

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Car Care and Repair Center

Graduation Project

COM- 400

Student: Alper Karkuş(20010680)

Supervisor: Mr. Ümit İLHAN

Lefkoşa – 2006

ACKNOWLEDGMENTS

"It is my pleasure to take this opportunity to emphasize my greate gratitude to man individuals who have given me a lot of supports during my five-year Undergraduation program in the Near East University.

First, I would like to thank my supervisor Mr.Ümit İlhan for his ivaluable advice and belief in my work and my self over the course of this Graduation Project..

Second, and thank my dearest parents who encouraged me to continue beyond my undergraduate studies, to my father who proceeded before me and to my mother who encouraged me along the way.

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

ABSTRACT

Data, gathered around us as a collection of facts, is of no use unless it is organized and represented in some meaningful form. Data represented in some meaningful form like, tables, charts, or graphs become information, which can be easily processed. The collection of data, usually referred to as the database, contains information about one particular enterprise. These days database are used by a variety of users and organizations, which are important tools in data processing DBMS, are designed to manage large bodies of database information. The program provides, manage and take hold of business transactions' record, customer records, car records, stock records, A scheduled user manual prepared for helping the users to select a suitable action.

This project has as its goal to develop software, processing information about activities of car care repair center. Software developed in this project contains both customer information, user information and information associated with care, sales of parts and purchase of car parts. I wish to develop this software for processing all activities of the company.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	i
ABSTRACTION	ii
TABLE OF CONTENTS	iii
CHAPTER ONE INTRODUCTION TO VISUAL BASIC.NET	1
1.1 Introduction to Programming	2
1.2 Introduction to Programming and VB .NET	2
1.3 Computer Operations	3
1.3.1 Input Data	3
1.3.2 Store data in memory	4
1.3.3 Perform arithmetic on data	4
1.3.4 Compare two values and select one of two alternative actions	4
1.3.5 Repeat a group of actions any number of times	4
1.3.6 Output the results of processing	4
1.4 Programs and Programming	5
1.5 Programming Languages	5
1.6 Programming in Windows	7
1.7 Windows in XP	8
1.8 THE VB.NET Language	9
1.9.1 Start Visual Studio .NET from the Windows Desktop	10
1.9.2 Visual Basic .NET Start Page Start page	10
1.9.3 New Project Dialog Window	11
1.9.4 VB .NET IDE Windows Workspace	11
1.9.5 VB .NET Applications	12
1.9.6 Toolbox	12
1.9.7 Solution Explorer Window	13
1.9.8 Properties Window -- Set Properties	14
1.9.9 Add Event Procedure	14
1.9.10 Test or Run Application	15
1.9.11 Saving and Recalling a Project	15
1.9.12 Example -- Step 1: Adding Controls	15
1.9.13 Example -- Step 2: Adding Properties	16
1.9.14 Example -- Step 3: Adding Events	16
1.9.15 Running an Application	16
1.9.16 Adding an Event Procedure	16
1.9.17 The MessageBox.Show Method	17
1.9.18 Message Box with Caption	18
1.9.19 Message Box with Buttons	18
1.9.20 Message Box with Default Button	19
1.9.21 Setting the Initial Object Properties	19
1.9.22 Looking at the Focus and Tab Sequence	19
1.9.22 Looking at the Focus and Tab Sequence	20
1.9.24 Statement	20
1.9.25 Help Facility	20
1.10 Programming in VB.NET	21

1.10.1 Step One: Define Problem	22
1.11.2 Step Two: Create Interface	23
1.10.3 Step Three: Develop Logic for Action Objects	23
1.10.4 Step Four: Write and Test Code for Action Objects	25
1.10.5 Step Five: Test Overall Project	27
1.10.6 Step Six: Document Your Project in Writing	27
CHAPTER TWO WHAT IS AN SQL SERVER?	28
2.1 INTRODUCTION TO SQL SERVER	28
2.2 Relational Database Management System	29
2.3 Data Storage Models	29
2.3.1 OLTP Databases	29
2.3.2 OLAP Databases	29
2.4 Client Applications	29
2.4.1 Transact-SQL	29
2.4.2 XML	30
2.4.3 MDX	30
2.4.5 OLE DB and ODBC APIs	30
2.4.6 ActiveX Data Objects and ActiveX Data Objects (Multidimensional)	30
2.4.7 English Query	30
2.5 CLIENT SERVER COMPONENT	30
2.5.1 Client-Server Architecture	31
2.5.2 Client Components	31
2.5.2.1 Client Application	31
2.5.2.2 Database API	31
2.5.2.3 Client Net-Library	31
2.5.3 Server Components	32
2.5.3.1 Server Net-Libraries	32
2.5.3.2 Open Data Services	32
2.5.3.3 Relational Engine	32
2.5.3.4 Storage Engine	32
2.6 Client-Server Communication Process	33
2.7 SQL SERVER SERVICES	33
2.7.1 Four SQL Server Services	33
2.7.1.1 MSSQLServer Service	33
2.7.1.2 SQLServerAgent Service	33
2.7.1.3 Microsoft Distributed Transaction Coordinator	34
2.7.1.4 Microsoft Search	34
2.7.2 Multiple Instances of SQL Server	34
2.8 SQL SERVER INTEGRATION	34
2.8.1 Client Components	34
2.8.2 Server Components	35
2.8.3 Internet Browsers and Third-Party Applications	36
2.8.3.1 Microsoft Internet Information Services	36
2.8.3.2 Integrating SQL Server with Other Microsoft Server Applications	36
2.9 SQL SERVER DATABASES	36
2.9.1 Types of Databases	36

2.9.2 Database Objects	36
2.9.2.1 Referring to SQL Server Objects	36
2.9.3 Fully Qualified Names	36
2.9.4 Partially Specified Names	37
2.9.5 System Tables	38
2.9.5.1 System Tables	38
2.9.5.2 Database Catalog	38
2.9.5.3 System Catalog	38
2.9.6 Metadata Retrieval	38
2.9.7 System Stored Procedures	38
2.9.8 System and Metadata Functions	38
2.9.9 Information Schema Views	39
2.10 SQL SERVER SECURITY	39
2.10.1 Login Authentication	39
2.10.2 Windows Authentication	39
2.10.3 SQL Server Authentication	39
2.10.4 Authentication Mode	40
2.10.5 Windows Authentication Mode	40
2.10.6 Mixed Mode	40
2.11 Database User Accounts and Roles	40
2.11.1 Database User Accounts	40
2.11.2 Roles	41
2.11.3 Types of Roles	41
2.11.3.1 Fixed Server Role	41
2.11.3.2 Fixed Database Roles	41
2.11.3.3 User-defined Database Roles	41
2.11.4 Permission Validation	41
2.12 WORKING WITH SQL SERVER	42
2.12.1 Administering a SQL Server Database	42
2.12.2 Common Administrative Tasks	42
2.12.3 SQL Server Enterprise Manager	42
2.12.4 SQL Server Administration Tools and Wizards	42
2.12.5 SQL Server Command Prompt Management Tools	42
2.12.6 SQL Server Help and SQL Server Books Online	43
2.12.7 Implementing a SQL Server Database	43
2.12.8 Selecting an Application Architecture for SQL Server	43
2.12.9 Software Architecture	43
2.12.10 Architectural Design	44
2.12.10.1 Intelligent Server (2-Tier)	44
2.12.10.2 Intelligent Client (2-Tier)	44
2.12.10.3 N-Tier	44
2.12.10.4 Internet	44
2.12.11 Designing Applications Using Database APIs	45
2.13 OLE DB	45
2.14 ADO	45
2.15 OVERVIEW OF PROGRAMMING SQL SERVER	46
2.15.1 Designing Enterprise Application Architecture	46
2.15.2 Identifying Logical Layers	46

2.15.3 Data Presentation Layer	46
2.15.4 Application Logic Layer	46
2.15.5 Data Services Layer	47
2.15.6 Designing Physical Layers	47
2.15.6.1 Using a Two-Tier Model	47
2.15.6.2 Using a Multi-Tier Model	48
2.15.7 Accessing Data	49
2.15.8 Accessing Data	49
2.15.8.1 Providing Universal Data Access	49
2.16 SQL SERVER PROGRAMMING TOOLS	49
2.16.1 SQL Query Analyzer	50
2.16.2 osql Utility	51
2.17 THE TRANSACTION-SQL PROGRAMMING LANGUAGE	52
2.17.1 Elements of Transact-SQL	52
2.17.2 Data Control Language Statements	52
2.17.3 Data Definition Language Statements	53
2.17.4 Data Manipulation Language Statements	53
2.17.5 SQL Server Object Names	54
2.17.6 SQL Server Object Names	55
2.18 ADDITIONAL LANGUAGE ELEMENTS	55
2.18.1 Local Variables	56
2.18.2 Operators	56
2.18.3 Types of Operators	56
2.18.3.1 Arithmetic	56
2.18.3.2 Comparison	56
2.18.3.3 String Concatenation	56
2.18.3.4 Logical	57
2.18.4 Operator Precedence Levels	57
2.18.5 Functions	57
2.18.5.1 Aggregate Functions	57
2.18.5.2 Scalar Functions	58
2.18.5.3 Rowset Functions	58
2.18.5.4 Convert Functions	58
2.18.5.5 Date Functions	58
2.18.5.6 User Functions	59
2.18.5.7 Column Property Functions	59
2.18.6 Control of Flow Language Elements	59
2.18.7 Statement Level	60
2.18.8 Row Level	61
2.18.9 Comments	62
2.18.9.1 In-Line Comments	62
2.18.9.2 Block Comments	63
2.19 WAYS TO EXECUTE TRANSACTION-SQL STATEMENTS	63
2.19.1 Dynamically Constructing Statements	64
2.19.2 Define a Batch by Using the GO Statement	65
2.19.3 How SQL Server Processes Batches	65
2.19.4 You Cannot Combine Some Statements in a Batch	66
2.19.4.1 Using Scripts	

INTRODUCTION

Nowadays the technology is developed a lot and started to use by anyone in the world no matter who he/she is. Because of the technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.

The computer program is also became a part of our lives. The people operate with programming language in everywhere. My project is Car Care Repair Center Program. This program is used to keep the information about the car and customer and stock information.

In my project the main point is making the user's job easy. It lets to the manager to manage the personnel, customer information easily. Also I coded this program to use in the second class care repair centers. The mean of these the center is local. Not consist of internet contact.

This program has many useful abilities. These are; defination of all details of car, if you wish, without saving information, care and repair , no the net total you are able to discount, taking parts from outside for repairing car and saving this information.

Also this program can produce some calculated reports; daily report, according to dates report and according to number plate reports,

The Project consist of introduction, three chapters and conclusion.

Chapter one describes the vb.net and introduction to programming, some examples about how to code basic processes in vb.net

Chapter two describes introducing to Microsoft SQL SERVER, understanding of Microsoft SQL SERVER and objects of Microsoft SQL SERVER

Chapter three includes the screenshots and user manual of Care Repair System. These screenshots are related to codes of the program which is explained in the appendix.

CHAPTER ONE

INTRODUCTION TO VISUAL BASIC.NET

1.1 Introduction to Programming

Many organizations are finding that in order to survive, they must be able to collect and process data efficiently and make the resulting information on their operations available to their employees. Successful organizations have found that the key to making this information available is having an effective information system that will carry out these operations. An **information system** is *the combination of technology (computers and people that enables an organization to collect data, store them, and transform them into information.*

To understand the concept of an information system fully, you need to understand the difference between data and information. **Data** are raw facts that are collected and stored by the information system. Data can be in the form of numbers, letters of the alphabet, images, sound clips, or even video clips. You are undoubtedly very familiar with many types of data, including names, dates, prices, and credit card numbers. By themselves, data are not very meaningful; however, when data are converted by the information system into **information**, the end result is meaningful. Once again, you are familiar with many forms of information, including written reports, lists, tables, and graphs. Information is what organizational employees use in their work.

To convert or process data into information electronically, software must direct the operations of the computer's operations. **Software** is composed of one or more lists of instructions called **programs**, and the process of creating these lists of instructions is termed **programming**. While computer hardware can be mass-produced on assembly lines like other consumer goods, software must be developed through the logical and creative capabilities of humans. Individuals or groups of individuals working together must develop the instructions that direct the operations of every computer in use today.

1.2 Introduction to Programming and VB .NET

The same is true whether the instructions are for the computer that controls your car's fuel system, the computer that controls the space shuttle, or the computer that prints the checks for the business at which you work.

While a great deal of programming work goes on at large software firms like Microsoft or Adobe Systems, much more programming is done at companies that produce nonsoftware goods and services. While you may think that these companies could buy off the shelf software like word processors or spreadsheets to run their business, in most cases companies must develop their own software to meet their particular needs. In fact, it has been said that the "software needed to be competitively different is generally not available from off-the-shelf packages" and that "...building...systems for unique [competitive] capability is often the single most important activity for an...organization.". This means that no matter how good off-the-shelf software becomes, there is always going to be demand for programmers to work in businesses and not-for-profit organizations. In fact, the demand for information systems employees is accelerating and the future is very bright for persons trained in this field.

Programming is actually part of a much larger process known as **systems development**.

This process involves a large scale effort to either create an entirely new information system or to update (maintain) an existing information system. In either case, systems development involves four primary steps: planning, analysis, design, and implementation. In the planning stage, it is decided what must be done to solve a problem or meet a need—create a new system, update an old one, or even, purchase a system from an outside source. Once it has been decided what must be done, the next step is to analyze the system that will be created. This may involve analyzing an existing system or analyzing the system that must be created. Once the analysis step is completed, the next step is to design the new or updated system. This design must be complete and detailed and leave nothing to chance or guesswork. Once the design is completed, the system can be implemented. It is in the implementation step that programming comes in. Programmers work with the results of the design step to create a series of computer programs that, together, will work as the needed information system. In many cases, the programmers will know little or nothing about the overall problem and must depend completely on the results of the design step. However, without the programming process, the information system will never be built or updated. Visual Basic .NET (VB .NET) and the entire Microsoft .NET framework is aimed at making this process possible. Given that programming is such an important part of building and maintaining information systems for organizations of all sizes, it is easy to see why individuals interested in working in the field of information systems must have some knowledge of computer programming. This book is written with the purpose of helping you become capable of writing computer programs that will solve business-related problems.

1.3 Computer Operations

Before we start our discussion of creating computer programs, it is useful to understand the six operations that all computers can carry out to process data into information. Understanding these operations will help you when you start writing programs. Martin, James, *Cybercorp: The New Business Revolution*, New York: AMACOM Books

These operations are the same regardless of whether we are discussing multi-user mainframe computers that handle large-scale processing, such as preparing the end-of term grade rolls or processing the university payroll, or small personal computers that are used today by a large proportion of office workers in the United States and other developed countries. The six operations that a computer can perform are:

1. Input data
2. Store data in internal memory
3. Perform arithmetic on data
4. Compare two values and select one of two alternative actions
5. Repeat a group of actions any number of times
6. Output the results of processing

Let's now discuss each of these operations in a little more detail.

1.3.1 Input Data: For a computer to be able to transform data into information, it must first be able to accept input of the data. Data are typically input from a keyboard or mouse, but they can also come from other sources such as a barcode reader like those used at checkout terminals. Input can also come from some type of sensor or from a data file on computer disk. For example, with a word processor, the letters of the alphabet, numbers, and punctuation symbols are the data that are processed by the computer. New

documents are created by entering data from the keyboard while existing documents are loaded from your hard drive or floppy disk.

1.3.2 Store data in memory: Once data have been input, they are stored in internal memory.

Each memory location holding a piece of data is assigned a name, which is used by the instructions to perform the processing. Since the values in a memory location can change as the process occurs, the memory locations are called **variables**. The current balance in your checking account would typically be stored in a single memory location

and be identified by a variable name. The instructions for processing this data are also stored in memory. In the earliest days of computing, the instructions (program) were not stored in memory and had to be entered one at a time to process the data. When the *stored program* concept was developed by John von Neumann, it was a tremendous breakthrough. With a stored program, the instructions can be executed as fast as they can be retrieved from memory to convert the data into usable information.

1.3.3 Perform arithmetic on data: Once the data and instructions have been input and stored, arithmetic operations can be performed on the variables representing the data to process them into information. This includes addition, subtraction, multiplication, division, and raising to a power. The processing chip of the computer carries out these operations by retrieving the data from memory and then performing the processing based on instructions from the programmer. You may ask how a word processor or computer game works if all the computer can do is perform arithmetic. The answer is that everything in a computer—numbers, letters, graphics, and so on—is represented by numbers, and all processing is handled through some type of arithmetic operation.

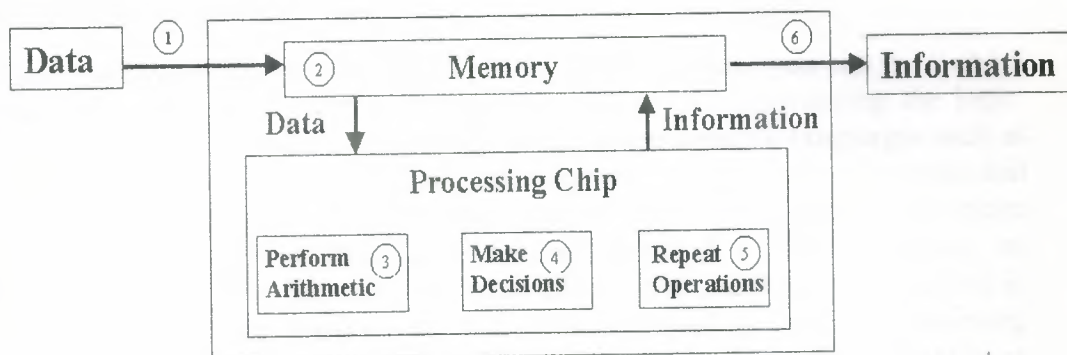
1.3.4 Compare two values and select one of two alternative actions: To do anything other than the simplest processing, a computer must be able to choose between two sets of instructions to execute. It does this by comparing the contents of two memory locations and, based on the result of that comparison, executing one of two groups of instructions. For example, when you carry out the spell-checking operation, the computer is checking each word to determine if it matches a word in the computer's dictionary. Based on the result of this comparison, the word is accepted or flagged for you to consider changing.

1.3.5 Repeat a group of actions any number of times: While you *could* carry out all of the above operations with a typewriter or handheld calculator, repeating actions is something the computer does better than any person or any other type of machine. Because a computer never tires or becomes bored, it can be instructed to repeat some action as many times as needed without fear of an error occurring from the constant repetition. The capability of a computer to repeat an operation is what most clearly sets it apart from all other machines. The spell-checking operation mentioned above is an example of a repeated action: The program repeatedly checks words until it comes to the end of the document.

1.3.6 Output the results of processing: Once the processing has been completed and the required information generated, to be of any use the information must be output. Output of processed information can take many forms: displayed on a monitor, printed on paper, stored on disk, as instructions to a machine, and so on. Output is accomplished by retrieving information from a memory location and sending it to the

output device. For example, when you complete your work with a word processor, the resulting information is displayed on your monitor and you probably will also print it for distribution to others.

These six operations are depicted in Figure 1-1, where each operation is numbered.



1.4 Programs and Programming

To carry out any of the six operations just discussed, you must be able to provide instructions to the computer in the form of a program. The most important thing about programming is that it is a form of *problem solving*, and the objective is to develop the step-by-step process—the logic—that will solve the problem. Step-by-step logic of this type is referred to as an algorithm. You have worked with algorithms before; a set of directions to a party is an algorithm, as is a recipe to make spaghetti sauce or to bake a cake. For a computer program, you must develop a set of instructions for solving a problem using *only* the six operations of a computer. This is the most difficult part of programming. Many times a program fails to work because the programmer attempts to write the program before developing the correct algorithm for solving the problem. Only after you have developed the logic of the solution can you consider actually writing the instructions for the computer.

Control Structures While it can be quite daunting to create the logic to solve a problem, remember that all computer programs can be created with only three types of logic or, as they are known in programming, **control structures**. The three control structures are sequence, decision, and repetition. The **sequence control structure** includes the input, storage, arithmetic, and output computer operations discussed earlier. It is so called because all four of these operations can be performed without any need to make a decision or repeat an operation. At its simplest, *sequence* means one program instruction follows another in order. Of course, it is up to the programmer to determine the proper sequence order for the instructions. The **decision control structure** is the same as the decision-making computer operation discussed earlier. It enables the programmer to control the flow of operations by having the user or data determine which operation is to be performed next.

Finally, the **repetition control structure** is used to repeat one or more operations. The number of repetitions depends on the user or the data, but the programmer must include a way to terminate the repetition process.

All algorithms are created using the six operations of a computer within combinations of these three control structures. Once you learn how to create the logic

for these three control structures, you will find that writing meaningful and useful programs is a matter of combining the structures to create more complex logic.

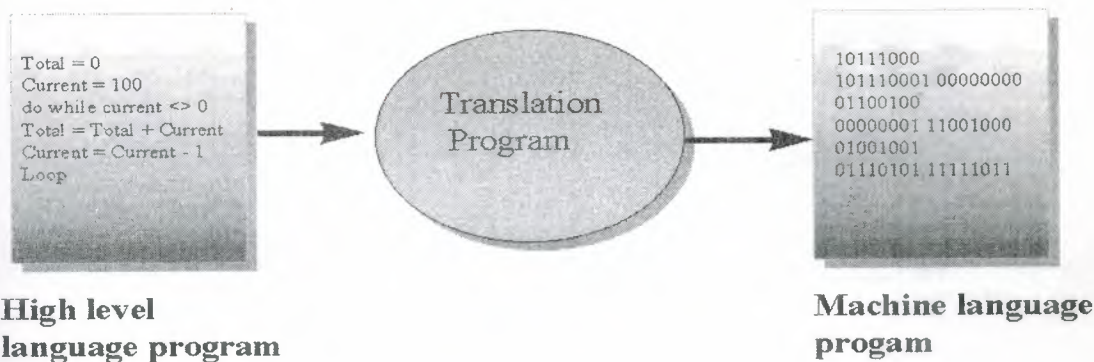
1.5 Programming Languages

Once you have developed the logic for solving the problem, you can think about writing the actual instructions that the computer will use in implementing the logic. Computer programs must be written in one of various **programming languages** such as VB .NET. These languages use a restricted vocabulary and a very structured syntax that the computer can understand. While a great deal of research is ongoing to create computers that can accept instructions using conversational English, currently no computers meet this criterion. So, until computers like C3-PO and R2D2, popularized in the *Star Wars* movies, are created, we are stuck with using these programming languages. Within the computer, the data and instructions are represented in the binary number system as a series of zeros and ones. This form of representation is used because the computer's only two electrical states—on and off—correspond to 1 and 0. Using a string of transistors that act as switches, the computer can represent a number, character, or instruction as a series of on-off states. All processing is carried out in the binary number system. For example, the computer carries out all arithmetic in binary instead of in the decimal number system that humans use. The binary form of the instructions is called **machine language**, since this is the language that computers use to carry out their operations. An example of the machine language statements necessary to sum the digits 1 to 100 for a computer using an Intel CPU.

1.5 Machine language program

```
10111000 00000000 00000000 Set Total Value to 0
10111001 00000000 01100100 Set Current Value to 100
00000001 11001000 Add Current value to Total Value
01001001 Subtract 1 from Current Value
01110101 11111011 If Current value is not 0, repeat
```

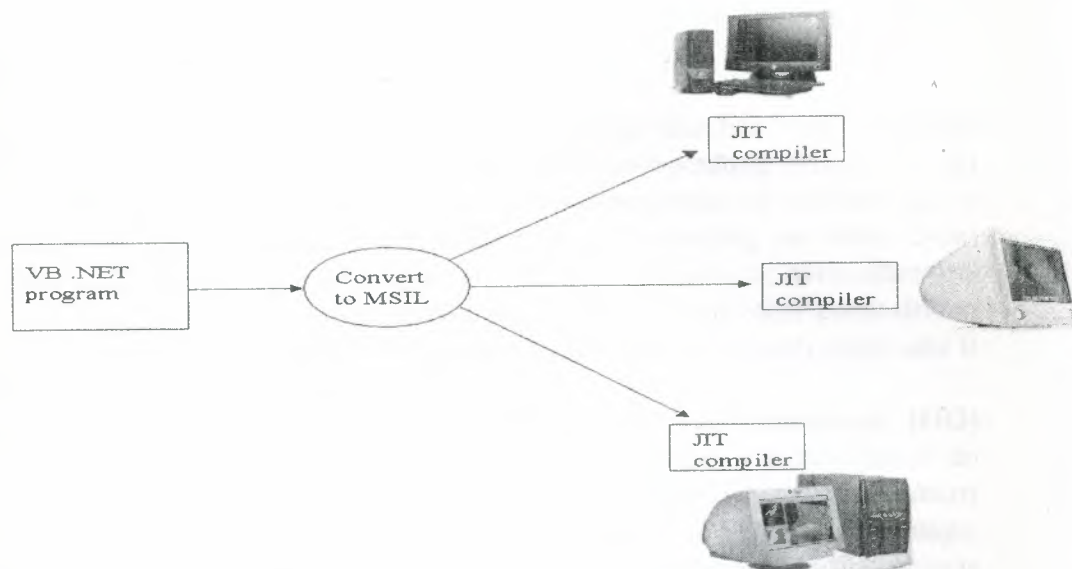
Programming the very first computers, which had to be done in binary, was very difficult and time-consuming. Now, we have English-like programming languages, like VB .NET which are referred to as **high-level languages** because they are close to the level of the human programmer rather than being close to the level of the machine. Before the statements in a high-level program can be used to direct the actions of a computer, they must be translated into machine language. Files on a Windows-based computer with an .exe file extension are machine-language programs that have been translated from some high-level language. They can be executed with no translation because they are already in a binary form. Until recently, this was a direct translation from high-level language to machine language by a software program known as a compiler or interpreter, depending on whether the code was translated as a unit or line byline as shown in Figure



The problem with this approach is that different types of computers have different machine languages so a program would have to be translated differently for an Apple computer than for a Windows computer. To make it possible for the same program to run on all types of machine, the concept of the **just-in-time (JIT) compiler** was developed. With this approach, the high-level program is translated or compiled into an intermediate form that is machine-independent. The two approaches to this use of a just-in-time compiler are Java from Sun Microsystems and the .NET framework from Microsoft of which VB .NET is a part. In the case of Java, the intermediate form is called **bytecode** and for the .NET framework, it is called **Microsoft Intermediate Language (MSIL)**. Once converted a VB .NET program is compiled into MSIL, the just-in-time compiler on any computer can convert it into machine language for that particular machine. This process is shown in Figure 1-4 for MSIL. Where the Java approach only works for programs written in Java, the .NET framework approach works for all languages that have been revised to work under that framework. At this time, these include VB .NET, C# (pronounced "c-sharp") .NET, and C++ (pronounced "c plus plus") .NET. This means that if you are using one of these language, it can be compiled in MSIL and combined with other programs in the MSIL and then sent to the JIT compiler, which for the .NET framework is called the *Common Language Runtime (CLR)*.

