NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CLIENT SERVER MODELLING OF DISTRIBUTIVE SYSTEMS

GRADUATION PROJECT
COM-400

Students: GURCAN BIROL KARAGOZ
CETIN TAHIR IZANCI

Supervisor: Assoc. Prof. Dr. RAHIB ABIYEV

NICOSIA-2002

ACKNOWLEDGEMENT

"First, We would like to thank our supervisor Asst. Prof. Dr. Rahib Abiyev for his advice and belief in our work.

Second, We would like to express my Graduate project to Near East University for the scholarship that made the work possible.

Third, We want to thank all our friends for their advice and support during project.

And finally, We want to thank our families for their power and support during the preparation of this project.

Thank you all!!

ABSTRACT

Increasing the complexity of the Internet and the LAN(Local Area Network) proceses, the people need more reliable, safe and fast systems, The effective way of achieving this problem is to use Client Server modeling systems.

The graduation project is developed by the information of Client Server modelling of Distributed Database systems. For this reason the applications for creating Distributed Database systems are described.

The aim of this project is the development of Client/Server technologies and by the aim of this idea, this project is developed. The structure of distributed systems are described and the design of distributed systems are given.

The main problem of client server system is the speed of transaction in operating systems and by the time the amount of data will be bigger and bigger,so that the speed will decrease and the problems will occur. To overcome this problem, the database server must be fast, reliable and safety.

The obtained results show us the importance of creating client server software to save time in life and this project is developed by the aim of this idea.

INTRODUCTION

Distributed database technology is one of the most important developments of the past decades.The maturation of data base management systems - DBMS - technology has coincided with significant developments in distributed computing and parallel processing technologies and the result is the emergence of distributed DBMSs and parallel DBMSs. These systems have started to become the dominant data management tools for highly intensive applications. The basic motivations for distributing databases are improved performance, increased availability,shareability, expandibility, and access flexibility. Although, there have been many researchstudies in these areas, some commercial systems can provide the whole functionality for distributed transaction processing. Important issues concerned in studies are database placement in the distributed environment, distributed query processing, distributed concurrency control algorithms, reliability and availability protocols and replication strategies.

As the subject is broad in a sense that the recent studies are all theoretical and most of the work being done in the past, only an overview of distributed database management could be made and presented in this report. Starting from the definition of a distributed databases, advantages,disadvantages, the main concern of promises and related transparency constraints are given. The main issues dealt in a distribution of data and applications are briefly discussed. The current trends and developments according to the researchers are summarized in the conclusion along with the self ideas.

The objective of this project is to investigate the development of Client Server modelling of Distributive Database systems.The project consists of introduction,six chapters and conclusion.

Chapter one describes Distributed Data processing,advantages and disadvantages of distributed systems.

Chapter two presents Distributed systems and software usage in Distributed sytems.The operations and the principles while developing these software.

Chapter three presents a brief explanation of MySql and connectivity,localization of MySql server.

Chapter four presents using MySql.Connecting and disconnecting from server,Creating database,tables and making queries.Mysql language and how to declare strings,integers,long integers and null values.

Chapter five presents how to connet the server and the usage of the software by the help of figures.

Conclusion presents the obtained the important results and contribution in the project.

TABLE OF CONTENTS

CHAPTER 1: DISTRIBUTED DATA PROCESSING

The term distributed processing (or distributed computing) has been used to refer to such diverse systems as multiprocessor systems, distributed data processing, and computer networks. Here are some of the other terms that have been used synonymously with distributed processing: distributed function, distributed computers or computing, networks, multiprocessors / multi computers, satellite processing/satellite computers, backend processing, dedicated/special-purpose computers, time-shared systems, and functionally modular systems.

Some degree of distributed processing goes on in any computer system, even on single-processor computers. Starting with the second-generation computers, the central processing unit (CPU) and input/output (I/O) functions have been separated and overlapped. This separation and overlap can be considered as one form of distributed processing. However, it should be quite clear that what we would like to refer to as distributed processing, or distributed computing, has nothing to do with this form of distribution of functions in a single-processor computer system.

Distributed computing system states is a number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks. The "processing element" referred to in this definition is a computing device that can execute a program on its own.

One fundamental question that needs to be asked is: What is being distributed? One of the things that might be distributed is the processing logic. In fact, the definition of a distributed computing system given above implicitly assumes that the processing logic or processing elements are distributed. Another possible distribution is according to function. Various functions of a computer system could be delegated to various pieces of hardware or software. A third possible mode of distribution is according to data. Data used by a number of applications may be distributed to a number of processing sites. Finally, control can be distributed. The control of the execution of various tasks might be distributed instead of being performed by one computer system. From the viewpoint of distributed database systems, these modes of distribution are all necessary and important. In the following sections we talk about these in more detail.

Distributed computing systems can be classified with respect to a number of criteria. Bochmann lists some of these criteria as follows: degree of coupling, interconnection structure, interdependence of components, and synchronization between components [Bochmann, 1983]. Degree of coupling refers to a measure that determines how closely the processing elements are connected together. This can be measured as the ratio of the amount of data exchanged to the amount of local processing performed in executing a task. If the communication is done over a computer network, there exists weak coupling among the processing elements. However, if components are shared, we talk about strong coupling. Shared components can be either primary memory or secondary storage devices. As for the interconnection structure, one can talk about those cases that have a point-to-point interconnection between processing elements, as opposed to those, which use a common interconnection channel. We discuss various interconnection structures. The processing elements might depend on each other quite strongly in the execution of a task, or this interdependence might be as minimal as passing messages at the beginning of execution and reporting results at the end.

Synchronization between processing elements might be maintained by synchronous or by asynchronous means. Note that some of these criteria are not entirely independent. For example, if the synchronization between processing elements is synchronous, one would expect the processing elements to be strongly interdependent, and possibly to work in a strongly coupled fashion.

The distributed processing better corresponds to the organizational structure of today's widely distributed enterprises, and that such a system is more reliable and more responsive. Data can be entered and stored where it is generated, without any need for physical (manual) movement. Furthermore, building a distributed system might make economic sense since the costs of memory and processing elements are decreasing continuously

The fundamental reason behind distributed processing is to be better able to solve the big and complicated problems, by using a variation of the well-known divide and-conquer rule. If the necessary software support for distributed processing can be developed, it might be possible to solve these complicated problems simply by dividing them into smaller pieces and assigning them to different software groups, which work on different computers and produce a system that runs on multiple processing elements but can work efficiently toward the execution of a common task.

This approach has two fundamental advantages from an economics standpoint. First, we are fast approaching the limits of computation speed for a single processing element. The only available route to more computing power, therefore, is to employ multiple processing elements optimally. This requires research in distributed processing as denned earlier, as well as in parallel processing, which is outside the scope. The second economic reason is that by attacking these problems in smaller groups working more or less autonomously, it might be possible to discipline the cost of software development. Indeed, it is well known that the cost of software has been increasing in opposition to the cost trends of hardware.

Distributed database systems should also be viewed within this framework and treated as tools that could make distributed processing easier and more efficient. It is reasonable to draw an analogy between what distributed databases might offer to the data processing world and what the database technology has already provided. There is no doubt that the development of general-purpose, adaptable, efficient distributed database systems will aid greatly in the task of developing distributed software.

1.1 Distributed database system

We can define a distributed database as a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the DDBS and makes the distribution transparent to the users. The two important terms in these definitions are "logically interrelated" and "distributed over a computer network." They help eliminate certain cases that have sometimes been accepted to represent a DDBS.

First, a DDBS is not a "collection of files" that can be individually stored at each node of a computer network. To form a DDBS, files should not only be logically related, but there should be structure among the files, and access should be via a common interface. It has sometimes been assumed that the physical distribution of data is not the most significant issue. The proponents of this view would therefore feel comfortable in labeling as a distributed database two (related) databases that reside in the same computer system. However, the physical distribution of data is very important. It creates problems that are not encountered when the databases reside in the same computer. Note that physical distribution does not necessarily imply that the computer systems be geographically far apart; they could actually be in the same room. It simply implies that the communication between them is done over a network instead of through shared memory, with the network as the only shared resource.

The definition above also rules out multiprocessor systems as DDBSs. A multiprocessor system is generally considered to be a system where two or more processors share some form of memory, either primary memory, in which case the multiprocessor is called tightly coupled, or secondary memory, when it is called loosely coupled. Sharing memory enables the processors to communicate without exchanging messages. With the improvements in microprocessor and VLSI technologies, other forms of multiprocessors have emerged with a number of microprocessors connected by a switch .

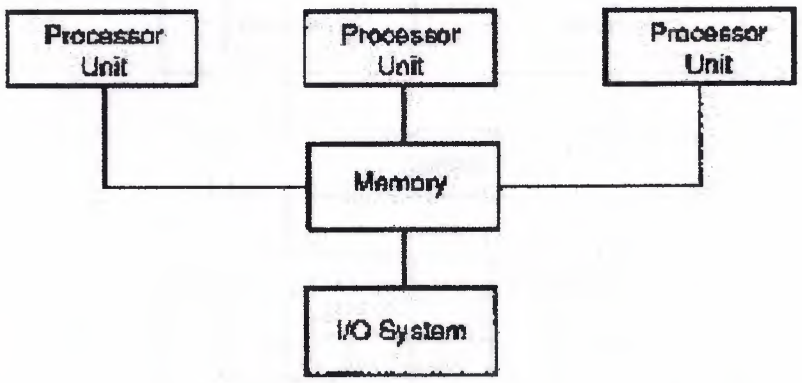


Figure 1.1 Tightly-Coupled Multiprocessor

Another distinction that is commonly made in this context is between shared everything and shared-nothing architectures. The former architectural model permits

