NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

SOFTWARE DESIGN FOR A COMPUTER COMPANY USING C# AND ASP.NET

Graduation Project
COM- 400

Students : Mesut Arık (20020476)

Supervisor : Assist Prof Dr Adil Amirjanov

Nicosia – 2006

ACKNOWLEDGEMENT

First, I would like to thank my supervisor Assist. Prof. Dr Adil Amirjanov for sharing his innovative ideas, and advices with me and showing his belief in my work. His commitments in this project are invaluable.

Second, I would like to express my gratitude to my family for their endless support and encouragement during my undergraduate studies in the university and making of this Graduation Project.

Finally, I would like to thank to all of my friends, starting from my home mates Göktuğ Ataç, Engin Alan and my best friends Süleyman Kerimoğlu, Sinan Özerenler and Ozan Akkoca.

ABSTRACT

Companies having big number of customers need complicated computer systems in order to respond their clients efficiently. Controlling a large-scale company without any computer system is almost impossible. If there is a lack of quality in the services of the organization, it would be inevitable to lose money and prestige.

Developments in the computer science technology allow programmers to write more complicated and powerful applications each day as a respond to the needs of the companies. Making a good software design is the most crucial part of developing a computer system. New programming concepts and design techniques are making it possible to make decent software designs. Design of a software could be extensively changed by the specific requirements of the company. So, a computer system should be flexible and also easy to maintain.

The aim of this project is to make a software design for a computer company which is selling computer components as the product.

The software lets the users control the stock and sales activities of a middle or big-tier computer company in a fast and reliable way. Since the users of this software will not necessarily have expertise in computers, Graphical User Interface (GUI) of the program must be user-friendly and let the users to do their jobs in a convenient way.

TABLE OF CONTENTS

INTRODUCTION

Object Oriented Software design has been developing rapidly since early 90's with the invention of Object Oriented Programming (OOP) Languages like C++, Java, and C# and object oriented software design tools and languages like UML.

The scope of this software is to keep the records of the items in a computer company's stocks, facilitate modification of items in the stocks, sales transactions, keeping employee information, and grant permissions to the employees. These kind of mission-critical task should be handled carefully, that is to say the software must be reliable and error-free.

There are two different permission types for the authorization to the system. First one is 'Administrator', which is for executing administrative tasks for managers, and second one is the 'Sales', which is for handling sales actions by the employees of the sales department.

Recording all the sales transaction information is important for the management and accounting departments in a company. These departments need to have full access to the records in order to understand the economical situation of the company. Also this information can be used for preparing reports regarding stocks or income/expenditure balances.

Chapter One starts with Introducing the Object Oriented Programming, and basic units involved in OOP. Also Object Oriented Programming concepts are described in this chapter.

Chapter Two presents the Microsoft's new software development platform, .NET Framework 2.0 and CLI technology which is in the heart of this platform. C# language is described as the OOP Language used in the project.

Chapter Three describes Object Oriented Software design techniques and Unified Markup Language (UML). Class diagrams of some of the classes used in the project are given.

Chapter Four describes the features of the Microsoft's SQL Server 2005. Structured Query Language (SQL) is also presented in this chapter. Data tables used in the project are given with their relationship diagrams.

Chapter Five presents the design features of the project, and shows the usage of the program for the users with different privileges to the system.

CHAPTER 1 OBJECT ORIENTED PROGRAMMING

1.1 Introduction to Object Oriented Programming

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. The idea behind object-oriented programming is that a computer program may be seen as comprising a collection of individual units, or *objects*, that act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions, or simply as a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine or actor with a distinct role or responsibility.

Object-oriented programming is claimed to promote greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering. Furthermore, proponents of OOP claim that OOP is easier to learn for those new to computer programming than previous approaches, and that the OOP approach is often simpler to develop and to maintain, lending itself to more direct analysis, coding, and understanding of complex situations and procedures than other programming methods.

In the past decade Java has emerged in wide use partially because of its similarity to C and to C++, but perhaps more importantly because of its implementation using a virtual machine that is intended to run code unchanged on many different platforms. This last feature has made it very attractive to larger development shops with heterogeneous environments. Microsoft's .NET initiative has a similar objective and includes/supports several new languages, or variants of older ones.

More recently, a number of languages have emerged that are primarily object-oriented yet compatible with procedural methodology, such as Python and Ruby. Besides Java, probably the most commercially important recent object-oriented languages are Visual Basic .NET and C# designed for Microsoft's .NET platform.

Just as procedural programming led to refinements of techniques such as structured programming, modern object-oriented software design methods include

refinements such as the use of design patterns, design by contract, and modeling languages (such as UML).

1.2 Object Oriented Programming Concepts

Object-oriented programming (OOP) emphasizes the following concepts:

- Class — the unit of definition of data and behavior (functionality) for some kind-of-thing. For example, the 'class of Dogs' might be a set which includes the various breeds of dogs. A class is the basis of modularity and structure in an object-oriented computer program. *A class should typically be recognizable to a non-programmer familiar with the problem domain,* and the code for a class should be (relatively) self-contained and independent (as should the code for any good pre-OOP function). With such modularity, the structure of a program will correspond to the aspects of the problem that the program is intended to solve. This simplifies the mapping to and from the problem and program.

- Object — an instance of a class, an object (for example, "Lassie" the Dog) is the run-time manifestation (*instantiation*) of a *particular exemplar* of a class. (For the class of dogs which contains breed types, an acceptable exemplar would only be the *subclass* 'collie'; "Lassie" would then be an object in that subclass.) Each object has its own data, though the code within a class (or a subclass or an object) may be shared for economy. .

- Method (also known as *message*) — how code can use an object of some class. A *method* is a form of subroutine operating on a single object. Methods may be divided into queries returning the current state and commands changing it: a `Dog` could have a query `Age` to say how old it is, and command `chase (Rabbit target)` to start it chasing a rabbit. A method may also do both, but some authorities recommend they be kept separate. Sometimes access to the data of an object is restricted to the methods of its class.
    - A member of a class or object is a method or a data item describing the state of an object. In some languages the general term is feature.

- Inheritance — a mechanism for creating subclasses, inheritance provides a way to define a (sub)class as a specialization or subtype or extension of a more general class: `Dog` is a subclass of `Canidae`, and `Collie` is a subclass of the (sub)class `Dog`. A subclass *inherits* all the members of its superclass(es), but it can extend their behaviour and add new members. Inheritance is the "is-a" relationship: a `Dog` *is a* `Canidae`. This is in contrast to composition, the "has-a" relationship: a `Dog` *has a* mother (another `Dog`) and *has a* father, etc.
    - Multiple inheritance – a `Dog` is both a `Pet` and a `Canidae` – is not always supported, as it can be hard both to implement and to use well.

- Encapsulation — ensuring that code outside a class sees only functional details of that class, but not implementation details. The latter are liable to change, and could allow a user to put an object in an inappropriate state. Encapsulation is achieved by specifying which classes may use the members of an object. The result is that each object exposes to any class a certain *interface* — those members accessible to that class. For example, an interface can ensure that puppies can only be added to an object of the class `Dog` by code in that class. Members are often specified as public, protected and private, determining whether they are available to all classes, sub-classes or only the defining class. Some languages go further: Java uses the protected keyword to restrict access also to classes in the same package, C# and VB.NET reserve some members to classes in the same assembly using keywords internal (C#) or Friend (VB.NET), and Eiffel allows one to specify which classes may access any member.

- Abstraction — the ability of a program to ignore the details of an object's (sub)class and work at a more generic level when appropriate; For example, "Lassie" the Dog may be treated as a Dog much of the time, but when appropriate she is abstracted to the level of Canidae (superclass of Dog) or Carnivora (superclass of Canidae), and so on.

- Polymorphism — polymorphism is behavior that varies depending on the class in which the behavior is invoked, that is, two or more classes can react *differently* to the *same message*. For example, if Dog is commanded to speak this may elicit a Bark; if Pig is commanded to speak this may elicit a Grunt.

1.2.1 Classes

In object-oriented programming, a class is a template definition of the methods and variables in a particular kind of object. Thus, an object is a specific instance of a class; it contains real values instead of variables.

Once you've identified an object, you generalize it as a class of objects and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called (no surprise here) an "object" or, in some environments, an "instance of a class." The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

The class is one of the defining ideas of object-oriented programming. Among the important ideas about classes are:

- A class can have subclasses that can inherit all or some of the characteristics of the class. In relation to each subclass, the class becomes the superclass.
- Subclasses can also define their own methods and variables that are not part of their superclass.
- The structure of a class and its subclasses is called the class hierarchy.

```
class Person
{
    private int age;
    private string hairColor;
    public int Age
    {
        get
        {
        return age;
        }
        set
        {
            if(value <= 65 && value >= 18)
            {
                age = value;
            }
```

```
            else
                age = 18;
        }
    }
    public string HairColor
    {
        get
        {
            return hairColor;
        }
        set
        {
            hairColor = value;
        }
    }
}
```

1.2.2 Inheritance

Inheritance is a way to form new classes (instances of which are called objects) using classes that have already been defined. The former, known as derived classes, take over (or inherit) attributes and behaviour of the latter, which are referred to as base classes. It is intended to help reuse of existing code with little or no modification.

Moreover, inheritance is also called generalization, because the is-a relationships capture a hierarchical relationship between classes of objects. For instance, a "fruit" is a generalization of "apple", "orange", "mango" and many others. We say that fruit is an abstraction of apple, orange, etc. Conversely, we can say that since apples are fruit (i.e. an apple is-a fruit), that they inherit all the properties common to all fruit, such as being a fleshy container for the seed of a plant.

Therefore, one of the powerful advantages of inheritance is that modules with sufficiently similar interfaces can be commanded by shared controlling code, reducing the complexity of the program. Inheritance therefore has another view, a *dual*, called polymorphism, which describes many pieces of code being controlled by shared control code.

Here are some typical uses of inheritance:

- Reusing code. If two or more classes have some things in common but also differ in some ways, the common elements can be put in an a single class definition that the other classes inherit. The common code is shared and need only be implemented once.
- Setting up a protocol. A class can declare a number of methods that its subclasses are expected to implement. The class might have empty versions of the methods, or it might implement partial versions that are to be incorporated into the subclass methods. In either case, its declarations establish a *protocol* that all its subclasses must follow.

  When different classes implement similarly named methods, a program is better able to make use of polymorphism in its design. Setting up a protocol that subclasses must implement helps enforce these naming conventions.

- Delivering generic functionality. One implementor can define a class that contains a lot of basic, general code to solve a problem, but doesn't fill in all the details. Other implementors can then create subclasses to adapt the generic class to their specific needs. For example, the Appliance class in the program that models water use might define a generic water-using device that subclasses would turn into specific kinds of appliances.

  Inheritance is thus both a way to make someone else's programming task easier and a way to separate levels of implementation.

- Making slight modifications. When inheritance is used to deliver generic functionality, set up a protocol, or reuse code, a class is devised that other classes are expected to inherit from. But you can also use inheritance to modify classes that aren't intended as superclasses. Suppose, for example, that there's an object that would work well in your program, but you'd like to change one or two things that it does. You can make the changes in a subclass.

1.2.3 Encapsulation

The need of encapsulation is to protect or prevent the code (data) from accidental corruption due to the silly little errors that we are all prone to make. In Object oriented programming data is treated as a critical element in the program development and data is packed closely to the functions that operate on it and protects it from accidental modification from outside functions. Encapsulation provides a way to protect data from accidental corruption. Rather than defining the data in the form of public, we can declare those fields as private.

To design effectively at any level of abstraction, you need to be able to leave details of implementation behind and think in terms of units that group those details under a common interface. For a programming unit to be truly effective, the barrier between interface and implementation must be absolute. The interface must *encapsulate* the implementation--hide it from other parts of the program. Encapsulation protects an implementation from unintended actions and inadvertent access.

1.2.4 Polymorphism

Polymorphism means allowing a single definition to be used with different types of data (specifically, different classes of objects). For instance, a polymorphic function definition can replace several type-specific ones, and a single polymorphic operator can act in expressions of various types.

Polymorphism is the ability of objects belonging to different types to respond to method calls of methods of the same name, each one according to an appropriate type-specific behaviour. The programmer (and the program) does not have to know the exact type of the object in advance, so this behavior can be implemented at run time (this is called *late binding* or *dynamic binding*).

The different objects involved only need to present a compatible interface to the clients (the calling routines). That is, there must be public methods with the same name and the same parameter sets in all the objects. In principle, the object types may be

unrelated, but since they share a common interface, they are often implemented as subclasses of the same parent class. Though it is not required, it is understood that the different methods will also produce similar results.

Polymorphism allows client programs to be written based only on the abstract interfaces of the objects which will be manipulated (interface inheritance). This means that future extension in the form of new types of objects is easy, if the new objects conform to the original interface. In particular, with object-oriented polymorphism, the original client program does not even need to be recompiled (only relinked) in order to make use of new types exhibiting new (but interface-conformant) behaviour.

CHAPTER 2 .NET FRAMEWORK AND C#

2.1 Fundamentals Of The .NET Framework

The Microsoft .NET Framework is a component of the Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering, and is intended to be used by most new applications created for the Windows platform. The framework is intended to make it easier to develop computer applications and to reduce the vulnerability of applications and computers to security threats.

The pre-coded solutions form the framework's class library and cover a large range of programming needs in areas including the user interface, data access, cryptography, numeric algorithms, and network communications. The functions of the class library are used by programmers who combine them with their own code to produce applications.

