NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CAR SALES AUTOMATION USING VISUAL BASIC

Graduation Project

COM – 400

Student: Malik M. Al-Mustafa (20020927)

Supervisor: Mr. Űmit SOYER

Nicosia - 2008

TABLE OF CONTENTS

ACKNOWLEDGEMENT

First of all I would like to thank Allah for guiding me through my study.

More over I want to pay special regards to my parents who are enduring these all expenses and supporting me in all events. I am nothing without their prayers. They also encouraged me in crises. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting.

Also, I feel proud to pay my special regards to my project supervisor "Mr. Űmit SOYER". He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project. I am really thankful to my teacher.

Also I have to give my best regards to the best teachers I ever had "Mr.Okan DONANGIL", "Assoc. Prof. Dr. Adnan KHASHMAN", "Assist. Prof. Dr. Kaan UYAR", "Dr. Űmit ILHAN"

The best of acknowledge, I want to honor my best friends and my brothers, who have supported me, suffer with me and shered my laughs, Eng. Mohammad M. Al-Mustafa and Eng. Bassem F. Al-Soudi. Thank you guys for every thing you did to me. Good luck guys and wish you a happy and successful life brothers.

ABSTRACT

This project shows the use of Visual Basic and Database applications and how to create a connection between them so the user can easily use them in selling cars field and store the information in the program's database and recall it in a single click of a button. In this project we can add users and assign a user name and password, authorization assigning, and deleting users. In cars field's, we can add cars, delete cars, classify some details for the cars, and edit there details. Then we can sell them to customers. In customers field's, we can also add information about the customers, edit there information, and delete customers. Finally, we can view reports which describe specific information which will be shown on chapter four.

INTRODUCTION

This project is about selling cars by using the visual basic programming and data base.

This project describes how the visual basic works, how I used it in my program, how I create connection with the data base, and how users can use this program.

This project includes 4 chapters:

The first chapter talks about the visual basic programming language with its coding and variable scope (including object variables) and procedure scope, how it works, how to create projects, and how to design projects.

The second chapter talks about the data base, how to create and activate it; also how to insert, edit, search and delete information in it.

Chapter three describes the connection between the data base and the visual basic and how they are working together in the same program.

Chapter four represents my program, diagrams and description to show how to use all the option that I add to this program.

CHAPTER 1

VISUAL BASIC

1.1 OVER VIEW

Visual Basic is a favorite programming environment for many programmers. When Visual Basic first appeared, it created a revolution in Windows programming, and that revolution continues till this day. Windows programming never been so easy, in a few steps we can construct and run programs. Visual Basic introduced unheard-of ease to Windows programming and changed programming from a chore to something very suitable to users.

Well start with an overview about the Visual Basic program. In this chapter, we will create the foundation well rely on later as we take a look at the basics of Visual Basic, including how to create Visual Basic projects and seeing what is in such projects. Well also get an overview of essential Visual Basic concepts like forms, controls, events, properties, methods, and so on. And well examine the structure of a Visual Basic program, taking a look at variables, variable scope, and modules. In other words, were going to lay bare the anatomy of a Visual Basic program here.

Most Visual Basic programmers do not have formal programming training and have to learn a lot of this material the hard way. As programming has matured, programmers have learned more and more about what are called best practices the programming techniques that make robust, easily debugged programs. Well take a look at those practices in this chapter, because they are becoming more and more essential for programmers in commercial environments these days, especially those programmers that work in teams. And well look at those practices from the viewpoint of programmers who program for a living; frequently there`s a gap between the way best practices are taught by academics and how they are actually needed by programmers facing the prospect of writing a 20,000-line program as part of a team of programmers.

1.2 CREATING A PROJECT IN VISUAL BASIC

There are three different editions of Visual Basic:

- The Learning Edition, which is the most basic edition .This edition allows you to write many different types of programs, but lacks a number of tools that the other editions have.

- The Professional Edition, designed for professionals. This edition contains all that the Learning Edition contains and more, such as the capability to write ActiveX controls and documents.

- The Enterprise Edition, which is the most complete Visual Basic edition. This edition is targeted towards professional programmers who may work in a team and includes additional tools such as Visual SourceSafe, a version-control system that coordinates team programming.

1.2.1 DESIGNING THE TIC-TAC-TOE PROGRAM

Using the Command Button tool in the Visual Basic toolbox, add a new command button to the main form in our program now, in the Properties window, change the Name property of this button from Command1 to Command in preparation for setting up a control array, and clear its Caption property so the button appears blank.

Next, add a second button to the form, and set its Name property to Command as well. When you do, Visual Basic opens a dialog box that states: _You already have a control named _Command_. Do you want to set up a control array?_ Click Yes to create a control array, which means we will be able to refer to our controls using an index instead of simply by name.

Add a total of nine buttons to the main form in our program, arranged in a 3×3 grid similar to a standard tic-tac-toe game, give each of the buttons the name Command, and clear their captions. That completes the preliminary design now were ready to write some code.

1.2.2 THE PARTS OF A VISUAL BASIC PROJECT

Projects can become quite advanced in Visual Basic, even containing subprojects of different types. From a programming point of view, however, standard Visual Basic projects usually contain just three types of items: global items, forms, and modules.

- **Forms:**

  Forms are familiar to all Visual Basic programmers, of course they are the templates you base windows on. Besides standard forms, Visual Basic also supports Multiple Document Interface (MDI) forms, as well as a whole number of predefined forms.

- **Modules:**

  Modules are collections of code and data that function something like objects in object-oriented programming (OOP), but without defining OOP characteristics like inheritance, polymorphism, and so on. The point behind modules is to enclose procedures and data in a way that hides them from the rest of the program. We well discuss the importance of doing this later in this chapter when we cover Visual Basic programming techniques and style; breaking a large program into smaller, self-contained modules can be invaluable for creating and maintaining code.

  You can think of well-designed modules conceptually as programming objects; for example, you might have a module that handles screen display that includes a dozen internal (unseen by the rest of the program) procedures and one or two procedures accessible to the rest of the program. In this way, the rest of the program only has to deal with one or two procedures, not a dozen.

- **Global Items:**

  Global items are accessible to all modules and forms in a project, and you declare them with the Public keyword. However, Microsoft recommends that you keep the number of global items to an absolute minimum and, in fact, suggests their use only when you need to communicate between forms. One reason to avoid global variables is their accessibility from anywhere in the program; while you are working with a global variable in one part of a program, another part of the program might be busy changing that variable, giving you unpredictable results.

- **Project Scope:**

  An objects scope indicates how much visibility it has throughout the project in the procedure where it`s declared, throughout a form or module, or global scope (which means it`s accessible everywhere). There are two types of scope in Visual Basic projects:

  - Variable scope (including object variables).
  - Procedure scope.

  We`ll take a look at both of them here as we continue our overview of Visual Basic projects and how the parts of those projects interact.

- **Variable Scope:**

  You declare variables in a number of ways. Most often, you use the Dim statement to declare a variable. If you do not specify the variable type when you use Dim, it creates a variant, which can operate as any variable type. You can specify the variable type using the as keyword like the following:

```
Dim IntegerValue as Integer
```

Besides Dim, you can also use ReDim to redimension space for dynamic arrays, Private to restrict it to a module or form, Public to make it global that is, accessible to all modules or forms or Static to make sure its value does not change between procedure calls.

There are three levels of variable scope in Visual Basic: at the procedure level, at the form or module level, and at the global level schematically.

When you are designing your program, Microsoft suggests you limit your variables to the minimum possible scope in order to make things simpler and to avoid conflicts. Next, we`ll takes a look at the other type of scope.

- **Procedure Scope:**

As with variables, you can restrict the scope of procedures, and you do that with the Private, Public, Friend, and Static keywords. The Private and Public keywords are the main keywords here; using them, you can specify if a subroutine or function is private to the module or form in which it is declared or public (that is, global) to all forms and modules. You use these keywords before the Sub or Function keywords like the following:

```
Private Function Returns7 ()

    Dim Retval

    Retval = 7

    Returns7 = Retval

End Function
```

You can also declare procedures as friend procedures with the Friend keyword.

Friend procedures are usually used in class modules (they are not available in standard modules, although you can declare them in forms) to declare that the procedure is available outside the class, but not outside the current project. This restricts those functions from being called if the current project serves as an OLE automation server, for example.

Besides the earlier declarations, you can also declare procedures as Static, which means that the variables in the procedure do not change between procedure calls, and that can be very useful in cases like this, where we support a counter variable that is incremented each time a function is called:

```
Static Function Counter ()

    Dim CounterValue as Integer

    CounterValue = CounterValue + 1

    Counter = CounterValue

End Sub
```

1.3 CODING IN VISUAL BASIC

The full construction of a commercial program is usually a project that involves many clear and definite steps. There have been whole volumes written on this topic, which are usually only interesting if you are a software project manager (or write computer books and have to know the details so you can write about them). Such books get pretty involved, encompassing ideas like module coupling and cohesion, bottom-up composition, incremental integration, and much more.

On the whole, however, one can break the software design process into steps like these (note that the explanation of each step is very flexible; there is no one-size-fits-all here):

- Requirements analysis Identify the problem for the software to tackle.
- Creating specifications Determine what exactly the software should do.
- Overall design Break the overall project into parts, modules …etc.
- Detailed design the actual data structures, procedures...etc.
- Coding Go from PDL to code.
- Debugging Solve design-time, compilation, and obvious errors.
- Testing Try to break the software.
- Maintenance React to user feedback and keep testing.

Each of these steps may have many subparts, of course. (For example, the maintenance part may take up as much time as the rest of the project takes together).

As the design process continues, a model of what the program does evolves. You use this model to get a conceptual handle on the software (while keeping in mind those models is usually flawed at some level).

Keeping the model in mind, then, many programmers use a program design language to start the actual coding process.

1.3.1 PROGRAM DESIGN LANGUAGE

Everyone seems to think that programmers use flowcharts, but the reality is usually different (flowcharts are nice to show to nonprogrammers, though). One tool that commercial programmers do find useful is program design language (PDL). Although there are formal specifications for PDL, many programmers simply regard this step as writing out what a program does in English as a sort of pseudo-code.

For example, if we want to create a new function named dblSqrt() that returns a numbers square root, we might write its PDL this way in English, where we break what the function does into steps:

```
Function dblSqrt ()

    Check if the input parameter is negative

    If the input parameter is negative, return -1
```

```
        If the input parameter is positive, return its
        square root

    End Function
```

When you actually write the code, the PDL can often become the comments in that code; for example, here`s the completed function:

```
    ' dblSqrt()

    'Purpose: Returns the passed parameter's square root

    'Inputs: dblParameter, the parameter whose square root
    we need

    ' Returns: The input value's square root

    Function dblSqrt(dblParameter As Double) As Double

        'Check if the input parameter is negative

        If dblParameter < 0 Then

            'If the input parameter is negative, return
            -1

            dblSqrt = -1

        Else

            'If the input parameter is positive, return
            its square root

            dblSqrt = Sqr(dblParameter)

        End If

    End Function
```

In this way, developing your program using PDL, where every line of PDL has one (and only one) specific task, can be very useful.

1.4 CODING TO GET THE MOST FROM VISUAL BASIC

In this section, we`ll discuss some best practices coding for Visual Basic. All of these practices come from professional programmers.

Avoid magic numbers when you can. A magic number is a number (excluding 0 or 1) that's hardwired right into your code as follows:

```
Function blnCheckSize(dblParameter As Double) As
Boolean

    If dblParameter > 1024 Then

        blnCheckSize = True

    Else

        blnCheckSize = False

    End If

End Function
```

Here, 1024 is a magic number. It`s better to declare such numbers as constants, especially if you have a number of them. When it`s time to change your code, you just have to change the constant declaration in one place, not try to find all the magic numbers scattered around your code.

Be modular. Putting code and data together into modules hides it from the rest of the program, makes it easier to debug, makes it easier to work with conceptually, and even makes load-time of procedures in the same module quicker. Being modular also called information-hiding (and encapsulation in true OOP) _is the backbone of working with larger programs. Divide and conquer is the idea here.

Program defensively. An example of programming defensively would be to check data passed to you in a procedure before using it. This can save a bug from propagating throughout your program and help pinpoint its source. Make no assumptions.

Visual Basic procedures should have only one purpose, ideally. This is also an aid in larger programs when things start to get complex. Certainly if a procedure has two distinct tasks, consider breaking it up.

Avoid deep nesting of conditionals or loops. Debugging deeply nested conditionals visually is very, very inefficient. If you need to, place some of the inner loops or conditionals in new procedures and call them. Three levels of nesting should be about the maximum.

Use access procedures to protect sensitive data. (This is part of programming defensively.) Access procedures are also called Get/Set procedures, and they are called by the rest of the program when you want to work with sensitive data. If the rest of the program must call a Set() procedure to set that data, you can test to make sure that the new value is acceptable, providing a screen between that data and the rest of the program.

Ideally, variables should always be defined with the smallest scope possible. Global variables can create enormously complex conditions. (In fact, Microsoft recommends that global variables should be used only when there is no other convenient way to share data between forms).

Do not pass global variables to procedures. If you pass global variables to procedures, the procedure you pass that variable to might give it one name (as a passed parameter) and also reference it as a global variable. This can lead to some serious bugs, because now the procedure has two different names for the variable.

Use the operator when linking strings and the + operator when working with numerical values. This is per Microsoft`s recommendations.

When you create a long string, use the underscore line-continuation character to create multiple lines of code. This is so you can read or debug the string easily. For example:

```
Dim Msg As String

Msg = "Well, there is a problem "_

&"with your program. I am not sure " _

&"what the problem is, but there is " _

&"definitely something wrong."
```

Avoid using variants if you can. Although convenient, they waste not only memory but time. You may be surprised by this. Remember, however, that Visual Basic has to convert the data in a variant to the proper type when it learns what is required, and that conversion actually takes a great deal of time.

Indent your code with four spaces per Microsoft is recommendations. Believe it or not, there have been serious studies undertaken here, and 2 to 4 spaces were found to be best. Be consistent.

Finally, watch out for one big Visual Basic pitfall: misspelled variables. Because you do not have to declare a variable in Visual Basic to use it, you might end up surprised when Visual Basic creates a new variable after you have misspelled a variable is name. For example, here is some perfectly legal code modified from our tic-tac-toe project that compiles and runs, but because of a misspelling xNoww for xNow it does not work at all:

```
Private Sub Command_Click(Index As Integer)

    If xNow Then

        Command (Index).Caption = "x"

    Else
```

```
            Command (Index).Caption = "o"

        End If

        xNoww = Not xNow

    End Sub
```

Because Visual Basic treats xNoww as a legal variable, this kind of bug is very hard to find when debugging.

TIP: Because Visual Basic auto-declares variables, it`s usually better to use variable names that say something (like intCurrentIndex) instead of ones that do not (like intDD35A) to avoid declaring a variable through misspelling its name. A better idea is to use Option Explicit to make sure all variables must be explicitly declared.

If you work in teams, use version control. There are several well-known utilities that help programmers work in teams, such as Microsoft`s Visual SourceSafe. This utility, which is designed to work with programming environments like Visual Basic, restricts access to code so that two programmers do not end up modifying independent copies of the same file.

CHAPTER 2

DATABASE

2.1 OVER VIEW

The purpose of a Database system such as Microsoft Access is to change data into information. Many people use those two terms interchangeably, but there is a world of difference between the two if you consider information as being the same as knowledge. Data is a collection of facts. Information is that data organized or presented in such a way as to be useful for decision making.

This shows actual voter registration data for a particular county shown in Access. It includes voters' names, addresses, registration information such as political party, and also the voting records for each person registered. It doesn't, of course, show who voters voted for (that's unavailable as data), but it does show whether and how the voters voted for each election cycle. A voter can vote by mail-in ballot, early voting, or at the polls.

2.2 RELATIONAL DATABASE

A relational Database, simply defined, is a Database that is made up of tables and columns that relate to one another. These relationships are based on a key value that is contained in a column. For example, you could have a table called Orders that contains all the information that is required to process an order, such as the order number, date the item was ordered, and the date the item was shipped. You could also have a table called Customers that contains all the data that pertains to customers, such as a name and address. These two tables could be related to each other.

