

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

**DATA BANK PROGRAMMING FOR CLEANING
COMPANY**

**Graduation Project
COM 400**

Student:	Barış Çelik
Supervisor:	Ümit SOYER

Nicosia – 2008

ACKNOWLEDGEMENTS

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

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

Finally ,I would also like to thank all my friends for their advice and support."

ABSTRACT

The aim of this project is that provide getting extra time and efficiency for cleaning companies. The basic idea is that input the informations easily into the computer and find out the informations from the computer in the shortest time. Also integrity and reliability is so important between the records.

So in my project, after getting the customer orders, an invoice is prepared for this job. Then employees' records are set up for this job and materials are signficated for each employee. Now we can find out the records about which employees worked in significant job and which cleaning materials were used for this job easily and efficiently. Which customer has how much money dept, when did the customer take the invoice and when did the customer give the receipt to the company. All these informations are gotten easily and we can keep the integrity between the records and we can get the reliability about these records.

Today's technology provides so many eases for our life. Especially we can see these eases in working life and its companies. Most of the companies needs extra time and efficiency to serve better their customers. Computers also main part of this efficiency and getting extra time. Companies need to save and use the records on necessary time about theirs customers, materials, bills, employees.. and so on. Also companies need to search and find out any information about these saved records in short time.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	1
CHAPTER 1	2
DELPHI BASICS	2
1.1 What is Delphi?	2
1.2 A Brief History of Borland's Delphi	3
1.3 Writing your first Delphi program	4
1.4 Delphi data types	7
1.5 Programming logic	15
1.6 Repeating sets of commands	22
1.7 Dates and times	26
1.8 Standard tab GUI components	33
CHAPTER 2	40
MICROSOFT ACCESS	40
2.1 Getting Started	40
2.1.1 A Few Terms	40
2.1.2 Getting Started	41
2.1.3 Blank Access database	41
2.1.4 Access database wizards, pages, and projects	42
2.1.5 Open an existing database	43
2.1.6 Converting to Access 2000	43
2.2 Screen Layouts	44
2.2.1 Database Window	44
2.2.2 Design View	44
2.2.3 Datasheet View	46
2.3 Creating Tables	46
2.3.1 Introduction to Tables	46
2.3.2 Create a Table in Design View	47
2.3.3 Field Properties	49
2.3.4 Primary Key	53
2.3.5 Indexes	53
2.3.6 Field Validation Rules	54
2.4 Datasheet Records	55
2.4.1 Adding Records	55
2.4.2 Editing Records	56
2.4.3 Deleting Records	56
2.4.4 Adding and Deleting Columns	56
2.4.5 Resizing Rows and Columns	56
2.4.6 Freezing Columns	57
2.4.7 Hiding Columns	57
2.4.8 Finding Data in a Table	58
2.4.9 Replace	59
2.4.10 Check Spelling and AutoCorrect	60
2.4.11 Print a Datasheet	60
2.5 Table Relationships	60
2.6 Queries	63

2.6.1	Introduction to Queries	63
2.6.2	Create a Query in Design View	63
2.6.3	Query Wizard	66
2.6.4	Find Duplicates Query	67
2.6.5	Delete a Query	70
2.7	Forms	70
2.7.1	Create Form by Using Wizard	70
2.7.2	Create Form in Design View	73
2.7.3	Adding Records Using A Form	74
2.7.4	Editing Forms	75
2.8	More Forms	77
2.8.1	Multiple-Page Forms Using Tabs	77
2.8.2	Conditional Formatting	77
2.8.3	Password Text Fields	78
2.8.4	Change Control Type.....	79
2.8.5	Multiple Primary Keys	79
CHAPTER 3	80
USER MANUAL	80
3.1	Relationships	80
3.2	Main Menu	81
3.3	Search Menu	83
3.3.1	Customers page	83
3.3.2	Bill Page	84
3.3.3	Employees Pages	85
3.3.4	Material Page	86
3.3.5	Cleaning Kind Page	87
CONCLUSION	88
REFERENCES	89
APPENDIX	91

INTRODUCTION

Many cleaning company can have many employees and many customers. Controlling these customers registration is not easy without a computer. If a company is big and have so many customers ,reaching its records by a computer provides so many advantages. For example the company can find out all the informations about its records that when the employees were worked ,what kind of cleaning materials were used,how much money has a customer dept , and like these; just typing a customer surname can be reached. Also The company can serve better and efficiently using the computer. But managing all these records is not easy. A good designed program can help the company about these better serving and efficiency.

Cleaning companies have to organize efficiently when a customer wants to serve. What kind of cleaning,how many employee will need,wich materials will be used have to be known to provide the best serve. In a program all these information can organize as correspond. Then after finish the work the program can calculate the customers balance and has to show the authorized person for these payment.

The objective of this project is to investigate the development of a data bank programming for cleaning companies. The project consist of introduction ,three chapters and conclusion.

Chapter one briefly explains what is delphi ,how we can use delphi and its components

Chapter two about Microsoft Access and creating new database. In chapter two we can see how tables are created ,how relationships are connected and important of integrity in a database programming

Chapter three explains database programming for a cleaning company and its components .

CHAPTER 1

1 DELPHI BASICS

1.1 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 A Brief History of Borland's Delphi

PascalDelphi uses the language Pascal, a third generation structured language. It is what is called a highly typed language. This promotes a clean, consistent programming style, and, importantly, results in more reliable applications. Pascal has a considerable heritage:

BeginningsPascal appeared relatively late in the history of programming languages. It probably benefited from this, learning from Fortran, Cobol and IBM's PL/1 that appeared in the early 1960's. Niklaus Wirth is claimed to have started developing Pascal in 1968, with a first implementation appearing on a CDC 6000 series computer in 1970.

Curiously enough, the C language did not appear until 1972. C sought to serve quite different needs to Pascal. C was designed as a high level language that still provided the low level access that assembly languages gave. Pascal was designed for the development of structured, maintainable applications.

The 1970'sIn 1975, Wirth teamed up with Jensen to produce the definitive Pascal reference book "Pascal User Manual and Report". Wirth moved on from Pascal in 1977 to work on Modula - the successor to Pascal.

The 1980'sIn 1982 ISO Pascal appears. The big event is in November 1983, when Turbo Pascal is released in a blaze of publicity. Turbo Pascal reaches release 4 by 1987. Turbo Pascal excelled on speed of compilation and execution, leaving the competition in its wake.

From Turbo Pascal to DelphiDelphi, Borland's powerful Windows? and Linux? programming development tool first appeared in 1995. It derived from the Turbo Pascal? product line.

As the opposition took heed of Turbo Pascal, and caught up, Borland took a gamble on an Object Oriented version, mostly based on the Pascal object orientation extensions. The risk paid off, with a lot of the success due to the thought underlying the design of the IDE (Integrated Development Environment), and the retention of fast compilation and execution.

This first version of Delphi was somewhat limited when compared to today's heavyweights, but succeeded on the strength of what it did do. And speed was certainly a key factor. Delphi went through rapid changes through the 1990's.

Delphi for Microsoft .Net. From that first version, Delphi went through 7 further iterations before Borland decided to embrace the competition in the form of the Microsoft? .Net architecture with the stepping stone Delphi 8 and then fully with Delphi 2005 and 2006. Delphi however still remains, in the opinion of the author, the best development tool for stand alone Windows and Linux applications. Pascal is a cleaner and much more disciplined language than Basic, and adapted much better to Object Orientation than Basic.

1.3 Writing your first Delphi program

Different types of application

Delphi allows you to create GUI (Graphical User Interface) or Console (text-only) applications (programs) along with many other types. We will concern ourselves here with the common, modern, GUI application.

Delphi does a lot of work for us - the programmer simply uses the mouse to click, drag, size and position graphical parts to build each screen of the application.

Each part (or element) can be passive (displaying text or graphics), or active (responding to a user mouse or keyboard action).

This is best illustrated with a very simple program.

Creating a simple 'Hello World' program

When you first run Delphi, it will prepare on screen a new graphical application. This comprises a number of windows, including the menu bar, a code editor, and the first screen (form) of our program. Do not worry about the editor window at the moment.

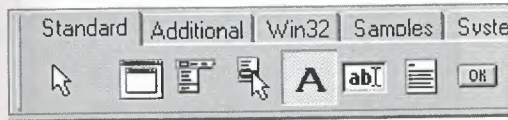
The form should look something like this :



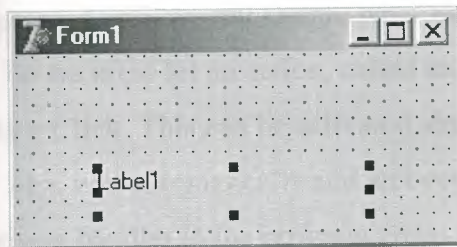
We have shown the form reduced in size for convenience here, but you will find it larger on your computer. It is a blank form, onto which we can add various controls and information. The menu window has a row of graphical items that you can add to the

form. They are in tabbed groups : **Standard**, **Additional**, **Win32** and so on.

We will select the simplest from the **Standard** collection. Click on the **A** image to select a Label. This **A** will then show as selected:



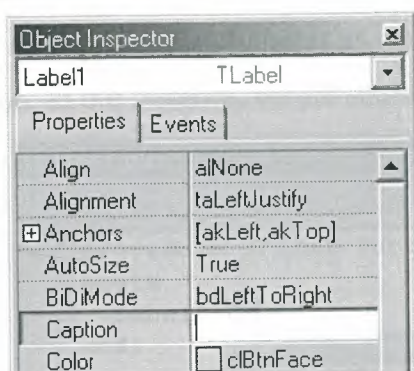
Having selected a graphical element, we then mark out on the form where we want to place the element. This is done by clicking and dragging. This gives us our first form element:



Changing graphical element properties

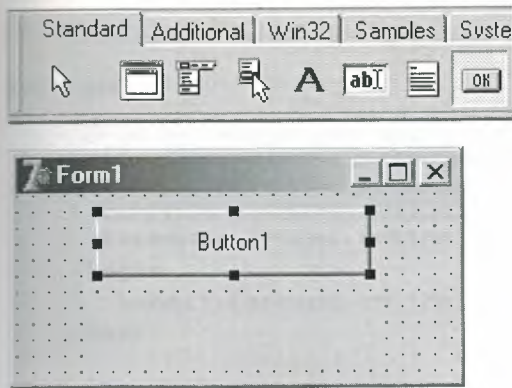
Notice that the graphical element contains the text **Label1** as well as resize corners. The text is called the **Caption**, and will appear when we run the application. This **Caption** is called a **Property** of the button. The label has many other properties such as height and width, but for now, we are only concerned with the caption.

Let us blank out the caption. We do this in the window called the **Object Inspector** (available under the **View** menu item if not already present):



Adding an active screen element

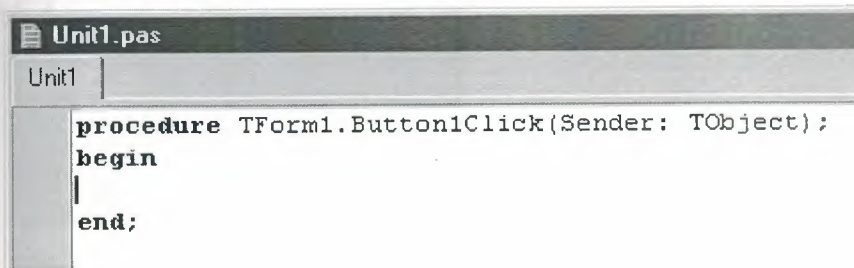
If we now return to the **Standard** graphical element collection, and select a button, shown as a very small button with **OK** on it, we can add this to the form as well:



We now have a label and a button on the form. But the button will do nothing when pressed until we tell Delphi what we want it to do.

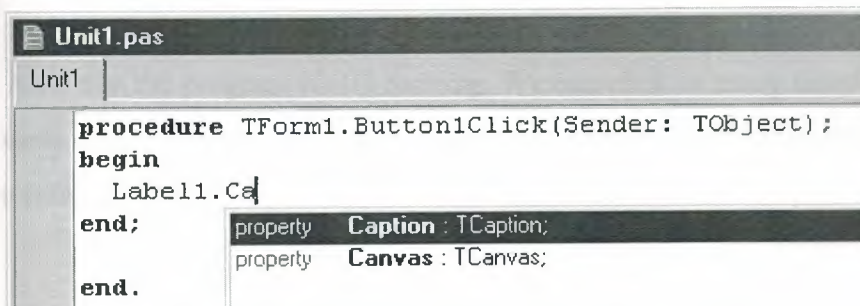
So we must set an action, called an **Event**, for the button. The main event for a button is a **Click**. This can be activated simply by double clicking the button on the form.

This will automatically add an event called **OnClick** for the button, and add a related event handler in the program code:



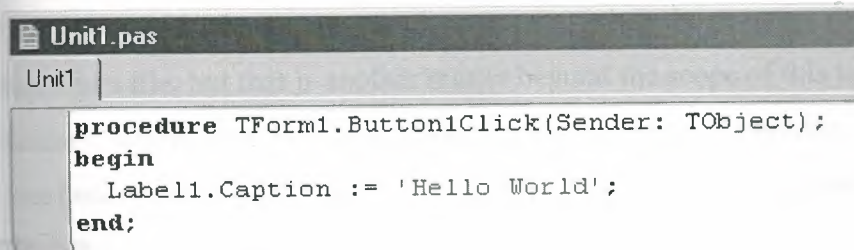
This 'skeleton' code will not do anything as it stands. We must add some code. Code that we add will run when the button is clicked. So let us change the label caption when the button is pressed.

As we type, Delphi helps us with a list of possible options for the item we are working on. In our instance, we are setting a Label caption:



Here you see that Delphi has listed all appropriate actions that start with **ca**. If we press **Enter**, Delphi will complete the currently selected item in the list. We assign a text value **'Hello World'** to the caption property. Note that we terminate this line of code with a **;** - all Delphi code statements end with this indicator. It allows us to write a

command spread across multiple lines - telling Delphi when we have finished the command.

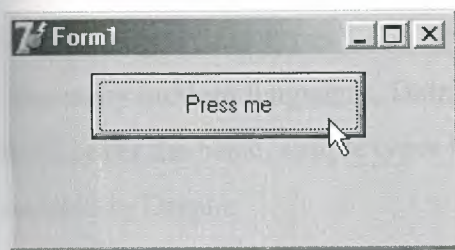


```
Unit1.pas
Unit1
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Caption := 'Hello World';
end;
```

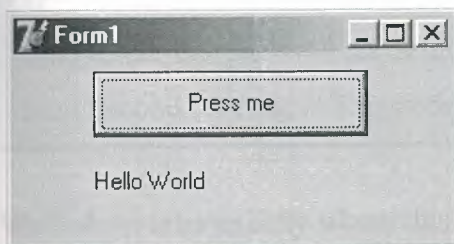
And we have now finished our very simple action - we will set the label to 'Hello World' when the button is pressed.

Running our first program

To run the program, we can click on the Green triangle (like a Video play button), or press **F9**. When the program runs it looks like this:



When we click on the button, we get:



and our program has set the Label text as we requested.

Note that the program is still running. We can click as many times as we like with the same outcome. Only when we close the program by clicking on the top right **X** will it terminate.

1.4 Delphi data types

Storing data in computer programs

For those new to computer programming, data and code go hand in hand. You cannot write a program of any real value without lines of code, or without data. A Word

Processor program has logic that takes what the user types and stores it in data. It also uses data to control how it stores and formats what the user types and clicks.

Data is stored in the memory of the computer when the program runs (it can also be stored in a file, but that is another matter beyond the scope of this tutorial). Each memory 'slot' is identified by a name that the programmer chooses. For example **LineTotal** might be used to name a memory slot that holds the total number of lines in a Word Processor document.

The program can freely read from and write to this memory slot. This kind of data is called a **Variable**. It can contain data such as a number or text. Sometimes, we may have data that we do not want to change. For example, the maximum number of lines that the Word Processor can handle. When we give a name to such data, we also give it its permanent value. These are called constants.

Simple Delphi data types

Like many modern languages, Delphi provides a rich variety of ways of storing data. We'll cover the basic, simple types here. Before we do, we'll show how to define a variable to Delphi:

```
var           // This starts a section of variables
  LineTotal : Integer; // This defines an Integer variable called LineTotal
  First,Second : String; // This defines two variables to hold strings of text
```

We'll show later exactly where this **var** section fits into your program. Notice that the variable definitions are indented - this makes the code easier to read - indicating that they are part of the **var** block.

Each variable starts with the name you choose, followed by a **:** and then the variable type. As with all Delphi statements, a **;** terminates the line. As you can see, you can define multiple variables in one line if they are of the same type.

It is very important that the name you choose for each variable is unique, otherwise Delphi will not know how to identify which you are referring to. It must also be different from the Delphi language keywords. You'll know when you have got it right when Delphi compiles your code OK (by hitting Ctrl-F9 to compile).

Delphi is not sensitive about the case (lower or upper) of your names. It treats

theCAT name the same as **TheCat**.

Number types

Delphi provides many different data types for storing numbers. Your choice depends on the data you want to handle. Our Word Processor line count is an unsigned Integer, so we might choose **Word** which can hold values up to 65,535. Financial or mathematical calculations may require numbers with decimal places - floating point numbers.

```
var
// Integer data types :
Int1 : Byte;    //          0 to 255
Int2 : ShortInt; //        -127 to 127
Int3 : Word;    //          0 to 65,535
Int4 : SmallInt; //       -32,768 to 32,767
Int5 : LongWord; //         0 to 4,294,967,295
Int6 : Cardinal; //         0 to 4,294,967,295
Int7 : LongInt; //       -2,147,483,648 to 2,147,483,647
Int8 : Integer; //       -2,147,483,648 to 2,147,483,647
Int9 : Int64; //  -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

// Decimal data types :
Dec1 : Single; // 7 significant digits, exponent -38 to +38
Dec2 : Currency; // 50+ significant digits, fixed 4 decimal places
Dec3 : Double; // 15 significant digits, exponent -308 to +308
Dec4 : Extended; // 19 significant digits, exponent -4932 to +4932
```

Some simple numerical variable useage examples are given below - fuller details on numbers is given in the [Numbers tutorial](#).

Text types

Like many other languages, Delphi allows you to store letters, words, and sentences in single variables. These can be used to display, to hold user details and so on. A letter is stored in a single character variable type, such as **Char**, and words and sentences stored in string types, such as **String**.

var

```
Str1 : Char;    // Holds a single character, small alphabet
Str2 : WideChar; // Holds a single character, International alphabet
Str3 : AnsiChar; // Holds a single character, small alphabet
Str4 : ShortString; // Holds a string of up to 255 Char's
Str5 : String;   // Holds strings of Char's of any size desired
Str6 : AnsiString; // Holds strings of AnsiChar's any size desired
Str7 : WideString; // Holds strings of WideChar's of any size desired
```

Some simple text variable usage examples are given below - fuller details on strings and characters is given in the [Text tutorial](#).

Logical data types

These are used in conjunction with programming logic. They are very simple:

var

```
Log1 : Boolean;    // Can be 'True' or 'False'
```

Boolean variables are a form of **enumerated** type. This means that they can hold one of a fixed number of values, designated by name. Here, the values can be **True** or **False**.

See the tutorials on [Logic](#) and [Looping](#) for further details.

Sets, enumerations and subtypes

Delphi excels in this area. Using sets and enumerations makes your code both easier to use and more reliable. They are used when categories of data are used. For example, you may have an enumeration of playing card suits. You literally enumerate the suit names. Before we can have an enumerated variable, we must define the enumeration values. This is done in a **type** section.

type

```
TSuit = (Hearts, Diamonds, Clubs, Spades); // Defines the enumeration
```

var

```
suit : TSuit;           // An enumeration variable
```

Sets are often confused with enumerations. The difference is tricky to understand. An

enumeration variable can have only one of the enumerated values. A set can have none, 1, some, or all of the set values. Here, the set values are not named - they are simply indexed slots in a numeric range. Confused? Well, here is an example to try to help you out. It will introduce a bit of code a bit early, but it is important to understand.

```
type
  TWeek = Set of 1..7;      // Set comprising the days of the week, by number
var
  week : TWeek;
begin
  week := [1,2,3,4,5];     // Switch on the first 5 days of the week
end;
```

See the [Set](#) reference, and the [Sets and enumerations tutorial](#) for further details. That tutorial introduces a further data type - a **subrange** type.

Using these simple data types

Variables can be read from and written to. This is called **assignment**. They can also be used in expressions and programming logic. See the [Text tutorial](#) and [Programming logic tutorial](#) for more about these topics.

Assigning to and from variables

Variables can be assigned from constant values, such as **23** and **'My Name'**, and also from other variables. The code below illustrates this assignment, and also introduces a further section of a Delphi program : the **const** (constants) section. This allows the programmer to give names to constant values. This is useful where the same constant is used throughout a program - a change where the constant is defined can have a global effect on the program.

Note that we use upper case letters to identify constants. This is just a convention, since Delphi is not case sensitive with names (it is with strings). Note also that we use **=** to define a constant value.

```
types
  TWeek = 1..7;           // Set comprising the days of the week, by number
```



```
TSuit = (Hearts, Diamonds, Clubs, Spades); // Defines an enumeration
```

const

```
FRED      = 'Fred';    // String constant
```

```
YOUNG_AGE  = 23;       // Integer constant
```

```
TALL : Single = 196.9;  // Decimal constant
```

```
NO         = False;    // Boolean constant
```

var

```
FirstName, SecondName : String; // String variables
```

```
Age          : Byte;    // Integer variable
```

```
Height       : Single;  // Decimal variable
```

```
IsTall       : Boolean; // Boolean variable
```

```
OtherName    : String;  // String variable
```

```
Week         : TWeek;   // A set variable
```

```
Suit         : TSuit;   // An enumeration variable
```

begin // Begin starts a block of code statements

```
FirstName := FRED;      // Assign from predefined constant
```

```
SecondName := 'Bloggs'; // Assign from a literal constant
```

```
Age      := YOUNG_AGE;  // Assign from predefined constant
```

```
Age      := 55;         // Assign from constant - overrides YOUNG_AGE
```

```
Height   := TALL - 5.5; // Assign from a mix of constants
```

```
IsTall   := NO;         // Assign from predefined constant
```

```
OtherName := FirstName; // Assign from another variable
```

```
Week     := [1,2,3,4,5]; // Switch on the first 5 days of the week
```

```
Suit     := Diamonds;   // Assign to an enumerated variable
```

end; // End finishes a block of code statements

FirstName is now set to **'Fred'**

SecondName is now set to **'Bloggs'**

Age is now set to **55**

Height is now set to **191.4**

IsTall is now set to **False**
OtherName is now set to **'Fred'**
Week is now set to **1,2,3,4,5**
Suit is now set to **Diamonds** (Notice no quotes)

Note that the third constant, **TALL**, is defined as a Single type. This is called a **typed constant**. It allows you to force Delphi to use a type for the constant that suits your need. Otherwise, it will make the decision itself.

Compound data types

The simple data types are like single elements. Delphi provides compound data types, comprising collections of simple data types.

These allow programmers to group together variables, and treat this group as a single variable. When we discuss programming logic, you will see how useful this can be.

Arrays

Array collections are accessed by index. An array holds data in indexed 'slots'. Each slot holds one variable of data. You can visualise them as lists. For example:

```
var
  Suits : array[1..4] of String; // A list of 4 playing card suit names

begin
  Suits[1] := 'Hearts'; // Assigning to array index 1
  Suits[2] := 'Diamonds'; // Assigning to array index 2
  Suits[3] := 'Clubs'; // Assigning to array index 3
  Suits[4] := 'Spades'; // Assigning to array index 4
end;
```

The array defined above has indexes 1 to 4 (1..4). The two dots indicate a range. We have told Delphi that the array elements will be string variables. We could equally have defined integers or decimals.

For more on arrays, see the [Arrays tutorial](#).

Records

Records are like arrays in that they hold collections of data. However, records can hold a mixture of data types. They are a very powerful and useful feature of Delphi, and one that distinguishes Delphi from many other languages.

Normally, you will define your own record structure. This definition is not itself a variable. It is called a data **type** (see Types for further on this). It is defined in a **type** data section. By convention, the record type starts with a **T** to indicate that it is a type not real data (types are like templates). Let us define a customer record:

```
type  
  TCustomer Record  
    firstName : string[20];  
    lastName  : string[20];  
    age       : byte;  
end;
```

Note that the strings are suffixed with **[20]**. This tells Delphi to make a fixed space for them. Since strings can be a variable length, we must tell Delphi so that it can make a record of known size. Records of one type always take up the same memory space.

Let us create a record variable from this record type and assign to it:

```
var  
  customer : TCustomer;      // Our customer variable  
begin  
  customer.firstName := 'Fred'; // Assigning to the customer record  
  customer.lastName  := 'Bloggs';  
  customer.age       := 55;  
end;
```

```
customer.firstName is now set to 'Fred'  
customer.lastName  is now set to 'Bloggs'  
customer.age       is now set to 55
```

Notice how we do not use an index to refer to the record elements. Records are very

friendly - we use the record element by its name, separated from the record name by a qualifying dot. See the [Records tutorial](#) for further on records.

Objects

Objects are collections of both data and logic. They are like programs, but also like data structures. They are the key part of the **Object oriented** nature of Delphi. See the [Object orientation tutorial](#) for more on this advanced topic.

Other data types

The remaining main object types in Delphi are a mixed bunch:

Files

File variables represent computer disk files. You can read from and write to these files using file access routines. This is a complex topic covered in [Files](#).

Pointers

Pointers are also the subject of an advanced topic - see [Pointer](#) reference. They allow variables to be indirectly referenced.

Variants

Variants are also an advanced topic - see [Variant](#). They allow the normal Delphi rigid type handling to be avoided. Use with care!

1.5 Programming logic

What is programming logic?

Programming in Delphi or any other language would not work without logic. Logic is the glue that holds together the code, and controls how it is executed. For example, supposing we were writing a word processor program. When the user presses the Enter key, we will move the cursor to a new line. The code would have a logical test for the user hitting the Enter key. If hit we do a line throw, if not, we continue on the same line.

If then else

In the above example, we might well use the **If** statement to check for the Enter key.

Simple if then else

Here is an example of how the if statement works:


```

var
  number : Integer;
  text : String;
begin
  number := Sqr(17);      // Calculate the square of 17
  if number > 400
  then text := '17 squared > 400' // Action when if condition is true
  else text := '17 squared <= 400'; // Action when if condition is false
end;

```

```

text is set to : '17 squared <= 400'

```

There are a number of things to note about the if statement. First that it spans a few lines - remember that Delphi allows statements to span lines - this is why it insists on a terminating ;

Second, that the **then** statement does not have a terminating ; -this is because it is part of the **if** statement, which is finished at the end of the **else** clause.

Third, that we have set the value of a text string when the **If** condition is successful - the **Then** clause - and when unsuccessful - the **Else** clause. We could have just done a **then** assignment:

```

if number > 400
then text := '17 squared > 400';

```

Note that here, the **then** condition is not executed (because 17 squared is not > 400), but there is no **else** clause. This means that the **if** statement simply finishes without doing anything.

Note also that the **then** clause now has a terminating ; to signify the end of the **if** statement.

Compound if conditions, and multiple statements

We can have multiple conditions for the **if** condition. And we can have more than one statement for the **then** and **else** clauses. Here are some examples:

```

if (condition1) And (condition2) // Both conditions must be satisfied
then
  begin
    statement1;
    statement2;
    ...
  end      // Notice no terminating ';' - still part of 'if'
else
  begin
    statement3;
    statement4;
    ...
  end;

```

We used **And** to join the if conditions together - both must be satisfied for the then clause to execute. Otherwise, the else clause will execute. We could have used a number of different logical primitives, of which **And** is one, covered under logical primitives below.

Nested if statements

There is nothing to stop you using if statements as the statement of an if statement. Nesting can be useful, and is often used like this:

```

if condition1
  then statement1
  else if condition2
    then statement2
    else statement3;

```

However, too many nested if statements can make the code confusing. The **Case** statement, discussed below, can be used to overcome a lot of these problems.

Logicial primitives

Before we introduce these, it is appropriate to introduce the Boolean data type. It is an enumerated type, that can have one of only two values : **True** or **False**. We will use it in place of a condition in the if clauses below to clarify how they work:

```
begin
  if false And false
  then ShowMessage('false and false = true');

  if true And false
  then ShowMessage('true and false = true');

  if false And true
  then ShowMessage('false and true = true');

  if true And true
  then ShowMessage('true and true = true');

  if false Or false
  then ShowMessage('false or false = true');

  if true Or false
  then ShowMessage('true or false = true');

  if false Or true
  then ShowMessage('false or true = true');

  if true Or true
  then ShowMessage('true or true = true');

  if false Xor false
  then ShowMessage('false xor false = true');

  if true Xor false
```

```

then ShowMessage('true xor false = true');

if false Xor true
then ShowMessage('false xor true = true');

if true Xor true
then ShowMessage('true xor true = true');

if Not false
then ShowMessage('not false = true');

if Not true
then ShowMessage('not true = true');
end;

```

```

true and true = true
false or true = true
true or false = true
true or true = true
false xor true = true
true xor false = true
not false = true

```

Note that the **Xor** primitive returns true when one, but not both of the conditions are true.