The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

The .NET Framework has two main components: the Common Language Runtime (CLR) and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. The CLR provides the appearance of an application virtual machine, so that programmers need not consider the capabilities of the specific CPU that will execute the program. It works like an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

2.2 Common Language Infrastructure (CLI or CLR)

The most important component of the .NET Framework lies in the Common Language Infrastructure, or CLI. The purpose of the CLI is to provide a language agnostic platform for application development, including, but not limited to, components for: exception handling, garbage collection, security, and interoperability. Microsoft's implementation of the CLI is called the Common Language Runtime, or CLR. The CLI is composed of five primary parts:

- Common Type System (CTS)
- Common Language Specification (CLS)
- Common Intermediate Language (CIL)
- Just-in-Time Compiler (JIT)
- Virtual Execution System (VES)

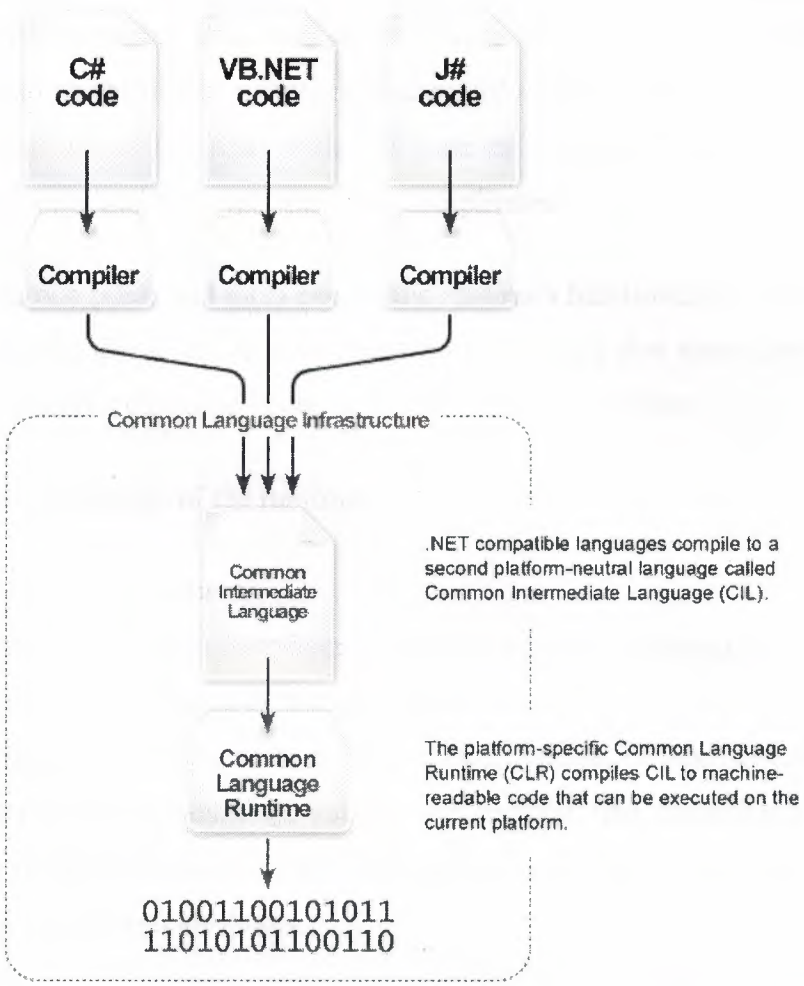

Figure 2.1 .NET Framework Common Language Infrastructure (CLI)

The common language runtime makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviors can be tightly integrated. This cross-language integration is possible because language compilers and tools that target the runtime use a common type system defined by the runtime, and they follow the runtime's rules for defining new types, as well as for creating, using, persisting, and binding to types.

As part of their metadata, all managed components carry information about the components and resources they were built against. The runtime uses this information to ensure that the component or application has the specified versions of everything it

needs, which makes the code less likely to break because of some unmet dependency. Registration information and state data are no longer stored in the registry where they can be difficult to establish and maintain. Rather, information about the types defined (and their dependencies) is stored with the code as metadata, making the tasks of component replication and removal much less complicated.

Language compilers and tools expose the runtime's functionality in ways that are intended to be useful and intuitive to developers. This means that some features of the runtime might be more noticeable in one environment than in another.

Following are some benefits of the runtime:

- Performance improvements.
- The ability to easily use components developed in other languages.
- Extensible types provided by a class library.
- New language features such as inheritance, interfaces, and overloading for object-oriented programming; support for explicit free threading that allows creation of multithreaded, scalable applications; support for structured exception handling and custom attributes.

Programmers also write managed code using the C# language, which provides the following benefits:

- Complete object-oriented design.
- Very strong type safety.
- A good blend of Visual Basic simplicity and C++ power.
- Garbage collection.
- Syntax and keywords similar to C and C++.

2.2.1 Common Type System (CTS)

The common type system supports two general categories of types, each of which is further divided into subcategories:

Value types :

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types :

Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

Variables that are value types each have their own copy of the data, and therefore operations on one variable do not affect other variables. Variables that are reference types can refer to the same object; therefore, operations on one variable can affect the same object referred to by another variable.

All types derive from the System.Object base type.

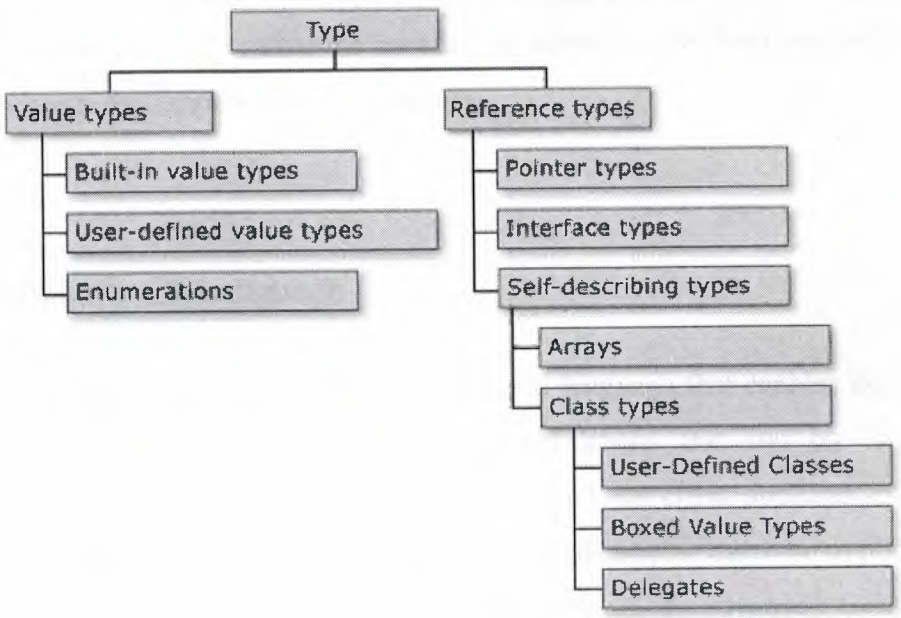

Figure 2.2 Type Classification in Common Type System

2.3 .NET Framework Class Library

The .NET Framework includes classes, interfaces, and value types that expedite and optimize the development process and provide access to system functionality. To facilitate interoperability between languages, the .NET Framework types are CLS-compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS).

The .NET Framework types are the foundation on which .NET applications, components, and controls are built. The .NET Framework includes types that perform the following functions:

- Represent base data types and exceptions.
- Encapsulate data structures.
- Perform I/O.
- Access information about loaded types.
- Invoke .NET Framework security checks.
- Provide data access, rich client-side GUI, and server-controlled, client-side GUI.

The .NET Framework provides a rich set of interfaces, as well as abstract and concrete (non-abstract) classes. You can use the concrete classes as is or, in many cases, derive your own classes from them. To use the functionality of an interface, you can either create a class that implements the interface or derive a class from one of the .NET Framework classes that implements the interface.

2.4 Introduction to C# Language

C# is an elegant and type-safe object-oriented language that enables developers to build a wide range of secure and robust applications that run on the .NET Framework. You can use C# to create traditional Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more. Microsoft Visual C# 2005 provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to

facilitate rapid application development based on version 2.0 of the C# language and the .NET Framework.

C# syntax is highly expressive, yet with less than 90 keywords, it is also simple and easy to learn. The curly-brace syntax of C# will be instantly recognizable to anyone familiar with C, C++ or Java. Developers who know any of these languages are typically able to begin working productively in C# within a very short time. C# syntax simplifies many of the complexities of C++ while providing powerful features such as nullable value types, enumerations, delegates, anonymous methods and direct memory access, which are not found in Java. C# also supports generic methods and types, which provide increased type safety and performance, and iterators, which enable implementers of collection classes to define custom iteration behaviors that are simple to use by client code.

As an object-oriented language, C# supports the concepts of encapsulation, inheritance and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the override keyword as a way to avoid accidental redefinition. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, C# facilitates the development of software components through several innovative language constructs, including:

- Encapsulated method signatures called delegates, which enable type-safe event notifications.
- Properties, which serve as accessors for private member variables.
- Attributes, which provide declarative metadata about types at run time.
- Inline XML documentation comments.

Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code, along with resources such as bitmaps and strings, is stored on disk in an executable file called an assembly, typically with an

extension of .exe or .dll. An assembly contains a manifest that provides information on the assembly's types, version, culture, and security requirements.

When the C# program is executed, the assembly is loaded into the CLR, which might take various actions based on the information in the manifest. Then, if the security requirements are met, the CLR performs just in time (JIT) compilation to convert the IL code into native machine instructions. The CLR also provides other services related to automatic garbage collection, exception handling, and resource management. Code that is executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code" which is compiled into native machine language that targets a specific system. The following diagram illustrates the compile-time and run time relationships of C# source code files, the base class libraries, assemblies, and the CLR.

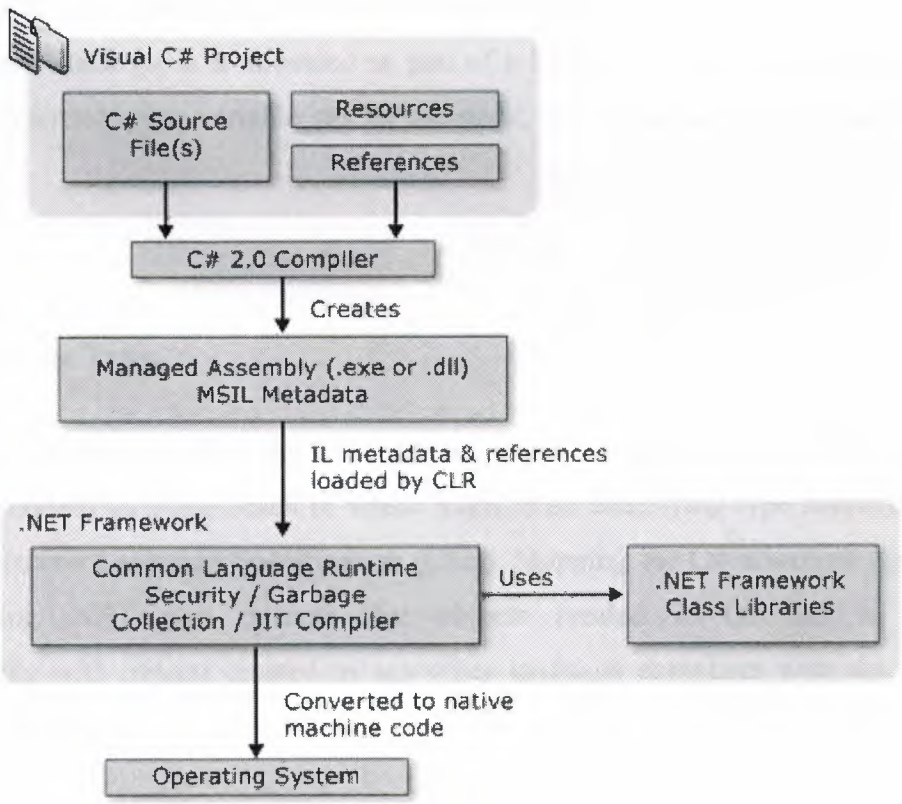

Figure 2.3 Compilation process of C# source code by the .NET Framework

2.5 C# Data Types

C# is a strongly typed language. In a strongly typed language you must declare the type of each object you create (e.g., integers, floats, strings, windows, buttons, etc.) and the compiler will help you prevent bugs by enforcing that only data of the right type is assigned to those objects. The type of an object signals to the compiler the size of that object (e.g., int indicates an object of 4 bytes) and its capabilities (e.g., buttons can be drawn, pressed, and so forth).

Like C++ and Java, C# divides types into two sets: *intrinsic* (built-in) types that the language offers and *user-defined* types that the programmer defines.

C# also divides the set of types into two other categories: *value* types and *reference* types.[11] The principal difference between value and reference types is the manner in which their values are stored in memory. A value type holds its actual value in memory allocated on the stack (or it is allocated as part of a larger reference type object). The address of a reference type variable sits on the stack, but the actual object is stored on the heap.

2.5.1 Built-in Data Types

The C# language offers the usual cornucopia of intrinsic (built-in) types one expects in a modern language, each of which maps to an underlying type supported by the .NET Common Language Specification (CLS). Mapping the C# primitive types to the underlying .NET type ensures that objects created in C# can be used interchangeably with objects created in any other language compliant with the .NET CLS, such as VB .NET.

Table 2.1 Built-in Data Types of the C# Language

Data type	Size in Bytes	Mapped to	Description
sbyte	1	System.SByte	Signed (values -128 to 127).
byte	1	System.Byte	Unsigned (values 0-255)
short	2	System.Int16	Signed (short) (values -32,768 to 32,767).
ushort	2	System.UInt16	Unsigned (short) (values 0 to 65,535).
int	4	System.Int32	Signed integer values between -2,147,483,647 and 2,147,483,647.
uint	4	System.UInt32	Unsigned integer values between 0 and 4,294,967,295.
long	8	System.Int64	Signed integers ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
ulong	8	System.UInt64	Unsigned integers ranging from 0 to 0xffffffffffffffff.
char	1	System.Char	Unicode characters
float	4	System.Single	Floating point number. Holds the values from approximately +/-1.5 * 10^{-45} to approximate +/-3.4 * 10^{38} with 7 significant figures.
double	8	System.Double	Double-precision floating point; holds the values from approximately +/-5.0 * 10^{-324} to approximate +/-1.7 * 10^{308} with 15-16 significant figures.
bool	1	System.Boolean	true or false.
decimal	8	System.Decimal	Fixed-precision up to 28 digits and the position of the decimal point. This is typically used in financial calculations. Requires the suffix "m" or "M."

CHAPTER 3. REQUIREMENT ANALYSIS

3.1 Introducing the UML

As the world becomes more complex, the computer-based systems that inhabit the world also must increase in complexity. They often involve multiple pieces of hardware and software, networked across great distances, linked to databases that contain mountains of information.

The key is to organize the design process in a way that clients, analysts, programmers and other involved in system development can understand and agree on. The UML provides the organization.

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

3.2 Goals of UML

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.