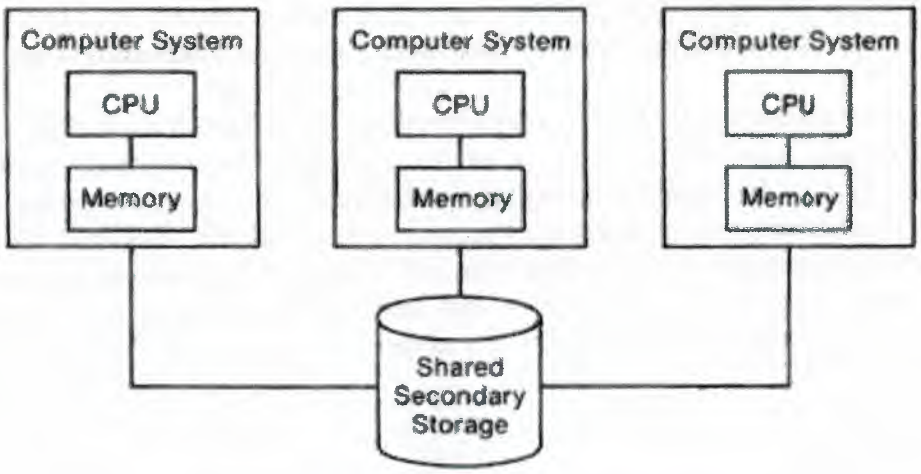


Figure 1.2 Loosely-Coupled Multiprocessor

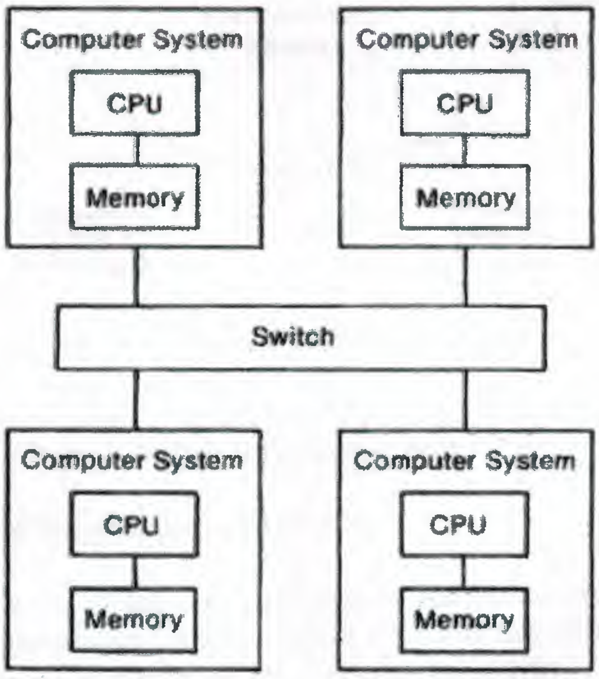


Figure 1.3 Switch-Based Multiprocessor System

each processor to access everything (primary and secondary memories, and peripherals) in the system and covers the three models that we described above. The shared nothing architecture is one where each processor has its own primary and secondary memories as well as peripherals, and communicates with other processors over a very high speed bus. In this sense the shared-nothing multiprocessors are quite similar to the distributed environment that we consider in this book. However, there are differences between the interactions in multiprocessor architectures and the rather loose interaction that is common in distributed computing environments. The fundamental difference is the mode of operation. A multiprocessor system design is rather symmetrical consisting of a number of identical processor and memory components, controlled by one or more copies of the same operating system, which is responsible for a strict control of the task assignment to each processor. This is not true in distributed computing systems, where heterogeneity of the operating system as well as the hardware is quite common.

In addition, a DDBS is not a system where, despite the existence of a network, the database resides at only one node of the network. In this case, the problems of database management are no different from the problems encountered in a centralized database environment. The database is centrally managed by one computer system and all the requests are routed to that site. The only additional consideration has to do with transmission delays. It is obvious that the existence of a computer network or a collection of "files" is not sufficient to form a distributed database system.

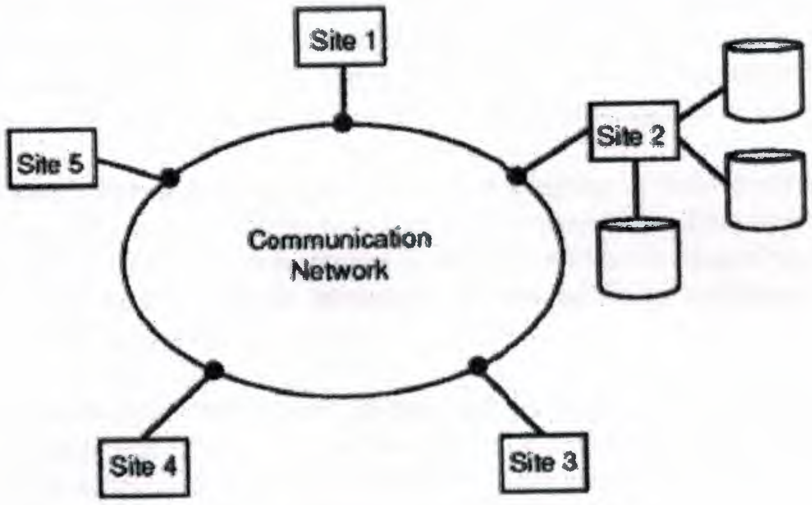


Figure 1.4 Central Database on a Network

At this point it might be helpful to look at an example of distributed database application that we can also use to clarify our subsequent discussions.

1.2 Advantages and Disadvantages of DDBSs

The distribution of data and applications has promising potential advantages. Note that these are potential advantages which the individual DDBSs aim to achieve. As such, they may also be considered as the objectives of DDBSs.

1.2.1 Advantages:

Local Autonomy. Since data is distributed, a group of users that commonly share such data can have it placed at the site where they work, and thus have local control. This permits setting and enforcing local policies regarding the use of the data. There are studies [D'Oliviera, 1977] indicating that the ability to partition the author ity and responsibility of information management is the major reason many business organizations consider distributed information systems. This is probably the most important sociological development that we have witnessed in recent years with respect to the use of computers.

Of course, the local autonomy issue is more important in those organizations that are inherently decentralized. For such organizations, implementing the information system in a decentralized manner might also be more suitable. On the other hand, for those organizations with quite a centralized structure and management style, decentralization might not be an overwhelming social or managerial issue.

In distributed system, the validity of local autonomy is obvious. It would be quite absurd to have an environment where all the record keeping is done locally, as it would be if information were shared among different sites in a manual fashion (either by exchanging hard copies of reports, or by exchanging magnetic tapes, disks, floppies, etc.).

Improved Performance: Again, because the regularly used data is proximate to the users, and given the parallelism inherent in distributed systems, it may be possible to improve the performance of database accesses. On the one hand, since each site handles only a portion of the database, contention for CPU and I/O services is not as severe as for centralized databases. On the other hand, data retrieved by a transaction may be stored at a number of sites, making it possible to execute the transaction in parallel.

Let us assume that in our example the record keeping is done centrally at the world headquarters, with remote access provided to the other sites. This would require the transmission to New York of each request generated in Phoenix inquiring about the inventory level of an item. It would probably be impossible to withstand the low performance of such an operation.

Improved Reliability/Availability: If data is replicated so that it exists at more than one site, a crash of one of the sites, or the failure of a communication link making some of these sites inaccessible, does not necessarily make the data impossible to reach. Furthermore, system crashes or link failures do not cause total system inoperability. Even though some of the data may be inaccessible, the DDBS can still provide limited service.

Obviously, if the inventory information at both warehouses is replicated at both sites, the failure at one of the sites would not make the information inaccessible to the rest of the organization. If proper facilities are set up, it might even be possible to give users at the failed site access to the remote information.

Economics: It is possible to view this from two perspectives. The first is in terms of communication costs. If databases are geographically dispersed and the applications running against them exhibit strong interaction of dispersed data, it may be much more economical to partition the application and do the processing locally at each site. Here the trade-off is between telecommunication costs and data communication costs. The second viewpoint is that it normally costs much less to put together a system of smaller computers with the equivalent power of a single big machine. In the 1960s and early 1970s, it was commonly believed that it would be possible to purchase a fourfold powerful computer if one spent twice as much. This was known as Grosh's law. With the advent of minicomputers, and especially microcomputers, this law is considered invalid.

The case about lower communication costs can easily be demonstrated in the example we have been considering. It is no doubt much cheaper in the long run to maintain a computer system at a site and keep data locally stored instead of having to incur heavy telecommunication costs for each request. The level of use when this becomes true can obviously change depending on the traffic patterns among sites, but it is quite reasonable to expect this to occur.

Expandability: In a distributed environment, it is much easier to accommodate increasing database sizes. Major system overhauls are seldom necessary; expansion can usually be handled by adding processing and storage power to the network. Obviously, it may not be possible to obtain a linear increase in "power," since this also depends on the overhead of distribution. However, significant improvements are still possible.

Share ability: Organizations that have geographically distributed operations normally store data in a distributed fashion as well. However, if the information system is not distributed, it is usually impossible to share these data and resources. A distributed database system therefore makes this sharing feasible.

1.2.2 Disadvantages

However, these advantages are offset by several problems arising from the distribution of the database.

Lack of Experience: General-purpose distributed database systems are not yet commonly used. What we have are either prototype systems or systems that are tailored to one application (e.g., airline reservations). This has serious consequences because the solutions that have been proposed for various problems have not been tested in actual operating environments.

Complexity: DDBS problems are inherently more complex than centralized database management ones, as they include not only the problems found in a centralized environment, but also a new set of unresolved problems. We discuss these new issues shortly.

Cost: Distributed systems require additional hardware (communication mechanisms, etc.), thus have increased hardware costs. However, the trend toward decreasing hardware costs does not make this a significant factor. A more important fraction of the cost lies in the fact that additional and more complex software and communication may be necessary to solve some of the technical problems. The development of software engineering techniques (distributed debuggers and the like) should help in this respect.

Distribution of Control: This point was stated previously as an advantage of DDBSs. Unfortunately, distribution creates problems of synchronization and coordination (the reasons for this added complexity are studied in the next section). Distributed control can therefore easily become a liability if care is not taken to adopt adequate policies to deal with these issues.

Security: One of the major benefits of centralized databases has been the control it provides over the access to data. Security can easily be controlled in one central location, with the DBMS enforcing the rules. However, in a distributed database system, a network is involved which is a medium that has its own security requirements. It is well known that there are serious problems in maintaining adequate security over computer networks. Thus the security problems in distributed database systems are by nature more complicated than in centralized ones.

Difficulty of Change: Most businesses have already invested heavily in their database systems, which are not distributed. Currently, no tools or methodologies exist to help these users convert their centralized databases into a DDBS. Research in heterogeneous databases and database integration is expected to overcome these difficulties.

CHAPTER 2: DISTRIBUTED SYSTEMS AND DISTRIBUTED SOFTWARE

2.1 Characteristic of distributed systems

Distributed computer environments are based on distributed computer systems which consist of a set of processing components connected by a communication network. The software systems running on the various processing components exchange data through the communication network. This type of system is also called loosely coupled distributed system.

Processing nodes can be composed of several processors which share memory. This shared memory is used to exchange information by the software executed on such a node. This type of system is called a tightly coupled distributed system. Some advantages of distributed systems are below shown:

• Increased Performance
Performance is generally defined in terms of average response time and through put. If processing capability can be located where it is required the response time can be highly reduced. Data can be processed locally before it is sent to other nodes for further processing. This increases throughput.

• Increased reliability
Normally nodes in a distributed system can take over the tasks of other nodes which are currently out of order. This means that a distributed system continues its work with reduced performance but with little or no reduction of functionality

• Increased flexibility
Additional functionality can be added to a distributed system or the number of users can be permanently increased. A distributed system allows this system growth by simply adding more processing nodes.

2.2 Parallel or Concurrent Programs

Parallel or concurrent programs are characterized by a set of statements interrelated by multiple control threads. Each sequence of statements executed by one or more control threads is called a process object (The term 'process' shall be used instead of 'process object' when it is clear from the context that we mean a process object).

The relationship between processes or threads and process objects is shown in the following figure.