Click on the primitives in blue above to learn how they can also be used for mathematical (bitwise) calculations.

Case statements

The **If** statement is useful when you have a simple two way decision. Either you go one way or another way. Case statements are used when you have a set of 3 or more alternatives.

A simple numerical case statement


```

var
  i : Integer;
begin
  i := RandomRange(15,20); // Generate a random number from 15 to 20
  Case i of
    15 : ShowMessage('Random number was fifteen');
    16 : ShowMessage('Random number was sixteen');
    17 : ShowMessage('Random number was seventeen');
    18 : ShowMessage('Random number was eighteen');
    19 : ShowMessage('Random number was nineteen');
    20 : ShowMessage('Random number was twenty');
  end;
end;

```

Random number was fifteen

The **RandomRange** routine generates a random number between two given values. However, each time you run the program, it will always start with the same pseudo random value (unless you use RandomSeed).

The case statement above routes the processing to just one of the statements. OK, the code is a bit silly, but it is used to illustrate the point.

Using the otherwise clause

Supposing we were not entirely sure what value our case statement was processing? Or we wanted to cover a known set of values in one fell swoop? The **Else** clause allows us to do that:

```

var
  i : Integer;
begin
  i := RandomRange(10,20); // Generate a random number from 10 to 20
  Case i of
    15 : ShowMessage('Random number was fifteen');
    16 : ShowMessage('Random number was sixteen');
  end;
end;

```

```

17 : ShowMessage('Random number was seventeen');
18 : ShowMessage('Random number was eighteen');
19 : ShowMessage('Random number was nineteen');
20 : ShowMessage('Random number was twenty');
else
  ShowMessageFmt('Unexpected number : %d',[i]);
end;
end;

```

```
Unexpected number : 10
```

Using enumeration case values

Just as with the **If** statement, the **Case** statement may use any ordinal type. This allows us to use the very readable enumeration type:

```

type
  TCar = (Nissan, Ford, Rover, Jaguar);  // An enumeration type
var
  car : TCar;                          // An enumeration variable
begin
  car := Rover;                        // Set this variable
  case car of
    Nissan : ShowMessage('We have a Nissan car');
    Ford   : ShowMessage('We have a Ford car');
    Rover  : ShowMessage('We have a Rover car');
    Jaguar : ShowMessage('We have a Jaguar car');
  end;
end;

```

```
We have a Rover car
```


1.6 Repeating sets of commands

Why loops are used in programming

One of the main reasons for using computers is to save the tedium of many repetitive tasks. One of the main uses of loops in programs is to carry out such repetitive tasks. A loop will execute one or more lines of code (statements) as many times as you want.

Your choice of loop type depends on how you want to control and terminate the looping.

The For loop

This is the most common loop type. **For** loops are executed a fixed number of times, determined by a count. They terminate when the count is exhausted. The count (loop) is held in a variable that can be used in the loop. The count can proceed upwards or downwards, but always does so by a value of 1 unit. This count variable can be a number or even an enumeration.

Counting up

Here is a simple example counting up using numeric values:

```
var
  count : Integer;
begin
  For count := 1 to 5 do
    ShowMessageFmt('Count is now %d',[count]);
  end;
```

```
Count is now 1
Count is now 2
Count is now 3
Count is now 4
Count is now 5
```

The ShowMessageFmt routine is useful for displaying information - click on it to read more.

Counting up using an enumeration

Enumerations (see [Enumeration](#) and [sets](#) to explore) are very readable ways of assigning values to variables by name. They can also be used to control For loops:

type

```
TWeekDay = (Monday=1, Tuesday, Wednesday, Thursday, Friday);
```

var

```
weekday : TWeekDay;
```

```
hours : array[TWeekDay] of byte;
```

begin

```
// Set up the hours every day to zero
```

```
for weekDay := Monday to Friday do
```

```
    hours[weekDay] := 0;
```

```
// Add an hour of overtime to the working hours on Tuesday to Thursday
```

```
for weekDay := Tuesday to Thursday do
```

```
    Inc(hours[weekDay]);
```

end;

```
hours[Monday] = 0
```

```
hours[Tuesday] = 1
```

```
hours[Wednesday] = 1
```

```
hours[Thursday] = 1
```

```
hours[Friday] = 0
```

Note the use of the **Inc** routine to increment the hours.

Counting down, using characters

We can also use single letters as the count type, because they are also ordinal types:

var

```
letter : Char;
```

begin

```
for letter := 'G' downto 'A' do
```

```
    ShowMessage('Letter = '+letter)
```

```
end;
```

```
Letter = G
```

```
Letter = F
```

```
Letter = E
```

```
Letter = D
```

```
Letter = C
```

```
Letter = B
```

```
Letter = A
```

The For statements in the examples above have all executed one statement. If you want to execute more than one, you must enclose these in a **Begin** and **End** pair.

The Repeat loop

The Repeat loop type is used for loops where we do not know in advance how many times we will execute. For example, when we keep asking a user for a value until one is provided, or the user aborts. Here, we are more concerned with the loop termination condition.

Repeat loops always execute at least once. At the end, the **Until** condition is checked, and the loop aborts if condition works out as true.

A simple example

```
var
  exit : Boolean;      // Our exit condition flag
  i : Integer;
begin
  i := 1;
  exit := False;      // do not exit until we are ready
  repeat
    Inc(i);           // Increment a count
    if Sqr(i) > 99
    then exit := true; // Exit if the square of our number exceeds 99
  until exit;         // Shorthand for 'until exit := true'
end;
```


Upon exit, i will be 10 (since $\text{Sqr}(10) > 99$)

Here we exit the repeat loop when a Boolean variable is true. Notice that we use a shorthand - just specifying the variable as the condition is sufficient since the variable value is either true or false.

Using a compound condition

```
var
  i : Integer;
begin
  i := 1;
  repeat
    Inc(i);          // Increment a count
  until (Sqr(i) > 99) or (Sqrt(i) > 2.5);
end;
```

Upon exit, i will be 7 (since $\text{Sqrt}(7) > 2.5$)

Notice that compound statements require separating brackets. Notice also that Repeat statements can accomodate multiple statements without the need for a begin/end pair. The repeat and until clauses form a natural pairing.

While loops

While loops are very similar to Repeat loops except that they have the exit condition at the start. This means that we use them when we wish to avoid loop execution altogether if the condition for exit is satisfied at the start.

```
var
  i : Integer;
begin
  i := 1;
  while (Sqr(i) <= 99) and (Sqrt(i) <= 2.5) do
    Inc(i);          // Increment a count
```



```
end;
```

```
Upon exit, i will be 7 (since Sqrt(7) > 2.5)
```

Notice that our original Repeat Until condition used Or as the compound condition joiner - we continued until either condition was met. With our While condition, we use And as the joiner - we continue whilst **neither** condition is met. Have a closer look to see why we do this. The difference is that we repeat an action until something or something else happens. Whereas we keep doing an action while neither something nor something else have happened.

1.7 Dates and times

Why have a tutorial just on dates and times?

Because they are a surprisingly complex and rich subject matter. And very useful, especially since Delphi provides extensive support for calculations, conversions and names.

The TDateTime data type

Date and time processing depends on the **TDateTime** variable. It is used to hold a date and time combination. It is also used to hold just date or time values - the time and date value is ignored respectively. TDateTime is defined in the **System** unit. Date constants and routines are defined in **SysUtils** and **DateUtils** units.

Let us look at some simple examples of assigning a value to a TDateTime variable:

```
var
    date1, date2, date3 : TDateTime;    // TDateTime variables
begin
    date1 := Yesterday;    // Set to the start of yesterday
    date2 := Date;         // Set to the start of the current day
    date3 := Tomorrow;     // Set to the start of tomorrow
    date4 := Now;          // Set to the current day and time
end;
```

```
date1 is set to something like 12/12/2002 00:00:00
date2 is set to something like 13/12/2002 00:00:00
date3 is set to something like 14/12/2002 00:00:00
date4 is set to something like 13/12/2002 08:15:45
```

Note : the start of the day is often called midnight in Delphi documentation, but this is misleading, since it would be midnight of the wrong day.

Some named date values

Delphi provides some useful day and month names, saving you the tedium of defining them in your own code. Here they are:

Short and long month names

Note that these month name arrays start with index = 1.

```
var
  month : Integer;
begin
  for month := 1 to 12 do // Display the short and long month names
  begin
    ShowMessage(ShortMonthNames[month]);
    ShowMessage(LongMonthNames[month]);
  end;
end;
```

The ShowMessage routine display the following information:

```
Jan
January
Feb
February
Mar
March
```


Apr
April
May
May
Jun
June
Jul
July
Aug
August
Sep
September
Oct
October
Nov
November
Dec
December

Short and long day names

It is important to note that these day arrays start with index 1 = Sunday. This is not a good standard (it is not **ISO 8601** compliant), so be careful when using with ISO 8601 compliant routines such as DayOfTheWeek

```
var
  day : Integer;
begin
  for day := 1 to 12 do // Display the short and long day names
  begin
    ShowMessage(ShortDayNames[day]);
    ShowMessage(LongDayNames[day]);
  end;
end;
```


The **ShowMessage** routine display the following information:

Sun

Sunday

Mon

Monday

Tue

Tuesday

Wed

Wednesday

Thu

Thursday

Fri

Friday

Sat

Saturday

Date and time calculations

The largest benefit of **TDateTime** is the range of calculations Delphi can do for you.

These can be found on the Delphi Basics home page, in the **Dates and Times/Calculations** option.

In the following examples, click on the name to learn more:

DayOfTheMonth Gives the day of month index for a **TDateTime** value

DaysBetween Gives the whole number of days between 2 dates

DaysInAMonth Gives the number of days in a month

DaysInAYear Gives the number of days in a year

DecodeDate Extracts the year, month, day values from a **TDateTime** var.

EncodeDate Build a **TDateTime** value from year, month and day values

IncDay Increments a **TDateTime** variable by + or - number of days

IsLeapYear Returns true if a given calendar year is a leap year

<u>MinutesPerDay</u> Gives the number of minutes in a day

Displaying date and time values

There are a number of routines that convert date and or time values to strings for display or file storage purposes, such as dateTimeToStr and TimeToString. But the most important is the FormatDateTime. It provides comprehensive formatting control, as illustrated by the following examples.

Using default formatting options

```
var
myDate : TDateTime;

begin
// Set up our TDateTime variable with a full date and time :
// 09/02/2000 at 05:06:07.008 (.008 milli-seconds)
myDate := EncodeDateTime(2000, 2, 9, 5, 6, 7, 8);

// Date only - numeric values with no leading zeroes (except year)
ShowMessage('      d/m/y = '+'
FormatDateTime('d/m/y', myDate));

// Date only - numeric values with leading zeroes
ShowMessage('      dd/mm/yy = '+'
FormatDateTime('dd/mm/yy', myDate));

// Use short names for the day, month, and add freeform text ('of')
ShowMessage(' ddd d of mmm yyyy = '+'
FormatDateTime('ddd d of mmm yyyy', myDate));

// Use long names for the day and month
ShowMessage('dddd d of mmmm yyyy = '+'
FormatDateTime('dddd d of mmmm yyyy', myDate));
```



```

// Use the ShortDateFormat settings only
ShowMessage('          dddddd = '+
             FormatDateTime('ddddd', myDate));

// Use the LongDateFormat settings only
ShowMessage('          ddddddd = '+
             FormatDateTime('dddddd', myDate));

ShowMessage("");

// Time only - numeric values with no leading zeroes
ShowMessage('          h:n:s.z = '+
             FormatDateTime('h:n:s.z', myDate));

// Time only - numeric values with leading zeroes
ShowMessage('          hh:nn:ss.zzz = '+
             FormatDateTime('hh:nn:ss.zzz', myDate));

// Use the ShortTimeFormat settings only
ShowMessage('          t = '+FormatDateTime('t', myDate));

// Use the LongTimeFormat settings only
ShowMessage('          tt = '+FormatDateTime('tt', myDate));

// Use the ShortDateFormat + LongTimeFormat settings
ShowMessage('          c = '+FormatDateTime('c', myDate));
end;

```

The ShowMessage routine shows the following outputs :

```

d/m/y = 9/2/00
dd/mm/yy = 09/02/00
ddd d of mmm yyyy = Wed 9 of Feb 2000

```


dddd d of mmmm yyyy = Wednesday 9 of February 2000

dddddd = 09/02/2000

dddddd = 09 February 2000

c = 09/02/2000 01:02:03

h:n:s.z = 1:2:3.4

hh:nn:ss.zzz = 01:02:03.004

t = 01:02

tt = 01:02:03

c = 09/02/2000 01:02:03

The above output uses default values of a number of formatting control variables. These are covered in the next section:

Formatting control variables

The variables and their default values are given below. Note that these control conversions of date time values to strings, and sometimes from strings to date time values (such as **DateSeparator**).

<u>DateSeparator</u>	= /
<u>TimeSeparator</u>	= :
<u>ShortDateFormat</u>	= dd/mm/yyyy
<u>LongDateFormat</u>	= dd mmm yyyy
<u>TimeAMString</u>	= AM
<u>TimePMString</u>	= PM
<u>ShortTimeFormat</u>	= hh:mm
<u>LongTimeFormat</u>	= hh:mm:ss
<u>ShortMonthNames</u>	= Jan Feb ...
<u>LongMonthNames</u>	= January, February ...
<u>ShortDayNames</u>	= Sun, Mon ...
<u>LongDayNames</u>	= Sunday, Monday ...
<u>TwoDigitYearCenturyWindow</u>	= 50

1.8 Standard tab GUI components

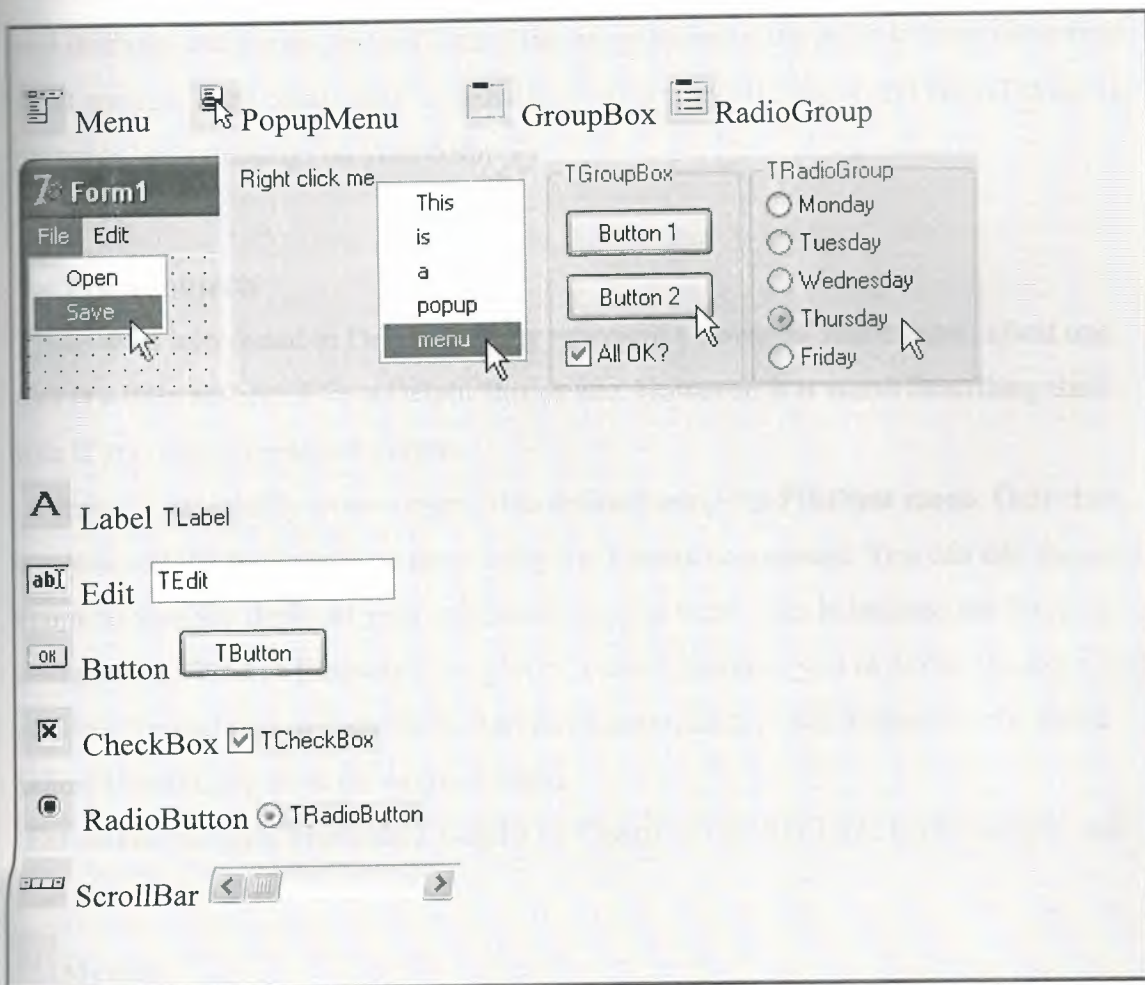
GUI components

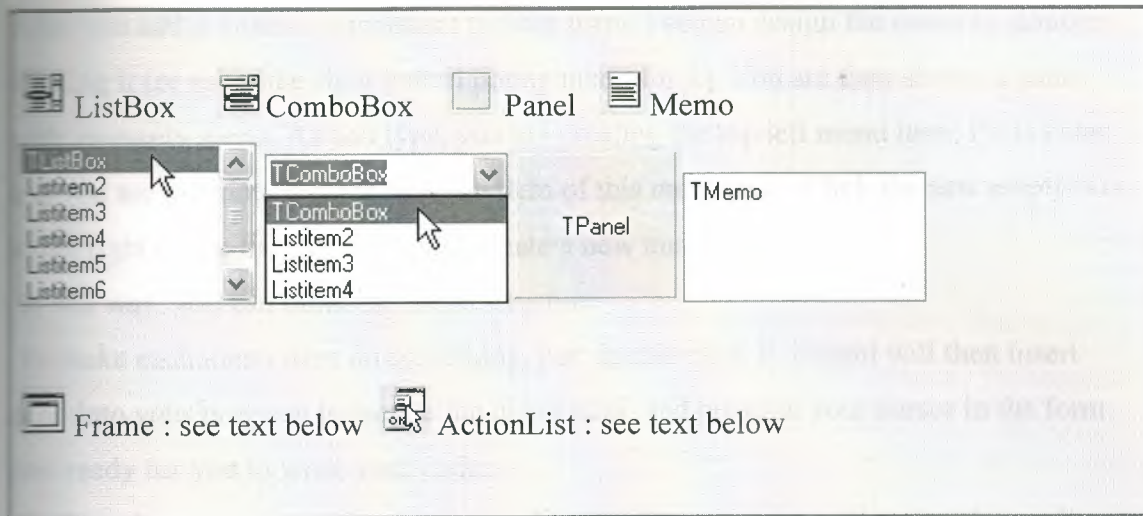
GUI stands for Graphical User Interface. It refers to the windows, buttons, dialogs, menus and everything visual in a modern application. A GUI component is one of these graphical building blocks. Delphi lets you build powerful applications using a rich variety of these components.

These components are grouped under a long set of tabs in the top part of the Delphi screen, starting with **Standard** at the left. We'll look at this Standard tab here. It looks something like this (Delphi allows you to tinker with nearly everything in its interface, so it may look different on your system):



Each of the components is itemised below with a picture of a typical GUI object they can create:





Note that the displayed components were taken from an XP computer. In order to get the new XP look (the XP 'themed' GUI look), you must add the **XP Manifest** component to you form. It is found under the **Win32** component tab:

 XP Manifest component.

We'll now cover each of the components in turn. Components have many properties and methods and events, but we'll keep the descriptions to the point to keep this article short enough. Each component is added to you form by clicking it and then clicking (or dragging and releasing) on your form.

Frame objects

These were introduced in Delphi 5. They represent a powerful mechanism, albeit one that is a little advanced for a Delphi Basics site. However, it is worth describing their role if you want to research further.

A frame is essentially a new object. It is defined using the **File|New** menu. Only then can you add the frame to your form using the **Frame** component. You can add the same frame to as many forms of your application as you want. This is because the frame is designed as a kind of template for a part of a form. It allows you to define the same look and feel for that part of each form. And more importantly, each instance of the frame inherits everything from the original frame.

For further reading, **Mastering Delphi** by **Cantu** covers this topic with example code.

Menus

After you add a TMenu component to your form, you can design the menu by double clicking it (or using the right button popup menu for it). You are then shown a panel with an empty menu. As you type, you are creating the top left menu item. Press enter and you are positioned at the first sub item of this menu item. Click the new empty box to the right of the first menu item to create a new menu item.

In this way, you can build the menu structure.

To make each menu item do something, just double click it. Delphi will then insert code into your program to handle the menu item, and position your cursor in the form unit ready for you to write your code.

Explore the popup menu for the menu editor to discover more options, such as sub-menus.

A menu can also be dynamically updated by your code.



Popup menus

A popup menu appears in many applications when you right click on something. For example, when you right click the Windows desktop. You create a popup menu by adding the popup menu component to your form and double clicking it. You then simply type in your menu item list.

You attach the popup menu to an existing form object (or the form itself) by selecting your new popup menu in the **PopupMenu** property of the object.

To activate the popup menu items, double click each in turn. Delphi will add the appropriate code to your form unit. You can then type in the code that each menu item should perform.

A popup menu can also be dynamically updated by your code.

A Labels

Labels are the simplest component. They are used to literally label things on a form, but the text, colours and so on can be changed by your code. For example, you can change the label colour when the mouse hovers over it, and can run code when the user clicks it. This makes the label like a web page link. Normally, they are just kept as plain, unchanging text.



Edit boxes

An edit box allows the user to type in a single line of text. For example, the name of the user. You set up the initial value with the **Text** property either at design time or when your code runs.



Memo boxes

A memo box displays a single string as a multi line wrapped text display. You cannot apply any formatting. The displayed lines are set using the **Lines** property. This may be set at design time as well as at run time.



Buttons

A button is the simplest active item. When clicked by a user, it performs some action. You can change the button label by setting the **Caption** property. Double clicking the button when designing adds code to your form to run when the button is clicked at run time.



Check boxes

Check boxes are used to give a user a **yes/no** choice. For example, whether to wrap text or not. The label is set using the **Caption** property. You can preset the check box to ticked by setting the **Checked** property to **true**.



Radio buttons

Radio buttons are used to give a user **multiple** choices. For example, whether to left, centre or right align text. The label is set using the **Caption** property. You can preset a radio button to selected by setting the **Checked** property to **true**.

You would normally use radio buttons in groups of two or more. The **TRadioGroup** component allows you to do this in a neat and dynamic way.



List boxes

List boxes provide selectable items. For example, a collection of fish names. If you set the **MultiSelect** property to **true**, you allow the user to select more than one. The items in the list are added using the **Items.Add** method, passing the string of each item as a parameter.

You can act upon an item being selected by setting the **OnClick** event (by double clicking it) to a procedure in your form unit.

The following example displays the selected list item in a dialog box:

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
  listBox : TListBox;
  index   : Integer;
begin
  // Cast the passed object to its correct type
  listBox := TListBox(Sender);

  // Get the index of the selected list item
  index := listBox.ItemIndex;

  // Display the selected list item value
  ShowMessage(listBox.Items[index]);
end;
```



Combo boxes

A combo box is like a list box, and is set up in the same way (see above). It just takes up less space on your form by collapsing to a single line when deselected, showing the chosen list item. It is not recommended to use one for multi line selection.



Scroll bars

Many components have built in scroll bars. For those that don't, you can use this to do your own scrolling. You link the scrollbar to your component by setting the **OnScroll** event. This gives you the details of the last scroll activity made by the user.



Group boxes

A group box is like a panel. It differs in that it gives a name to the collection of components that you add to it. This title is set with the **Caption** property. Use a group box to help the user see what controls affect one particular aspect of the application.



Radio group panels

Radio buttons are used to give a user a **multiple** choices. For example, whether to left, centre or right align text. Unlike individual radio buttons, a group is only set up by your code. You define the buttons by calling the **Items.Add** method of the **TRadioGroup** object, passing the caption string of each radio button as a parameter. You can reference each button by using the **Buttons** indexed property. You might, for example, choose the third button to be checked. For example :