5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

The UML certain number of graphical elements combined into diagrams. Because it is a language, the UML has rules for combining these elements.

The purpose of the diagrams is to present multiple views of a system, and this set of multiple views is called a model.

3.3 UML Diagrams

Class Diagram

Things naturally fall into categories (computers, automobiles, trees...). We refer to these categories as classes. Class diagram provide the representations used by the developers. UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Names of abstract classes are in italics. Relationships between classes are the connecting links.

association -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
aggregation -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole.
generalization -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass.

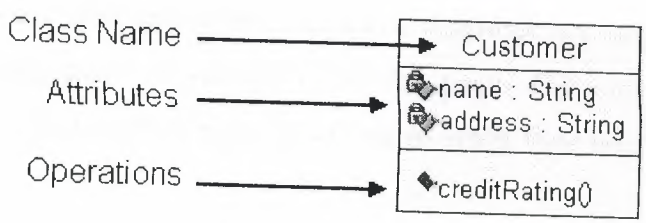

Figure 3.1 An Example Class Diagram

Use Case Diagram

A use case is a description of a system's behavior from a user's standpoint. For system developers, this is a valuable tool: it's a tried-and-true technique for gathering system requirements from a user's point of view. That's important if the goal is to build a system that real people can use. In graphical representations of use cases a symbol for the actor is used .

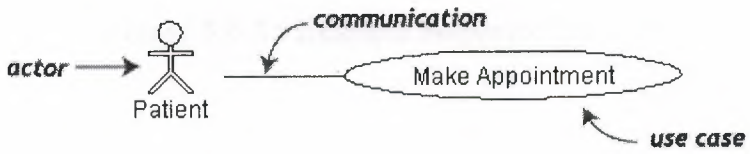

Figure 3.2 An Example Use Case Diagram

State Diagram

At any given time, an object is in particular state. State diagrams represent these states, and their changes during time. Every state diagram starts with symbol that represents start state, and ends with symbol for the end state.

Sequence Diagram

Class diagrams and object diagrams represent static information. In a functioning system, however, objects interact with one another, and these interactions occur over time. The UML sequence diagram shows the time-based dynamics of the interaction.
A sequence diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

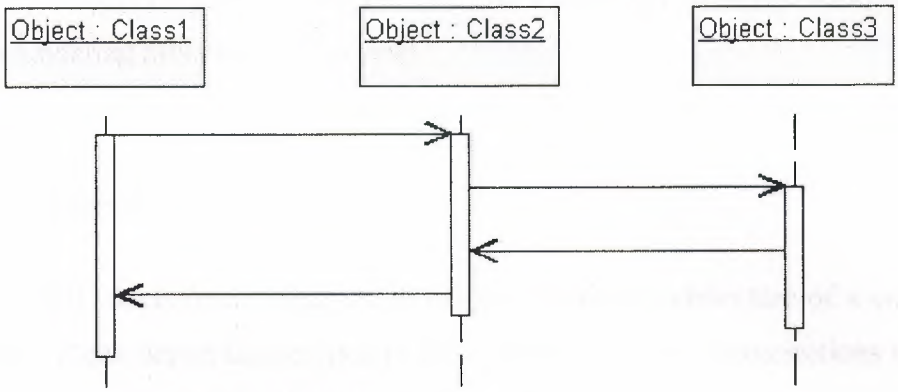

Figure 3.3 An Example Sequence Diagram

Activity Diagram

The activities that occur within a use case or within an object's behavior typically occur in a sequence. This sequence is represented with activity diagrams.

An activity diagram is essentially a fancy flowchart. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

Collaboration Diagram

The elements of a system work together to accomplish the system's objectives, and a modeling language must have a way of representing this. The UML collaboration diagram is designed for this purpose.

Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links.

Component Diagram

Today in software engineering we have team-based development efforts, where everyone has to work on different component. That's important to have a component diagram in modeling process of the system.

Deployment Diagram

The UML deployment diagram shows the physical architecture of a computer-based system. It can depict the computers and devices, show their connections with one another, and show the software that sits on each machine.

3.4 Classes Used In the Project

In a software developed using an Object Oriented Programming Language like C#, design process is one of the most important processes. Any mistake in design of the program will result some problems which need a lot of effort to recover. Using classes helps programmer to structure the software easily and manage it without much work. Maintainability another important aspect of a software. Design decisions must always consider maintainability issues of the program.

3.4.1 ComponentBase Class

This project uses several classes for executing various tasks. ComponentBase class is an abstract class which means it is not used as an instance and only served as a base class for the other component classes such as Mainboard or Cpu. It contains common attributes for all of the components in the stocks like price, retail_price, quantity and brand.

By inheritance, other component classes inherit these attributes and also their get() and set() methods.

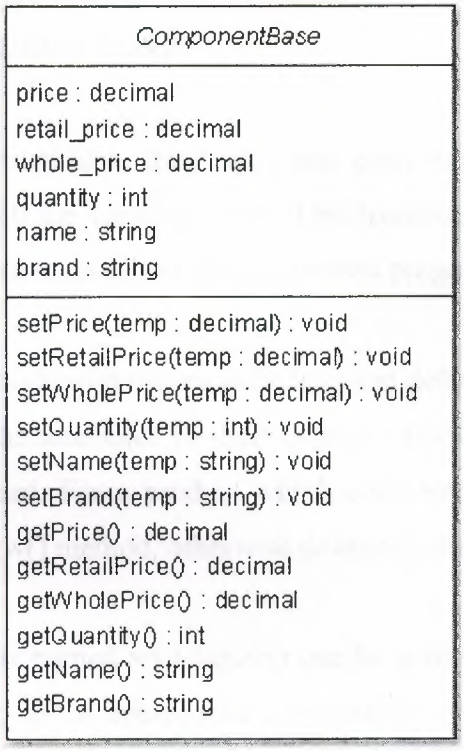

Figure 3.4 Class Diagram of the ComponentBase Class

3.4.2 Mainboard Class

The Mainboard class is used for representing the mainboards in the system. While storing an item to the database or retrieving an item from the database, program uses an instance of this class to represent that complex data type.

Attribute named 'cpu' holds the brand and type of the cpus that particular mainboard represents. It would contain LGA775 or Socket478 for the Intel processors for example. Keeping this data is important because it will be used in further operations in the program.

Attribute named 'graphics' holds the type of interface which the motherboard supports.It would contain AGP of PCI-E as the slot name on the mainboard.

Attribute 'hdd' holds the name of the interface which the motherboards supports for hard drive connections. For instance it would be SATA, SATA2, or IDE.

Attribute memory holds the memory module type that particular mainboard can operate. It would be SD, DDR or DDR2.

All of these attributes have their set() and get() methods for modifying and retrieving the data in which the variable holds. This technique provides encapsulation, which is a very important concept of the object oriented programming.

There are two methods with names defaults() and defaultsNew(). There methods are used for initializing the attributes to their default values depending on where the instance of the class is used. For a product which considered as new for the system, object calls the defaultsNew() method, otherwise defaults() method will be called.

Also there is a class named setAdapter() can be seen in the class diagram. This method is used for setting up the appropriate connections and filling the database with the correct information.

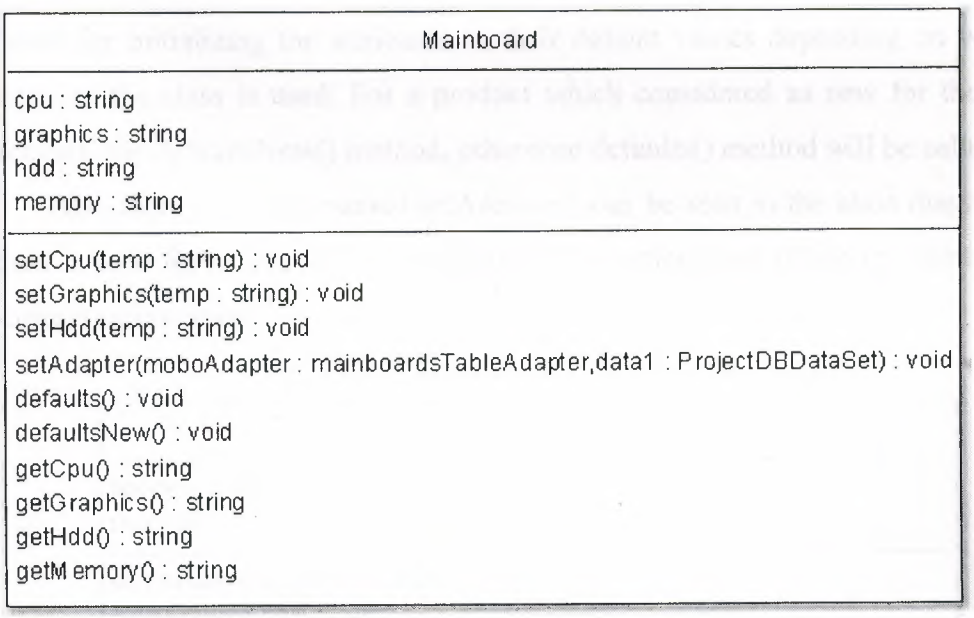

Figure 3.5 Class Diagram of the Mainboard Class

3.4.3 Cpu Class

The Cpu class is used for representing the processors in the system. While storing an item to the database or retrieving an item from the database, program uses an instance of this class to represent that complex data type.

Attribute named 'socket' holds the socket type of the processor which the object represents. It would contain LGA775 or Socket478 for the Intel processors for example. Keeping this data is important because it will be used in further operations in the program along with the data from the mainboards table.

Attribute named 'mhz' which is type of integer holds the maximum speed of the processeor in Mhz. It would contain an integer value like 3000, 2800 etc.

All of these attributes have their set() and get() methods for modifying and retrieving the data in which the variable holds. This technique provides encapsulation, which is a very important concept of the object oriented programming.

There are two methods with names defaults() and defaultsNew(). There methods are used for initializing the attributes to their default values depending on where the instance of the class is used. For a product which considered as new for the system, object calls the defaultsNew() method, otherwise defaults() method will be called.

Also there is a class named setAdapter() can be seen in the class diagram. This method is used for setting up the appropriate connections and filling the database with the correct information.

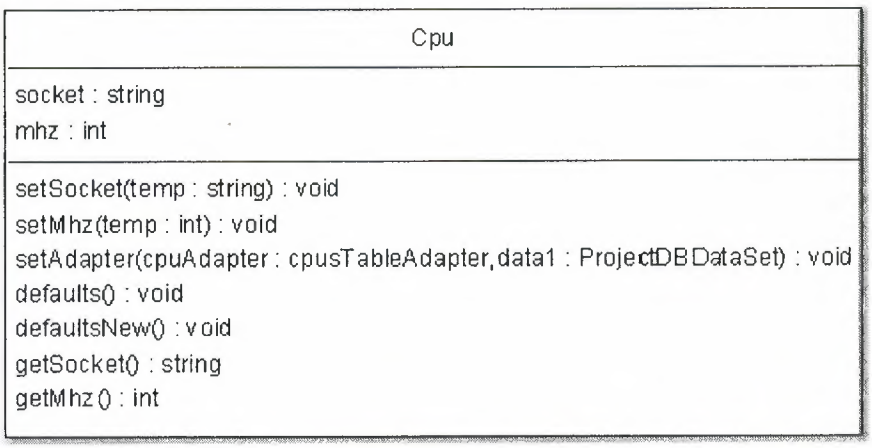

Figure 3.6 Class Diagram of the Cpu Class

Those two classes are subclasses of the ComponentBase Class. That means the two class also implements the methods of their parent class such as setPrice(), setQuantity() and so on. This is the very neat way of applying object oriented programming techniques to the project.

There are 13 component classes like Mainboard and Cpu, and one ComponentBase class. All of these 13 classes are subclassees of the ComponentBase class , therefore they share the attributes and methods of the ComponentBase class ass the common attributes and methods. If there would be some modifications needed in the products information schema, it would be easily accomplished by applying changes only to base class without touching any of the subclasses.

3.4.4 Computer Class

Computer class is used for representing a newly created computer regarding the needs of the customer by an employee. It shows all of the information about the new computer to provide a complex data type which can be used when the data need to be recorded to the database. It holds all components in the attributes as a string and a total price of the computer in a float. get() and set() methods are also implemented for encapsulating the data.

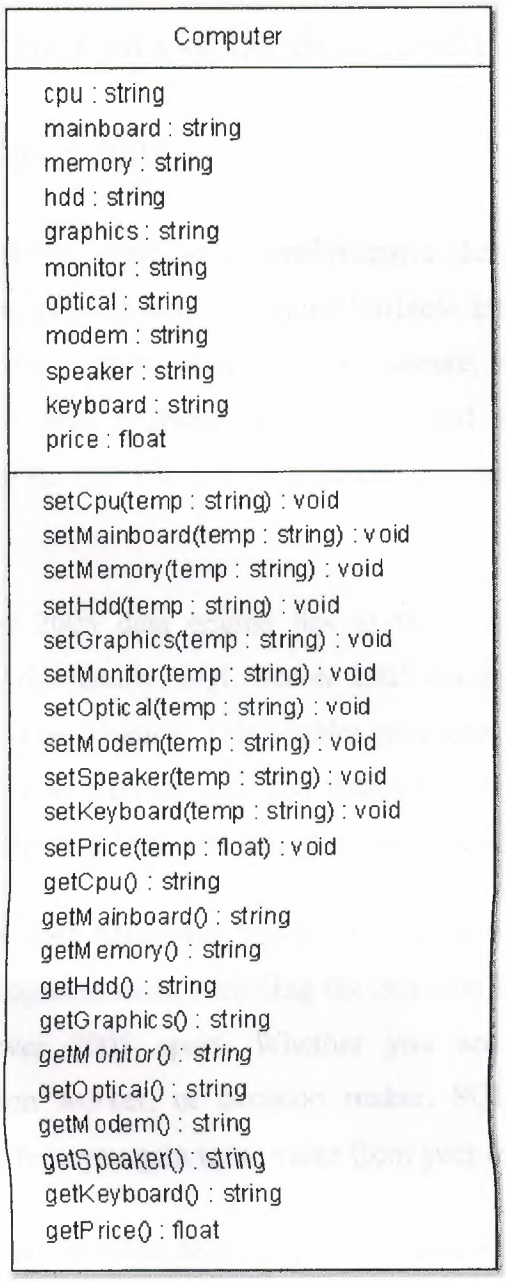

Figure 3.7 Class Diagram of the Computer Class

CHAPTER 4. DATABASE DESIGN & SQL SERVER 2005

4.1 Microsoft SQL Server 2005

Microsoft SQL Server 2005 is a comprehensive database platform providing enterprise-class data management with integrated business intelligence (BI) tools. The SQL Server 2005 database engine provides more secure, reliable storage for both relational and structured data, enabling you to build and manage highly available, performant data applications that you and your people can use to take your business to the next level.