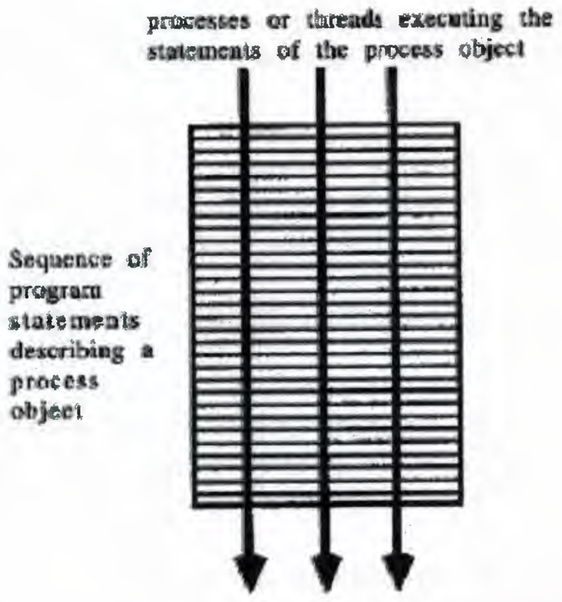

processes or threads executing the statements of the process object
Sequence of program statements describing a process object

Figure 2.1: Process/Threads and Process Objects

The statements (operations) of the individual processes are executed overlapped or interleaved or both. If a single processor is multiplexed among several concurrent processes, the machine instructions of these processes can only be interleaved in time. For a certain time slice, the processor is assigned to a process in order to execute the statements of a process object. Assigning a processor to another process is called context switching. This type of concurrency is also called multitasking. The following figure shows an example of how a processor is shared between several processes.

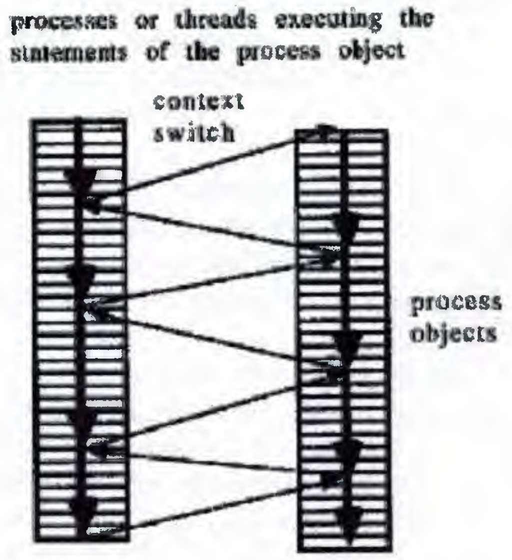

processes or threads executing the statements of the process object
context switch
process objects

Figure 2.2: Multitasking

Machine instructions of processes running on different processors can be overlapped at each node at which a processor is available. These are distributed programs.

Concurrent or parallel programs are either interleaved, distributed, or both. For a programmer it is not necessary to know whether multitasking or a distributed system is used to run his program.

Normally the processes of a concurrent program share the resources such as processor, memory, disk, and databases, and if they cooperate in order to reach a common goal they exchange information and synchronize their activities.

Their are two reasons to structure a program in parallel executable process objects:

1. Fine grain parallelism is mainly used to accelerate large numerical computations. This type of parallism is often achieved by using vector processors and the pipelining of operations. It is mainly implemented by hardware.

2. Structural parallelism is used if the structure of the task to be performed is fundamentally parallel. The process objects are a very important concept for structuring programs in certain application areas, e.g. operating systems, real time systems, and communication systems. Especially in real time systems which must react to external events, processes (objects) are used to achieve separation of the tasks /FAPA88/. Each process handles a related set of events and cooperates with other processes to achieve a common purpose. In order to cooperate, processes exchange information either via shared data or via messages.

2.3 Networked Computing

2.3.1. Network Structure and the Remote Procedure Call Concept

Network computing is characterized by several sequences of jobs, which arrive independently at various nodes. The jobs are designed and implemented more or less independently of each other and are only loosely coupled. The distributed system serves primarily as a resource-sharing network.

A very common example of resource sharing is the file server. All files are located on a dedicated node in a distributed system. Software components running on other nodes send their file access requests to the file server software. The file server executes these requests and returns the results (to the clients).

In addition to file servers many other kinds of servers such as print servers, compute servers, data base servers, and mail servers have been implemented As with the file server, clients send their requests to the appropriate server and receive the results for further processing. Servers process the requests from the various clients more or less independently of each other. The programs running on the clients can be viewed as being designed and developed independently of each other.

The following figure shows the concept of client/server systems.

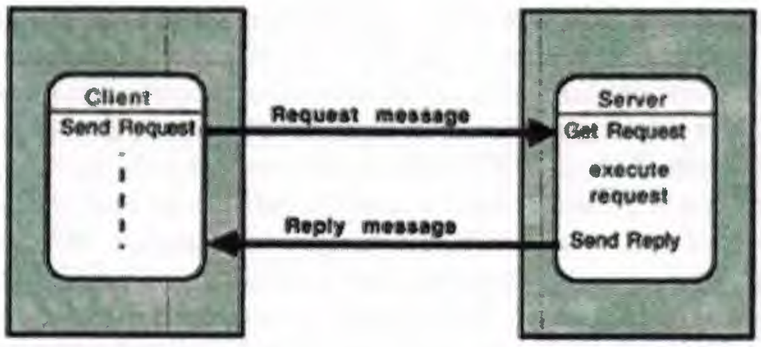


Figure 2.3: The Concept of Client/Server System

In client server systems, the clients represent the users of a distributed system and servers represent different operating system functions or a commonly used application.

The following figure shows a simple example of a client server system.

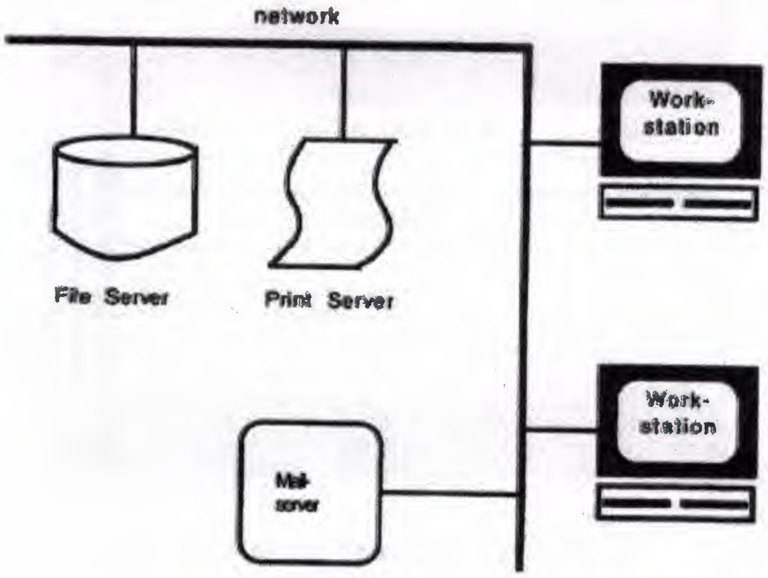


Figure 2.4: A Small Client/Server System

This system has a print server, a file server, and the clients (users) which run on workstations (WS) and personal computers (PC). The server software and the client software can run on the same type of computer. The different nodes are connected by a local area network.

From a user's point of view a client/server system can hardly be distinguished from a central system, e.g. a user cannot see whether a file is located on his local system or on a remote file server node. For the user the client/server system appears to be a very convenient and flexible central computing system. Mostly the user does not know whether a file is stored on his PC or on a file server. To the user, the storage capacity of the server appears to be a part of the PC storage capacity. Client/server systems are also very flexible. For a new application a specialized new server can be added e.g. data base systems run on specialized data base servers, which have short access times. Database applications are primarily controlled by the local client; all the data is stored at the data base server and special computations are executed by a compute server. The application program running on the client, calls the required functions provided by the servers. This is done mainly by way of remote procedure calls (RPC). An RPC resembles a procedure call except that it is used in distributed systems. The following is a description of how the RPC works. The program running on the client looks like a normal sequential program. The services of a particular server are invoked via a remote procedure call. The caller of a remote procedure is stopped until the invoked remote procedure is finished and the server has provided the results to the calling client in the same way that parameters are returned by a procedure. The servers are used in the same way that library procedures are used. This means that remote procedure calls hide the distribution of the functions of the system even at the program level. The programmer does not need to concern himself with the system distribution.

The figure below shows the basic structure of a client/server system.

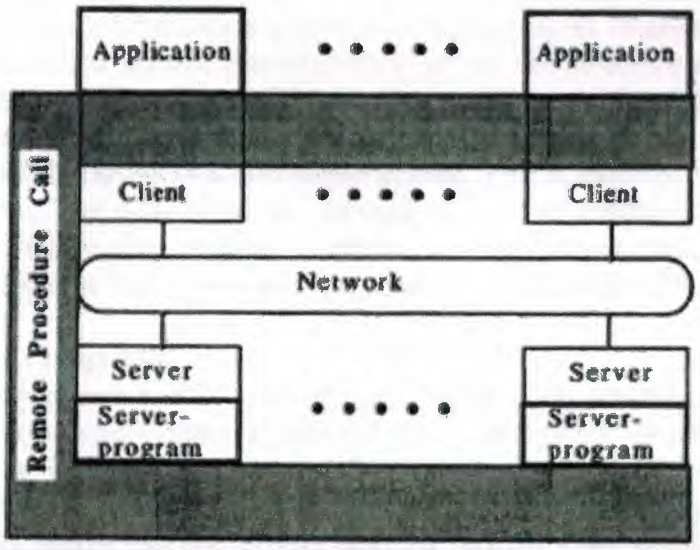


Figure 2.5: Remote Procedure Call Concept

2.3.2.Distributed Computing Environment (DCE)

The Distributed Computing Environment is a comprehensive integrated set of tools which supports network computing in a heterogeneous computing environment. This set of technologies has been selected by the Open Systems Foundation (OSF) to support the development of distributed applications for heterogeneous computer networks. The following figure shows the OSF DCE architecture.

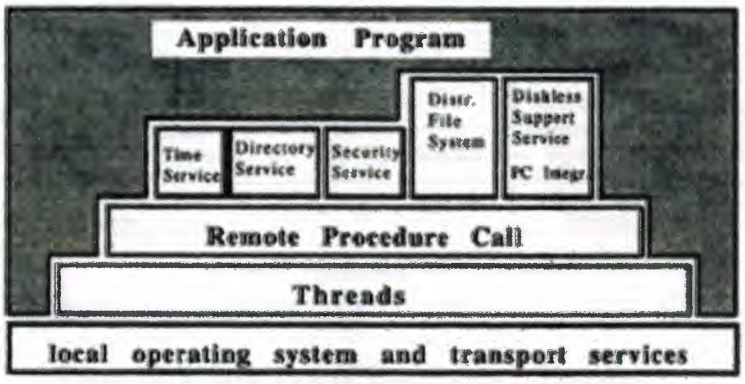


Figure 2.6: Architecture of OSFDCE

In the DCE client and server programs are executed by threads i.e. processes. Threads use an RPC in order to communicate with each other and binary semaphores and conditional variables for synchronization. In the DCE remote procedure calls are supported by directory services (DCE Call Directory Service) and security services (DCE Security Service). Directory services map logical names to physical addresses. If a client calls a particular service provided by a server, the directory service is used to find the appropriate server. The DCE security service provides features for secure communication and controlled access to resources. Distribute Time Service provides precise clock synchronization in a distributed system. This is required for event logging, error recovery, etc. The distributed file service allows the sharing of files across the whole system. Finally the diskless support service allows workstations to use background disk files on file servers as if they were local disks /SCHILL93/, /OSF92/.