```
// Set the third button to be pre-selected (index starts at 0)
RadioGroup1.Buttons[2].Checked := true;
```



Empty panels

When building your form, you might want to add many components. These may fall into logical groups. If so, you can add each group to a panel, and use the panel to position the whole group on the form. The panel name can be blanked out by setting the **Caption** property.

You can even hide the panel by setting the **BevelOuter** and **BevelInner** properties to **bvNone**.



Action lists

Action lists are a large topic on their own. They allow you to define, for example, menus with sub-items that are also shown as buttons on your application. Only one **action** is defined, regardless of the number of references to it.

For further reading, **Mastering Delphi** by **Cantu** covers this topic.

CHAPTER 2

2 MICROSOFT ACCESS

2.1 Getting Started

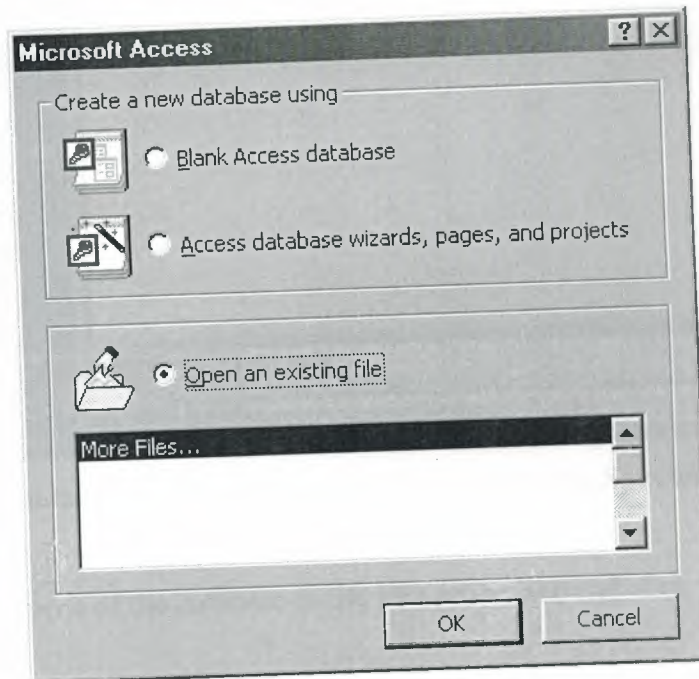
2.1.1 A Few Terms

These words are used often in Access so you will want to become familiar with them before using the program and this tutorial.

- A **database** is a collection of related information.
- An **object** is a competition in the database such as a table, query, form, or macro.
- A **table** is a grouping of related data organized in fields (columns) and records (rows) on a datasheet. By using a common field in two tables, the data can be combined. Many tables can be stored in a single database.
- A **field** is a column on a datasheet and defines a data type for a set of values in a table. For a mailing list table might include fields for first name, last name, address, city, state, zip code, and telephone number.
- A **record** in a row on a datasheet and is a set of values defined by fields. In a mailing list table, each record would contain the data for one person as specified by the intersecting fields.
- **Design View** provides the tools for creating fields in a table.
- **Datasheet View** allows you to update, edit, and delete in formation from a table.

2.1.2 Getting Started

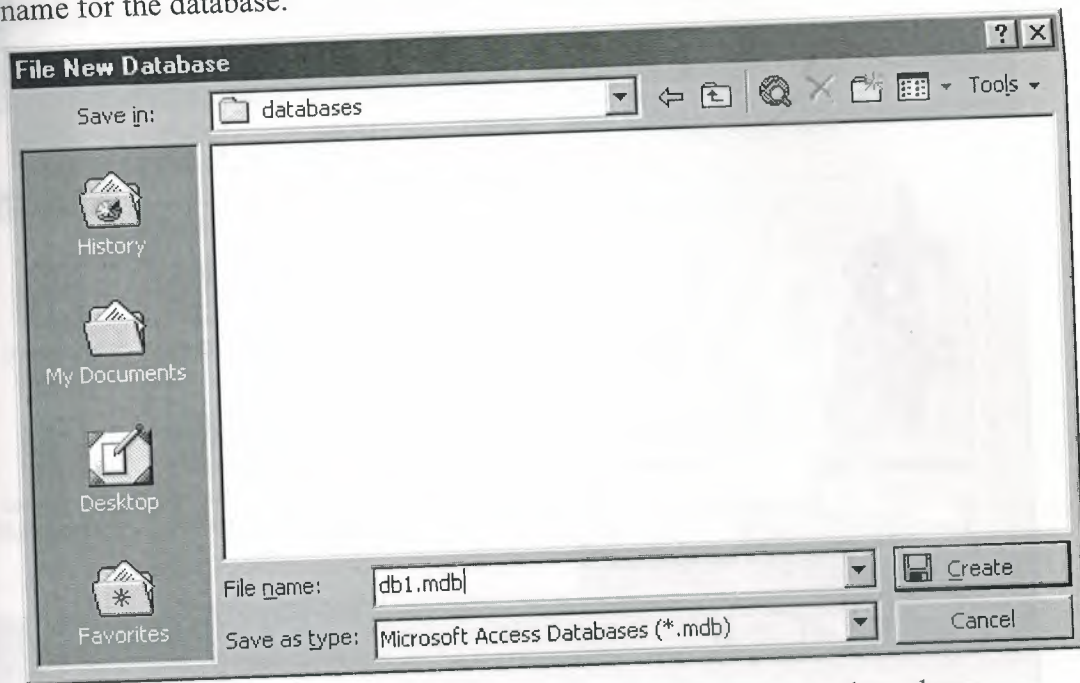
After opening Access, you will be presented with the window shown below. Select one of the first two options if you are creating a new database, or the third if you want to edit an existing database. All three choices are explained in detail below.



2.1.3 Blank Access database

1. Unlike Word documents, Excel worksheets, and Power Point presentations, you must save an Access database before you start working on it. After selecting "Blank Access database", you will first be prompted to specify a location and

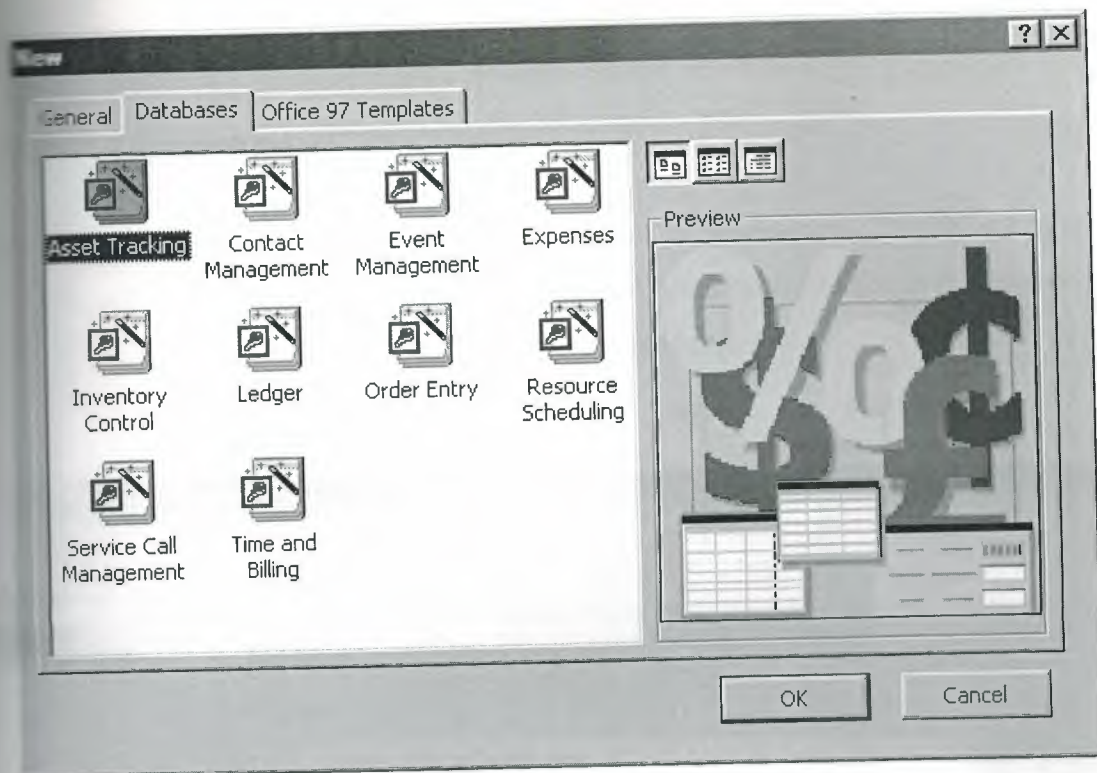
name for the database.



2. Find the folder where the database should reside in the **Save in** drop-down menu.
3. Type the name of the database in the **File name** line and click the **Create** button.

2.1.4 Access database wizards, pages, and projects

Access' wizards and layout are existing database structures that only need data input. Select a database type and click **OK**. Name the database on the next screen.



2.1.5 Open an existing database

If the database was opened recently on the computer, it will be listed on the main window. Highlight the database name and click **OK**. Otherwise, highlight "More Files..." in the list and click **OK**. From the subsequent window, click the "Look In:" drop-down menu to find the folder where the database is located, highlight the database name in the listing and click **OK**.

2.1.6 Converting to Access 2000

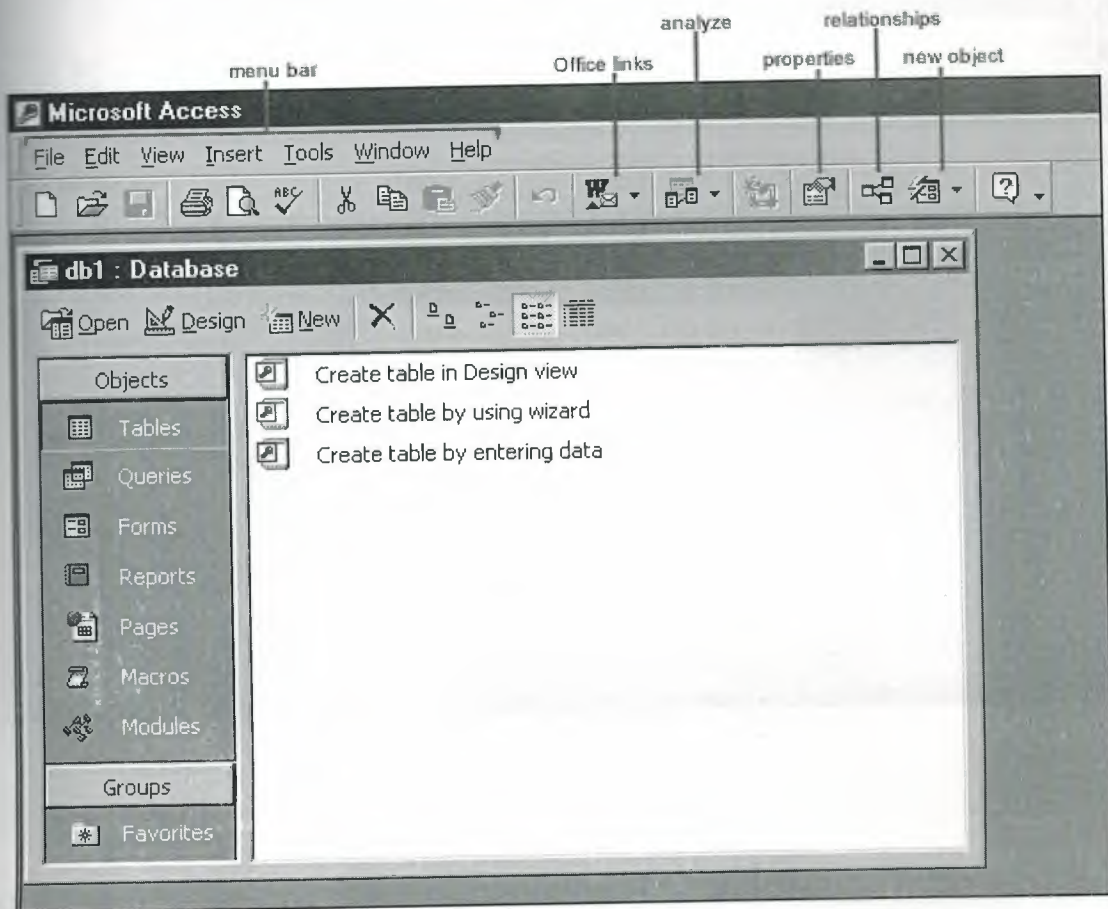
Before opening an existing file that was created in a previous version of Access, it must first be converted to Access 2000 format. Convert a database by following these steps:

1. Open Access and select **Tools|Database Utilities|Convert Database|To Current Access Database Version** from the menu bar.
2. Select the database that should be converted and click the **Convert** button.
3. The new version will be a completely separate database and the old one will remain intact so you must then name the new version of the database.

2.2 Screen Layouts

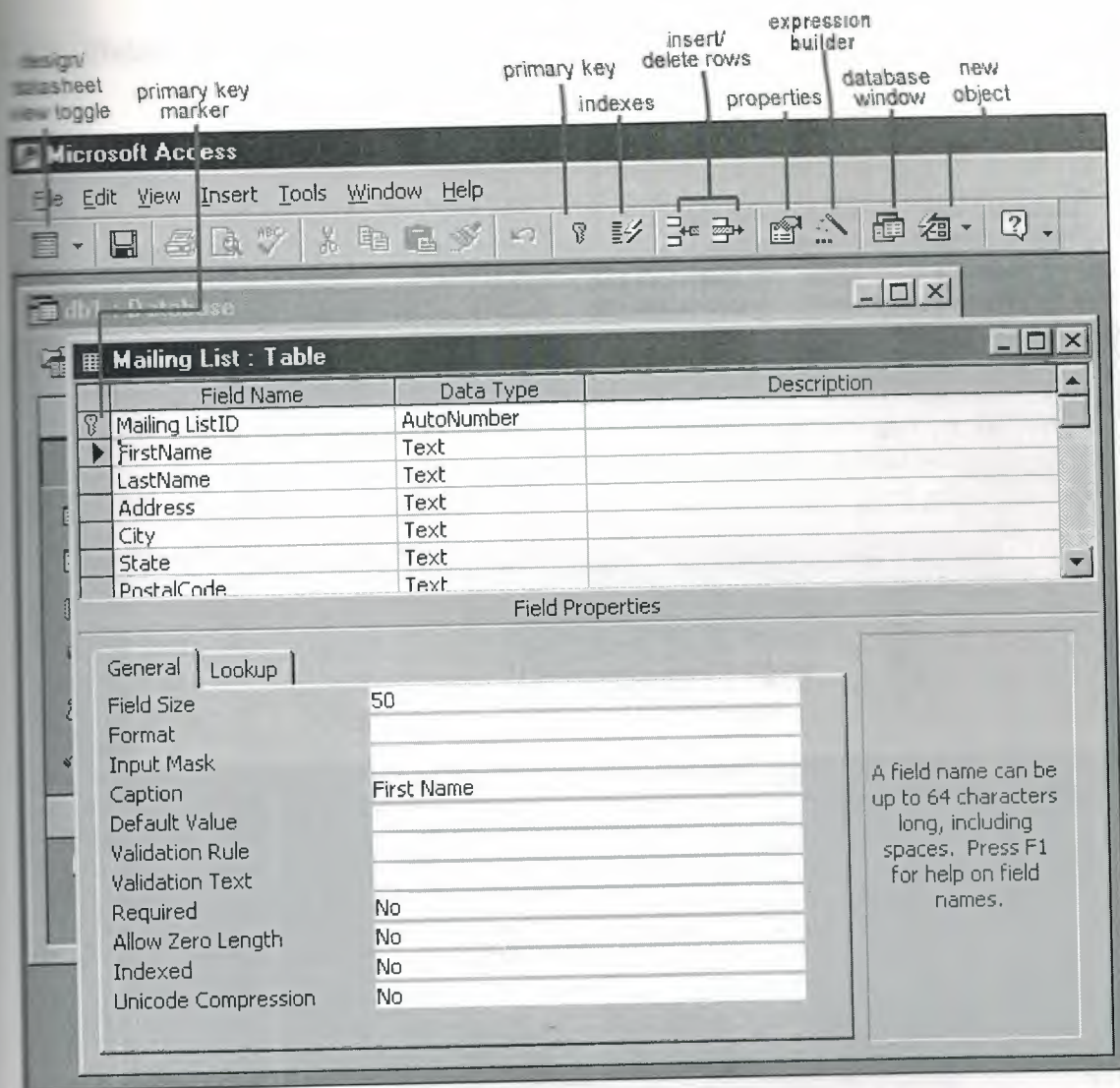
2.2.1 Database Window

The Database Window organizes all of the objects in the database. The default tables listing provides links for creating tables and will list all of the tables in the database when they have been added.



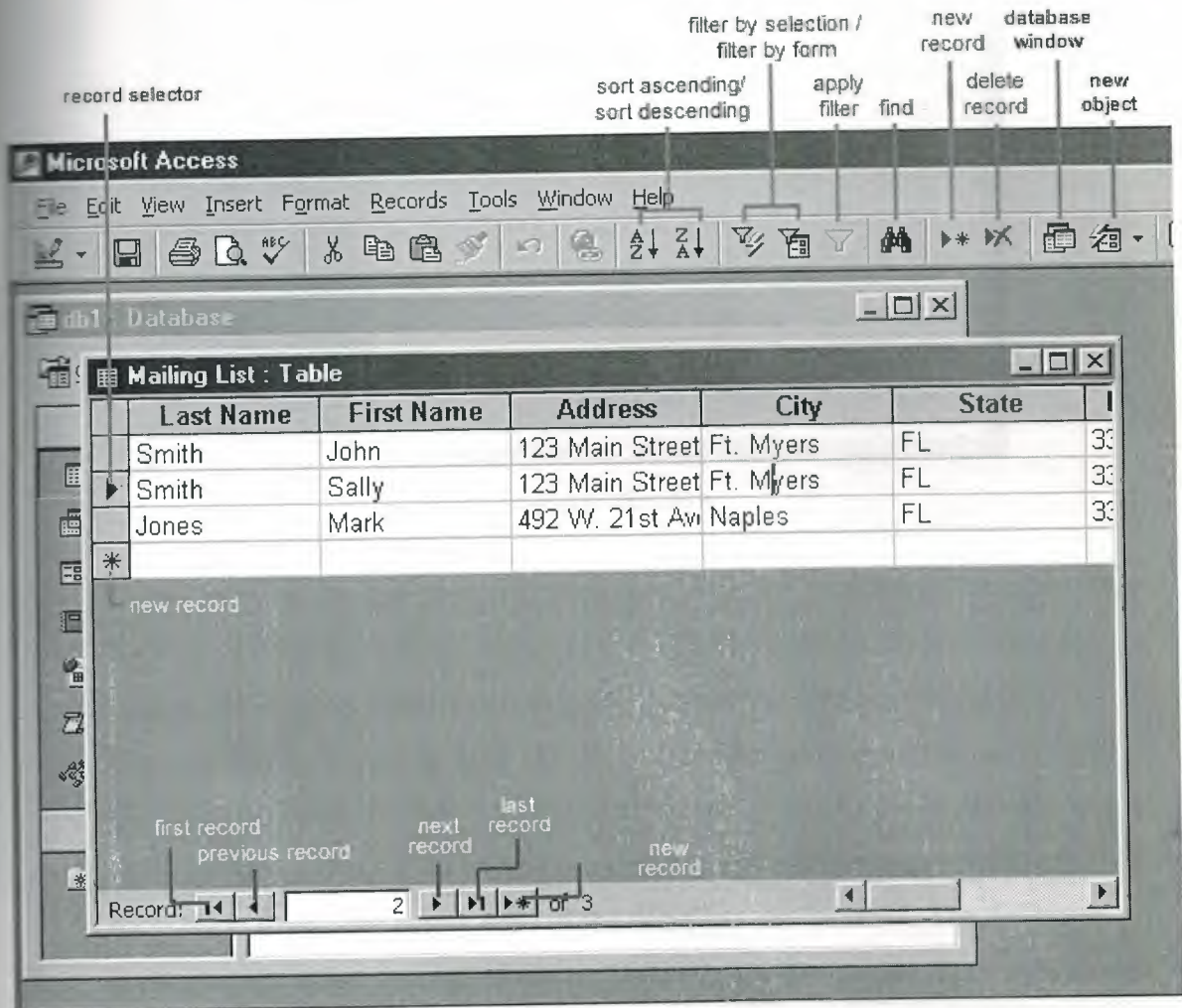
2.2.2 Design View

Design View customizes the fields in the database so that data can be entered.



2.2.3 Datasheet View

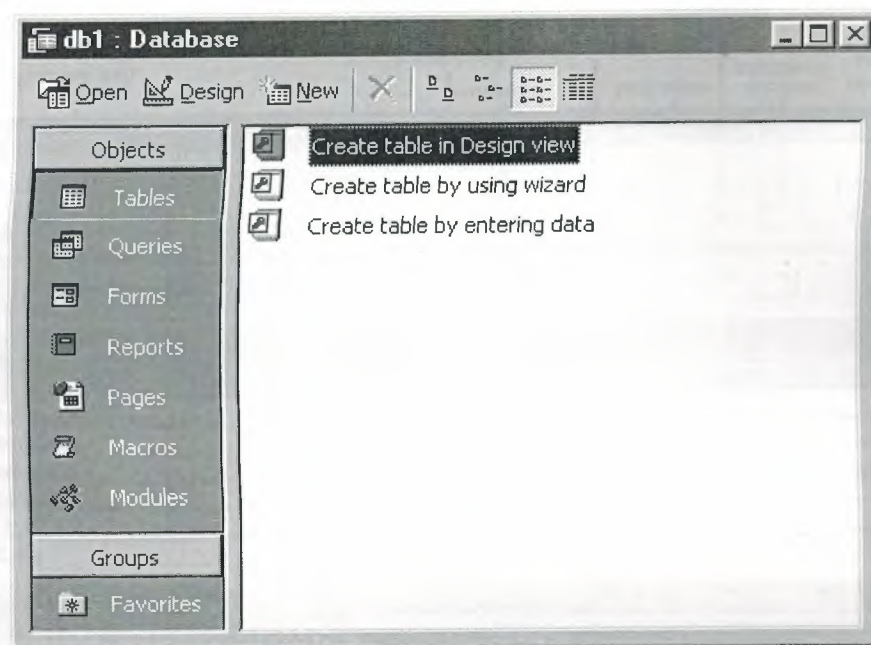
The datasheet allows you to enter data into the database



2.3 Creating Tables

2.3.1 Introduction to Tables

Tables are grids that store information in a database similar to the way an Excel worksheet stores information in a workbook. Access provides three ways to create a table for which there are icons in the Database Window. Double-click on the icons to create a table.

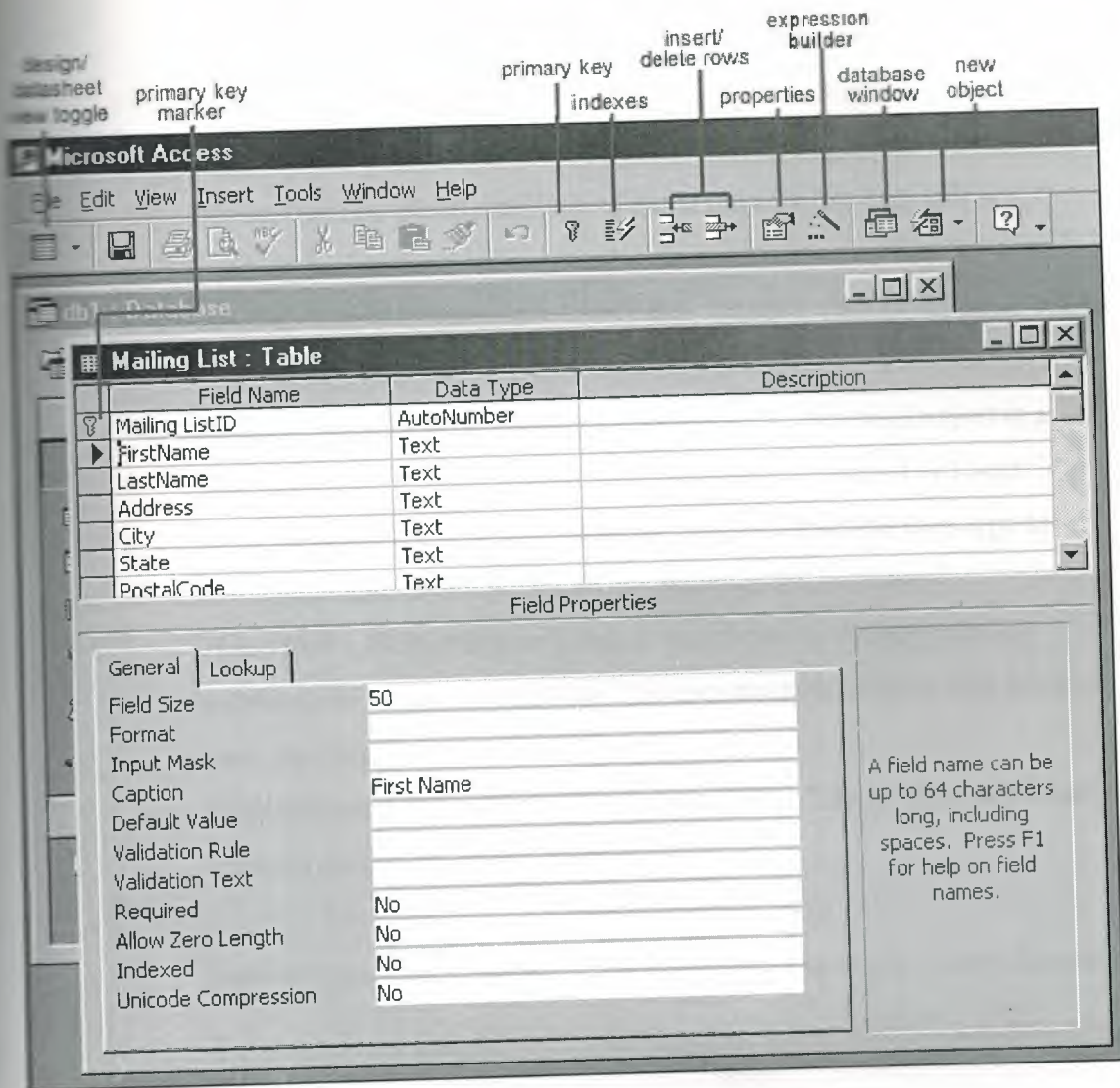


the Database Window

- **Create table in Design view** will allow you to create the fields of the table. This is the most common way of creating a table and is explained in detail below.
- **Create table using wizard** will step you through the creation of a table.
- **Create table by entering data** will give you a blank datasheet with unlabelled columns that looks much like an Excel worksheet. Enter data into the cells and click the **Save** button. You will be prompted to add a primary key field. After the table is saved, the empty cells of the datasheet are trimmed. The fields are given generic names such as "Field1", "Field2", etc. To rename them with more descriptive titles that reflect the content of the fields, select **Format|Rename Column** from the menu bar or highlight the column, right-click on it with the mouse, and select **Rename Column** from the shortcut menu.

2.3.2 Create a Table in Design View

Design View will allow you to define the fields in the table before adding any data to the datasheet. The window is divided into two parts: a top pane for entering the field name, data type, and an option description of the field, and a bottom pane for specifying field properties.



- **Field Name** - This is the name of the field and should represent the contents of the field such as "Name", "Address", "Final Grade", etc. The name can not exceed 64 characters in length and may include spaces.
- **Data Type** is the type of value that will be entered into the fields.
 - **Text** - The default type, text type allows any combination of letters and numbers up to a maximum of 255 characters per field record.
 - **Memo** - A text type that stores up to 64,000 characters.
 - **Number** - Any number can be stored.
 - **Date/Time** - A date, time, or combination of both.
 - **Currency** - Monetary values that can be set up to automatically include a dollar sign (\$) and correct decimal and comma positions.
 - **AutoNumber** - When a new record is created, Access will automatically assign a unique integer to the record in this field. From the General

options, select Increment if the numbers should be assigned in order or random if any random number should be chosen. Since every record in a datasheet must include at least one field that distinguishes it from all others, this is a useful data type to use if the existing data will not produce such values.

- **Yes/No** - Use this option for True/False, Yes/No, On/Off, or other values that must be only one of two.
- **OLE Object** - An OLE (Object Linking and Embedding) object is a sound, picture, or other object such as a Word document or Excel spreadsheet that is created in another program. Use this data type to embed an OLE object or link to the object in the database.
- **Hyperlink** - A hyperlink will link to an Internet or Intranet site, or another location in the database. The data consists of up to four parts each separated by the pound sign (#):
DisplayText#Address#SubAddress#ScreenTip. The Address is the only required part of the string. Examples:

Internet hyperlink example: FGCU Home Page#http://www.fgcu.edu#

Database link example: #c:\My Documents\database.mdb#MyTable

- **Description** (optional) - Enter a brief description of what the contents of the field are.
- **Field Properties** - Select any pertinent properties for the field from the bottom pane.

2.3.3 Field Properties

Properties for each field are set from the bottom pane of the Design View window.

- **Field Size** is used to set the number of characters needed in a text or number field. The default field size for the text type is 50 characters. If the records in the field will only have two or three characters, you can change the size of the field to save disk space or prevent entry errors by limiting the number of characters allowed. Likewise, if the field will require more than 50 characters, enter a

number up to 255. The field size is set in exact characters for Text type, but options are give for numbers:

- **Byte** - Positive integers between 1 and 255
- **Integer** - Positive and negative integers between -32,768 and 32,768
- **Long Integer (default)** - Larger positive and negative integers between - 2 billion and 2 billion.
- **Single** - Single-precision floating-point number
- **Double** - Double-precision floating-point number
- **Decimal** - Allows for Precision and Scale property control
- **Format** conforms the data in the field to the same format when it is entered into the datasheet. For text and memo fields, this property has two parts that are separated by a semicolon. The first part of the property is used to apply to the field and the second applies to empty fields.

Text and memo format.

Text Format			
Format	Datasheet Entry	Display	Explanation
@@@-@@@	1234567	123-4567	@ indicates a required character or space
@@@-@@@&	123456	123-456	& indicates an optional character or space
<	HELLO	hello	< converts characters to lowercase
>	hello	HELLO	> converts characters to uppercase
@\!	Hello	Hello!	\ adds characters to the end

@;"No Data Entered"	Hello	Hello	
@;"No Data Entered"	(blank)	No Data Entered	

Number format. Select one of the preset options from the drop down menu or construct a custom format using symbols explained below:

Number Format			
Format	Datasheet Entry	Display	Explanation
###,##0.00	123456.78	123,456.78	0 is a placeholder that displays a digit or 0 if there is none. # is a placeholder that displays a digit or nothing if there is none.
\$###,##0.00	0	\$0.00	
###.00%	.123	12.3%	% multiplies the number by 100 and added a percent sign

Currency format. This formatting consists of four parts separated by semicolons:
format for positive numbers; format for negative numbers; format for zero values; format for Null values.

Currency Format	
Format	Explanation
\$##0.00;(\$##0.00)[Red];\$0.00;"none"	Positive values will be normal currency format, negative numbers will be red in parentheses, zero is entered for zero values, and "none" will be written for Null values.

Date format. In the table below, the value "1/1/01" is entered into the datasheet,

and the following values are displayed as a result of the different assigned formats.

Date Format		
Format	Display	Explanation
dddd","mmm d","yyyy	Monday, January 1, 2001	dddd, mmmm, and yyyy print the full day name, month name, and year
ddd","mmm ". " d", "yy	Mon, Jan. 1, '01	ddd, mmm, and yy print the first three day letters, first three month letters, and last two year digits
"Today is " dddd	Today is Monday	
h:n:s: AM/PM	12:00:00 AM	"n" is used for minutes to avoid confusion with months

Yes/No fields are displayed as check boxes by default on the datasheet. To change the formatting of these fields, first click the Lookup tab and change the Display Control to a text box. Go back to the General tab choices to make formatting changes. The formatting is designated in three sections separated by semicolons. The first section does not contain anything but the semicolon must be included. The second section specifies formatting for Yes values and the third for No values.

Yes/No Format	
Format	Explanation
;"Yes"[green];"No"[red]	Prints "Yes" in green or "No" in red

- **Default Value** - There may be cases where the value of a field will usually be the same for all records. In this case, a changeable default value can be set to prevent typing the same thing numerous times. Set the Default Value property.

2.3.4 Primary Key

Every record in a table must have a primary key that differentiates it from every other record in the table. In some cases, it is only necessary to designate an existing field as the primary key if you are certain that every record in the table will have a different value for that particular field. A social security number is an example of a record whose values will only appear once in a database table.

Designate the primary key field by right-clicking on the record and selection **Primary Key** from the shortcut menu or select **Edit|Primary Key** from the menu bar. The primary key field will be noted with a key image to the left. To remove a primary key, repeat one of these steps.

If none of the existing fields in the table will produce unique values for every record, a separate field must be added. Access will prompt you to create this type of field at the beginning of the table the first time you save the table and a primary key field has not been assigned. The field is named "ID" and the data type is "autonumber". Since this extra field serves no purpose to you as the user, the autonumber type automatically updates whenever a record is added so there is no extra work on your part. You may also choose to hide this column in the datasheet as explained on a later page in this tutorial.

2.3.5 Indexes

Creating indexes allows Access to query and sort records faster. To set an indexed field, select a field that is commonly searched and change the Indexed property to **Yes (Duplicates OK)** if multiple entries of the same data value are allowed or **Yes (No Duplicates)** to prevent duplicates.



2.3.6 Field Validation Rules

Validation Rules specify requirements (change word) for the data entered in the worksheet. A customized message can be displayed to the user when data that violates the rule setting is entered. Click the expression builder ("...") button at the end of the Validation Rule box to write the validation rule. Examples of field validation rules include $\neq 0$ to not allow zero values in the record, and ??? to only all data strings three characters in length.