The relational Database model was developed by E.F. Codd back in the early 1970s. He proposed that a Database should consist of data stored in columns and tables that could be related to each other. This kind of thinking was very different from the hierarchical file system that was used at the time. His thinking truly revolutionized the way Databases are created and used.

A relational Database is very intuitive. It mimics the way people think. People tend to group similar objects together and break down complex objects into simpler ones. Relational Databases are true to this nature. Because they mimic the way you think, they are easy to use and learn. In later days, you will discover how easy a relational Database is to design and learn.

Most modern Databases use a relational model to accomplish their tasks. SQL is no different. It truly conforms to the relational model. This further adds to the ease of use of SQL.

2.3 CHANGING DATA INTO INFORMATION

Now let's take that data and organize it into information. Suppose that in the 1998 congressional race the Democratic candidate lost by 3,216 votes. Using the Count() function built into Access, the Database user notes that Republican voters mailed in 5,423 more ballots than the Democratic voters. This is information. Using it, the Democrats can see that if their candidate had emphasized mail-in balloting more (perhaps by mailing out applications for such ballots), he might have won.

2.4 ACCESS DATABASE

2.4.1 MAINTAINING ACCESS DATABASES

You've probably heard it before it's not if you'll lose data, but when. Computers aren't infallible, power fails, and parts especially disk drives go bad. The only defense is to back up your data. A full discussion of backup devices is beyond the intended scope of this book.

The two fundamental methods are some sort of network backup—either through your LAN or by subscription over the Internet—and backup to removable media such as tape, removable hard disk, or writable CDs. Pick a method and use it.

2.4.1.1 REPAIRING IN PLACE

Access files are somewhat more susceptible to corruption due to power failure than other non-Database programs, such as Word. Although you should be always able

(in all modes) Democrats by 7,987 voters. This is rather irrefutable information that the Democrats fielded a candidate attractive to Republicans, or that the Republican candidate wasn't what Republicans wanted in a congressman, or both.

2.4.2 THE ACCESS USER INTERFACE

Access 2007 has a new user interface designed to be not only easier for the beginner to navigate through, but also to make the life of the Access expert simpler. The concept of the user interface stems from two metaphors, the bar and tab interface common to all Microsoft Office applications, plus the object collection concept from object-oriented programming. The original object metaphor is purely abstract, whereas the translation into Access user interface is concrete.

Like so many concepts in small computers, gaining familiarity with Access' interface is best done by a hands-on approach, so let's get started. Launch Access by choosing it from the Start, Programs menu entry. In some administrative (network) installs, Access will be part of a group under Programs, in which case you'll need to locate where Access is to launch it. For most people, Access will be an entry directly under Start, Programs. Upon launching, Access will offer you several choices.

If this is the first time you've launched Access, you won't have any entries in the list box at the bottom of the dialog box. From top to bottom the three post-launch options are:

- Create a New Database Using a Blank Access Database. This will create a new container (explained in the following text), ready for you to populate with your Database objects.
- Create a New Database Using Access Database Wizards, Pages, and Projects. This will also create a new Database, but by use of a wizard or two to give you a quick start.
- Open an Existing File. This will allow you to choose from a list or browse for an existing Database to open.

In addition, you can click the Cancel button to open Access with no Database loaded. Because setup will register Microsoft Access 2000 with your operating system,

you can also launch Access with a Database loaded by double-clicking on the Database (files with .mdb or .mde extensions) from the Explorer.

For this, a first tour of Access, locate the North wind sample Database supplied with Office. Highlight it, and then click Open or double-click on its entry in the list box. If you need to browse for it, highlight More Files and click OK. That action will open up a standard File Open browsing dialog box.

2.4.2.1 NAVIGATING THE DATABASE VIEW WINDOW

After you've opened the North wind, this is called the Database view and you'll become very familiar with its functions and features as you work with Access. This window contains all the objects in your Database and toolbars for manipulation of these objects, and provides starting points for working with a Database.

The new window on your screen is also an object in the Microsoft hierarchy of objects, a fact that you'll want to remember when you start working with objects in VBA or macro code. For now, though, we'll refer to this window as the Database View window for the object Northwind: Database, which appears in the title bar. This window is divided into three main parts:

- The toolbar with actions and view selections.
- The left pane, which lists the types, or classes, of available objects within all Access Databases, such as tables and forms.
- The right pane, which shows a listing of the individual objects within the selected class on the left pane.

One new feature here is the Group class option on the left pane, which you'll learn more about in the "Groups" section, later in this lesson. As you can see by clicking through the various objects in the left pane, the North wind Database has several objects as a part of its application.

The new object, the one that says North wind: Database in its title bar, is the Database view or Database container, as it's sometimes called because it contains various objects that make up a Database system. It displays all the items or objects

within your project collected by classification. The series of buttons on the left side of the container allows you to choose from different types of objects, such as tables or reports. Click on the Forms entry (or any entry other than the one currently selected) and the right pane will reflect all the Database objects so as to show the one class selected within the Database.

You'll see a set of icons telling you what actions on the toolbar you can perform on the objects listed in the Database view panes. The first eight entries, from left to right, allow you to do the following:

- **Open:** Launch an object in its native mode, such as for data entry. Access uses the term view for different object modes.
- **Design:** Launch an object in such a way as to allow you to edit its structure rather than its data.
- **New:** Create a new object of the type highlighted within the Object list.
- **Delete:** Delete the highlighted object. This functions only for objects created by a user or developer, not for the objects that appear in a new Database.
- **Large Icons:** Display the Database objects in large icon view, analogous to the same view in Windows Explorer or My Computer.
- **Small Icons:** Display the Database objects in small icon view, analogous to the same view in Windows Explorer or My Computer.
- **List:** Display the Database objects in list view, analogous to the same view in Windows Explorer or My Computer.
- **Details:** Display the Database objects in Details view, analogous to the same view in Windows Explorer or My Computer.

Right-clicking is alive and well in Microsoft Access 2000. Right-click on any true object (as opposed to an action within the Database view) and you'll see a context (or shortcut) menu containing all the actions within the Database view as well as a few more. True to its name, the context menu for each class of objects will vary.

2.5 SQL DATABASE

SQL is a full-featured relational Database management system. It is very stable and has proven itself over time.SQL has been in production for over 10 years.

SQL is a multithreaded server. *Multithreaded* means that every time someone establishes a connection with the server, the server program creates a thread or process to handle that client's requests. This makes for an extremely fast server. In effect, every client who connects to a SQL server gets his or her own thread.

SQL is also fully ANSI SQL92-compliant. It adheres to all the standards set forth by the American National Standards Institute. The developers at TcX take these standards seriously and have carefully adhered to them.

Note that ANSI SQL92 is a set of standards for the Structured Query Language that was agreed on in 1992 by the American National Standards Institute.

Another valuable feature of SQL is its online help system. All commands for SQL are given at a command prompt. To see which arguments the commands take or what the utility or command does, all you have to do is type the command and include the -help or -? Switch. This will display a slew of information about the command.

Yet another feature of SQL is its portability it has been ported to almost every platform. This means that you don't have to change your main platform to take advantage of SQL. And if you do want to switch, there is probably a SQL port for your new platform.

SQL also has many different application programming interfaces (APIs). They include APIs for Perl, TCL, Python, C/C++, Java (JDBC), and ODBC. So no matter what your company's expertise is, SQL has a way for you to access it.

SQL is also very cheap. For an unlicensed, full version of SQL, the cost is nothing. To license your copy will currently cost you $200. This is an incredible deal, considering what you are getting for your money. Database systems that provide half the features that SQL has can cost tens of thousands of dollars. SQL can do what they do better and for less.

CHAPTER 3

VISUAL BASIC AND DATABASE

3.1 OVERVIEW

When Visual Basic first started working with Databases, it used the Microsoft Jet Database engine, which is what Microsoft Access uses. Using the Jet engine represented a considerable advance for Visual Basic, because now you could work with all kinds of data formats in the fields of a Database: text, numbers, integers, longs, singles, doubles, dates, binary values, OLE objects, currency values, Boolean values, and even memo objects (up to 1.2GB of text). The Jet engine also supports SQL, which Database programmers found attractive.

To support the Jet Database engine, Microsoft added the data control to Visual Basic, and you can use that control to open Jet Database (.mdb) files. Microsoft also added a set of Data Access Objects (DAO) to Visual Basic:

- **DB Engine:** The Jet Database engine.
- **Workspace:** An area can hold one or more Databases.
- **Database:** A collection of tables.
- **Table Def:** The definition of a table.
- **Query Def:** The definition of a query.
- **Record set:** The set of records that make up the result of a query.
- **Field:** A column in a table.
- **Index:** An ordered list of records.
- **Relation:** Stored information about the specific relationship between tables.

3.2 OPENING DATABASE

To open an existing DAO Database, you use the DAO OpenDatabase method, passing it the name of the Database to open, and these arguments:

```
Set Database = workspace.OpenDatabase (dbname, [options [,
read-only _[, connect]]])
```

Here are the arguments for OpenDatabase:

- Dbname: The name of an existing Database file, or the data source name (DSN) of an ODBC data source.
- Options: Setting options to True opens the DAO Database in exclusive mode; setting it to False (the default) opens the Database in shared mode.
- Read-Only: True if you want to open the Database with read-only access, or False (the default) if you want to open the Database with read/write access.
- Connect-Optional: A Variant (String subtype) that specifies various connection information, including passwords.

Let's see an example to make this clearer. In our DAO code example, the daocode project (see the first topic in this chapter), the user can click the Open Database menu item to open a Database. In the program, we get the name of the Database the user wants to open with a Common Dialog control, and open the Database like this:

```
Private Sub OpenDatabase_Click()
    CommonDialog1.ShowO
  If CommonDialog1.FileName <> "" Then
    Set db = _
    DBEngine.Workspaces(0).OpenDatabase(CommonDialog1
    .FileName)
```

Next, if you know the name of the table you want to open in the Database, you can open that table by name immediately with the OpenRecordset method. However, because we let the user set the name of tables in the Databases we create in the daocode project, we don't know the names of the tables in the Database we've opened. Instead, we'll open the first user-defined table in this Database. When you open a DAO Database, there are a number of system tables already in it, so to open the first user-defined table; we find the index of that table in the TableDefs collection by first skipping the system tables (which have the dbSystemObject flag set in their Attributes properties):

```
Private Sub OpenDatabase_Click()
    Dim tablelindex As Intege
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
```

```
Set db = _
DBEngine.Workspaces(0).OpenDatabase(CommonDi
alog1.FileName)
table1index = 0
While  (db.TableDefs(table1index).Attributes
And dbSystemObject)
     table1index = table1index + 1
Wend
```

We'll open the first table after the system tables. We open a new record set for that table with the OpenRecordset method and fill the text boxes Text1 and Text2 in the program's main window with the fields of the first record in that table (note that in this example program, we are assuming the table we're opening has at least one record):

```
Private Sub OpenDatabase_Click()

     Dim table1index As Integer

     CommonDialog1.ShowOpen

     If CommonDialog1.FileName <> "" Then

          Set db = _

          DBEngine.Workspaces(0).OpenDatabase(Com
          monDialog1.FileName)

          table1index = 0

          While(db.TableDefs(table1index).Attribu
          tes And dbSystemObject)

               table1index = table1index + 1

          Wend

          Set dbrecordset = db.OpenRecordset_
```

```
            (db.TableDefs(tablelindex).Name,
            dbOpenTable)

            Set td = db.TableDefs(tablelindex)

            Text1.Text = dbrecordset.fields(0)

            Text2.Text = dbrecordset.fields(1)

        End If

    End Sub
```

3.3 ADDING A RECORD TO A RECORD SET

To add a new record to a DAO record set, you use the AddNew method (this method takes no parameters). After you've updated the fields of the current record, you save that record to the Database with the Update method.

Here's an example using AddNew. When the user clicks the Add button in our DAO code example, the daocode project (see the first topic in this chapter), we execute the AddNew method on the program's record set and clear the two data field text boxes:

```
Private Sub Command1_Click()

    dbrecordset.AddNew

    Text1.Text = ""

    Text2.Text = ""

End Sub
```

Now users can enter data for the new record's fields and click the program's Update button. When they click the Update Database button, the new data is written to the Database.

3.4 EDITING A RECORD IN A RECORD SET

Besides adding new records to the record set, users might want to edit the existing records. To do that, you use the Edit method like this in our DAO code example, the daocode project (see the first topic in this chapter):

```
Private Sub Command2_Click()

    dbrecordset.Edit

End Sub
```

After users edit the data in the record's fields (by entering new data in the text fields in the daocode project's main window), they must update the Database with the new data, and they do that in the daocode project by clicking the Update Database button. That button executes the Update method, as we'll see in the next topic.

3.5 UPDATING A RECORD IN A RECORD SET

When the user changes the data in a record or adds a new record, we must update the Database to record that change, and you use the record set Update method to do that:

```
recordset.Update ([type [, force]]
```

Here are the arguments in this function:

- **Type-Constant:** indicating the type of update, as specified in Settings (ODBCDirect workspaces only).
- **Force-Boolean:** value indicating whether or not to force the changes into the Database, regardless of whether the data has been changed by another user (ODBCDirect workspaces only).

When the user clicks the Update button in our DAO code example, the daocodev project we will update the Database with the new data for the current record, then we get the new data for the current record from the text boxes Text1 and Text2, where the

user has entered that data, and load the data into the record set's fields using the field's collection:

```
Private Sub Command3_Click ()

    dbrecordset.fields(0) = Text1.Text

    dbrecordset.fields(1) = Text2.Text

    ...

End Sub
```

After loading the data into the current record's fields, we save that record to the Database using the Update method:

```
Private Sub Command3_Click()

    dbrecordset.fields(0) = Text1.Text

    dbrecordset.fields(1) = Text2.Text

    dbrecordset.Update

End Sub
```

3.6 MOVING TO THE FIRST RECORD IN A RECORD SET

To make the first record in a record set the current record, you use the MoveFirst method. For example, here's how we move to the first record when the user clicks the appropriate button in our DAO:

```
    Private Sub Command4_Click()

    dbrecordset.MoveFirst

    ...

End Sub
```

After moving to the first record, we display that record's fields in the two text boxes in the program, Text1 and Text2:

```
Private Sub Command4_Click()

    dbrecordset.MoveFirst

    Text1.Text = dbrecordset.fields(0)

    Text2.Text = dbrecordset.fields(1)

End Sub
```

3.7 MOVING TO THE LAST RECORD IN A RECORD SET

To make the last record in a record set the current record, you use the MoveLast method. For example, here's how we move to the last record when the user clicks the appropriate button in our DAO code example, the daocode project (see the first topic in this chapter):

```
Private Sub Command7_Click()

    dbrecordset.MoveLast

    ...

End Sub
```

After moving to the last record, we display that record's fields in the two text boxes in the program, Text1 and Text2:

```
Private Sub Command7_Click()

    dbrecordset.MoveLast

    Text1.Text = dbrecordset.fields(0)

    Text2.Text = dbrecordset.fields(1)
```

```
End Sub
```

3.8 DELETING A RECORD IN A RECORD SET

To delete a record in a DAO record set, you use the Delete method, and then you update the record set.

For example, when the user clicks the Delete button in our DAO code example, the daocode project (see the first topic in this chapter), we clear the two text boxes, Text1 and Text2 that display the data for the current record and delete that record:

```
Private Sub Command8_Click()

    Text1.Text = ""

    Text2.Text = ""

    dbrecordset.Delete

End Sub
```

3.9 SEARCHING A RECORD SET

You can search a record set with an index; we just set its Index property to the index we want to search and then set its Seek property to the string we want to search for. Let's see an example. When the user selects the Search menu item in our DAO code example, the daocode project (see the first topic in this chapter), we install the index based on the first field in the record set and show the dialog box named Search:

```
Private Sub Search_Click()

    Set dbindex = td.Indexes(0)

    dbrecordset.Index = dbindex.Name

    SearchForm.Show

End Sub
```

After the user dismisses the Search... dialog box, we retrieve the text to search for from that dialog box's text box and place that text in the record set's Seek property, along with the command "=", which indicates we want to find exact matches to the search text:

```
Sub SearchTable()

    dbrecordset.Seek "=", SearchForm.Text1.Text

    ...
```

Besides =, you can also search using <, <=, >=, and >. When the search is complete, we display the found record in the daocode project's main text boxes, Text1 and Text2:

```
Sub SearchTable()

    dbrecordset.Seek "=", SearchForm.Text1.Text

    Text1.Text = dbrecordset.fields(0)

    Text2.Text = dbrecordset.fields(1)

End Sub
```

CHAPTER 4

X-SELLING CARS

4.1 OVERVIEW

This project is software which provides us some options such as adding, updating, and selling cars.