2.3.3.Cooperative Computing

In cooperative computing a set of processes runs on several processing nodes. These processes cooperate to reach a common goal and together they form a distributed program. This is different from the client/server systems described above. In cooperative systems the processes, which comprise the distributed program are coupled very closely. This means that the closely coupled processes are executed on a loosely coupled system.

In cooperative systems, the distribution of computing capability is not hidden behind programming concepts. The different program sections running on different computers comprise a single program; but it can be seen at the programming level that the program sections are executed concurrently. These different program sections are also processes. Processes form a very important concept for central systems, client server systems and cooperative systems. If processes have to work together to perform their task, they must exchange data and synchronize their execution. Programming systems for concurrent systems contain communication and synchronization concepts. Cooperative programming resembles a human organization which works together to achieve a common goal. Its members must communicate with each other and must synchronize their activities. The following figure shows the basic structure of cooperative systems.

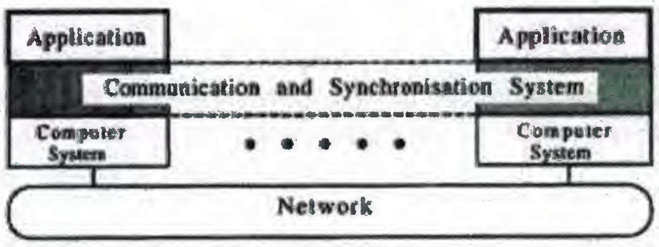


Figure 2.7: Structure of Cooperative Systems

Cooperative systems are mainly used for the automation of technical processes and the implementation of communication software, etc. Technical processes in the mostly part consist of several parallel activities, for example checking the level of a tank has to be done in parallel with controlling the rate of flow of a pump. Therefore the structure of technical process control software is very similar to the structure of the technical process to be controlled. For the automation of technical processes such as manufacturing control systems, the environment of the program, the technical process, is considered as a set of processes which interact with software processes. This means that several processes which can be implemented in different ways work together to perform their task.

2.4 Communication Software Systems

A communication system consists of a communication network and the communication software, which runs on the various processing nodes (referred to as host systems). The communication software provides a more or less convenient communication service for the application software. The application software on each node uses the communication service to exchange messages with the application software running on other nodes. The communication service is based on the underlying network (A network is usually made up of lines and several switching nodes although most local area networks do not contain switching nodes).

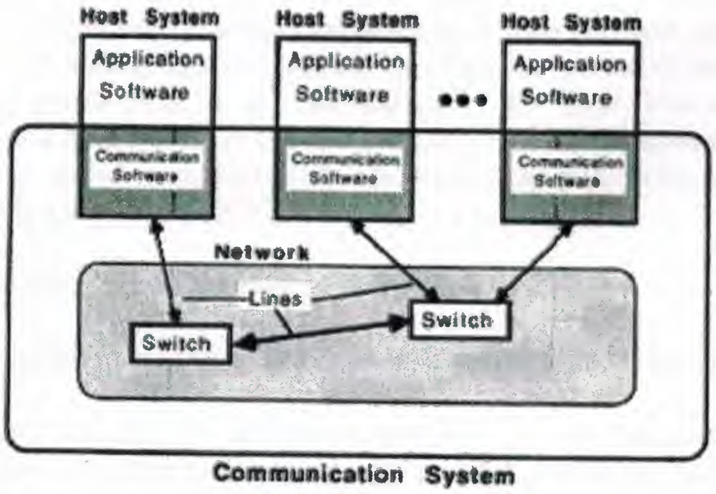


Figure 2.8: Structure of Communication Systems

In order to provide a convenient communication service the communication software systems also exchange messages. This message exchange is based on the simpler communication mechanism provided directly by the network. For example the network provides a communication service, which only allows the transfer of a single byte. The communication service provided by the communication software allows byte strings of a fixed or even an unlimited length to be sent or received. This can be implemented in the following way:

The application software of a host system A wants to send a sequence of bytes to the application software of a host system B. The sequence of bytes is given to the communication system by the application system. The communication system on host system A sends a byte with the length of the byte string (the number of bytes) to the communication system on host system B. The communication system on host system B sends back an acknowledgement. This is a byte with a certain value. After the communication software on host system A has received the acknowledgement it starts to transfer the bytes of the byte string. When system B has received the number of bytes indicated in the first byte it again sends an acknowledgement. After sending the acknowledgement, the communication software on host system B gives the received byte string to the application software.

This communication sequence which implements the transfer of a byte string is just a simplistic illustration of what communication software can do.

As the example above shows, the communication between the communication software systems follows well-defined rules. These rules are called protocols. The need to provide convenient communication services for the application software leads to software communication protocols, which can be extremely complex and must be organized in layers. Each layer offers an improved communication service to the layer above. The widely used reference model for Open Systems Interconnection (OSI) defined by the International Standard Organization (ISO) proposes seven protocol layers /IS07498/. Each layer provides a certain service to the layer above. The service provided by a layer is implemented by the protocol specific to its layer and by the services of the layer below. In a host system the services specific to the layer are realized by protocol entities. The layer protocol is defined between protocol entities of the same layer. These exchange information by using the service of the layer below. In each host system there must be at least one entity per layer. The set of entities of different layers in a host system is called a protocol stack. The implementation of these protocol stacks is called communication software. Communication software has the following execution properties /DROB86/:

• interleaved execution of several entities on the same system

• distributed execution of entities of the same layer on different systems.

Interleaved and distributed computations are usually modeled as systems of parallel processes. Processes executing in parallel normally have to exchange information if they are to cooperate in solving a common task. One or more processes model entities. Using or providing a service means exchanging information with processes representing entities of the layer below or above. The figure above shows

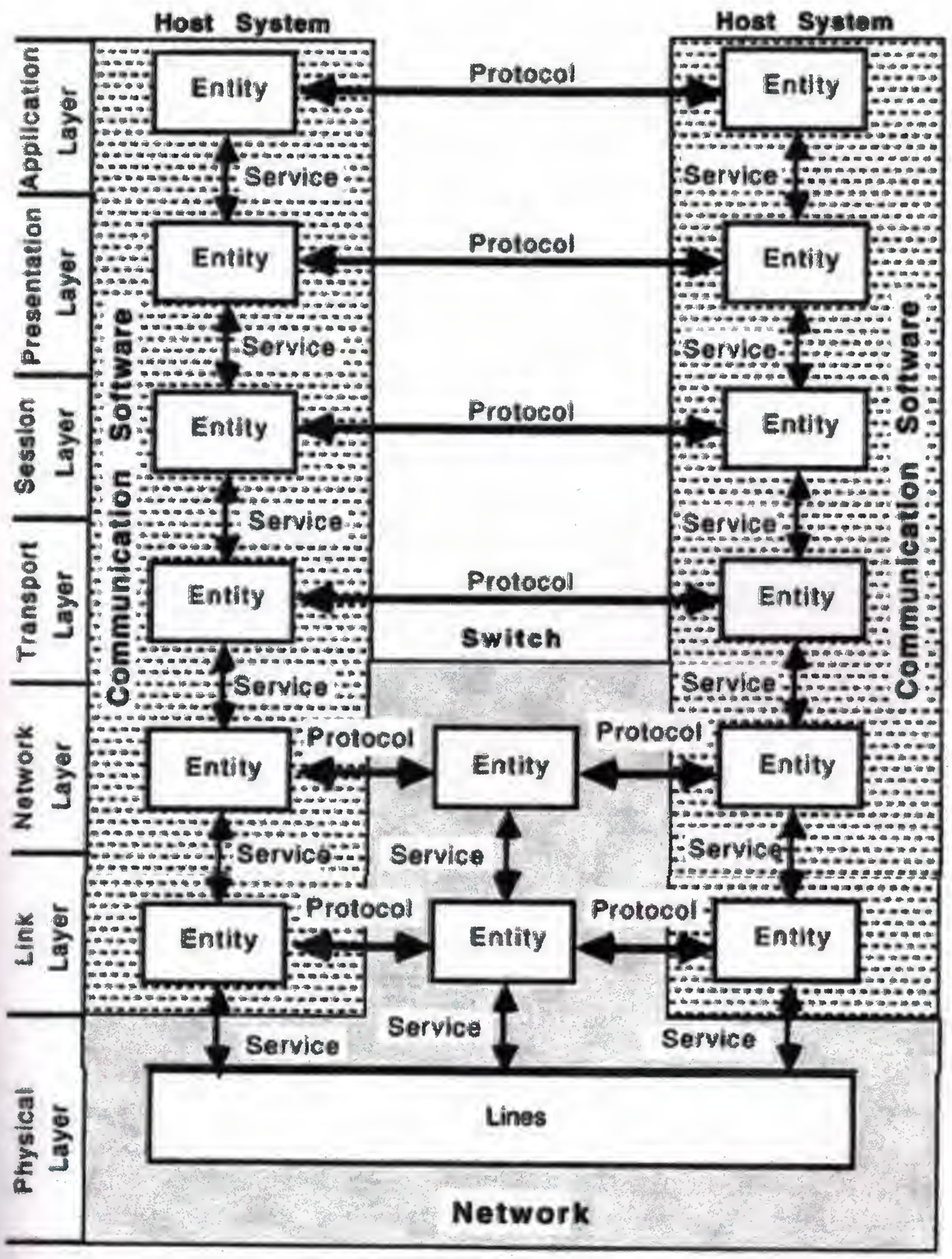


Figure 2.9: Structure of Communication Software

the structure of communication software systems based on the ISO/OSI reference model. Protocol stacks in the different host systems are implemented independently of each other and are embedded in the communication systems. This means that the implementation of a communication system to support communication in a distributed program is itself a distributed program.

2.4 .2 Technical Process Control Software Systems

Another important example of cooperative computing is a distributed technical process control system.

The basic structure of technical systems controlled by computer systems is shown in the following figure /NEHM84/.