1.6 Programming in Windows

As you are probably aware, most personal computers today run some form of the Microsoft Windows operating system such as Windows 95, Windows 98, Windows ME, Windows 2000, or Windows XP. With Windows being the primary operating system for personal computers, learning to program in the Windows environment has become a critical skill for anybody interested in working in information systems. To program in Windows, you first need to understand a little about how Windows works.



To understand the workings of Windows, you need to understand three key concepts:

windows, events, and messages. A **window** is any rectangular region on the screen with its own boundaries. All components run in their own windows. For example when you use your word processor, a document window displays the text you are entering and editing. When you retrieve a file, you do this from a dialog box that is a window. Similarly, when an error message is displayed, this is done in a window.

Finally, the menu bar and all of the icons or buttons on the toolbar across the top or side of your screen are also windows.

1.7 Windows in XP

When an event occurs, the corresponding window sends a **message** to the operating system, which processes the message and then broadcasts it to other windows.

When they receive a message, the other windows take actions based on their own set of instructions. Programming in Windows requires that you learn how to work with windows, events, and messages. For this reason, programming in Windows is usually termed **event-driven programming**, because all actions are driven by events. While this may sound complicated, languages like VB .NET make it easier to create Windows-based applications that work with Windows by providing you with the necessary tools. Event-driven programming is quite different from *traditional* approaches to programming where the program itself controls the actions that will take place and the order in which those actions will occur. With *traditional* programs, execution of the program starts with the first instruction and continues through the remaining instructions, making decisions as to which instructions will be executed depending on the data that are input. The main program may use smaller subprograms to handle parts of the processing. This type of programming is referred to as **procedural programming**, and it works very well for such activities as processing a large number of grades at the end of the term or printing payroll checks at the end of the pay period. However, with the move toward widespread use of GUI, the trend is toward using eventdriven programming.

1.8 THE VB.NET Language

As discussed above, VB .NET is a computer language that has been developed to help you create programs that will work with the Windows operating system. It is an event-driven language that does not follow a predefined sequence of instructions; it responds to events to execute different sets of instructions depending on which event occurs. The order in which events—such as mouse clicks, keystrokes, or even other sets of instructions—occur controls the order of events in VB .NET and other event-driven languages. For that reason, an event-driven program can execute differently each time it is run, depending on what events occur.

In addition to being event-driven, VB .NET is an **object-oriented (OO) language**; that is, it uses software objects which can respond to events. This is an important improvement over previous versions of Visual Basic which were *almost* object-oriented since they failed to have all of the characteristics of a true OO language. What distinguishes object-oriented programming from earlier languages is that objects combine programming instructions or **code** with data. Previous attempts to structure programs in such a way that large problems could be broken down into smaller problems separated the code from the data. The problem with this approach is that if the data changes, then the code may not work with the new data. With object-oriented programming, the combination of code and data avoids this problem. For example, instead of writing code to deal with customers and then using this code with different customer data for each customer, we combine the code and data into an **object** for each customer. The objects for multiple customers are very similar with the exception of the data component, so you can use them in similar ways.

I will just introduce you to some of the concepts of OO. First, consider the fact that each of the windows discussed above as a part of the Windows operating system is an object, as are a wide variety of other shapes, including buttons, click boxes, menus, and so on. There are also many objects in Windows that are unseen since they are pure computer code, but have the same characteristics as visual objects. The beauty of VB .NET is that, unlike many other OO languages, you do not have to know how to create objects to use them. VB .NET automatically creates for you, the programmer, new instances of a many objects from a wide variety of built-in templates. To understand Object-oriented programming, we need to understand a number of concepts and terminology. First, in order to create an object, you must first create a **class**, that is, a *template with data and procedures from which objects are created*. One way of looking at a class is to think of it as the *cookie cutter* and the actual object as the resulting *cookie*.² All of the actual work in creating an object is accomplished in creating the class; an object is created by defining it to be an **instance** of a class. Objects have two key concepts: properties and methods. **Properties** of objects are simply the attributes associated with the object such as their name, color, and so on. **Methods** are a set of predefined activities that an object can carry out.

Three key characteristics of OO programming are encapsulation, inheritance, and polymorphism. **Encapsulation** refers to a key concept: It should never be possible to work with variables in an object directly; they must be addressed through the object's properties and methods. This implies a *black-box* view of an object, in which the programmer does not need to know what is going on inside the object, but only needs to know how to work with the object's methods and properties.

The second key concept in object-oriented programming, **inheritance**, refers to the capability to create *child* classes that descend from a *parent* class. This capability

makes it easier to build a new child classes by having them inherit properties and methods from a parent class.

Finally, **polymorphism** is related to inheritance in that a child class can inherit all of the characteristics and capabilities of the parent class but then add or modify some of them so the child class is different from the parent class. It is important to note that the instructions for a method are already a part of VB .NET, but the programmer must write the instructions to tell the object how to respond to an event.

Working with VB .NET involves combining objects with the instructions on how each object should respond to a given event. For example, you might have a button for which the instructions are to display a message; instructions for another button might be to exit the program or, as it is called in VB .NET, the *solution*. These instructions are referred to as the code for the program. The code for VB .NET is written in a much-updated form of one of the oldest computer languages around—Basic, which was first used in 1960. The version of Basic used in VB .NET has been improved in many ways, but it retains one of the key advantages of the original language compared to other languages: It is very easy to use and understand.

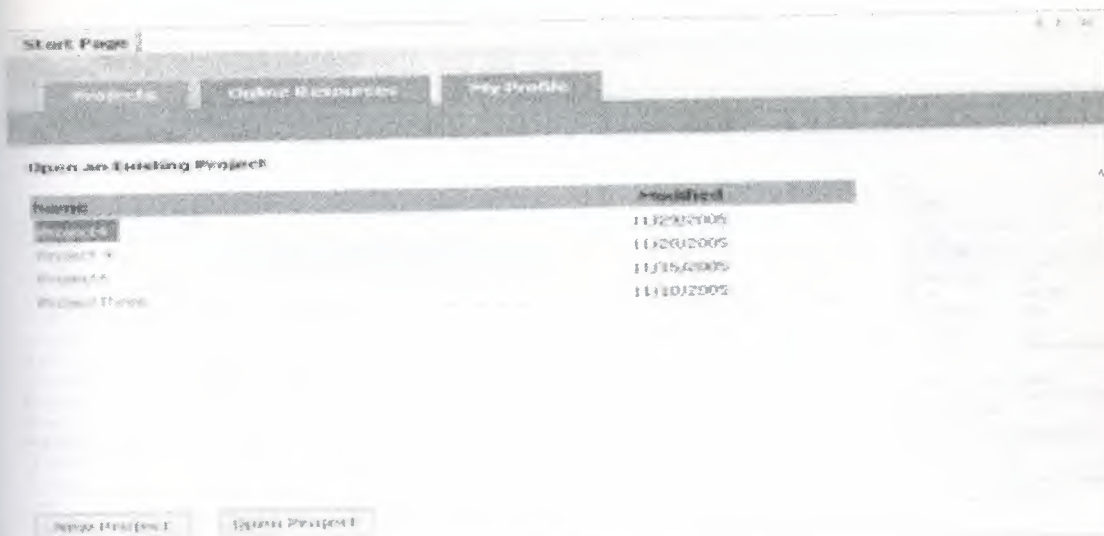
1.9.1 Start Visual Studio .NET from the Windows Desktop



1.9.2 Visual Basic .NET Start Page Start page

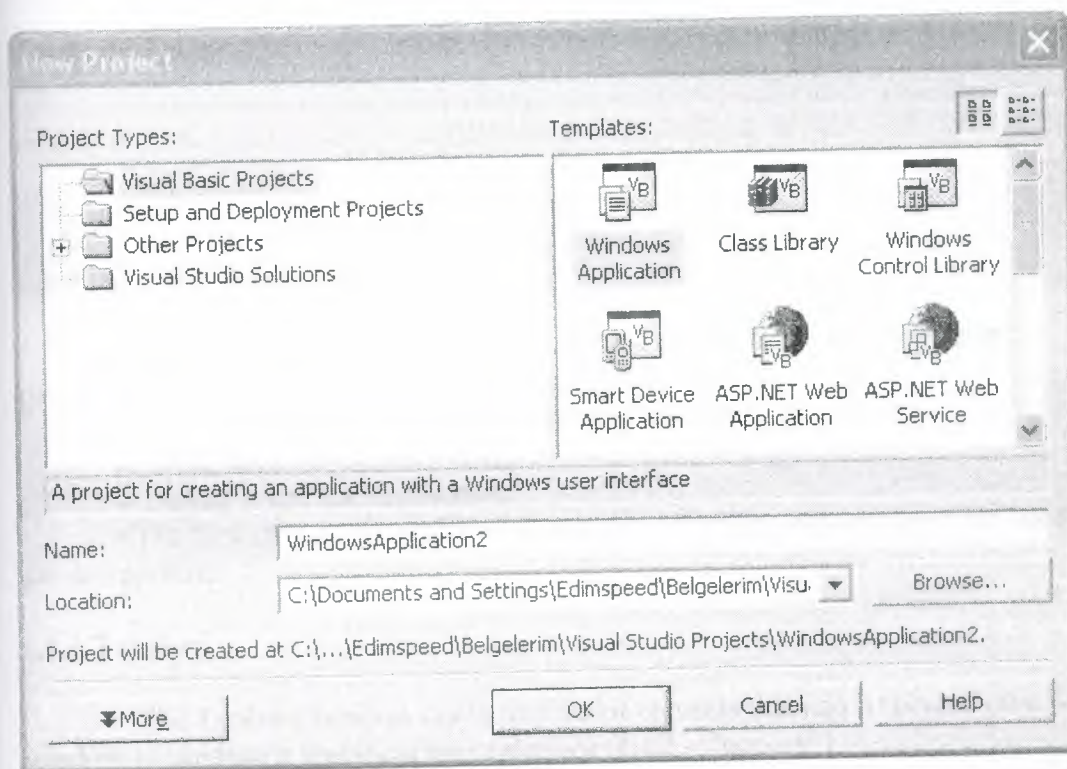
Allows the programmer to

- open recent projects,
- open any previously saved project, and
- create a new project.



1.9.3 New Project Dialog Window

Clicking the New project button on the Start Page to open the New Project



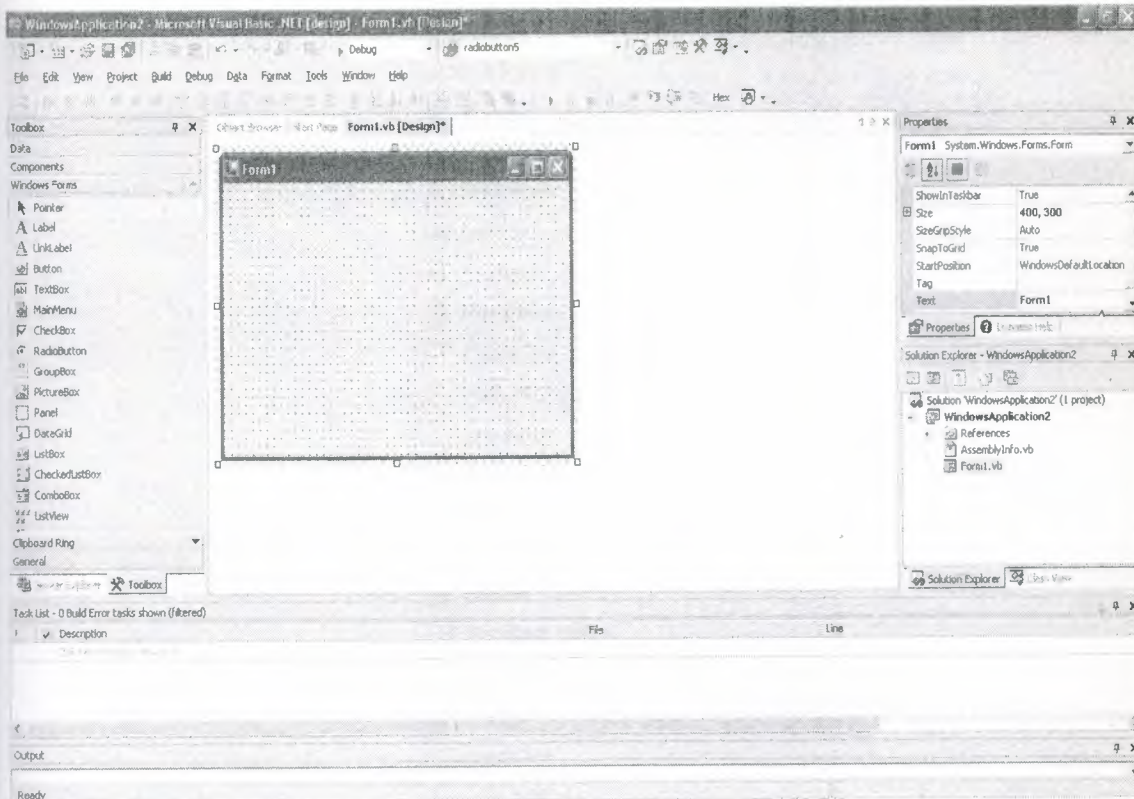
1.9.4 VB .NET IDE Windows Workspace

- When a new project is created, the GUI designer component of the IDE is Displayed.
- The IDE also has two other components: a code editor and a debugger.
- The IDE offers all of the capabilities of a Windows-based application, such as

the

ability to resize and close any of the child windows, as well as the overall

parent window.



1.9.5 VB .NET Applications

- The steps that are required for creating a Visual Basic application are:

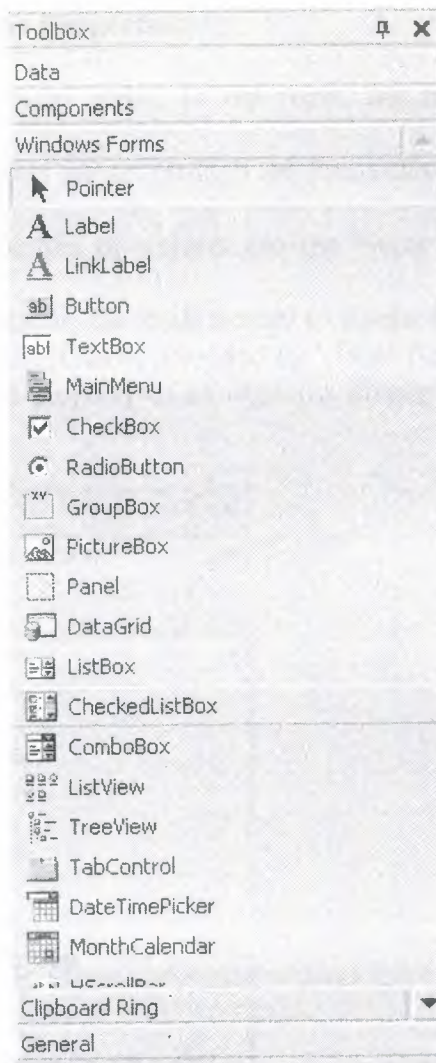
1. Create the graphical user interface (GUI).
2. Set the properties of each object on the interface.
3. Write the code or add events.
4. Debug -- test the application.

• The first step, creating the GUI, consists of adding objects from the Toolbox to the design form.

1.9.6 Toolbox

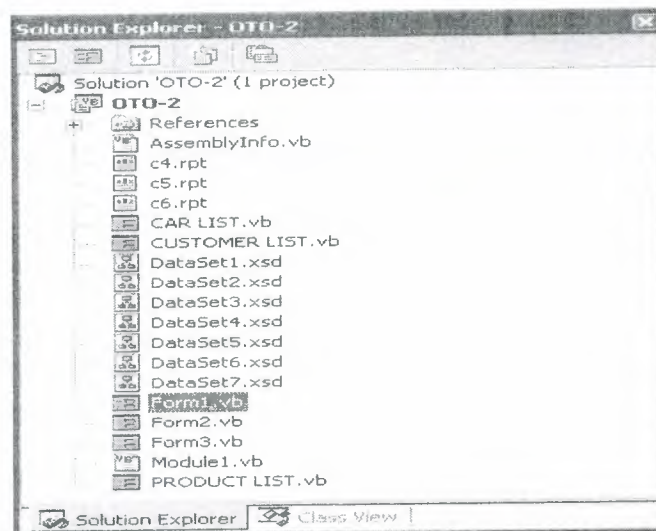
• The Toolbox window contains a set of controls that can be placed on a Form window to produce a graphical user interface (GUI – “goo-ey”).

• The toolbox can be opened by choosing the command Toolbox in the View menu.



1.9.7 Solution Explorer Window

- Solution Explorer Window provides an easy access to different application files including files contains forms and codes.



1.9.8 Properties Window -- Set Properties

- Once objects have been added to the form, the next step is setting the properties of the objects.
- The properties of objects are set through the Properties window or code (inside the program).
- Two important properties of objects are the Name property and the Text property.
- The Name property allows the programmer to assign a descriptive name to an object, rather than using the default name provided by Visual Basic for that object.
- The value of the Text property of an object is displayed to the user when the application is running.



1.9.9 Add Event Procedure

- An event procedure is a procedure or event handler executed when that event occurs.
- The first line of a procedure is a header line.
- A header line begins with the optional keyword Private and must contain the keyword Sub, the name of the procedure, and a set of parentheses.
- The last line of each procedure consists of the keywords End Sub.
- All statements from the header line to and including the End Sub statement are collectively referred to as the procedure's body.

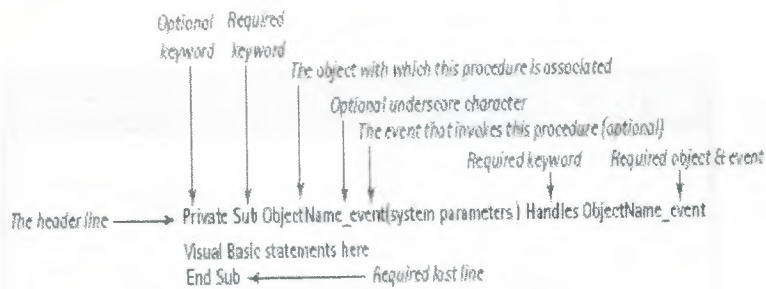


Figure 2-19 The Structure of an Event Procedure

- The first and last lines of a procedure, consisting of the header line and the End Sub statement, are referred to as the procedure's template.

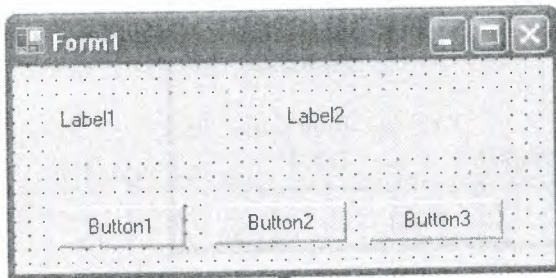
1.9.10 Test or Run Application

- You can run your program at any time during program development:
 - Select the Debug Menu and click Start or
 - Press the F5 function key or
 - Use the hot key sequence Alt+D, then press the S key
- Design time: when an application is being developed
- Run time: when a program is executing

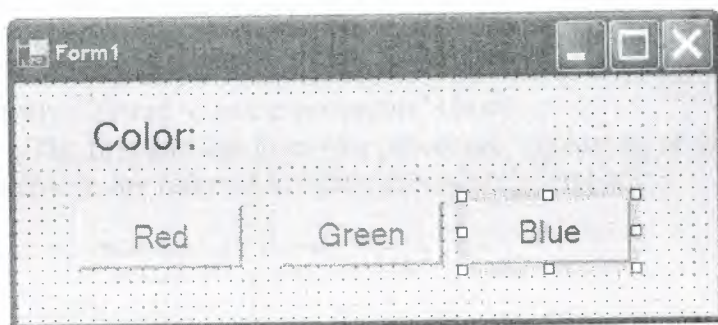
1.9.11 Saving and Recalling a Project

- To save an application
 - Click the File menu and then click Save All or
 - Click the SaveAll icon in the Standard Toolbar
- To retrieve a project:
 - Select Open Solution from the File menu

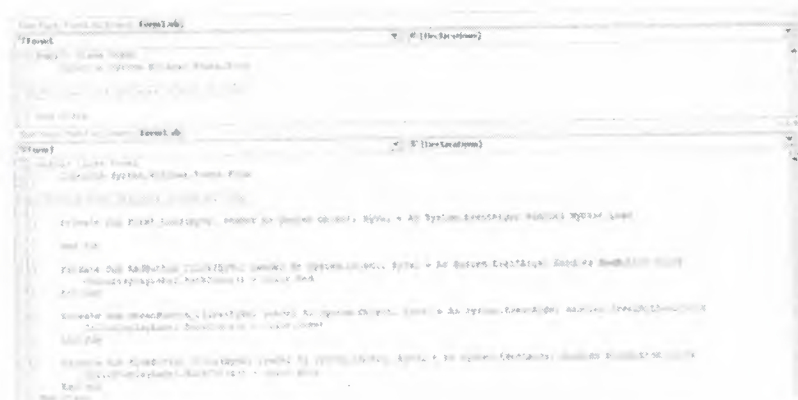
1.9.12 Example – Step 1: Adding Controls



1.9.13 Example -- Step 2: Adding Properties



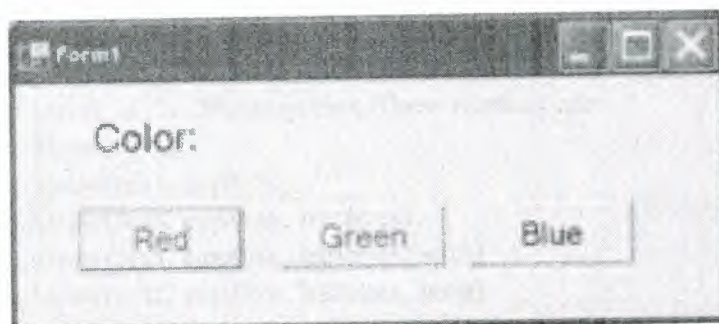
1.9.14 Example -- Step 3: Adding Events



1.9.15 Running an Application

A project being developed can be executed by using any one of the following three methods:

- Select the Debug Menu and click Start.
- Press the F5 function key.
- Use the hot key sequence Alt+D, then press the S key.



1.9.16 Adding an Event Procedure

- A procedure that is executed when an event occurs is referred to as an event procedure or event handler.

- The first line of a procedure is always a header line. A header line begins with the optional keyword `Private` and must contain the keyword `Sub`, the name of the procedure, and a set of parentheses.
- The last line of each procedure consists of the keywords `End Sub`.
- All statements from the header line to and including the `End Sub` statement are collectively referred to as the procedure's body.
- The first and last lines of a procedure, consisting of the header line and the `End Sub` statement, are referred to as the procedure's template

```

If RadioButton4.Checked = True Then
    GroupBox10.Visible = True
Else : GroupBox10.Visible = False
    GroupBox14.Visible = False

End If
End Sub

Private Sub TabPage3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    GroupBox10.Visible = True
    GroupBox14.Visible = False
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    SqlDataAdapter4.Fill(DataSet11.otocar)
    CrystalReportViewer1.ReportSource = xx
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)

    SqlDataAdapter8.Fill(DataSet11.otocar)
    DataGrid5.Visible = False
    DataGrid4.Visible = False
    DataGrid6.Visible = False
    DataGrid7.Visible = False
    DataGrid8.Visible = False
    GroupBox17.Visible = True
    TextBox9.Text = ""
    TextBox8.Text = ""
    TextBox10.Text = ""

```

1.9.17 The `MessageBox.Show` Method

- The **`MessageBox.Show`** method can be used in a procedure to display a message to the user through a message box.

- The message box also contains a title and an icon.
- The general forms of the **`MessageBox.Show`** method are:

`MessageBox.Show(text)`

`MessageBox.Show(text, caption)`

`MessageBox.Show(text, caption, buttons)`

`MessageBox.Show(text, caption, buttons, icon)`

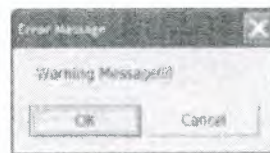
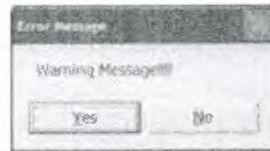
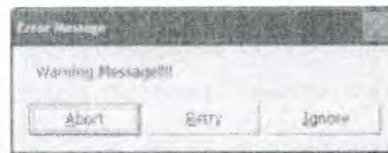
`MessageBox.Show(text, caption, buttons, icon)`

1.9.18 Message Box with Caption



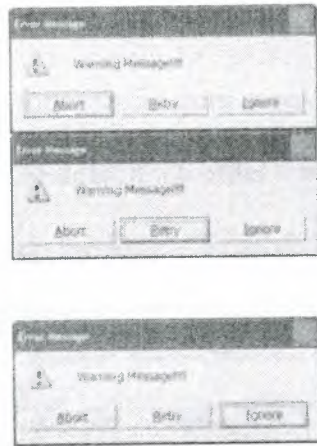
```
MessageBox.Show("Warning Message!!!!")
```

1.9.19 Message Box with Buttons



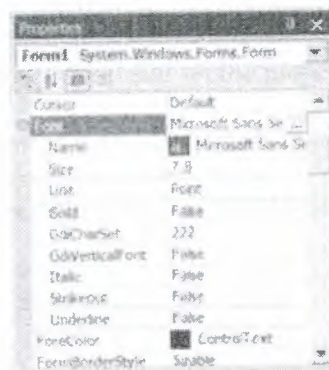
```
MessageBox.Show("Warning Message!!!!", "Error  
Message",  
    MessageBoxButtons.AbortRetryIgnore)  
MessageBox.Show("Warning Message!!!!", "Error  
Message",  
    MessageBoxButtons.YesNo)  
MessageBox.Show("Warning Message!!!!", "Error  
Message",  
    MessageBoxButtons.OKCancel)  
Other values  
    MessageBoxButtons.OK  
    MessageBoxButtons.RetryCancel  
    MessageBoxButtons.YesNoCancel
```


1.9.20 Message Box with Default Button



- `MessageBoxDefaultButton.Button1` - specifies the leftmost button and is the default.
- `MessageBoxDefaultButton.Button2` - specifies the second button from the left.
- `MessageBoxDefaultButton.Button3` - specifies the third button from the left.
- When the button is clicked, the value returned is one of the following:
 - `DialogResult.Abort`
 - `DialogResult.Cancel`
 - `DialogResult.Ignore`
 - `DialogResult.No`
 - `DialogResult.OK`
 - `DialogResult.Retry`
 - `DialogResult.Yes`

1.9.21 Setting the Initial Object Properties



Once controls have been added to a form, we can use the Properties window to change one or more of the properties of the controls.

1.9.22 Looking at the Focus and Tab Sequence

- When an application is run and a user is looking at the form, only one of the form's controls will have *input focus*, or *focus*.

- Only objects which are capable of responding to user input through either the keyboard or mouse can receive focus.
- In order to receive the focus, a control must have its *Enabled*, *Visible*, and *TabStop* properties set to True.
- The sequence in which the focus shifts from control to control as the tab key is pressed by the user is called the *tab sequence*.
- The programmer can alter the default tab order – which is obtained from the sequence in which controls are placed on the form – by modifying an object's *TabIndex* value.