The SQL Server 2005 data engine lies at the core of this enterprise data management solution. Additionally, SQL Server 2005 combines the best in analysis, reporting, integration, and notification. This enables your team to build and deploy cost-effective BI solutions with which they can drive data into every corner of your business through scorecards, dashboards, Web services, and mobile devices.

Close integration with Microsoft Visual Studio, the Microsoft Office System, and a suite of new development tools, including the Business Intelligence Development Studio, sets SQL Server 2005 apart. Whether you are a developer, database administrator, information worker, or decision maker, SQL Server 2005 provides innovative solutions that help you gain more value from your data.

Figure below shows the layout of the SQL Server 2005 data platform.

Figure 4.1 SQL Server 2005 data platform layout on the Windows Environment

4.2 Features of Microsoft SQL Server 2005

Database Engine

The Database Engine is the core service for storing, processing and securing data. The Database Engine provides controlled access and rapid transaction processing to meet the requirements of the most demanding data consuming applications within your enterprise. The Database Engine also provides rich support for sustaining high availability.

Analysis Services

Analysis Services delivers online analytical processing (OLAP) and data mining functionality for business intelligence applications. Analysis Services supports OLAP by allowing you to design, create, and manage multidimensional structures that contain data aggregated from other data sources, such as relational databases. For data mining applications, Analysis Services enables you to design, create, and visualize data mining models. These mining models can be constructed from other data sources by using a wide variety of industry-standard data mining algorithms.

Integration Services

Integration Services is a platform for building high performance data integration solutions, including packages that provide extract, transform, and load (ETL) processing for data warehousing. Integration Services includes a rich set of built-in tasks and transformations; tools for constructing packages; and the Integration Services service for running and managing packages.

Replication

Replication is a set of technologies for copying and distributing data and database objects from one database to another, and then synchronizing between databases to maintain consistency. By using replication, you can distribute data to different locations and to remote or mobile users by means of local and wide area networks, dial-up connections, wireless connections, and the Internet.

Reporting Services

Reporting Services delivers enterprise, Web-enabled reporting functionality so you can create reports that draw content from a variety of data sources, publish reports in various formats, and centrally manage security and subscriptions.

Notification Services

Notification Services is an environment for developing and deploying applications that generate and send notifications. You can use Notification Services to generate and send timely, personalized messages to thousands or millions of subscribers, and can deliver the messages to a variety of devices.

Service Broker

Service Broker helps developers build scalable, secure database applications. This new Database Engine technology provides a message-based communication platform that enables independent application components to perform as a functioning whole. Service Broker includes infrastructure for asynchronous programming that can be used for applications within a single database or a single instance, and also for distributed applications.

Full-Text Search

Full-Text Search contains the functionality you can use to issue full-text queries against plain character-based data in SQL Server tables. Full-text queries can include words and phrases, or multiple forms of a word or phrase.

4.3 Structured Query Language (SQL)

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

There are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

4.3.1 SQL Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.

The most important DDL statements in SQL are:

- CREATE TABLE - creates a new database table
- ALTER TABLE - alters (changes) a database table
- DROP TABLE - deletes a database table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

4.3.2 SQL Data Manipulation Language (DML)

SQL (Structured Query Language) is a syntax for executing queries. But the SQL language also includes a syntax to update, insert, and delete records.

These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- SELECT - extracts data from a database table
- UPDATE - updates data in a database table
- DELETE - deletes data from a database table
- INSERT INTO - inserts new data into a database table

The SQL SELECT Statement :

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

```
SELECT column_name(s)
FROM table_name
```

To select all columns from the "Persons" table, a * symbol used instead of column names, like this:

```
SELECT * FROM Persons
```

When the list of columns following the SELECT SQL command is replaced with asterix (*) all table columns are returned. Word of caution here, it's always better to explicitly specify the columns in the SELECT list, as this will improve your query performance significantly.

The DISTINCT keyword is used to return only distinct (different) values.

The SELECT statement returns information from table columns. DISTINCT keyword returns only the distinct elements

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

```
SELECT DISTINCT column_name(s)
FROM table_name
```

```
SELECT DISTINCT Company FROM Orders
```

The WHERE Clause :

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

```
SELECT column FROM table
WHERE column operator value
```

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Table 4.1 SQL Operators that can be used in combination with WHERE clause

To select only the persons living in the city "Sandnes", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

SQL uses single quotes around text values (most database systems will also accept double quotes). Numeric values should not be enclosed in quotes.

For text values:

```
SELECT * FROM Persons WHERE FirstName='Tove'
```

For numeric values:

```
SELECT * FROM Persons WHERE Year>1965
```

The LIKE Condition:

The LIKE condition is used to specify a search for a pattern in a column.

```
SELECT column FROM table
WHERE column LIKE pattern
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

The INSERT INTO Statement:

The INSERT INTO statement is used to insert new rows into a table.

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

Also the columns to be changed can be specified.

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

```
INSERT INTO Persons
VALUES ('Mesut', 'ARIK', 'LEFKOSA', 'Near East University')
```

The Update Statement :

The UPDATE statement is used to modify the data in a table.

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

```
UPDATE Person SET FirstName = 'Mesut'
WHERE LastName = 'ARIK'
```

To update several rows in one query:

```
UPDATE Person
SET Address = 'NEU, City = 'LEFKOSA'
WHERE LastName = 'ARIK'
```

The DELETE Statement :

The DELETE statement is used to delete rows in a table.

```
DELETE FROM table_name
WHERE column_name = some_value
```

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
or
DELETE * FROM table_name
```

ORDER BY Statement :

The ORDER BY clause is used to sort the rows.

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

For example, to display the companies in alphabetical order AND the ordernumbers in numerical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

AND & OR :

AND and OR join two or more conditions in a WHERE clause.

The AND operator displays a row if ALL conditions listed are true. The OR operator displays a row if ANY of the conditions listed are true.

```
SELECT * FROM Persons
WHERE FirstName='Mesut'
AND LastName='ARIK'
```

Also OR an AND operators can be combined in single query.

```
SELECT * FROM Persons WHERE
(FirstName='Mesut' OR FirstName='Murat')
AND LastName='ARIK'
```

BETWEEN ... AND :

The BETWEEN ... AND operator selects a range of data between two values. These values can be numbers, text, or dates.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

```
SELECT * FROM Persons
WHERE LastName
BETWEEN 'ARIK' AND 'KAYA'
```

4.3.3 SQL Join Operator

Tables in a database can be related to each other with keys. A primary key is a column with a unique value for each row. The purpose is to bind data together, across tables, without repeating all of the data in every table.

The *join* clause combines columns of one table to that of another to create a single table. Join matches up a column with one table to a column in another table. A join query does not alter either table, but temporarily combines data from each table to be viewed as a single table. There are three types of join statements, inner, left, and right.

Inner Join: An inner join returns all rows that result in a match.

```
SELECT employees.Lastname, employees.Firstname, invoices.Sale,
invoices.Price
FROM employees
INNER JOIN invoices
ON employees.id = invoices.EmployeeID
```

Left Join: A Left join returns all rows of the left of the conditional even if there is no right column to match.

```
SELECT field1, field2, field3
FROM first_table
LEFT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Right Join: A right join will display rows on the right side of the conditional that may or may not have a match.

```
SELECT field1, field2, field3
FROM first_table
RIGHT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

4.4 Data Tables Used In Project

customers

This is the table where customer information and sales information about shopping is stored. Also the employee name who did that particular sales transaction is in this table.

itemsadded

New items added to stock are recorded in this table along with the employee name who entered that item.

saleditems

Items sold are stored in this table as individual entities like they are displayed in the receipt. Also the name of the item is stored.

users

This table is only accessible by administrators control panel. All employee information and permissions to the system is stored here.

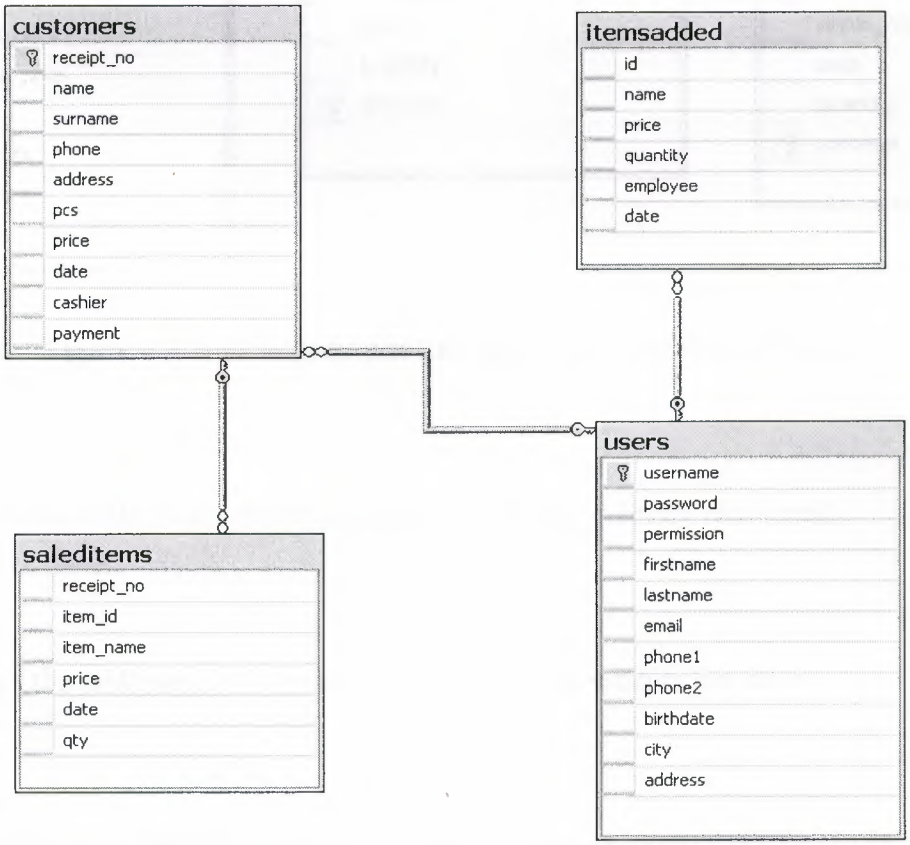

Figure 4.2 Main database tables and relationships between them.

mainboards

This the table where the product records of type mainboard are stored.

printers

This the table where the product records of type printer are stored.

monitors

This the table where the product records of type monitor are stored.

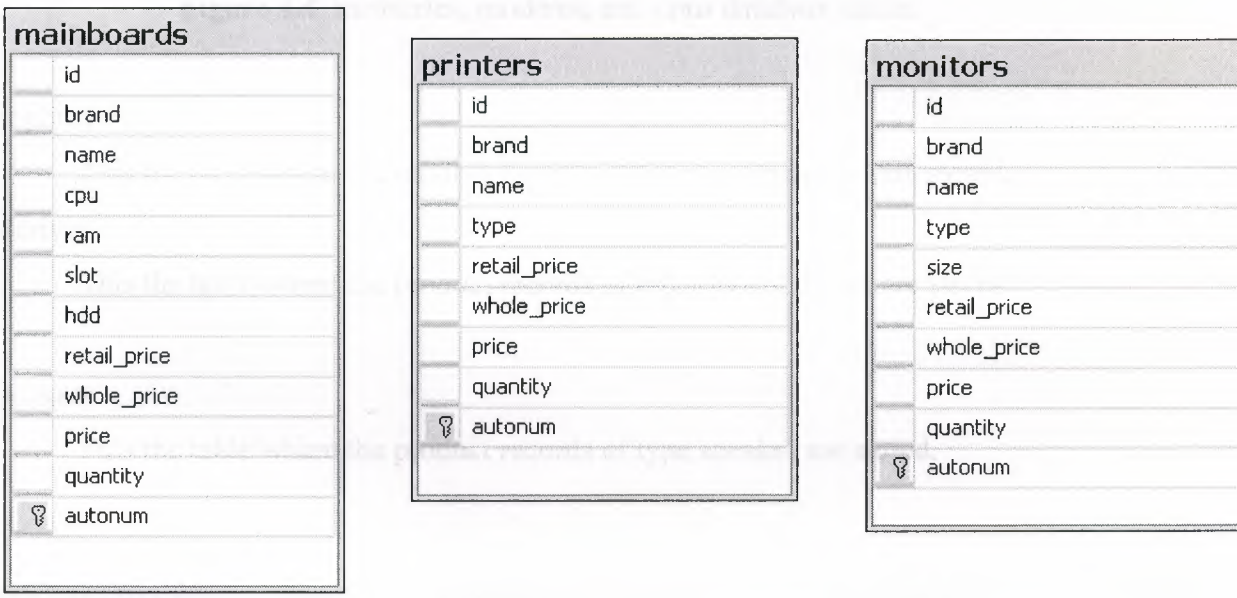

Figure 4.3 mainboards, printers, and printers database tables.

memories

This the table where the product records of type memory are stored.

modem

This the table where the product records of type modem are stored.

cpus

This the table where the product records of type processor are stored.