Input Masks

An input mask controls the value of a record and sets it in a specific format. They are similar to the Format property, but instead display the format on the datasheet before the data is entered. For example, a telephone number field can be formatted with an input mask to accept ten digits that are automatically formatted as "(555) 123-4567". The blank field would look like () - . An input mask to a field by following these steps:

1. In design view, place the cursor in the field that the input mask will be applied to.
2. Click in the white space following **Input Mask** under the **General** tab.
3. Click the "..." button to use the wizard or enter the mask, (@@) @@-@@@@, into the field provided. The following symbols can be used to create an input mask from scratch:

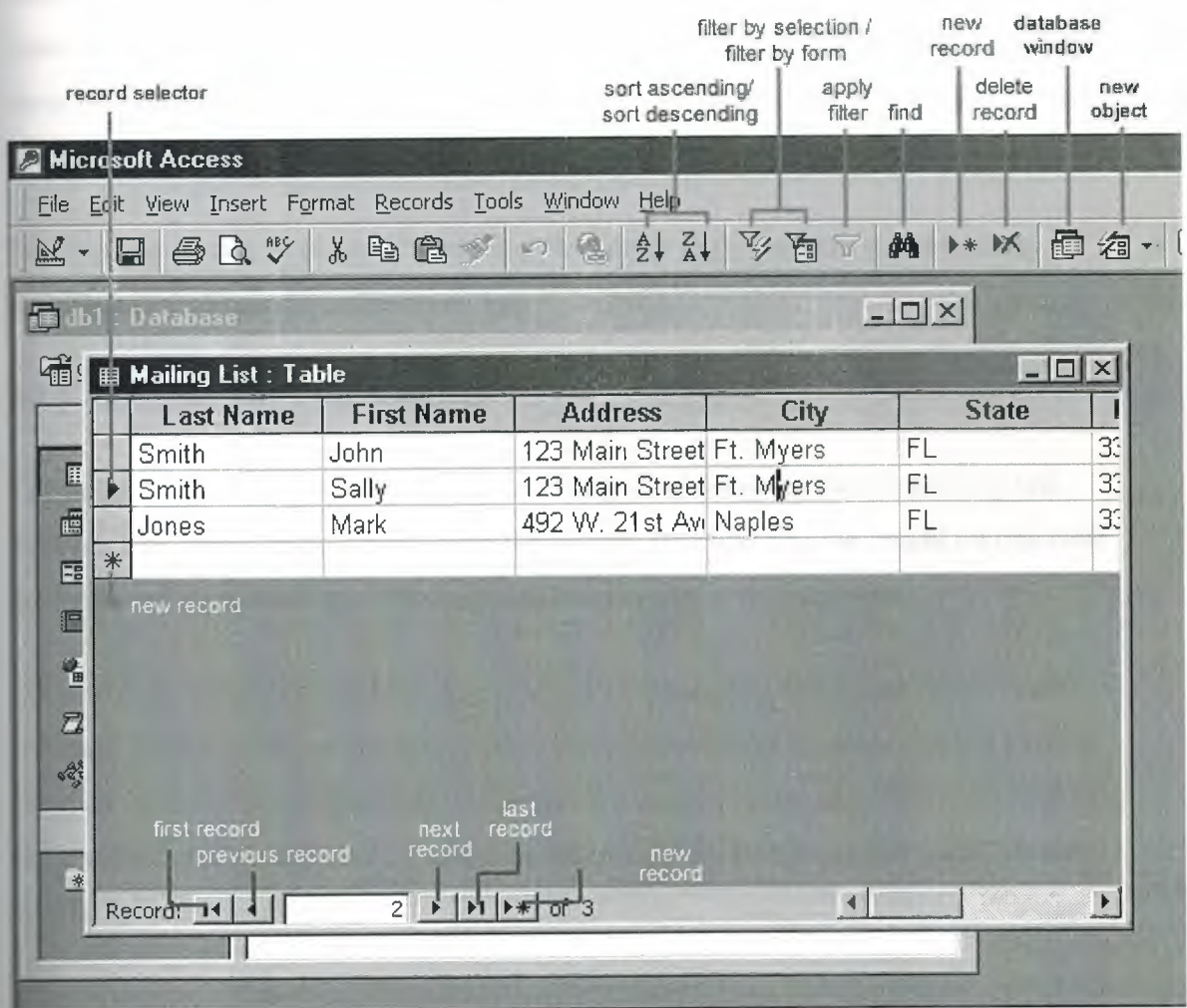
Input Mask Symbols	
Symbol	Explanation
A	Letter or digit
0	A digit 0 through 9 without a + or - sign and with blanks displayed as zeros
9	Same as 0 with blanks displayed as spaces
#	Same as 9 with +/- signs
?	Letter

L	Letter A through Z
C or &	Character or space
<	Convert letters to lower case
>	Convert letters to upper case

2.4 Datasheet Records

2.4.1 Adding Records

Add new records to the table in datasheet view by typing in the record beside the asterisk (*) that marks the new record. You can also click the new record button at the bottom of the datasheet to skip to the last empty record.



2.4.2 Editing Records

To edit records, simply place the cursor in the record that is to be edited and make the necessary changes. Use the arrow keys to move through the record grid. The previous, next, first, and last record buttons at the bottom of the datasheet are helpful in maneuvering through the datasheet.

2.4.3 Deleting Records

Delete a record on a datasheet by placing the cursor in any field of the record row and select **Edit|Delete Record** from the menu bar or click the **Delete Record** button on the datasheet toolbar.

2.4.4 Adding and Deleting Columns

Although it is best to add new fields (displayed as columns in the datasheet) in design view because more options are available, they can also be quickly added in datasheet view. Highlight the column that the new column should appear to the left of by clicking its label at the top of the datasheet and select **Insert|Column** from the menu bar.

Entire columns can be deleted by placing the cursor in the column and selecting **Edit|Delete Column** from the menu bar.

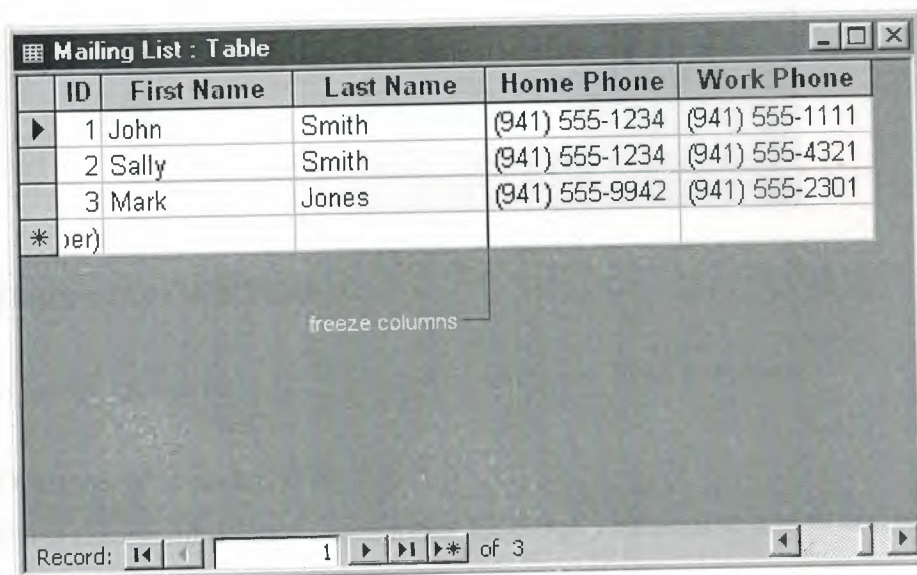
2.4.5 Resizing Rows and Columns

The height of rows on a datasheet can be changed by dragging the gray sizing line between row labels up and down with the mouse. By changing the height on one row, the height of all rows in the datasheet will be changed to the new value.

Column width can be changed in a similar way by dragging the sizing line between columns. Double click on the line to have the column automatically fit to the longest value of the column. Unlike rows, columns on a datasheet can be different widths. More exact values can be assigned by selecting **Format|Row Height** or **Format|Column Width** from the menu bar.

2.4.6 Freezing Columns

Similar to freezing panes in Excel, columns on an Access table can be frozen. This is helpful if the datasheet has many columns and relevant data would otherwise not appear on the screen at the same time. Freeze a column by placing the cursor in any record in the column and select **Format|Freeze Columns** from the menu bar. Select the same option to unfreeze a single column or select **Format|Unfreeze All Columns**.



The screenshot shows a window titled "Mailing List : Table" containing a table with the following data:

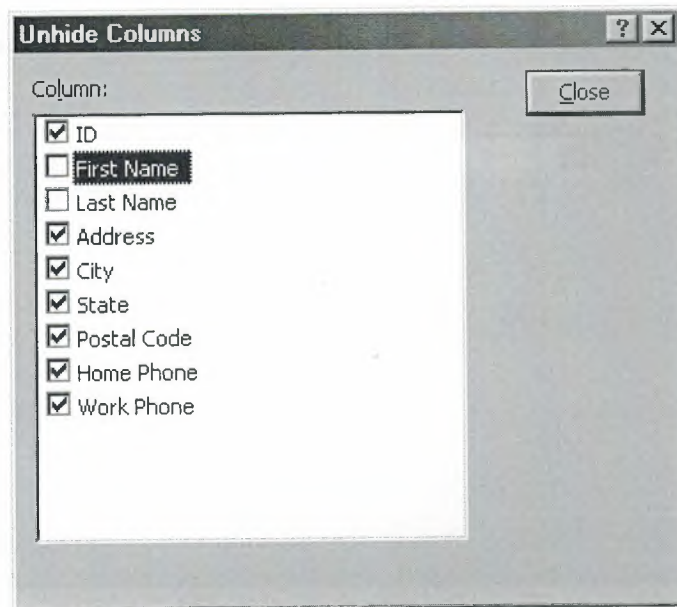
	ID	First Name	Last Name	Home Phone	Work Phone
▶	1	John	Smith	(941) 555-1234	(941) 555-1111
	2	Sally	Smith	(941) 555-1234	(941) 555-4321
	3	Mark	Jones	(941) 555-9942	(941) 555-2301
*	per)				

Below the table, a status bar indicates "Record: 1 of 3". A callout box labeled "freeze columns" points to the "Home Phone" column header.

2.4.7 Hiding Columns

Columns can also be hidden from view on the datasheet although they will not be deleted from the database. To hide a column, place the cursor in any record in the column or highlight multiple adjacent columns by clicking and dragging the mouse along the column headers, and select **Format|Hide Columns** from the menu bar.

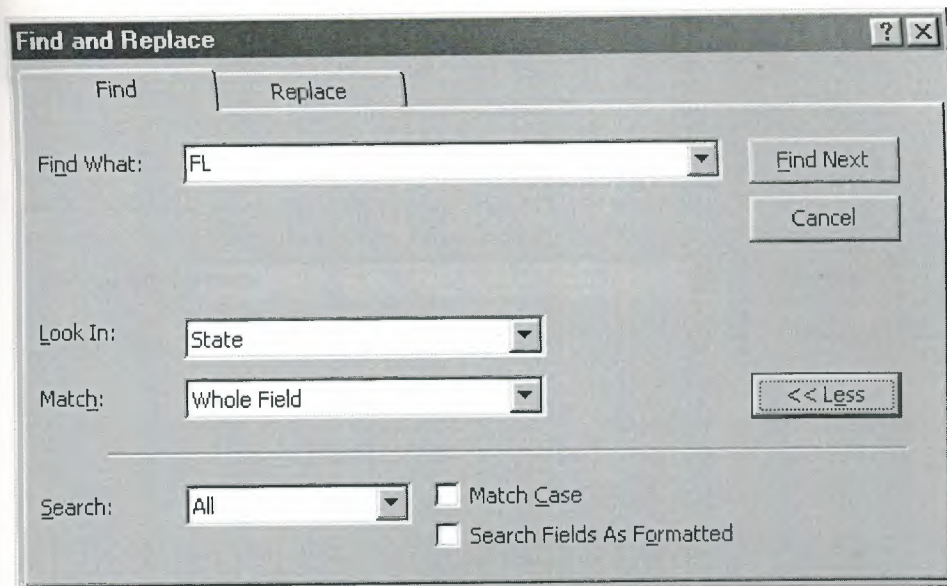
To show columns that have been hidden, select **Format|Unhide Columns** from the menu bar. A window displaying all of the fields in the table will be listed with check boxes beside each field name. Check the boxes beside all fields that should be visible on the data table and click the **Close** button.



2.4.8 Finding Data in a Table

Data in a datasheet can be quickly located by using the **Find** command.

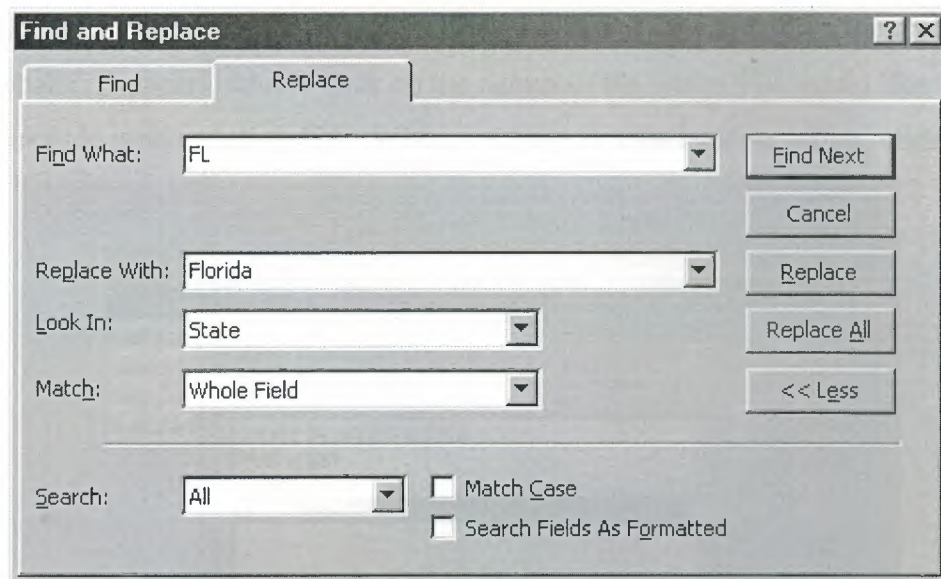
1. Open the table in datasheet view.
2. Place the cursor in any record in the field that you want to search and select **Edit|Find...** from the menu bar.
3. Enter the value criteria in the **Find What:** box.
4. From the **Look In:** drop-down menu, define the area of the search by selecting the entire table or just the field in the table you placed your cursor in during step 2.
5. Select the matching criteria from **Match:** to and click the **More >>** button for additional search parameters.
6. When all of the search criteria is set, click the **Find Next** button. If more than one record meets the criteria, keep clicking **Find Next** until you reach the correct record.



2.4.9 Replace

The replace function allows you to quickly replace a single occurrence of data with a new value or to replace all occurrences in the entire table.

1. Select **Edit|Replace...** from the menu bar (or click the **Replace** tab if the Find window is already open).
2. Follow the steps described in the Find procedure for searching for the data that should be replaced and type the new value of the data in the **Replace With:** box.
3. Click the **Find Next** button to step through occurrences of the data in the table and click the **Replace** button to make single replacements. Click **Replace All** to change all occurrences of the data in one step.



2.4.10 Check Spelling and AutoCorrect

The spell checker can be used to flag spelling errors in text and menu fields in a datasheet. Select **Tools|Spelling** from the menu bar to activate the spell checker and make corrections just as you would using Word or Excel. The AutoCorrect feature can automatically correct common spelling errors such as two INitial CAPITALs, capitalizing the first letter of the first word of a sentence, and anything you define. Select **Tools|AutoCorrect** to set these features.

2.4.11 Print a Datasheet

Datasheets can be printed by clicking the **Print** button on the toolbar or select **File|Print** to set more printing options.

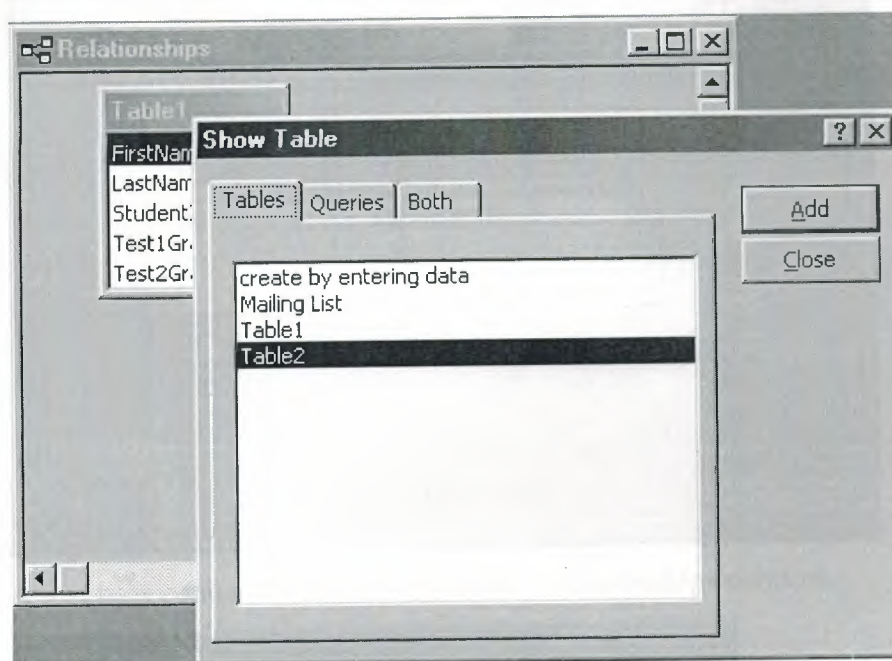
2.5 Table Relationships

Table Relationships

To prevent the duplication of information in a database by repeating fields in more than one table, table relationships can be established to link fields of tables together. Follow the steps below to set up a relational database:

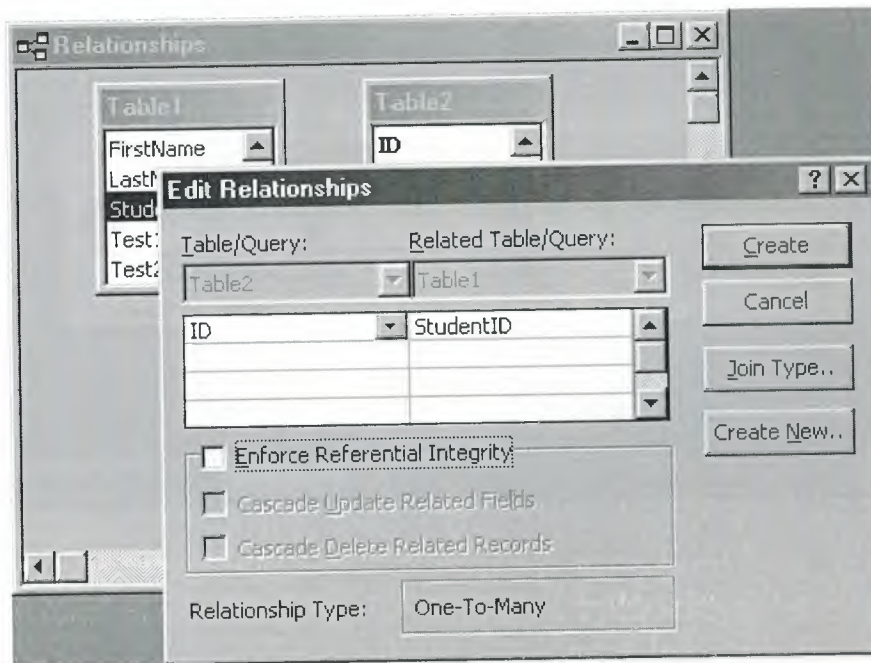
1. Click the **Relationships** button on the toolbar. 

2. From the **Show Table** window (click the **Show Table** button on the toolbar to make it appear), double click on the names of the tables you would like to include in the relationships. When you have finished adding tables, click **Close**.

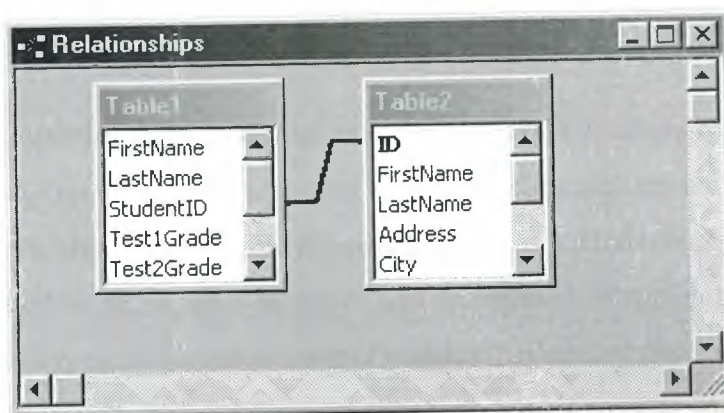


3. To link fields in two different tables, click and drag a field from one table to the corresponding field on the other table and release the mouse button. The **Edit Relationships** window will appear. From this window, select different fields if necessary and select an option from Enforce Referential Integrity if necessary. These options give Access permission to automatically make changes to referential tables if key records in one of the tables is deleted. Check the **Enforce Referential Integrity** box to ensure that the relationships are valid and that the data is not accidentally deleted when data is added, edited, or deleted. Click

Create to create the link.



4. A line now connects the two fields in the Relationships window.



5. The datasheet of a relational table will provide expand and collapse indicators to view subdatasheets containing matching information from the other table. In the example below, the student address database and student grade database were related and the two can be shown simultaneously using the expand feature. To expand or collapse all subdatasheets at once, select

Format|Subdatasheet|Expand All or Collapse All from the toolbar.

ID	First Name	Last Name	Address	Test1Grade	Test2Grade	Test3Grade	CourseAverage
977422811	John	Smith	123 Main Street Ft.	95	85	90	90
1002552704	Jane	Jones	456 Elm Ave. Ft.	0	0	0	0

Record: 1 of 1

2.6 Queries

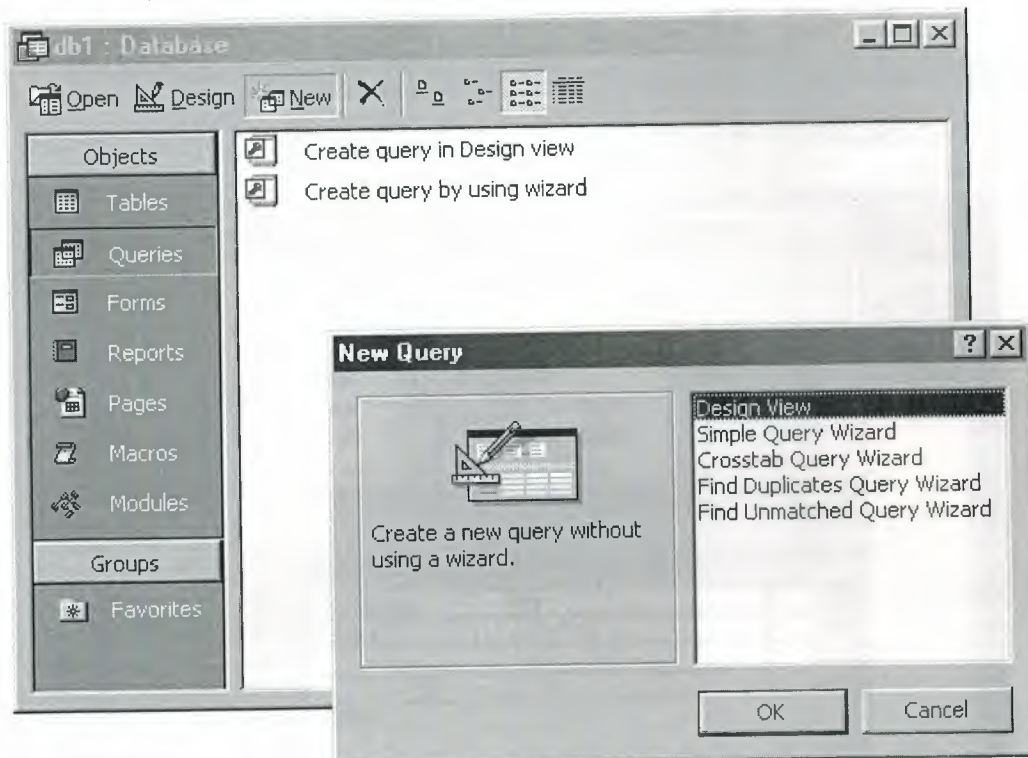
2.6.1 Introduction to Queries

Queries select records from one or more tables in a database so they can be viewed, analyzed, and sorted on a common datasheet. The resulting collection of records, called a **dynaset** (short for dynamic subset), is saved as a database object and can therefore be easily used in the future. The query will be updated whenever the original tables are updated. Types of queries are *select queries* that extract data from tables based on specified values, *find duplicate* queries that display records with duplicate values for one or more of the specified fields, and *find unmatched* queries display records from one table that do not have corresponding values in a second table.

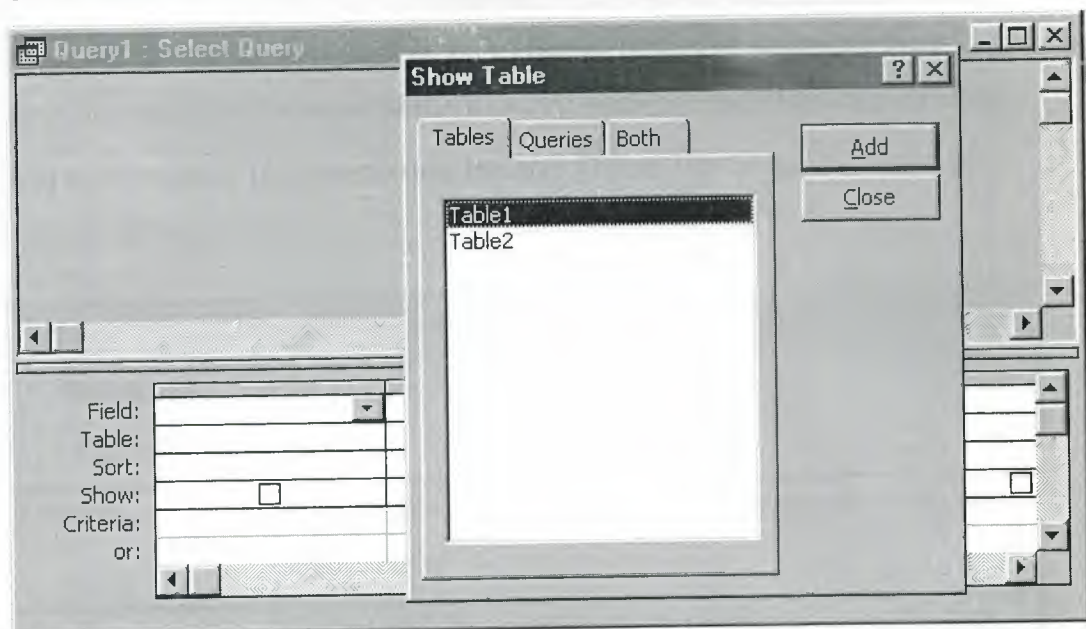
2.6.2 Create a Query in Design View

Follow these steps to create a new query in Design View:

1. From the Queries page on the Database Window, click the **New** button.

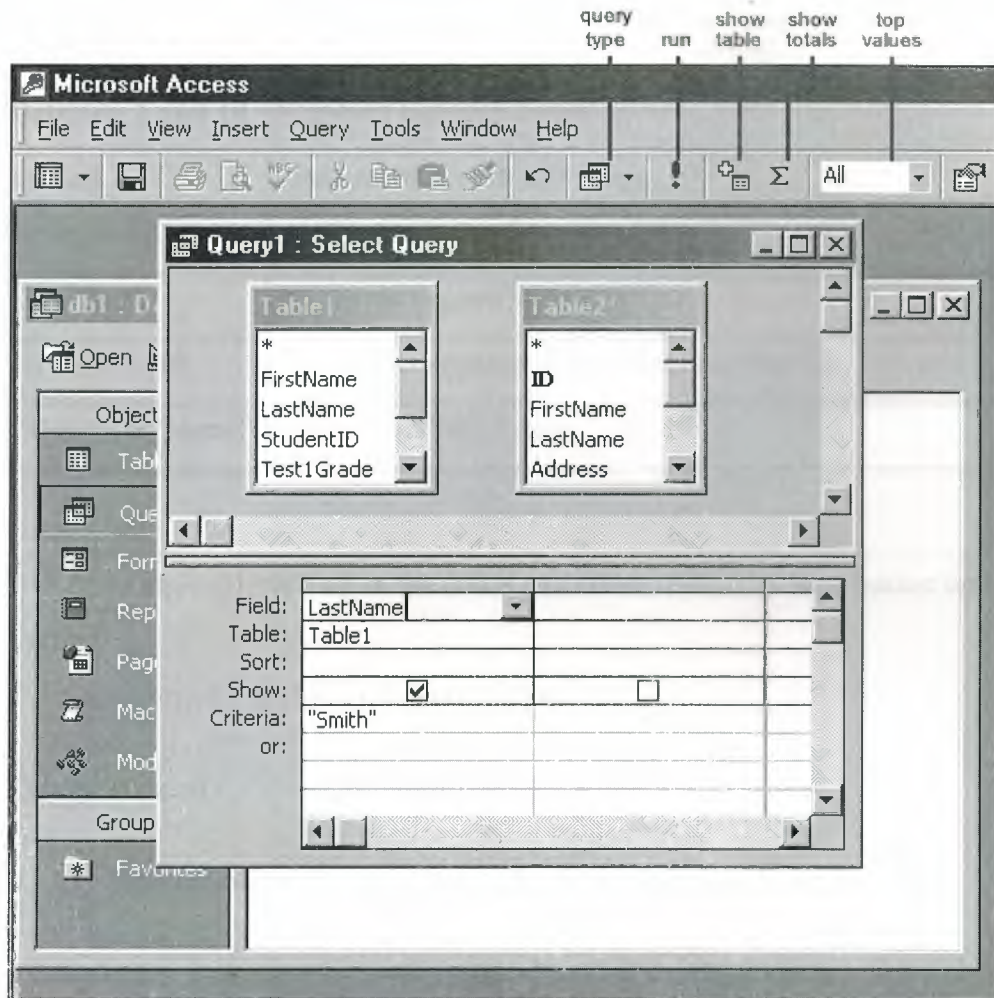



2. Select Design View and click **OK**.
3. Select tables and existing queries from the **Tables** and **Queries** tabs and click the **Add** button to add each one to the new query.
4. Click **Close** when all of the tables and queries have been selected.



5. Add fields from the tables to the new query by double-clicking the field name in the table boxes or selecting the field from the **Field:** and **Table:** drop-down

menus on the query form. Specify sort orders if necessary.



6. Enter the criteria for the query in the **Criteria:** field. The following table provides examples for some of the wildcard symbols and arithmetic operators that may be used. The **Expression Builder**  can also be used to assist in writing the expressions.

Query Wildcards and Expression Operators	
Wildcard / Operator	Explanation
? Street	The question mark is a wildcard that takes the place of a single letter.
43th *	The asterisk is the wildcard that represents a number of characters.
<100	Value less than 100

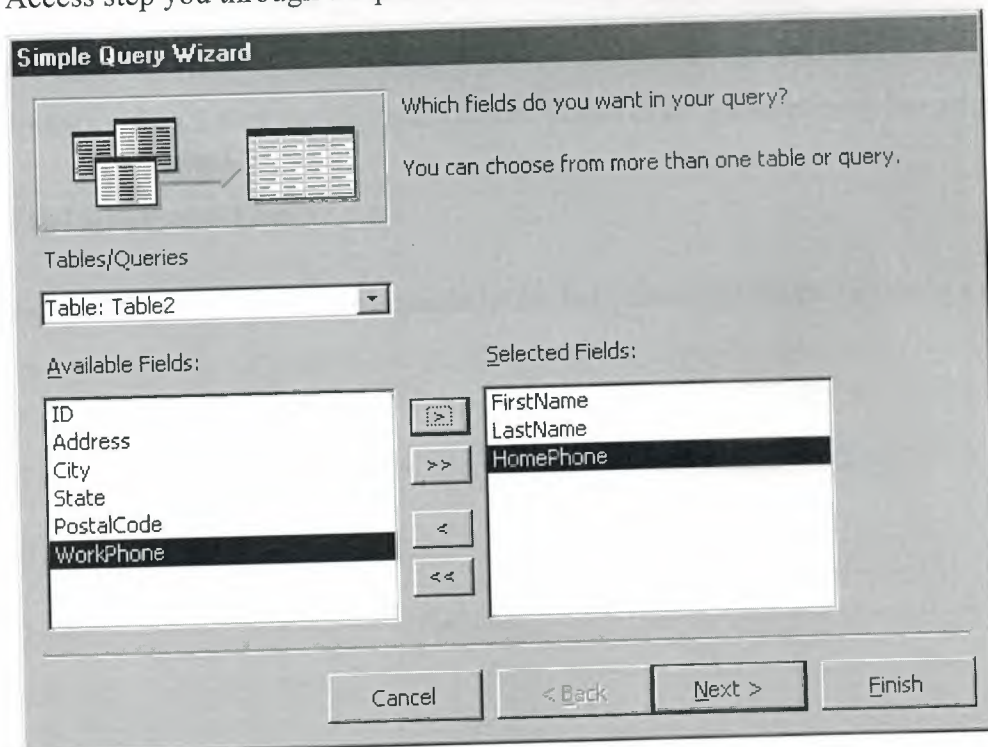
>=1	Value greater than or equal to 1
<>"FL"	Not equal to (all states besides Florida)
Between 1 and 10	Numbers between 1 and 10
Is Null	Finds records with no value
Is Not Null	or all records that have a value
Like "a*"	All words beginning with "a"
>0 And <=10	All numbers greater than 0 and less than 10
"Bob" Or "Jane"	Values are Bob or Jane

- 7.
8. After you have selected all of the fields and tables, click the **Run** button on the toolbar.
9. Save the query by clicking the **Save** button.

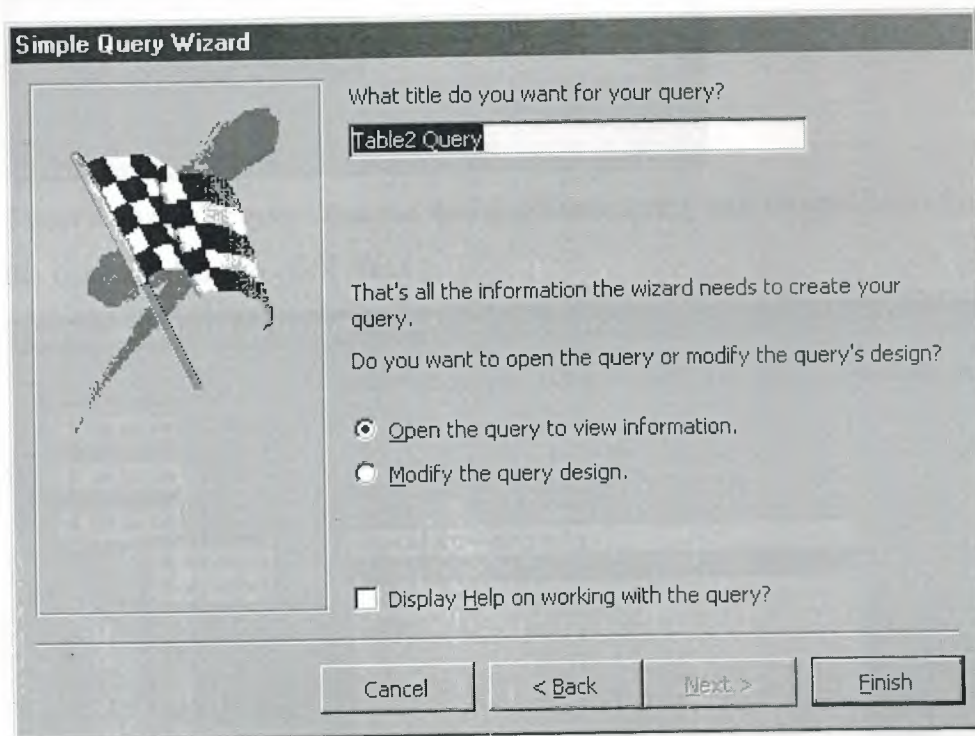
2.6.3 Query Wizard

Access' Query Wizard will easily assist you to begin creating a select query.

1. Click the **Create query by using wizard** icon in the database window to have Access step you through the process of creating a query.



2. From the first window, select fields that will be included in the query by first selecting the table from the drop-down **Tables/Queries** menu. Select the fields by clicking the > button to move the field from the Available Fields list to Selected Fields. Click the double arrow button >> to move all of the fields to Selected Fields. Select another table or query to choose from more fields and repeat the process of moving them to the Selected Fields box. Click **Next >** when all of the fields have been selected.

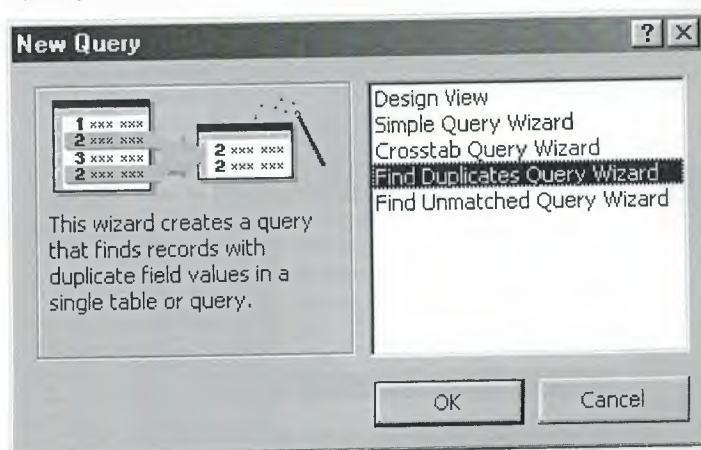


3. On the next window, enter the name for the query and click **Finish**.
4. Refer to steps 5-8 of the previous tutorial to add more parameters to the query.

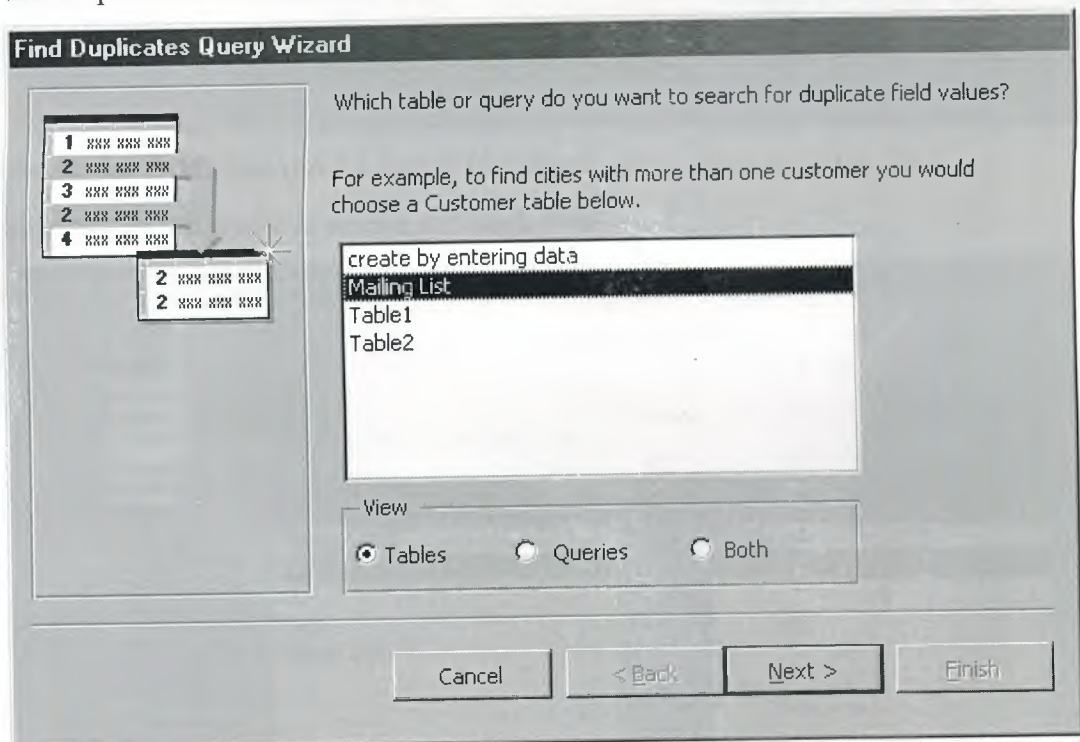
2.6.4 Find Duplicates Query

This query will filter out records in a single table that contain duplicate values in a field.

1. Click the **New** button on the Queries database window, select **Find Duplicates Query Wizard** from the **New Query** window and click **OK**.

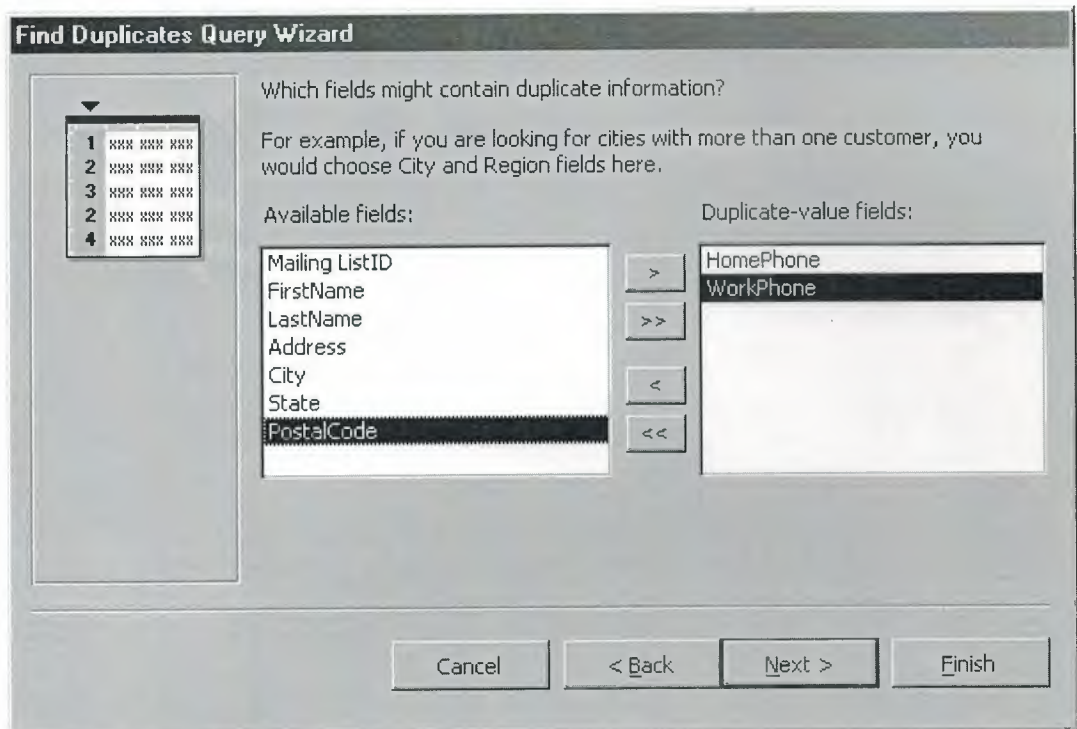


2. Select the table or query that the find duplicates query will be applied to from the list provided and click **Next >**.



3. Select the fields that may contain duplicate values by highlighting the names in the Available fields list and clicking the **>** button to individually move the fields to the Duplicate-value fields list or **>>** to move all of the fields. Click **Next >**

when all fields have been selected.



Find Duplicates Query Wizard

Which fields might contain duplicate information?

For example, if you are looking for cities with more than one customer, you would choose City and Region fields here.

Available fields:

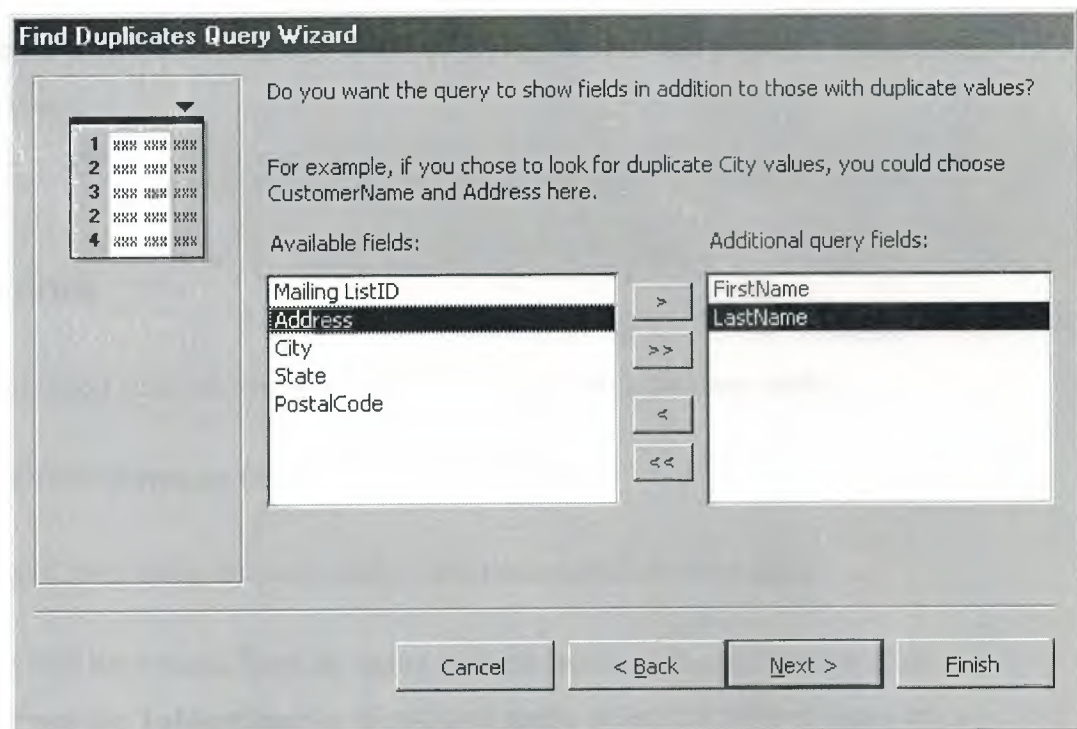
- Mailing ListID
- FirstName
- LastName
- Address
- City
- State
- PostalCode

Duplicate-value fields:

- HomePhone
- WorkPhone

Buttons: Cancel, < Back, Next >, Finish

4. Select the fields that should appear in the new query along with the fields selected on the previous screen and click **Next >**.



Find Duplicates Query Wizard

Do you want the query to show fields in addition to those with duplicate values?

For example, if you chose to look for duplicate City values, you could choose CustomerName and Address here.