4.2 PROJECT DESCRIPTION

The project flows as shown below:

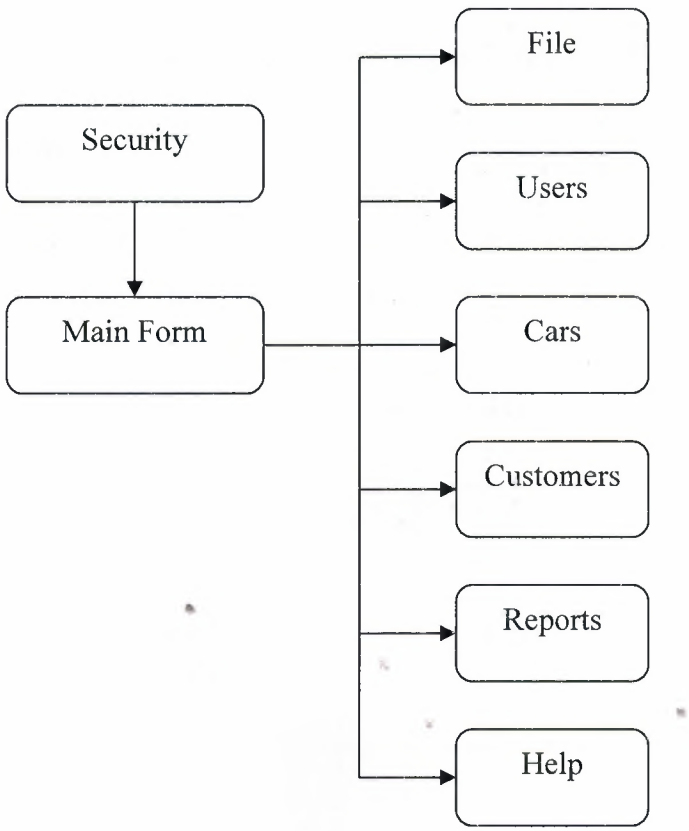

Figure 4.1 Flow diagram of the project

4.3 SECURITY

It's the first form that appears after starting the program, which allows only the registered users to access to the program.

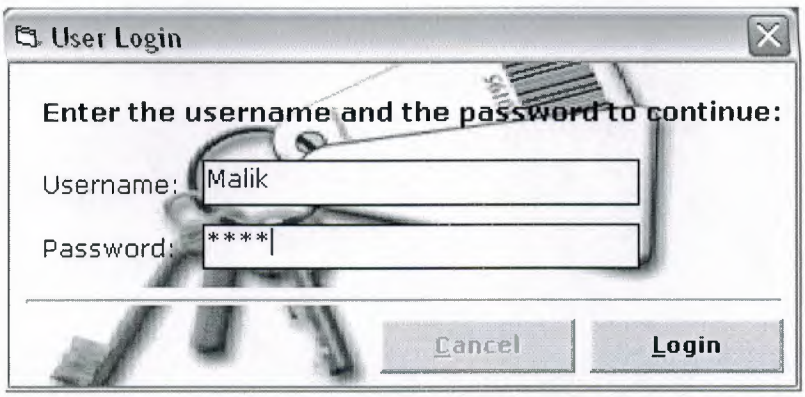

Figure 4.2 User Login Form

4.4 MAIN FORM

This is the user main interface which contains several menus such as File, Users, Cars, Customers, Report and Help. From this form we can access to any of these menus.

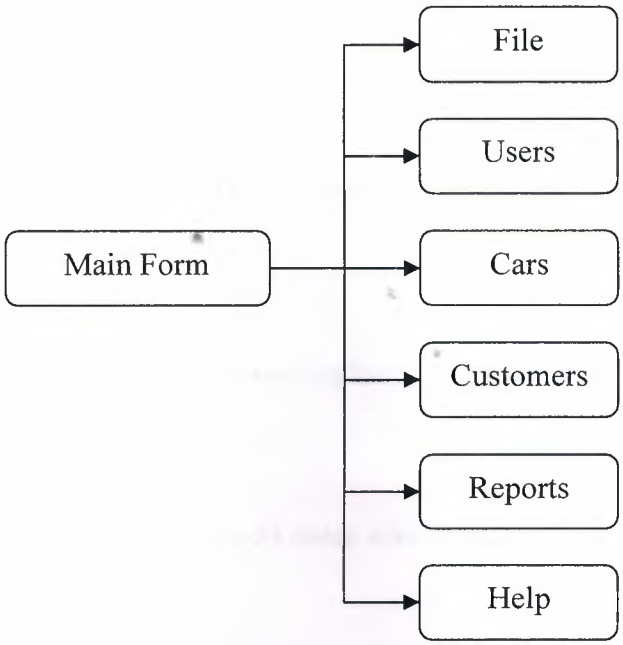

Figure 4.3 Main Form Diagram

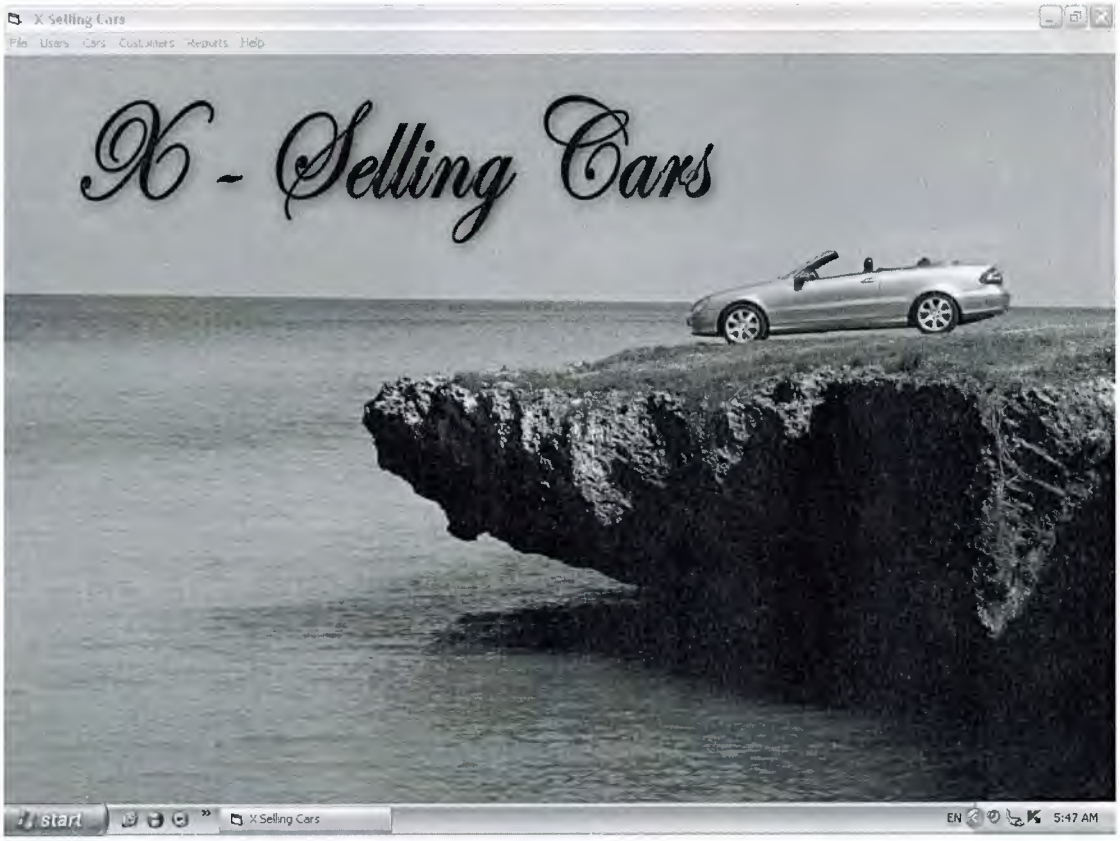

Figure 4.4 Main Form

4.4.1 FILE MENU

The file menu contains two options, Lock Application and Exit. The lock application option locks the program and it required a user name and password so the program will be unlocked again. The exit option will terminate the program.

4.4.2 USER MENU

The user menu also contains two options, Add User and Search for User.

4.4.2.1 ADD USER

Add user form allows us to add users who can use this application.

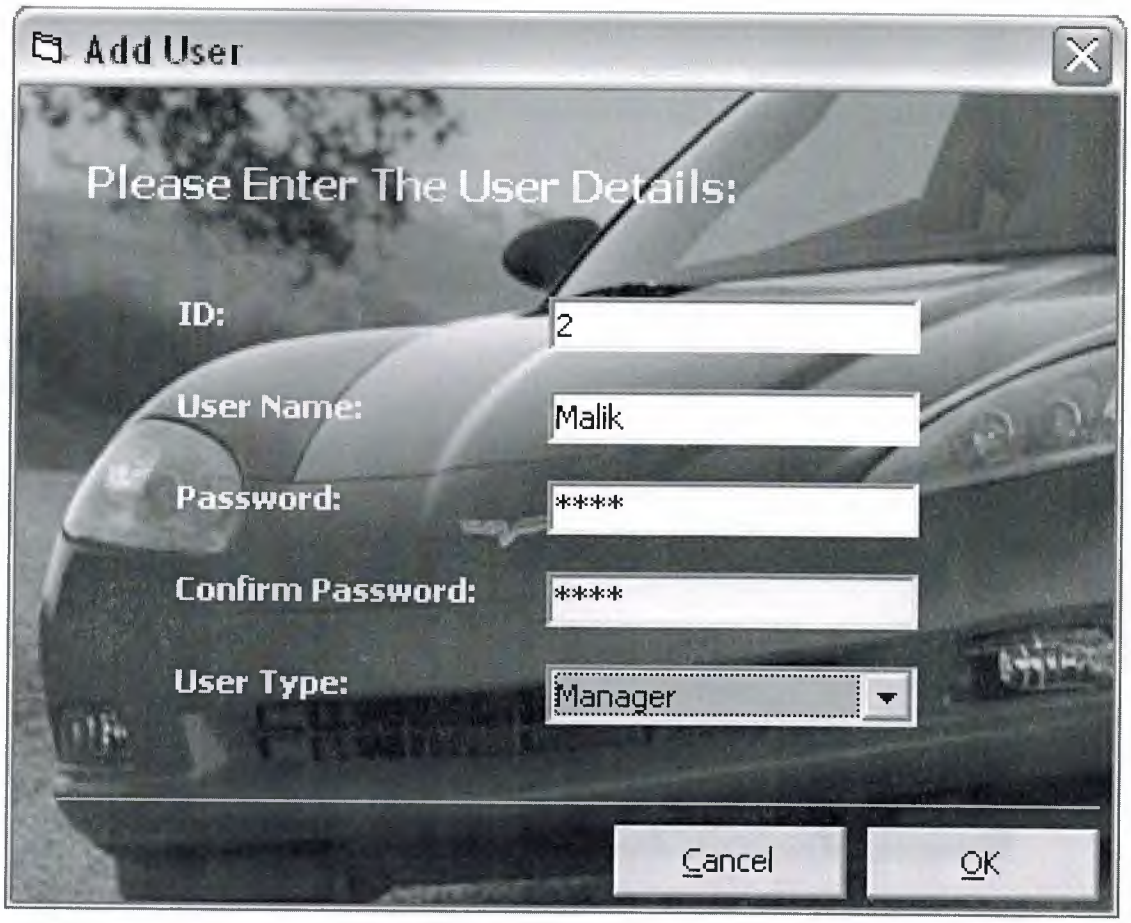

Figure 4.5 Add User Form

In add user form we got ID, User Name, Password, Confirm Password and User Type. The ID field generates by it self to set a different ID to each user. The user name and password fields are required so the user can use it to access the program. The confirm password field is to make sure of the password that assigned in the password field. User type field is very important to identify the authority of the user as follows:

- Manager: can access all in all of the program properties.
- Salesman: can only access the sell car part and nothing else.
- User: can only view the cars that are available.

4.4.2.2 SEARCH FOR USER

Search for user form allows us to search for users.

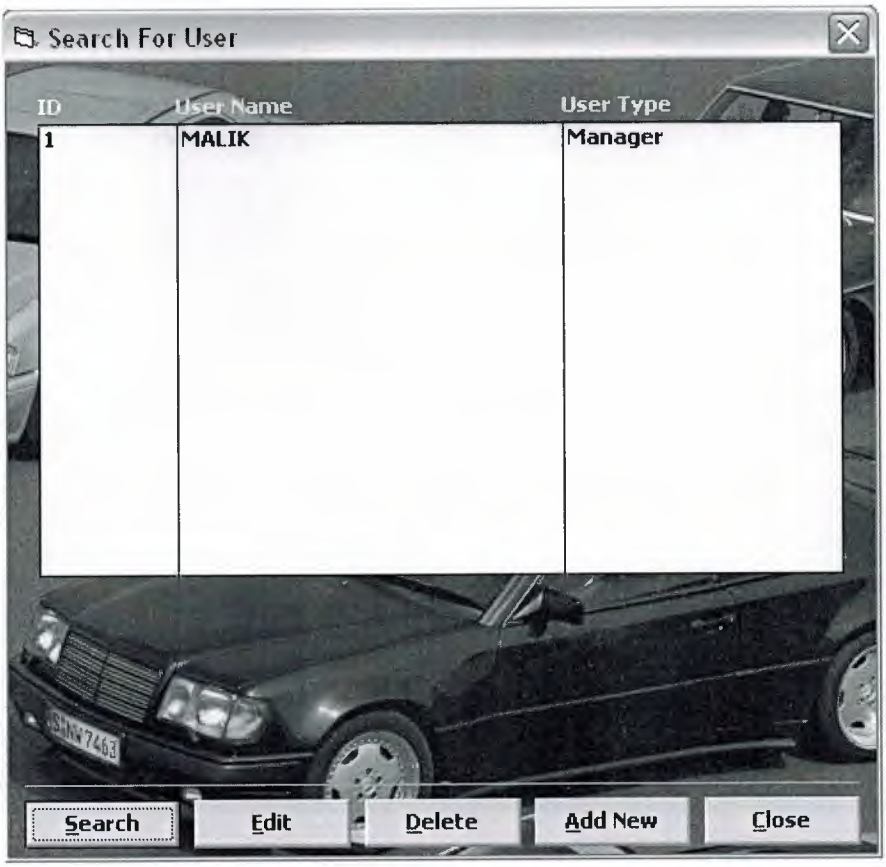

Figure 4.6 Search for User Form

In search for user form, we can Search, Edit, Add new and Delete users. If we click the Search button the following form appears.

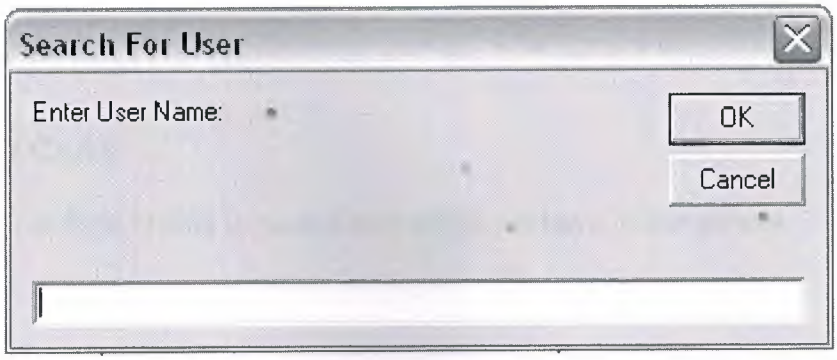

Figure 4.7 Search for User (Using User Name)

By entering the user name we can find the user that we are looking for. If we click the Edit button the following form appears.