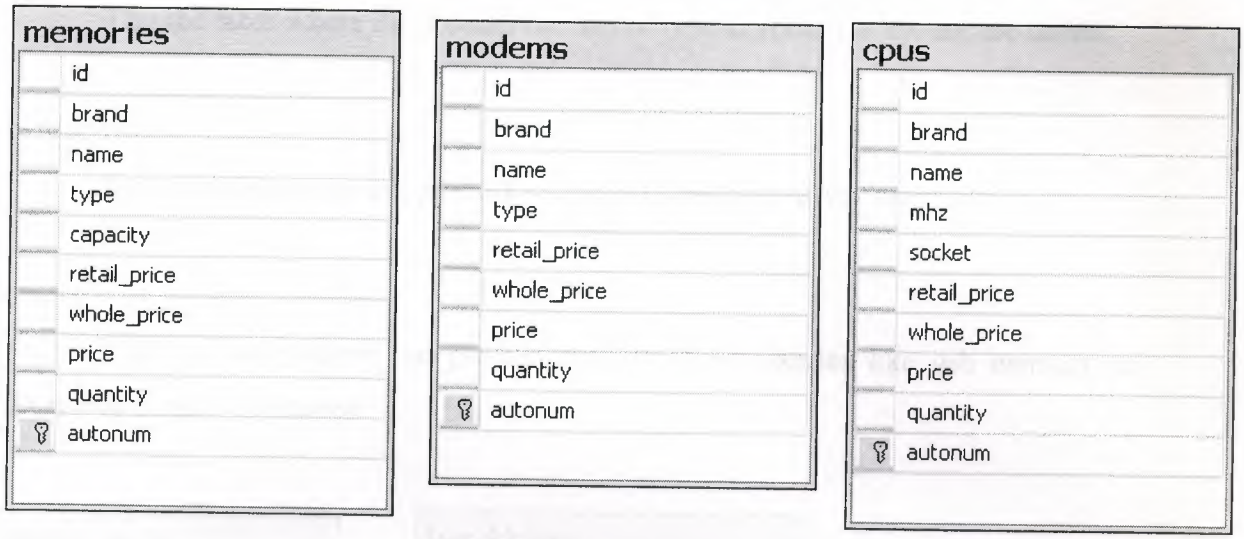

Figure 4.4 memories, modems, and cpus database tables.

graphics

This the table where the product records of type graphics card are stored.

cddrives

This the table where the product records of type optical drives are stored.

speakers

This the table where the product records of type speaker are stored.

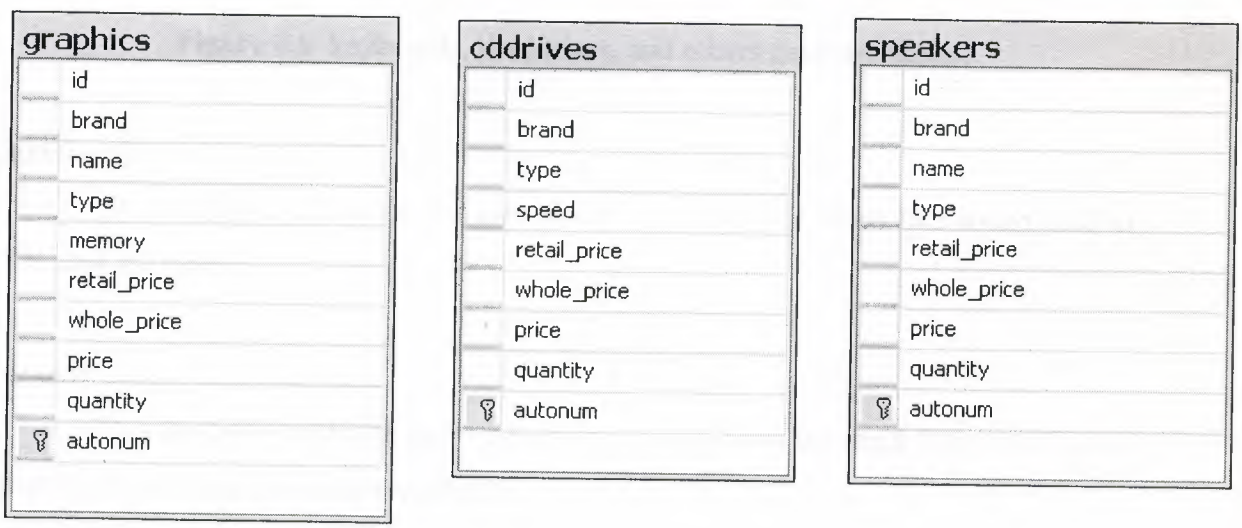

Figure 4.5 graphics, cddrives, and speakers database tables.

keyboards

This the table where the product records of type keyboard or mouse are stored.

harddrives

This the table where the product records of type hard drives are stored.

others

This the table where the product records of accessories like usb memory or bluetooth dongle are stored.

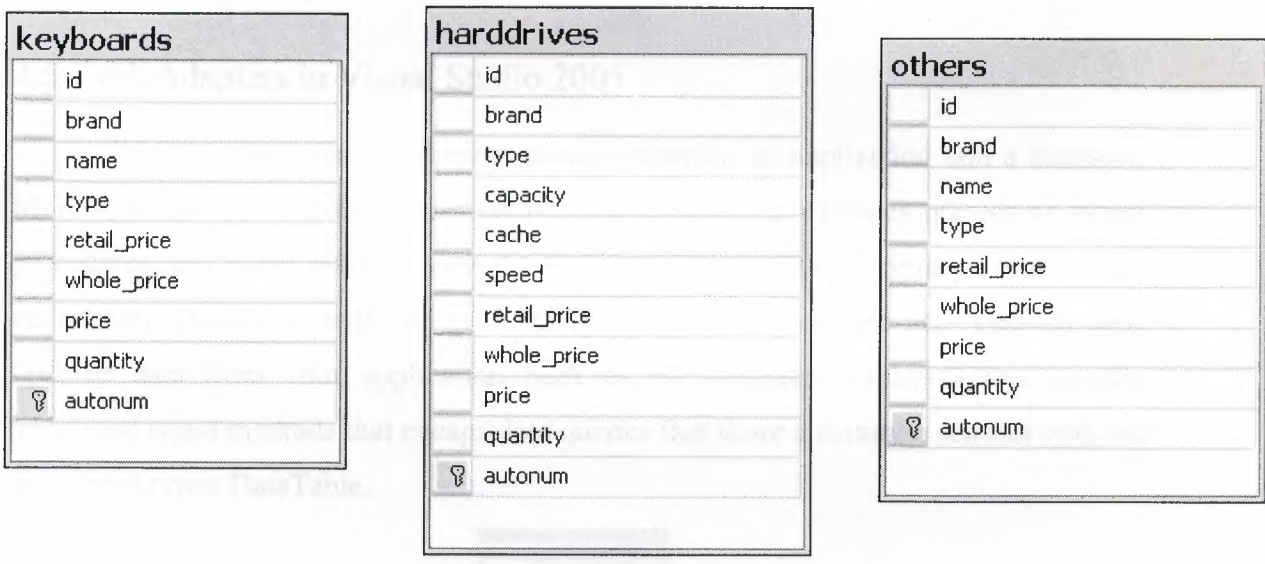

Figure 4.6 keyboards, harddrives, and others database tables.

additionals

This the table where the product records of additional cards like sound card or TV card are stored.

deleted

This the table holds the items which are deleted from the stock because of some problems about the products availability.

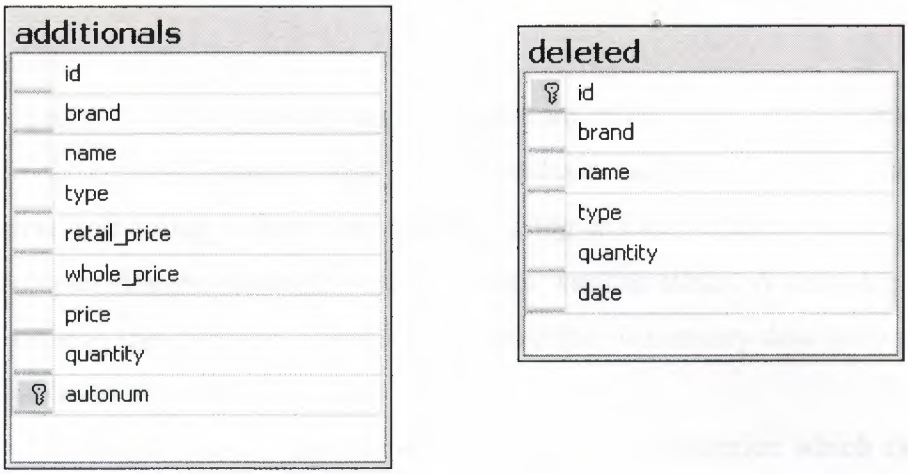

Figure 4.7 additionals and deleted database tables.

4.5 TableAdapters in Visual Studio 2005

TableAdapters provide communication between an application and a database. More specifically, a TableAdapter connects to a database, executes queries or stored procedures, and either returns a new data table populated with the returned data or fills an existing DataTable with the returned data. TableAdapters are also used to send updated data from your application back to the database. TableAdapters provide additional typed methods that encapsulate queries that share a common schema with the associated typed DataTable.

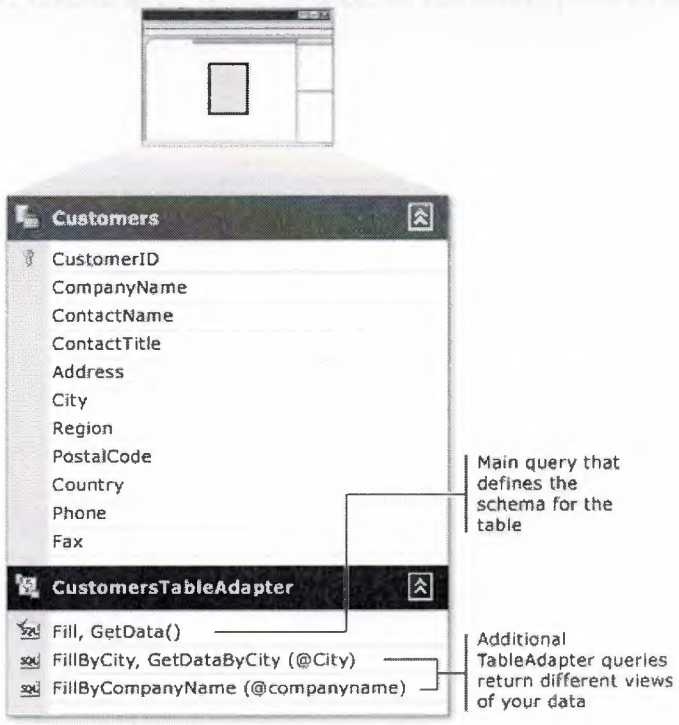

Figure 4.8 A Sample TableAdapter

Tables in which product specifications are stored must be associated with each other in some way. Visual Studio 2005 provides a neat way to do this by creating a DataTable and filling it from the database using a TableAdapter. Using a sql query, TableAdapter fills the datatable with data from various tables. A datatable can be used like a table in the database, despite it represent the in memory data gathered from the actual tables of the database.

The project uses a DataTable with the name idandprice which is filled with idandpriceTableAdapter TableAdapter. It contains id numbers and prices of all items in the tables mainboards, cpus, memories, harddrives, monitors, cddrives, printers, graphics, modems, speakers, , keyboards, additionals and others

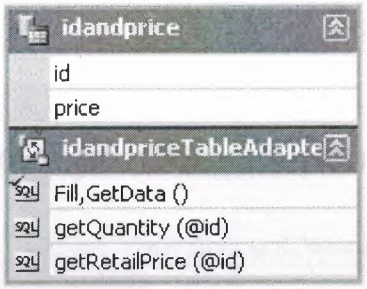

Figure 4.9 idandpriceTableAdapter, which is used for retrieving price of an item.

This project is written in C# programming language with Visual C # 2005 Express Editions. It is based on .NET Framework 2.0 which provides latest technologies in the Windows Operating System. Microsoft SQL Server 2005 Express Edition is used as a Relational Database Management System (RDBMS).

The software can be used in any computer running Microsoft Windows Operating System with .NET Framework 2.0 installed.

5.1 Login Window

When the application is started, it greets the user with the login window which is asking an authorized user name and password. If user enters wrong username or password or leaves the text boxes blank, a popup message will notify the user.

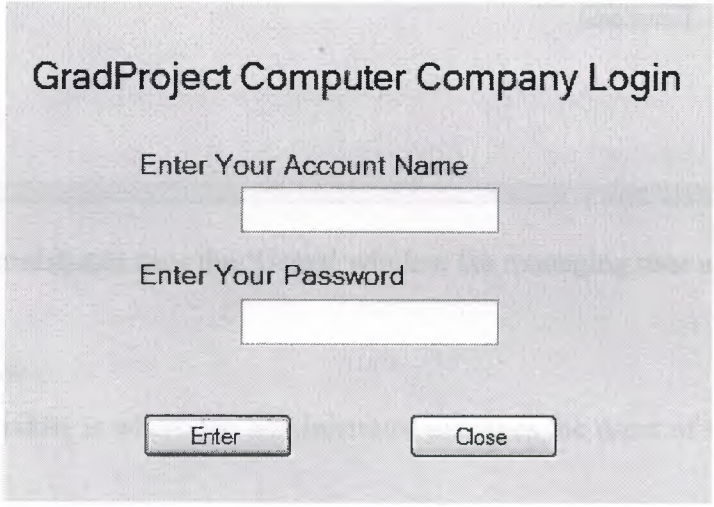

Figure 5.1 Login Window of The Program

5.2 Administrator's Panel

System operates different depending on the users' permission. A user may have an administrator account, a manager for example or the user may belong to the sales department, who is responsible for sales transactions. If the user logs in to the system with an administrator account, window in the below figure will come up. Administrator privileges can be used for doing managerial tasks in the company. Managing user accounts and products are the two most important of these tasks.