Available fields:

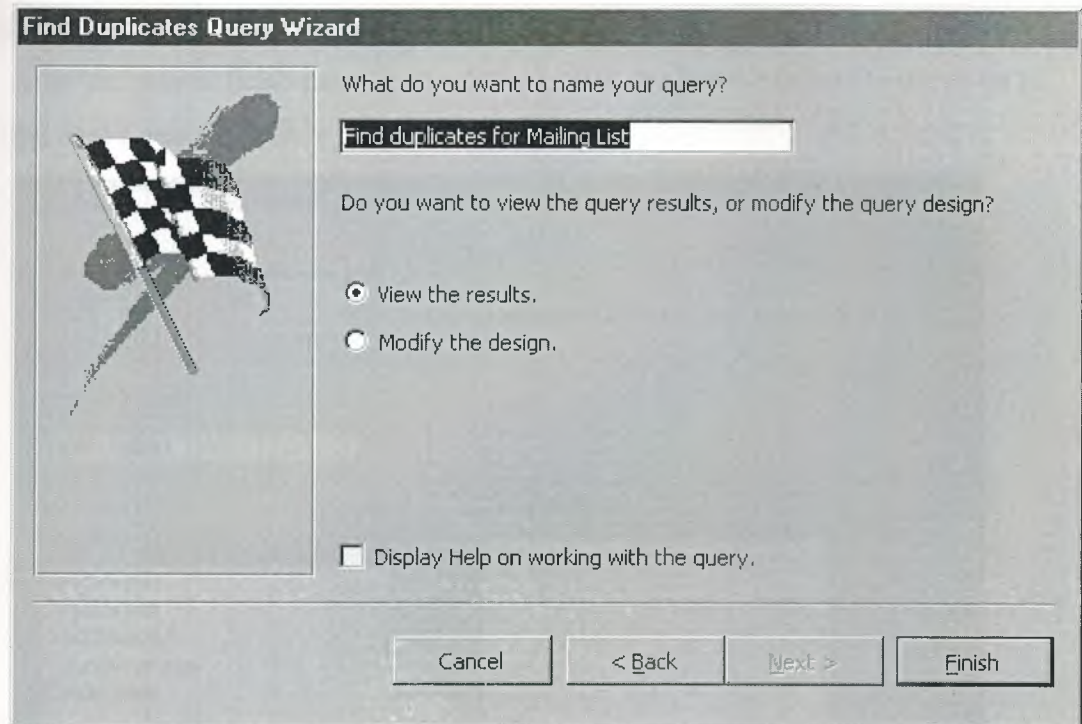
- Mailing ListID
- Address
- City
- State
- PostalCode

Additional query fields:

- FirstName
- LastName

Buttons: Cancel, < Back, Next >, Finish

5. Name the new query and click **Finish**.



2.6.5 Delete a Query

To delete a table from the query, click the table's title bar and press the **Delete** key on the keyboard.

2.7 Forms

Forms are used as an alternative way to enter data into a database table.

2.7.1 Create Form by Using Wizard

To create a form using the assistance of the wizard, follow these steps:

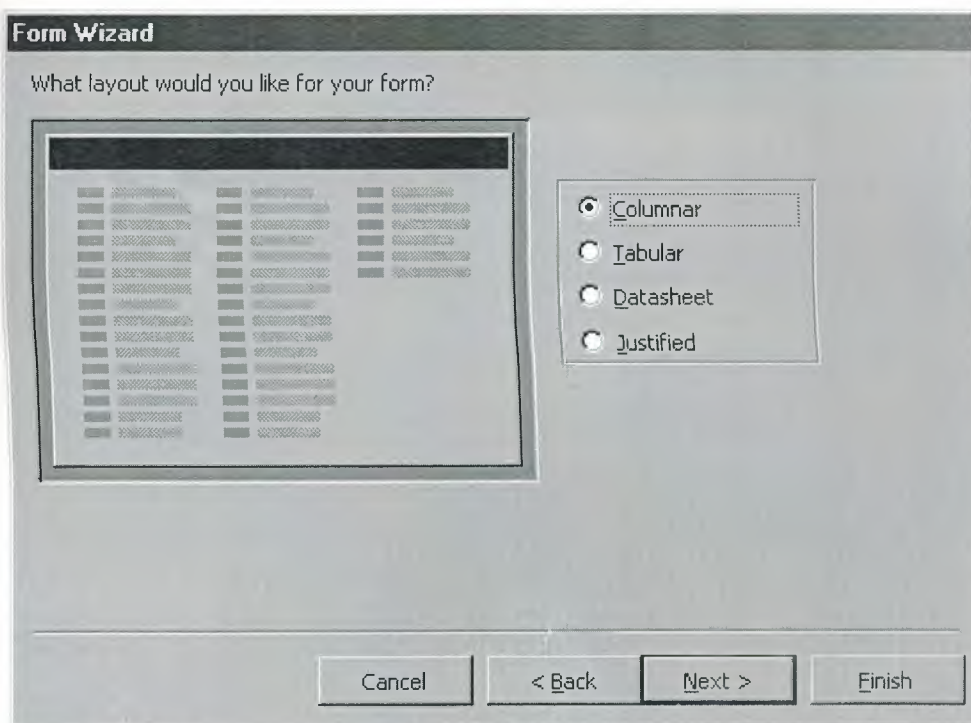
1. Click the **Create form by using wizard** option on the database window.
2. From the **Tables/Queries** drop-down menu, select the table or query whose datasheet the form will modify. Then, select the fields that will be included on the form by highlighting each one the **Available Fields** window and clicking the single right arrow button > to move the field to the **Selected Fields** window. To move all of the fields to Select Fields, click the double right arrow button >>. If

you make a mistake and would like to remove a field or all of the fields from the Selected Fields window, click the left arrow < or left double arrow << buttons. After the proper fields have been selected, click the **Next >** button to move on to the next screen.

The image shows the 'Form Wizard' dialog box. At the top, it asks 'Which fields do you want on your form?' and 'You can choose from more than one table or query.' Below this, there's a 'Tables/Queries' section with a dropdown menu showing 'Table: Table1'. Underneath, there are two lists: 'Available Fields:' and 'Selected Fields:'. The 'Available Fields:' list contains 'StudentID', 'Test1Grade', 'Test2Grade', 'Test3Grade', 'CourseAverage', and 'ExtraCredit'. The 'Selected Fields:' list is currently empty. Between these two lists are four buttons: '>', '>>', '<<', and '<'. At the bottom of the dialog are four buttons: 'Cancel', '< Back', 'Next >', and 'Finish'.

3. On the second screen, select the layout of the form.
 - **Columnar** - A single record is displayed at one time with labels and form fields listed side-by-side in columns
 - **Justified** - A single record is displayed with labels and form fields are listed across the screen
 - **Tabular** - Multiple records are listed on the page at a time with fields in columns and records in rows
 - **Datasheet** - Multiple records are displayed in Datasheet View

Click the **Next >** button to move on to the next screen.

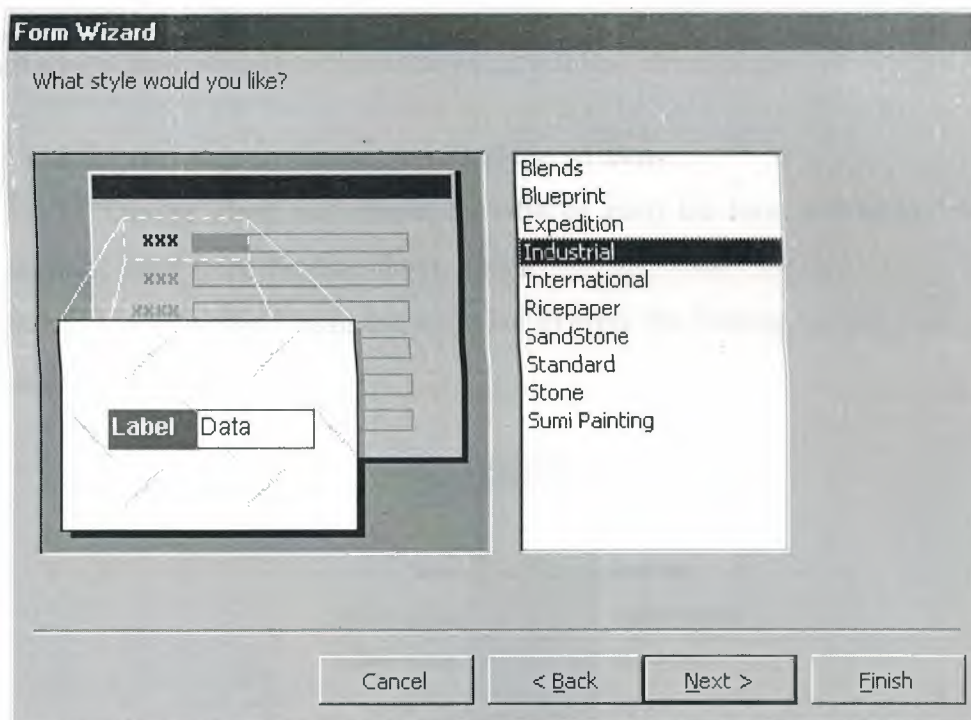


Form Wizard

What layout would you like for your form?

The dialog box shows a preview of a form layout on the left, which is a grid of fields. On the right, there are four radio button options: **Columnar** (selected), **Tabular**, **Datasheet**, and **Justified**. At the bottom, there are four buttons: **Cancel**, **< Back**, **Next >**, and **Finish**.

4. Select a visual style for the form from the next set of options and click **Next >**.



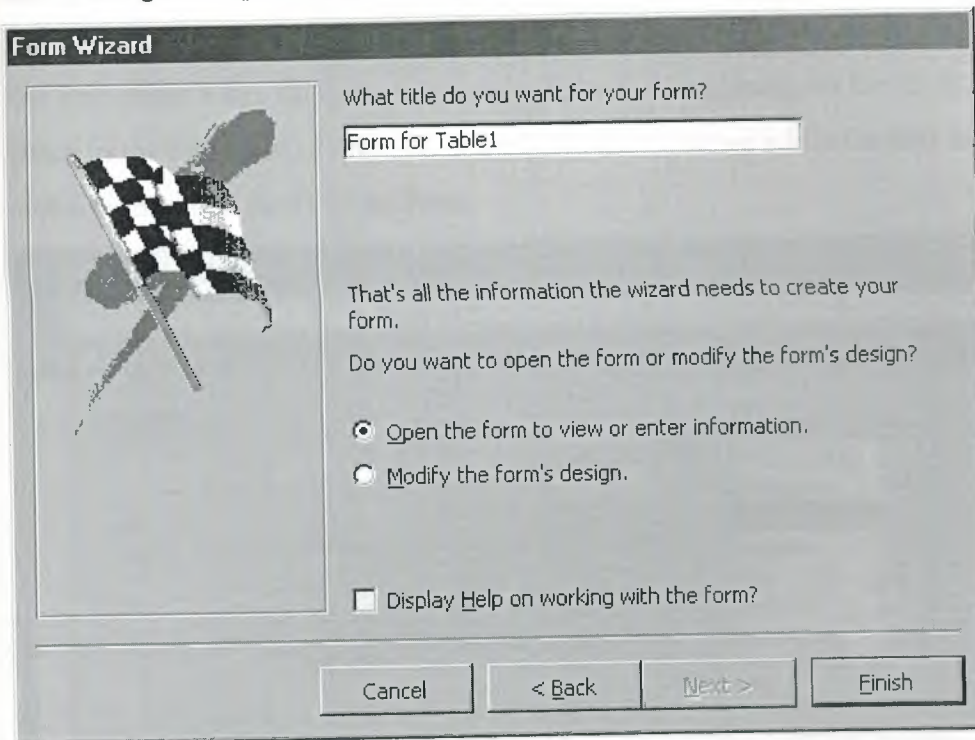
Form Wizard

What style would you like?

The dialog box shows a preview of a form layout on the left, which is a grid of fields. On the right, there is a list of styles: **Blends**, **Blueprint**, **Expedition**, **Industrial** (selected), **International**, **Ricepaper**, **SandStone**, **Standard**, **Stone**, and **Sumi Painting**. At the bottom, there are four buttons: **Cancel**, **< Back**, **Next >**, and **Finish**.

5. On the final screen, name the form in the space provided. Select "Open the form to view or enter information" to open the form in Form View or "Modify the

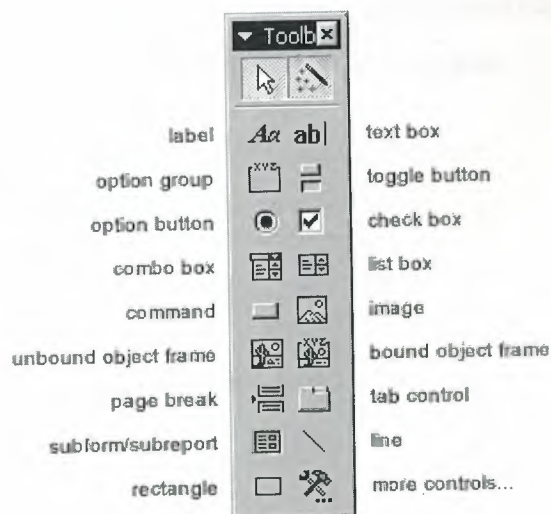
form's design" to open it in Design View. Click **Finish** to create the form.



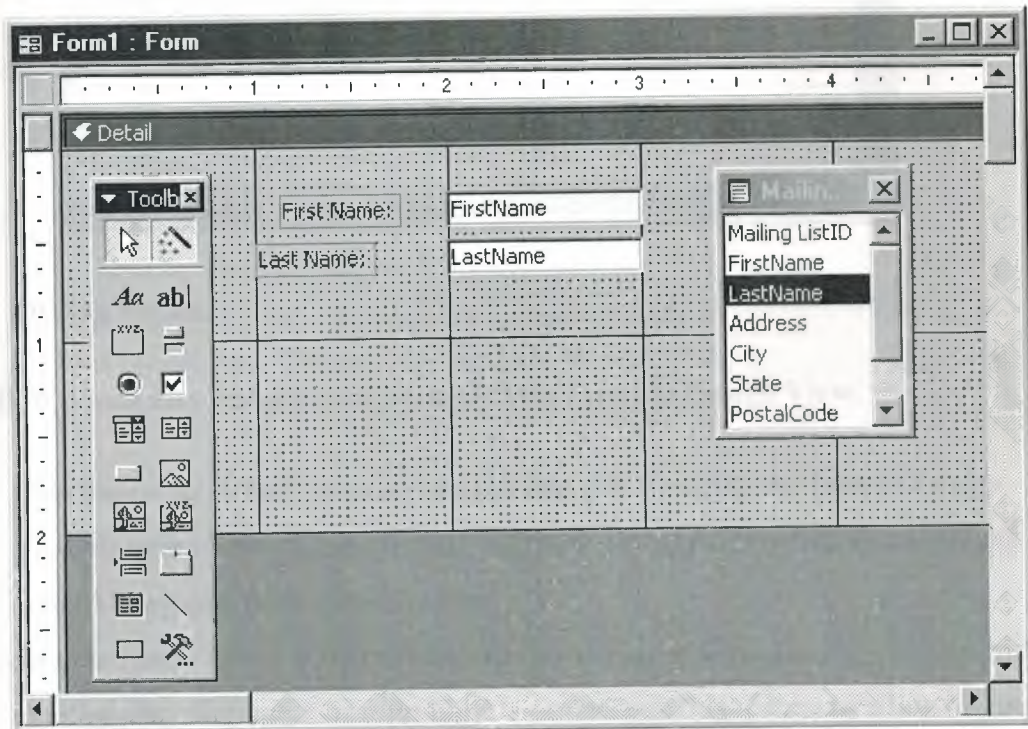
2.7.2 Create Form in Design View

To create a form from scratch without the wizard, follow these steps:


1. Click the **New** button on the form database window.
2. Select "Design View" and choose the table or query the form will be associated with the form from the drop-down menu.
3. Select **View|Toolbox** from the menu bar to view the floating toolbar with additional options.



4. Add controls to the form by clicking and dragging the field names from the Field List floating window. Access creates a text box for the value and label for the field name when this action is accomplished. To add controls for all of the fields in the Field List, double-click the Field List window's title bar and drag all of the highlighted fields to the form.



2.7.3 Adding Records Using A Form

Input data into the table by filling out the fields of the form. Press the **Tab** key to move from field to field and create a new record by clicking **Tab** after the last field of the last record. A new record can also be created at any time by clicking the **New Record** button  at the bottom of the form window. Records are automatically saved as they are entered so no additional manual saving needs to be executed.

StudentID	
Test1Grade	80
Test2Grade	95
Test3Grade	90
CourseAverage	0
ExtraCredit	No

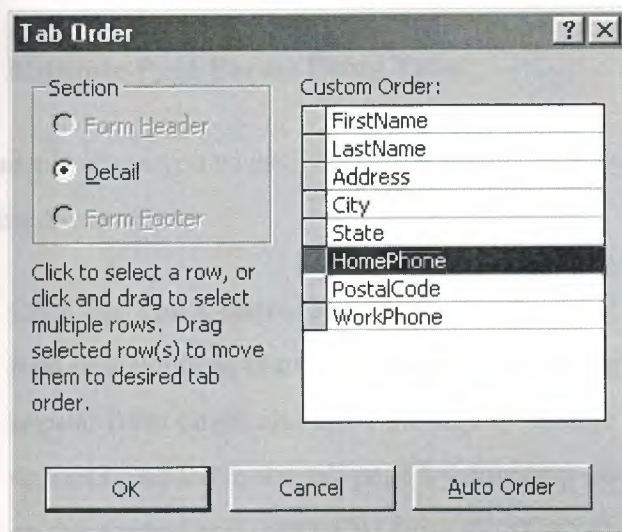
Record: 1 of 3

2.7.4 Editing Forms

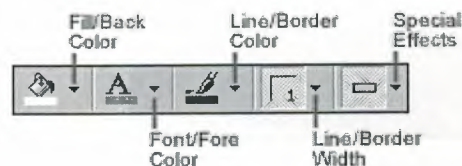
The follow points may be helpful when modifying forms in Design View.

- **Grid lines** - By default, a series of lines and dots underlay the form in Design View so form elements can be easily aligned. To toggle this feature on and off select **View|Grid** from the menu bar.
- **Snap to Grid** - Select **Format|Snap to Grid** to align form objects with the grid to allow easy alignment of form objects or uncheck this feature to allow objects to float freely between the grid lines and dots.
- **Resizing Objects** - Form objects can be resized by clicking and dragging the handles on the edges and corners of the element with the mouse.
- **Change form object type** - To easily change the type of form object without having to create a new one, right click on the object with the mouse and select **Change To** and select an available object type from the list.
- **Label/object alignment** - Each form object and its corresponding label are bounded and will move together when either one is moved with the mouse. However, to change the position of the object and label in relation to each other (to move the label closer to a text box, for example), click and drag the large handle at the top, left corner of the object or label.
- **Tab order** - Alter the tab order of the objects on the form by selecting **View|Tab Order...** from the menu bar. Click the gray box before the row you would like to

change in the tab order, drag it to a new location, and release the mouse button.



- **Form Appearance** - Change the background color of the form by clicking the **Fill/Back Color** button on the formatting toolbar and click one of the color swatches on the palette. Change the color of individual form objects by highlighting one and selecting a color from the **Font/Fore Color** palette on the formatting toolbar. The font and size, font effect, font alignment, border around each object, the border width, and a special effect can also be modified using the formatting toolbar:



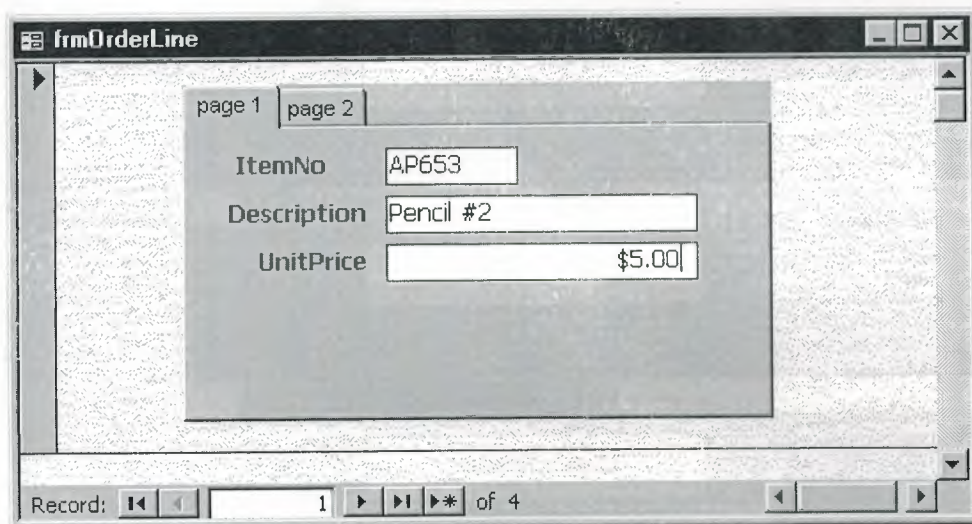
- **Page Header and Footer** - Headers and footers added to a form will only appear when it is printed. Access these sections by selecting **View|Page Header/Footer** on the menu bar. Page numbers can also be added to these sections by selecting **Insert|Page Numbers**. A date and time can be added from **Insert|Date and Time....** Select **View|Page Header/Footer** again to hide these sections from view in Design View.

2.8 More Forms

2.8.1 Multiple-Page Forms Using Tabs

Tab controls allow you to easily create multi-page forms. Create a tab control by following these steps:

1. Click the **Tab Control** icon on the toolbox and draw the control on the form.
2. Add new controls to each tab page the same way that controls are added to regular form pages and click the tabs to change pages. Existing form controls cannot be added to the tab page by dragging and dropping. Instead, right-click on the control and select **Cut** from the shortcut menu. Then right-click on the tab control and select **Paste**. The controls can then be repositioned on the tab control.

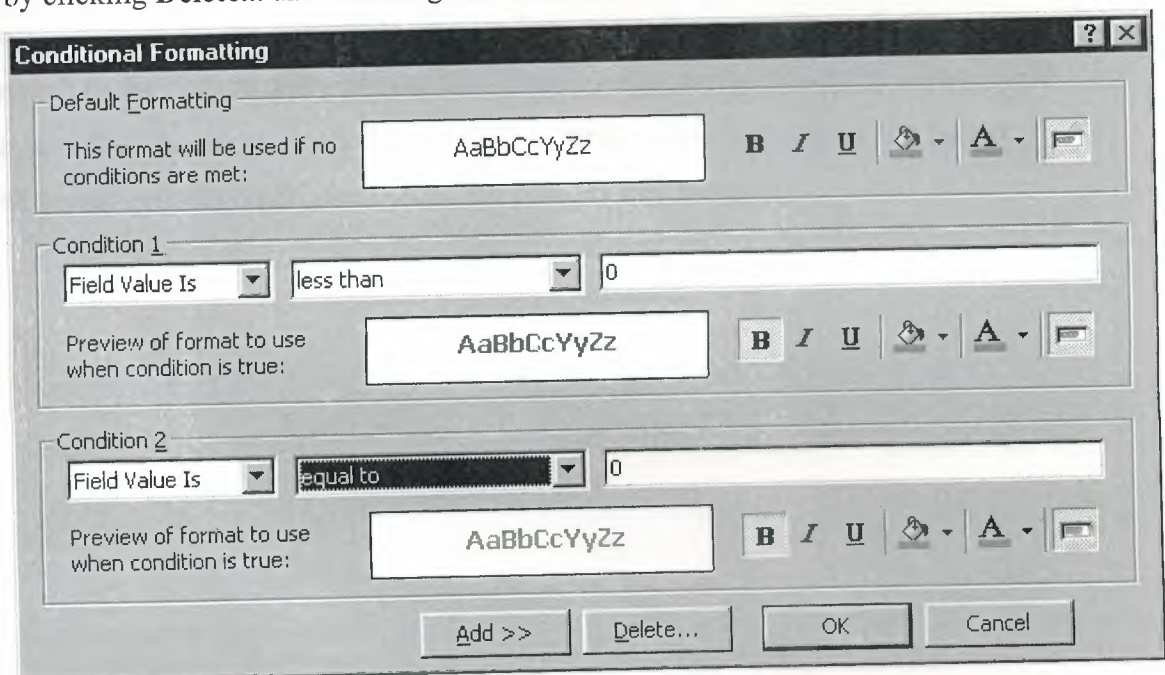


- *Add new tabs or delete tabs* by right-clicking in the tab area and choosing **Insert Page** or **Delete Page** from the shortcut menu.
- *Reorder the tabs* by right-clicking on the tab control and selecting **Page Order**.
- *Rename tabs* by double-clicking on a tab and changing the **Name** property under the **Other** tab.

2.8.2 Conditional Formatting

Special formatting that depends on the control's value can be added to text boxes, lists, and combo boxes. A default value can set along with up to three conditional formats. To add conditional formatting to a control element, follow these steps:

1. Select the control that the formatting should be applied to and select **Format|Conditional Formatting** from the menu bar.
2. Under **Condition 1**, select one of the following condition types:
 - **Field Value Is** applies formatting based upon the value of the control. Select a comparison type from the second drop-down menu and enter a value in the final text box.
 - **Expression Is** applies formatting if the expression is true. Enter a value in the text box and the formatting will be added if the value matches the expression.
 - **Field Has Focus** will apply the formatting as soon as the field has focus.
3. Add additional conditions by clicking the **Add >>** button and delete conditions by clicking **Delete...** and checking the conditions to erase.



2.8.3 Password Text Fields

To modify a text box so each character appears as an asterisk as the user types in the information, select the text field in Design View and click **Properties**. Under the **Data** tab, click in the **Input Mask** field and then click the button [...] that appears. Choose "Password" from the list of input masks and click **Finish**. Although the user will only see asterisks for each character that is typed, the actual characters will be saved in the database.

2.8.4 Change Control Type

If you decide the type of a control needs to be changed, this can be done without deleting the existing control and creating a new one although not every control type can be converted and those that can have a limited number of types they can be converted to. To change the control type, select the control on the form in Design View and choose **Format|Change To** from the menu bar. Select one of the control types that is not grayed out.

2.8.5 Multiple Primary Keys

To select two fields for the composite primary key, move the mouse over the gray column next to the field names and note that it becomes an arrow. Click the mouse, hold it down, and drag it over all fields that should be primary keys and release the button. With the multiple fields highlighted, click the primary key button.

CHAPTER 3

3 USER MANUAL

3.1 Relationships

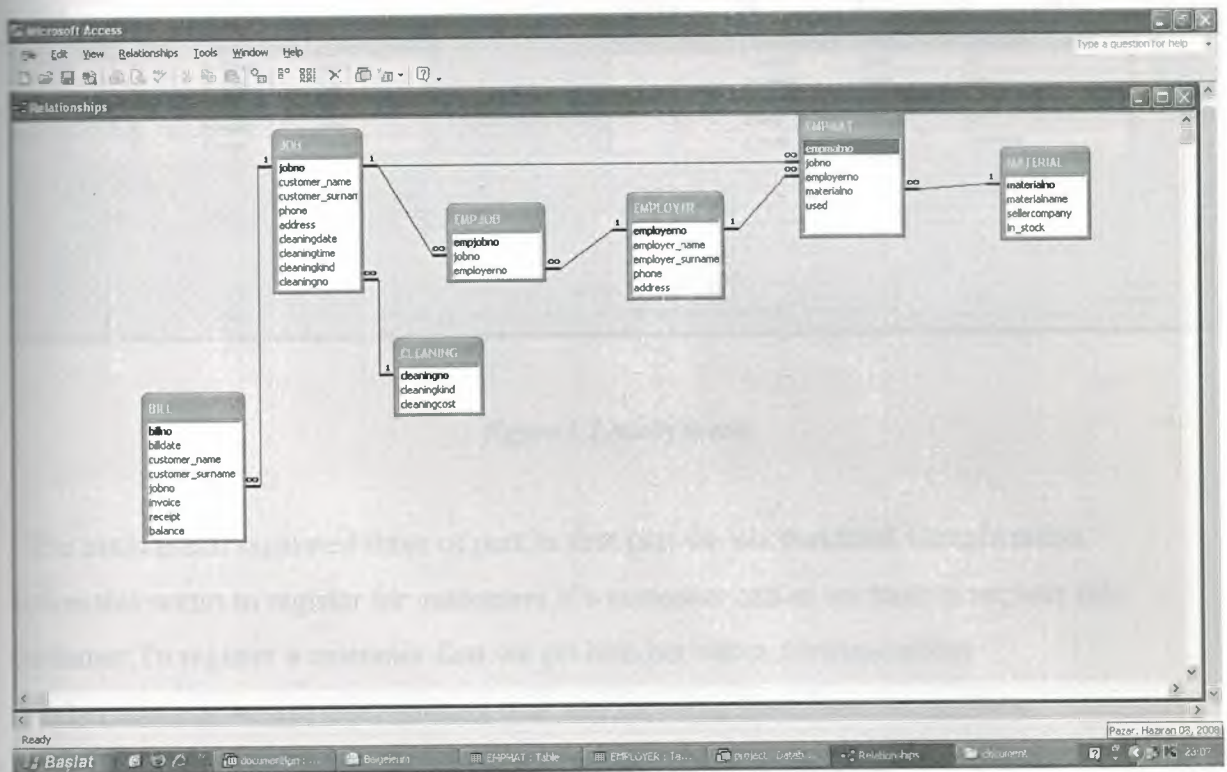


figure 3.1. relationships

The relationships tables on the microsoft access screen gives us information about tables will use for the project and theirs relationships.

3.2 Main Menu

The screenshot shows a software application titled "MAIN MENU". It is divided into two main sections. The top section is for customer registration, featuring a table with columns: jobno, customer_name, customer_surname, phone, address, cleaningdate, cleaningtime, and cleaningkind. Below this table are buttons for "NEW REGISTRATION", "SAVE", "UPDATE", "CANCEL", and "DELETE". The bottom section is for employer selection, with input fields for "JOBNO" and "EMPLOYER_SURNAME". It contains two tables: one for selecting employers (columns: empjob, jobno, employer_name, employer_surname) and another for selecting materials (columns: empmatno, jobno, employer_name, employer_surname, materialname, used). Below these tables are buttons for "ADD", "SAVE", "UPDATE", and "DELETE". At the very bottom, there is a "SEARCH" input field and an "EXIT" button.

jobno	customer_name	customer_surname	phone	address	cleaningdate	cleaningtime	cleaningkind
3	hasan	küsmez	533455667	lefkosa	10/12/2008	12/30/1899 12:30:00	home
4	mustafa	yavuz	5443457668	lefkosa	12/10/2008	12/30/1899 13:30:00	home
5	nuriye	soydan	5334454467	lefkosa	12/12/2008	12/30/1899 15:30:00	office
6	mehmet	öz	5334565643	gine	10/12/2008	12/30/1899 16:00:00	office
7	haydar	deveci	5334556654	magosa	10/11/2008	12/30/1899 17:30:00	store

empjob	jobno	employer_name	employer_surname
1	3	mehmet	onurlu
2	3	ahmet	sari
3	3	onur	tan
4	4	haydar	sami
5	4	burak	aktan

empmatno	jobno	employer_name	employer_surname	materialname	used
1	3	mehmet	onurlu	cil	5
2	3	mehmet	onurlu	ajax	3
3	3	ahmet	sari	muscle	2
4	3	onur	tan	cil	1
5	5	haydar	sami	durumoods	5

figure3.2.main menu

The main menu separated three of part. In first part we see that there is registration section. this section to register for customers. If a customer call us we have to register this customer. To register a customer first we get him/her name, surname, phone number, address, workin date, working time and cleaning kind. After we get these information we push the button 'new registration'. There are some buttons also in section one to update, save or delete a registration. After getting the informations about cleaning kind and time we set employers to work in this job.

Second part of main menu (left bottom on the screen) choosing the employers to send for working. In turn we write jobno and employer surname. After choosing the employers now we can choose which materials for which employers.

In section three of main menu is choosing the materials. We write first jobno then employer name, surname and using material name and the last one is quantity of using material. We push the button add we can add this information for database.

There is a search button bottom of the screen also. This button is for searching another informations like customer information, employer information, billing, material information and cleanin information. The last button is the exit. when we push this button the program will halt.