1.9.23 Adding Additional Event Procedures

- Once we have added objects to a form and set the properties of these controls, the next step in creating a Visual Basic application is to supply these objects with event code.
- Each object can have many events associated with it.
- To enter the event code for an object, we can double-click the object to open its Code window.

1.9.24 Statement

- All *statements* belong to one of two categories of statements:
 - executable statements
 - nonexecutable statements.
- An *executable* statement causes some specific action to be performed by the compiler or interpreter.
- A *nonexecutable* statement describes some feature of either the program or its data but does not cause the computer to perform any action.

1.9.25 Help Facility

- Dynamic Help

- The Dynamic Help window displays a list of help topics that changes as you perform operations
- To open the Dynamic Help window, click Help on the menu bar and then click Dynamic Help
 - Context-sensitive Help
- Context-sensitive Help immediately displays a relevant article for a topic
- To use this facility, select an object and press F1

1.10 Programming in VB.NET

Creating an application using an OO programming language such as VB .NET is much easier than working with a traditional programming language. Instead of having to develop the logic for the entire program as you would with a procedural language, you can divide up the program logic into small, easily handled parts by working with objects and events. For each object, you determine the events that you want the object to respond to and then develop code to have the object provide the desired response. All of the necessary messages between objects in Windows are handled by VB .NET, thereby significantly reducing the work you must do to create an application. The manner in which you create a VB .NET project is also different from traditional programming. Instead of having to create an entire program before testing any part of it, with VB .NET you can use **interactive development** to create an object, write the code for it, and test it before going onto other objects. For example, assume a store named Vintage DVDs that rents only “old” movies on DVD has asked you to create a VB .NET project that includes calculating taxes on a DVD rental and sums the taxes and price to compute the amount due. With VB .NET, you can create the objects and code to calculate the taxes and amount due and test them to ensure their correctness, before going on to the rest of the project. While creating an application in VB .NET is easier than working with a procedural language, you still need to follow a series of steps to ensure correctness and completeness of the finished product. These steps are:

1. Define problem
2. Create interface
3. Develop logic for action objects
4. Write and test code for action objects
5. Test overall project
6. Document project in writing

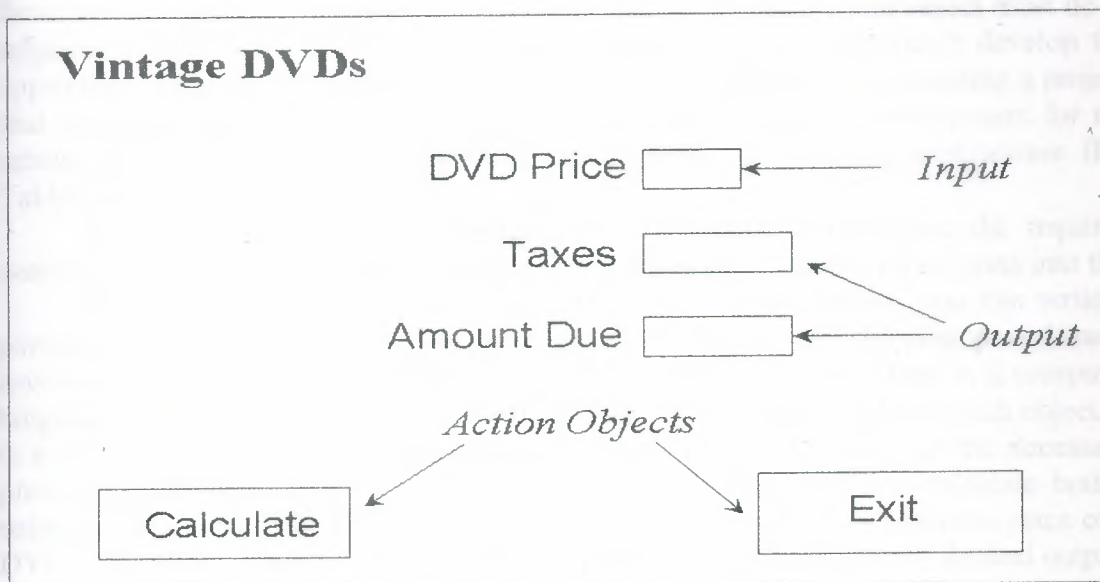
It should be noted that it may be necessary to repeat or iterate through these steps to arrive at an acceptable final solution to the original problem. In the next sections, we will discuss each of these steps and apply them to a part of the situation just mentioned, that is, creating an application to calculate the taxes and amount due on a DVD rental.

1.10.1 Step One: Define Problem

Before we can hope to develop any computer application, it is absolutely necessary to clearly define our objective, that is, the problem to be solved. Only then can we begin to develop the correct logic to solve the problem and incorporate that logic into a computer application. Ensuring that the correct problem is being solved requires careful study of why a problem exists. Maybe an organization is currently handling some repetitive process manually and wants to use a computer to automate it. Or maybe management has a complicated mathematical or financial problem that cannot be solved by hand. Or maybe a situation has occurred or will occur that cannot be handled by an existing program. The problem identification step should include identification of the data to be *input* to the program and the desired results to be *output* from the program. Often these two items will be specified by a person or an agency other than the programmer. Much grief can be avoided if these input and output requirements are incorporated into the programmer's thinking at this early stage of program development. Unclear thinking at this stage may cause the programmer to write a program that does not correctly solve the problem at hand, or a program that correctly solves the wrong problem, or a combination of both! Therefore the programmer *must* spend as much time as is necessary to truly identify and understand the problem.

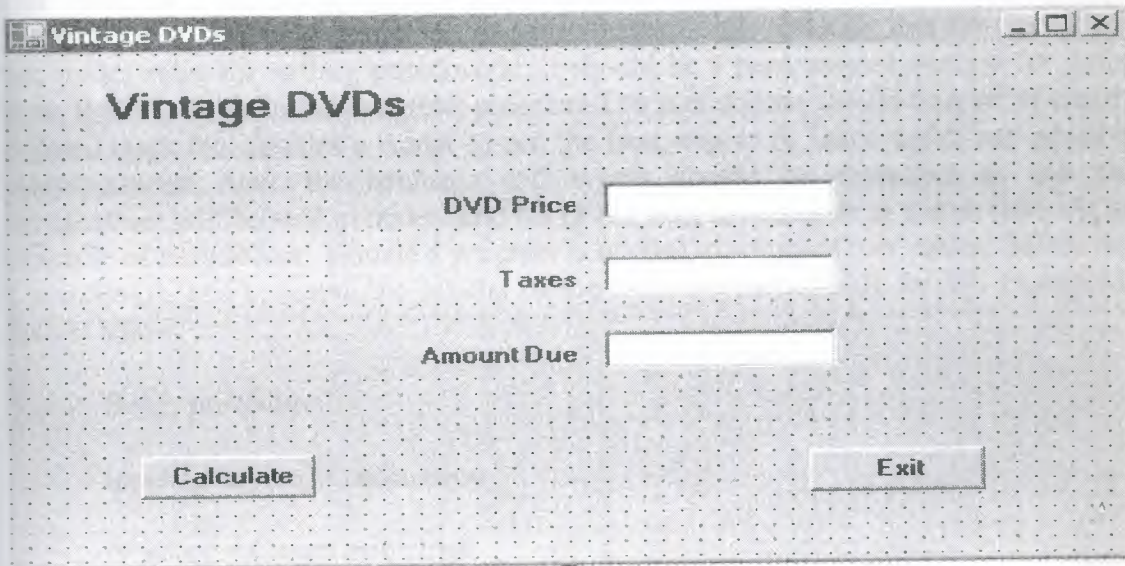
Because VB .NET is a *visual* language, a good way to understand what is required to solve the problem is to sketch the interface showing the various objects that will be part of the project. Not only does this help you understand the problem, it is also a good way for you to communicate your understanding to other people. As a part of this sketch, you should denote the input and output objects and the objects for which code is needed to respond to events, the so-called **action objects**. A sketch of the proposed solution for the DVD rental problem is shown in Figure . In looking at , you will see one input—the price of the DVD—and two outputs—the taxes and the amount due. There are also two action objects—a calculation button and an exit button. If there are multiple forms, they should all be sketched with input, output, and action objects denoted as in Figure

1.11.2 Step Two: Create Interface



Once you have defined the problem and, using a sketch of the interface, have decided on the objects that are necessary for your project, you are ready to create the interface. Creating the interface with VB .NET is quite easy: You select objects from those available and place them on the form. This process should follow the sketch done earlier. While you have not yet been introduced to the wide variety of objects available for creating VB .NET projects, we can work on the logic for the Vintage DVDs problem with just four types of objects: the form, buttons for action, textboxes for input and output, and labels for descriptors. The interface in VB .NET is shown in Figure

1.10.3 Step Three: Develop Logic for Action Objects



Once the problem has been clearly identified and the interface created, the next step is to develop the logic for the action objects in the interface. This is the step in the development process where you have to think about what each action object must do in response to an event. No matter how good your interface, if you don't develop the appropriate logic for the action objects, you will have great difficulty creating a project that solves the problem defined earlier. To help with this logical development for the action objects, there are two useful tools for designing programming applications: IPO Tables and pseudocode. **IPO**

(Input/Processing/Output) **Tables** show the inputs to an object, the required outputs for that object, and the processing that is necessary to convert the inputs into the desired outputs. Once you have an IPO Table for an object, you can write a pseudocode procedure to complete the logic development step. Writing **pseudocode** involves writing the code for the object in structured English rather than in a computer language. Once you have developed an IPO Table and the pseudocode for each object, it is a very easy step to write a **procedure** in VB.NET that will carry out the necessary processing. **IPO Table** Let's begin by developing the logic for the Calculate button using an IPO Table. The IPO Table for the Calculate button has as input the price of a DVD. The processing involves the calculation necessary to compute the desired output: the amount of the sale. As mentioned earlier, in many cases the program designer will have no control over the input and output. They will be specified by somebody else—

either the person for whom the application is being developed or, if you are a member of a team and are working on one part of the overall application, the overall design. Once you are given the specified input and output, your job is to determine the processing necessary to convert the inputs into desired outputs. Figure 1-9 shows the IPO table for the calculation button. IPO tables are needed for all objects that involve input, output, and processing. We won't do one for the Exit button since it simply terminates the project. **Pseudocode** Once you have developed the IPO tables for each action object, you should then develop a pseudocode procedure for each one. Pseudocode is useful for two reasons. First, you can write the procedure for the object in English without worrying about the special syntax and grammar of a computer language. Second, pseudocode provides a relatively direct link between the IPO Table and the computer code for the object, since you use English to write instructions that can then be converted into program instructions. Often, this conversion from pseudocode statement to computer language instruction is virtually line for line. There are no set rules for writing pseudocode; it should be a personalized method for going from the IPO Table to the computer program. The pseudocode should be a set of clearly defined steps that enables a reader to see the next step to be taken under any possible circumstances. Also, the language and syntax should be consistent so that the programmer will be able to understand his or her own pseudocode at a later time. As an example of pseudocode, assume a program is needed to compare two values, Salary and Commission, and to output the smaller of the two. The pseudocode for this example is shown below:

Begin procedure

Input Salary and Commission

If Salary < Commission then

Output Salary

Else

Output Commission

End Decision

End procedure

IPO Table for Calculate Button

Input	Processing	Output
Video price	$Taxes = 0.07 \times Price$ $Amount\ due = Price + Taxes$	Taxes Amount due

In this pseudocode, it is easy to follow the procedure. Note that parts of it are indented to make it easier to follow the logic. The important point to remember about pseudocode is that it expresses the logic for the action object to the programmer in the same way that a computer language expresses it to the computer. In this way, pseudocode is like a personalized programming language. Now let's write a pseudocode procedure for the Vintage DVDs Calculate object. Note that the pseudocode program matches the IPO Table shown in Figure

Begin procedure

Input DVD Price

Taxes = 0.07 x DVD Price

Amount Due = DVD Price + Taxes

Output Taxes and Amount Due

End procedure

While we have only one object in our small example for which an IPO Table and pseudocode are needed, in most situations you will have numerous objects for which

you will need to develop the logic using these tools.

1.10.4 Step Four: Write and Test Code for Action Objects

Once you have created the VB .NET interface and developed the logic for the action objects using IPO Tables and pseudocode, you must write procedures in VB .NET for each action object. This code should provide instructions to the computer to carry out one or more of the six operations listed earlier—that is, input data, store data in internal memory, perform arithmetic on data, compare two values and select one of two alternative actions, repeat a group of actions any number of times, and output the results of processing. While creating the interface is important, writing the code is the essence of developing an application. Since you have not yet been introduced to the rules for writing code in VB .NET for the various objects, we will defer a full discussion of this step until the end and beyond. However, you should be able to see the similarity between the VB .NET event procedure displayed in VB Code Box 1-1 and the pseudocode version shown earlier. The differences are due to the way VB .NET handles input and output. Input is from the Text property of the first textbox, named txtDVDPrice. Output goes to the Text property of the two textboxes named txtTaxes and txtAmountDue. There are also statements that begin with the word Dim, to declare the variables, and comment statements that begin with an apostrophe ('). Once you have written the code for an action object, the second part of this step is to test that object and correct any errors; don't wait until the entire project is completed. Use the interactive capabilities of VB .NET to test the code of each and every object as it is written. This process is referred to as **debugging**—trying to remove all of the errors or “bugs.” Because VB .NET automatically checks each line of the code of an object for syntax or

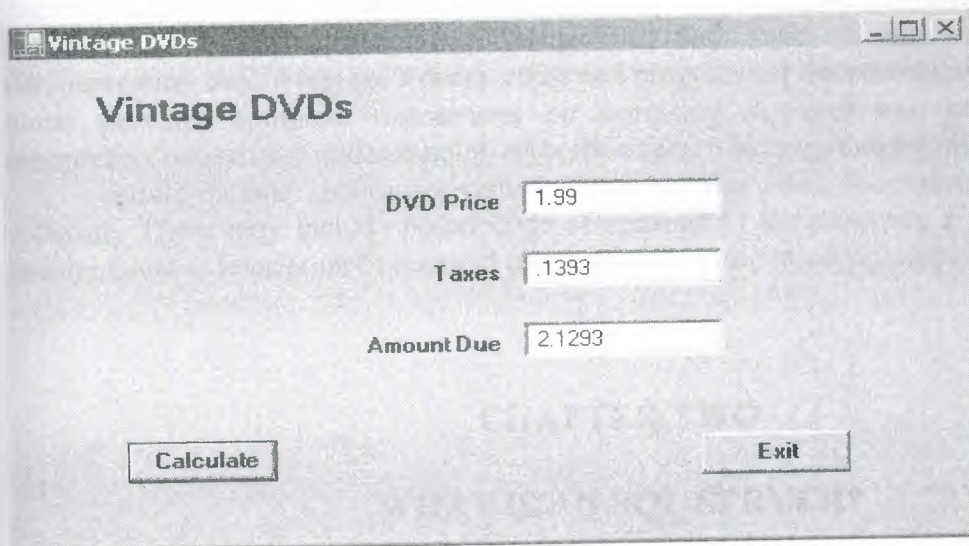
vocabulary errors, the debugging process is much easier than in other languages. However, even if all the syntax and vocabulary are correct, the code for an object still may be incorrect—either in the manner in which it carries out the logic or in the logic itself. The best way to find and correct such errors is to use **test data** for which the results are known in advance. If the results for the object do not agree with the results from the hand calculations, an error exists, either in the logic or in the hand calculations. After the hand calculations have been verified, the logic must be checked. For example, if the results of the Calculate button came out different from what was expected, then we would need to look for a problem in the data or the logic. In the case of the Calculate button, we want to determine if the code shown in VB Code Box 1-1 will actually compute and output to the textboxes the *correct* taxes and amount due for the DVD price entered in the first textbox. Figure shows the result of clicking the Calculate button for a DVD with a price of \$1.99. Note that the results, while correct, are not *exactly* what you might expect. Instead of rounded values of \$.14 for the taxes and \$2.13 for the amount due, the answers are the exact values of \$.1393 and \$2.1293. This is because we have not *formatted* the answers as dollar and cents. This will be done when we revisit this problem in next. While the answers for this set of test data are correct, this does not mean it will work for all test data. Testing requires that a wide variety of test data be used to assure that the code for the object works under all circumstances. Since the Calculate button appears to work, we can now write the code for the Exit button, which consists of one instruction: End. If this command also works, then we are ready to move on to the next step in the application development process: testing the overall project

CODE BOX

VB.NET computation of Taxes and Amount Due

```
Private Sub cmdCalc_Click(ByVal eventSender As System.Object, _  
    ByVal eventArgs As System.EventArgs) Handles cmdCalc.Click  
    Const sngTaxRate As Single = 0.07 'Use local tax rate  
    Dim decPrice as Decimal, decAmountDue As Decimal  
    Dim decTaxes As Decimal  
    decPrice = CDec(txtDVDPrice.Text)  
    decTaxes = decPrice * sngTaxRate 'Compute taxes  
    decAmountDue = decPrice + decTaxes 'Compute amount due  
    txtTaxes.Text = CStr(decTaxes)  
    txtAmountDue.Text = CStr(decAmountDue)  
    txtDVDPrice.Text = Cstr(decPrice)  
End Sub
```


Testing the Calculate button



Vintage DVDs

DVD Price 1.99

Taxes .1393

Amount Due 2.1293

Calculate Exit

1.10.5 Step Five: Test Overall Project

Once you have tested the code for each action object individually, the next step is to test the overall project and correct any errors that may still exist or that may be the result of incorrect communication between objects. At this stage it is necessary to determine whether the results obtained from the project meet the objectives outlined in the Problem Definition step. If the project does not meet the final user's needs, then the developer must analyze the results and the objectives to find out where they diverge. After the analysis, the developer should trace through the program development procedure and correct the algorithm, IPO Tables, pseudocode, and final code for one or more objects to find the cause of the difference between the objectives and the final project.

1.10.6 Step Six: Document Your Project in Writing

An important part of writing any computer software is the documentation of the software. **Documentation** can be defined as *the written descriptions of the software that aid users and other programmers*. It includes both internal descriptions of the code instructions and external descriptions and instructions. Documentation helps users by providing instructions and suggestions on using the software. Documentation helps other programmers who may need to make changes or correct the programs. Internal documentation usually includes *comments* within the program that are intermingled with the program statements to explain the purpose and logic of the program elements. In VB Code Box, the statements beginning with an apostrophe (') are examples of internal documentation. This type of documentation is essential to the maintenance of software, especially by someone other than the original programmer. By being able to read the original programmer's purpose for a part of a program or a program statement, a different programmer can make any needed corrections or revisions. Without internal documentation, it can be extremely difficult for anyone to understand the purpose of

parts of the program. And, if a programmer is unclear about what's going on in the program, making needed changes will be very difficult. In this text, because we will be explaining the code in detail, we do not include the level of internal documentation that *should* be found in the projects you create both here and in your work. Written documentation includes books, manuals, and pamphlets that give instructions on using the software and also discuss the objectives and logic of the software. The documentation should include a user's guide and programmer documentation. The *user's guide* provides complete instructions on accessing the software, entering data, interpreting output, and understanding error messages. The *programmer documentation* should include various descriptive documents that allow for maintenance of the software. These may include pseudocode of sections of the program, a listing of the program, and a description of required input values and the resulting output.

CHAPTER TWO

WHAT IS AN SQL SERVER?

You use SQL Server to manage two types of databases-online transaction processing (OLTP) databases, and online analytical processing (OLAP) databases. Typically, separate clients access the databases by communicating over a network. You can scale SQL Server up to terabyte-size databases and down to small business servers and portable computers. You can scale SQL Server out to multiple servers by using Windows Clustering in Windows 2000.

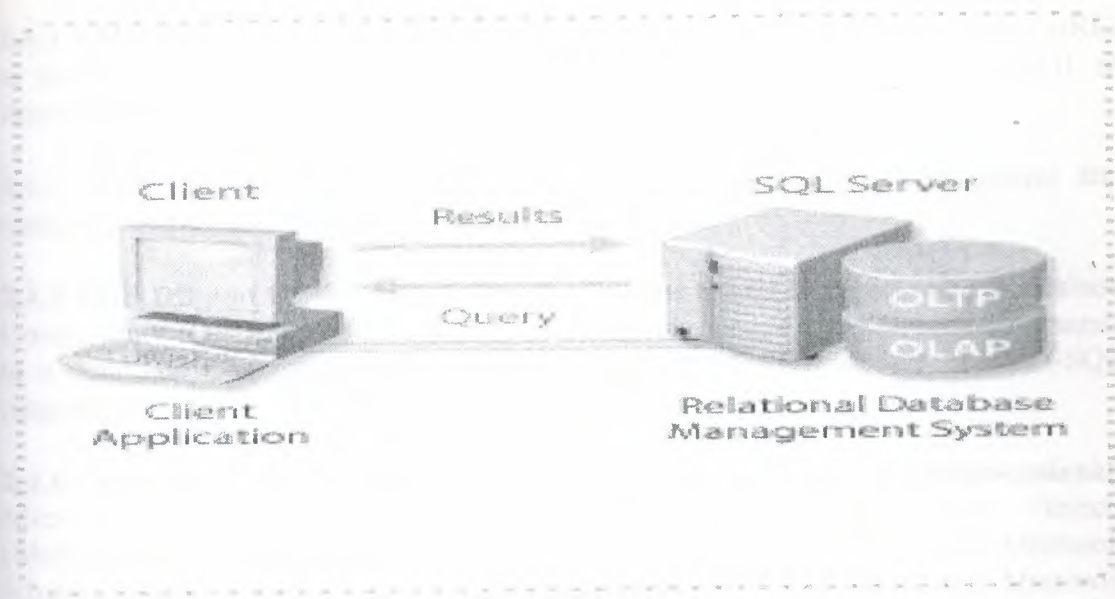
In this section we will learn about the following topics:

- Introduction to SQL Server
- Client-server components
- Client-server communication process
- SQL Server services

2.1 INTRODUCTION TO SQL SERVER

You can use SQL Server to perform transaction processing, store and analyze data, and build new applications. SQL Server is a family of products and technologies that meets the data storage requirements of OLTP and OLAP environments. SQL Server is a relational database management system (RDBMS) that:

- Manages data storage for transactions and analysis.
- Responds to requests from client applications.
- Uses Transact-SQL, Extensible Markup Language (XML), multidimensional expressions (MDX), or SQL Distributed Management Objects (SQL-DMO) to send requests between a client and SQL Server.



2.2 Relational Database Management System

The RDBMS of SQL Server is responsible for:

- Maintaining the relationships among data in a database. Ensuring that data is stored correctly and that the rules defining the relationships among data are not violated.
- Recovering all data to a point of known consistency, in the event of a system failure.

2.3 Data Storage Models

SQL Server manages OLTP and OLAP databases

2.3.1 OLTP Databases Data in an OLTP database is generally organized into relational tables to reduce redundant information and to increase the speed of updates. SQL Server enables a large number of users to perform transactions and simultaneously change real-time data in OLTP databases. Examples of OLTP databases include airline ticketing and banking transaction systems.

2.3.2 OLAP Databases OLAP technology organizes and summarizes large amounts of data so that an analyst can evaluate data quickly and in real time. SQL Server 2000 Analysis Services organizes this data to support a wide array of enterprise solutions, from corporate reporting and analysis to data modeling and decision support.

2.4 Client Applications

Users do not access SQL Server and Analysis Services directly; instead, they use separate client applications written to access the data. These applications access SQL Server by using:

2.4.1 Transact-SQL This query language, a version of Structured Query Language (SQL), is the primary database query and programming language that SQL Server uses.

2.4.2 XML This format returns data from queries and stored procedures by using URLs or templates over Hypertext Transfer Protocol (HTTP). You also can use XML to insert, delete, and update values in a database.

2.4.3 MDX The MDX syntax defines multidimensional objects and queries and manipulates multidimensional data in OLAP databases.

2.4.5 OLE DB and ODBC APIs Client applications use OLE DB and Open Database Connectivity (ODBC), application programming interfaces (APIs) to send commands to a database. Commands that you send through these APIs use the Transact-SQL language.

2.4.6 ActiveX Data Objects and ActiveX Data Objects (Multidimensional) Microsoft ActiveX® Data Objects (ADO) and ActiveX Data Objects (Multidimensional) (ADO MD) wrap OLE DB for use in languages such as Microsoft Visual Basic®, Visual Basic for Applications, Active Server Pages, and Microsoft Internet Explorer Visual Basic Scripting. You use ADO to access data in OLTP databases. You use ADO MD to access data in Analysis Services data cubes.

2.4.7 English Query This application provides an Automation API that lets users resolve natural-language questions instead of writing complex Transact-SQL or MDX statements about information in a database. For example, users are able to ask the question, "What are the total sales for Region 5?"

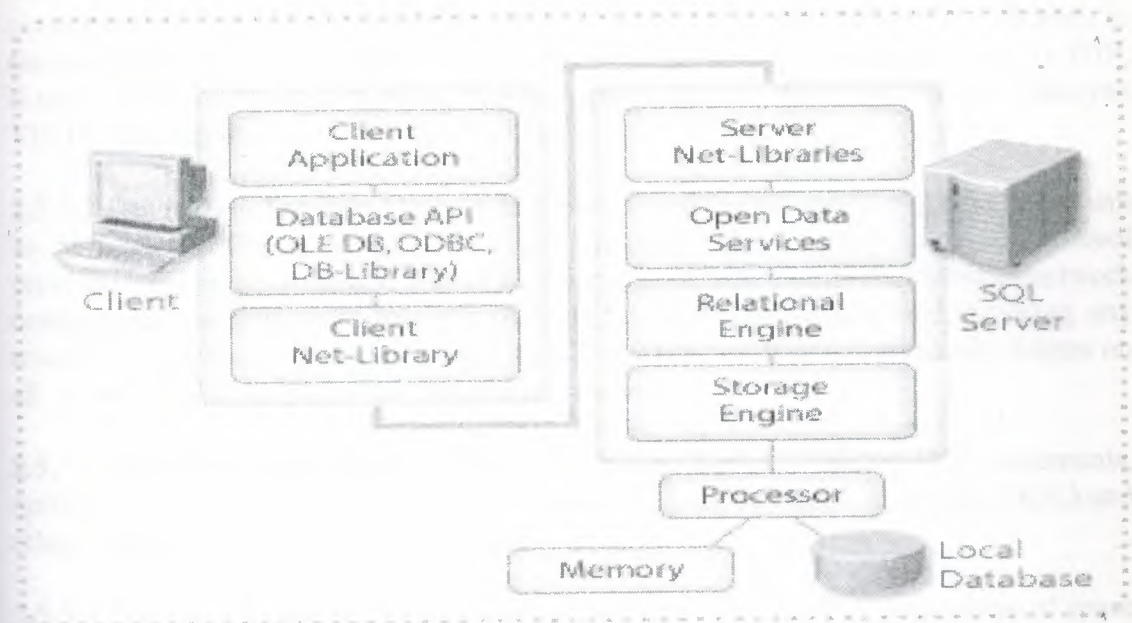
2.5 CLIENT SERVER COMPONENT

SQL Server consists of client and server components that store and retrieve data. SQL Server uses layered communication architecture to isolate applications from the underlying network and protocols. This architecture allows you to deploy the same application in different network environments.