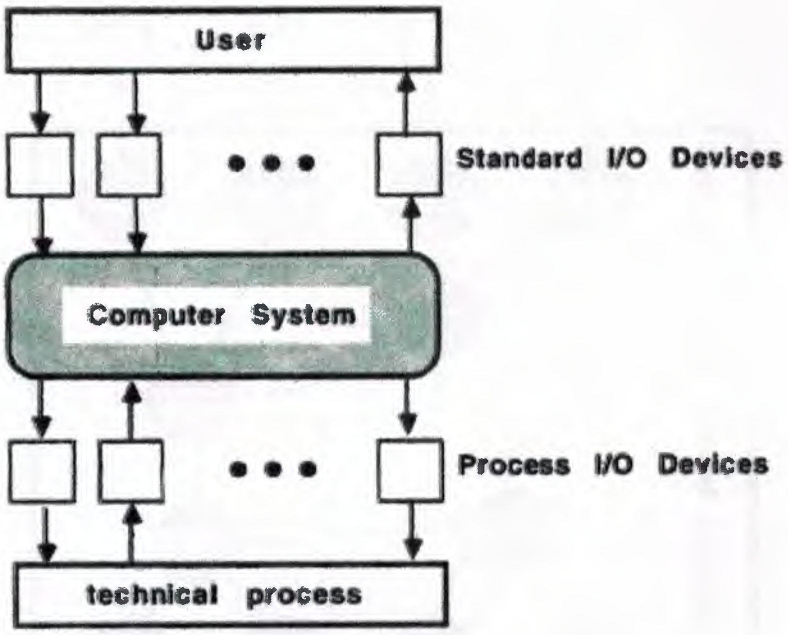


Figure 2.10: Structure of Process Control Systems

The communication between computer systems and technical systems must meet hard real time requirements, whereas the communication with the user is more or less dialogue-oriented with less emphasis on time conditions (except in the case emergency signals such as fire alarms). For the sake of simplicity, we will focus on the relationship between technical systems and real-time computer systems.

A technical system consists of several mutually independent functional units which communicate via appropriate interfaces with the computer system. Therefore the real time program must react to several simultanous inputs. This implies the structuring of a process control software system that takes into account a number of processes. Each process handles a certain group of signals.

The basic requirement for a process control software system is the capability to follow the changes of the technical system as fast as possible. The information in the process control software must be as close as possible to the state of the technical system. The easiest way to achieve this is to design a process for each interface element. This leads to the software system structure shown in the following figure /NEHM84/.

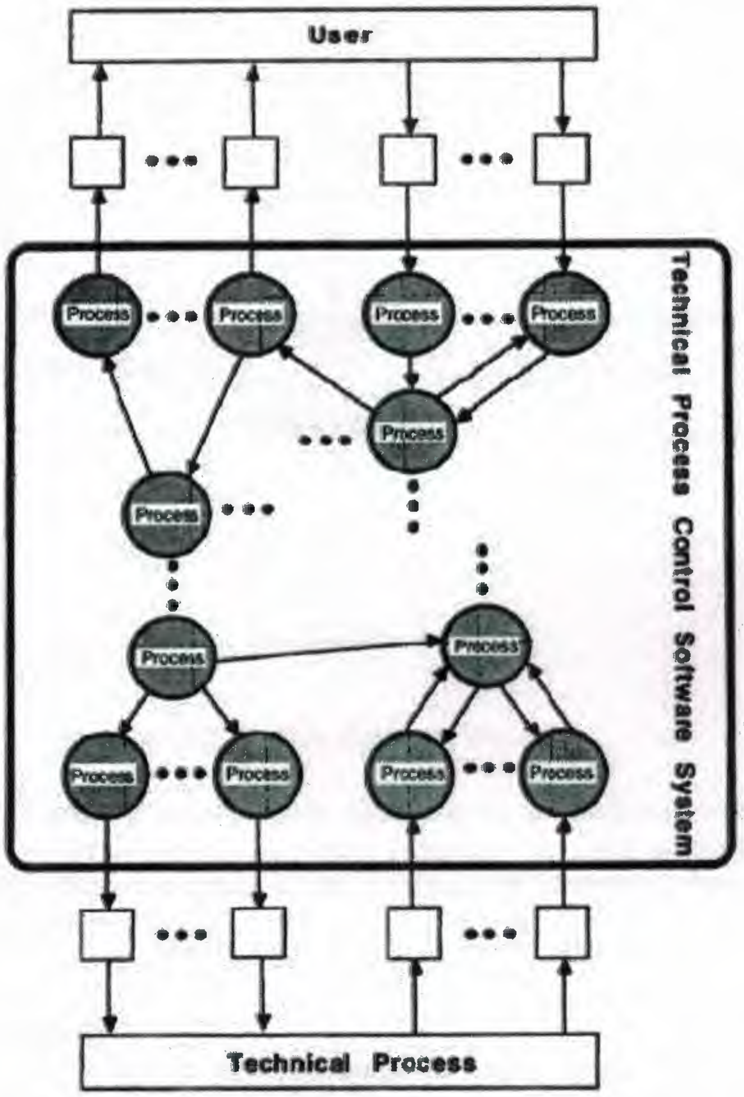


Figure 2.11: Structure of Process Control Software

Software system processes can run on a single centralized system or can be distributed over several computer systems. In the latter case it is possible to locate the computers close to the device or the plant being controlled. The main advantages of distributed solutions are:

• reduction of wiring costs

• faster response

• easier development and maintenance

• a higher degree of fault tolerance

2.4 .3 Electronic Data Interchange (EDI)

Electronic Data Interchange (EDI) is the computer-to-computer exchange of inter and intra company technical and business data, based on the use of standards /DIGIT90/ (see figure below of the EDI business model).

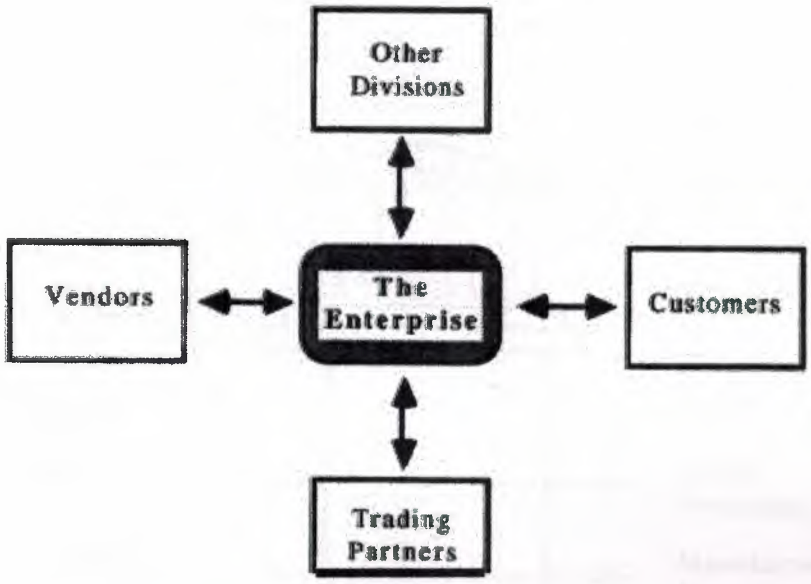


Figure 2.12: EDI Business Model

These data can be structured or unstructured. Exchanging unstructured data follows specific communication standards although the data content is not in a structured format. More important is the exchange of structured data. Examples of structured data exchange are:

- Trade Data Interchange

  This type of EDI document exchange is mainly used to automate business processes. Examples of trade data interchanges include a request for quotation (RfQ), purchase orders, purchase order acknowledgements, etc. Each company and industry has its own requirements for the structure and contents of these documents. A number of specific industry and national bodies have been formed with the intention of standardizing the format and content of messages. For the chemical industry CEFIC is the EDI standard and for the auto industry the related EDI standard is called ODETTE. The standard defined by CCITT is called EDIFACT. In order to exchange EDIFACT documents very often the CCITT E-Mail standard X.400 is recommended /HILL90/.

- Electronic Funds Transfer

  Payment against invoices, electronic point of sale (EPOS) and clearing systems are examples of electronic funds transfer.

- Technical Data Interchange

  Improvement in technical communication can play a key role in determining the success of a project. There is a growing demand from traders for communication between their CAD (computer aided design) workstation and the workstations of important vendors.

The following example shows how the different types of EDI interactions are used to handle a business process.

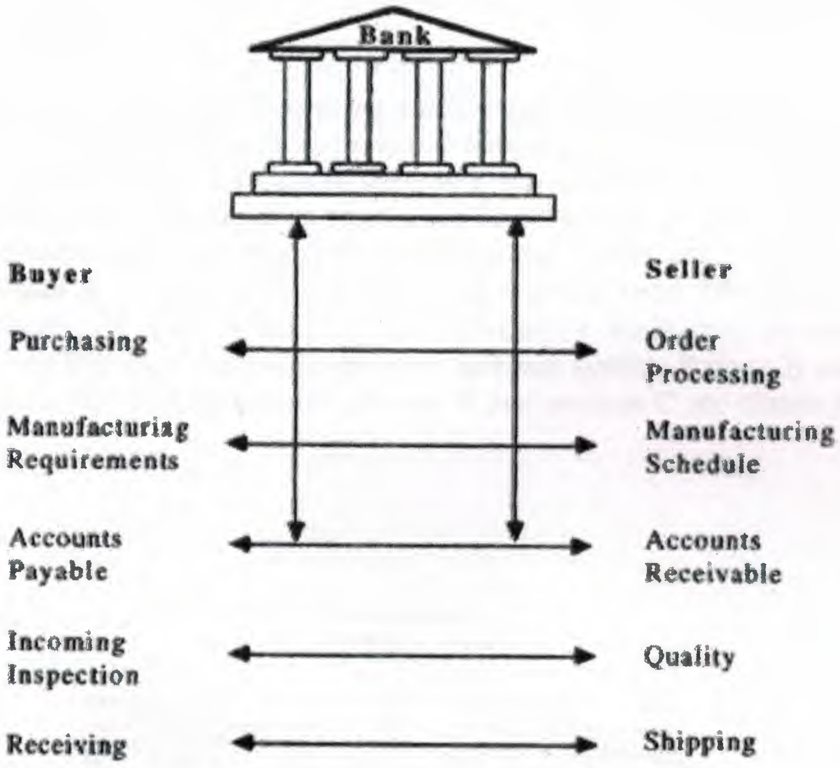


Figure 2.13: EDI in a Business Process

2.4.4 Groupware

In organizations people work together to reach a common goal. The formal interaction between members of an organization is described by structures and procedures. Additionally there exist informal interactions which are very important. Both types of interactions can and should be supported by computers. Computer Supported Cooperative Work (CSCW) deals with the study and development of computer systems called groupware, which purpose it is to facilitate these formal and informal interactions . CSCW projects can be classified into four types namely:

1. Groups which are not geographically distributed and require common access in real time Examples: presentation software, group decision systems

2. Groups which are geographically distributed and require common access in real time Examples: video conferencing, screen sharing

3. Asynchronous collaboration among people who are geographically distributed. Examples: notes conferences, joint editing

4. Asynchronous collaboration among people who are not geographically distributed Examples: project management, personal time schedule management

Groupware requires computers connected by a network. Thus groupware systems are distributed systems. Members of a group share data and exchange messages. Therefore groupware software systems are combinations of network and cooperative computing.

2.5 Combination of Network Computing and Cooperative Computing

Cooperative computing can be combined with client server systems. Processes in a distributed system can have access to servers. From the standpoint of a client server system the processes of a cooperative system can be considered as client processes. In a technical process control software system a process can collect data from the technical process. This data is stored in a file located on a file server node. The following figure shows an example of a combination of a cooperative and a client/server system. Process A, Process B and Process C form a cooperative software system. Process B and Process C use the file server. This means that process B and process C are clients of the file server.