```

In a
sec:
ser
scr
clo

```

3.3 Search Menu

3.3.1 Customers page

jobno	customer_name	customer_surname	phone	address	cleaningdate	cleaningtime	cleaningkind
3	hasan	kümez	5334455667	lefkosa	10/12/2008	12/30/1899 12:30:00	home
4	mustafa	yavuz	5443457668	lefkosa	12/10/2008	12/30/1899 13:30:00	home
5	nuriye	soydan	5334454467	lefkosa	12/12/2008	12/30/1899 15:30:00	office
6	mehmet	öz	5334565643	gine	10/12/2008	12/30/1899 16:00:00	office
7	haydar	deveci	5334556654	magosa	10/11/2008	12/30/1899 17:30:00	store

employerno	employer_name	employer_surname	phone	address
2	mehmet	onurlu	5334454334	lefkosa
3	haydar	sani	5333344556	lefkosa
4	ahmet	sarı	5334554332	lefkosa
5	onur	tan	5334432234	gine

figure 3.3.customers page

In search menu there are five category for searching.the first one is the customers.In this section we can search any customer by jobno ,name,surname,phone or cleaning date by search fields which replaced top of the screen.There is employers table bottom of the screen.this table is to find out which employers are worked for signed jobno.We can see clearly how many employers worked who are these and what are theirs informations.

3.3.2 Bill Page

SEARCH

CUSTOMERS **BILL** EMPLOYERS MATERIAL CLEANING KIND

SEARCH BY:

BILL NO BILL DATE NAME SURNAME JOBNO

billno	billdate	customer_name	customer_surname	jobno	invoice	receipt	balance
2	12/11/2008	hasan	kümeze	3	30	20	10
3	12/15/2008	mustafa	yavuz	4	30	30	0
4	12/12/2008	nuriye	soydan	5	40	40	0
5	12/22/2008	naciye	kaderli	8	40	0	40

PAYING

BALANCE

jobno	customer_name	customer_surname	phone	address	clearingdate	clearingtime	clearingkind
3	hasan	kümeze	533455687	lefkosa	10/12/2008	12/30/1899 12:30:00	home
4	mustafa	yavuz	5443457668	lefkosa	12/10/2008	12/30/1899 13:30:00	home
5	nuriye	soydan	5334454467	lefkosa	12/12/2008	12/30/1899 15:30:00	office
6	mehmet	öz	5334565643	gine	10/12/2008	12/30/1899 16:00:00	office
7	haydar	deveci	5334555654	magosa	10/11/2008	12/30/1899 17:30:00	store

figure 3.4.bill page

The second section of search menu is for billing. After the work we have to give a invoice to customer. In main menu when we register a customer we typing the cleaning kind the cleaning cost is typed into invoice field in bill table automatically. We can find out the customer that we want to give a receipt by search alternative again. After we find out the customers we can see how much borrow for customer. When the time for paying we ask to the customer how much will him/her pay for own job. After decision the quantity payment now we can type the quantity of maney into the paying field on the left of the screen and we push the button balance.

3.3.3 Employees Pages

7 SEARCH

CUSTOMERS | BILL | EMPLOYERS | MATERIAL | CLEANING KIND

SEARCH BY:

EMPLOYERNO NAME SURNAME PHONE

employerino	employer_name	employer_surname	phone	address
2	mehmet	onurlu	5334454334	lefkosa
3	haydar	sami	5333344556	lefkosa
4	ahmet	sani	5334554332	lefkosa
5	onur	tan	5334432234	giine
6	burak	aktan	5334432233	lefkosa

NAME SURNAME PHONE ADDRESS

ADD SAVE UPDATE DELETE

empmatno	jobno	employer_name	employer_surname	materialname	used
1	3	mehmet	onurlu	oil	5
2	3	mehmet	onurlu	ajax	3
3	3	ahmet	sani	muscle	2
4	3	onur	tan	oil	1
5	5	haydar	sami	durumoods	5

figure 3.5 employees page

The third section of the search menu is employers table. In this section there are two sections. The first one is searching the employers by the searching alternative. Also there are some buttons to add the new employer or to delete the leaving employer. There is material information at the bottom of the table. This table shows us which employer used which materials and how many. We registered this information when we register the employers for working in main menu.

3.3.4 Material Page

MATERIAL NO	NAME	SELLER COMPANY	IN STOCK
1	cil	aktaş ltd.	50
2	ajax	aktaş ltd.	33
3	duuu moods	aktaş ltd.	42
4	musscle	aktaş ltd.	15

MATERIAL NAME

SELLER COMPANY

IN STOCK

ADD SAVE UPDATE DELETE

figure 3.6.material page

The fourth section of search menu is material. This table gives us material names, seller company we bought them and how many material is there in the stock. We can search for a material via searching fields again. When we register an employer on the main menu we type how many pattern will use the employer. During the time we type the quantity of the material kind decreases the quantity in stock balance in the material section. Also we can add new materials when the material finish in stock. There are some buttons for register new materials in this section.

3.3.5 Cleaning Kind Page

SEARCH!

CUSTOMERS | BILL | EMPLOYERS | MATERIAL | CLEANING KIND

SEARCH BY:

CLEANING NO KIND COST

cleaningno	cleaningkind	cleaningcost
1	home	30
2	office	40
3	store	70

CLEANING KIND

CLEANING COST

ADD SAVE UPDATE DELETE

figure 3.7.cleaning kind page

The fifth and last section is cleaning for search menu. In this section gives us informations about cleaning kind and its cost. We can search any kind of cleaning and its cost via searching fields. Also we can add new cleaning type or update the cost for any existing cleaning kind or we can delete for any cleaning kind.

CONCLUSION

I used delphi for this program because of delphi is usable to prepare a databank programming. Also Microsoft Access was easy to use and clear for my development.

First time it was hard to use delphi for me because it is so detail program but later I examined some books and some internet addresses and I learned necessary informations about Delphi to develop my project.

This development teach me how can I use Delphi and how can I develop a databank programming via using Microsoft Access. Now I know so many things about delphi and microsoft Access.

My development can be used in any cleaning company. Because while I were developing my program I researched many cleaning companies and I examine how do these companies work.

REFERENCES

[1] Neil Moffat , "Delphi Basics" from the World Wide Web

<http://www.delphibasics.co.uk/index.html>

[2] © FGCU 2007. Florida Gulf Coast University is an equal opportunity/affirmative action institution from the World Wide Web

<http://www.fgcu.edu/support/office2000/access/>

[3] Delphi 7 Ezel Balkan Book

APPENDIX

```
program Project1;
```

```
uses
```

```
Forms,
```

```
Unit1 in 'Unit1.pas' {Form1},
```

```
Unit2 in 'Unit2.pas' {Form2},
```

```
Unit3 in 'Unit3.pas' {Form3};
```

```
{ $R *.res }
```

```
begin
```

```
Application.Initialize;
```

```
Application.CreateForm(TForm3, Form3);
```

```
Application.CreateForm(TForm1, Form1);
```

```
Application.CreateForm(TForm2, Form2);
```

```
Application.Run;
```

```
end.
```

```
unit Unit1;
```

interface

uses

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

type