2.5.1 Client-Server Architecture

SQL Server uses client-server architecture to separate the workload into tasks that run on server computers and those that run on client computers:

- The client is responsible for business logic and presenting data to the user. The client typically runs on one or more computers, but it also can run on the server computer along with SQL Server.
- SQL Server manages databases and allocates the available server resources such as memory, network bandwidth, and disk operations among multiple requests.



Client-server architecture allows you to design and deploy applications to enhance a variety of environments. Client programming interfaces provide the means for applications to run on separate client computers and communicate to the server over a network.

2.5.2 Client Components

The client components in the communication architecture include:

2.5.2.1 Client Application A client application sends Transact-SQL statements and receives result sets. You develop an application by using a database API. The application has no knowledge of the underlying network protocols used to communicate with SQL Server.

2.5.2.2 Database API Database API (OLE DB, ODBC) uses a provider, driver, or DLL to pass Transact-SQL statements and receive result sets. This is an interface that an application uses to send requests to SQL Server and to process results that SQL Server returns.

2.5.2.3 Client Net-Library A client Net-Library manages network connections and routing on a client. This is a communication software component that packages the database requests and results for transmission by the appropriate network protocol.

2.5.3 Server Components

The server components in the communication architecture include:

2.5.3.1 Server Net-Libraries SQL Server can monitor multiple Net-Libraries concurrently. The client Net-Library must match one of the server Net-Libraries to communicate successfully. SQL Server supports network protocols such as TCP/IP, Named Pipes, NWLink, IPX/SPX, VIA ServerNet II SAN, VIA GigaNet SAN, Banyan VINES, and AppleTalk.

2.5.3.2 Open Data Services Open Data Services makes data services appear to a client as SQL Server by providing a network interface for handling network protocol processes and server routines. This is a component of SQL Server that handles network connections, passing client requests to SQL Server for processing and returning any results and replies to SQL Server clients. Open Data Services automatically listens on all server Net-Libraries that are installed on the server.

2.5.3.3 Relational Engine The relational engine parses Transact-SQL statements, optimizes and executes execution plans, processes data definition language (DDL) and other statements, and enforces security.

2.5.3.4 Storage Engine The storage engine manages database files and the use of space in the files, builds and reads data from physical pages, manages data buffers and physical input/output (I/O), controls concurrency, performs logging and recovery operations, and implements utility functions such as Database Consistency Checker (DBCC), backup, and restore.

2.6 Client-Server Communication Process

Clients and servers typically communicate over a network. The following sequence uses a query to illustrate the typical client-server communication process using a database API:

1-A client application submits a query. The client calls the database API and passes the query. The database API uses a provider, driver, or DLL to encapsulate the query in one or more Tabular Data Stream (TDS) packets and pass packets to the client Net-Library.

2-The client Net-Library packages the TDS packets into network protocol packets. The client Net-Library calls a Windows interprocess communication (IPC) API to send the network protocol packets to a server Net-Library by using the network protocol stack of the operating system. The appropriate server Net-Library extracts the TDS packets from the network protocol packets and passes the TDS packets to Open Data Services.

3-Open Data Services extracts the query from the TDS packets and passes the query to the relational engine. The relational engine then compiles the query into an optimized execution plan. It executes the execution plan. The relational engine communicates with the storage engine by using the OLE DB interface.

4-The storage engine transfers data from a database to data buffers and then passes rowsets containing data to the relational engine. The relational engine combines the rowsets into the final result set and passes the result set to Open Data Services.

5-Open Data Services packages the result set and returns it to the client application by using a server Net-Library, the network protocol stack, the client Net-Library, and the database API. The result set can also be returned in XML format.

2.7 SQL SERVER SERVICES

The SQL Server services include MSSQLServer service, SQLServerAgent service, Microsoft Distributed Transaction Coordinator (MS DTC), and Microsoft Search. Although these SQL Server services usually run as services on Windows 2000, they also can run as applications.

2.7.1 Four SQL Server Services

SQL Server includes four services, which are installed by default with a new installation: MSSQLServer service, SQLServerAgent service, Microsoft Distributed Transaction Coordinator, and Microsoft Search.

2.7.1.1 MSSQLServer Service

MSSQLServer service is the database engine. It is the component that processes all Transact-SQL statements and manages all files that comprise the databases on the server. MSSQLServer service:

- Allocates computer resources among multiple concurrent users.
- Prevents logic problems, such as timing requests from users who want to update the same data at the same time.
- Ensures data consistency and integrity.

2.7.1.2 SQLServerAgent Service

SQLServerAgent service works in conjunction with SQL Server to create and manage alerts, local or multiserver jobs, and operators. Consider the following about SQLServerAgent service:

- Alerts provide information about the status of a process, such as when a job is complete or when an error occurs.
- SQLServerAgent service includes a job creation and scheduling engine that automates tasks.
- SQLServerAgent service can send e-mail messages, page an operator, or start another application when an alert occurs. For example, you can set an alert to occur when a database or transaction log is almost full or when a database backup is successful.

2.7.1.3 Microsoft Distributed Transaction Coordinator

MS DTC allows clients to include several different sources of data in one transaction. MS DTC coordinates the proper completion of distributed transactions to ensure that

all updates on all servers are permanent-or, in the case of errors, that all modifications are cancelled.

2.7.1.4 Microsoft Search

Microsoft Search is a full-text engine that runs as a service in Windows 2000. Full-text support involves the ability to issue queries against character data and the creation and maintenance of the indexes that facilitate these queries.

2.7.2 Multiple Instances of SQL Server

Multiple instances of the SQL Server may run concurrently on the same computer. Each instance of SQL Server has its own set of system and user databases that are not shared between instances. Each instance operates as if it were on a separate server. Applications can connect to each SQL Server database engine instance on a computer in nearly the same way that they connect to SQL Server database engines running on different computers.

When you specify only the computer name, you work with the default instance. You must specify the *computer_name\instance_name* to connect to a named instance.

2.8 SQL SERVER INTEGRATION

INTEGRATING SQL SERVER WITH OPERATING SYSTEMS

SQL Server includes client and server components that run on various operating systems.

2.8.1 Client Components

The client components from all SQL Server 2000 editions, except SQL Server Windows CE Edition, run on all editions of Windows 2000, versions of Microsoft Windows NT®, on Microsoft Windows Millennium Edition (Me), Microsoft Windows 98, and Microsoft Windows 95.

All client components from SQL Server 2000 CE edition run exclusively on the Windows CE operating system.

2.8.2 Server Components

The various editions of SQL Server allow it to run on all editions of Windows 2000, versions of Windows NT, Windows Me, Windows 98, and Windows CE. Specific versions of the operating systems and editions of SQL Server limit server components. Microsoft Windows NT Server 4.0, Service Pack 5 (SP5) or later must be installed as a minimum requirement for all SQL Server 2000 editions. Only the server components, such as the database engine and the Analysis server, are limited to specific versions of the operating systems. For example, although the database engine for Microsoft SQL Server 2000 Enterprise Edition does not run on Microsoft Windows 2000 Professional, Microsoft Windows NT Workstation, Windows Me, or Windows 98, you can use the

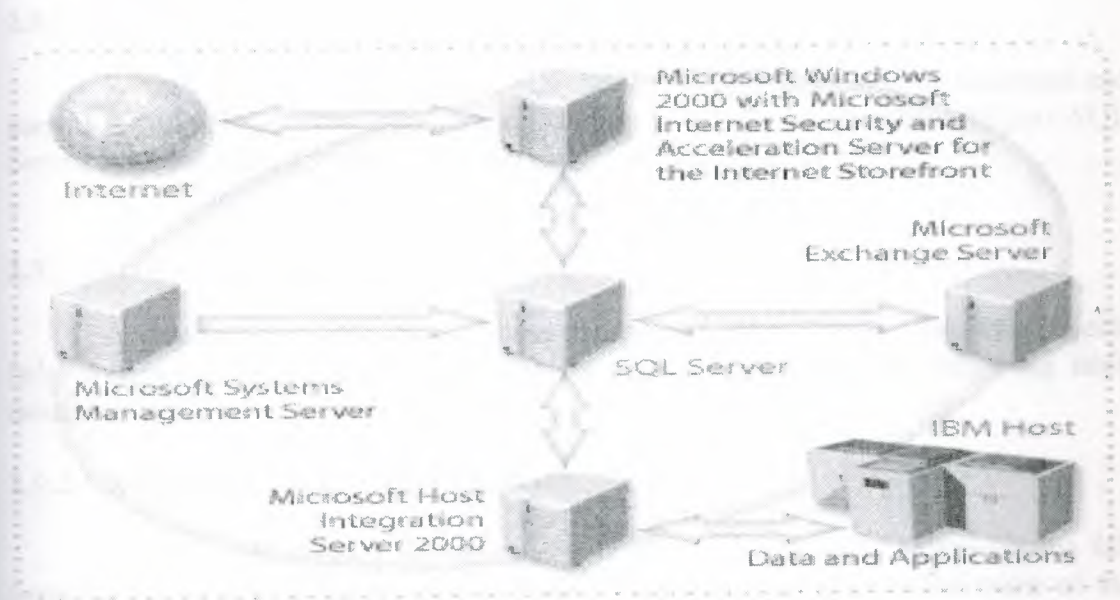
SQL Server 2000 Enterprise Edition compact disc to install the client software on any of these operating systems. Windows NT 4.0 Terminal Server does not support SQL Server 2000.

2.8.3 Internet Browsers and Third-Party Applications

Internet browsers and third-party client applications running on various operating systems also can access SQL Server.

2.8.3.1 Microsoft Internet Information Services SQL Server uses Microsoft Internet Information Services (IIS) so that Internet browsers can access a SQL Server database by using the HTTP protocol

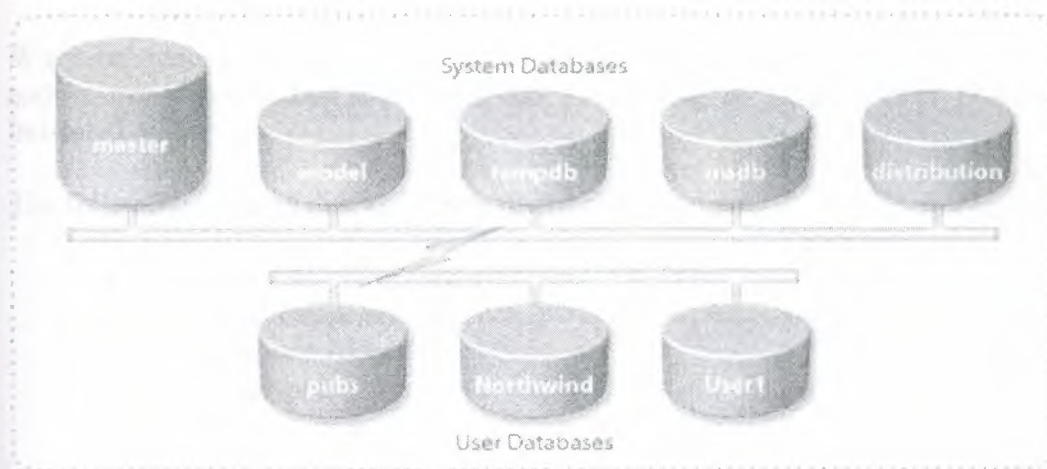
2.8.3.2 Integrating SQL Server with Other Microsoft Server Applications



SQL Server integrates well with other Microsoft server applications. Microsoft provides a group of server applications that work together to help you build business solutions.

2.9 SQL SERVER DATABASES

Each SQL Server has two types of databases: *system databases* and *user databases*. System databases store information about SQL Server as a whole. SQL Server uses system databases to operate and manage the system. User databases are databases that users create.



2.9.1 Types of Databases

When SQL Server is installed, SQL Server Setup creates system databases and sample user databases. The Distribution Database is installed when you configure SQL Server for replication activities.

2.9.2 Database Objects

A database is a collection of data, tables, and other objects. Database objects help you structure data and define data integrity mechanisms. The following table describes SQL Server database objects.

2.9.2.1 Referring to SQL Server Objects

You can refer to SQL Server objects in several ways. You can specify the full name of the object (its fully qualified name), or specify only part of the name of the object name and allow SQL Server to determine the rest of the name from the context in which you are working.

2.9.3 Fully Qualified Names

The complete name of a SQL Server object includes four identifiers-the server name, database name, owner name, and object name in the following format:

server.database.owner.object

An object name that specifies all four parts is known as a fully qualified name. Each object that you create in SQL Server must have a unique, fully qualified name. For example, you can have two tables named **Orders** in the same database as long as they belong to different owners. Also, column names must be unique within a table or view.

2.9.4 Partially Specified Names

When referencing an object, you do not always have to specify the server, database, and owner. Intermediate identifiers can be omitted as long as their positions are indicated by periods.

The following list contains valid formats for object names:

- *server.database.owner.object*
- *database.owner.object*
- *database..object*
- *owner.object*
- *object*

When you create an object and do not specify the different parts of the name, SQL Server uses the following defaults:

- Server defaults to the current instance on the local server.
- Database defaults to the current database.
- Owner defaults to the user name in the specified database associated with the login ID of the current connection.

A user that is a member of a role can explicitly specify the role as the object owner. A user that is a member of the **db_owner** or **db_ddladmin** role in a database should specify the **dbo** user account as the owner of an object. This practice is recommended.

The following example creates an **OrderHistory** table in the **Northwind** database.

```
CREATE TABLE Northwind.dbo.OrderHistory  
  
(OrderID int,  
  
ProductID int,  
  
UnitPrice money,  
  
Quantity int,  
  
Discount decimal)
```

Most object references use three-part names and default to the local server. Four-part names are generally used for distributed queries or remote stored procedure calls.

SQL Server supports a three-part naming convention when referring to the current server. The SQL-92 standard also supports a three-part naming convention. The terms used in both naming conventions are different

2.9.5 System Tables

SQL Server stores information, called metadata, about the system and objects in databases for an instance of SQL Server. *Metadata* is information about data. Metadata includes information about the properties of data, such as the type of data in a column (numeric, text, and so on), or the length of a column. It can also be information about the structure of data or information that specifies the design of objects.

2.9.5.1 System Tables The information about data in system tables includes configuration information and definitions of all of the databases and database objects in the instance of SQL Server. Users should not directly modify any system table.

2.9.5.2 Database Catalog Each database (including **master**) contains a collection of system tables that store metadata about that specific database. This collection of system tables is the database catalog. It contains the definition of all of the objects in the database, as well as permissions.

2.9.5.3 System Catalog The system catalog, found only in the **master** database, is a collection of system tables that stores metadata about the entire system and all other databases. Most system tables begin with the **sys** prefix

2.9.6 Metadata Retrieval

When you write applications that retrieve metadata from system tables, you should use system stored procedures, system functions, or system-supplied information schema views.

You can query a system table in the same way that you do any other database table to retrieve information about the system. However, you should not write scripts that directly query system tables, because if the system tables change in future product versions, your scripts may fail or may not provide accurate information.

2.9.7 System Stored Procedures

To make it easier for you to gather information about the state of the server and database objects, SQL Server provides a collection of prewritten queries called system stored procedures. The names of most system stored procedures begin with the **sp_** prefix. The following table describes three commonly used system stored procedures.

The following example executes a system stored procedure to get information on the **Employees** table.

```
EXEC sp_help Employees
```

2.9.8 System and Metadata Functions

System and metadata functions provide a method for querying system tables from within Transact-SQL statements. The following example uses a system function in a query to retrieve the user name for a user ID of 10.

```
SELECT USER_NAME(10)
```


2.9.9 Information Schema Views

Information schema views provide an internal, system table-independent view of the SQL Server metadata. These views conform to the ANSI SQL standard definition for the information schema. Each information schema view contains metadata for all data objects stored in that particular database. The following example queries an information schema view to retrieve a list of tables in a database.

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

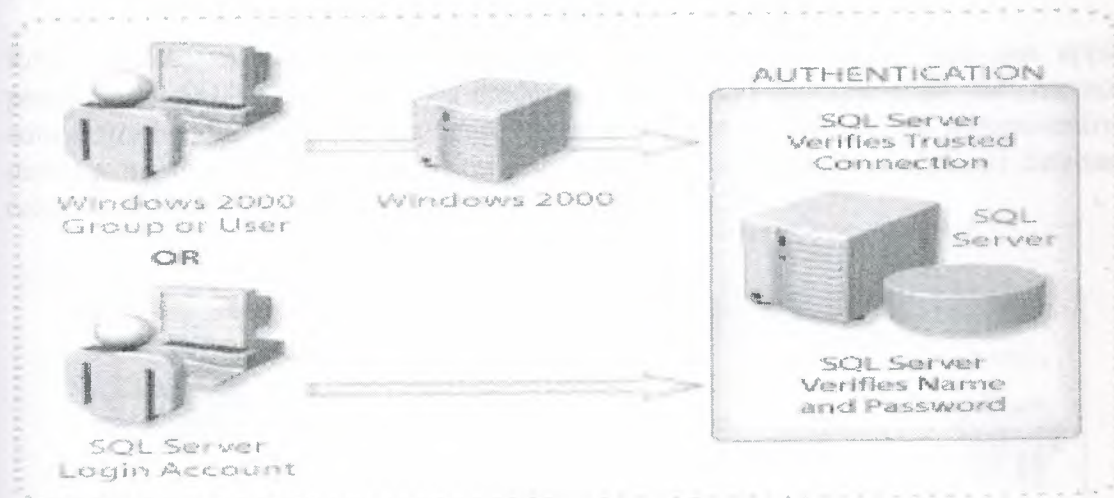
2.10 SQL SERVER SECURITY

2.10.1 Login Authentication

A user must have a login account to connect to SQL Server. SQL Server recognizes two login authentication mechanisms-Windows Authentication and SQL Server Authentication-each of which has a different type of login account.

2.10.2 Windows Authentication

When using Windows Authentication, a Windows 2000 account or group controls user access to SQL Server. A user does not provide a SQL Server login account when connecting. A SQL Server system administrator must define either the Windows 2000 account or the Windows 2000 group as a valid SQL Server login account.



2.10.3 SQL Server Authentication

When using SQL Server Authentication, a SQL Server system administrator defines a SQL Server login account and password. Users must supply both SQL Server logins and passwords when they connect to SQL Server.

2.10.4 Authentication Mode

When SQL Server is running on Windows 2000, a system administrator can specify that it run in one of two authentication modes:

2.10.5 Windows Authentication Mode Only Windows 2000 authentication is allowed. Users cannot specify a SQL Server login account.

2.10.6 Mixed Mode Users can connect to SQL Server with Windows Authentication or SQL Server Authentication.

2.11 Database User Accounts and Roles

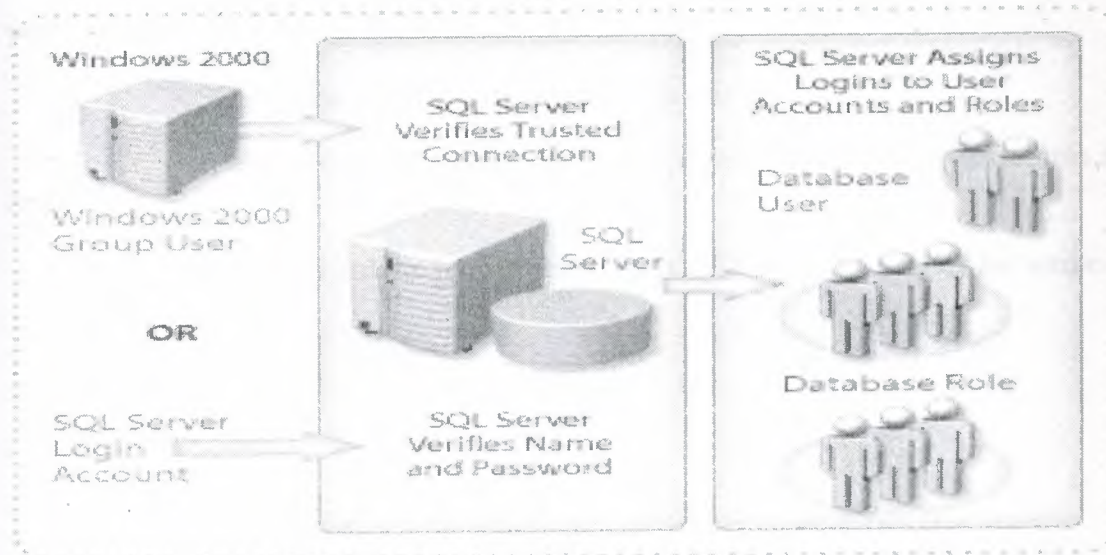
After users have been authenticated by Windows 2000 or SQL Server and have been allowed to log in to SQL Server, they must have accounts in a database. User accounts and roles identify a user within a database and control ownership of objects and permissions to execute statements.

2.11.1 Database User Accounts

The user accounts that apply security permissions are Windows 2000 users or groups or SQL Server login accounts. User accounts are specific to a database.

2.11.2 Roles

Roles enable you to assemble users into a single unit to which you can apply permissions. SQL Server provides predefined server and database roles for common administrative functions so that you can easily grant a selection of administrative permissions to a particular user. You also can create your own user-defined database roles. In SQL Server, users can belong to multiple roles.



2.11.3 Types of Roles

SQL Server enables three types of roles to help manage permissions: fixed server roles, fixed database roles, and user-defined database roles.

2.11.3.1 Fixed Server Role

Fixed server roles provide groupings of administrative privileges at the server level. They are managed independently of user databases at the server level.

2.11.3.2 Fixed Database Roles

Fixed database roles provide groupings of administrative privileges at the database level. The following table describes the fixed database roles in SQL Server 2000.

2.11.3.3 User-defined Database Roles

You also can create your own database roles to represent work performed by a group of employees in your organization. You do not have to grant and revoke permissions from each person. If the function of a role changes, you easily can change the permissions for the role and have the changes apply automatically to all members of the role.

2.11.4 Permission Validation

Within each database, you assign permissions to user accounts and roles to perform (or restrict) certain actions. SQL Server accepts commands after a user has successfully accessed a database. SQL Server takes the following steps when validating permissions: When the user performs an action, such as executing a Transact-SQL statement or choosing a menu option, the client sends Transact-SQL statements to SQL Server.

1. When SQL Server receives a Transact-SQL statement, it checks that the user has permission to execute the statement.
2. SQL Server then performs one of two actions:
 - If the user does not have the proper permissions, SQL Server returns an error.
 - If the user has the proper permissions, SQL Server performs the action.



2.12 WORKING WITH SQL SERVER

2.12.1 Administering a SQL Server Database

SQL Server provides graphical and command prompt tools and utilities to administer SQL Server. It also includes different types of Help to assist you.

2.12.2 Common Administrative Tasks

Administering a SQL Server database involves:

- Installing, configuring, and securing SQL Server.
- Building databases. Tasks include allocating disk space to the database and log, transferring data into and out of the database, defining and implementing database security, creating automated jobs for repetitive tasks, and setting up replication to publish data to multiple sites.
- Managing ongoing activities, such as importing and exporting data, backing up and restoring the database and log, and monitoring and tuning the database. SQL Server includes tools and wizards for administering and managing the server, designing and creating databases, and querying data. It also provides online Help.

2.12.3 SQL Server Enterprise Manager

SQL Server provides an administrative client, SQL Server Enterprise Manager, which is a Microsoft Management Console (MMC) snap-in. MMC is a shared user interface for the management of Microsoft server applications.

2.12.4 SQL Server Administration Tools and Wizards

SQL Server provides a number of administrative tools and wizards that assist with particular aspects of its administration.

2.12.5 SQL Server Command Prompt Management Tools

SQL Server command prompt management tools allow you to enter Transact-SQL statements and execute script files.

2.12.6 SQL Server Help and SQL Server Books Online

SQL Server offers different types of Help to assist you.

2.12.7 Implementing a SQL Server Database

Implementing a SQL Server database means planning, creating, and maintaining a number of interrelated components. The nature and complexity of a database application, as well as the process of planning it, can vary greatly. For example, a database can be relatively simple, designed for use by a single person, or it can be large and complex, designed to handle all the banking transactions for hundreds of thousands of clients. Regardless of the size and complexity of the database, implementing it usually involves:

- Designing the database so that your application uses hardware optimally and allows for future growth; identifying and modeling database objects and application logic; and specifying the types of information for each object and type of relationship.
- Creating the database and database objects, including tables, data integrity mechanisms, data entry and retrieval objects (often stored procedures), appropriate indexes, and security.
- Testing and tuning the application and database. When you design a database, you want to ensure that the database performs important functions correctly and quickly. In conjunction with correct database design, the correct use of indexes, RAID, and filegroups are essential to achieving good performance.
- Planning deployment, which includes analyzing the workload and recommending an optimal index configuration for your SQL Server database.

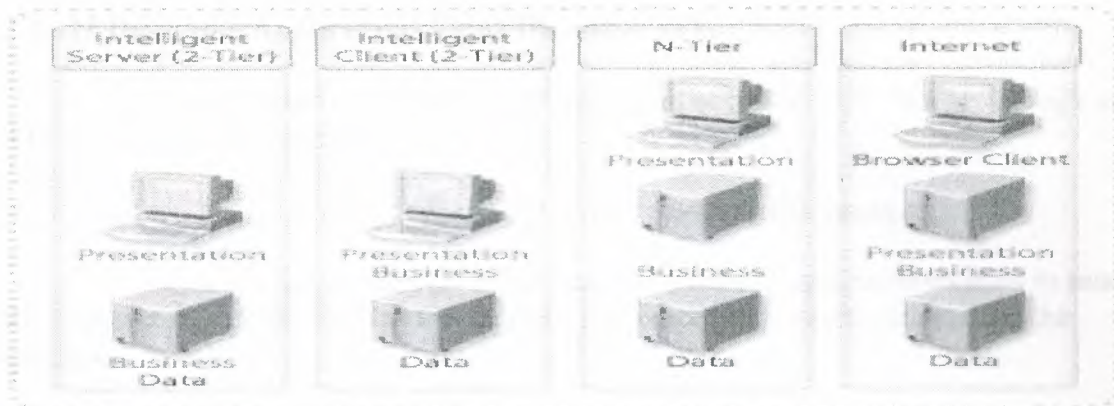
2.12.8 Selecting an Application Architecture for SQL Server

Planning a database design requires knowledge of the business functions that you want to model and the database concepts and features that you use to represent those business functions.

Before you design an application for SQL Server, it is important to take time designing a database to model the business accurately. A well-designed database requires fewer changes and generally performs more efficiently. The architecture that you select affects how you develop, deploy, and manage your software application.

2.12.9 Software Architecture

You can use one of several application architectures to implement client/server applications. However, selecting a layered application approach affords flexibility and a choice of management options. You can divide software applications into three logical layers, which can physically reside on one or more servers.



2.12.10 Architectural Design

Typical application deployment options include:

2.12.10.1 Intelligent Server (2-Tier) Most processing occurs on the server, with the client handling presentation services. In many instances, most of the business services logic is implemented in the database. This design is useful when clients do not have sufficient resources to process the business logic. However, the server can become a bottleneck because database and business services compete for the same hardware resources. Corporate applications designed from a database-centric point of view are an example of this design.