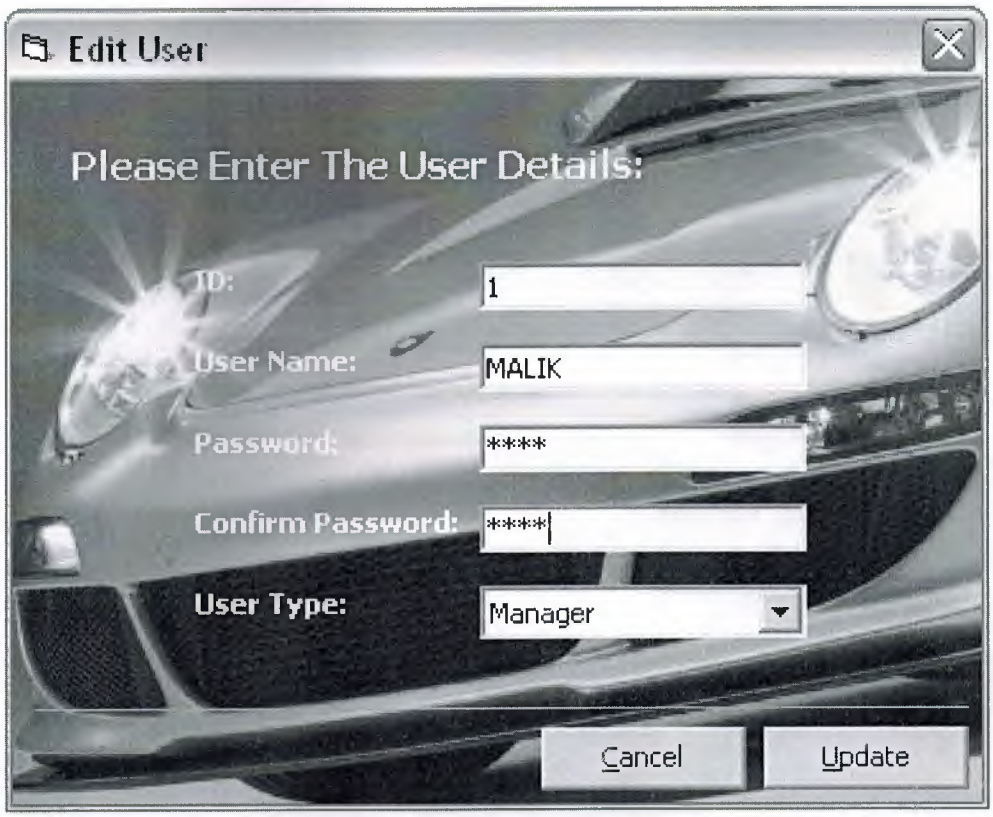

Figure 4.8 Edit User Form

In this form we can edit the user details and save them by pressing the Update button.

4.4.3 CARS MENU

The cars menu also contains two options, Add car and Search for a Car.

4.4.3.1 ADD CARS

Add car form allows us to add cars which we have in our gallery.

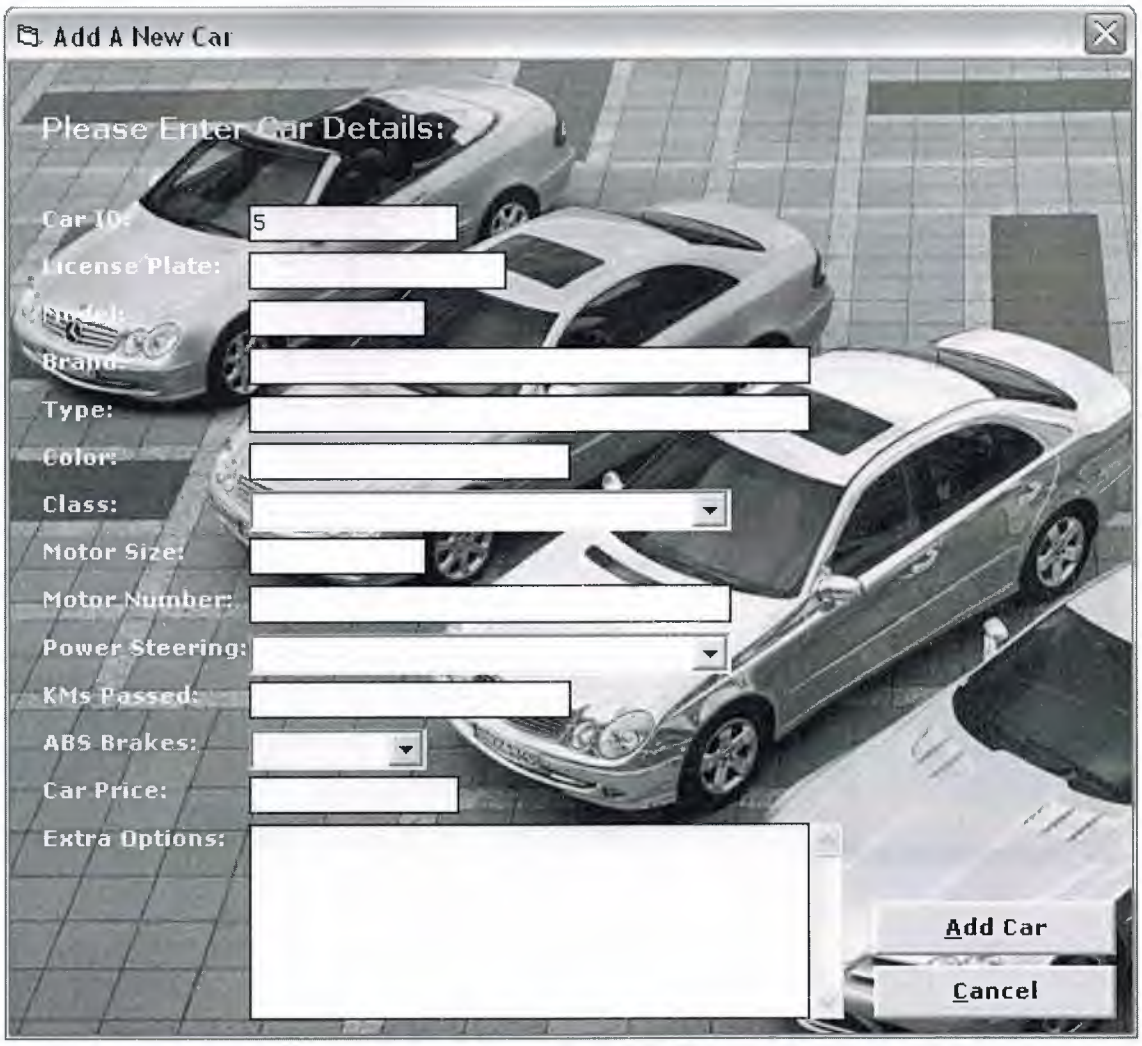

Figure 4.9 Add A New Car Form

In add a new car form we got some details which are important to the customer to see what kind of cars we have.

4.4.3.2 SEARCH FOR CAR

Search for car form allows us to search for cars.

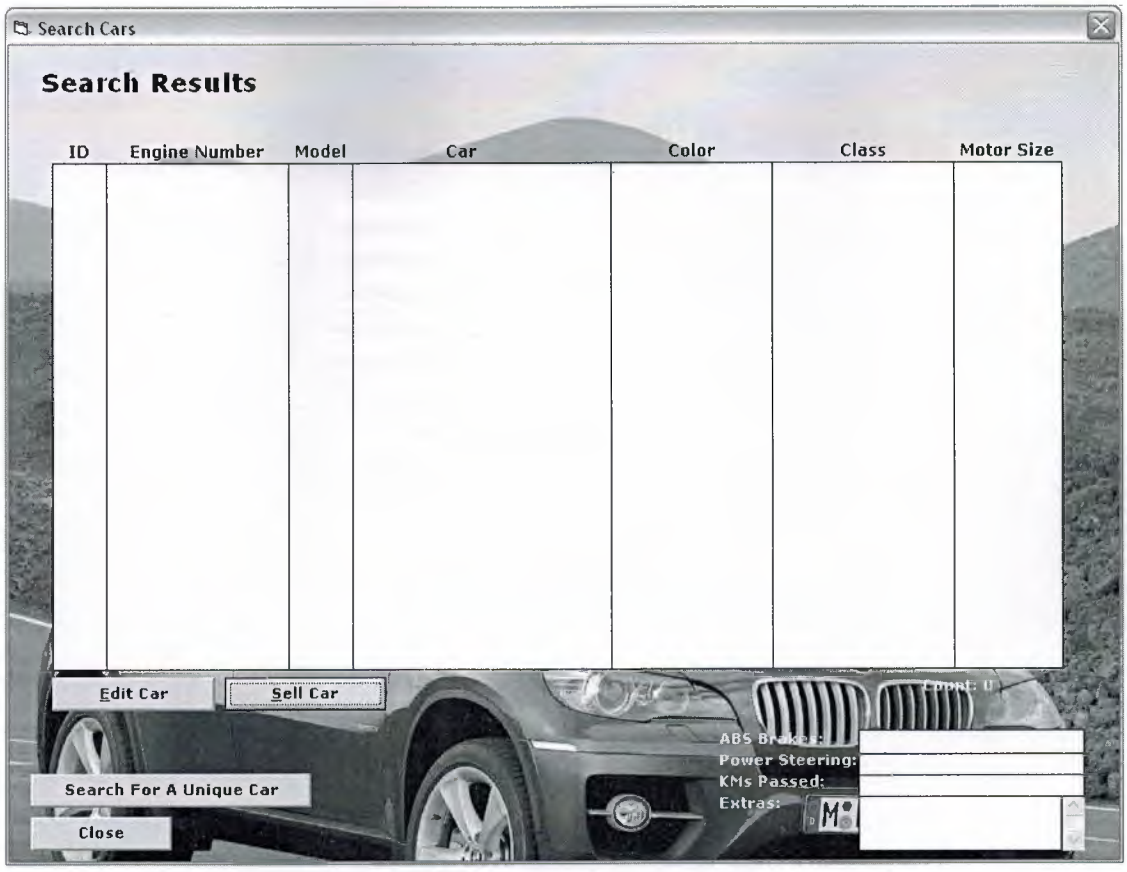

Figure 4.10 Search for Car Form

In search for car form, we can Sell, Edit and Unique search for a car. If we click the Edit button the following form appears.

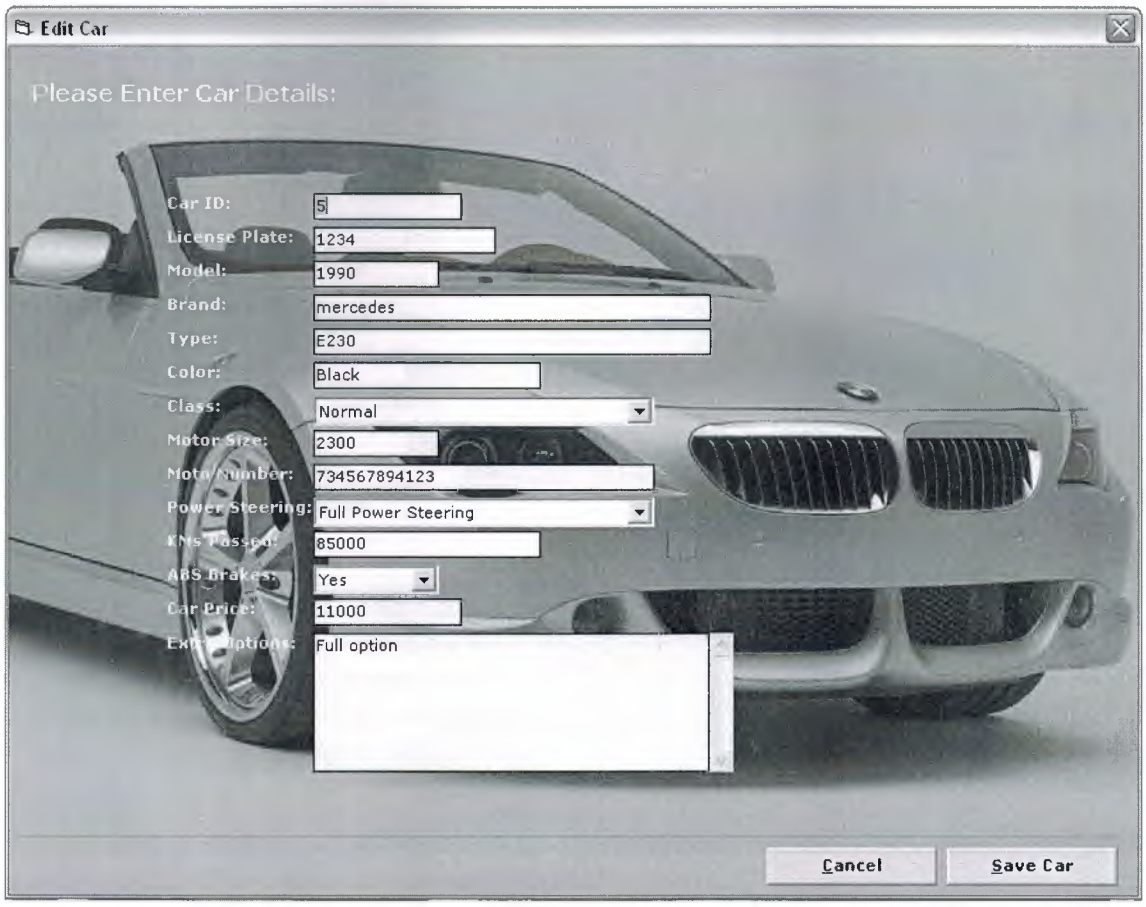

Figure 4.11 Edit Cars Form

In this form we can edit the car details and save them.

If we want to sell a car, from search for car form, if we click the sell car button after selecting the desired car, the following form appears.

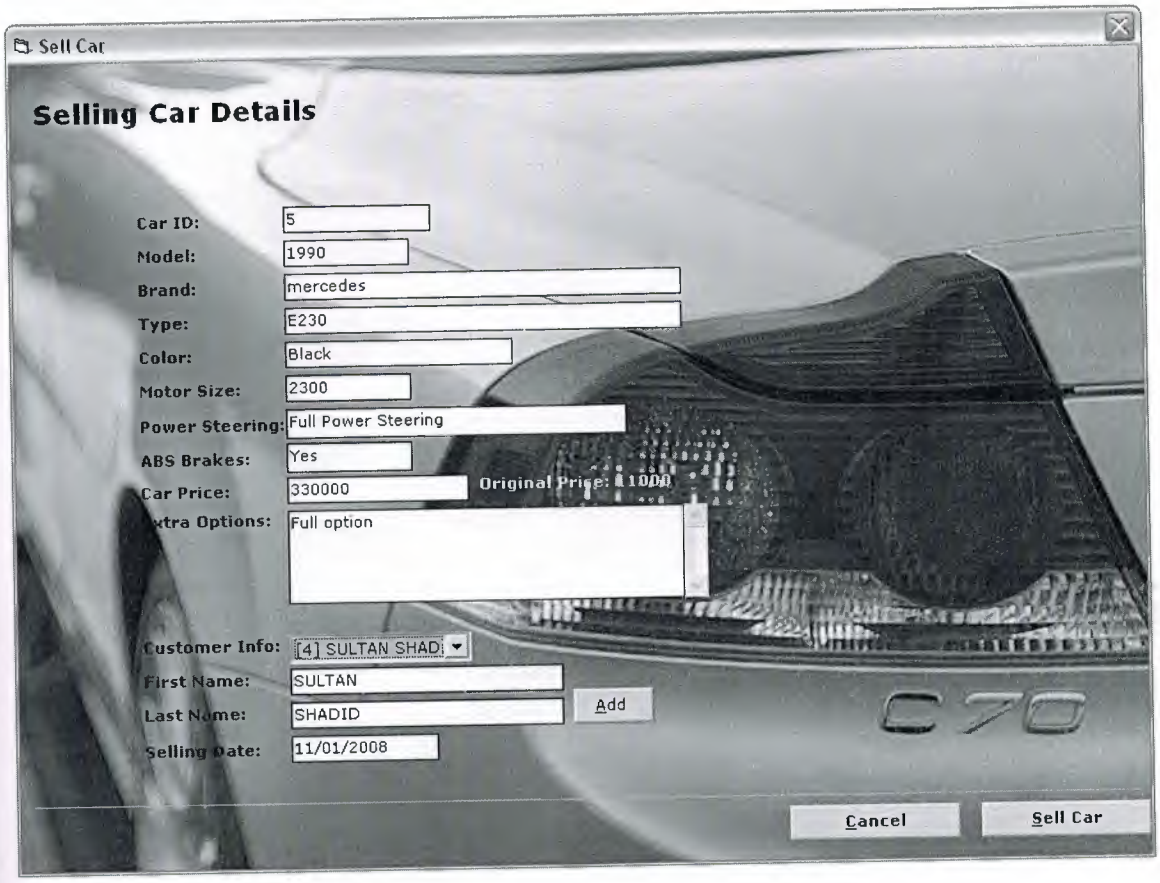

Figure 4.12 Sell Car Form

When this form appears, the car's details are already placed. By filling the customer info section, the car is ready to be sold by pressing the Sell Car button.