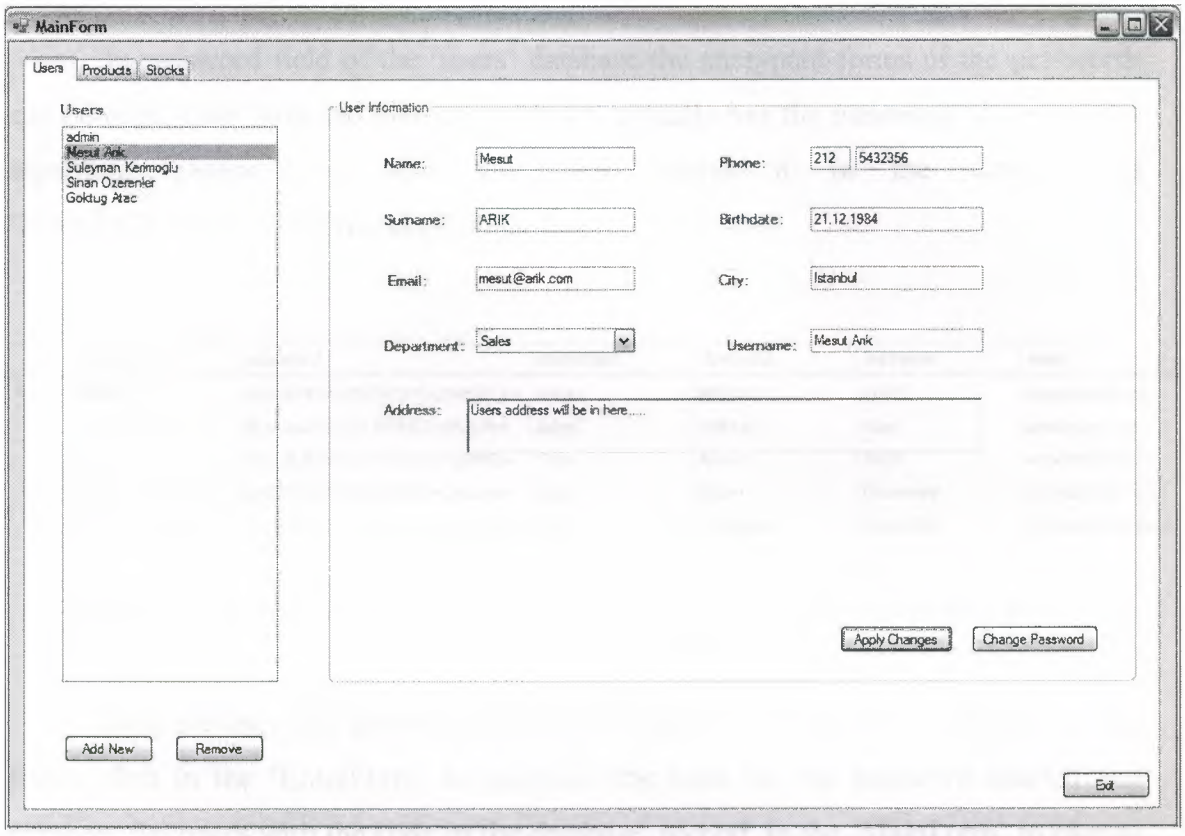

Figure 5.2 Administrator uses the 'Users' window for managing user accounts.

5.2.1 'Users' Window

The users window is where the administrator manages the users of the system in the company.

Manager can add new user to the system.

Manager can remove an existing user from the system

Manager can change a user's permission, to grant him/her different privileges.

Manager can modify a user's information, such as address, phone number or email address.

Manager can change the password of a user.

For the security reasons, users' passwords are stored in the database using Secure Hash Algorithm (SHA1) encryption.

The Secure Hash Algorithm takes a message of less than 2^{64} bits in length and produces a 160-bit message digest which is designed so that it should be computationally expensive to find a text which matches a given hash.

In password field of the 'users' database the encrypted forms of the passwords can be seen. User with the username 'Mesut' actually has the password '2' but SHA1 algorithm generates a hash code and stores it in the database as 'NWoZK3kTsExUV00Ywo1G5jlUKKs='

username	password	permission	firstname	lastname	email
admin	2kuSN7rMzfGcB2DKt67EqDWQELA=	Admin	MESUT2	ARIK2	saysalone@hot...
Goktug Atac	d95o2uzYI7q7tY7bHI4U1xBug7s=	Sales	Göktuğ	Ataç	gok@atac.com
Mesut	NWoZK3kTsExUV00Ywo1G5jlUKKs=	Sales	Mesut	ARIK	mesut@arik.com
Sinan Ozerenler	G2RTiSRzpGfQc3LUXrBavCAxZHo=	Sales	Sinan	Özerenler	sinan@insan.com
Suleyman Kerim...	Od+HUoMxjTGv5aP/Sg4yU+IEXkM=	Sales	Süleyman	Kerimoğlu	kerimoglus@sulo...

Figure 5.3 A view from the 'users' table showing SHA1 digests of passwords

Here are the codes generating the SHA1 digest. This function is called from two forms, first in the 'EnterForm' to calculate the hash for the password entered and compare the result with the code in the database, second, in the 'MainForm' to change the password of an existing user. Function takes a string as the parameter, which is the original data which should be hashed, and returns a string which contains the hashed code for the original data.

```
public static string getSHA1(string value)
   {
      System.Security.Cryptography.SHA1CryptoServiceProvider sha
       = new System.Security.Cryptography.SHA1CryptoServiceProvider();

      byte[] byteRepresentation = nicodeEncoding.UTF8.GetBytes(value);

      byte[] hashedPwdInBytes = sha.ComputeHash(byteRepresentation);

      string hashedPassword =Convert.ToBase64String(hashedPwdInBytes);

      return hashedPassword;
   }
```

5.2.2 'Products' Window

When user clicks the Products tab, a window that lets the user to select components by category will appear. In this window user can select products form 13 category. On the upper-right corner of the window there is a button named 'Options', when user clicks this button, bottom side of the frame expands and various filtering options for the currently selected category appears. By clicking those radio buttons user can filter out some products which are not suitable for the customers needs.

Selecting a product from grid and right-clicking it opens a popup menu which has two items, Edit Item and Delete Item.

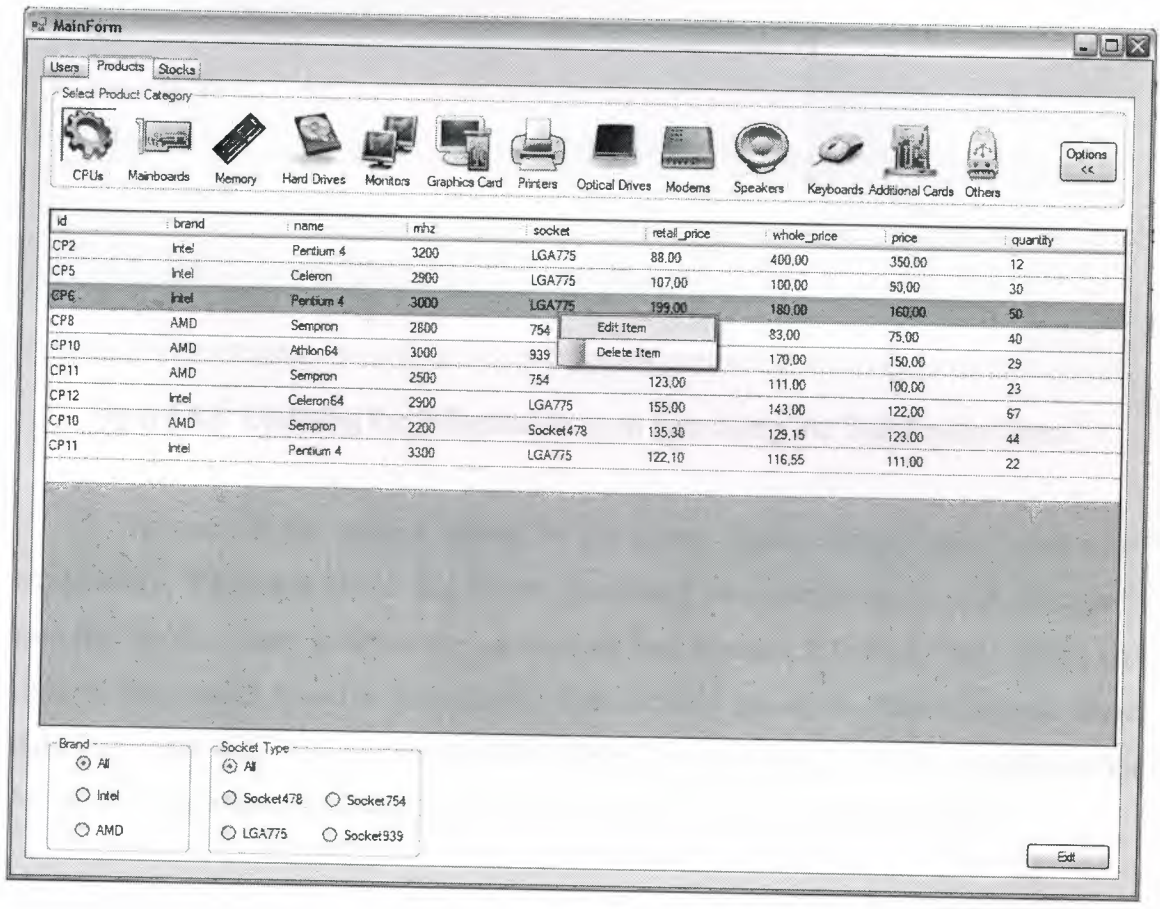

Figure 5.4 'Products' window

As its name implies 'Edit Item' lets the user to edit selected products specifications. When user clicks this option a new window, EditProductForm appears. In this window user can edit all information about the product and also change the current quantity of product in the stock. After making some changes user must click

'Apply Changes' button to confirm the changes and write to the database, or click 'Cancel' button to close EditProductForm window without changing anything.

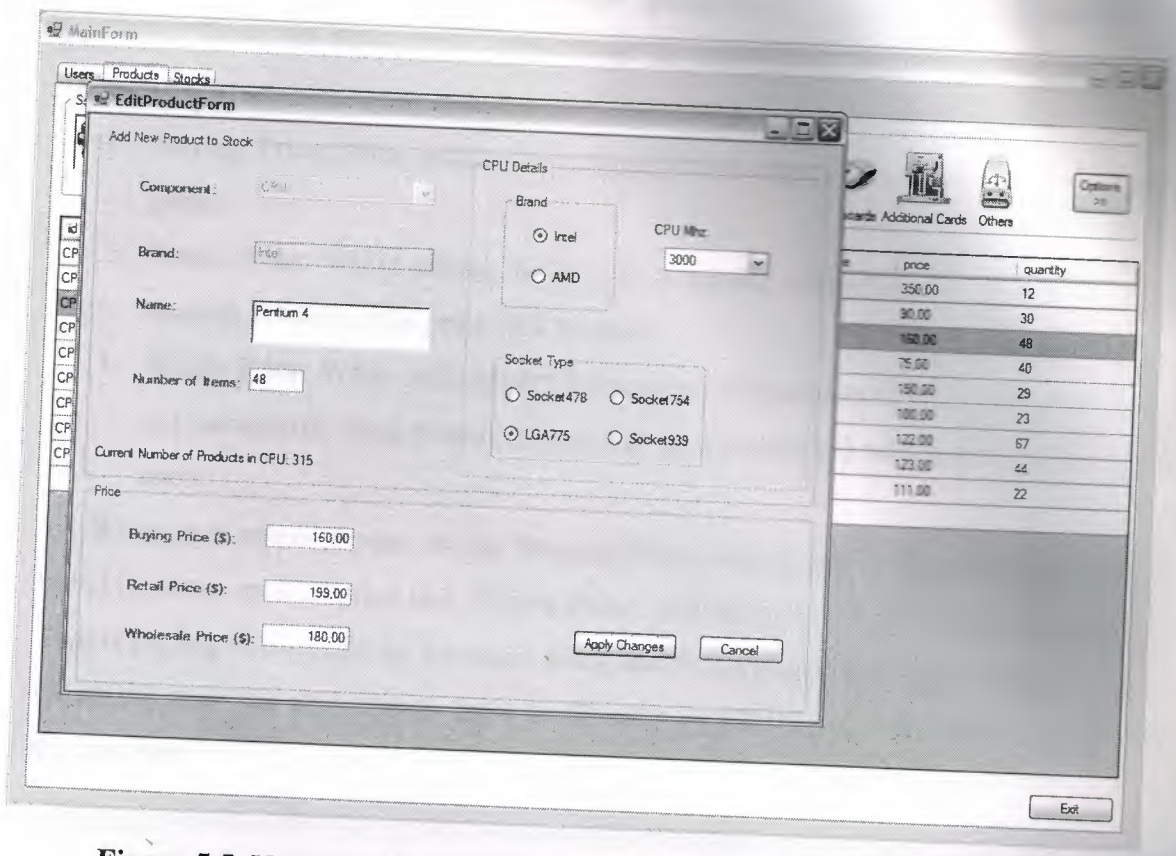

Figure 5.5 Updating the Information of an item using the 'EditProductForm'

The task of the second option in the popup menu, Delete Item, is also self explanatory. When user clicks this button, a message box prompting the user if he/she is sure that he/she wants to delete the selected product appears. Clicking 'Yes' button will remove the product from the table from which the item belongs to. But in order to avoid data loss, item will not be deleted entirely from the database, it will be recorded to the 'deleted' table which contains all the deleted items from all component tables.

5.2.3 'Stocks' Window

When user clicks the 'Stocks' tab, a window that lets the user to enter new items to the stock and to see the basic statistics about economic situation of the company appears.

Upper half of the window is used for adding new products to the stock. User selects the category in which the item belongs to from the Dropdown list labeled 'Component'. Each category of products has different properties, so when user selects a category from

the list, the corresponding options appears in a group box with the name 'Component Details'. Using these options, user can enter the specifications of the product. On the right hand side of the 'Component Details' group box user can enter the price for the products. There are three price values must be entered when adding a product to the database.

1- Buying Price: Price which the company buys one piece of that item. (Incoming price)
2- Retail Price: While selling an item to a regular customer who doesn't buy big amount of items, this price will be used.
3- Whole Price: While selling very big number of products to a customer (this does not necessarily be a person, it can also be a company) whole sale price will be used.

When user enters a price to the 'Buying Price' automatically 'Retail Price' will set to %110 of the entered price and 'Whole Price' will set to %105 of the entered price. That is putting %10 profit for the retail sales, and %5 profit for the whole sales.
Of course user can also change these profit rates by clicking the up-down buttons near the text box.