2.12.10.2 Intelligent Client (2-Tier) Most processing occurs on the client, with the server handling data services. This design is widely used. However, network traffic can be heavy and transactions longer, which can affect performance. Applications developed for small organizations with products such as Microsoft Access are an example of this design.

2.12.10.3 N-Tier Processing is divided among a database server, an application server, and clients. This approach separates logic from data services, and you easily can add more application servers or database servers as needed. However, the potential for complexity increases, and this approach may be slower for small applications. Multitiered enterprise applications and applications developed with transaction processing monitors are examples of this design.

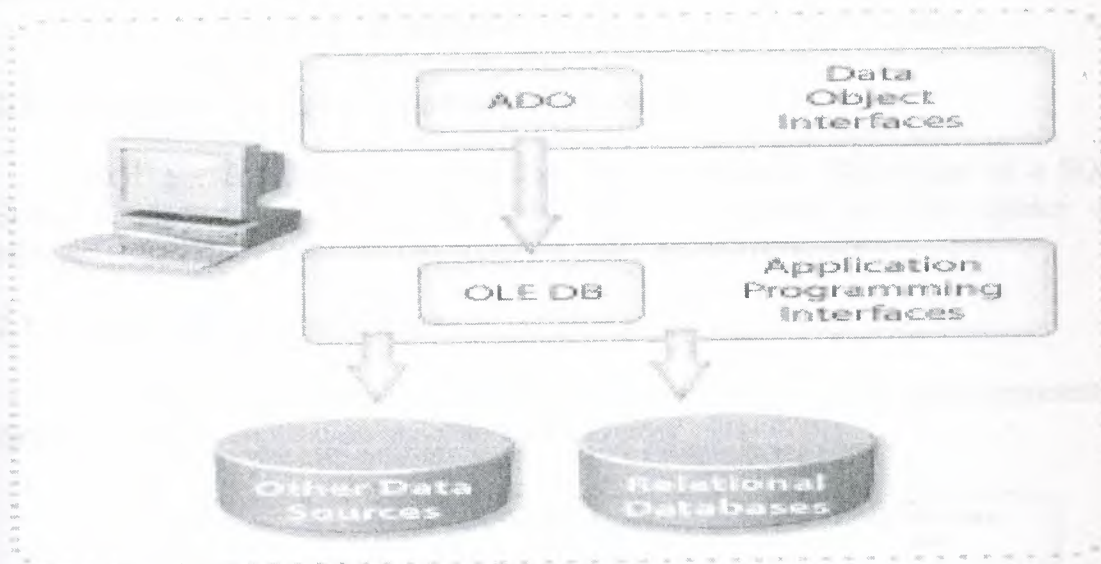
2.12.10.4 Internet Processing is divided into three layers, with the business and presentation services residing on the Web server and the clients using Internet browsers. SQL Server uses XML support for presentation of data to browsers. SQL Server can support any client that has a browser, and software does not need to be maintained on the client. An example of this design is a Web site that uses several Web servers to manage connections to clients and a single SQL Server database that services requests for data. You can access SQL Server over HTTP by using a URL. This allows you to directly access database objects and execute template files. This is not recommended for environments that must be highly secure and in which performance is critical.

2.12.11 Designing Applications Using Database APIs

You can develop a database application that accesses SQL Server through an API. A database API contains two parts:

- Transact-SQL language statements passed to the database.
- A set of functions or object-oriented interfaces and methods used to send the Transact-SQL statements to the database and process the results returned by the database.

Examples of relational database applications include data entry applications for airline ticketing and banking transaction systems.



2.13 OLE DB

OLE DB is a Component Object Model (COM)-based API. This API is a library of COM interfaces that enables universal access to diverse data sources.

SQL Server includes a native OLE DB provider. The provider supports applications written by using OLE DB, or other APIs that use OLE DB, such as ADO. Through the native provider, SQL Server also supports objects or components using OLE DB, such as ActiveX, ADO, or Microsoft .NET Enterprise Servers.

2.14 ADO

This database API defines how to write an application to connect to a database by using OLE DB and how to pass Transact-SQL commands to a database. ADO is an application-level interface that uses OLE DB. Because ADO uses OLE DB as its foundation, it benefits from the data access infrastructure that OLE DB provides, yet it shields the application developer from the necessity of programming COM

interfaces. Developers can use ADO for general-purpose access programs in business applications (Accounting, Human Resources, and Customer Management), and can use OLE DB for tool, utility, or system-level development (development tools and database utilities).

2.15 OVERVIEW OF PROGRAMMING SQL SERVER

After completing this part, we will be able to:

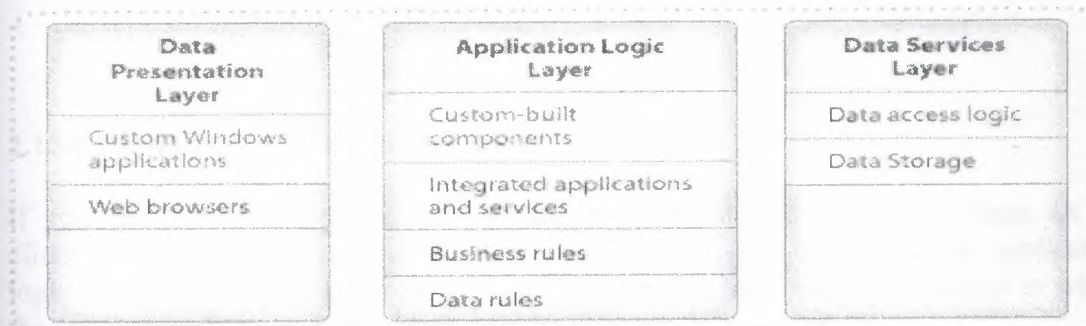
- Describe the concepts of enterprise-level application architecture.
- Describe the primary Microsoft® SQL Server™ 2000 programming tools.
- Explain the difference between the two primary programming tools in SQL Server.
 - Describe the basic elements of Transact-SQL.
 - Describe the use of local variables, operators, functions, control of flow statements, and comments.
 - Describe the various ways to execute Transact-SQL statements.

2.15.1 Designing Enterprise Application Architecture

SQL Server is often part of a distributed application. The design of a SQL Server implementation for an enterprise solution depends on your choice of architecture and how you intend to distribute logic across applications

2.15.2 Identifying Logical Layers

Enterprise application architecture contains logical layers. The layers represent data presentation, application logic, and data services.



2.15.3 Data Presentation Layer

The data presentation layer is also referred to as user services and allows users to browse and manipulate data. The two main types of client applications are custom Microsoft Windows® applications and Web browsers. The data presentation layer uses the services that the application logic layer provides.

2.15.4 Application Logic Layer

This layer contains the application logic that defines rules and processes. It allows for scalability; instead of many clients directly accessing a database (with each client

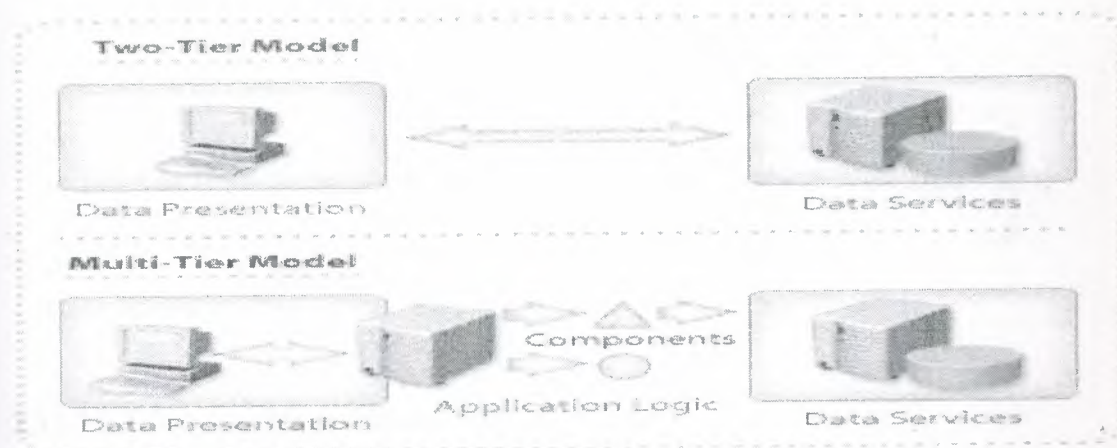
requiring a separate connection), clients can connect to business services that, in turn, connect to the data servers. Business services can be custom-built components or integrated applications and services, such as Web services. The application logic layer can also contain components that make use of transaction services, messaging services, or object and connection management services.

2.15.5 Data Services Layer

Data services include data access logic and data storage. These services can include SQL Server stored procedures to manage data traffic and integrity on the database server.

2.15.6 Designing Physical Layers

You can physically place logical layers in a distributed environment in a variety of ways. Although all logical layers can exist on one computer, it is typical to distribute the logical layers in a two-tier or multi-tier model. This allows you to implement logic, business rules, and processing where they are most effective.



2.15.6.1 Using a Two-Tier Model

If you use this model, you can locate the presentation and application logic on the client and the data services on a server. Alternatively, you can locate the application logic in stored procedures on the server. You can also have a mixed solution in which the application logic is divided between the client and the server.

Two-tier designs are less common than multi-tier designs, due to the growing popularity of Internet applications. They are not as scalable and may not be as easy to maintain as multi-tier designs are.

2.15.6.2 Using a Multi-Tier Model

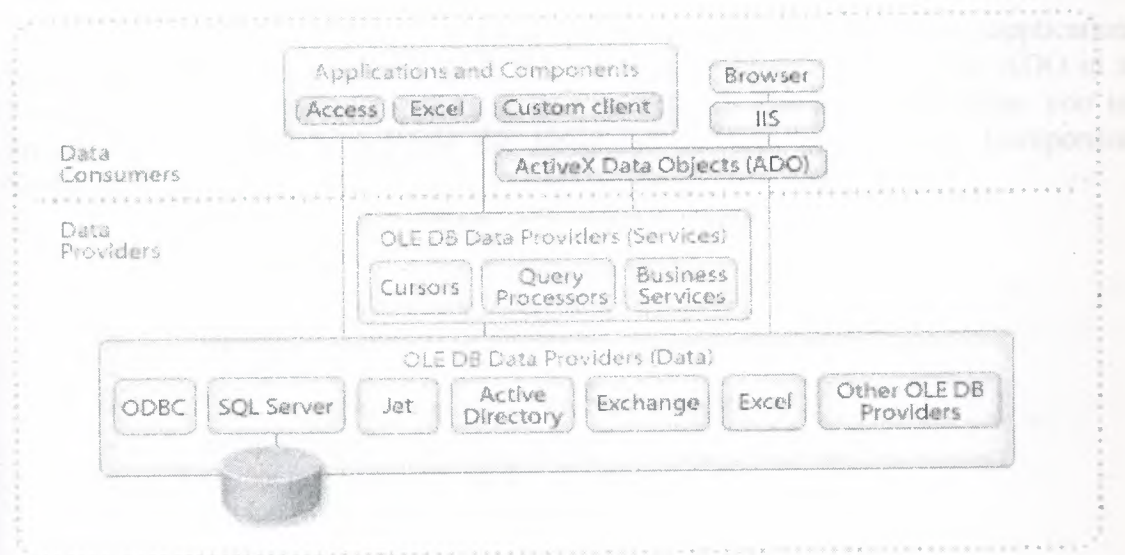
The multi-tier model, also known as three-tier or n-tier, allows you to distribute logic across applications. Business rules can be separate from the client or the database. When this model is applied to the Internet, you can divide presentation services between a browser client and a Microsoft Internet Information Services (IIS) Web

server; the Web server formats the Web pages that the browser displays.

The multi-tier model is scalable for large client bases and many applications, and you can spread the workload among many computers. A multi-tier model is easy to manage because you can isolate a change to one business rule without affecting others. Also, an update to an Active Server Page on a Web server automatically updates all clients.

2.15.7 Accessing Data

Microsoft technologies allow you to access enterprise data by using a wide range of pre-built clients or custom clients that use a data access-programming interface.



2.15.7.1 Using Pre-Built Clients

You can use pre-built client applications to access data on SQL Server. The data retrieval logic is part of the client application.

Microsoft Office 2000 includes Microsoft Access and Microsoft Excel. When part of a multi-tier solution, you use these applications primarily for presentation services. However, you can also use them for application logic and data services. These applications allow users to browse server-side data and perform ad hoc queries. You can use them to retrieve SQL Server data or as a client in a multi-tier design. You can also use Office 2000 as a development environment for building data access applications. Access and Excel are examples of pre-built clients that offer a range of functionality. You can also use pre-built clients that only offer presentation services, such as a browser that communicates with IIS.

2.15.7.2 Building Custom Clients

You can build custom clients by using a data access programming interface and a development environment, such as Microsoft Visual Studio® version 6.0 Enterprise Edition.

2.15.8 Accessing Data

2.15.8.1 Providing Universal Data Access

Custom clients may need to access many different data sources in the enterprise. Microsoft Data Access Components (MDAC) is an interface that allows communication with different data sources. You can use the following MDAC components to facilitate communication:

- *OLE DB*. A set of Component Services interfaces that provides uniform access to data stored in diverse information sources. OLE DB enables you to access relational and nonrelational data sources.
- *Microsoft ActiveX® Data Objects (ADO)*. An easy-to-use application programming interface (API) to any OLE DB data provider. You can use ADO in a broad range of data access application scenarios. OLE DB and ADO allow you to create data components that use the integrated services provided by Component Services.

ADO allows you to:

- Open and maintain connections.
- Create ad hoc queries.
- Execute stored procedures on SQL Server.
- Retrieve results and use cursors.
- Cache query results on the client.
- Update rows in the database.
- Close connections.

2.16 SQL SERVER PROGRAMMING TOOLS

SQL Server 2000 offers several programming tools, including SQL Query Analyzer and the **osql** utility. SQL Query Analyzer is a Windows-based application, and **osql** is a utility that you can run from a command prompt.

2.16.1 SQL Query Analyzer

You can use SQL Query Analyzer to view query statements and results at the same time. You also can use it for writing, modifying, and saving Transact-SQL scripts.

SQL Query Analyzer provides the following features:

- Customized marking of syntax elements. As you write a query, SQL Query Analyzer highlights keywords, character strings, and other language elements; you can customize how they appear.
- Multiple query windows, each with its own connection.
- Customizable views of result sets. You can view results in default result set form or in a grid so that you can manipulate them as you would a table.

- Graphical execution plans that describe how SQL Server executes the query. You can view the optimized plan of execution and verify your syntax.

- The ability to execute portions of a script. You can select portions of a script, and SQL Server will execute only those portions.

2.16.2 osql Utility

The **osql** utility allows you to write Transact-SQL statements, system procedures, and script files. It uses Open Database Connectivity (ODBC) to communicate with the server. You start the utility directly from the operating system with the case-sensitive arguments listed below. Once started, **osql** accepts Transact-SQL statements and sends them to SQL Server interactively. **Osql** formats and displays the results on the screen. Use the QUIT or EXIT commands to exit from **osql**.

Syntax

```
osql -U login_id [-e] [-E] [-p] [-n] [-d db_name] [-q "query"] [-Q "query"]
[-c cmd_end] [-h headers] [-w column_width] [-s col_separator]
[-t time_out] [-m error_level] [-L] [-?] [-r {0 | 1}]
[-H wksta_name] [-P password] [-R]
[-S server_name] [-i input_file] [-o output_file] [-a packet_size]
[-b] [-O] [-l time_out]
```

The following table describes the most commonly used arguments.

Argument	Description
-U login_id	Is the user login ID. Login IDs are case sensitive. If neither the -U or -P option is used, SQL Server uses the currently logged in user account and will not prompt for a password.
-E	Uses a trusted connection instead of requesting a password.
-?	Displays the syntax summary of osql switches.
-P password	Is a user-specified password. If the -P option is not used, osql prompts for a password. If the -P option is used at the end of the command prompt without any password, osql uses the default password (NULL). Passwords are case sensitive. If neither the -U or -P option is used, SQL Server uses the currently logged in user account and will not prompt for a password.

-S server_name

Specifies the SQL Server to which to connect. `server_name` is the name of the server computer on the network. This option is required if you execute **osql** from a remote computer on the network.

-i input_file

Identifies the file that contains a batch of Transact-SQL statements or stored procedures. You can use the less than (<) symbol instead of **-i**.

-o output_file

Identifies the file that receives output from **osql**. You can use the greater than (>) symbol in place of **-o**. If the input file is Unicode, the output file will be Unicode if you specify **-o**. If the input file is not Unicode, the output file is OEM.

-b

Specifies that **osql** exits and returns a Microsoft MS-DOS® ERRORLEVEL value when an error occurs. The value returned to the DOS ERRORLEVEL variable is 1 when the SQL Server error message has a severity of 10 or greater; otherwise, the value returned is 0. MS-DOS batch files can test the value of DOS ERRORLEVEL and handle the error appropriately.

2.17 THE TRANSACTION-SQL PROGRAMMING LANGUAGE

Transact-SQL is the SQL Server implementation of the entry-level ANSI-SQL International Standards Organization (ISO) standard. The ANSI-SQL compliant language elements of Transact-SQL can be executed from any entry-level ANSI-SQL compliant product. Transact-SQL also contains additional language elements that are unique to it.

2.17.1 Elements of Transact-SQL

As you write and execute Transact-SQL statements, you will use different languages statements, which are used to determine who can see or modify the data, create objects in the database, and query and modify the data. You should follow the rules for naming SQL Server objects, and become familiar with the naming guidelines for database objects.

In this part I try to explain will learn about the following topics:

- Data control language statements
- Data definition language statements

- Data manipulation language statements
- SQL server object names
- Naming guidelines

2.17.2 Data Control Language Statements

You use Data Control Language (DCL) statements to change the permissions associated with a database user or role.

By default, only members of the **sysadmin**, **dbcreator**, **db_owner**, or **db_securityadmin** role can execute DCL statements.

Example

This example grants the **public** role permission to query the **Products** table.

```
USE Northwind
GRANT SELECT ON Products TO public
```

2.17.3 Data Definition Language Statements

Data Definition Language (DDL) statements define the database by creating databases, tables, and user-defined data types. You also use DDL statements to manage your database objects. Some DDL statements include:

- **CREATE** *object_type object_name*.
- **ALTER** *object_type object_name*.
- **DROP** *object_type object_name*.

By default, only members of the **sysadmin**, **dbcreator**, **db_owner**, or **db_ddladmin** role can execute DDL statements. In general, it is recommended that no other accounts be allowed to create database objects. If users create their own objects in databases, then each object owner is required to grant the proper permissions to each user of those objects. This causes an administrative burden and should be avoided. Restricting statement permissions to these roles also avoids problems with object ownership that can occur when an object owner has been dropped from a database, or when the owner of a stored procedure or view does not own the underlying tables. If multiple user accounts create objects, the **sysadmin** and **db_owner** roles can use the **SETUSER** function to impersonate other users or the **sp_changeobjectowner** system stored procedure to change the owner of an object.

Example

The following script creates a table called **Client** in the **ClassNorthwind** database. It includes **CustomerID**, **Company**, **Contact**, and **Phone** columns.

```
USE ClassNorthwind
CREATE TABLE Client
```


(CustomerID int, Company varchar(40), Contact varchar(30), Phone char(12))

2.17.4 Data Manipulation Language Statements

DML statements work with the data in the database. By using DML statements, you can change data or retrieve information. DML statements include:

- SELECT
- INSERT
- UPDATE
- DELETE

By default, only members of the **sysadmin**, **dbcreator**, **db_owner**, **db_datawriter**, and **db_datareader** roles can execute DML statements.

Example

This example retrieves the category ID, product name, product ID, and unit price of the products in the **Northwind** database.

```
SELECT CategoryID, ProductName, CategoryID, ProductID, UnitPrice FROM
Northwind..Products
```

2.17.5 SQL Server Object Names

SQL Server provides a series of standard naming rules for object identifiers and a method of using delimiters for identifiers that are not standard. It is recommended that you name objects by using the standard identifier characters, if possible.

Standard Identifiers

Standard identifiers can contain from one to 128 characters, including letters, symbols (`_`, `@`, or `#`), and numbers. No embedded spaces are allowed in standard identifiers. You should observe the following rules for using identifiers:

- The first character must be an alphabetic character of a-z or A-Z.
- After the first character, identifiers can include letters, numerals, or the `@`, `$`, `#`, or `_` symbol.
 - Identifier names starting with a symbol have special uses:
 - An identifier beginning with the at sign (`@`) denotes a local variable or parameter.
 - An identifier beginning with a number sign (`#`) denotes a temporary table or procedure.
 - An identifier beginning with a double-number sign (`##`) denotes a global temporary object.

2.17.6 SQL Server Object Names

Delimited Identifiers

If an identifier complies with all of the rules for the format of identifiers, you can use it with or without delimiters. If an identifier does not comply with one or more of the rules for the format of identifiers, it must always be delimited.

You can use delimited identifiers in the following situations:

- When names contain embedded spaces
- When reserved words are used for object names or portions of object names. You must enclose delimited identifiers in brackets or quotation marks when you use them in Transact-SQL statements.
- Bracketed identifiers are delimited by square brackets ([]):

```
SELECT * FROM [Blanks In Table Name]
```

Note: You can always use bracketed delimiters, regardless of the status of the SET QUOTED_IDENTIFIER option.

- Quoted identifiers are delimited by quotation marks (""):

```
SELECT * FROM "Blanks in Table Name"
```

You can use quoted identifiers only if the SET QUOTED_IDENTIFIER option is on.

Naming Guidelines

Guidelines for naming database objects are important for identifying the type of object and to promote ease in troubleshooting or debugging. When naming database objects, you should:

- Use meaningful names where possible.

For example, for a column that contains the name of customers, you could name the column **Chr_Name_Of_Customer**. A prefix of **Chr** in the column name denotes a **character** data type.

- Keep names short.

For example, although the column name **Chr_Name_Of_Customer** is meaningful, you could shorten the column name to **Name** or **Chr_Name**.

- Use a clear and simple naming convention.

Decide what works best for your situation, and be consistent. Avoid naming conventions that are too complex, because they can become difficult to remember. For

example, you can remove vowels if an object name must resemble a keyword (such as a backup stored procedure named **Bckup**).

- Chose an identifier that distinguishes the type of object, especially when using views and stored procedures.

System administrators often mistake views for tables, an oversight that can cause unexpected problems. For example, if you create a view that joins two tables, you could name that view, **SoldView**.

- Keep object names and user names unique.

For example, avoid creating a **Sales** table and a **sales** role in the same database.

2.18 ADDITIONAL LANGUAGE ELEMENTS

2.18.1 Local Variables

Variables are language elements with assigned values. You can use local variables in Transact-SQL.

You define a local variable in a DECLARE statement and then assign it an initial value with either the SET or SELECT statement. Use the SET statement when the desired value is known. Use the SELECT statement when you must look up the desired value in a table. After you establish the value of the variable, you can use it in the statement, batch, or procedure in which it was declared. A batch is a set of Transact-SQL statements that are submitted together and executed as a group. A local variable is shown with one at sign (@) preceding its name.

Syntax

```
DECLARE {@local_variable data_type} [...n]
```

```
SET @local_variable_name = expression
```

Example

The following example declares two variables. It uses the SET statement to establish the value of the @vLastName variable and the SELECT statement to look up the value of the @vFirstName variable. It then prints both variables.

```
DECLARE @vLastName char(20),  
        @vFirstName varchar(11)  
SET @vLastName = 'Dodsworth'  
SELECT @vFirstName = FirstName
```

```
FROM Northwind..Employees
WHERE LastName = @vLastName
PRINT @vFirstName + ' ' + @vLastNameGO
```

Result

Anne Dodsworth

2.18.2 Operators

Operators are symbols that perform mathematical computations, string concatenations, and comparisons between columns, constants, and variables. You can combine them and use them in search conditions. When you combine them, the order in which SQL Server processes the operators is based on a predefined precedence.

Partial Syntax

```
{constant | column_name | function | (subquery)}
  [{arithmetic_operator | string_operator |
  AND | OR | NOT}
  {constant | column_name | function | (subquery)}...]
```

2.18.3 Types of Operators

SQL Server supports four types of operators: arithmetic, comparison, string concatenation, and logical

2.18.3.1 Arithmetic

Arithmetic operators perform computations with numeric columns or constants. Transact-SQL supports multiplicative operators, including multiplication (*), division (/), and modulo (%)—the integer remainder after integer division—and the addition (+) and subtraction (-) additive operators.

2.18.3.2 Comparison

Comparison operators compare two expressions. You can make comparisons between variables, columns, and expressions of similar type.

2.18.3.3 String Concatenation

The string concatenation operator (+) concatenates string values. String functions handle all other string manipulation.

2.18.3.4 Logical

The logical operators AND, OR, and NOT connect search conditions in WHERE clauses.

2.18.4 Operator Precedence Levels

If you use multiple operators (logical or arithmetic) to combine expressions, SQL Server processes the operators in order of their precedence, which may affect the resulting value.

SQL Server handles the most deeply nested expression first. In addition, if all arithmetic operators in an expression share the same level of precedence, the order is from left to right.

2.18.5 Functions

Transact-SQL provides many functions that return information. Functions take input parameters and return values that can be used in expressions. The Transact-SQL programming language provides three types of functions, aggregate, scalar, and rowset.

2.18.5.1 Aggregate Functions

Aggregate functions operate on a collection of values but return a single, summarizing value.

Example 1

This example determines the average of the UnitPrice column for all products in the Products table.

```
SELECT AVG(UnitPrice) FROM Products
```

2.18.5.2 Scalar Functions

Scalar functions operate on a single value and then return a single value. You can use these functions wherever an expression is valid. You can group scalar functions into the categories in the following table.

Example 2

This metadata function example returns the name of the database currently in use.

```
SELECT DB_NAME() AS 'database'
```

2.18.5.3 Rowset Functions

Rowset functions can be used like table references in a Transact-SQL statement.

Example 3

This example performs a distributed query to retrieve information from the **EMP** table.

```
SELECT *
FROM OPENQUERY(OracleSvr, 'SELECT ENAME, EMPNO FROM
SCOTT.EMP')
```

2.18.5.4 Convert Functions

You commonly use functions when converting date data from the format of one country to that of another.

Note: To change date formats, you should use the CONVERT function with the style option to determine the date format that will be returned.

Example 4

This example demonstrates how you can convert dates to different styles.

```
SELECT 'ANSI:' AS Region,
       CONVERT (varchar(30), GETDATE(), 102) AS Style
UNION
SELECT 'European:', CONVERT(vchar(30), GETDATE(), 113)
UNION
SELECT 'Japanese:', CONVERT(vchar(30), GETDATE(), 111)
```

2.18.5.5 Date Functions

This example uses the DATEFORMAT option of the SET statement to format dates for the duration of a connection. This setting is used only in the interpretation of character strings as they are converted to date values and has no effect on the display of date values.

Example 5

```
SET DATEFORMAT dmy

GO
DECLARE @vdate datetime

SET @vdate = '29/11/00'

SELECT @vdate
```

2.18.5.6 User Functions

This example returns the current user name and the application that the user is using for the current session or connection. The user in this example is a member of the **sysadmin** role.