If we want to make a unique search for a car, by clicking Search for a Unique Car the following form appears.

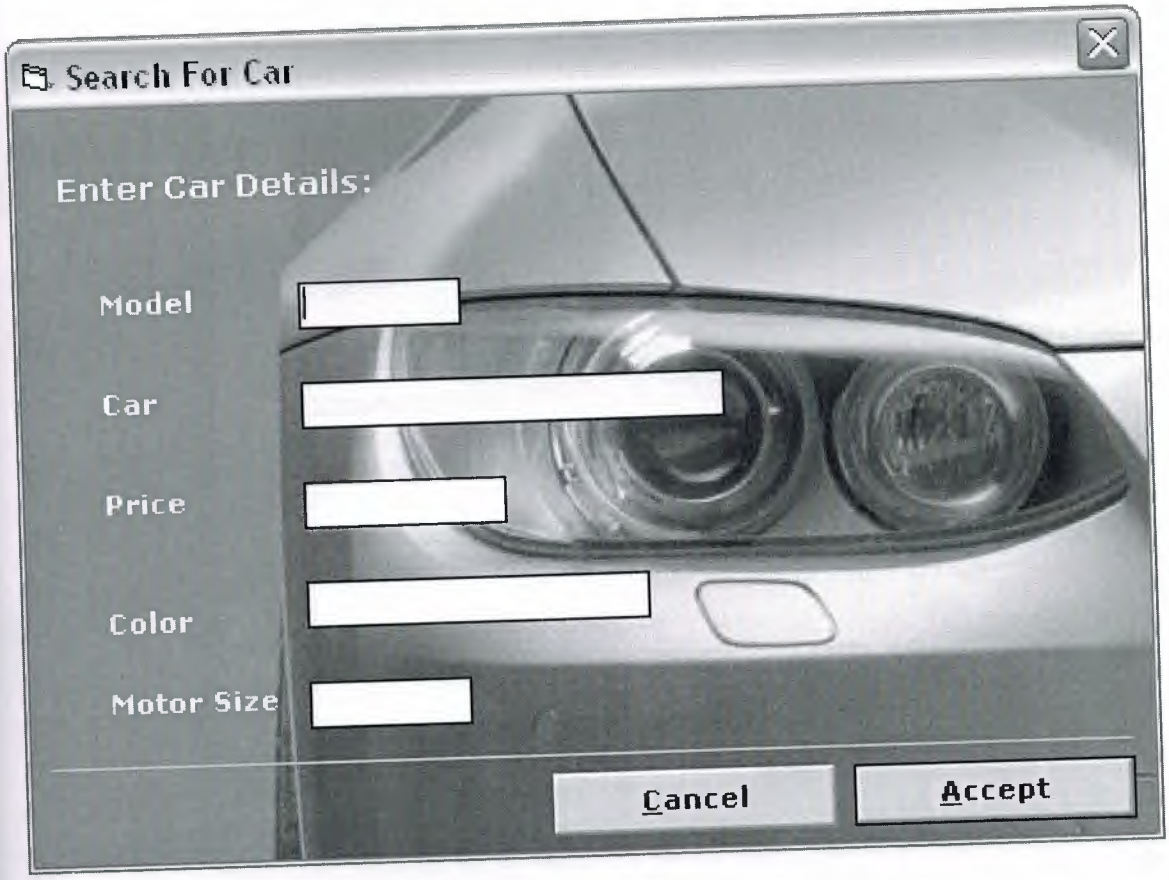

Figure 4.13 Unique Search Form

In this form, we can find any car we want by entering the favorable details that the customer looks for in a car so we can find him the car with the details that he or she looks for.

4.4.4 CUSTOMERS MENU

The customers menu also contains two options, Add customer and Search for Customers.

4.4.4.1 ADD CUSTOMERS

Add customer form allows us to add customers to our database so we can sell them cars without filling there information each time we want to sell them cars. Add customers has the following form.

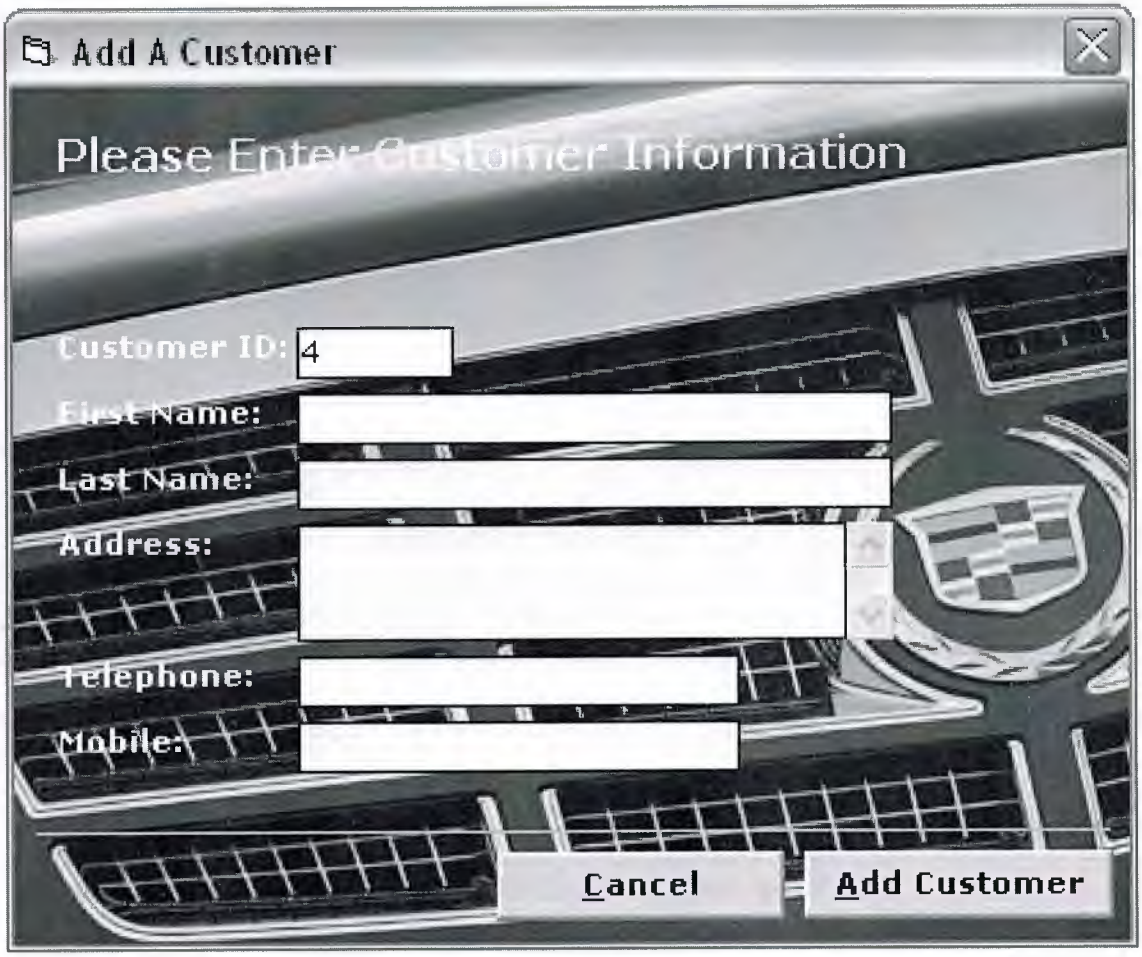

Figure 4.14 Add A New Customer

In add a new customer form we got some details which are related to the customers such as First Name, Last Name, Address, Telephone Number and Mobile Number. After filling the customer information, we can save it in out database by clicking Add Customer.

4.4.4.2 SEARCH FOR CUSTOMER

Search for customer form allows us to search for customers.

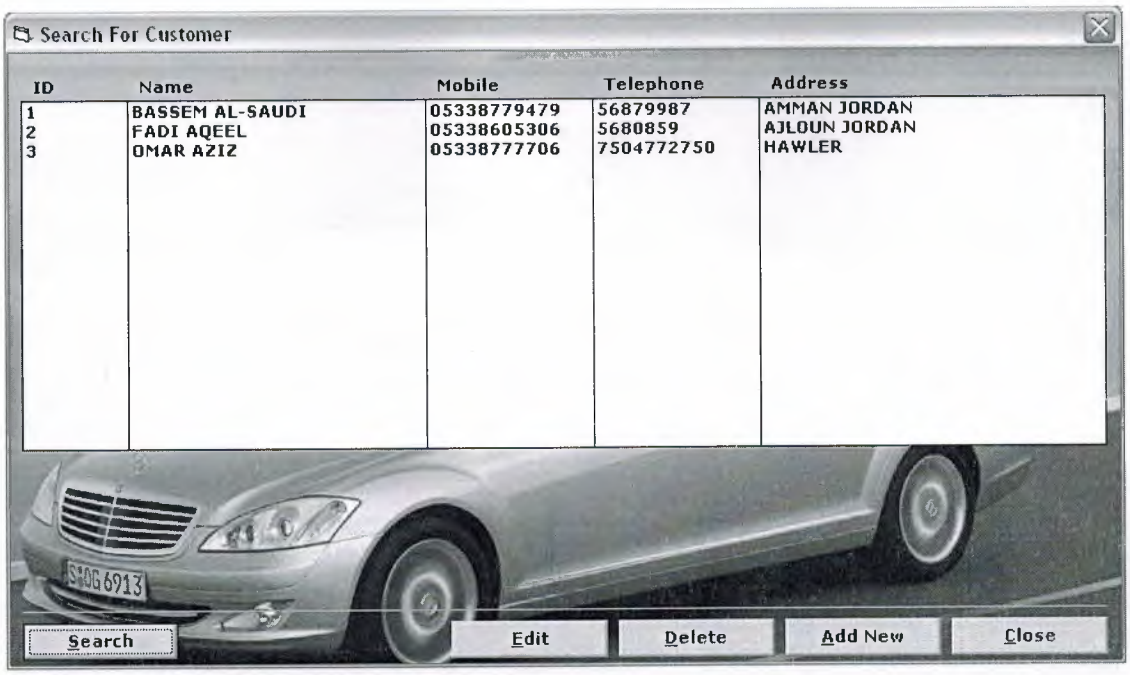

Figure 4.15 Search for Customer Form

In search for customer form, we can Search, Edit, Delete and Add new customers. If we click the Edit button the following form appears.

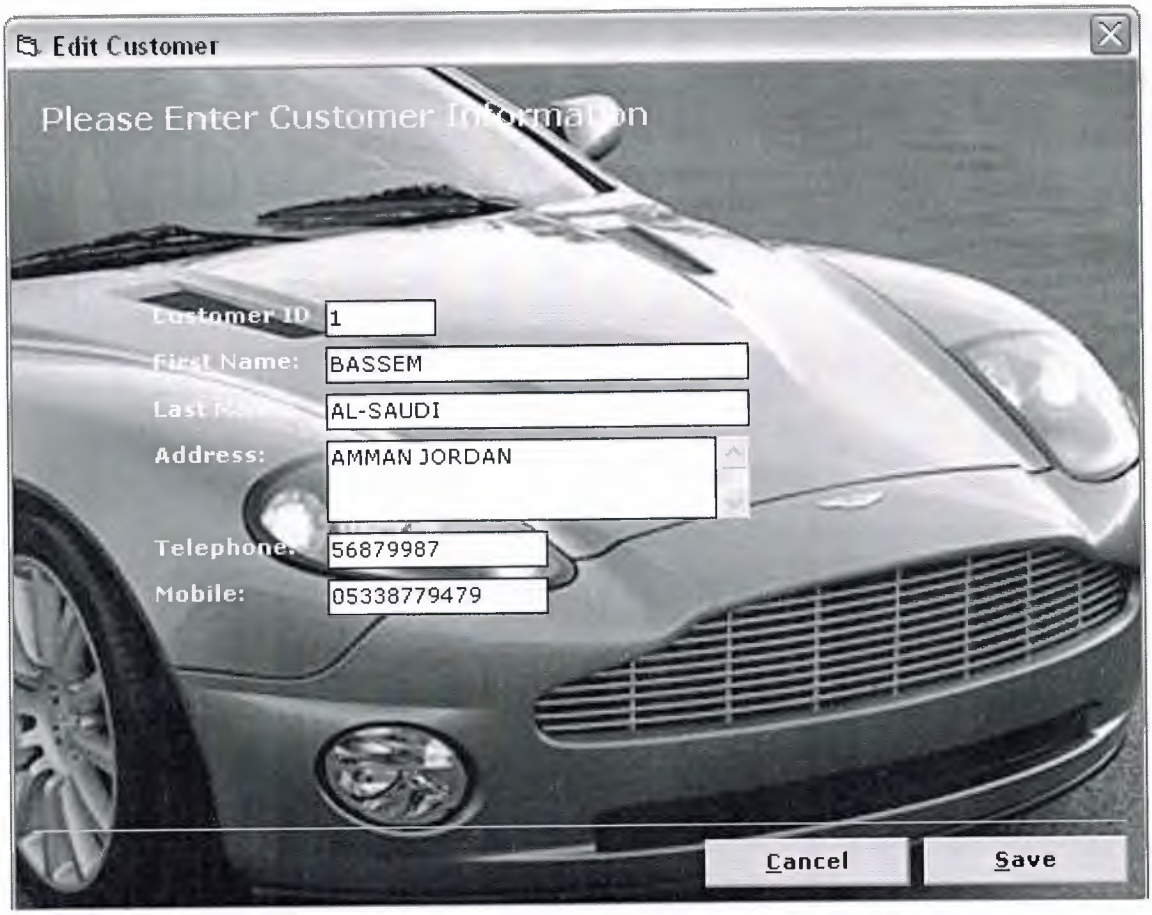

Figure 4.16 Edit Customers Form

In this form we can edit the customer information and save them in the database. If we click Search in the Search for Customer Form, the following form will appear.

Figure 4.17 Search for Customer Form (Using Customer Name)

When we write the customer name in the previous form we can find the customer we wanted. Also in the Search for Customer form, we can delete customers from our database, and add new customer by using the same form we described previously in this chapter.

4.4.5 REPORT MENU

In this menu we can view several kinds of reports:

- Customer report: shows who bought a car from us.
- Buyers report: shows the buyers between two dates we select.
- Cars report: shows the cars that came to our gallery.
- Sold car (invoice): shows the customers and what cars they bought.
- Sold car (report): shows the sold cars between two dates we select.