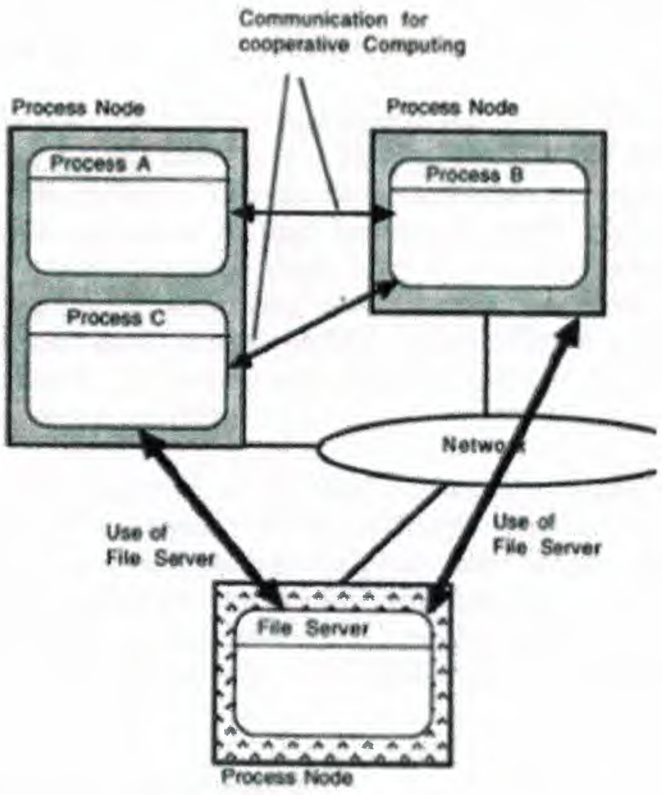


Figure 2.14: Combination of Cooperative and Client Server System

CHAPTER 3: WHAT IS MySQL?

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL. Since computers are very good at handling large amounts of data, database management plays a central role in computing, as stand-alone utilities, or as parts of other applications.

MySQL is a relational database management system. A relational database stores data in separate tables rather than putting all the data in one big store room. This adds speed and flexibility. The tables are linked by defined relations making it possible to combine data from several tables on request. The SQL part of MySQL stands for "Structured Query Language" the most common standardised language used to access databases.

MySQL is Open Source Software. Open Source means that it is possible for anyone to use and modify. Anybody can download MySQL from the Internet and use it without paying anything. Anybody so inclined can study the source code and change it to fit their needs. MySQL uses the GPL (GNU General Public License).

3.1 Why Use MySQL

MySQL is very fast, reliable, and easy to use. If that is what you are looking for, you should give it a try. MySQL also has a practical set of features developed in close cooperation with our users. You can find a performance comparison of MySQL to some other database managers on our benchmark page. MySQL was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Though under constant development, MySQL today offers a rich and useful set of functions. The connectivity, speed, and security make MySQL highly suited for accessing databases on the Internet.

MySQL is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and several programming interfaces. We also provide MySQL as a multi-threaded library which you can link into your application to get a smaller, faster, easier to manage product. MySQL has a lot of contributed software available. It is very likely that you will find that your favorite application or language already supports MySQL.

3.2 The Main Features of MySQL

The following list describes some of the important characteristics of MySQL. Internals and Portability

- Written in C and C++. Tested with a broad range of different compilers.
- No memory leaks. MySQL has been tested with Purify, a commercial memory leakage detector.

- Works on many different platforms.
- Uses GNU Automake, Autoconf, and Libtool for portability.
- Fully multi-threaded using kernel threads. This means it can easily use multiple CPUs if available.
- Very fast B-tree disk tables with index compression.
- A very fast thread-based memory allocation system.
- Very fast joins using an optimised one-sweep multi-join.
- In-memory hash tables which are used as temporary tables.
- SQL functions are implemented through a highly optimised class library and should be as fast as possible! Usually there isn't any memory allocation at all after query initialisation.

3.2.1 Column Types

- Many column types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME,TIMESTAMP, YEAR, SET, and ENUM types.
- Fixed-length and variable-length records.
- All columns have default values. You can use INSERT to insert a subset of a table's columns; those columns that are not explicitly given values are set to their default values.

3.2.2 Commands and Functions

- Full operator and function support in the SELECT and WHERE parts of queries. For example:

```
        mysql> SELECT   CONCAT( first_name , " ", last_name )   FROM
table_name            WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL GROUP BY and ORDER BY clauses. Support for group functions (COUNT(), COUNT(DISTINCT ...), AVG(), STD(), SUM(), MAX() and MIN()).
- Support for LEFT OUTER JOIN and RIGHT OUTER JOIN with ANSI SQL and ODBC syntax.
- Aliases on tables and columns are allowed as in the SQL92 standard.
- DELETE, INSERT, REPLACE, and UPDATE return the number of rows that were changed (affected). It is possible to return the number of rows matched instead by setting a ag when connecting to the server.
- The MySQL-specific SHOW command can be used to retrieve information about databases, tables, and indexes. The EXPLAIN command can be used to determine how the optimiser resolves a query.
- Function names do not clash with table or column names. For example, ABS is a valid column name. The only restriction is that for a function call, no spaces are allowed between the function name and the `(' that follows it.
- You can mix tables from different databases in the same query Security.
- A privilege and password system that is very flexible and secure, and allows host-based verification. Passwords are secure because all password traffic is encrypted when you connect to a server.

3.2.3 Scalability and Limits

- Handles large databases. We are using MySQL with some databases that contain 50,000,000 records and we know of users that uses MySQL with 60,000 tables and about 5,000,000,000 rows.
- Up to 32 indexes per table are allowed. Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 500 bytes (this may be changed when compiling MySQL). An index may use a prefix of a CHAR or VARCHAR field.

3.2.4 Connectivity

- Clients may connect to the MySQL server using TCP/IP Sockets, Unix Sockets (Unix), or Named Pipes (NT).

3.2.5 Localization

- The server can provide error messages to clients in many languages.
- Full support for several different character sets.
- All data is saved in the chosen character set. All comparisons for normal string columns are case insensitive.
- Sorting is done according to the chosen character set (the Swedish way by default). It is possible to change this when the MySQL server is started up. MySQL supports many different character sets that can be specified at compile and run time.

3.2.6 Clients and Tools

- All MySQL programs can be invoked with the --help or -? options to obtain online assistance.

CHAPTER 4: USING MYSQL

Mysql (sometimes referred to as the “terminal monitor" or just “monitor") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you place your queries in a file beforehand, then tell mysql to execute the contents of the file. To see a list of options provided by mysql, invoke it with the --help option:

```
shell> mysql –help
```

4.1 Connecting to and Disconnecting from the Server

To connect to the server, you'll usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

The ******** represents your password; enter it when mysql displays the Enter password Prompt. If that works, you should see some introductory information followed by a

```
Mysql > prompt;
Shell > mysql   -h   host    -u   user   -p   Enter password: ********
```

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 459 to server version: 3.22.20a-log.

Type 'help' for help.

```
mysql>
```

The prompt tells you that mysql is ready for you to enter commands.Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing QUIT at the

```
mysql> prompt:

mysql> QUIT

Bye
```

You can also disconnect by pressing Control-D.

4.2 Entering Queries

At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how mysql works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown below following the mysql> prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+-------------------+
| version() | CURRENT_DATE |
+--------------+-------------------+
| 3.22.20a-log | 1999-03-19      |
+--------------+-------------------+

1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about mysql:

A command normally consists of a SQL statement followed by a semicolon. (There are some exceptions where a semicolon is not needed. QUIT, mentioned earlier, is one of them. We'll get to others later.When you issue a command, mysql sends it to the server for execution and displays the results, then prints another mysql to indicate that it is ready for another command.

Mysql displays query output as a table (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), mysql labels the column using the expression itself. Mysql shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency.

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;

mysql> select version(), current_date;

mysql> SeLeCt vErSiOn(), current_DATE;
```

Here's another query. It demonstrates that you can use mysql as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+-----------+
| SIN(PI()/4) | (4+1)*5  |
+-------------+-----------+
| 0.707107 | 25          |
+-------------+-----------+
```

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+------------------------+
| version()              |
+--------------          +
| 3.22.20a-log           |
+------------------------+
+------------------------+
| NOW()                  |
+------------------------+
| 1999-03-19 00:15:33 |
+------------------------+
```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. mysql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, mysql accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+-------------------+-----------------+
| USER() | CURRENT_DATE        |
+-------------------+-----------------+
| joesmith@localhost | 1999-03-18 |
+-------------------+-----------------+
```

In this example, notice how the prompt changes from mysql> to -> after you enter the first line of a multiple-line query. This is how mysql indicates that it hasn't seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you will always be aware of what mysql is waiting for. If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing \c:

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

Here, too, notice the prompt. It switches back to mysql> after you type \c, providing feedback to indicate that mysql is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that mysql is in:

Prompt	Meaning
mysql>	Ready for new command.
->	Waiting for next line of multiple-line command.
'>	Waiting for next line, collecting a string that begins with a single quote (`').
">	Waiting for next line, collecting a string that begins with a double quote (`"').

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, mysql waits for more input:

```
mysql> SELECT USER()
```

If this happens to you (you think you've entered a statement but the only response is a
prompt, most likely mysql is waiting for the semicolon. If you don't notice what the
prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and mysql will execute it:

```
mysql> SELECT USER()
-> ;
+-----------------------+
| USER()                |
+-----------------------+
| joesmith@localhost    |
+-----------------------+
```

The '> and "> prompts occur during string collection. In MySQL, you can write strings surrounded by either `' or `"' characters (for example, 'hello' or "goodbye"), and mysql lets you enter strings that span multiple lines. When you see a '> or "> prompt, it means that you've entered a line containing a string that begins with a `'' or `"' quote character, but have not yet entered the matching quote that terminates the string. That's fine if you really are entering a multiple-line string, but how likely is that? Not very. More often, the '> and "> prompts indicate that you've inadvertantly left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;">
```

If you enter this SELECT statement, then press Enter and wait for the result, nothing will happen. Instead of wondering why this query takes so long, notice the clue provided by the "> prompt. It tells you that mysql expects to see the rest of an unterminated string. (Do you see the error in the statement? The string "Smith is missing the second quote.)At this point, what do you do? The simplest thing is to cancel the command. However,you cannot just type \c in this case, because mysql interprets it as part of the string that it is collecting! Instead, enter the closing quote character (so mysql knows you've finished the string), then type \c:

```
        mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;">
"\c
        mysql>
```

The prompt changes back to mysql>, indicating that mysql is ready for a new command.It's important to know what the '> and "> prompts signify, because if you mistakenly enter an unterminated string, any further lines you type will appear to be ignored by mysql including a line containing QUIT! This can be quite confusing, especially if you don't know that you need to supply the terminating quote before you can cancel the current command.

4.3 Creating and Using a Database

Suppose you have several pets in your home (your menagerie) and you'd like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables.

Create a database
Create a table
Load data into the table
Retrieve data from the table in various ways
Use multiple tables

The menagerie database will be simple (deliberately), but it is not difficult to think of real world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records.