Example 6

```
USE Northwind  
SELECT user_name(), app_name()
```

2.18.5.7 Column Property Functions

This example determines whether the **FirstName** column in the **Employees** table of the **Northwind** database allows null values.

A result of zero (false) means that null values are not allowed, and a result of one (true) means that null values are allowed. Notice that the **OBJECT_ID** function is embedded in the **COLUMNPROPERTY** function. This allows you to retrieve the **object id** of the **Employees** table.

Example 7

```
USE Northwind  
  
SELECT COLUMNPROPERTY(OBJECT_ID('Employees'),  
'FirstName', 'AllowsNull')
```

2.18.6 Control of Flow Language Elements

Transact-SQL contains several language elements that control the flow of logic in a statement. It also contains the **CASE** expression that allows you to use conditional logic on one row at a time in a **SELECT** or **UPDATE** statement.

2.18.7 Statement Level

The following language elements enable you to control the flow of logic in a script:

BEGIN...END Blocks These elements enclose a series of Transact-SQL statements so that SQL Server treats them as a unit.

IF...ELSE Blocks These elements specify that SQL Server should execute the first alternative if a certain condition is true. Otherwise, SQL Server should execute the second alternative.

WHILE Constructs These elements execute a statement repeatedly as long as the specified condition is true. BREAK and CONTINUE statements control the operation of the statements inside a WHILE loop.

Example 1

This example determines whether a customer has any orders before deleting the customer from the customer list.

```
USE Northwind

IF EXISTS (SELECT OrderID FROM Orders
WHERE CustomerID = 'Frank')

PRINT '*** Customer cannot be deleted ***'

ELSE

BEGIN

DELETE Customers WHERE CustomerID = 'Frank'

PRINT '*** Customer deleted ***'

END
```

2.18.8 Row Level

A CASE expression lists predicates, assigns a value for each, and then tests each one. If the expression returns a true value, the CASE expression returns the value in the WHEN clause. If the expression is false, and you have specified an ELSE clause, SQL Server returns the value in the ELSE clause. You can use a CASE expression anywhere that you use an expression.

Syntax

CASE *expression*

{WHEN *expression* THEN *result*} [,...n]

[ELSE *result*]

END

Example

The following example reviews the inventory status of products in the **Products** table and returns messages based on the quantities available and quantities back ordered, and whether the product has been discontinued.

```
SELECT ProductID, 'Product Inventory Status' =  
  
CASE  
  
WHEN (UnitsInStock < UnitsOnOrder AND Discontinued = 0)  
  
    THEN 'Negative Inventory - Order Now!'  
  
WHEN ((UnitsInStock-UnitsOnOrder) < ReorderLevel AND  
  
    Discontinued = 0)  
  
    THEN 'Reorder level reached- Place Order'  
  
WHEN (Discontinued = 1) THEN '***Discontinued***'  
  
ELSE 'In Stock'  
  
END  
FROM Northwind..Products
```

2.18.9 Comments

Comments are non-executing strings of text placed in statements to describe the action that the statement is performing or to disable one or more lines of the statement. You can use comments in one of two ways-in line with a statement, or as a block.

2.18.9.1 In-Line Comments

You can create in-line comments by using two hyphens (--) to set a comment apart from a statement. Transact-SQL ignores text to the right of the comment characters. You can also use this commenting character to disable lines of a statement.

Example 1

This example uses an in-line comment to explain what a calculation is doing.

```
SELECT ProductName ,(UnitsInStock + UnitsOnOrder) AS Max - inventory ,  
SupplierID FROM Products
```

Example 2

This example uses a second set of in-line comments, as represented by the second set of hyphens (--), to prevent the execution of a section (SupplierID) of a statement.

```
SELECT ProductName ,(UnitsInStock + UnitsOnOrder) AS Max -inventory  
--,SupplierID FROM Products
```

2.18.9.2 Block Comments

You can create multiple line blocks of comments by placing one comment character (/*) at the start of the comment text, typing your comments, and then concluding the comment with a closing comment character (*/). Use this character designator to create one or more lines of comments or comment headers-descriptive text that documents the statements that follow it. Comment headers often include the author's name, creation and last modification dates of the script, version information, and a description of the action that the statement performs.

Example 3

This example shows a comment header that spans several lines. The two asterisks (**) preceding each line improve readability.

```
/*  
** This code retrieves all rows of the products table  
** and displays the unit price, the unit price increased  
** by 10 percent, and the name of the product.  
*/  
SELECT UnitPrice, (UnitPrice * 1.1), ProductName  
FROM Products
```

Example 4

This section of a script is commented to prevent it from executing. This can be helpful when debugging or troubleshooting a script file.

```
/*  
DECLARE @v1 int  
SET @v1 = 0  
WHILE @v1 < 100  
    BEGIN  
        SELECT @v1 = (@v1 + 1)  
        SELECT @v1  
    END  
*/
```


2.19 WAYS TO EXECUTE TRANSACTION-SQL STATEMENTS

2.19.1 Dynamically Constructing Statements

You can build statements dynamically so that they are constructed at the same time that SQL Server executes a script. To build a statement dynamically, use the EXECUTE statement with a series of string literals and variables that are resolved at execution time.

Dynamically constructed statements are useful when you want SQL Server to assign the value of the variable when it executes the statement. For example, you can create a dynamic statement that performs the same action on a series of database objects.

Syntax

EXECUTE ({@str_var | 'tsql_string'} + [{@str_var | 'tsql_string'}...])
You set options dynamically, and variables and temporary tables that you create dynamically last only as long as it takes for SQL Server to execute the statement.

Consider the following facts about the EXECUTE statement:

- The EXECUTE statement executes statements composed of character strings in a Transact-SQL batch. Because these are string literals, be sure that you add spaces in the appropriate places to ensure proper concatenation.
- The EXECUTE statement can include a string literal, a string local variable, or a concatenation of both.
- All items in the EXECUTE string must consist of character data; you must convert all numeric data before you use the EXECUTE statement.
- You cannot use functions to build the string for execution.
- You can create any valid Transact-SQL statements dynamically, including functions
- You can nest EXECUTE statements.

Example 1

This example demonstrates how you can use a dynamically executed statement to specify a database context other than the one you are currently in, and then use it to select all of the columns and rows from a specified table. In this example, the change of the database context to the **Northwind** database lasts only for the duration of the query. The current database context is unchanged. By using a stored procedure, the user could pass the database and table information into the statement as parameters, and then query a specific table in a database.

```
DECLARE @dbname varchar(30), @tablename varchar(30)
```

```
SET @dbname = 'Northwind'
```

```
SET @tablename = 'Products'
```

```
EXECUTE
```

```
('USE ' + @dbname + ' SELECT ProductName FROM ' + @tablename)
```

Result

ProductName

Alice Mutton
Aniseed Syrup
Boston Crab

Meat

...

Example 2

This example demonstrates how you can use a dynamically executed statement to change a database option for the duration of the statement. The following statement does not return a count of the number of rows affected.

```
EXECUTE ('SET NOCOUNT ON ' + 'SELECT LastName, ReportsTo  
FROM Employees WHERE ReportsTo IS NULL')
```

Result

LastName	ReportsTo
Fuller	NULL

Using Batches

You can also submit one or more statements in a batch.

One or More Transact-SQL Statements Submitted Together

Batches can be run interactively or as part of a script. A script can include more than one batch of Transact-SQL statements.

2.19.2 Define a Batch by Using the GO Statement

Use a GO statement to signal the end of a batch. GO is not a universally accepted Transact-SQL statement; only SQL Query Analyzer and the **osql** utility accept it. Applications based on the ODBC or OLE DB APIs generate a syntax error if they attempt to execute a GO statement.

2.19.3 How SQL Server Processes Batches

SQL Server optimizes, compiles, and executes the statements in a batch together. However, the statements do not necessarily execute as a recoverable unit of work.

The scope of user-defined variables is limited to a batch, so a variable cannot be referenced after a GO statement.

2.19.4 You Cannot Combine Some Statements in a Batch

SQL Server must execute certain object creation statements in their own batches in a script, because of the way that the objects are defined. Each of the following statements is defined by including an object definition header followed by the AS keyword (indicating that one or more statements follow). The object definitions are delimited by the GO statement; SQL Server recognizes the end of the object definition when it reaches the GO statement:

- CREATE PROCEDURE
- CREATE VIEW
- CREATE TRIGGER
- CREATE RULE
- CREATE DEFAULT

Example 1

If you want to use more than one of the non-combinable statements, you must submit multiple batches, as the following script indicates.

```
CREATE DATABASE ...  
CREATE TABLE ...  
GO
```

```
CREATE VIEW1 ...  
GO  
CREATE VIEW2 ...  
GO
```

Example 2

The following example is a batch that fails. To execute it correctly, insert a GO statement before each CREATE TRIGGER statement.

```
CREATE DATABASE ...  
CREATE TABLE ...  
CREATE TRIGGER ...  
CREATE TRIGGER ...  
GO
```

Example 3

The following example shows how to group the statements of Example 2 so that they execute correctly.

```
CREATE DATABASE ...  
    CREATE TABLE ...  
GO
```

```
CREATE TRIGGER ...  
GO
```

```
CREATE TRIGGER ...  
GO
```

2.19.4.1 Using Scripts

Scripts are one of the most common ways to execute Transact-SQL statements. A script is one or more Transact-SQL statements that are saved as a file. You can write and save scripts in SQL Query Analyzer or in any text editor, such as Notepad. Save the script file by using the .sql file name extension.

You can open and execute the script file in SQL Query Analyzer or the **osql** utility (or another query tool). Saved scripts are very useful when recreating databases or data objects, or when you must use a set of statements repeatedly. Format Transact-SQL statements to be legible to others. Use indenting to indicate levels of relationships.

2.19.4.2 Using Transactions

Transactions, like batches, are groups of statements that are submitted as a set. However, SQL Server handles transactions as a single unit of work, and the transaction succeeds or fails as a whole. This process maintains data integrity. Transactions can span multiple batches.

Preface a transaction with a **BEGIN TRANSACTION** statement, and terminate it with a **COMMIT TRANSACTION** or **ROLLBACK TRANSACTION** statement. When a transaction is committed, SQL Server makes the changes to that transaction permanent. When a transaction is rolled back, SQL Server returns any rows affected by the transaction to their pretransaction states.

Partial Syntax

```
BEGIN TRANSACTION
```

```
COMMIT / ROLLBACK TRANSACTION
```

Example

In the following example, \$100 is debited from the savings account of customer

number 78910, and \$100 is credited to the customer's checking account. The customer transferred \$100 from savings to checking.

```
BEGIN TRANSACTION
  UPDATE savings
    SET balance = (amount - 100)
    WHERE custid = 78910
  IF @@ERROR <> 0
    BEGIN
      RAISERROR ('Transaction not completed due to
        savings account problem.', 16, -1)
      ROLLBACK TRANSACTION
    END
  UPDATE checking
    SET balance = (amount + 100)
    WHERE custid = 78910
  IF @@ERROR <> 0
    BEGIN
      RAISERROR ('Transaction not completed due to
        checking account problem.', 16, -1)
      ROLLBACK TRANSACTION
    END
  COMMIT TRANSACTION
```

2.19.4.3 Using XML

XML is a programming language that Web developers can use to present data from a SQL Server database to Web pages.

2.19.4.3.1 Allowing Client Browser to Format Data

When using the FOR XML clause in the SELECT statement, SQL Server:

- Returns the results of a query as a character string.
- Returns the attributes of the data, such as column and table names, as tags. A client browser can then use these tags to format the returned data.

2.19.4.3.2 Specifying the FOR XML AUTO Option

You can specify the FOR XML AUTO option to return query results in a standardized format. Each table in the FROM clause for which at least one column is listed in the SELECT clause is represented as an XML element. An element includes both data and attributes that describe the data.

Example 1

This example selects three columns from two joined tables. Notice that the results combine all of the columns into a single text string.

```

SELECT Orders.OrderID, Shippers.CompanyName, Orders.CustomerID

FROM Orders JOIN Shippers

ON Orders.shipvia = Shippers.ShipperID

WHERE OrderID < 10250

FOR XML AUTO

```

Result

XML_F52E2B61-18A1-11d1-B105-00805F49916B

```

-----
<Orders                OrderID="10248"                CustomerID="VINET">
  <Shippers            CompanyName="Federal            Shipping"/>
</Orders>
<Orders                OrderID="10249"                CustomerID="TOMSP">
  <Shippers CompanyName="Speedy Express"/></Orders>

```

2.19.4.3.3 Specifying the FOR XML RAW Option

In some cases, Web developers do not want the automatic formatting. You can specify the RAW option to transform each row in the result set into an XML element with a generic identifier row as the element tag.

Example 2

Compare the result from this example with that of Example 1. This example returns the same data, but the formatting is more generic. Notice that the tables are not named, and the columns are not grouped by table name.

```

SELECT Orders.OrderID, Shippers.CompanyName, Orders.CustomerID
FROM Orders JOIN Shippers
ON Orders.shipvia = Shippers.ShipperID
WHERE OrderID < 10250
FOR XML RAW

```

Result

XML_F52E2B61-18A1-11d1-B105-00805F49916B

```

-----
<row OrderID="10248"
CompanyName="Federal Shipping"
CustomerID="VINET"/>
<row OrderID="10249"
CompanyName="Speedy Express"
CustomerID="TOMSP"/>

```


2.19.4.3.4 Identifying Limitations of Using the FOR XML Clause

A SELECT statement that contains the FOR XML clause reformats the output for the SQL Server client. Because of these changes, you cannot use a query output in XML format as an input for further SQL Server processing.

You cannot use XML formatted output in:

- A nested SELECT statement.
- A SELECT INTO statement.
- A COMPUTE BY clause.
- Stored procedures that are called in an INSERT statement.
- A view definition or a user-defined function that returns a rowset.

CHAPTER THREE

SCREEN SHOT AND USER MANUAL

When you open the visual basic.net and select the care-repair program and run it, the program will greet with login page (figure 3.1). This page includes two verifying part.

Firstly you enter the right password and username. After that you should click to log in button to enter the program. Otherwise if you want to exit from program it is enough to clicking the exit button.

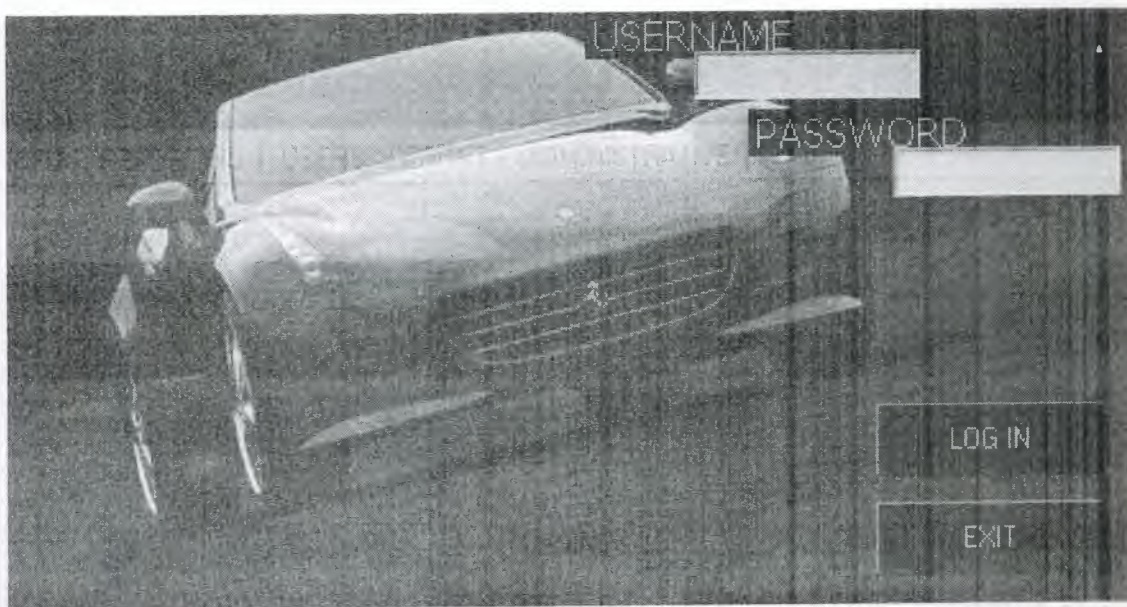


Figure 3.1

Welcome to home page. From this page you are able to all attain all things. To more speed I used to tab controls. When you clicking the any of these tab pages new page is opening as shown below figure 3.2

Figure 3.2

First tab page is care\repair tab page

Figure 3.3

On this page there are 3 groupbox. When you come the select groupbox there are two radio button to select the care type. firstly we click to periodic care. after that third groupbox is opened to select to standart care services and click to calculate button to find the done standart care services total and by entering the information of car and customer you save the info of service to your database with clicking 'save to database' button. When the entering the car information if car information was saved before customer name comes from database and written second textbox. Otherwise program

throw the message to ask for saving the all information about car and customer. If you click the the 'yes ' button ,service accepting page (figure 3.5) is opened. If you click 'no' process is continue without saving all information.

If you select the second radiobutton which is instant care other form is loaded.

Figure 3.4

As it became on the periodic care there are customer info's and done jobs menu. You enter the done job and its price and calculate button ,price seems on the total price textbox and entering the info about car by your hand as it became on the periodic care and then click the save button without any error all informations is successfully saved to database.

When you click the second tab page servise accepting is opened and first record seems as you see.this page includes all information about car. With 'find record ' button you check the whether the car info was saved or not. If it is the first coming to service info's entering.if it is second or more coming all info's seems on the textboxes. At the bottom of page there are four button to go around the records. There are four buttons at the right of the page to saving,cleaing,updating and deleting record.

CARE/REPAIR SERVICE ACCEPTING TAKING OUTSIDE STOCK INFO REPORTING USERS ACCOUNT ADMINISTRATIVE

PLAQUE: 34a34

CUSTOMER NO: 6

CUSTOMER NAME: alper koralus

TRADEMARK: leno

MODEL: megan

YEAR: 2001

COLOR: blue

POST CODE: 3434

REGION: avclar

CITY: istanbul

PHONE: 8761544

TAX NO: 11234

E-MAIL: 6

ACCOUNT OPEN: 25.06.2006

Figure 3.5

When you coming to third tab page (figure 3.6) the taking outside info's are coming. Because of this page using is when the car is coming to service and the car needs to some part and you don't have this part you have to take this part from outside and you have to under the save this info's for accounting your money.

CARE/REPAIR SERVICE ACCEPTING TAKING OUTSIDE STOCK INFO REPORTING USERS ACCOUNT ADMINISTRATIVE

PRODUCT NAME:

UNIT PRICE: YTL

QUANTITY:

TOTAL PRICE: YTL

FROM WHERE:

FOR WHICH CAR:

SELLING PRICE: YTL

DATE: 02.06.2006

PRO NAME	UNIT PRICE	QUANTITY	TOTAL PRICE	FROM WHERE	PRO WHICH CAR	SELLING PRICE	DATE
adems	55,0000	2	110,0000	kacem	43m33	500,0000	12.12.2006
alper	55,0000	2	110,0000	metrak	56f44	5666,0000	11.11.2006
kard	45,0000	2	90,0000	konya	34g44	100,0000	24.05.2006
asem	34,0000	2	68,0000	asaf	34f44	189,0000	24.05.2006
ali	22,0000	2	44,0000	izmir	23m44	120,0000	24.05.2006
www	1,0000	1	1,0000	asaf	12m44	2,0000	25.05.2006
kuruy	30,0000	2	60,0000	toknoren	12m44	70,0000	26.05.2006
buz	34,0000	3	102,0000	eror	12m44	110,0000	26.05.2006
kapak	2,0000	4	8,0000	kardem	43m33	10,0000	29.05.2006

Figure 3.6

On this page you either see the all taking outside or save the new part to database. When you want to list of the outside of the specific car, enter the number plate and click the button.

PRO NO	PRO NAME	UNIT PRICE	QUANTITY	TOTAL	FROM WHERE	DATE
1	kapak	22,000	2	44,000	erciklar	26.05.2006 00
2	okle	31,000	3	93,000	karalar	26.05.2006 00
3	buji	23,000	2	46,000	karalar	29.05.2006 00
4	balata	32,000	2	64,000	ademden	29.05.2006 00
5	marş dinamo	35,000	5	175,000	ademden	29.05.2006 00
6	aks	44,000	5	220,000	ademden	29.05.2006 00
7	fren tel	66,000	4	264,000	ahmet	23.05.2006 00
8	conta	23,000	3	69,000	samden	29.05.2006 00
9	contack	22,000	2	44,000	ademden	26.05.2006 00
10	bjon	12,000	3	36,000	ademden	29.05.2006 00

Figure 3.7

When you coming to stock info page you either see all the stocks or saving the new part in figure 3.7

NUMBER PLATE	NAME	TOTAL	DOWN PAYMENT	REMAINING	DATE
Grand Total					

Figure 3.8

In the report page there are one tab controls which includes the daily report, between to date report and according to number plate. On daily report page the date is taking from the system and if there is any record on date it is displayed and ready to sending printer or etc.

On the third of the report page the program want a entering number plate to display and report with exiting records on figure 3.10



Figure 3.11

When we come to tab page of user account firstly page is seems as only one groupbox and whenever you select the any of these other groupboxes begin to seem. Firstly we select the new user radiobutton and as you see ,for new user part is displayed(figure 3.11)



figure3.12

when we select the delete user radiobutton and as you see ,exiting user part is displayed(figure 3.12)



Figure 3.13

The last part of user account tab page is update user. when it is selected update user part is opened and get exiting record to replace it new ones.

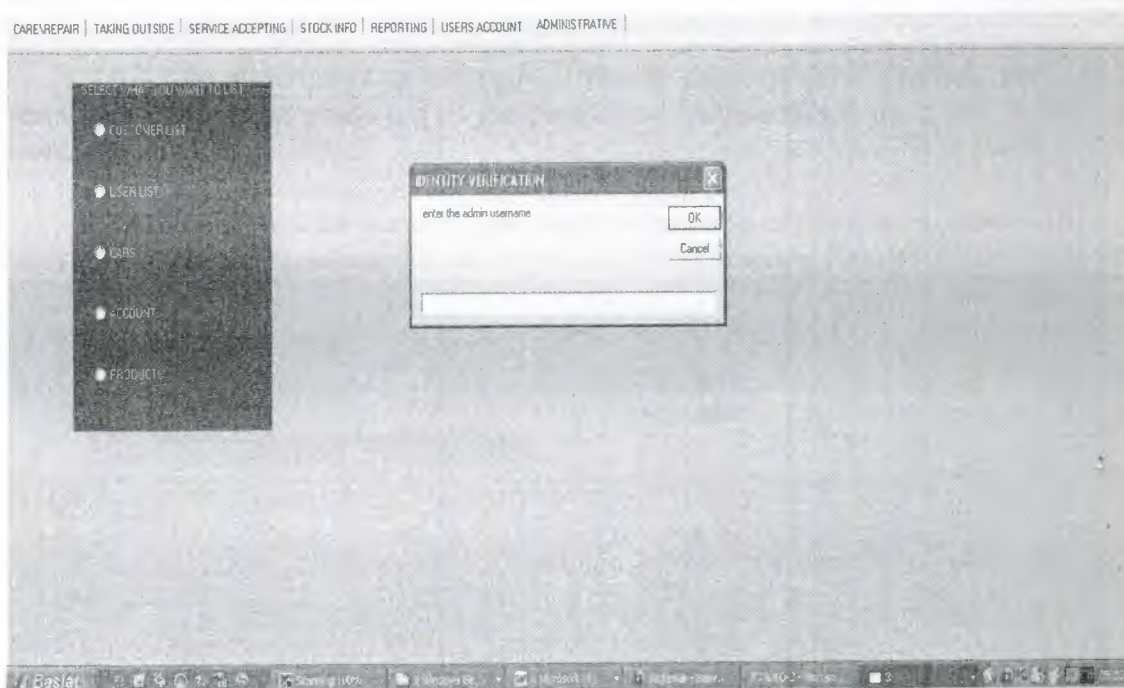


Figure 3.14

When the coming to administrative tab page firstly when it is loaded only the groupbox only displayed and when you try to select any of these program throw a message to display the selected radiobutton.(figure 3.14)

You write administrator username and enter then other message is thrown by program to entering the admin password.(figure 3.15)

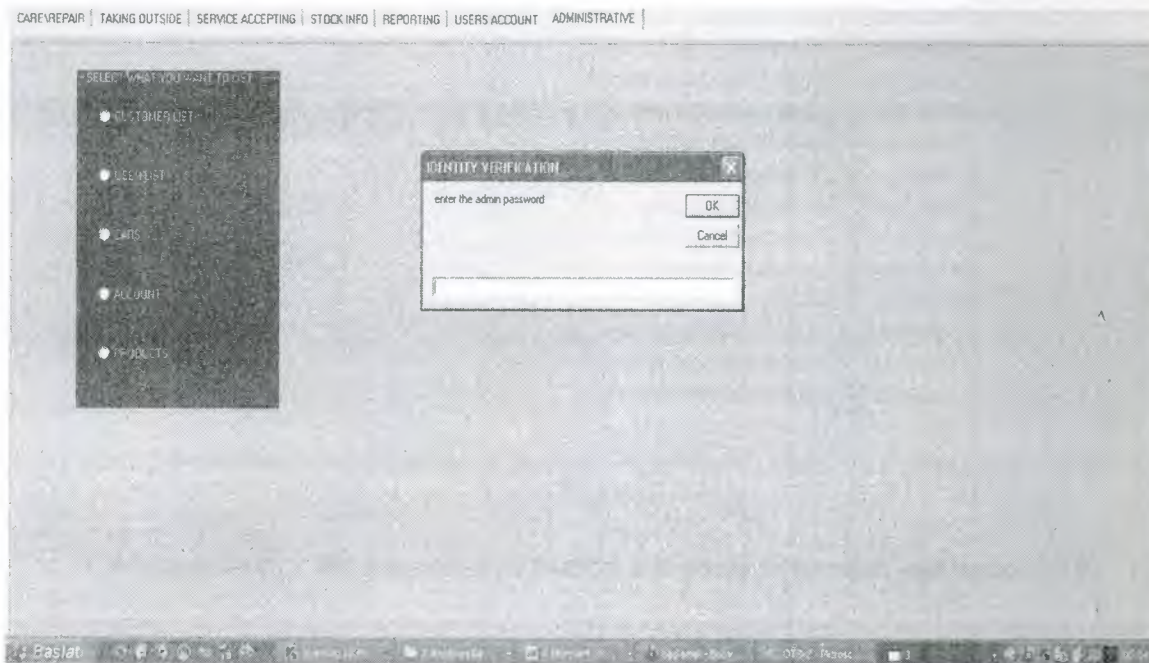


Figure 3.15

When you enter the password correctly program display the detail of records.(figure 3.15)

After the description of main part I want to mention about shortcut part . as you see the program have at the top of the page some buttons which are going to what you want immediatelly.