4.4.6 HELP MENU

This menu shows some of the programmer's information

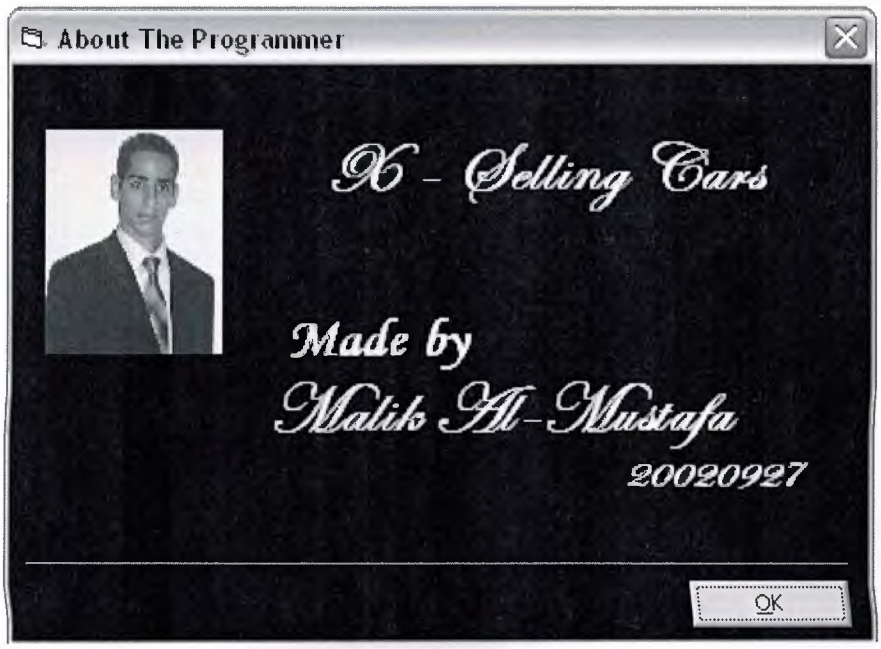

Figure 4.18 ABOUT

CONCLUSION

Visual Basic is a favorite programming environment of many programmers. When Visual Basic first appeared, it created a revolution in Windows programming, and that revolution continues to this day. Never before Windows programming had been so easy just build the program you want, and then run it. Visual Basic introduced unheard-of ease to Windows programming and changed programming from a chore to something very fun.

REFERANCE

[1] Visual Basic 6 Black Book

[2] Sams Teach Yourself Microsoft Acce

[3] Sam's Teach Yourself MySQL in 21 Da

[4] Visual Basic For Dummies 2005

[5] http://www.sqlcourse.com/table.html

[6] http://www.howstuffworks.com

[7] http//www.Computerhope.com

APPINDEX

```
Option Explicit

Private Sub cmdOk_Click()
    Unload Me
End Sub


Option Explicit

Private Sub cmdAddCar_Click()
    If CheckTexts = False Then Exit Sub

    DB.Execute ("INSERT INTO CARS(ID, LCSPLT, MODEL, BRAND,
TYPE, COLOR, CLASS, MOTORSIZE, " & _
                        "MOTORNUM, POWERST, KMPASSED,
ABSBRAKE, ORIGINALPRICE, EXTRAS, SOLDPRICE) " & _
                        "VALUES(" & txtCarDet(0).Text & ",
" & _
                        "'" & txtCarDet(9).Text & "', " & _
                        txtCarDet(1).Text & ", " & _
                        "'" & txtCarDet(10).Text & "', " &
_
                        "'" & txtCarDet(2).Text & "', " & _
                        "'" & txtCarDet(3).Text & "', " & _
                        comClass.ListIndex & ", " & _
                        txtCarDet(4).Text & ", " & _
                        "'" & txtCarDet(5).Text & "', " & _
                        comSteering.ListIndex & ", " & _
                        txtCarDet(6).Text & ", " & _
                        IIf(comBrakes.ListIndex = 0,
DB_TRUE, DB_FALSE) & ", " & _
                        txtCarDet(7).Text & ", " & _
                        "'" & txtCarDet(8).Text & "', 0)")

    If MsgBox("Car Was Added Successfully!" & vbNewLine &
"Add Another?", _
        vbYesNo + vbInformation, "Add A New Car") = vbYes
Then

        Call ClearTexts
        Call MakeID

        txtCarDet(0).SetFocus
    Else

        Unload Me
    End If
End Sub
```

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub comBrakes_KeyDown(KeyCode As Integer, Shift As
Integer)
    If KeyCode = vbKeyReturn Then txtCarDet(7).SetFocus
End Sub

Private Sub comClass_KeyDown(KeyCode As Integer, Shift As
Integer)
    If KeyCode = vbKeyReturn Then txtCarDet(4).SetFocus
End Sub

Private Sub comSteering_KeyDown(KeyCode As Integer, Shift
As Integer)
    If KeyCode = vbKeyReturn Then txtCarDet(6).SetFocus
End Sub

Private Sub Form_Load()
    Call MakeID
End Sub

Private Sub txtCarDet_GotFocus(Index As Integer)
    SendKeys "{home}+{end}"
End Sub

Private Sub txtCarDet_KeyDown(Index As Integer, KeyCode As
Integer, Shift As Integer)
    If KeyCode = vbKeyReturn Then
        If Index = 0 Then
            txtCarDet(9).SetFocus
        ElseIf Index = 3 Then
            comClass.SetFocus
        ElseIf Index = 5 Then
            comSteering.SetFocus
        ElseIf Index = 6 Then
            comBrakes.SetFocus
        ElseIf Index = 8 And Shift = 0 Then
            cmdAddCar.SetFocus
        ElseIf Index = 8 And Shift = 1 Then

        ElseIf Index = 9 Then
            txtCarDet(1).SetFocus
        Else
            txtCarDet(Index + 1).SetFocus
        End If
    End If
End Sub
```

```
Private Sub txtCarDet_KeyPress(Index As Integer, KeyAscii
As Integer)
    If (Index = 1 Or Index = 4 Or Index = 6) Then KeyAscii
= IntInput(KeyAscii)
    If (Index = 5 Or Index = 9) Then KeyAscii =
StrInput(KeyAscii)
    If (Index = 7) Then KeyAscii = CurInput(KeyAscii)
End Sub

Private Function CheckTexts() As Boolean
    Dim i As Integer

    For i = 1 To 9
        If txtCarDet(i).Text = "" Then

            MsgBox "Please Fill All Text Boxes!"
            txtCarDet(i).SetFocus

            CheckTexts = False
            Exit Function
        End If
    Next i

    If comClass.ListIndex < 0 Then
        CheckTexts = False
        MsgBox "Please Select A Class!"
        comClass.SetFocus
        Exit Function

    ElseIf comSteering.ListIndex < 0 Then
        CheckTexts = False
        MsgBox "Please Select Steering Type!"
        comClass.SetFocus
        Exit Function

    ElseIf comBrakes.ListIndex < 0 Then
        CheckTexts = False
        MsgBox "Please Select Brakes Type!"
        comBrakes.SetFocus
        Exit Function
    End If

    CheckTexts = True
End Function

Private Sub MakeID()
    Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
    Set TB = DB.OpenRecordset("SELECT MAX(ID) FROM CARS")
```

```
    If TB.Fields(0) > 0 Then
        txtCarDet(0).Text = TB.Fields(0) + 1
    Else
        txtCarDet(0).Text = 1
    End If
End Sub

Private Sub ClearTexts()
    Dim i As Integer

    For i = 1 To 10
        txtCarDet(i).Text = ""
    Next i

    comBrakes.ListIndex = -1
    comClass.ListIndex = -1
    comSteering.ListIndex = -1
End Sub

Option Explicit

Private Sub MakeCustID()
    Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
    Set TB = DB.OpenRecordset("SELECT MAX(ID) FROM
CUSTOMERS")

    If TB.Fields(0) > 0 Then
        txtCusID.Text = TB.Fields(0) + 1
    Else
        txtCusID.Text = 1
    End If
End Sub

Private Sub cmdAddCust_Click()
    If IsNumeric(txtCusMob.Text) = False Then
        MsgBox "Please enter a valid mobile number!",
vbCritical, "Error"
        txtCusMob.SetFocus
        SendKeys "{home}+{end}"
        Exit Sub
    End If

    If txtCusFName.Text = "" Or txtCusLName.Text = "" Or
txtCusMob.Text = "" Then
        MsgBox "Please enter all customer details!",
vbCritical, "Error"

    Else
        Dim SQL As String
```

```
        SQL = "INSERT INTO CUSTOMERS (ID, FNAME, LNAME,
ADDRESS, MOBILE, TEL) VALUES("
        SQL = SQL & txtCusID.Text & ", '" &
txtCusFName.Text & "', '" & txtCusLName.Text & "', '"
        SQL = SQL & txtCusAdd.Text & "', '" &
txtCusMob.Text & "', '" & txtCustTel.Text & "');"

        DB.Execute SQL

        MsgBox "Customer Was Added Successfully.",
vbInformation, "Successful"

        If frmSellCar.Visible = True Then
            frmSellCar.txtCustFName.Text = txtCusFName.Text
            frmSellCar.txtCustLName.Text = txtCusLName.Text
        End If

        Unload Me
    End If
End Sub

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    Call MakeCustID
End Sub

Private Sub txtCusAdd_KeyPress(KeyAscii As Integer)
    KeyAscii = StrInput(KeyAscii)
End Sub

Private Sub txtCusFName_KeyPress(KeyAscii As Integer)
    KeyAscii = StrInput(KeyAscii)
End Sub

Private Sub txtCusLName_KeyPress(KeyAscii As Integer)
    KeyAscii = StrInput(KeyAscii)
End Sub

Private Sub txtCusMob_KeyPress(KeyAscii As Integer)
    KeyAscii = TelNumInput(KeyAscii)
End Sub

Private Sub txtCustTel_KeyPress(KeyAscii As Integer)
    KeyAscii = TelNumInput(KeyAscii)
End Sub