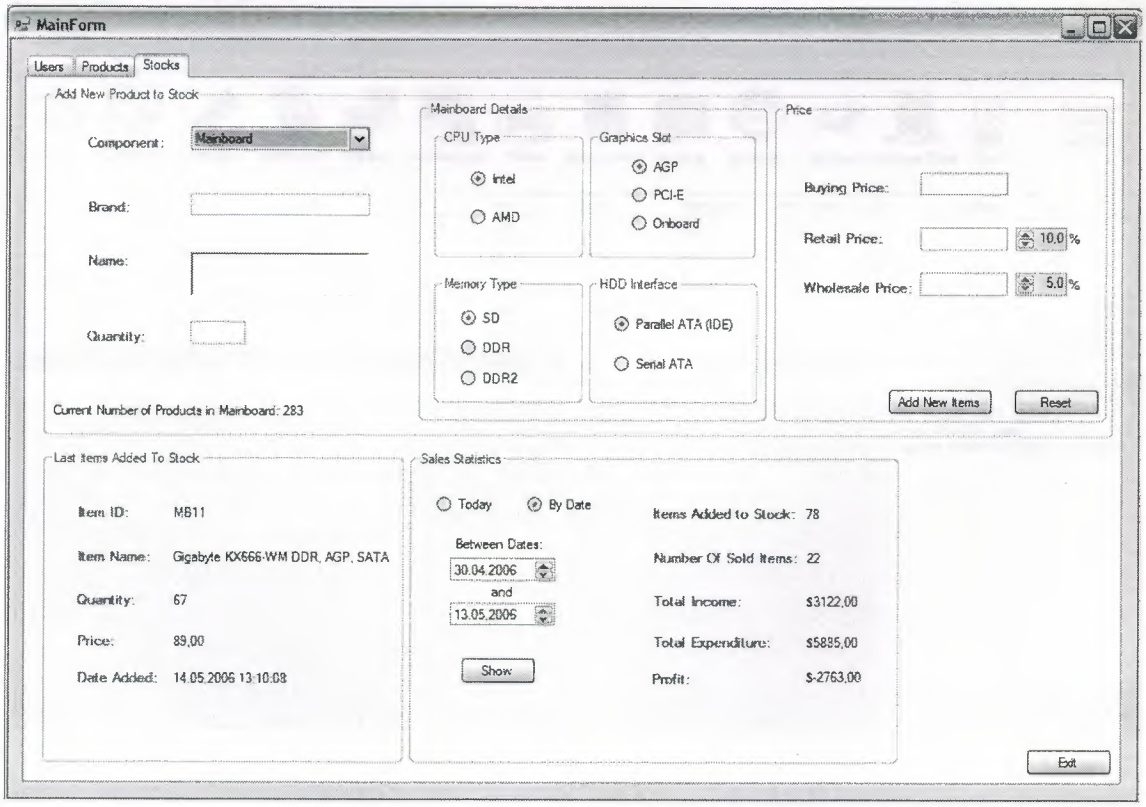

Figure 5.6 'Stocks' window

Lower half of the window gives some information about the stocks and sales to the user. Lower left side shows the last added item to the stock and when it was added. 'Sales Statistics' part shows the sales made current day or sales made in a period of time. It lists number of items added to stock, number of items sold, total income earned from the sales, total money spent for buying products, and profit. These statistics give user a brief idea about the current situation of the sales'.

5.3 Sales Department's Panel

5.3.1 'Products' Window

When an employee from the sales department logs in to the system, window in the following figure appears. It is very similar to the 'Products' window that the managers are using. The difference show itself when the user selects an item from the list and right-clicks it. There is only one option in the menu, 'Edit Item' and 'Delete Item' options are not in the menu because it is not the task of a salesperson, and it is only done by a manager. Instead there is 'Add to Cart' option which adds selected item to the current sales transaction.

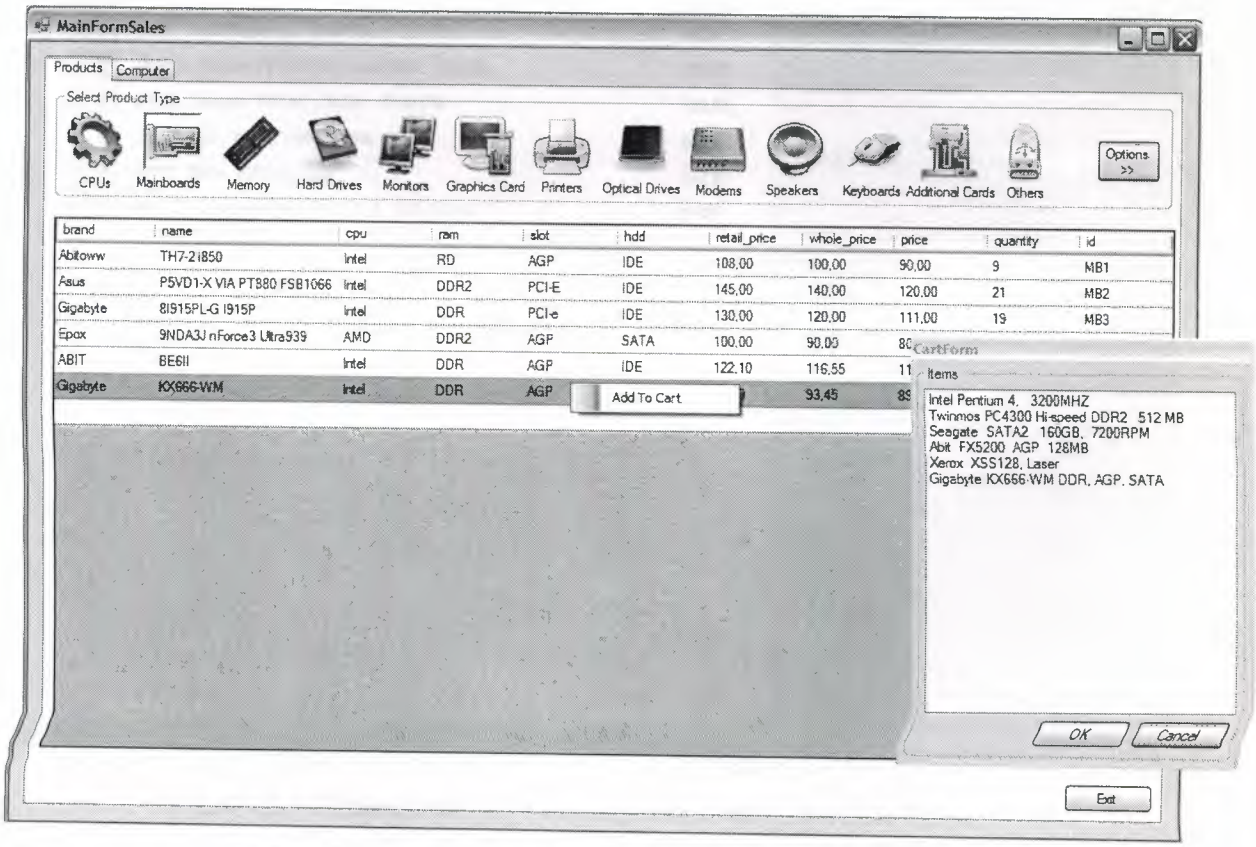

Figure 5.7 'Products' window of the sales department

5.3.2 'RecordSales' Window

After adding the products which the customer needs, salesman clicks 'OK' button in order to open sales transaction window, or 'Cancel' button to cancel the transaction.

When user clicks 'OK' button 'RecordSaleForm' will be opened.

In this window user can see the list of the items selected along with their prices and the quantities. If the customer needs to buy more than one of the same item it can be set in this window. As user changes the quantity of an item, price of that item and total price of the items also re-calculated accordingly.

Along with showing selected items and their prices, also the customer information is requested in this window. This information will be recorded to the 'Customers' database table and will be used in the sales receipt.

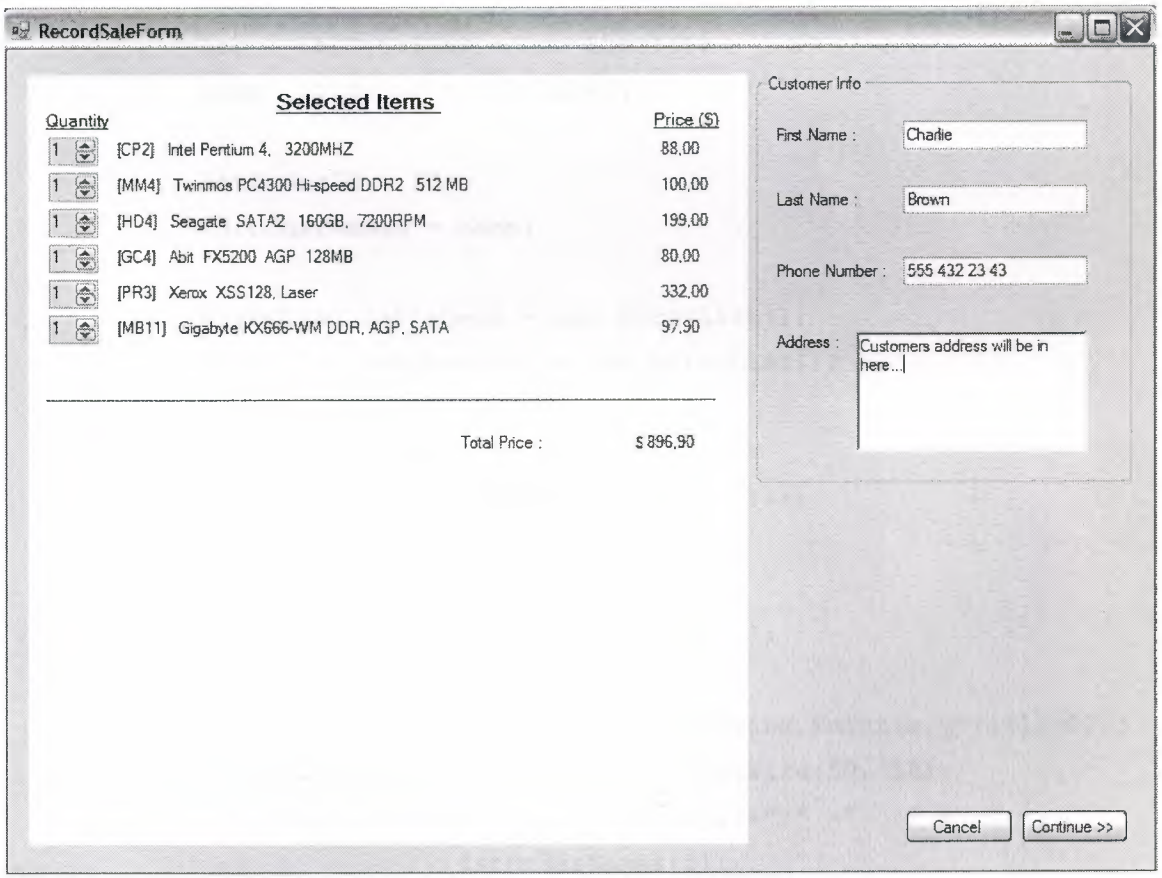

Figure 5.8 Starting the sales transaction. List of selected items.

This form displays the selected products using the data transferred from the 'CartForm' which holds the selected items from the 'MainForm'. ArrayLists are used for transferring data between forms. Because number of items selected can not be predicted, a fixed size array would not be efficient to use. ArrayList Collection Class of the .NET Framework is a dynamic array which has many other advanced features.
Two ArrayLists carry data from 'CartForm' to the 'RecordSaleForm', namely `lstOrderNames` for the names of the products and `lstOrderID` for the id codes of the products.

Here are the codes from the 'RecordSaleForm' to fill the ArrayLists with the ArrayList objects from the 'CartForm'.

```
public void setItems(ArrayList ID, ArrayList name)
{
    Label lblTotal = new Label();
    Label lblTotalBar = new Label();
    Label lbldolar = new Label();

    lstOrderID = ID;
    lstOrderNames = name;

    ArrayList lstLabels = new ArrayList();
    ArrayList cmbQuantity = new ArrayList();

    int i,x=60,y=25;
    for (i = 0; i < lstOrderID.Count; i++)
    {
        lstLabels.Add(new Label());
        Label lbl = (Label)lstLabels[i];
        lbl.AutoSize = true;
        lbl.Location = new System.Drawing.Point(x,y*(i+1)+20);
        lbl.Size = new System.Drawing.Size(50, 50);
        lbl.Text = "[" + lstOrderID[i]+"]" +"    " +
                   lstOrderNames[i];

        lstCmbQuantity.Add(new NumericUpDown());
        NumericUpDown cmb = (NumericUpDown)lstCmbQuantity[i];
        cmb.Maximum = Convert.ToInt32(
```

```
    idandpriceTableAdapter.getQuantity(
        lstOrderID[i].ToString()));
cmb.Minimum = 1;
cmb.Value = 1;
cmb.Name = "nn" + i.ToString();
cmb.ReadOnly = true;
cmb.Size = new System.Drawing.Size(35, 30);
cmb.Location = new System.Drawing.Point(
                            x - 45, y * (i + 1) + 18);
cmb.ValueChanged +=new EventHandler(cmb_ValueChanged);


lstPrices.Add(new Label());
Label lblPrice = (Label)lstPrices[i];
lblPrice.AutoSize = true;
lblPrice.Name = "lblPrice" + i.ToString();
lblPrice.Location = new System.Drawing.Point(
                                    440, y * (i + 1) + 20);
lblPrice.Size = new System.Drawing.Size(50, 50);
lblPrice.Text = idandpriceTableAdapter.getRetailPrice(
   lstOrderID[i].ToString()).ToString();
if (i == lstOrderID.Count - 1)
{
    lblTotalBar.Text = "----------------------------";
    lblTotalBar.Size = new Size(500, 30);
    lblTotalBar.Location = new Point(
                            10, y * (i + 3) + 20);


    lblTotal.Text = "Total Price : ";
    lblTotal.Location = new Point(
                            300, y * (i + 4) + 25);
    lblTotal.Size = new Size(100, 50);


    lbldolar.Text = "$";
    lbldolar.Location = new Point(
                            422, y * (i + 4) + 25);
    lbldolar.Size = new Size(50, 30);

    lblTotalPrice.Text =
```

```
                    Convert.ToString(calculateTotal());
                lblTotalPrice.Location = new Point(
                                        430, y * (i + 4) + 25);
                lblTotalPrice.Size = new Size(50, 30);
            }

            pnlList.Controls.Add(lbl);
            pnlList.Controls.Add(cmb);
            pnlList.Controls.Add(lblPrice);
            pnlList.Controls.Add(lblTotal);
            pnlList.Controls.Add(lblTotalBar);
            pnlList.Controls.Add(lblTotalPrice);
            pnlList.Controls.Add(lbldolar);

        }
    }
```

After setting the quantities for the items and filling the user information fields, user clicks 'Continue' button to progress through the last step of the sales transaction.