When you click the periodic care button it is going to first page (figure 3.16)



Figure 3.16

When you click the instant care button it is going to instant care form (figure 3.17)

Figure 3.17

When you click the periodic care button it is going to product list (figure 3.18)

SRD NO	PRG NAME	UNIT PRICE	Q. IN STOCK	TOTAL PRICE	PRGM NAME	DATE
1	kapak	22,0000	2	44,0000	erkekalar	26.05.2006
2	çizile	31,0000	3	93,0000	karalar	26.05.2006
3	buğ	23,0000	2	46,0000	karalar	29.05.2006
4	balata	32,0000	2	64,0000	adamlar	29.05.2006
5	mayı dinamo	35,0000	5	175,0000	adamlar	28.05.2006
6	akı	44,0000	5	220,0000	adamlar	29.05.2006

Figure 3.18

When you click the periodic care button it is going to car list (figure 3.19)



Figure 3.19

When you click the periodic care button it is going to customer list figure 3.20

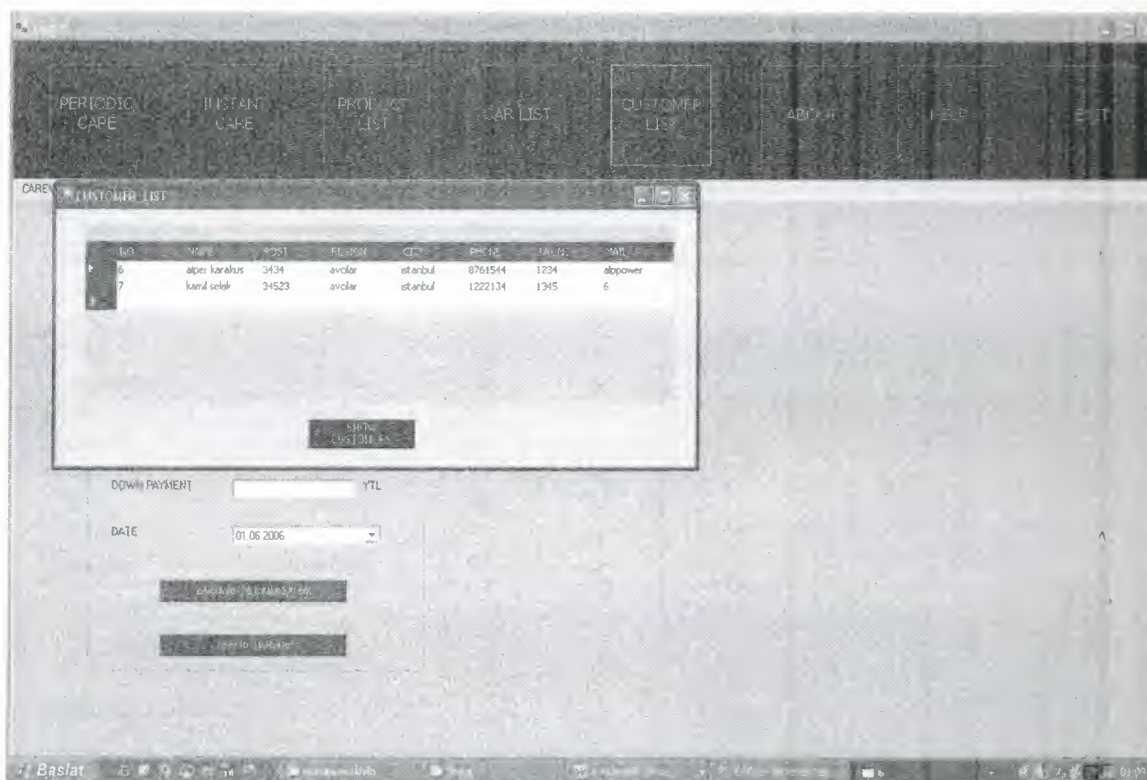


Figure 3.20

After the visual explanation of the program I want to mention about database,tables and its contents. When you click the sql server 2000 shortcut,server is opened and there is tree on the left of the page. Click local server and the screen is seems like that;

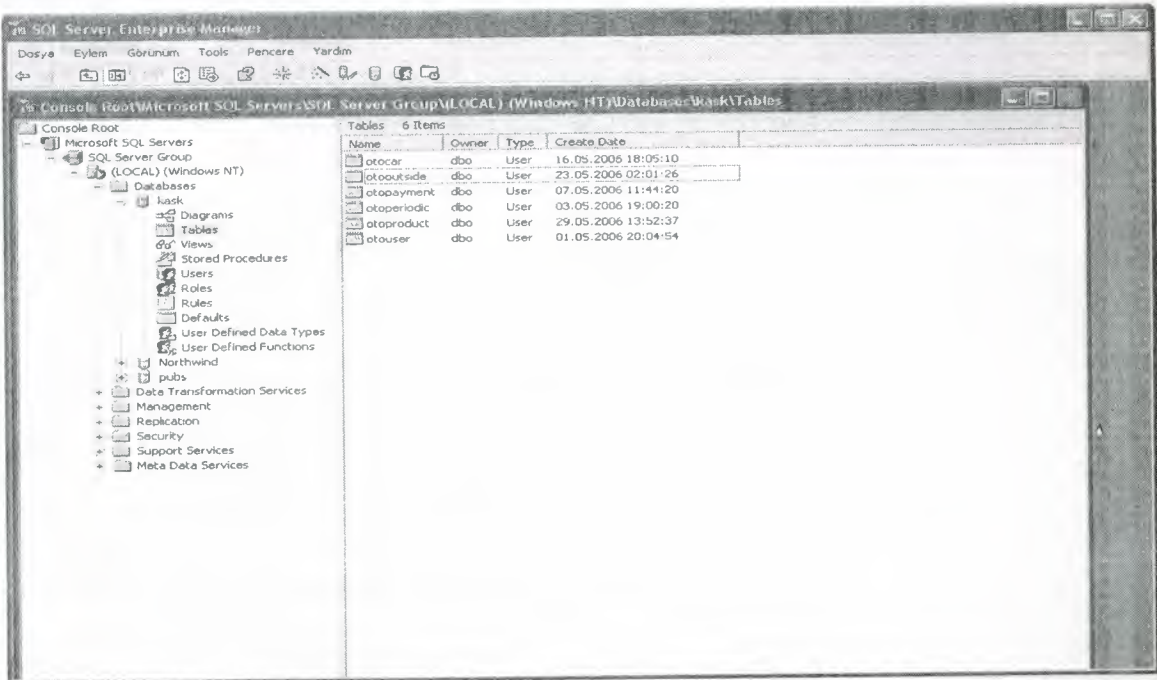


Figure 3.21

The right of the screen seem the tables. Now lets incur the tables.

Otocar table

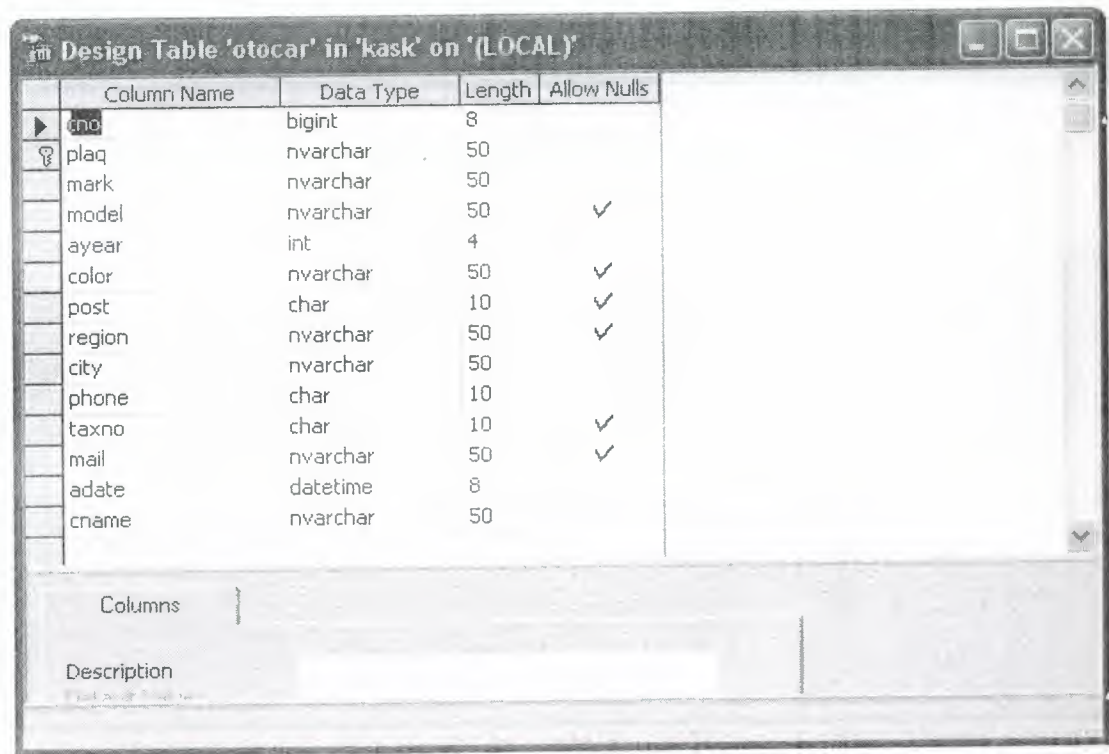


Figure 3.22

Otooutside table

Column Name	Data Type	Length	Allow Nulls
pronaime	nvarchar	50	
unitprice	money	8	
quantity	numeric	9	
totalprice	money	8	✓
fromwhere	nvarchar	50	
forwhich	char	10	
selling	money	8	
dating	datetime	8	✓

Columns

Description

Default Value

Precision

Scale

Identity

Identity Increment

Identity Seed

Formula

Collation

<database default>

Figure 3.23

Otopayment table

Column Name	Data Type	Length	Allow Nulls
plac	char	10	
cname	nvarchar	50	
total	money	8	
donepay	money	8	
dating	datetime	8	✓

Columns

Description

Default Value

Precision

Scale

Identity

Identity Increment

Identity Seed

Formula

Collation

<database default>

Figure 3.24

Otoperiodic table

Design Table 'otoperiodic' in 'kask' on '(LOCAL)'

Column Name	Data Type	Length	Allow Nulls
bakim	nvarchar	50	
fiyati	money	8	

Columns

Description
Default Value
Precision
Scale
Identity
Identity Seed
Identity Increment
Is Primary
Formula
Collation <database default>

Figure 3.25

Otoproduct table

Design Table 'otoproduct' in 'kask' on '(LOCAL)'

Column Name	Data Type	Length	Allow Nulls
prono	bigint	8	
pronomie	nvarchar	50	
unitprice	money	8	
quantity	numeric	9	
total	money	8	
fromwhere	nvarchar	50	✓
tdate	smalldatetime	4	✓

Columns

Description
Default Value
Precision
Scale
Identity
Identity Seed
Identity Increment
Is Primary
Formula
Collation

figure 3.26

otouser table

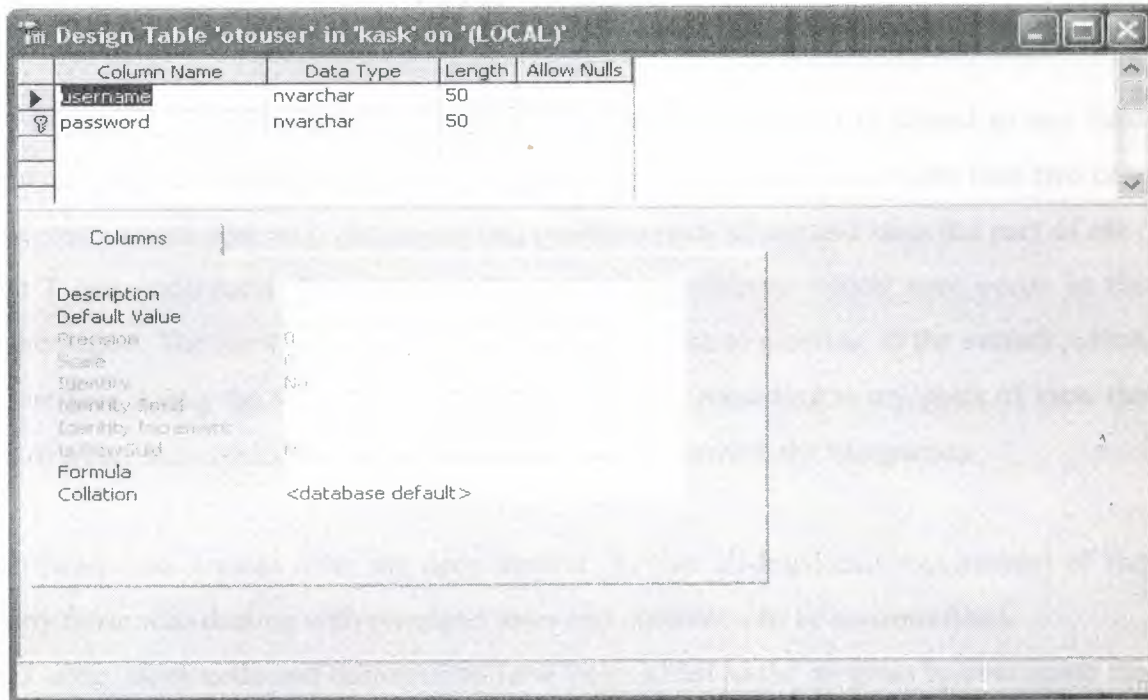


Figure 3.27

CONCLUSION

Practically implementation of software for business though it is related to any field needs a devoted and complete life cycle. In this project I personally visit more than two car repair center, which deal with care to car and purchase parts of car and sales the parts of car, that I can understand their requirements and the problems, which may occur in the implementation. The most important thing that I would like to mention, is the attitude, which is to be faced during the life cycle of the Company. And according to my point of view the reason of most unsuccessful project is misunderstanding between the two parties.

The software was created after the deep analysis, so that all important requirements of the company those who dealing with computer sales and purchase can be accomplished. Product code, stock code and customer no have been added in the program to overcome the mistakes, which may occur. Plus a lock table and form has been generated which contain the tire no with name, so by mistake it cannot be merged with each other. Reports are also generated with the help of the Queries for the update purpose. Which contain all information with dates. The chapters of the software are also organized in such a manner so that all the information related to database and programming language can be understood easily, chapter one contains advance information about the software.

PENDIX A

SOURCE CODES

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Public t As Integer = 0
    Public xx As New C4
    Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton1.CheckedChanged

        If RadioButton1.Checked = True Then
            GroupBox1.Visible = True
            GroupBox2.Visible = False
            GroupBox3.Visible = False
        End If
    End Sub

    Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton2.CheckedChanged
        If RadioButton2.Checked = True Then
            GroupBox2.Visible = True
            GroupBox1.Visible = False
            GroupBox3.Visible = False
        End If
    End Sub

    Private Sub RadioButton3_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton3.CheckedChanged
        If RadioButton3.Checked = True Then
            GroupBox3.Visible = True
            GroupBox1.Visible = False
            GroupBox2.Visible = False
        End If
    End Sub

    Private Sub Label9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)

    End Sub

    Private Sub RadioButton4_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton4.CheckedChanged
        If RadioButton4.Checked = True Then
            GroupBox10.Visible = True
            GroupBox14.Visible = True
        End If

        If RadioButton4.Checked = True Then
    End Sub
```

```
        GroupBox10.Visible = True
    Else : GroupBox10.Visible = False
        GroupBox14.Visible = False
```

```
End If
End Sub
```

```
Private Sub TabPage3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TabPage3.Click
    GroupBox10.Visible = True
    GroupBox14.Visible = False
End Sub
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    SqlDataAdapter4.Fill(DataSet11.otocar)
    CrystalReportViewer1.ReportSource = xx
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)
```

```
    SqlDataAdapter8.Fill(DataSet11.otocar)
    DataGrid5.Visible = False
    DataGrid4.Visible = False
    DataGrid6.Visible = False
    DataGrid7.Visible = False
    DataGrid8.Visible = False
    GroupBox17.Visible = True
    TextBox9.Text = ""
    TextBox8.Text = ""
    TextBox10.Text = ""
    TextBox11.Text = ""
    TextBox12.Text = ""
    TextBox13.Text = ""
    ComboBox1.Text = ""
    TextBox15.Text = ""
    TextBox16.Text = ""
    TextBox17.Text = ""
    TextBox18.Text = ""
    TextBox19.Text = ""
    TextBox20.Text = ""
    DateTimePicker1.Text = ""
    'SqlDataAdapter4.Fill(DataSet11.otocar)
    GroupBox11.Visible = False
    DataGrid1.Visible = False
    GroupBox10.Visible = False
    HelpProvider1.SetHelpString(TextBox26, "PLEASE ENTER PLAQUE")
End Sub
```

```
Private Sub RadioButton5_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioButton5.CheckedChanged
    Dim f2 As New Form2
    f2.Show()
```

```
End Sub
```

```
Private Sub save_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles save.Click
```



```
If TextBox3.Text = "" Or TextBox4.Text = "" Then
    MsgBox("username ve pass boş bırakma")
    Exit Sub
```

```
End If
Try
```

```
col.CommandText = " delete from otouser where username='" &
TextBox3.Text & "' and password='" & TextBox4.Text & "'"
```

```
col.Connection = c2
```

```
c2.Open()
```

```
s1 = col.ExecuteNonQuery
```

```
Dim a As String
```

```
a = TextBox3.Text & "---" & TextBox4.Text
```

```
If s1 > 0 Then
```

```
    MsgBox(a & "silinmiştir")
```

```
End If
```

```
If s1 = 0 Then
```

```
    MsgBox("silinemedi")
```

```
End If
```

```
Catch ex As Exception
```

```
    MsgBox(ex.Message)
```

```
Finally
```

```
    c2.Close()
```

```
TextBox3.Text = ""
```

```
TextBox4.Text = ""
```

```
End Try
```

```
End Sub
```

```
Private Sub update_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

```
    Dim c1 As New SqlClient.SqlConnection
```

```
    Dim col As New SqlClient.SqlCommand
```

```
    c1.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
```

```
    c1.Open()
```

```
    col.Connection = c1
```

```
    Dim s As Integer = 0
```

```
    If TextBox5.Text = "" Or TextBox6.Text = "" Or TextBox37.Text = "" Or
TextBox7.Text = "" Then
```

```

Dim c1 As New SqlClient.SqlConnection
Dim co1 As New SqlClient.SqlCommand
Dim co2 As New SqlClient.SqlCommand
c1.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
Dim dr As SqlClient.SqlDataReader
Dim s1 As Integer = 0
If TextBox1.Text = "" And TextBox2.Text = "" Then
    MsgBox("username ve pass boş bırakma")
    Exit Sub

End If
Try
    col.CommandText = "select * from otouser"
    co2.CommandText = " insert into otouser(username,password)
values('" & TextBox1.Text & "', '" & TextBox2.Text & "')"
    co1.Connection = c1
    co2.Connection = c1
    c1.Open()
    dr = col.ExecuteReader
    Do While dr.Read
        If TextBox1.Text = dr("username") Then
            MsgBox("farklı bir kullanıcı ismi giriniz")
            dr.Close()
            c1.Close()
            Exit Sub
        End If
    Loop
    dr.Close()
    s1 = co2.ExecuteNonQuery
    Dim a As String
    a = TextBox1.Text & "---" & TextBox2.Text
    If s1 > 0 Then MsgBox(a & " kaydedilmiştir")

Catch ex As Exception
    MsgBox(ex.Message)
Finally
    c1.Close()
    dr.Close()
    TextBox1.Text = ""
    TextBox2.Text = ""
End Try

```

End Sub

```

Private Sub delete_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles delete.Click
    Dim c2 As New SqlClient.SqlConnection
    Dim co1 As New SqlClient.SqlCommand

```

```

    c2.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"

```

```

Dim s1 As Integer = 0

```



```

        If TextBox10.Text = dr("plaq") Then
            MsgBox("buarac önceden gelmiş")

        End If
    Loop
    dr.Close()
    sl = co2.ExecuteNonQuery

    If sl > 0 Then MsgBox(" kaydedilmiştir")

Catch ex As Exception
    MsgBox(ex.Message)
Finally
    cl.Close()
    dr.Close()
End Try

End Sub

Private Sub DateTimePicker1_ValueChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs)

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim cl As New SqlClient.SqlConnection
    Dim col As New SqlClient.SqlCommand
    cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
    Dim dr As SqlClient.SqlDataReader
    Dim sl As Integer = 0
    If TextBox10.Text = "" Then
        MsgBox("username ve pass boş bırakma")
        Exit Sub
    End If
    Try
        col.Connection = cl
        cl.Open()
        col.CommandText = "select * from otocar"
        dr = col.ExecuteReader
        Do While dr.Read
            If dr("plaq") = TextBox10.Text Then 'burada cno yerine plaq
kullanmamız gerekiyor fakat plaq p.k olduğu içindatabaseden çağırıyoruz.bunu
sor'
                MsgBox("buarac önceden gelmiş")
                TextBox10.Text = dr("plaq")
                TextBox9.Text = dr("cno")
                TextBox8.Text = dr("cname")
                TextBox11.Text = dr("mark")
                TextBox12.Text = dr("model")
                TextBox13.Text = dr("color")
                ComboBox1.Text = dr("ayear")
                TextBox15.Text = dr("post")
                TextBox16.Text = dr("region")
                TextBox17.Text = dr("city")
            End If
        Loop
    Catch ex As Exception
        MsgBox(ex.Message)
    Finally
        cl.Close()
        dr.Close()
    End Try
End Sub

```

```
MsgBox("verileri giriniz")
Exit Sub
```

```
End If
col.CommandText = " update otouser set username='" & TextBox7.Text &
"' , password='" & TextBox37.Text & "' where username='" & TextBox5.Text & "'
and password='" & TextBox6.Text & "' "
s = col.ExecuteNonQuery
If s > 0 Then
    MsgBox("degistirildi")
End If
cl.Close()
```

```
End Sub
```

```
Private Sub Button2_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
```

```
t5 = DateTimePicker1.Value
m5 = t5.Month & "." & t5.Day & "." & t5.Year
```

```
Dim cl As New SqlClient.SqlConnection
Dim col As New SqlClient.SqlCommand
Dim co2 As New SqlClient.SqlCommand
cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
Dim dr As SqlClient.SqlDataReader
Dim sl As Integer = 0
If TextBox10.Text = "" And TextBox9.Text = "" Then
    'GroupBox6.Visible = False
    'GroupBox7.Visible = False
    'GroupBox8.Visible = False
    MsgBox("username ve pass boş bırakma")
    Exit Sub
```

```
Else
    GroupBox6.Visible = True
    GroupBox7.Visible = True
    GroupBox8.Visible = True
```

```
End If
Dim a As DateTime
a = DateTimePicker1.Text
```

```
Try
```

```
col.CommandText = "select * from otocar"
co2.CommandText = " insert into
otocar(plaq,cname,mark,model,ayear,color,post,region,city,phone,taxno,mail,ada
te) values('" & TextBox10.Text & "','" & TextBox8.Text & "','" &
TextBox11.Text & "','" & TextBox12.Text & "','" & TextBox13.Text & "','" &
ComboBox1.Text & "','" & TextBox15.Text & "','" & TextBox16.Text & "','" &
TextBox17.Text & "','" & TextBox18.Text & "','" & TextBox19.Text & "','" &
TextBox20.Text & "','" & m5 & "')"
col.Connection = cl
co2.Connection = cl
cl.Open()
dr = col.ExecuteReader
```

```
Do While dr.Read
```



```

        TextBox18.Text = dr("phone")
        TextBox19.Text = dr("taxno")
        TextBox20.Text = dr("mail")
        DateTimePicker1.Text = dr("adate")
    Else
        MsgBox("bu aracın servise ilk gelişi")
    End If
End Sub

```

```

Catch ex As Exception
    MsgBox(ex.Message)
Finally
    cl.Close()
    dr.Close()
End Try

```

```
End Sub
```

```

Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    TextBox8.Text = ""
    TextBox9.Text = ""
    TextBox10.Text = ""
    TextBox11.Text = ""
    TextBox12.Text = ""
    TextBox13.Text = ""
    ComboBox1.Text = ""
    TextBox15.Text = ""
    TextBox16.Text = ""
    TextBox17.Text = ""
    TextBox18.Text = ""
    TextBox19.Text = ""
    TextBox20.Text = ""
    DateTimePicker1.Text = ""
End Sub

```

```

Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    t = 0
    Dim cl As New SqlClient.SqlConnection
    Dim cm As New SqlClient.SqlCommand
    Dim cml As New SqlClient.SqlCommand

```

```

    cl.ConnectionString = "data source=alper;initial
catalog=kask;integrated security=true "
    Dim dr As SqlClient.SqlDataReader
    Try
        cl.Open()
        cm.Connection = cl
        cm.CommandText = "select * from otoperiodic"
        dr = cm.ExecuteReader

```

```

    Do While dr.Read 'burdada fiyatların hepsini topluyor????'

