CommandText:='SELECT * FROM TIME_TABLE WHERE
INSTRUCTOR_CODE="'+Edit1.Text+"' OR ASSISTANT_CODE="'+Edit1.Text+"'";
    ADODataset2.Open;
  end
  else
  if (Edit2.Text<>") then
  begin
    ADODataset1.Close;
    ADODataset1.CommandText:='SELECT * FROM EMPLOYEES WHERE
NAME="'+Edit2.Text+"'";
    ADODataset1.Open;
    ADODataset1.Edit;
    Edit1.Text:=ADODataset1.FieldByName('CODE').AsString;
    ADODataset2.Close;
    ADODataset2.CommandText:='SELECT * FROM TIME_TABLE WHERE
INSTRUCTOR_CODE="'+Edit1.Text+"' OR ASSISTANT_CODE="'+Edit1.Text+"'";
    ADODataset2.Open;
  end;
end;

```

```

procedure TUpdateEmpForm.BitBtn2Click(Sender: TObject);
begin
  try
    ADODataset1.Post;
    ResetForm();
  except on Error:Exception do
    begin
      MessageDlg('Record Can not be saved
!'+#13+Error.Message,mtError,[mbOk],0);
    end;
  end;
end;

```

```

procedure TUpdateEmpForm.ResetForm();
begin
  Edit1.Text:="";
  Edit2.Text:="";
  ADODataset1.Close;
  ADODataset2.Close;
  Edit1.SetFocus;
end;

end.

```

## 7.ADD COURSE UNIT

```
unit AddCourseUnit;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, DB, ADOdb, StdCtrls, Buttons, DBCtrls, Mask, ExtCtrls;
```

```
type
```

```

TAddCourseForm = class(TForm)
  Panel1: TPanel;
  Panel2: TPanel;
  Label1: TLabel;
  Label2: TLabel;
  Label4: TLabel;
  DBEdit1: TDBEdit;
  DBEdit2: TDBEdit;
  Panel3: TPanel;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  BitBtn3: TBitBtn;
  ADODataset1: TADODataset;
  DataSource1: TDataSource;

```

```

ADODataset1CODE: TWideStringField;
ADODataset1NAME: TWideStringField;
ADODataset1LEVEL: TIntegerField;
Label3: TLabel;
ADODataset1DEPARTMENT: TWideStringField;
DBLookupComboBox1: TDBLookupComboBox;
ADODataset2: TADODataset;
DataSource2: TDataSource;
DBComboBox2: TDBComboBox;
procedure BitBtn2Click(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure Panel2Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

```

```

var
  AddCourseForm: TAddCourseForm;

```

```

implementation

```

```

uses MainUnit;

```

```

{$R *.dfm}

```

```

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

```

```

procedure TAddCourseForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action:=cafree;
end;

```

```

procedure TAddCourseForm.BitBtn3Click(Sender: TObject);
begin
  DataSource1.AutoEdit:=TRUE;
  ADODataset1.Cancel;
  ADODataset1.Insert;
  DBEdit1.SetFocus;
end;

```

```

procedure TAddCourseForm.BitBtn1Click(Sender: TObject);

```

```
begin
  if (DBEdit1.Text=") or (DBEdit2.Text=") or (DBComboBox2.Text=") or
  (DBLookupComboBox1.Text=") then
  begin
    MessageDlg(' Please Fill All Fields !',mtError,[mbOK],0);
  end
  else
  begin
    try
      ADODataset1.Post;
    except on Error:Exception do
      begin
        MessageDlg('Record Can not be saved
!'+#13+Error.Message,mtError,[mbOk],0);
        ADODataset1.Delete;
        BitBtn3Click(Sender);
      end;
    end;
    BitBtn3Click(sender);
  end;
end;

procedure TAddCourseForm.FormShow(Sender: TObject);
begin
  BitBtn3Click(Sender);
end;

procedure TAddCourseForm.Panel2Click(Sender: TObject);
begin

end;

end.
```