Use the SHOW statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
```

The mysql database is required because it describes user access privileges. The test database is often provided as a workspace for users to try things out. If the test database exists, try to access it:

```
mysql> USE test

Database changed
```

Note that USE, like QUIT, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The USE statement is special in another way too : it must be given on a single line. You can use the test database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose you want to call yours menagerie. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO your_mysql_name;
```

where your_mysql_name is the MySQL user name assigned to you.

4.4 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this command:

```
mysql> USE menagerie

Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a mysql session. You can do this by issuing a USE statement as shown above. Alternatively, you can select the database on the command line when you invoke mysql. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Note that menagerie is not your password on the command just shown. If you want to supply your password on the command line after the -p option, you must do so with no intervening space (for example, as –pmypassword, not as -p mypassword). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

4.5 Creating a Table

Creating the database is the easy part, but at this point it's empty, as SHOW TABLES will tell you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you will need and what columns will be in each of them.

You'll want a table that contains a record for each of your pets. This can be called the pet table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex. How about age? That might be of interest, but it's not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it's better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:
You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you'll soon need to send out birthday greetings, for that computer-assisted personal touch). You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died. You can probably think of other types of information that would be useful in the pet table, but the ones identified so far are sufficient for now: name, owner, species, sex, birth, and death.

Use a CREATE TABLE statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

VARCHAR is a good choice for the name, owner, and species columns because the column values will vary in length. The lengths of those columns need not all be the same, and need not be 20. You can pick any length from 1 to 255, whatever seems most reasonable to you. (If you make a poor choice and it turns out later that you need a longer field, MySQL provides an ALTER TABLE statement). Animal sex can be represented in a variety of ways, for example, "m" and "f", or perhaps "male" and "female". It's simplest to use the single characters "m" and "f". The use of the DATE data type for the birth and death columns is a fairly obvious choice.
Now that you have created a table, SHOW TABLES should produce some output:

To verify that your table was created the way you expected, use a DESCRIBE statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra              |
+---------+-------------+------+-----+---------+-------+
| name | varchar(20) | YES | | NULL |                     |
| owner | varchar(20) | YES | | NULL |                    |
| species | varchar(20) | YES | | NULL |                    |
| sex | char(1) | YES | | NULL |                           |
| birth | date | YES | | NULL |                            |
| death | date | YES | | NULL |                            |
+---------+-------------+------+-----+---------+-------+
```

You can use DESCRIBE any time, for example, if you forget the names of the columns in your table or what types they are.

4.6 Loading Data into a Table

After creating your table, you need to populate it. The LOAD DATA and INSERT statements are useful for this. Suppose your pet records can be described as shown below. (Observe that MySQL expects dates in YYYY-MM-DD format; this may be different than what you are used to.)

```
name owner species sex birth death
Fluffy Harold cat f 1993-02-04
Claws Gwen cat m 1994-03-17
Buffy Harold dog f 1989-05-13
Fang Benny dog m 1990-08-27
Bowser Diane dog m 1998-08-31 1995-07-29
Chirpy Gwen bird f 1998-09-11
Whistler Gwen bird 1997-12-09
Slim Benny snake m 1996-04-29
```

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, than load the contents of the file into the table with a single statement.

You could create a text file `pet.txt' containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the CREATE TABLE statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use NULL values. To represent these in your text file, use N. For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler Gwen bird \N 1997-12-09 \N
```

To load the text _le `pet.txt' into the pet table, use this command:

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

You can specify the column value separator and end of line marker explicitly in the LOAD DATA statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt' properly.

When you want to add new records one at a time, the INSERT statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the CREATE TABLE statement. Suppose Diane gets a new hamster named Puffball.

You could add a new record using an INSERT statement like this:

```
mysql> INSERT INTO pet
-> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Note that string and date values are specified as quoted strings here. Also, with INSERT, you can insert NULL directly to represent a missing value. You do not use \N like you do with LOAD DATA.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several INSERT statements rather than a single LOAD DATA statement.

4.7 Retrieving Information from a Table

The SELECT statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

what_to_select indicates what you want to see. This can be a list of columns, or * to
indicate *all columns." which_table indicates the table from which you want to retrieve
data. The WHERE clause is optional. If it's present, conditions_to_satisfy specifies
conditions that rows must satisfy to qualify for retrieval.

4.8 Selecting All Data

The simplest form of SELECT retrieves everything from a table:

```
mysql> SELECT * FROM pet;
```

```
+---------+--------+---------+-----+------------+------------+
| name | owner | species | sex | birth | death                |
+---------+--------+---------+-----+------------+------------+
| Fluffy | Harold | cat | f | 1993-02-04 | NULL               |
| Claws | Gwen | cat | m | 1994-03-17 | NULL                 |
| Buffy | Harold | dog | f | 1989-05-13 | NULL               |
| Fang | Benny | dog | m | 1990-08-27 | NULL                 |
| Bowser | Diane | dog | m | 1998-08-31 | 1995-07-29      |
| Chirpy | Gwen | bird | f | 1998-09-11 | NULL               |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL         |
| Slim | Benny | snake | m | 1996-04-29 | NULL              |
| Puffball | Diane | hamster | f | 1999-03-30 | NULL        |
+----------+--------+---------+------+------------+------------+
```

This form of SELECT is useful if you want to review your entire table, for instance, after you've just loaded it with your initial dataset. As it happens, the output just shown reveals an error in your data file: Bowser appears to have been born after he died! Consulting your original pedigree papers, you find that the correct birth year is 1989, not 1998.

There are least a couple of ways to fix this:
_ Edit the file `pet.txt' to correct the error, then empty the table and reload it using
DELETE and LOAD DATA:

```
mysql> SET AUTOCOMMIT=1; # Used for quick re-create of the table
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

However, if you do this, you must also re-enter the record for Puffball.
Fix only the erroneous record with an UPDATE statement:

```
mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Bowser";
```

As shown above, it is easy to retrieve an entire table. But typically you don't want to do that, particularly when the table becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

4.8.1 Selecting Particular Rows

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
+--------+-------+---------+------+------------+------------+
| name | owner | species | sex | birth | death |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded now as 1989, not 1998.String comparisons are normally case insensitive, so you can specify the name as "bowser",
"BOWSER", etc. The query result will be the same.

You can specify conditions on any column, not just name. For example, if you want to know which animals were born after 1998, test the birth column:

```
mysql> SELECT * FROM pet WHERE birth >= "1998-1-1";
+----------+-------+---------+------+-----------+-------+
| name | owner | species | sex | birth | death |
+----------+-------+---------+------+-----------+-------+
| Chirpy | Gwen | bird | f | 1998-09-11 | NULL |
138 MySQL Technical Reference for Version 4.0.1-alpha
| Puffball | Diane | hamster | f | 1999-03-30 | NULL |
+----------+-------+---------+------+-----------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = "dog" AND sex = "f";
+-------+--------+---------+------+-----------+-------+
| name | owner | species | sex | birth | death |
+-------+--------+---------+------+-----------+-------+
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-------+--------+---------+------+-----------+-------+
```

The preceding query uses the AND logical operator. There is also an OR operator:

```
mysql> SELECT * FROM pet WHERE species = "snake" OR species = "bird";
+----------+-------+---------+------+-----------+-------+
| name | owner | species | sex | birth | death |
+----------+-------+---------+------+-----------+-------+
| Chirpy | Gwen | bird | f | 1998-09-11 | NULL |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL |
| Slim | Benny | snake | m | 1996-04-29 | NULL |
+----------+-------+---------+------+-----------+-------+
```

AND and OR may be intermixed. If you do that, it's a good idea to use parentheses to indicate how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = "cat" AND sex = "m")
-> OR (species = "dog" AND sex = "f");
+-------+--------+---------+------+-----------+-------+
| name | owner | species | sex | birth | death |
+-------+--------+---------+------+-----------+-------+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-------+--------+---------+------+-----------+-------+
```

4.8.2 Selecting Particular Columns

If you don't want to see entire rows from your table, just name the columns in which you're interested, separated by commas. For example, if you want to know when your animals were born, select the name and birth columns:

```
Mysql> SELECT name, birth FROM pet;
```

```
+----------+---------------+
| name | birth |
+----------+---------------+
| Fluffy | 1993-02-04   |
| Claws | 1994-03-17  |
| Buffy | 1989-05-13   |
| Fang | 1990-08-27   |
| Bowser | 1989-08-31|
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
|Slim | 1996-04-29    |
| Puffball | 1999-03-30 |
```

To find out who owns pets, use this query:
mysql> SELECT owner FROM pet;

```
+--------+
| owner |
+--------+
| Harold |
| Gwen |
| Harold |
| Benny |
| Diane |
| Gwen |
| Gwen |
| Benny |
| Diane |
+--------+
```

However, notice that the query simply retrieves the owner field from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword DISTINCT:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner |
+--------+
| Benny |
| Diane |
| Gwen |
| Harold |
+--------+
```

You can use a WHERE clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

mysql> SELECT name, species, birth FROM pet

```
-> WHERE species = "dog" OR species = "cat";

+--------+---------+------------+
| name | species | birth |
+--------+---------+------------+
| Fluffy | cat | 1993-02-04     |
| Claws | cat | 1994-03-17      |
| Buffy | dog | 1989-05-13      |
| Fang | dog | 1990-08-27       |
| Bowser | dog | 1989-08-31     |
+--------+---------+------------+
```

4.8.3 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. However, it's often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an ORDER BY clause. Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name | birth |
+----------+------------+
| Buffy | 1989-05-13 |
| Bowser | 1989-08-31 |
| Fang | 1990-08-27 |
| Fluffy | 1993-02-04 |
| Claws | 1994-03-17 |
| Slim | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To sort in reverse order, add the DESC (descending) keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name | birth |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim | 1996-04-29 |
| Claws | 1994-03-17 |
| Fluffy | 1993-02-04 |
| Fang | 1990-08-27 |
| Bowser | 1989-08-31 |
| Buffy | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns. For example, to sort by type of animal, then by birth date within animal type with youngest animals first, use the following query:

mysql> SELECT name, species, birth FROM pet ORDER BY species, birth DESC;

```
+----------+---------+------------+
| name | species | birth |
+----------+---------+------------+
| Chirpy | bird | 1998-09-11 |
| Whistler | bird | 1997-12-09 |
| Claws | cat | 1994-03-17 |
| Fluffy | cat | 1993-02-04 |
| Fang | dog | 1990-08-27 |
| Bowser | dog | 1989-08-31 |
Chapter 3: Introduction to MySQL: A MySQL Tutorial 141
| Buffy | dog | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim | snake | 1996-04-29 |
+----------+---------+------------+
```

4.9 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates. To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURRENT_DATE,
-> (YEAR(CURRENT_DATE)-YEAR(birth))
-> - (RIGHT(CURRENT_DATE,5)<RIGHT(birth,5))
-> AS age
-> FROM pet;
```

```
+----------+------------+--------------+--------+
| name | birth | CURRENT_DATE       | age |
+----------+------------+--------------+--------+
| Fluffy | 1993-02-04 | 2001-08-29     | 8   |
| Claws | 1994-03-17 | 2001-08-29      |  7 |
| Buffy | 1989-05-13 | 2001-08-29      | 12 |
| Fang | 1990-08-27 | 2001-08-29      | 11 |
| Bowser | 1989-08-31 | 2001-08-29    | 11 |
| Chirpy | 1998-09-11 | 2001-08-29    | 2 |
| Whistler | 1997-12-09 | 2001-08-29 | 3 |
| Slim | 1996-04-29 | 2001-08-29 | 5  |
| Puffball | 1999-03-30 | 2001-08-29 | 2 |
+----------+------------+--------------+------+
```

Here, YEAR() pulls out the year part of a date and RIGHT() pulls of the rightmost characters that represent the MM-DD (calendar year) part of the date. The part of the expression that compares the MM-DD values evaluates to 1 or 0, which adjusts the year difference down a year if CURRENT_DATE occurs earlier in the year than birth. The full expression is somewhat ungainly, so an alias (age) is used to make the output column label more meaningful.