Option Explicit
```

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdOk_Click()
    On Error Resume Next
    If txtUsr.Text = "" Then
        MsgBox "No Username Entered!", vbCritical, "Error"
        txtUsr.SetFocus
        Exit Sub

    ElseIf txtPwd.Text = "" Then
        MsgBox "No Password Entered!", vbCritical, "Error"
        txtPwd.SetFocus
        Exit Sub

    ElseIf txtCnfm.Text = "" Then
        MsgBox "Please Confirm Your Password!", vbCritical,
"Error"
        txtCnfm.SetFocus
        Exit Sub

    ElseIf combType.ListIndex = -1 Then
        MsgBox "Please Select The User Type!", vbCritical,
"Error"
        combType.SetFocus
        Exit Sub

    ElseIf txtPwd.Text <> txtCnfm.Text Then
        MsgBox "Password And Confirmed Password Does Not
Match!", vbCritical, "Error"
        txtCnfm.SetFocus
        Exit Sub
    End If

    DB.Execute "INSERT INTO USERS VALUES(" & txtID.Text &
", '" & txtUsr.Text & "', '" & txtPwd.Text & "','" &
combType.List(combType.ListIndex) & "');"

    If MsgBox("User Was Added Successfully, Add Another?",
vbQuestion + vbYesNo, "Add User") = vbYes Then
        Call Form_Load
        txtUsr.Text = ""
        txtPwd.Text = ""
        txtCnfm.Text = ""
        combType.ListIndex = -1
    Else
        Unload Me
    End If
```

```
End Sub

Private Sub Form_Load()
    Dim NewID As Long
    Set TB = DB.OpenRecordset("SELECT MAX(ID) FROM USERS;")

    If TB.EOF Then
        NewID = 1
    Else
        NewID = TB.Fields(0) + 1
    End If

    txtID.Text = NewID
End Sub

Option Explicit

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdOk_Click()
    On Error Resume Next
    If txtUsr.Text = "" Then
        MsgBox "No Username Entered!", vbCritical, "Error"
        txtUsr.SetFocus
        Exit Sub

    ElseIf txtPwd.Text = "" Then
        MsgBox "No Password Entered!", vbCritical, "Error"
        txtPwd.SetFocus
        Exit Sub

    ElseIf txtCnfm.Text = "" Then
        MsgBox "Please Confirm Your Password!", vbCritical,
"Error"
        txtCnfm.SetFocus
        Exit Sub

    ElseIf combType.ListIndex = -1 Then
        MsgBox "Please Select The User Type!", vbCritical,
"Error"
        combType.SetFocus
        Exit Sub

    ElseIf txtPwd.Text <> txtCnfm.Text Then
        MsgBox "Password And Confirmed Password Does Not
Match!", vbCritical, "Error"
        txtCnfm.SetFocus
        Exit Sub
```

```
    End If

    DB.Execute "INSERT INTO USERS VALUES(" & txtID.Text &
", '" & txtUsr.Text & "', '" & txtPwd.Text & "', '" &
combType.List(combType.ListIndex) & "');"

    If MsgBox("User Was Added Successfully, Add Another?",
vbQuestion + vbYesNo, "Add User") = vbYes Then
        Call Form_Load
        txtUsr.Text = ""
        txtPwd.Text = ""
        txtCnfm.Text = ""
        combType.ListIndex = -1
    Else
        Unload Me
    End If
End Sub

Private Sub Form_Load()
    Dim NewID As Long
    Set TB = DB.OpenRecordset("SELECT MAX(ID) FROM USERS;")

    If TB.EOF Then
        NewID = 1
    Else
        NewID = TB.Fields(0) + 1
    End If

    txtID.Text = NewID
End Sub

Option Explicit

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdUpdateCust_Click()
    If IsNumeric(txtCusMob.Text) = False Then
        MsgBox "invalid mobile number!", vbCritical,
"Error"
        txtCusMob.SetFocus
        SendKeys "{home}+{end}"
        Exit Sub
    End If

    If txtCusFName.Text = "" Or txtCusLName.Text = "" Or
txtCusMob.Text = "" Or txtCusTel.Text = "" Then
        MsgBox "Please fill all customer details!",
vbCritical, "Error"
```

```
    Else
        Dim SQL As String

        SQL = "UPDATE CUSTOMERS SET FNAME='" &
txtCusFName.Text & "', LNAME='" & txtCusLName.Text & "',
ADDRESS='" & txtCusAdd.Text & "', MOBILE='" &
txtCusMob.Text & "', TEL='" & txtCusTel.Text & "' WHERE
ID=" & txtCusID.Text

        DB.Execute SQL

        MsgBox "Customer Information Was Updated
Successfully.", vbInformation, "Successful"
        Unload Me
    End If

End Sub

Private Sub txtCusFName_KeyPress(KeyAscii As Integer)
    KeyAscii = StrInput(KeyAscii)
End Sub

Private Sub txtCusLName_KeyPress(KeyAscii As Integer)
    KeyAscii = StrInput(KeyAscii)
End Sub

Private Sub txtCusMob_KeyPress(KeyAscii As Integer)
    KeyAscii = IntInput(KeyAscii)
End Sub

Private Sub txtCusTel_KeyPress(KeyAscii As Integer)
    KeyAscii = IntInput(KeyAscii)
End Sub

Option Explicit

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdOk_Click()
    On Error Resume Next
    If txtUsr.Text = "" Then
        MsgBox "No Username Entered!", vbCritical, "Error"
        txtUsr.SetFocus
        Exit Sub

    ElseIf txtPwd.Text = "" Then
        MsgBox "No Password Entered!", vbCritical, "Error"
```

```
        txtPwd.SetFocus
        Exit Sub

    ElseIf txtCnfm.Text = "" Then
        MsgBox "Please Confirm Your Password!", vbCritical,
"Error"
        txtCnfm.SetFocus
        Exit Sub

    ElseIf combType.ListIndex = -1 Then
        MsgBox "Please Select The User Type!", vbCritical,
"Error"
        combType.SetFocus
        Exit Sub

    ElseIf txtPwd.Text <> txtCnfm.Text Then
        MsgBox "Password And Confirmed Password Does Not
Match!", vbCritical, "Error"
        combType.SetFocus
        Exit Sub
    End If

    DB.Execute "UPDATE USERS SET USERNAME='" & txtUsr.Text
& "', PASSWORD='" & txtPwd.Text & "', REST='" &
combType.List(combType.ListIndex) & "' WHERE ID=" &
txtID.Text & ";"

    MsgBox "User Was Updated Successfully?", vbInformation,
"Edit User"
    Unload Me

End Sub


Option Explicit

Private Sub cmdExit_Click()
    End
End Sub

Private Sub cmdLogin_Click()
    Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
    Set TB = DB.OpenRecordset("SELECT * FROM USERS WHERE
USERNAME='" & txtUser.Text & "'")

    If TB.EOF = False Then
        If TB("PASSWORD") = txtPass.Text Then

            myUserID = TB.Fields("ID")
```

```
            UserRestType = IIf(IsNull(TB.Fields("REST")) =
True, "", TB.Fields("REST"))

            If UserRestType = "Manager" Then
                If frmMain.Visible = False Then
frmMain.Show
            Else
                MsgBox "Access To The Main Program Is
Forbidden!", vbCritical, "Sales Manager"
                If frmMain.Visible Then Unload Me: Unload
frmMain
                frmSrchForCar.Show
                frmSrchForCar.cmdEditCar.Enabled = False
            End If
            Unload Me
        Else
            MsgBox "Wrong Password!", vbCritical, "Error"
        End If
    Else
        MsgBox "User Not Found!", vbCritical, "Error"
    End If
End Sub

Private Sub Form_Load()
    ChDir App.Path
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode
As Integer)
    If Not UnloadMode = vbFormCode Then Cancel = Not
cmdExit.Enabled
End Sub

Private Sub txtPass_KeyDown(KeyCode As Integer, Shift As
Integer)
    If KeyCode = vbKeyReturn Then Call cmdLogin_Click
End Sub

Private Sub txtUser_KeyDown(KeyCode As Integer, Shift As
Integer)
    If KeyCode = vbKeyReturn Then txtPass.SetFocus
End Sub

Option Explicit

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode
As Integer)
    If Not UnloadMode = vbFormCode Then
        Cancel = True
        Call mnuExit_Click
```

```
    End If
End Sub

Private Sub mnuAbout_Click()
    frmAbout.Show vbModal
End Sub

Private Sub mnuAddCus_Click()
    frmAddCustomer.Show vbModal
End Sub

Private Sub mnuAddNew_Click()
    frmAddCar.Show vbModal
End Sub

Private Sub mnuAddUser_Click()
    frmAddUser.Show vbModal
End Sub

Private Sub mnuBuyRep_Click()
    frmReportCust.Show vbModal
End Sub

Private Sub mnuCarRev_Click()
    frmReport.Show vbModal
End Sub

Private Sub mnuCarRprt_Click()
    rprtCars.Show vbModal
End Sub

Private Sub mnuCusRep_Click()
    rprtCust.Show vbModal
End Sub

Private Sub mnuExit_Click()
    If MsgBox("Are you sure you want to exit?", vbYesNo +
vbQuestion, "Exit") = vbYes Then End
End Sub

Private Sub mnuInvoices_Click()
    rprtInvoice.Show vbModal
End Sub

Private Sub mnuLockApp_Click()
    Load frmLogin
    frmLogin.cmdExit.Enabled = False
    frmLogin.Show vbModal
End Sub
```

```
Private Sub mnuSrch_Click()
    frmSrchForCar.Show vbModal
End Sub

Private Sub mnuSrchCus_Click()
    frmSrchForCust.Show vbModal
End Sub

Private Sub mnuSrchUsr_Click()
    frmUsers.Show vbModal
End Sub

Option Explicit

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode
As Integer)
    If Not UnloadMode = vbFormCode Then
        Cancel = True
        Call mnuExit_Click
    End If
End Sub

Private Sub mnuAbout_Click()
    frmAbout.Show vbModal
End Sub

Private Sub mnuAddCus_Click()
    frmAddCustomer.Show vbModal
End Sub

Private Sub mnuAddNew_Click()
    frmAddCar.Show vbModal
End Sub

Private Sub mnuAddUser_Click()
    frmAddUser.Show vbModal
End Sub

Private Sub mnuBuyRep_Click()
    frmReportCust.Show vbModal
End Sub

Private Sub mnuCarRev_Click()
    frmReport.Show vbModal
End Sub

Private Sub mnuCarRprt_Click()
    rprtCars.Show vbModal
End Sub
```

```
Private Sub mnuCusRep_Click()
    rprtCust.Show vbModal
End Sub

Private Sub mnuExit_Click()
    If MsgBox("Are you sure you want to exit?", vbYesNo +
vbQuestion, "Exit") = vbYes Then End
End Sub

Private Sub mnuInvoices_Click()
    rprtInvoice.Show vbModal
End Sub

Private Sub mnuLockApp_Click()
    Load frmLogin
    frmLogin.cmdExit.Enabled = False
    frmLogin.Show vbModal
End Sub

Private Sub mnuSrch_Click()
    frmSrchForCar.Show vbModal
End Sub

Private Sub mnuSrchCus_Click()
    frmSrchForCust.Show vbModal
End Sub

Private Sub mnuSrchUsr_Click()
    frmUsers.Show vbModal
End Sub
```

```
Option Explicit

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode
As Integer)
    If Not UnloadMode = vbFormCode Then
        Cancel = True
        Call mnuExit_Click
    End If
End Sub

Private Sub mnuAbout_Click()
    frmAbout.Show vbModal
End Sub

Private Sub mnuAddCus_Click()
    frmAddCustomer.Show vbModal
End Sub

Private Sub mnuAddNew_Click()
    frmAddCar.Show vbModal
End Sub

Private Sub mnuAddUser_Click()
    frmAddUser.Show vbModal
End Sub

Private Sub mnuBuyRep_Click()
    frmReportCust.Show vbModal
End Sub

Private Sub mnuCarRev_Click()
    frmReport.Show vbModal
End Sub

Private Sub mnuCarRprt_Click()
    rprtCars.Show vbModal
End Sub

Private Sub mnuCusRep_Click()
    rprtCust.Show vbModal
End Sub

Private Sub mnuExit_Click()
    If MsgBox("Are you sure you want to exit?", vbYesNo +
vbQuestion, "Exit") = vbYes Then End
End Sub

Private Sub mnuInvoices_Click()
    rprtInvoice.Show vbModal
End Sub
```

```
Private Sub mnuLockApp_Click()
    Load frmLogin
    frmLogin.cmdExit.Enabled = False
    frmLogin.Show vbModal
End Sub

Private Sub mnuSrch_Click()
    frmSrchForCar.Show vbModal
End Sub

Private Sub mnuSrchCus_Click()
    frmSrchForCust.Show vbModal
End Sub

Private Sub mnuSrchUsr_Click()
    frmUsers.Show vbModal
End Sub

Option Explicit
Public OriginalPrice As Currency

Private Sub cmdAddCust_Click()
    frmAddCustomer.Show vbModal

    Call Form_Load
End Sub

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdSellCar_Click()
    Dim SQL As String, curProfit As Currency

    If Val(txtCarDet(5).Text) <= 0 Then
        MsgBox "Car Price Cannot Be Zero!", vbCritical,
"Error"

    ElseIf Combo1.ListIndex = -1 Then
        MsgBox "No Customer Was Selected!", vbCritical,
"Error"

    Else
        curProfit = CCur(txtCarDet(5).Text) - OriginalPrice

        SQL = "UPDATE CARS SET SOLDPRICE=" &
txtCarDet(5).Text & _
              ", PROFIT=" & curProfit & _
```

```
                ", CUSTID=" &
Combo1.ItemData(Combo1.ListIndex) &
                ", SELLDATE='" & Format$(Now, "dd/mm/yyyy") &
"',
                "', STATUS='SOLD" &
                "' WHERE ID=" & txtCarDet(0).Text

        DB.Execute SQL

        Set TB = DB.OpenRecordset("SELECT *,* FROM
CUSTOMERS, CARS WHERE CUSTOMERS.ID=" &
Combo1.ItemData(Combo1.ListIndex) & "AND CARS.ID=" &
txtCarDet(0).Text)

        With rprtInv

.Sections("Section4").Controls("lblInvNum").Caption =
CStr(TB.Fields("CUSTOMERS.ID")) +
CStr(TB.Fields("CARS.ID"))

.Sections("Section4").Controls("lblInvDate").Caption =
Format$(Now, "DD/MM/YYYY")

            .Sections("Section2").Controls("lblID").Caption
= TB.Fields("CARS.ID")

.Sections("Section2").Controls("lblModel").Caption =
TB.Fields("MODEL")

.Sections("Section2").Controls("lblType").Caption =
TB.Fields("BRAND") & " " & TB.Fields("TYPE")

.Sections("Section2").Controls("lblColor").Caption =
TB.Fields("COLOR")

.Sections("Section2").Controls("lblClass").Caption =
GetCarClass(TB.Fields("CLASS"))

.Sections("Section2").Controls("lblMot").Caption =
TB.Fields("MOTORSIZE") & " cc"

.Sections("Section2").Controls("lblPrice").Caption =
TB.Fields("SOLDPRICE")


.Sections("Section2").Controls("lblCusName").Caption =
TB.Fields("FNAME") & " " & TB.Fields("LNAME")

.Sections("Section2").Controls("lblCusMob").Caption =
TB.Fields("MOBILE")
```

```
.Sections("Section2").Controls("lblCusAdd").Caption =
TB.Fields("ADDRESS")

.Sections("Section2").Controls("lblTotalAmnt").Caption =
TB.Fields("SOLDPRICE")

            .Show vbModal
        End With

        MsgBox "Car Was Sold Successfully!", vbInformation,
"Sold"
        Unload Me
    End If
End Sub

Private Sub Combo1_Click()
    Dim MyNm$
    MyNm = Combo1.List(Combo1.ListIndex)
    MyNm = Right$(MyNm, Len(MyNm) - 4)
    txtCustFName.Text = Left$(MyNm, InStr(MyNm, " "))
    txtCustLName.Text = Right$(MyNm, Len(MyNm) -
Len(txtCustFName.Text))
End Sub

Private Sub comBrakes_Click()
    txtABS.Text = comBrakes.List(comBrakes.ListIndex)
End Sub

Private Sub comSteering_Click()
    txtPS.Text = comSteering.List(comSteering.ListIndex)
End Sub

Private Sub Form_Load()
    txtDate.Text = Format(Now, "DD/MM/YYYY")
    Set TB = DB.OpenRecordset("SELECT ID, FNAME, LNAME FROM
CUSTOMERS")

    Combo1.Clear

    While Not TB.EOF
        Combo1.AddItem "[" & TB.Fields("ID") & "] " &
TB.Fields("FNAME") & " " & TB.Fields("LNAME")
        Combo1.ItemData(Combo1.ListCount - 1) =
TB.Fields("ID")

        DoEvents
        TB.MoveNext
    Wend
End Sub
```

```
Option Explicit

Private Sub cmdAccept_Click()
    Dim sTemp As String
    Dim F As Boolean

    If Not txtModel.Text = "" Then
        F = True
        sTemp = sTemp & "MODEL=" & txtModel.Text
    End If

    If Not txtType.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "TYPE LIKE " & "'" & txtType.Text &
"*'": F = True
    End If

    If Not txtPrice.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "ORIGINALPRICE=" & txtPrice.Text: F
= True
    End If

    If Not txtColor.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "COLOR=" & "'" & txtColor.Text &
"'": F = True
    End If

    If Not txtMotorSize.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "MOTORSIZE=" & txtMotorSize.Text: F
= True
    End If

    If sTemp = "" Then sTemp = "ID>0"
    frmSrchForCar.UniqueSQL = sTemp

    Unload Me
End Sub

Private Sub cmdCancel_Click()
    frmSrchForCar.UniqueSQL = "ID>0"
    Unload Me
End Sub

Option Explicit
```

```
Private Sub cmdAccept_Click()
    Dim sTemp As String
    Dim F As Boolean

    If Not txtModel.Text = "" Then
        F = True
        sTemp = sTemp & "MODEL=" & txtModel.Text
    End If

    If Not txtType.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "TYPE LIKE " & "'" & txtType.Text &
"*'": F = True
    End If

    If Not txtPrice.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "ORIGINALPRICE=" & txtPrice.Text: F
= True
    End If

    If Not txtColor.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "COLOR=" & "'" & txtColor.Text &
"'": F = True
    End If

    If Not txtMotorSize.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "MOTORSIZE=" & txtMotorSize.Text: F
= True
    End If

    If sTemp = "" Then sTemp = "ID>0"
    frmSrchForCar.UniqueSQL = sTemp

    Unload Me
End Sub

Private Sub cmdCancel_Click()
    frmSrchForCar.UniqueSQL = "ID>0"
    Unload Me
End Sub

Option Explicit

Private Sub cmdAccept_Click()
    Dim sTemp As String
    Dim F As Boolean
```

```
    If Not txtModel.Text = "" Then
        F = True
        sTemp = sTemp & "MODEL=" & txtModel.Text
    End If

    If Not txtType.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "TYPE LIKE " & "'" & txtType.Text &
"*'": F = True
    End If

    If Not txtPrice.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "ORIGINALPRICE=" & txtPrice.Text: F
= True
    End If

    If Not txtColor.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "COLOR=" & "'" & txtColor.Text &
"'": F = True
    End If

    If Not txtMotorSize.Text = "" Then
        If F = True Then sTemp = sTemp & " AND "
        sTemp = sTemp & "MOTORSIZE=" & txtMotorSize.Text: F
= True
    End If

    If sTemp = "" Then sTemp = "ID>0"
    frmSrchForCar.UniqueSQL = sTemp

    Unload Me
End Sub

Private Sub cmdCancel_Click()
    frmSrchForCar.UniqueSQL = "ID>0"
    Unload Me
End Sub

Option Explicit

Dim CarOpt() As CAROPTIONS
Public UniqueSQL As String
Public IsChanged As Boolean
Private LastSQL As String


Private Sub cmdClose_Click()
    Unload Me
```

```
End Sub

Private Sub cmdEditCar_Click()
    Dim iSel As Integer

    If lstID.ListIndex = -1 Then
        MsgBox "Please Select A Car!", vbInformation, "No
Selection"
        Exit Sub
    End If

    iSel = lstID.List(lstID.ListIndex)

    Set TB = DB.OpenRecordset("SELECT * FROM CARS WHERE
ID=" & iSel)

    Load frmEditCar

    With frmEditCar
        .txtCarDet(0) = TB.Fields("ID")
        .txtCarDet(9).Text =
IIf(IsNull(TB.Fields("LCSPLT")), "Empty",
TB.Fields("LCSPLT"))