5.3.3 'RecordSale' Window – Completing the Transaction

When user clicks 'Continue' button, the window in the figure below opens. This is the last window of the sales transaction. Main purpose of this window is to display an overview of the sales process, cashier name, customer name. Most important component of this window is the area where the print preview of the sales receipt is shown.

After selecting how the payment will be made, user clicks the 'Finish' button. Then system processes the data and shows a message whether the sales transaction is successful or not. If it is successful, system prints out the receipt, reduces the number of items from the components database, adds customer information to the 'customers' database table and returns to the main form.

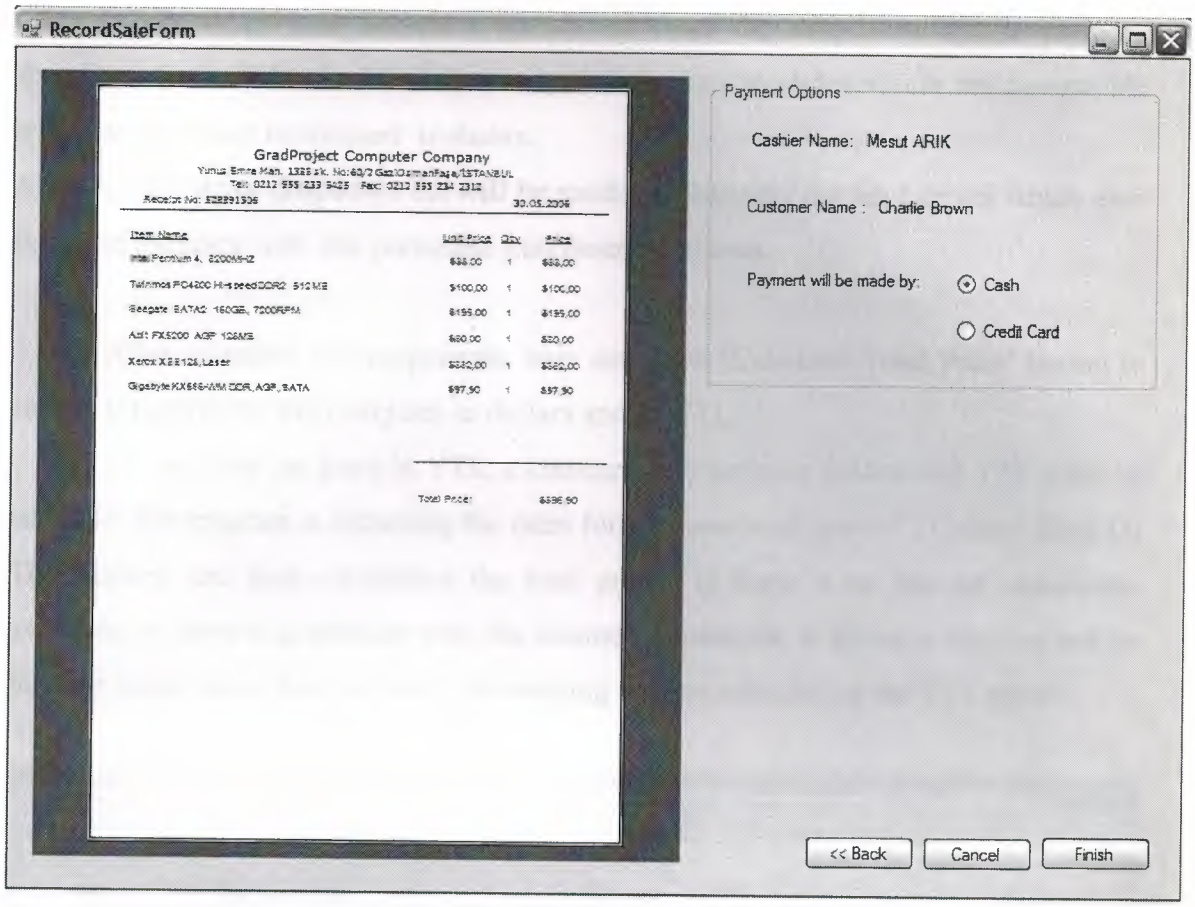

Figure 5.9 'RecordSale' Window Previewing the Sales Receipt.

5.3.4 'Computer' Window

When user clicks 'Computer' tab on the main form, a new window which will help salesman to build a new computer for the customer appears. System assumes that a new computer must consists of 10 components, which are Mainboard, Cpu, Memory, Hard Drive, Graphics Card, Monitor, Optical Drive, Modem, Speaker and Keyboard&Mouse.

When user selects an item from a dropdown list, price of that item will be displayed near the name of the product.

It helps the user to select compatible components for building a computer. For example an Intel processor can only be installed on a Intel compatible mainboard, or a mainboard having an AGP slot can not operate a PCI-E graphics card.

So, when user selects a processor from the dropdown list, according to its brand and conection interface, mainboards shown in the mainboards dropdown list is changed to show compatible models only.

Again when user selects a mainboard from the dropdown list,the memory dropdown list will be modified that only the memory modules which are compatible with that particular mainboard is shown.
Also the hard drive dropdown list will be modified that only the hard drives which uses the same interface with that particular mainboard is shown.

After selecting all components, user can click 'Calculate Total Price' button to see the total price of the computer in dollars and in YTL.

To calculate the price in YTL, a currency rate between dollars and YTL must be used. So, the program is retrieving the rates form "www.tcmb.gov.tr" , Central Bank Of The Turkey, and then calculating the total price. If there is no internet connection available, or there is a problem with the internet connection, it shows a warning and let the user know about that and continue working without calculating the YTL price.

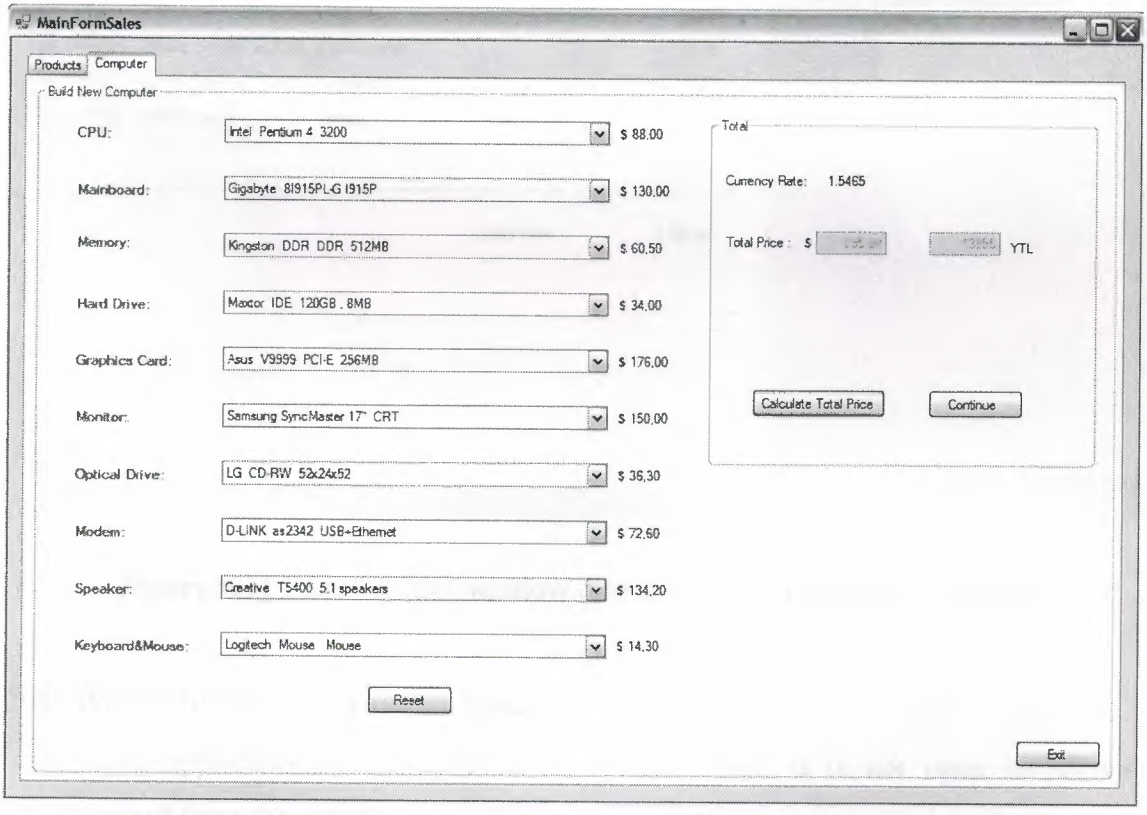

Figure 5.10 'Computer' window helps the user to select components for a new computer.

When user finishes selecting the parts and satisfies the customer with the price, he/she clicks the continue button in order to start sales transaction.

Clicking this button display the 'RecordSaleForm' which was shown in the first page of the main form before. User can see which components the new computer is going to have and set the quantities of the items if need be.

Again entering the customer information and clicking 'Continue' button will lead the user to the last page which shows the payment options and sales receipt.

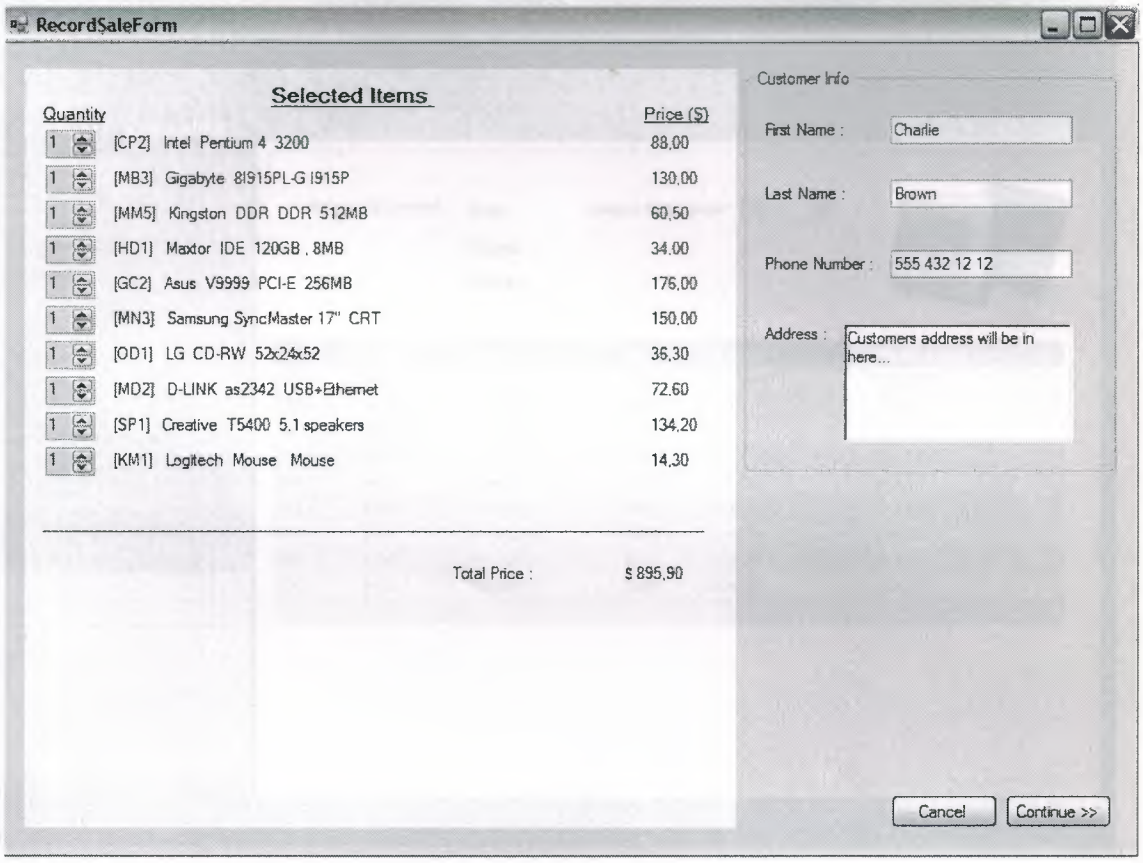

Figure 5.11 'RecordSale' window displaying list of the selected items

5.4 Web Site For The Project Database

The application is designed for in-company use, it is not open to anyone unauthorized from the outside world. But since marketing is commercial work, its target is to sell goods to the people on the outside. For this particular situation, customers come to the shop and ask for the products they need whether it is in the stocks or not. And if it is, depending on the price customer decides to buy or not to buy the product.

This method may be effective but it is far from practical. Customers should not have to come to the company just for asking whether or not the product they want is in stock. Setting some portions of the database accessible by public is the solution to that problem.

Developing a web site and putting the products to the public on this site is the best way to reach the customers.

Figure 5.12 Web site lets the customers to see the products on the stocks from wherever they want.

CONCLUSION

Object Oriented Programming approach is the most convenient method of developing scalable and maintainable software. Designing software for large systems involves modeling of the classes, use cases and actors, which are the essential elements of the Unified Modeling Language (UML).

This project is developed using the latest software development technologies for the Microsoft Windows platform. It is integrating the various components of MS Windows environments; Managing a relational database that is running on the MS SQL Server using the ADO.NET data objects from C# codes and ASP.NET.

By using the object oriented programming techniques, it has highly maintainable source codes therefore it would be easy to add new features to the program. For the future improvements, extended reporting services could be added as the accounting panel for the managerial staff.

It was a great opportunity for me to apply OOP techniques that I have learned throughout my undergraduate studies in the University.

REFERENCES

[1] Wikipedia, Object Oriented Programming Website:
"*http://en.wikipedia.org/wiki/Object-oriented_programming*"

[2] Jesse Liberty, *Programming C# , Second Edition,* O'Reilly & Associates Inc. Sebastopol, CA, February 2002

[3] NeXT Software, Inc. Object Oriented Programming Concepts, Website:
"*http://www.toodarkpark.org/computers/objc/oop.html*"

[4] Wikipedia , .NET Framework, Website :
" *http://en.wikipedia.org/wiki/.NET_Framework*"

[5] Object Oriented Analysis and Design Team, Kennesaw State University, Spring 2001,
"*http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm*"

[6]G. Andrew Duthie, Matthew MacDonald, *ASP.NET in a Nutshell,* O'Reilly & Associates Inc. Sebastopol, CA, June 2002