```
TForm1 = class(TForm)
  PageControl1: TPageControl;
  TabSheet1: TTabSheet;
  TabSheet2: TTabSheet;
  Panel1: TPanel;
  ADOConnection1: TADOConnection;
  DataSource1: TDataSource;
  DBGrid1: TDBGrid;
  Panel2: TPanel;
  DBEdit2: TDBEdit;
  DBEdit3: TDBEdit;
  DBEdit4: TDBEdit;
  Button1: TButton;
  Button2: TButton;
  Button3: TButton;
  Button4: TButton;
  Button5: TButton;
  Panel3: TPanel;
  DBGrid2: TDBGrid;
  Edit1: TEdit;
  Label1: TLabel;
  ADOQuery1: TADOQuery;
  Panel4: TPanel;
  ADOQuery2: TADOQuery;
  DataSource2: TDataSource;
  Edit2: TEdit;
```

ADOQuery3: TADOQuery;
DataSource3: TDataSource;
DBGrid4: TDBGrid;
Button6: TButton;
Button8: TButton;
Button9: TButton;
Button10: TButton;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
Label3: TLabel;
ADOQuery4: TADOQuery;
DataSource4: TDataSource;
DBGrid5: TDBGrid;
DBGrid3: TDBGrid;
ADOQuery5: TADOQuery;
DataSource5: TDataSource;
DBGrid6: TDBGrid;
Label2: TLabel;
Label4: TLabel;
Label5: TLabel;
Button11: TButton;
Button12: TButton;
ADOQuery6: TADOQuery;
DataSource6: TDataSource;
DBGrid7: TDBGrid;
Edit3: TEdit;
Button13: TButton;
DBLookupComboBox1: TDBLookupComboBox;
MaskEdit2: TMaskEdit;
MaskEdit1: TMaskEdit;
MaskEdit3: TMaskEdit;
Button14: TButton;
Button15: TButton;

Button16: TButton;
Button17: TButton;
Button18: TButton;
Button19: TButton;
DBLookupComboBox2: TDBLookupComboBox;
Edit7: TEdit;
DBEdit6: TDBEdit;
Edit8: TEdit;
Button22: TButton;
Button21: TButton;
Button23: TButton;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
MaskEdit4: TMaskEdit;
DBEdit1: TDBEdit;
DBEdit5: TDBEdit;
DBEdit7: TDBEdit;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Button7: TButton;
DBLookupListBox1: TDBLookupListBox;
Button24: TButton;
Panel5: TPanel;
Button20: TButton;
Button25: TButton;
Label16: TLabel;
Label17: TLabel;
procedure Edit1Change(Sender: TObject);

```
procedure TabSheet2Show(Sender: TObject);
procedure TabSheet1Show(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure Button9Click(Sender: TObject);
procedure Button10Click(Sender: TObject);
procedure DBGrid3CellClick(Column: TColumn);
procedure Edit4Change(Sender: TObject);
procedure Edit5Change(Sender: TObject);
procedure Edit6Change(Sender: TObject);
procedure DBGrid5CellClick(Column: TColumn);
procedure DBGrid4CellClick(Column: TColumn);
procedure Button11Click(Sender: TObject);
procedure Button12Click(Sender: TObject);
procedure Edit3Change(Sender: TObject);
procedure Button13Click(Sender: TObject);
procedure MaskEdit1Click(Sender: TObject);
procedure MaskEdit3Click(Sender: TObject);
procedure Button14Click(Sender: TObject);
procedure Button15Click(Sender: TObject);
procedure Button16Click(Sender: TObject);
procedure Button17Click(Sender: TObject);
procedure Button18Click(Sender: TObject);
procedure Button19Click(Sender: TObject);
procedure Button20Click(Sender: TObject);
procedure DBLookupComboBox2Click(Sender: TObject);
procedure Button22Click(Sender: TObject);
procedure Button21Click(Sender: TObject);
procedure Button23Click(Sender: TObject);
procedure MaskEdit4Change(Sender: TObject);
procedure MaskEdit1Change(Sender: TObject);
procedure MaskEdit2Change(Sender: TObject);
procedure Button1Click(Sender: TObject);
```

```

procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure Button24Click(Sender: TObject);
procedure Button25Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);

```

```

private

```

```

    procedure add_click;
    procedure update_click;
    procedure empmat_add_click;
    { Private declarations }

```

```

public

```

```

    { Public declarations }

```

```

end;

```

```

var

```

```

    Form1: TForm1;
    update1:integer;
    degisken:integer;
    implementation

```

```

uses Unit2;

```

```

{$R *.dfm}

```

```

procedure TForm1.Edit1Change(Sender: TObject);

```

```

var

```

```

    jobno:integer;

```

```

begin

```



```

adoquery1.close;
adoquery1.SQL.Clear;
adoquery1.SQL.Add('select * from job where jobno like'+#39+(edit1.Text)+'%'+#39);
adoquery1.Open;
adoquery1.Refresh;
jobno:=adoquery1.fieldbyname('jobno').AsInteger;
edit2.Text:=inttostr(jobno);
edit3.Text:=inttostr(jobno);
end;

```

```

procedure TForm1.TabSheet2Show(Sender: TObject);

```

```

var

```

```

jobno:integer;

```

```

begin

```

```

    maskedit3.Text:=timetostr(time);

```

```

    edit1.SetFocus;

```

```

    adoquery2.FieldByName('givenjob').Visible:=true;

```

```

    repeat

```

```

        adoquery1.Next;

```

```

    until (adoquery1.Eof = true);

```

```

    jobno:=adoquery1.fieldbyname('jobno').AsInteger;

```

```

    edit2.Text:=inttostr(jobno);

```

```

    edit3.Text:=inttostr(jobno);

```

```

    adoquery4.Active:=false;

```

```

    adoquery4.Active:=true;

```

```

    adoquery3.Active:=false;

```

```

    adoquery3.Active:=true;

```

```

    adoquery5.Active:=false;

```

```
adoquery5.Active:=true;
```

```
adoquery2.Active:=false;
```

```
adoquery2.Active:=true;
```

```
adoquery6.Active:=false;
```

```
adoquery6.Active:=true;
```

```
end;
```

```
procedure TForm1.TabSheet1Show(Sender: TObject);
```

```
begin
```

```
edit1.Clear;
```

```
maskedit1.Text:=datetostr(date);
```

```
dbedit5.Text:=maskedit1.Text;
```

```
repeat
```

```
adoquery1.Next;
```

```
until (adoquery1.Eof = true)
```

```
end;
```

```
procedure TForm1.Edit2Change(Sender: TObject);
```

```
begin
```

```
adoquery3.close;
```

```
adoquery3.SQL.Clear;
```

```
adoquery3.SQL.Add('select
```

```
jobno,employer_name,employer_surname,employerno,leavingtime,returningtime,empjo
```

```
bno from empjob where jobno like'+#39+(edit2.Text)+'%'+#39);
```

```
adoquery3.Open;
```

```
adoquery3.FieldByName('empjobno').Visible:=false;
```

```
end;
```

```
procedure TForm1.Button6Click(Sender: TObject);
```

```

begin
var
empjob_employerno:integer;
employer_employerno:integer;
begin
    if degisken=2 then
        if update1=2 then
            begin
                adoquery3.Cancel;
                edit1.SetFocus;
                update1:=0;
                exit;
            end;
        until (true)
        if adoquery3.FieldName('employer_surname').AsString="" then exit;
        button6.Enabled:=false;

        while (true)
        begin
            if update1=1 then
                begin
                    adoquery3.Cancel;
                    dbgrid5.Visible:=true;
                    label2.Visible:=true;
                    dbgrid3.Visible:=false;
                    label5.Visible:=false;

                    adoquery3.Cancel;
                    label3.Visible:=true;
                    edit4.Visible:=true;
                    edit5.Visible:=true;
                    edit6.Visible:=true;

                    if degisken=1 then

```



```

begin
  if dbgrid3.Visible=false then dbgrid3.Visible:=true;
  if dbgrid6.Visible=false then dbgrid6.Visible:=true;
  button12.Visible:=true;
end;

if degisken=0 then button11.Visible:=true;

update1:=0;
exit;
end;

repeat
  adoquery2.Next;
until (adoquery2.Eof = true);
empjob_employerno:=adoquery3.FieldByName('employerno').AsInteger;
employer_employerno:= adoquery2.FieldByName('employerno').AsInteger;

while (employer_employerno<>empjob_employerno) do
  begin
    adoquery2.Prior;
    employer_employerno:= adoquery2.FieldByName('employerno').AsInteger;
  end;
adoquery3.cancel;

adoquery2.Edit;
adoquery2.UpdateRecord;
adoquery2.FieldByName('givenjob').AsBoolean:=false;
adoquery2.Post;
adoquery4.Active:=false;
adoquery4.Active:=true;
  adoquery5.Active:=false;
adoquery5.Active:=true;

```

```

edit1.SetFocus;

end;

procedure TForm1.Button8Click(Sender: TObject);
var
a:word;
empjob_employerno:integer;
employer_employerno:integer;
begin
if adoquery3.FieldByName('employer_surname').AsString="" then
begin
edit1.SetFocus;
exit;
end;
if adoquery3.FieldByName('returningtime').AsString="" then
begin
repeat
adoquery2.Next;
until (adoquery2.Eof = true);
empjob_employerno:=adoquery3.FieldByName('employerno').AsInteger;
employer_employerno:= adoquery2.FieldByName('employerno').AsInteger;

while (employer_employerno<>empjob_employerno) do
begin
adoquery2.Prior;
employer_employerno:= adoquery2.FieldByName('employerno').AsInteger;
end;

```

```

a:=application.messagebox('are you sure to send back?','warning',36);
if (a=idyes)then
  begin
adoquery3.Delete;
adoquery3.prior;
adoquery2.Edit;
adoquery2.UpdateRecord;
adoquery2.FieldName('givenjob').AsBoolean:=false;
adoquery2.Post;
end;
adoquery4.Active:=false;
adoquery4.Active:=true;
adoquery5.Active:=false;
adoquery5.Active:=true;
edit1.SetFocus;
end
else
begin
showmessage ('this job seems completed');
showmessage('you can use "Add or Delete" button');
end;
end;

procedure TForm1.Button9Click(Sender: TObject);
begin
if adoquery3.FieldName('employer_surname').AsString="" then
begin
edit1.SetFocus;
exit;
end;
button6.Enabled:=false;

if update1=2 then
begin

```



```

adoquery3.post;

edit1.SetFocus;
update1:=0;
exit;
end;

if update1=1 then
begin
dbgrid5.Visible:=true;
label2.Visible:=true;
dbgrid3.Visible:=false;
label5.Visible:=false;
label3.Visible:=true;
edit4.Visible:=true;
edit5.Visible:=true;
edit6.Visible:=true;

    if degisken=1 then
begin
if dbgrid3.Visible=false then dbgrid3.Visible:=true;
if dbgrid6.Visible=false then dbgrid6.Visible:=true;
button12.Visible:=true;
end;
if degisken=0 then button11.Visible:=true;
update1:=0;
end;
try
adoquery3.Post;
except
begin
showmessage('this table is not in "edit" or "insert" mode')
end;
end;

```

```

adoquery2.Active:=false;
adoquery2.Active:=true;
adoquery4.Active:=false;
adoquery4.Active:=true;
adoquery5.Active:=false;
adoquery5.Active:=true;

```

```

edit1.SetFocus;

```

```

edit1.SetFocus;

```

```

end;

```

```

procedure TForm1.add_click();
var
  aktar1:string;
  aktar2:string;
  aktar3:integer;
  aktar4:integer;
  employer_employerno:integer;
  convenient_employerno:integer;
begin
  repeat
    adoquery3.Next;
  until (adoquery3.Eof = true);
  while adoquery3.Bof<>true do
    begin
      if adoquery3.FieldName('returningtime').AsString<>" then
        begin
          showmessage('this job seems complited');
          showmessage('you can use "Add or Delete" button');
          exit;
        end;
      adoquery3.Prior;
    end;
  end;
end;

```

```

end;

if adoquery3.FieldName('returningtime').AsString<>" then
begin
showmessage('this job seems complited');
showmessage('you can use "Add or Delete" button');
exit;
end;

button6.Enabled:=true;
adoquery3.Append;
aktar3:=strtoint(edit2.text);
adoquery3.FieldName('jobno').Asinteger:=aktar3;
aktar1:= adoquery4.FieldName('employer_name').AsString;
adoquery3.FieldName('employer_name').AsString:=aktar1;
aktar2:=adoquery4.FieldName('employer_surname').AsString;
adoquery3.FieldName('employer_surname').AsString:=aktar2;
aktar4:= adoquery4.FieldName('employerno').AsInteger;
adoquery3.FieldName('employerno').AsInteger:=aktar4;

repeat
adoquery2.Next;
until (adoquery2.Eof = true);
convenient_employerno:=adoquery4.FieldName('employerno').AsInteger;
employer_employerno:= adoquery2.FieldName('employerno').AsInteger;

while (employer_employerno<>convenient_employerno) do
begin
adoquery2.Prior;
employer_employerno:= adoquery2.FieldName('employerno').AsInteger;
end;

```



```

adoquery2.Edit;
adoquery2.UpdateRecord;
adoquery2.FieldByName('givenjob').AsBoolean:=true;
adoquery2.Post;

adoquery4.Active:=false;
adoquery4.Active:=true;
adoquery5.Active:=false;
adoquery5.Active:=true;

end;

procedure TForm1.update_click();
var
aktar1:string;
aktar2:string;
aktar3:integer;
aktar4:integer;
begin
    aktar3:=strtoint(edit2.text);
    adoquery3.FieldByName('jobno').Asinteger:=aktar3;
    aktar1:= adoquery2.FieldByName('employer_name').AsString;
    adoquery3.FieldByName('employer_name').AsString:=aktar1;
    aktar2:=adoquery2.FieldByName('employer_surname').AsString;
    adoquery3.FieldByName('employer_surname').AsString:=aktar2;
    aktar4:= adoquery2.FieldByName('employerno').AsInteger;
    adoquery3.FieldByName('employerno').AsInteger:=aktar4;
end;

procedure TForm1.Button10Click(Sender: TObject);
var
empjob_employerno:integer;
employer_employerno:integer;
begin

```

```

if adoquery3.FieldName('employer_surname').AsString="" then
begin
    edit1.SetFocus;
    exit;
end;

if adoquery3.FieldName('returningtime').AsString="" then
begin
    showmessage('you can update after this job completed');
    edit1.SetFocus;
    exit;
end;

update1:=1;
button6.Enabled:=true;
dbgrid5.Visible:=false;
label2.Visible:=false;
dbgrid6.Visible:=false;
label4.Visible:=false;
dbgrid3.Visible:=true;
label5.Visible:=true;

label3.Visible:=false;
edit4.Visible:=false;
edit5.Visible:=false;
edit6.Visible:=false;
button11.Visible:=false;
button12.Visible:=false;

repeat
    adoquery2.Next;
until (adoquery2.Eof = true);
empjob_employerno:=adoquery3.FieldName('employerno').AsInteger;
employer_employerno:= adoquery2.FieldName('employerno').AsInteger;

```

```

while (employer_employerno<>empjob_employerno) do
begin
    adoquery2.Prior;
    employer_employerno:= adoquery2.FieldByName('employerno').AsInteger;
end;

adoquery3.edit;
adoquery3.UpdateRecord;
edit1.SetFocus;
end;

procedure TForm1.DBGrid3CellClick(Column: TColumn);
begin
    if update1=2 then
    begin
        update_click;
        exit;
    end;

    if update1=1 then
        update_click;

end;

procedure TForm1.Edit4Change(Sender: TObject);
begin
    adoquery4.close;
    adoquery4.SQL.Clear;
    adoquery4.SQL.Add('select employerno,employer_name,employer_surname,givenjob
from employer where givenjob=false and employerno like'+#39+(edit4.Text)+'%'+#39
);
    adoquery4.Open;

```



```

adoquery4.Active:=false;
adoquery4.Active:=true;

if edit4.Text="" then
begin
    adoquery4.close;
    adoquery4.SQL.Clear;
    adoquery4.SQL.Add('select employerno,employer_name,employer_surname,givenjob
from employer where givenjob=false order by employer_name');
    adoquery4.Open;
end;

end;

```

```

procedure TForm1.Edit5Change(Sender: TObject);
begin
    adoquery4.close;
    adoquery4.SQL.Clear;
    adoquery4.SQL.Add('select employerno,employer_name,employer_surname,givenjob
from employer where givenjob=false and employer_name
like'+#39+(edit5.Text)+'%'+#39);
    adoquery4.Open;
    adoquery4.FieldName('givenjob').Visible:=false;

```

```

    if edit5.Text="" then
begin
    adoquery4.close;
    adoquery4.SQL.Clear;
    adoquery4.SQL.Add('select employerno,employer_name,employer_surname,givenjob
from employer where givenjob=false order by employer_name');
    adoquery4.Open;
end;

```

end;

procedure TForm1.Edit6Change(Sender: TObject);

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select employerno,employer_name,employer_surname,givenjob
from employer where givenjob=false and employer_surname
like'+#39+(edit6.Text)+'%'+#39);

adoquery4.Open;

adoquery4.FieldByName('givenjob').Visible:=false;

if edit6.Text="" then

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select employerno,employer_name,employer_surname,givenjob
from employer where givenjob=false order by employer_name');

adoquery4.Open;

end;

end;

procedure TForm1.DBGrid5CellClick(Column: TColumn);

begin

add_click;

end;

procedure TForm1.DBGrid4CellClick(Column: TColumn);

begin

button6.Enabled:=false;

edit7.Text:="";

if degisken=3 then

```

empmat_add_click;

end;

procedure TForm1.Button11Click(Sender: TObject);
begin
    degisken:=1;
    label4.Visible:=true;
    label5.Visible:=true;
    dbgrid6.Visible:=true;
    dbgrid3.Visible:=true;
    button12.Visible:=true;
    button11.Visible:=false;
    edit1.setfocus;

end;

procedure TForm1.Button12Click(Sender: TObject);
begin
    degisken:=0;
    label4.Visible:=false;
    label5.Visible:=false;
    dbgrid3.Visible:=false;
    dbgrid6.Visible:=false;
    button12.Visible:=false;
    button11.Visible:=true;
    edit1.SetFocus;

end;

procedure TForm1.Edit3Change(Sender: TObject);
begin
    adoquery6.close;
    adoquery6.SQL.Clear;

```



```

adoquery6.SQL.Add('select
jobno,employer_name,employer_surname,employerno,materialname,used from empmat
where jobno like'+#39+(edit3.Text)+'%'+#39);
adoquery6.Open;

end;

procedure TForm1.Button13Click(Sender: TObject);
var
jobno:integer;
begin
edit1.Text:="";
repeat
adoquery1.Next;
until (adoquery1.Eof = true);
jobno:= adoquery1.FieldByName('jobno').AsInteger;
edit1.Text:=inttostr(jobno);
edit1.SetFocus;
end;

procedure TForm1.MaskEdit1Click(Sender: TObject);
begin
maskedit1.Text:=datetostr(date);
dbedit5.Text:=maskedit1.Text;
end;

procedure TForm1.MaskEdit3Click(Sender: TObject);
begin
maskedit3.Text:=timetostr(time);

end;

procedure TForm1.Button14Click(Sender: TObject);
var

```

```
leaving:string;
```

```
begin
```

```
    if update1=2 then
```

```
    begin
```

```
        leaving:=maskedit3.Text;
```

```
        adoquery3.FieldByName('leavingtime').AsString:=leaving;
```

```
        exit;
```

```
    end;
```

```
if update1=1 then
```

```
begin
```

```
adoquery3.Edit;
```

```
    adoquery3.UpdateRecord;
```

```
    adoquery3.FieldByName('leavingtime').AsString:=maskedit3.Text;
```

```
    adoquery3.Post;
```

```
    dbgrid5.Visible:=true;
```

```
label2.Visible:=true;
```

```
dbgrid3.Visible:=false;
```

```
label5.Visible:=false;
```

```
    if degisken=1 then
```

```
    begin
```

```
        label4.Visible:=true;
```

```
        label5.Visible:=true;
```

```
        dbgrid6.Visible:=true;
```

```
        dbgrid3.Visible:=true;
```

```
        button12.Visible:=true;
```

```
    end;
```

```
    if degisken=0 then
```

```
    begin
```

```
        button11.Visible:=true;
```

```
        dbgrid3.Visible:=false;
```

```
        label5.Visible:=false;
```

end;

button6.Enabled:=false;

update1:=0;

exit;

end;

repeat

adoquery3.Next;

until (adoquery3.Eof = true);

while adoquery3.bof <> true do

begin

if adoquery3.FieldName('leavingtime').AsString="" then

begin

adoquery3.Edit;

adoquery3.UpdateRecord;

adoquery3.FieldName('leavingtime').AsString:=maskedit3.Text;

adoquery3.Post;

end;

adoquery3.Prior;

end;

button6.Enabled:=false;

edit1.SetFocus;

end;

procedure TForm1.Button15Click(Sender: TObject);

var

empjob_employerno:integer;

employer_employerno:integer;

returning:string;

begin


```
if update1=2 then
```

```
begin
```

```
returning:=maskedit3.Text;
```

```
adoquery3.FieldByName('returningtime').AsString:=returning;
```

```
exit;
```

```
end;
```

```
if update1=1 then
```

```
begin
```

```
if adoquery3.FieldByName('leavingtime').AsString="" then
```

```
begin
```

```
showmessage('you have to fill leaving time first');
```

```
exit;
```

```
end;
```

```
adoquery3.Edit;
```

```
adoquery3.UpdateRecord;
```

```
adoquery3.FieldByName('returningtime').AsString:=maskedit3.Text;
```

```
adoquery3.Post;
```

```
dbgrid5.Visible:=true;
```

```
label2.Visible:=true;
```

```
dbgrid3.Visible:=false;
```

```
label5.Visible:=false;
```

```
if degisken=1 then
```

```
begin
```

```
label4.Visible:=true;
```

```
label5.Visible:=true;
```

```
dbgrid6.Visible:=true;
```

```
dbgrid3.Visible:=true;
```

```
button12.Visible:=true;
```

```
end;
```

```

if degisken=0 then
begin
button11.Visible:=true;
dbgrid3.Visible:=false;
label5.Visible:=false;
end;

if update1=2 then
begin
adoquery3.FieldByName('returningtime').AsString:=maskedit3.Text;
end;

button6.Enabled:=false;
update1:=0;
exit;
end;

if adoquery3.FieldByName('leavingtime').AsString="" then
begin
showmessage('you have to fill leaving time first');
exit;
end;

repeat
adoquery3.Next;
until (adoquery3.Eof = true);

repeat
if adoquery3.FieldByName('returningtime').AsString<>" then
begin
degisken:=2;
end;
adoquery3.Prior;
until(adoquery3.Bof=true);

```

```
if degisken=2 then
```

```
begin
```

```
repeat
```

```
adoquery3.Next;
```

```
until (adoquery3.Eof = true);
```

```
while adoquery3.bof <> true do
```

```
begin
```

```
if adoquery3.FieldName('returningtime').AsString="" then
```

```
begin
```

```
adoquery3.Edit;
```

```
adoquery3.UpdateRecord;
```

```
adoquery3.FieldName('returningtime').AsString:=maskedit3.Text;
```

```
adoquery3.Post;
```

```
end;
```

```
adoquery3.Prior;
```

```
end;
```

```
adoquery5.Active:=false;
```

```
adoquery5.Active:=true;
```

```
button6.Enabled:=false;
```

```
edit1.SetFocus;
```

```
degisken:=0;
```

```
end;
```

```
repeat
```



```

adoquery3.Next;
until (adoquery3.Eof = true);
    repeat
adoquery2.Next;
until (adoquery2.Eof = true);

while adoquery3.bof <> true do
begin
    if adoquery3.FieldName('returningtime').AsString="" then
    begin
        empjob_employerno:=adoquery3.FieldName('employerno').AsInteger;
        employer_employerno:= adoquery2.FieldName('employerno').AsInteger;
        while (employer_employerno<>empjob_employerno) do
            begin
                adoquery2.Prior;
                employer_employerno:=
adoquery2.FieldName('employerno').AsInteger;
            end;
            adoquery2.Edit;
            adoquery2.UpdateRecord;
            adoquery2.FieldName('givenjob').AsBoolean:=false;
            adoquery2.Post;

            adoquery3.Edit;
            adoquery3.UpdateRecord;
            adoquery3.FieldName('returningtime').AsString:=maskedit3.Text;
            adoquery3.Post;
        end;
        adoquery3.Prior;
    end;
    adoquery4.Active:=false;
    adoquery4.Active:=true;

```

```

adoquery2.Active:=false;
adoquery2.Active:=true;

adoquery5.Active:=false;
adoquery5.Active:=true;
button6.Enabled:=false;
edit1.SetFocus;
end;

procedure TForm1.Button16Click(Sender: TObject);
begin
  repeat
    adoquery3.Next;
  until (adoquery3.Eof = true);
  while adoquery3.Bof<>true do
  begin
    if adoquery3.FieldName('returningtime').AsString="" then
    begin
      showmessage('you can Add or Delete after returning time will be filled');
      exit;
    end;
    adoquery3.Prior;
  end;
  if adoquery3.FieldName('returningtime').AsString="" then
  begin
    showmessage('this choice for just complited jobs');
    exit;
  end;
  button16.Visible:=false;
  button17.Visible:=true;
  button18.Visible:=true;
  button8.Visible:=false;
  button19.Visible:=true;

```

```
button10.Visible:=false;
```

```
label2.Visible:=false;
```

```
label3.Visible:=false;
```

```
label4.Visible:=false;
```

```
label5.Visible:=false;
```

```
dbgrid3.Visible:=false;
```

```
dbgrid5.Visible:=false;
```

```
dbgrid6.Visible:=false;
```

```
edit4.Visible:=false;
```

```
edit5.Visible:=false;
```

```
edit6.Visible:=false;
```

```
button11.Visible:=false;
```

```
button12.Visible:=false;
```

```
edit1.SetFocus;
```

```
end;
```

```
procedure TForm1.Button17Click(Sender: TObject);
```

```
begin
```

```
update1:=2;
```

```
adoquery3.Append;
```

```
button6.Enabled:=true;
```

```
dbgrid5.Visible:=false;
```

```
label2.Visible:=false;
```

```
dbgrid6.Visible:=false;
```

```
label4.Visible:=false;
```

```
dbgrid3.Visible:=true;
```

```
label5.Visible:=true;
```

```
end;
```

```
procedure TForm1.Button18Click(Sender: TObject);
```

```
var
```



```

a:word;
begin

    a:=application.messagebox('are you sure?','warning',36);
    if (a=idyes)then
    begin
        adoquery3.Delete;
        adoquery3.prior;

        edit1.SetFocus;
    end;

end;

procedure TForm1.Button19Click(Sender: TObject);
begin
    button8.Visible:=true;
    button10.Visible:=true;
    button16.Visible:=true;
    button17.Visible:=false;
    button18.Visible:=false;
    button19.Visible:=false;

    label2.Visible:=true;
    label3.Visible:=true;

    dbgrid5.Visible:=true;

    edit4.Visible:=true;
    edit5.Visible:=true;
    edit6.Visible:=true;

    if degisken=1 then
    begin

```

```

label4.Visible:=true;
label5.Visible:=true;
dbgrid6.Visible:=true;
dbgrid3.Visible:=true;
button12.Visible:=true;
end;
if degisken=0 then
begin
button11.Visible:=true;
dbgrid3.Visible:=false;
label5.Visible:=false;
end;
end;

procedure TForm1.Button20Click(Sender: TObject);
begin
form2.show;
form1.Hide;

end;

procedure TForm1.DBLookupComboBox2Click(Sender: TObject);
begin
edit8.Text:=form2.ADOQuery2.fieldbyname('materialname').AsString;

end;

procedure TForm1.Button22Click(Sender: TObject);
begin
adoquery6.Cancel;

```

```
dblookupcombobox2.Enabled:=false;
```

```
degisken:=0;
```

```
end;
```

```
procedure TForm1.empmat_add_click;
```

```
var
```

```
aktar1:string;
```

```
aktar2:string;
```

```
aktar3:integer;
```

```
aktar4:integer;
```

```
begin
```

```
    aktar3:= adoquery3.FieldByName('jobno').AsInteger;
```

```
    adoquery6.FieldByName('jobno').AsInteger:=aktar3;
```

```
    aktar1:= adoquery3.FieldByName('employer_name').AsString;
```

```
    adoquery6.FieldByName('employer_name').AsString:=aktar1;
```

```
    aktar2:=adoquery3.FieldByName('employer_surname').AsString;
```

```
    adoquery6.FieldByName('employer_surname').AsString:=aktar2;
```

```
    aktar4:= adoquery3.FieldByName('employerno').AsInteger;
```

```
    adoquery6.FieldByName('employerno').AsInteger:=aktar4;
```

```
end;
```

```
procedure TForm1.Button21Click(Sender: TObject);
```

```
var
```

```
edit_qty:integer;
```

```
materialname_material:string;
```

```
remain_material:integer;
```

```
remain_showmaterial:integer;
```

```
begin
```



```

if update1=10 then
begin
adoquery6.Post;
degisken:=0;
update1:=0;
exit;
end;

if
(adoquery6.FieldName('employer_surname').AsString<>")and(adoquery6.FieldName('used').AsInteger<>0) then exit;

if adoquery6.FieldName('materialname').AsString=" then
begin
showmessage('you have to choose a material');
exit;
end;
if edit7.Text=" then
begin
showmessage('you have to type quantity for this material');
exit;
end;

remain_showmaterial:=form2.adoquery2.fieldbyname('expr1001').AsInteger;
if remain_showmaterial < strtoint(edit7.text) then
begin
showmessage('there is no enough this product in _stock');
exit;
end;
repeat

```