```

```
If dr("bakim") = "engineoil" Then
    If CheckBox1.Checked = True Then
        t = t + dr("fiyati")

    End If
End If

If dr("bakim") = "ventilatorbelt" Then
    If CheckBox9.Checked = True Then
        t = t + dr("fiyati")

    End If
End If

If dr("bakim") = "stopoil" Then
    If CheckBox8.Checked = True Then
        t = t + dr("fiyati")

    End If
End If

If dr("bakim") = "steeringoil" Then
    If CheckBox10.Checked = True Then
        t = t + dr("fiyati")

    End If
End If

If dr("bakim") = "sparks" Then
    If CheckBox4.Checked = True Then
        t = t + dr("fiyati")

    End If
End If

If dr("bakim") = "oilfilter" Then
    If CheckBox2.Checked = True Then
        t = t + dr("fiyati")

    End If
End If

If dr("bakim") = "gearboxoil" Then
    If CheckBox11.Checked = True Then
        t = t + dr("fiyati")

    End If
End If
```


End Sub

```
Private Sub TabControl1_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
TabControl1.SelectedIndexChanged
    If TabPage5.Focus = True Then
        sqldataadapter3.Fill(dataset11.otooutside)

    End If
    If TabPage8.Focus = True Then
        Dim z As New CrystalDecisions.Shared.ParameterValues
        Dim z1 As New CrystalDecisions.Shared.ParameterDiscreteValue

        Dim asd = Date.Now.ToShortDateString
        z1.Value = asd
        z.Add(z1)
        xx.DataDefinition.ParameterFields("@gdate").ApplyCurrentValues(z)
        CrystalReportViewer1.ReportSource = xx
    End If
End Sub
```

```
Private Sub RadioButton6_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioButton6.CheckedChanged
    SqlDataAdapter5.Fill(DataSet11.otoproduct)
    If RadioButton6.Checked = True Then
        DataGrid1.Visible = True
        GroupBox11.Visible = False
        Button9.Visible = False
    End If
End Sub
```

```
Private Sub RadioButton10_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioButton10.CheckedChanged
    If RadioButton10.Checked = True Then
        GroupBox11.Visible = True
        Button9.Visible = True
    End If
End Sub
```

```
Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
    tar = DateTimePicker3.Value
    mmm = tar.Month & "." & tar.Day & "." & tar.Year

    Dim c1 As New SqlClient.SqlConnection

    Dim co2 As New SqlClient.SqlCommand
    c1.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"

    Dim s1 As Integer = 0

    Try
        c1.Open()
```

```
If dr("bakim") = "frontstop" Then
    If CheckBox6.Checked = True Then
        t = t + dr("fiyati")
```

```
    End If
End If
```

```
If dr("bakim") = "batterywater" Then
    If CheckBox12.Checked = True Then
        t = t + dr("fiyati")
```

```
    End If
End If
```

```
If dr("bakim") = "backstop" Then
    If CheckBox7.Checked = True Then
        t = t + dr("fiyati")
```

```
    End If
End If
If dr("bakim") = "antifiriz" Then
    If CheckBox5.Checked = True Then
        t = t + dr("fiyati")
    End If
```

```
End If
If dr("bakim") = "airfilter" Then
    If CheckBox3.Checked = True Then
        t = t + dr("fiyati")
```

```
    End If
End If
```

```
'If CheckBox4.Checked = True Then
't = dr("fiyati")
```

```
'End If
TextBox30.Text = t
```

Loop

Catch ex As Exception

```
End Try
End Sub
```

```
Private Sub TabPage4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TabPage4.Click
    'Dim t1 As Date = TextBox41.Text
    'TextBox41.Text = String.Format("{dd/MM/yyyy},t1")
```



```
s1 = co2.ExecuteNonQuery  
Dim a As String
```

```
If s1 > 0 Then MsgBox(" kaydedilmiştir")  
TextBox31.Text = ""  
TextBox32.Text = ""  
TextBox33.Text = ""  
TextBox34.Text = ""  
TextBox35.Text = ""  
TextBox36.Text = ""
```

```
Catch ex As Exception  
    MsgBox(ex.Message)  
Finally
```

```
    SqlDataAdapter5.Fill(DataSet11.otoproduct)  
    cl.Close()
```

```
End Try
```

```
End Sub
```

```
Private Sub Button12_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs)
```

```
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)  
    If cm.Position = cm.Count - 1 Then  
        MsgBox("son kayıttasınız")
```

```
    Else  
        cm.Position += 1  
    End If
```

```
End Sub
```

```
Private Sub Button11_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs)
```

```
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)  
    If cm.Position = 0 Then  
        MsgBox("ilk kayıttasınız")
```

```
    Else  
        cm.Position -= 1  
    End If
```

```
End Sub
```

```
Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs)
```

```
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)  
    If cm.Position = 0 Then  
        MsgBox("zaten ilk kayıttasınız")
```

```
    Else  
        cm.Position = 0
```

```

        co2.Connection = cl
        co2.CommandText = " insert into
otooutside(proname,unitprice,quantity,totalprice,fromwhere,forwhich,selling,da
ting) values('" & TextBox21.Text & "', " & TextBox22.Text & ", " &
TextBox23.Text & ", " & TextBox24.Text & ", '" & TextBox25.Text & "', '" &
TextBox26.Text & "', " & TextBox27.Text & ", '" & mmm & "')"

```

```

        s1 = co2.ExecuteNonQuery
        Dim a As String
        a = TextBox1.Text & "---" & TextBox2.Text
        If s1 > 0 Then MsgBox(a & " kaydedilmiştir")

```

```

Catch ex As Exception
    MsgBox(ex.Message)
Finally
    cl.Close()
    DataSet11.Clear()
    SqlDataAdapter3.Fill(DataSet11.otooutside)

```

```

        TextBox21.Text = ""
        TextBox22.Text = ""
        TextBox23.Text = ""
        TextBox24.Text = ""
        TextBox25.Text = ""
        TextBox26.Text = ""
        TextBox27.Text = ""

```

```

End Try

```

```

End Sub

```

```

Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button8.Click
    DataView1.RowFilter = "forwhich='" & TextBox39.Text & "'"
End Sub

```

```

Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button9.Click
    t4 = DateTimePicker4.Value
    m4 = t4.Month & "." & t4.Day & "." & t4.Year
    Dim cl As New SqlClient.SqlConnection

```

```

    Dim co2 As New SqlClient.SqlCommand
    cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"

```

```

    Dim s1 As Integer = 0

```

```

Try

```

```

    cl.Open()
    co2.Connection = cl
    co2.CommandText = " insert into

```

```

otoproduct(prono,proname,unitprice,quantity,total,fromwhere,tdate) values(" &
TextBox31.Text & ", ' " & TextBox32.Text & " ', " & TextBox33.Text & ", " &
TextBox34.Text & ", " & TextBox35.Text & ", ' " & TextBox36.Text & " ', ' " &
m4 & " ')"

```



```

        If TextBox29.Text = dr("plaq") Then
            MsgBox("aracın zaten bir hesabı var!!!!")
            TextBox28.Text = dr("cname")
            TextBox30.Text = dr("total")
            TextBox42.Text = dr("donepay")
            DateTimePicker2.Text = dr("dating")
            dr.Close()
            cl.Close()
            Exit Sub
        End If
    Loop
    dr.Close()
    s1 = co2.ExecuteNonQuery

    If s1 > 0 Then MsgBox(" kaydedilmiştir")

    Catch ex As Exception
        MsgBox(ex.Message)
    Finally
        cl.Close()
        dr.Close()
    End Try
End If

End Sub

Private Sub DataGrid2_Navigate(ByVal sender As System.Object, ByVal ne As
System.Windows.Forms.NavigateEventArgs) Handles DataGrid2.Navigate

End Sub

Private Sub Button14_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button14.Click
    cm.EndCurrentEdit()
    MsgBox("update process is succesfull")
End Sub

Private Sub Button15_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button15.Click
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)

    If cm.Count <> 0 Then
        cm.RemoveAt(cm.Position)
        MsgBox("silindi")
    Else
        MsgBox(" there is no item to delete ")
    End If
    cm.Refresh()

End Sub

Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CheckBox1.CheckedChanged

```

```

End If
End Sub

Private Sub Button13_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)
    If cm.Position = cm.Count - 1 Then
        MsgBox("zaten son kayıttasınız")
    Else
        cm.Position -= 1
    End If
End Sub

Private Sub Button16_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button16.Click
    Dim a As Integer
    Dim b As Integer
    a = TextBox30.Text
    b = TextBox42.Text
    If b > a Then
        MsgBox("done payment can not greater then total price",
MsgBoxStyle.Information, "SYSTEM-ERROR-CONTROL")
        TextBox28.Text = ""
        TextBox29.Text = ""
        TextBox30.Text = ""
        TextBox42.Text = ""

    Else

        tar = DateTimePicker2.Value
        mmm = tar.Month & "." & tar.Day & "." & tar.Year

        Dim cl As New SqlClient.SqlConnection
        Dim col As New SqlClient.SqlCommand
        Dim co2 As New SqlClient.SqlCommand
        cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
        Dim dr As SqlClient.SqlDataReader
        Dim sl As Integer = 0
        If TextBox29.Text = "" Then
            MsgBox("plaq'ı boş bırakma")
            Exit Sub
        End If
        Try
            col.CommandText = "select * from otopayment"
            co2.CommandText = "insert into
otopayment(plaq,cname,total,donepay,dating) values('" & TextBox29.Text & "','
'" & TextBox28.Text & "',' & TextBox30.Text & "','" & TextBox42.Text & "','" &
mmm & "')"
            col.Connection = cl
            co2.Connection = cl
            cl.Open()
            dr = col.ExecuteReader
            Do While dr.Read

```


End Sub

```
Private Sub TextBox9_Enter(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox9.Enter
    MessageBox.Show("Not enter any record this fields", "Not Enter",
    MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
```

```
Private Sub DataView2_ListChanged(ByVal sender As System.Object, ByVal e
As System.ComponentModel.ListChangedEventArgs)
```

End Sub

```
Private Sub TabPage5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TabPage5.Click
```

End Sub

```
Private Sub TextBox39_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox39.KeyPress
    If e.KeyChar = ChrW(13) Then
        DataView1.RowFilter = "forwhich='" & TextBox39.Text & "'"
    End If
End Sub
```

```
Private Sub Button7_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
    Dim c1 As New SqlClient.SqlConnection
    Dim col As New SqlClient.SqlCommand

    c1.ConnectionString = "data source=ALPER;initial
    CATALOG=kask;integrated security=true"
    c1.Open()
    col.Connection = c1

    Dim s As Integer = 0
    If TextBox5.Text = "" Or TextBox6.Text = "" Or TextBox37.Text = "" Or
    TextBox7.Text = "" Then
        MsgBox("verileri giriniz")
        Exit Sub
    End If

    col.CommandText = " update otouser set username='" & TextBox7.Text &
    "'", password='" & TextBox37.Text & "'" where username='" & TextBox5.Text & "'"
    and password='" & TextBox6.Text & "'" "
    s = col.ExecuteNonQuery
    If s > 0 Then
        MsgBox("degistirildi")
    End If
    c1.Close()
    TextBox5.Text = ""
    TextBox6.Text = ""
    TextBox7.Text = ""
    TextBox37.Text = ""
```

End Sub

```
Private Sub RadioButton13_CheckedChanged(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles RadioButton13.CheckedChanged
```

 DataGrid5.Visible = False

 DataGrid4.Visible = False

 DataGrid6.Visible = False

 DataGrid7.Visible = True

 DataGrid8.Visible = False

 SqlDataAdapter11.Fill(DataSet41.otopayment)

End Sub

```
Private Sub RadioButton15_CheckedChanged(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles RadioButton15.CheckedChanged
```

 DataGrid5.Visible = False

 DataGrid4.Visible = False

 DataGrid6.Visible = False

 DataGrid7.Visible = False

 DataGrid8.Visible = True

 SqlDataAdapter13.Fill(DataSet61.otoproduct)

End Sub

```
Private Sub RadioButton8_CheckedChanged(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles RadioButton8.CheckedChanged
```

 DataGrid5.Visible = False

 DataGrid4.Visible = True

 DataGrid6.Visible = False

 DataGrid7.Visible = False

 DataGrid8.Visible = False

 SqlDataAdapter6.Fill(DataSet11.otouser)

End Sub

```
Private Sub Button23_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button23.Click
```

 Application.Exit()

End Sub

```
Private Sub Button5_Click_3(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button5.Click
```

 TabControl1.SelectedIndex = 0

End Sub

```
Private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object, ByVal  
e As System.EventArgs) Handles CheckBox3.CheckedChanged
```

End Sub

```
Private Sub DataView2_ListChanged_1(ByVal sender As System.Object, ByVal e  
As System.ComponentModel.ListChangedEventArgs) Handles DataView2.ListChanged
```

End Sub


```

Private Sub Button10_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button10.Click
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)
    If cm.Position = 0 Then
        MsgBox("you are still on first record")
    Else
        cm.Position = 0
    End If
End Sub

```

```

Private Sub Button11_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button11.Click
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)
    If cm.Position = 0 Then
        MsgBox("you are on first record")
    Else
        cm.Position -= 1
    End If
End Sub

```

```

Private Sub Button12_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button12.Click
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)
    If cm.Position = cm.Count - 1 Then
        MsgBox("you are on last record")
    Else
        cm.Position += 1
    End If
End Sub

```

```

Private Sub Button13_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button13.Click
    cm = CType(Me.BindingContext(DataView2), CurrencyManager)
    If cm.Position = cm.Count - 1 Then
        MsgBox("you are still on last record")
    Else
        cm.Position = cm.Count - 1
    End If
End Sub

```

```

Private Sub RadioButton7_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioButton7.CheckedChanged
    SqlDataAdapter9.Fill(DataSet21.otocar)

    If RadioButton7.Checked = True Then
        DataGrid5.Visible = True
        DataGrid4.Visible = False
        DataGrid6.Visible = False
        DataGrid7.Visible = False
        DataGrid8.Visible = False
    End If

```

```

End Sub

```

```
Private Sub TabPage9_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles TabPage9.Click
```

```
End Sub
```

```
Private Sub RadioButton9_CheckedChanged(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles RadioButton9.CheckedChanged
```

```
    DataGridView5.Visible = False  
    DataGridView4.Visible = False  
    DataGridView6.Visible = True  
    DataGridView7.Visible = False  
    DataGridView8.Visible = False
```

```
    SqlDataAdapter10.Fill(DataSet31.otocar)
```

```
End Sub
```

```
Private Sub TabPage9_Enter(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles TabPage9.Enter
```

```
    Dim ad As String
```

```
    Dim soyad As String
```

```
    ad = InputBox("enter the admin username", "IDENTITY VERIFICATION")
```

```
    If ad = "admin" Then
```

```
        soyad = InputBox("enter the admin password", "IDENTITY  
VERIFICATION")
```

```
        If soyad = "master" Then
```

```
            GroupBox16.Visible = True
```

```
        Else
```

```
            MsgBox("entered password is invalid")
```

```
            GroupBox16.Visible = False
```

```
        End If
```

```
    Else
```

```
        MsgBox("entered username is invalid")
```

```
        GroupBox16.Visible = False
```

```
    End If
```

```
End Sub
```

```
Private Sub TabPage9_Leave(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles TabPage9.Leave
```

```
    GroupBox16.Visible = True
```

```
End Sub
```

```
Private Sub Button24_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button24.Click
```

```
    tar = DateTimePicker5.Value
```

```
    mmm = tar.Month & "." & tar.Day & "." & tar.Year
```

```
    tar2 = DateTimePicker6.Value
```

```
    mmm2 = tar2.Month & "." & tar2.Day & "." & tar2.Year
```

```
    Dim A As Date = DateTimePicker5.Value
```

```
    Dim b As Date = DateTimePicker6.Value
```

```
    Dim basla As String = A.ToShortDateString
```

```
    Dim bitir As String = b.ToShortDateString
```

```
    ' DataView6.RowFilter = "dating=>'{0}' and dating<='{1}'", basla,
```

```
    bitir"
```


End Sub

```
Private Sub Button20_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button20.Click
    Dim cl As New CUSTOMER_LIST
    cl.Show()
```

End Sub

```
Private Sub CrystalReportViewer1_Load_1(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CrystalReportViewer1.Load
```

End Sub

```
Private Sub Button25_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button25.Click
```

```
    Dim xx As New c5
    st = DateTimePicker7.Value
    en = DateTimePicker8.Value
```

```
    Dim z As New CrystalDecisions.Shared.ParameterValues
    Dim zz As New CrystalDecisions.Shared.ParameterValues
```

```
    Dim z1 As New CrystalDecisions.Shared.ParameterDiscreteValue
    Dim z2 As New CrystalDecisions.Shared.ParameterDiscreteValue
```

```
    Dim urun1 = st.ToShortDateString
    Dim urun2 = en.ToShortDateString
```

```
    z1.Value = urun1
    z2.Value = urun2
    z.Add(z1)
    zz.Add(z2)
```

```
    xx.DataDefinition.ParameterFields("@gdate1").ApplyCurrentValues(z)
    xx.DataDefinition.ParameterFields("@gdate2").ApplyCurrentValues(zz)
```

```
    CrystalReportViewer2.ReportSource = xx
```

End Sub

```
    Dim x As Integer
```

```
    Dim y As Integer
```

```
Private Sub TextBox34_LostFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox34.LostFocus
```

```
    x = TextBox33.Text
    y = TextBox34.Text
    TextBox35.Text = (x * y).ToString
```

End Sub

```
Private Sub TabPage8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TabPage8.Click
```

```
If TabPage8.Focus = True Then
    TabControl2.Show()
```

```
End If
End Sub
```

```
Private Sub TextBox23_LostFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox23.LostFocus
    Dim a As Integer
    Dim b As Integer
    a = TextBox22.Text
    b = TextBox23.Text
    TextBox24.Text = (a * b).ToString
End Sub
```

```
Private Sub Button17_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button17.Click
    f2.Show()
End Sub
```

```
Private Sub Button19_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button19.Click
    f5.Show()
End Sub
```

```
Private Sub TextBox29_Leave(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox29.Leave
    If TextBox29.Text = "" Then
        MsgBox("can not empty this cell")
    Exit Sub
```

```
End If
Dim cl As New SqlClient.SqlConnection
Dim col As New SqlClient.SqlCommand
cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
Dim dr As SqlClient.SqlDataReader
Dim sl As Integer = 0
Dim a As MsgBoxResult
```

```
Try
```

```
col.CommandText = "select * from otocar"
col.Connection = cl
cl.Open()
dr = col.ExecuteReader
Do While dr.Read
    If dr("plaq") = TextBox29.Text Then
        TextBox28.Text = dr("cname")
    Else
```

```
        If MsgBox("araç kayıtlı değil,kaydetmek istermisiniz",
MsgBoxStyle.YesNo, "SYSTEM INFORMATION") = MsgBoxResult.Yes Then
```



```

TabControl1.SelectedIndex = 1
TextBox8.Text = ""
TextBox9.Text = ""
TextBox10.Text = ""
TextBox11.Text = ""
TextBox12.Text = ""
TextBox13.Text = ""
TextBox15.Text = ""
TextBox16.Text = ""
TextBox17.Text = ""
TextBox18.Text = ""
TextBox19.Text = ""
TextBox20.Text = ""

```

```
Else
```

```
    TabControl1.SelectedIndex = 0
```

```
End If
```

```
End If
```

```
Exit Do
```

```
Loop
```

```
dr.Close()
```

```
cl.Close()
```

```
Catch ex As Exception
```

```
    MsgBox(ex.Message)
```

```
Finally
```

```
    cl.Close()
```

```
    dr.Close()
```

```
    cl.Close()
```

```
End Try
```

```
End Sub
```

```

Private Sub Button18_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button18.Click
    f6.Show()

```

```
End Sub
```

```

Private Sub TabControl2_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
TabControl2.SelectedIndexChanged

```

```
    If TabPage1.Focus = True Then
```

```
        Dim xx As New c4
```

```
    End If
```

```
    If TabPage2.Focus = True Then
```

```
        Dim xx As New c5
```

```
    End If
```

```
End Sub
```

```
Private Sub Button26_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button26.Click
```

```
Dim xx As New c6
```

```
Dim z As New CrystalDecisions.Shared.ParameterValues
```

```
Dim z1 As New CrystalDecisions.Shared.ParameterDiscreteValue
```

```
Dim urun1 = TextBox14.Text
```

```
z1.Value = urun1
```

```
z.Add(z1)
```

```
xx.DataDefinition.ParameterFields("@numberplate").ApplyCurrentValues(z)
```

```
CrystalReportViewer3.ReportSource = xx
```

```
End Sub
```

```
End Class
```



```
Public Class Form2
```

```
    Inherits System.Windows.Forms.Form
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click
```

```
    If TextBox15.Text = "" Then
```

```
        TextBox15.Text = 0
```

```
    End If
```

```
    If TextBox16.Text = "" Then
```

```
        TextBox16.Text = 0
```

```
    End If
```

```
    If TextBox17.Text = "" Then
```

```
        TextBox17.Text = 0
```

```
    End If
```

```
    If TextBox18.Text = "" Then
```

```
        TextBox18.Text = 0
```

```
    End If
```

```
    If TextBox19.Text = "" Then
```

```
        TextBox19.Text = 0
```

```
    End If
```

```
    If TextBox20.Text = "" Then
```

```
        TextBox20.Text = 0
```

```
    End If
```

```
    If TextBox21.Text = "" Then
```

```
        TextBox21.Text = 0
```

```
    End If
```

```
    If TextBox22.Text = "" Then
```

```
        TextBox22.Text = 0
```

```
    End If
```

```
    If TextBox23.Text = "" Then
```

```
        TextBox23.Text = 0
```

```
    End If
```

```
    If TextBox30.Text = "" Then
```

```
        TextBox30.Text = 0
```

```
    End If
```

```
    If TextBox31.Text = "" Then
```

```
        TextBox31.Text = 0
```

```
    End If
```

```
    If TextBox32.Text = "" Then
```

```
        TextBox32.Text = 0
```

```
    End If
```

```
    If TextBox33.Text = "" Then
```

```
        TextBox33.Text = 0
```

```
    End If
```

```
    If TextBox34.Text = "" Then
```

```
        TextBox34.Text = 0
```

```
    End If
```

```
    If TextBox29.Text = "" Then
```

```
        TextBox29.Text = 0
```

```
    End If
```

```
Dim a As Integer = 0
```

```
Dim b As Integer = 0
```

```
Dim c As Integer = 0
```

```
Dim d As Integer = 0
```

```
Dim u As Integer = 0
Dim f As Integer = 0
Dim g As Integer = 0
Dim h As Integer = 0
Dim i As Integer = 0
Dim j As Integer = 0
Dim k As Integer = 0
Dim l As Integer = 0
Dim m As Integer = 0
Dim n As Integer = 0
Dim o As Integer = 0
```

```
a = CType(TextBox34.Text, Integer)
b = CType(TextBox15.Text, Integer)
c = CType(TextBox16.Text, Integer)
d = CType(TextBox17.Text, Integer)
u = CType(TextBox18.Text, Integer)
f = CType(TextBox19.Text, Integer)
g = CType(TextBox20.Text, Integer)
h = CType(TextBox21.Text, Integer)
i = CType(TextBox22.Text, Integer)
j = CType(TextBox23.Text, Integer)
k = CType(TextBox30.Text, Integer)
l = CType(TextBox31.Text, Integer)
m = CType(TextBox32.Text, Integer)
n = CType(TextBox33.Text, Integer)
o = CType(TextBox29.Text, Integer)
```

```
TextBox13.Text = a + b + c + d + u + f + g + h + i + j + k + l + m + n
```

```
+ o
```

```
End Sub
```

```
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
    If TextBox11.Text = "" Or TextBox12.Text = "" Then
        'Panel1.Enabled = False
        TextBox13.Enabled = True
```

```
    Else 'Panel1.Enabled = True
        TextBox13.Enabled = True
```

```
End If
```

```
End Sub
```

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

```
End Sub
```

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
```



```

Dim c As Integer
Dim b As Integer
c = TextBox13.Text
b = TextBox24.Text
If b > c Then
    MsgBox("done payment can not greater then total price",
MsgBoxStyle.Information, "SYSTEM-ERROR-CONTROL")
    TextBox11.Text = ""
    TextBox12.Text = ""
    TextBox13.Text = ""
    TextBox24.Text = ""

Else
    tar3 = DateTimePicker1.Value
    mmm3 = tar.Month & "." & tar.Day & "." & tar.Year
    Dim cl As New SqlClient.SqlConnection
    Dim col As New SqlClient.SqlCommand
    Dim co2 As New SqlClient.SqlCommand
    Dim a As DateTime
    a = DateTimePicker1.Text
    cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
    Dim dr As SqlClient.SqlDataReader
    Dim s1 As Integer = 0
    If TextBox12.Text = "" Then
        MsgBox("plaq'ı boş bırakma")
        Exit Sub

    End If
    Try
        col.CommandText = "select * from otopayment"
        co2.CommandText = "insert into
otopayment(plaq,cname,total,donepay,dating) values('" & TextBox12.Text & "',
'" & TextBox11.Text & "'," & TextBox13.Text & "," & TextBox24.Text & "','" &
mmm3 & "')"
        col.Connection = cl
        co2.Connection = cl
        cl.Open()
        dr = col.ExecuteReader
        Do While dr.Read
            If TextBox12.Text = dr("plaq") Then
                MsgBox("aracın zaten bir hesabı var!!!!")
                TextBox11.Text = dr("cname")
                TextBox13.Text = dr("total")
                TextBox24.Text = dr("donepay")
                DateTimePicker1.Text = dr("dating")

                dr.Close()
                cl.Close()
                Exit Sub
            End If
        Loop
        dr.Close()
        s1 = co2.ExecuteNonQuery

        If s1 > 0 Then MsgBox(" kaydedilmiştir")
    
```

```

Catch ex As Exception
    MsgBox(ex.Message)
Finally
    cl.Close()
    dr.Close()
End Try
End If

```

```
End Sub
```

```

Private Sub TextBox12_Leave(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox12.Leave
    If TextBox12.Text = "" Then
        MsgBox("can not empty this cell")
    Exit Sub

```

```

End If
Dim cl As New SqlClient.SqlConnection
Dim col As New SqlClient.SqlCommand
cl.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
Dim dr As SqlClient.SqlDataReader
Dim sl As Integer = 0
Dim a As MsgBoxResult

```

```
Try
```

```

col.CommandText = "select * from otocar"
col.Connection = cl
cl.Open()
dr = col.ExecuteReader
Do While dr.Read

```

```

    If dr("plaq") = TextBox12.Text Then
        TextBox11.Text = dr("cname")

```

```

    Else
        a = MsgBox("araç kayıtlı değil,kaydetmek istermisiniz",
MsgBoxStyle.YesNo, "SYSTEM INFORMATION")
        If a.Yes Then

```

```

            fl.TabControl1.SelectedIndex = 1
            TextBox8.Text = ""
            TextBox9.Text = ""
            TextBox10.Text = ""
            TextBox11.Text = ""
            TextBox12.Text = ""
            TextBox13.Text = ""
            TextBox15.Text = ""
            TextBox16.Text = ""
            TextBox17.Text = ""
            TextBox18.Text = ""
            TextBox19.Text = ""
            TextBox20.Text = ""

```

```

        Else
            fl.TabControl1.SelectedIndex = 0
        End If
    End If

```



```

    Loop
    dr.Close()
    cl.Close()

Catch ex As Exception
    MsgBox(ex.Message)
Finally
    cl.Close()
    dr.Close()
    cl.Close()
End Try
End Sub
End Class

```

```

Public Class Form3
    Inherits System.Windows.Forms.Form
    Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim c As New SqlClient.SqlConnection
        Dim co As New SqlClient.SqlCommand
        Dim dr As SqlClient.SqlDataReader
        Try

            c.ConnectionString = "data source=ALPER;initial
CATALOG=kask;integrated security=true"
            c.Open()
            co.Connection = c
            co.CommandText = " select * from otouser"
            dr = co.ExecuteReader
            Dim s As Integer = 0
            Do While dr.Read
                If dr("username") = TextBox1.Text And dr("password") =
TextBox2.Text Then
                    s = 1
                    Exit Do
                End If
            Loop
            If s = 0 Then
                MsgBox("girilen degerler yanliş")
            End Sub

        End If
    End Sub

```

```

Dim Fl As New Form1
Fl.Show()

Catch ex As SqlClient.SqlException
    MsgBox(ex.Message)

End Try

c.Close()
dr.Close()

End Sub

Private Sub SqlDataAdapter1_RowUpdated(ByVal sender As System.Object,
ByVal e As System.Data.SqlClient.SqlRowUpdatedEventArgs) Handles
SqlDataAdapter1.RowUpdated

End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Application.Exit()

End Sub

End Class

Public Class PRODUCT_LIST
    Inherits System.Windows.Forms.Form

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        SqlDataAdapter1.Fill(DataSet11.otoproduct)
    End Sub

Public Class CAR_LIST
    Inherits System.Windows.Forms.Form

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        SqlDataAdapter1.Fill(DataSet11.otocar)
    End Sub

Public Class CUSTOMER_LIST
    Inherits System.Windows.Forms.Form

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        SqlDataAdapter1.Fill(DataSet71.otocar)

    End Sub

```