        .txtCarDet(1).Text = TB.Fields("MODEL")
        .txtCarDet(10).Text = TB.Fields("BRAND")
        .txtCarDet(2).Text = TB.Fields("TYPE")
        .txtCarDet(2).Text = TB.Fields("TYPE")
        .txtCarDet(3).Text = TB.Fields("COLOR")
        .comClass.ListIndex = TB.Fields("CLASS")
        .txtCarDet(4).Text = TB.Fields("MOTORSIZE")
        .txtCarDet(5).Text = TB.Fields("MOTORNUM")
        .comSteering.ListIndex = TB.Fields("POWERST")
        .txtCarDet(6).Text = TB.Fields("KMPASSED")
        .comBrakes.ListIndex = IIf(TB.Fields("ABSBRAKE") =
DB_TRUE, 0, 1)
        .txtCarDet(7).Text = TB.Fields("ORIGINALPRICE")
        .txtCarDet(8).Text = TB.Fields("EXTRAS")

        .Show vbModal
    End With

    If IsChanged = False Then Exit Sub

    Set TB = DB.OpenRecordset("SELECT * FROM CARS;")


    If TB.EOF = False Then
        Call ClearLists
        TB.MoveLast
        TB.MoveFirst
```

```
        Dim i As Long
        ReDim CarOpt(1 To TB.RecordCount) As CAROPTIONS

        i = 1
        Dim x As CAROPTIONS

        While Not TB.EOF
            If TB.Fields("SOLDPRICE") <= 0 Then
                lstID.AddItem TB.Fields("ID")
                lstMotorNum.AddItem TB.Fields("MOTORNUM")
                lstModel.AddItem TB.Fields("MODEL")
                lstType.AddItem TB.Fields("BRAND") & " " &
TB.Fields("TYPE")
                lstColor.AddItem TB.Fields("COLOR")

                lstClass.AddItem
GetCarClass(TB.Fields("CLASS"))

                lstMotor.AddItem TB.Fields("MOTORSIZE") & "
cc"

                CarOpt(i).OP_ABS = TB.Fields("ABSBRAKE")
                CarOpt(i).OP_KMPASSED =
TB.Fields("KMPASSED")
                CarOpt(i).OP_POWERST = TB.Fields("POWERST")
                CarOpt(i).OP_EXTRAS = TB.Fields("EXTRAS")
            End If

            i = i + 1
            TB.MoveNext
            DoEvents
        Wend
    End If
End Sub

Private Sub cmdSearch_Click()
    frmSrchDet.Show vbModal

    If UniqueSQL = "" Then
        Exit Sub
    Else
        Call ClearLists

        Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
        Set TB = DB.OpenRecordset("SELECT ID, MODEL, BRAND,
TYPE, COLOR, CLASS, MOTORSIZE, ABSBRAKE, POWERST, EXTRAS,
MOTORNUM, KMPASSED, SOLDPRICE FROM CARS WHERE " & UniqueSQL
& ";")
```

```
        LastSQL = UniqueSQL

        If TB.EOF = False Then
            TB.MoveLast
            TB.MoveFirst

            Dim i As Long
            i = 1

            While Not TB.EOF
                If TB.Fields("SOLDPRICE") <= 0 Then
                    lstID.AddItem TB.Fields("ID")
                    lstMotorNum.AddItem
TB.Fields("MOTORNUM")
                    lstModel.AddItem TB.Fields("MODEL")
                    lstType.AddItem
IIf(IsNull(TB.Fields("BRAND")) = True, "",
TB.Fields("BRAND")) & " " & TB.Fields("TYPE")
                    lstColor.AddItem TB.Fields("COLOR")

                    lstClass.AddItem
GetCarClass(TB.Fields("CLASS"))

                    lstMotor.AddItem TB.Fields("MOTORSIZE")
& " cc"

                End If

                i = i + 1
                TB.MoveNext
                DoEvents
            Wend
        Else
            MsgBox "Sorry, The Selected Car Was Not
Found!", vbInformation, "Not Found"
        End If
    End If
    UniqueSQL = ""
End Sub

Private Sub cmdSellCar_Click()
    Dim iSel As Integer

    If lstID.ListIndex = -1 Then
        MsgBox "Please Select A Car!", vbInformation, "No
Selection"
        Exit Sub
    End If

    iSel = lstID.List(lstID.ListIndex)
```

```
    Load frmSellCar

    Set TB = DB.OpenRecordset("SELECT * FROM CARS WHERE
ID=" & iSel)
    With frmSellCar
        .txtCarDet(0) = TB.Fields("ID")
        .txtCarDet(1).Text = TB.Fields("MODEL")
        .txtCarDet(7).Text =
IIf(IsNull(TB.Fields("BRAND")), "", TB.Fields("BRAND"))
        .txtCarDet(2).Text = TB.Fields("TYPE")
        .txtCarDet(3).Text = TB.Fields("COLOR")
        .txtCarDet(4).Text = TB.Fields("MOTORSIZE")
        .comSteering.ListIndex = TB.Fields("POWERST")
        .comBrakes.ListIndex = IIf(TB.Fields("ABSBRAKE") =
DB_TRUE, 0, 1)
        .OriginalPrice = TB.Fields("ORIGINALPRICE")
        .txtCarDet(6).Text = TB.Fields("EXTRAS")
        .Label6.Caption = TB.Fields("ORIGINALPRICE")
        .Show vbModal
    End With

    Set TB = DB.OpenRecordset("SELECT ID, MODEL, TYPE,
COLOR, CLASS, MOTORSIZE, ABSBRAKE, POWERST, EXTRAS,
MOTORNUM, KMPASSED, SOLDPRICE FROM CARS;")

    If TB.EOF = False Then
        Call ClearLists
        TB.MoveLast
        TB.MoveFirst

        Dim i As Long
        ReDim CarOpt(1 To TB.RecordCount) As CAROPTIONS

        i = 1
        Dim x As CAROPTIONS

        While Not TB.EOF
            If TB.Fields("SOLDPRICE") <= 0 Then
                lstID.AddItem TB.Fields("ID")
                lstMotorNum.AddItem TB.Fields("MOTORNUM")
                lstModel.AddItem TB.Fields("MODEL")
                lstType.AddItem TB.Fields("TYPE")
                lstColor.AddItem TB.Fields("COLOR")

                lstClass.AddItem
GetCarClass(TB.Fields("CLASS"))

                lstMotor.AddItem TB.Fields("MOTORSIZE") & "
cc"
```

```
                CarOpt(i).OP_ABS = TB.Fields("ABSBRAKE")
                CarOpt(i).OP_KMPASSED =
TB.Fields("KMPASSED")
                CarOpt(i).OP_POWERST = TB.Fields("POWERST")
                CarOpt(i).OP_EXTRAS = TB.Fields("EXTRAS")
            End If

            i = i + 1
            TB.MoveNext
            DoEvents
        Wend
    End If

End Sub

Private Sub Form_Load()
    Call ClearLists

    Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
    Set TB = DB.OpenRecordset("SELECT ID, MODEL, BRAND,
TYPE, COLOR, CLASS, MOTORSIZE, ABSBRAKE, POWERST, EXTRAS,
MOTORNUM, KMPASSED, SOLDPRICE FROM CARS;")


    If TB.EOF = False Then
        TB.MoveLast
        TB.MoveFirst

        Dim i As Long
        ReDim CarOpt(1 To TB.RecordCount) As CAROPTIONS

        i = 1
        Dim x As CAROPTIONS

        While Not TB.EOF
            If TB.Fields("SOLDPRICE") <= 0 Then
                lstID.AddItem TB.Fields("ID")
                lstMotorNum.AddItem TB.Fields("MOTORNUM")
                lstModel.AddItem TB.Fields("MODEL")
                lstType.AddItem
 IIf(IsNull(TB.Fields("BRAND")) = True, "",
 TB.Fields("BRAND")) & " " & TB.Fields("TYPE")
                lstColor.AddItem TB.Fields("COLOR")

                lstClass.AddItem
 GetCarClass(TB.Fields("CLASS"))

                lstMotor.AddItem TB.Fields("MOTORSIZE") & "
 cc"
```

```
                CarOpt(i).OP_ABS = TB.Fields("ABSBRAKE")
                CarOpt(i).OP_KMPASSED =
TB.Fields("KMPASSED")
                CarOpt(i).OP_POWERST = TB.Fields("POWERST")
                CarOpt(i).OP_EXTRAS = TB.Fields("EXTRAS")
            End If

            i = i + 1
            TB.MoveNext
            DoEvents
        Wend
    Else
        MsgBox "Database Is Empty!" & vbNewLine & "No Cars
To View.", vbInformation, "Empty Database"
    End If

    If UserRestType = "User" Then cmdSellCar.Enabled =
False
End Sub

Private Sub lstClass_Click()
    MakeFixed
End Sub

Private Sub lstClass_KeyDown(KeyCode As Integer, Shift As
Integer)
    MakeFixed
End Sub

Private Sub lstClass_KeyPress(KeyAscii As Integer)
    MakeFixed
End Sub

Private Sub lstClass_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstClass_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstClass_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstColor_Click()
```

```
    MakeFixed
End Sub

Private Sub lstColor_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstColor_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstColor_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstID_Click()
    txtABS.Text =
IIf(CarOpt(lstID.List(lstID.ListIndex)).OP_ABS = False,
"Available", "Unavailable")
    txtPOWERST.Text =
GetSteering(CarOpt(lstID.List(lstID.ListIndex)).OP_POWERST)
    txtKMSPASSED.Text =
CarOpt(lstID.List(lstID.ListIndex)).OP_KMPASSED
    txtEXTRAS.Text =
CarOpt(lstID.List(lstID.ListIndex)).OP_EXTRAS
End Sub

Private Sub MakeFixed()
    lstClass.ListIndex = lstID.ListIndex
    lstColor.ListIndex = lstID.ListIndex
    lstModel.ListIndex = lstID.ListIndex
    lstMotor.ListIndex = lstID.ListIndex
    lstMotorNum.ListIndex = lstID.ListIndex
    lstType.ListIndex = lstID.ListIndex
    lblCount.Caption = "Count: " & lstID.ListCount
End Sub

Private Sub lstID_KeyDown(KeyCode As Integer, Shift As
Integer)
    MakeFixed
End Sub

Private Sub lstID_KeyPress(KeyAscii As Integer)
    MakeFixed
End Sub
```

```
Private Sub lstID_KeyUp(KeyCode As Integer, Shift As
Integer)
    MakeFixed
End Sub

Private Sub lstID_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstID_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstID_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstModel_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstModel_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstModel_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstMotor_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstMotor_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstMotor_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub
```

```
Private Sub lstMotorNum_MouseDown(Button As Integer, Shift
As Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstMotorNum_MouseMove(Button As Integer, Shift
As Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstMotorNum_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstType_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstType_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub lstType_MouseUp(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    MakeFixed
End Sub

Private Sub Timer1_Timer()
     MakeFixed
End Sub

Private Sub ClearLists()
    lstID.Clear
    lstModel.Clear
    lstClass.Clear
    lstMotor.Clear
    lstType.Clear
    lstColor.Clear
    lstMotorNum.Clear
End Sub

Option Explicit

Private Sub cmdAdd_Click()
    frmAddCustomer.Show vbModal
    Call RefreshLists
End Sub
```

```
Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdDelete_Click()
    If MsgBox("Are You Sure You Wish To Delete '" &
lstName.List(lstName.ListIndex) & "' ?", vbYesNo +
vbQuestion, "Delete Customer") = vbYes Then
        DB.Execute "DELETE * FROM CUSTOMERS WHERE ID=" &
lstID.List(lstID.ListIndex)
        MsgBox "Customer Deleted!", vbInformation, "Delete
Customer"
        Call RefreshLists
    End If
End Sub

Private Sub cmdEdit_Click()
    If lstID.ListIndex = -1 Then
        MsgBox "No Customer Selected!", vbCritical, "Edit
Customer"
    Else
        Dim iSel As Integer
        iSel = lstID.ListIndex

        Load frmEditCustomer
        Set TB = DB.OpenRecordset("SELECT * FROM CUSTOMERS
WHERE ID=" & lstID.List(iSel))

        With frmEditCustomer
            .txtCusID = TB.Fields("ID")
            .txtCusFName.Text = TB.Fields("FNAME")
            .txtCusLName.Text = TB.Fields("LNAME")
            .txtCusMob.Text = TB.Fields("MOBILE")
            .txtCusAdd.Text = TB.Fields("ADDRESS")
            .txtCusTel.Text = TB.Fields("TEL")

            .Show vbModal
        End With
    End If
    Call RefreshLists
End Sub

Private Sub cmdSearch_Click()
    Dim x$
    x = InputBox("Enter Customer Name:", "Search For
Customer")
    If Not x = "" Then
        Dim i As Integer
        For i = 0 To lstID.ListCount
```

```
            If LCase$(Left$(lstName.List(i), Len(x))) =
LCase(x) Then
                MsgBox "Selected Customer Was Found!",
vbInformation, "Found"
                lstID.ListIndex = i
                Call lstID_Click
                Exit Sub
            End If
            DoEvents
        Next i
    Else
        Exit Sub
    End If
    MsgBox "Selected Customer Was Not Found!",
vbInformation, "Not Found"
    Call RefreshLists
End Sub

Private Sub Form_Load()
    Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
    Set TB = DB.OpenRecordset("SELECT * FROM CUSTOMERS")

    RefreshLists
End Sub

Private Sub RefreshLists()
    lstID.Clear
    lstName.Clear
    lstMobile.Clear
    lstTel.Clear
    lstAdd.Clear

    Set TB = DB.OpenRecordset("SELECT * FROM CUSTOMERS")
    TB.Fields.Refresh

    If TB.EOF = False Then
        While Not TB.EOF
            lstID.AddItem TB.Fields("ID")
            lstName.AddItem TB.Fields("FNAME") &" " " &
TB.Fields("LNAME")
            lstMobile.AddItem TB.Fields("MOBILE")
            lstTel.AddItem TB.Fields("TEL")
            lstAdd.AddItem TB.Fields("ADDRESS")

            TB.MoveNext
            DoEvents
        Wend
    Else
        MsgBox "No Customers Found In Database!",
vbInformation, "Database Empty"
```

```
    End If
End Sub

Private Sub lstAdd_Click()
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstAdd_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstAdd_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstAdd_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstID_Click()
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstID_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub
```

```
Private Sub lstID_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstID_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstMobile_Click()
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstMobile_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstMobile_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstMobile_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_Click()
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
```

```
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstTel_Click()
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstTel_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub

Private Sub lstTel_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub
```

```
Private Sub lstTel_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstAdd.ListIndex = lstID.ListIndex
    lstMobile.ListIndex = lstID.ListIndex
    lstTel.ListIndex = lstID.ListIndex
End Sub


Option Explicit

Private Sub cmdAdd_Click()
    frmAddUser.Show vbModal
    Call RefreshLists
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdDelete_Click()
    If MsgBox("Are You Sure You Wish To Delete '" &
lstName.List(lstName.ListIndex) & "' ?", vbYesNo +
vbQuestion, "Delete User") = vbYes Then
        DB.Execute "DELETE * FROM USERS WHERE ID=" &
lstID.List(lstID.ListIndex)
        MsgBox "User Deleted!", vbInformation, "Delete
User"
        Call RefreshLists
    End If
End Sub

Private Sub cmdEdit_Click()
    If lstID.ListIndex = -1 Then
        MsgBox "No User Selected!", vbCritical, "Edit User"
    Else
        Dim iSel As Integer
        iSel = lstID.ListIndex

        Load frmEditUser
        Set TB = DB.OpenRecordset("SELECT * FROM USERS
WHERE ID=" & lstID.List(iSel))

        With frmEditUser
            .txtID = TB.Fields("ID")
            .txtUsr.Text = TB.Fields("USERNAME")
            '.txtPwd.Text = TB.Fields("PASSWORD")
            '.txtCnfm.Text = TB.Fields("PASSWORD")
```

```
            If TB.Fields("REST") = "Manager" Then
.combType.ListIndex = 0
            If TB.Fields("REST") = "Salesman" Then
.combType.ListIndex = 1
            If TB.Fields("REST") = "User" Then
.combType.ListIndex = 2

            .Show vbModal
        End With
    End If
    Call RefreshLists
End Sub

Private Sub cmdSearch_Click()
    Dim x$
    x = InputBox("Enter User Name:", "Search For User")
    If Not x = "" Then
        Dim i As Integer
        For i = 0 To lstID.ListCount
            If LCase$(Left$(lstName.List(i), Len(x))) =
LCase(x) Then
                MsgBox "Selected User Was Found!",
vbInformation, "Found"
                lstID.ListIndex = i
                Call lstID_Click
                Exit Sub
            End If
            DoEvents
        Next i
    Else
        Exit Sub
    End If
    MsgBox "Selected User Was Not Found!", vbInformation,
"Not Found"
    Call RefreshLists
End Sub

Private Sub Command1_Click()

End Sub

Private Sub Form_Load()
    Set DB = OpenDatabase(App.Path & "\carsdb.mdb")
    Set TB = DB.OpenRecordset("SELECT * FROM USERS")

    RefreshLists
End Sub

Private Sub RefreshLists()
    lstID.Clear
```

```
    lstName.Clear
    lstRest.Clear

    Set TB = DB.OpenRecordset("SELECT * FROM USERS")
    TB.Fields.Refresh

    If TB.EOF = False Then
        While Not TB.EOF
            lstID.AddItem TB.Fields("ID")
            lstName.AddItem TB.Fields("USERNAME")
            lstRest.AddItem TB.Fields("REST")

            TB.MoveNext
            DoEvents
        Wend
    Else
        MsgBox "No Users Found In Database!",
vbInformation, "Database Empty"
    End If
End Sub

Private Sub lstID_Click()
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstID_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstID_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstID_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstRest_Click()
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub
```

```
Private Sub lstRest_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstRest_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstRest_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_Click()
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_KeyDown(KeyCode As Integer, Shift As
Integer)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_MouseDown(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub

Private Sub lstName_MouseMove(Button As Integer, Shift As
Integer, x As Single, Y As Single)
    lstName.ListIndex = lstID.ListIndex
    lstRest.ListIndex = lstID.ListIndex
End Sub
```