```

form2.adoquery1.Prior;
until(form2.adoquery1.bof=true);
edit_qty:=strtoint(edit7.text);
while form2.adoquery1.eof<>true do
begin
    materialname_material:=form2.adoquery1.fieldbyname('materialname').AsString;

    if materialname_material=edit8.Text then
    begin
        remain_material:=form2.adoquery1.fieldbyname('remain').AsInteger;

        if edit_qty>remain_material then
        begin
            edit_qty:=(edit_qty) - (remain_material);
            form2.adoquery1.Edit;
            form2.adoquery1.UpdateRecord;
            form2.adoquery1.FieldByName('remain').AsInteger:=0;
            form2.adoquery1.Post;
        end
        else
        begin
            remain_material:=(remain_material) - (edit_qty);
            edit_qty:=0;
            form2.adoquery1.Edit;
            form2.adoquery1.UpdateRecord;
            form2.adoquery1.FieldByName('remain').AsInteger:=remain_material;
            form2.adoquery1.Post;
        end ;
    end;
    form2.adoquery1.Next;
end;
form2.adoquery2.Active:=false;
form2.adoquery2.Active:=true;

```

```

adoquery6.FieldByName('used').AsInteger:=strtoint(edit7.Text);
adoquery6.Post;
degisken:=0;
dblookupcombobox2.Enabled:=false;
edit7.Text:="";

end;

```

```

procedure TForm1.Button23Click(Sender: TObject);

```

```

var

```

```

a:word;

```

```

materialname_material:string;

```

```

remain_material:integer;

```

```

used_empmat:integer;

```

```

materialname_empmat:string;

```

```

begin

```

```

a:=application.messagebox('are you sure?','warning',36);

```

```

if (a=idyes)then

```

```

begin

```

```

repeat

```

```

form2.ADOQuery1.Next;

```

```

until(form2.ADOQuery1.Eof=true);

```

```

materialname_empmat:=adoquery6.fieldbyname('materialname').AsString;

```

```

used_empmat:=adoquery6.fieldbyname('used').AsInteger;

```

```

while form2.ADOQuery1.Bof<>true do

```

```

begin

```

```

materialname_material:=form2.ADOQuery1.fieldbyname('materialname').AsString;

```



```

if materialname_material= materialname_empmat then
begin
remain_material:=form2.ADOQuery1.fieldbyname('remain').AsInteger;
remain_material:=remain_material + used_empmat;

form2.ADOQuery1.Edit;
form2.ADOQuery1.UpdateRecord;
form2.ADOQuery1.FieldByName('remain').AsInteger:=remain_material;
form2.ADOQuery1.Post;

form2.ADOQuery2.Active:=false;
form2.ADOQuery2.Active:=true;

adoquery6.Delete;
adoquery6.prior;
degisken:=0;

exit;
end;
form2.ADOQuery1.Prior;
end;
end;

procedure TForm1.MaskEdit4Change(Sender: TObject);
begin
dbedit1.Text:=maskedit4.Text;
end;

```

```
procedure TForm1.MaskEdit1Change(Sender: TObject);  
begin  
  dbedit5.Text:=maskedit1.Text;  
end;
```

```
procedure TForm1.MaskEdit2Change(Sender: TObject);  
begin  
  dbedit7.Text:=maskedit2.Text;  
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);  
begin
```

```
  adoquery1.Append;  
  dbedit5.Text:=maskedit1.Text;  
  dbedit2.SetFocus;  
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);  
begin  
  adoquery1.Post;  
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);  
begin  
  adoquery1.Edit;  
  adoquery1.UpdateRecord;  
  dbedit5.Text:=maskedit1.Text;  
  
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);  
begin  
  adoquery1.Cancel;
```

end;

procedure TForm1.Button5Click(Sender: TObject);

var

a:word;

begin

a:=application.messagebox('are you sure?','warning',36);

if (a=idyes)then

begin

adoquery1.Delete;

adoquery1.prior;

end;

end;

procedure TForm1.Button7Click(Sender: TObject);

begin

degisken:=3;

if adoquery6.FieldName('employer_surname').AsString="" then

begin

adoquery6.Append;

if degisken = 3 then dblookupcombobox2.Enabled:=true;

end;

if (adoquery6.FieldName('employer_surname').AsString<>") and

(adoquery6.FieldName('used').AsInteger=0) then

begin

showmessage('you have to save this record first');

end;

if (adoquery6.FieldName('employer_surname').AsString<>") and

(adoquery6.FieldName('used').AsInteger<>0) then

begin

adoquery6.Append;


```

if degisken = 3 then dblookupcombobox2.Enabled:=true;
end;
end;
procedure TForm1.Button24Click(Sender: TObject);
begin
if adoquery6.FieldByName('employer_name').AsString="" then begin exit;end;
degisken:=3;
update1:=10;
adoquery6.Edit;
adoquery6.UpdateRecord;
end;

```

```

procedure TForm1.Button25Click(Sender: TObject);
var
a:word;
begin
a:=application.messagebox('are you sure?','warning',36);
if (a=idyes)then
begin
halt;
end;
end;

```

```

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Caption:='Main Menu';
pagecontrol1.ActivePage:=tabsheet1;
end;

```

```

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
if Key = #13 then begin

```

```

Key := #0;
if (Sender is TDBGrid) then
  TDBGrid(Sender).Perform(WM_KeyDown, VK_Tab, 0)
else
  Perform(Wm_NextDlgCtl, 0, 0);
end;
end;
end.

```

```

unit Unit2;

```

```

interface

```

```

uses

```

```

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

```

```

type

```

```

  TForm2 = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet5: TTabSheet;
    ADOQuery1: TADOQuery;

```

DataSource1: TDataSource;
ADOQuery2: TADOQuery;
DataSource2: TDataSource;
ADOQuery3: TADOQuery;
DataSource3: TDataSource;
Panel1: TPanel;
DBGrid3: TDBGrid;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
MaskEdit1: TMaskEdit;
MaskEdit2: TMaskEdit;
DBLookupComboBox1: TDBLookupComboBox;
Panel2: TPanel;
Button2: TButton;
Button3: TButton;
Button4: TButton;
Button5: TButton;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Panel3: TPanel;
DataSource4: TDataSource;
DBGrid4: TDBGrid;
DataSource5: TDataSource;
DBGrid5: TDBGrid;
ADOTable1: TADOTable;
ADOTable2: TADOTable;

Edit5: TEdit;
ADOQuery4: TADOQuery;
DataSource6: TDataSource;
Panel4: TPanel;
DBGrid6: TDBGrid;
Panel5: TPanel;
ADOQuery5: TADOQuery;
DataSource7: TDataSource;
ADOQuery6: TADOQuery;
DataSource8: TDataSource;
DBGrid7: TDBGrid;
Edit6: TEdit;
Edit7: TEdit;
Edit8: TEdit;
Edit9: TEdit;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Button6: TButton;
Button7: TButton;
Button8: TButton;
Button9: TButton;
Button10: TButton;
DBGrid9: TDBGrid;
Edit10: TEdit;
Label12: TLabel;
MaskEdit3: TMaskEdit;
Edit11: TEdit;
Edit12: TEdit;
Edit13: TEdit;
Edit14: TEdit;
Edit15: TEdit;

Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
Label20: TLabel;
DataSource9: TDataSource;
DataSource10: TDataSource;
Panel6: TPanel;
Panel7: TPanel;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
MaskEdit4: TMaskEdit;
MaskEdit5: TMaskEdit;
Button11: TButton;
Button12: TButton;
Button13: TButton;
Button14: TButton;
Button15: TButton;
Panel8: TPanel;
Edit16: TEdit;
Edit17: TEdit;
Edit18: TEdit;
Edit4: TEdit;
Edit19: TEdit;
Label21: TLabel;
Label22: TLabel;
Label23: TLabel;
Label24: TLabel;
Label25: TLabel;
DBEdit1: TDBEdit;

DBGrid11: TDBGrid;
DataSource11: TDataSource;
Edit20: TEdit;
Label26: TLabel;
DBGrid10: TDBGrid;
DBGrid8: TDBGrid;
ADOQuery7: TADOQuery;
ADOQuery8: TADOQuery;
ADOQuery9: TADOQuery;
ADOQuery10: TADOQuery;
DataSource12: TDataSource;
Panel9: TPanel;
DBGrid12: TDBGrid;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
Panel10: TPanel;
Button16: TButton;
Button17: TButton;
Button18: TButton;
Button19: TButton;
Button20: TButton;
Panel11: TPanel;
DBGrid1: TDBGrid;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
MaskEdit6: TMaskEdit;
Panel12: TPanel;
Button21: TButton;
Button22: TButton;
Button23: TButton;
Button24: TButton;
Button25: TButton;


```

DBEdit14: TDBEdit;
DBGrid2: TDBGrid;
Edit21: TEdit;
Edit22: TEdit;
Edit23: TEdit;
Label27: TLabel;
Label28: TLabel;
Label29: TLabel;
Label30: TLabel;
DBEdit15: TDBEdit;
DBEdit16: TDBEdit;
DBEdit17: TDBEdit;
Panel13: TPanel;
Button1: TButton;
Button26: TButton;
Label31: TLabel;
Label32: TLabel;
Label33: TLabel;
Label34: TLabel;
procedure Button1Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure Edit3Change(Sender: TObject);
procedure Edit5Change(Sender: TObject);
procedure Edit6Change(Sender: TObject);
procedure Edit7Change(Sender: TObject);
procedure Edit8Change(Sender: TObject);
procedure Edit9Change(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure Button9Click(Sender: TObject);
procedure Button10Click(Sender: TObject);
procedure DBGrid7CellClick(Column: TColumn);

```

```

procedure DBGrid9CellClick(Column: TColumn);
procedure TabSheet2Show(Sender: TObject);
procedure MaskEdit3DbClick(Sender: TObject);
procedure Edit11Change(Sender: TObject);
procedure Edit12Change(Sender: TObject);
procedure Edit13Change(Sender: TObject);
procedure Edit14Change(Sender: TObject);
procedure Edit15Change(Sender: TObject);
procedure Label20Click(Sender: TObject);
procedure Button11Click(Sender: TObject);
procedure Button12Click(Sender: TObject);
procedure Button13Click(Sender: TObject);
procedure Button14Click(Sender: TObject);
procedure Button15Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Edit4Change(Sender: TObject);
procedure Edit19Change(Sender: TObject);
procedure Edit16Change(Sender: TObject);
procedure Edit17Change(Sender: TObject);
procedure Edit18Change(Sender: TObject);
procedure MaskEdit5Change(Sender: TObject);
procedure Edit20Change(Sender: TObject);
procedure Button16Click(Sender: TObject);
procedure Button17Click(Sender: TObject);
procedure Button18Click(Sender: TObject);
procedure Button19Click(Sender: TObject);
procedure Button20Click(Sender: TObject);
procedure MaskEdit6Change(Sender: TObject);
procedure Button21Click(Sender: TObject);
procedure DBEdit13Change(Sender: TObject);
procedure Button22Click(Sender: TObject);
procedure Button23Click(Sender: TObject);
procedure Button24Click(Sender: TObject);

```

```

procedure Button25Click(Sender: TObject);
procedure Edit21Change(Sender: TObject);
procedure Edit22Change(Sender: TObject);
procedure Edit23Change(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure MaskEdit2Change(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure TabSheet1Show(Sender: TObject);
procedure MaskEdit4Change(Sender: TObject);
procedure MaskEdit1Change(Sender: TObject);
procedure MaskEdit1Click(Sender: TObject);
procedure Button26Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);

```

```

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

```

```

var
Form2: TForm2;
balance:integer;
implementation

```

```

uses Unit1;

```

```

{$R *.dfm}

```

```

procedure TForm2.Button1Click(Sender: TObject);
begin
form1.show;
form2.Hide;

```


end;

procedure TForm2.Edit1Change(Sender: TObject);

begin

adoquery3.close;

adoquery3.SQL.Clear;

adoquery3.SQL.Add('select * from job where jobno like'+#39+(edit1.Text)+'%'+#39);

adoquery3.Open;

end;

procedure TForm2.Edit2Change(Sender: TObject);

begin

adoquery3.close;

adoquery3.SQL.Clear;

adoquery3.SQL.Add('select * from job where customer_name
like'+#39+(edit2.Text)+'%'+#39);

adoquery3.Open;

end;

procedure TForm2.Edit3Change(Sender: TObject);

begin

adoquery3.close;

adoquery3.SQL.Clear;

adoquery3.SQL.Add('select * from job where customer_surname
like'+#39+(edit3.Text)+'%'+#39);

adoquery3.Open;

end;

procedure TForm2.Edit5Change(Sender: TObject);

begin

adoquery3.close;

adoquery3.SQL.Clear;

```

adoquery3.SQL.Add('select * from job where cleaningdate
like'+#39+(edit5.Text)+'%'+#39);
adoquery3.Open;
end;

```

```

procedure TForm2.Edit6Change(Sender: TObject);
begin
    adoquery6.close;
    adoquery6.SQL.Clear;
    adoquery6.SQL.Add('select * from job where jobno like'+#39+(edit6.Text)+'%'+#39);
    adoquery6.Open;
    if degisken<>5 then
    begin
        adoquery4.close;
        adoquery4.SQL.Clear;
        adoquery4.SQL.Add('select * from BILL where jobno
like'+#39+(edit6.Text)+'%'+#39);
        adoquery4.Open;
    end;
end;

```

```

procedure TForm2.Edit7Change(Sender: TObject);
begin
    adoquery6.close;
    adoquery6.SQL.Clear;
    adoquery6.SQL.Add('select * from job where customer_name
like'+#39+(edit7.Text)+'%'+#39);
    adoquery6.Open;
end;

```

```

procedure TForm2.Edit8Change(Sender: TObject);
begin
    adoquery6.close;
    adoquery6.SQL.Clear;

```

```

adoquery6.SQL.Add('select * from job where customer_surname
like'+#39+(edit8.Text)+'%'+#39);
adoquery6.Open;
end;

```

```

procedure TForm2.Edit9Change(Sender: TObject);
begin
    adoquery6.close;
    adoquery6.SQL.Clear;
    adoquery6.SQL.Add('select * from job where cleaningdate
like'+#39+(edit9.Text)+'%'+#39);
    adoquery6.Open;
end;

```

```

procedure TForm2.Button6Click(Sender: TObject);

```

```

begin
    degisken:=4;
    edit10.Text:="";
    repeat
        adoquery4.next;
    until (adoquery4.eof = true);

```

```

    balance:=adoquery4.FieldByName('balance').AsInteger;

```

```

    adoquery4.Append;
end;

```

```

procedure TForm2.Button7Click(Sender: TObject);

```

```

begin
    if degisken=5 then
        begin
            adoquery4.Post;
            adoquery4.Refresh;

```



```

edit10.Enabled:=true;
degisken:=0;
exit;
end;

if (adoquery4.FieldName('receipt').Asinteger=0) and
(adoquery4.FieldName('balance').Asinteger=0)and (edit10.Text='') then
begin
adoquery4.FieldName('balance').Asinteger:=
adoquery4.FieldName('invoice').Asinteger;
adoquery4.Post;
adoquery4.Refresh;
edit10.Enabled:=true;
degisken:=0;
exit;
end;

if balance=0 then
begin
showmessage('this person has not dept');
exit;
end;

if strtoint(edit10.Text)>balance then
begin
showmessage('receipt can not bigger than balance');
exit;
end;

balance:=balance-strtoint(edit10.Text);
adoquery4.FieldName('balance').Asinteger:=balance;
adoquery4.FieldName('receipt').Asinteger:=strtoint(edit10.text);
adoquery4.Post;

```

```

adoquery4.Refresh;
edit10.Enabled:=true;
degisken:=0;

end;

procedure TForm2.Button8Click(Sender: TObject);
begin
adoquery4.Cancel;
edit10.Enabled:=true;
degisken:=0;
end;

procedure TForm2.Button9Click(Sender: TObject);
begin
degisken:=5;
adoquery4.Edit;
adoquery4.UpdateRecord;
end;

procedure TForm2.Button10Click(Sender: TObject);
var
a:word;
begin
a:=application.messagebox('are you sure?','warning',36);
if (a=idyes)then
begin
adoquery4.Delete;
adoquery4.prior;
end;
end;

procedure TForm2.DBGrid7CellClick(Column: TColumn);
var

```

```
aktar1:string;  
aktar2:string;  
aktar3:integer;  
aktar4:string;
```

```
begin
```

```
if (degisken=4) or (degisken=5)then
```

```
begin
```

```
    aktar3:= adoquery6.FieldName('jobno').AsInteger;  
    adoquery4.FieldName('jobno').Asinteger:=aktar3;  
    aktar1:= adoquery6.FieldName('customer_name').AsString;  
    adoquery4.FieldName('customer_name').AsString:=aktar1;  
    aktar2:=adoquery6.FieldName('customer_surname').AsString;  
    adoquery4.FieldName('customer_surname').AsString:=aktar2;  
    aktar4:= adoquery6.FieldName('cleaningkind').Asstring;  
    adoquery4.FieldName('cleaningkind').Asstring:=aktar4;  
end;  
end;
```

```
procedure TForm2.DBGrid9CellClick(Column: TColumn);
```

```
var
```

```
aktar:integer;
```

```
begin
```

```
if (degisken=4) or (degisken=5)then
```

```
begin
```

```
    aktar:= adoquery5.FieldName('cleaningcost').AsInteger;  
    adoquery4.FieldName('invoice').Asinteger:=aktar;  
    edit10.Enabled:=false;  
end;  
end;
```

```
procedure TForm2.TabSheet2Show(Sender: TObject);
```


begin

maskedit3.Text:=datetostr(date);

adoquery6.Active:=false;

adoquery6.Active:=true;

adoquery5.Active:=false;

adoquery5.Active:=true;

adoquery4.Active:=false;

adoquery4.Active:=true;

end;

procedure TForm2.MaskEdit3DbClick(Sender: TObject);

var

aktar4:string;

begin

if (degisken=4)or (degisken=5) then

begin

aktar4:= maskedit3.Text;

adoquery4.FieldByName('billdate').AsString:=aktar4;

end;

end;

procedure TForm2.Edit11Change(Sender: TObject);

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select * from BILL where billno

like'+#39+(edit11.Text)+'%'+#39);

adoquery4.Open;

end;

procedure TForm2.Edit12Change(Sender: TObject);

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select * from BILL where billdate
like'+#39+(edit12.Text)+'%'+#39);

adoquery4.Open;

end;

procedure TForm2.Edit13Change(Sender: TObject);

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select * from BILL where jobno
like'+#39+(edit13.Text)+'%'+#39);

adoquery4.Open;

end;

procedure TForm2.Edit14Change(Sender: TObject);

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select * from BILL where customer_name
like'+#39+(edit14.Text)+'%'+#39);

adoquery4.Open;

end;

procedure TForm2.Edit15Change(Sender: TObject);

begin

adoquery4.close;

adoquery4.SQL.Clear;

adoquery4.SQL.Add('select * from BILL where customer_surname
like'+#39+(edit15.Text)+'%'+#39);

adoquery4.Open;

end;

```
procedure TForm2.Label20Click(Sender: TObject);
```

```
begin
```

```
    maskedit3.Text:=datetostr(date);
```

```
end;
```

```
procedure TForm2.Button11Click(Sender: TObject);
```

```
begin
```

```
    adoquery7.Append;
```

```
end;
```

```
procedure TForm2.Button12Click(Sender: TObject);
```

```
begin
```

```
    adoquery7.Post;
```

```
    adoquery7.Active:=false;
```

```
    adoquery7.Active:=true;
```

```
end;
```

```
procedure TForm2.Button13Click(Sender: TObject);
```

```
begin
```

```
    degisken:=6;
```

```
    adoquery7.Edit;
```

```
    adoquery7.UpdateRecord;
```

```
end;
```

```
procedure TForm2.Button14Click(Sender: TObject);
```

```
begin
```

```
    adoquery7.Cancel;
```

```
end;
```

```
procedure TForm2.Button15Click(Sender: TObject);
```

```
var
```

```
    a:word;
```

```
begin
```



```

a:=application.messagebox('are you sure?','warning',36);
if (a=idyes)then
begin
adoquery7.Delete;
adoquery7.prior;
end;

end;

```

```

procedure TForm2.Button2Click(Sender: TObject);
begin
adoquery3.Edit;
adoquery3.UpdateRecord;
dbedit16.Text:=maskedit1.Text;
dbedit17.Text:= maskedit2.Text;
dbedit15.Text:=maskedit4.Text;
dbedit2.SetFocus;
end;

```

```

procedure TForm2.Button3Click(Sender: TObject);
begin

adoquery3.Post;
end;

```

```

procedure TForm2.Edit4Change(Sender: TObject);
begin
adoquery3.close;
adoquery3.SQL.Clear;
adoquery3.SQL.Add('select * from job where phone like'+#39+(edit4.Text)+'%'+#39);
adoquery3.Open;
end;

```

```

procedure TForm2.Edit19Change(Sender: TObject);

```

```

begin
    adoquery7.close;
adoquery7.SQL.Clear;
adoquery7.SQL.Add('select * from employer where phone
like'+#39+(edit19.Text)+'%'+#39);
adoquery7.Open;

end;

procedure TForm2.Edit16Change(Sender: TObject);
begin
    adoquery7.close;
adoquery7.SQL.Clear;
adoquery7.SQL.Add('select * from employer where employerno
like'+#39+(edit16.Text)+'%'+#39);
adoquery7.Open;

    adoquery9.close;
adoquery9.SQL.Clear;
adoquery9.SQL.Add('select * from empmat where employerno
like'+#39+(edit16.Text)+'%'+#39);
adoquery9.Open;
    adoquery9.Refresh;

end;

procedure TForm2.Edit17Change(Sender: TObject);
begin
    adoquery7.close;
adoquery7.SQL.Clear;

```

```
adoquery7.SQL.Add('select * from employer where employer_name  
like'+#39+(edit17.Text)+'%'+#39);  
adoquery7.Open;
```

```
adoquery9.close;  
adoquery9.SQL.Clear;  
adoquery9.SQL.Add('select * from empmat where employer_name  
like'+#39+(edit17.Text)+'%'+#39);  
adoquery9.Open;  
adoquery9.Refresh;
```

```
end;
```

```
procedure TForm2.Edit18Change(Sender: TObject);  
begin  
adoquery7.close;  
adoquery7.SQL.Clear;  
adoquery7.SQL.Add('select * from employer where employer_surname  
like'+#39+(edit18.Text)+'%'+#39);  
adoquery7.Open;
```

```
adoquery9.close;  
adoquery9.SQL.Clear;  
adoquery9.SQL.Add('select * from empmat where employer_surname  
like'+#39+(edit18.Text)+'%'+#39);  
adoquery9.Open;  
adoquery9.Refresh;
```


end;

procedure TForm2.MaskEdit5Change(Sender: TObject);

begin

dbedit1.Text:= maskedit5.Text;

end;

procedure TForm2.Edit20Change(Sender: TObject);

begin

adoquery9.close;

adoquery9.SQL.Clear;

adoquery9.SQL.Add('select * from empmat where jobno
like'+#39+(edit20.Text)+'%'+#39);

adoquery9.Open;

adoquery8.close;

adoquery8.SQL.Clear;

adoquery8.SQL.Add('select * from job where jobno like'+#39+(edit20.Text)+'%'+#39);

adoquery8.Open;

end;

procedure TForm2.Button16Click(Sender: TObject);

begin

adoquery10.Append;

dbedit5.SetFocus;

end;

procedure TForm2.Button17Click(Sender: TObject);

begin

adoquery10.Post;

adoquery10.Active:=false;

adoquery10.Active:=true;

```

end;

procedure TForm2.Button18Click(Sender: TObject);
begin
adoquery10.edit;
adoquery10.UpdateRecord;
end;

procedure TForm2.Button19Click(Sender: TObject);
begin
adoquery10.Cancel;

adoquery10.Active:=false;
adoquery10.Active:=true;
end;

procedure TForm2.Button20Click(Sender: TObject);
var
a:word;
begin
a:=application.messagebox('are you sure?','warning',36);
if (a=idyes)then
begin
adoquery10.Delete;
adoquery10.prior;
end;

end;

procedure TForm2.MaskEdit6Change(Sender: TObject);
begin
dbedit12.Text:=maskedit6.Text;
end;

```

```

procedure TForm2.Button21Click(Sender: TObject);
begin
adoquery1.Append;
end;

procedure TForm2.DBEdit13Change(Sender: TObject);
begin
dbedit14.Text:=dbedit13.Text;
end;

procedure TForm2.Button22Click(Sender: TObject);
begin
adoquery1.Post;

adoquery1.Active:=false;
adoquery1.Active:=true;

adoquery2.Active:=false;
adoquery2.Active:=true;
end;

procedure TForm2.Button23Click(Sender: TObject);
begin
adoquery1.Edit;
adoquery1.UpdateRecord;
end;

procedure TForm2.Button24Click(Sender: TObject);
begin
adoquery1.Cancel;

adoquery1.Active:=false;
adoquery1.Active:=true;

```



```
adoquery2.Active:=false;  
adoquery2.Active:=true;  
end;
```

```
procedure TForm2.Button25Click(Sender: TObject);  
var  
a:word;  
begin  
a:=application.messagebox('are you sure?','warning',36);  
if (a=idyes)then  
begin  
adoquery1.Delete;  
adoquery1.prior;  
adoquery2.Active:=false;  
adoquery2.Active:=true;  
  
end;
```

```
end;
```

```
procedure TForm2.Edit21Change(Sender: TObject);  
begin  
adoquery1.close;  
adoquery1.SQL.Clear;  
adoquery1.SQL.Add('select * from MATERIAL where materialname  
like'+#39+(edit21.Text)+'%'+#39);  
adoquery1.Open;  
end;
```

```
procedure TForm2.Edit22Change(Sender: TObject);  
begin  
adoquery1.close;  
adoquery1.SQL.Clear;
```

```

adoquery1.SQL.Add('select * from MATERIAL where sellercompany
like'+#39+(edit22.Text)+'%'+#39);
adoquery1.Open;
end;

```

```

procedure TForm2.Edit23Change(Sender: TObject);
begin
  adoquery1.close;
  adoquery1.SQL.Clear;
  adoquery1.SQL.Add('select * from MATERIAL where buyingdate
like'+#39+(edit23.Text)+'%'+#39);
  adoquery1.Open;
end;

```

```

procedure TForm2.FormCreate(Sender: TObject);
begin
  pagecontrol1.ActivePage:=tabsheet1;
  form2.Caption:='Search';
end;

```

```

procedure TForm2.MaskEdit2Change(Sender: TObject);
begin
  dbedit17.Text:= maskedit2.Text;
end;

```

```

procedure TForm2.Button4Click(Sender: TObject);
begin
  adoquery3.Cancel;
end;

```

```

procedure TForm2.Button5Click(Sender: TObject);
var
  a:word;
begin

```

```
a:=application.messagebox('are you sure?','warning',36);
if (a=idyes)then
begin
adoquery3.Delete;
adoquery3.prior;
end;

end;
```

```
procedure TForm2.TabSheet1Show(Sender: TObject);
begin
maskedit1.Text:=datetostr(date);
dbedit16.Text:=maskedit1.Text;
```

```
adoquery3.Active:=false;
adoquery3.Active:=true;
```

```
adotable1.Active:=false;
adotable1.Active:=true;
```

```
adotable2.Active:=false;
adotable2.Active:=true;
end;
```

```
procedure TForm2.MaskEdit4Change(Sender: TObject);
begin
dbedit15.Text:=maskedit4.Text
end;
```

```
procedure TForm2.MaskEdit1Change(Sender: TObject);
begin
dbedit16.Text:=maskedit1.Text;
end;
```



```

procedure TForm2.MaskEdit1Click(Sender: TObject);
begin
    maskedit1.Text:=datetostr(date);
    dbedit16.Text:=maskedit1.Text;
end;

procedure TForm2.Button26Click(Sender: TObject);
var
    a:word;
begin
    a:=application.messagebox('are you sure?','warning',36);
    if (a=idyes)then
    begin
        halt;
    end;
end;

end;

procedure TForm2.FormKeyPress(Sender: TObject; var Key: Char);
begin
    if Key = #13 then begin
        Key := #0;
        if (Sender is TDBGrid) then
            TDBGrid(Sender).Perform(WM_KeyDown,VK_Tab,0)
        else
            Perform(Wm_NextDlgCtl,0,0);
    end;
end;

end.

```

```
unit Unit3;
```

```
interface
```

uses

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

type

TForm3 = class(TForm)

Edit1: TEdit;

Edit2: TEdit;

Label1: TLabel;

Label2: TLabel;

Button1: TButton;

Button2: TButton;

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form3: TForm3;

implementation

uses Unit1;

{ \$R *.dfm }

procedure TForm3.Button1Click(Sender: TObject);

begin

halt;

end;


```
procedure TForm3.Button2Click(Sender: TObject);
begin
if (edit1.Text='baris') and (edit2.Text='1234') then
begin
form1.show;
form3.Hide;
end
else
begin
showmessage('invalid username or password');

end;
end;
```

```
procedure TForm3.FormCreate(Sender: TObject);
begin
form3.Caption:='LOG IN';
end;

end.
```