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an ORDER BY name clause to sort the output by name:

```
mysql> SELECT name, birth, CURRENT_DATE,
-> (YEAR(CURRENT_DATE)-YEAR(birth))
-> - (RIGHT(CURRENT_DATE,5)<RIGHT(birth,5))
-> AS age


-> FROM pet ORDER BY name;
```

```
+----------+------------+--------------+------+
| name | birth | CURRENT_DATE | age |
+----------+------------+--------------+------+
| Bowser | 1989-08-31 | 2001-08-29 | 11 |
| Buffy | 1989-05-13 | 2001-08-29 | 12 |
| Chirpy | 1998-09-11 | 2001-08-29 | 2 |
| Claws | 1994-03-17 | 2001-08-29 | 7 |
| Fang | 1990-08-27 | 2001-08-29 | 11 |
| Fluffy | 1993-02-04 | 2001-08-29 | 8 |
| Puffball | 1999-03-30 | 2001-08-29 | 2 |
| Slim | 1996-04-29 | 2001-08-29 | 5 |
| Whistler | 1997-12-09 | 2001-08-29 | 3 |
+----------+------------+--------------+------+
```

To sort the output by age rather than name, just use a different ORDER BY clause:

```
mysql> SELECT name, birth, CURRENT_DATE,
-> (YEAR(CURRENT_DATE)-YEAR(birth))
-> - (RIGHT(CURRENT_DATE,5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY age;

+----------+------------+---------------+------+
| name | birth | CURRENT_DATE | age |
+----------+------------+---------------+------+
| Chirpy | 1998-09-11 | 2001-08-29 | 2 |
| Puffball | 1999-03-30 | 2001-08-29 | 2 |
| Whistler | 1997-12-09 | 2001-08-29 | 3 |
| Slim | 1996-04-29 | 2001-08-29 | 5 |
| Claws | 1994-03-17 | 2001-08-29 | 7 |
| Fluffy | 1993-02-04 | 2001-08-29 | 8 |
| Fang | 1990-08-27 | 2001-08-29 | 11 |
| Bowser | 1989-08-31 | 2001-08-29 | 11 |
| Buffy | 1989-05-13 | 2001-08-29 | 12 |
+----------+------------+---------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether or not the death value is NULL. Then, for those with non-NULL values, compute the difference between the death and birth values:

```
mysql> SELECT name, birth, death,
-> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
-> AS age
-> FROM pet WHERE death IS NOT NULL ORDER BY age;

+--------+------------+------------+------+
| name | birth | death | age |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 | 5 |
+--------+------------+------------+------+
```

The query uses death IS NOT NULL rather than death != NULL because NULL is a special value.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the birth column. MySQL provides several date-part extraction functions, such as YEAR(), MONTH(), and DAYOFMONTH(). MONTH() is the appropriate function here. To see how it works, run a simple query that displays the value of both birth and MONTH(birth)

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

```
+----------+-----------+-------------+
| name | birth | MONTH(birth) |
+----------+-----------+-------------+
| Fluffy | 1993-02-04 | 2 |
| Claws | 1994-03-17 | 3 |
| Buffy | 1989-05-13 | 5 |
| Fang | 1990-08-27 | 8 |
| Bowser | 1989-08-31 | 8 |
| Chirpy | 1998-09-11 | 9 |
| Whistler | 1997-12-09 | 12 |
| Slim | 1996-04-29 | 4 |
| Puffball | 1999-03-30 | 3 |
+----------+-----------+-------------+
```

Finding animals with birthdays in the upcoming month is easy, too. Suppose the current month is April. Then the month value is 4 and you look for animals born in May (month 5) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

```
+-------+-----------+
| name | birth |
+-------+-----------+
| Buffy | 1989-05-13 |
+-------+-----------+
```

There is a small complication if the current month is December, of course. You don't just add one to the month number (12) and look for animals born in month 13, because there is no such month. Instead, you look for animals born in January (month 1).You can even write the query so that it works no matter what the current month is. That way you don't have to use a particular month number in the query. DATE_ADD() allows you to add a time interval to a given date. If you add a month to the value of NOW(), then extract the month part with MONTH(), the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MONTH(DATE_ADD(NOW(), INTERVAL 1
MONTH));
```

A different way to accomplish the same task is to add 1 to get the next month after the current one (after using the modulo function (MOD) to wrap around the month value to 0 if it is currently 12):

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MOD(MONTH(NOW()), 12) + 1;
```

Note that MONTH returns a number between 1 and 12. And MOD(something,12) returns a number between 0 and 11. So the addition has to be after the MOD(), otherwise we would go from November (11) to January (1).

4.10 Working with NULL Values

The NULL value can be surprising until you get used to it. Conceptually, NULL means missing value or unknown value and it is treated somewhat differently than other values. To test for NULL, you cannot use the arithmetic comparison operators such as =, <, or !=. To

In MySQL, 0 or NULL means false and anything else means true. The default truth value from a boolean operation is 1.This special treatment of NULL is why, in the previous section, it was necessary to determine which animals are no longer alive using death IS NOT NULL instead of death != NULL.

4.11 Using More Than one Table

The pet table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs:

To contain the pet name so you know which animal each event pertains to.
A date so you know when the event occurred.
A field to describe the event.
An event type field, if you want to be able to categorize events.
Given these considerations, the CREATE TABLE statement for the event table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
-> type VARCHAR(15), remark VARCHAR(255));
```

As with the pet table, it's easiest to load the initial records by creating a tab-delimited text file containing the information:

```
Fluffy 1995-05-15 litter 4 kittens, 3 female, 1 male
Buffy 1993-06-23 litter 5 puppies, 2 female, 3 male
Buffy 1994-06-19 litter 3 puppies, 3 female
Chirpy 1999-03-21 vet needed beak straightened
Slim 1997-08-03 vet broken rib
Bowser 1991-10-12 kennel
Fang 1991-10-12 kennel
Fang 1998-08-28 birthday Gave him a new chew toy
Claws 1998-03-17 birthday Gave him a new ea collar
Whistler 1998-12-09 birthday First birthday
```

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE "event.txt" INTO TABLE event;
```

Based on what you've learned from the queries you've run on the pet table, you should be able to perform retrievals on the records in the event table; the principles are the same.

But when is the event table by itself insufficient to answer questions you might ask?

Suppose you want to find out the ages of each pet when they had their litters. The event

table indicates when this occurred, but to calculate the age of the mother, you need her

birth date. Because that is stored in the pet table, you need both tables for the query:

```
mysql> SELECT pet.name, (TO_DAYS(date) - TO_DAYS(birth))/365 AS age,
remark
-> FROM pet, event
-> WHERE pet.name = event.name AND type = "litter";

+--------+------+-----------------------------+
| name | age | remark |
+--------+------+-----------------------------+
| Fluffy | 2.27 | 4 kittens, 3 female, 1 male |
| Buffy | 4.12 | 5 puppies, 2 female, 3 male |
| Buffy | 5.10 | 3 puppies, 3 female |
+--------+------+-----------------------------+
```

There are several things to note about this query:

The FROM clause lists two tables because the query needs to pull information from both of them.

When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a name column. The query uses WHERE clause to match up records in the two tables based on the name values.

Because the name column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name. You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the pet table with itself to pair up males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
-> FROM pet AS p1, pet AS p2
-> WHERE p1.species = p2.species AND p1.sex = "f" AND p2.sex = "m";


+--------+-----+--------+-----+---------+
| name | sex | name | sex | species |
+--------+-----+--------+-----+---------+
| Fluffy | f | Claws | m | cat |
| Buffy | f | Fang | m | dog |
| Buffy | f | Bowser | m | dog |
+--------+-----+--------+-----+---------+
```

In this query, we specify aliases for the table name in order to refer to the columns and keep straight which instance of the table each column reference is associated with.

4.12 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie |
+------------+
```

To find out what tables the current database contains (for example, when you're not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| event |
| pet |
+---------------------+
```

If you want to find out about the structure of a table, the DESCRIBE command is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;

+---------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name | varchar(20) | YES | | NULL | |
| owner | varchar(20) | YES | | NULL | |
| species | varchar(20) | YES | | NULL | |
| sex | char(1) | YES | | NULL | |
| birth | date | YES | | NULL | |
| death | date | YES | | NULL | |
+---------+-------------+------+-----+---------+-------+
```

Field indicates the column name, Type is the data type for the column, Null indicates whether or not the column can contain NULL values, Key indicates whether or not the column is indexed, and Default specifies the column's default value. If you have indexes on a table, SHOW INDEX FROM table_name produces information about them.

4.13 Using AUTO INCREMENT

The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (id mediumint not null auto_increment,
name char(30) not null,
primary key (id));
INSERT INTO animals (name) values
("dog"),("cat"),("penguin"),("lax"),("whale");
SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name |
+----+---------+
| 1 | dog |
| 2 | cat |
| 3 | penguin |
| 4 | lax |
| 5 | whale |
+----+---------+
```

For MyISAM and BDB tables you can specify AUTO_INCREMENT on secondary column in a multi-column key. In this case the generated value for the autoincrement column is calculated as MAX(auto_increment_column)+1) WHERE prefix=given-prefix. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (grp enum ('fish','mammal','bird') not null,
id mediumint not null auto_increment,
name char(30) not null,
primary key (grp,id));

INSERT INTO animals (grp,name)
values ("mammal","dog"),("mammal","cat"),("bird","penguin"),

("SELECT * FROM animals order by grp,id;
```

Which returns:

```
+--------+----+---------+
| grp | id | name |
+--------+----+---------+
| fish | 1 | lax |
| mammal | 1 | dog |
| mammal | 2 | cat |
| mammal | 3 | whale |
| bird | 1 | penguin |
+--------+----+---------+
```

Note that in this case, the auto increment value will be reused if you delete the row with the biggest auto increment value in any group. You can get the used AUTO_INCREMENT key with the LAST_INSERT_ID() SQL function or the mysql_insert_id() API function.

4.14 MySql Language Structure

MySQL has a very complex, but intuitive and easy to learn SQL interface. This chapter describes the various commands, types, and functions you will need to know in order to use MySQL efficiently and effectively. This chapter also serves as a reference to all functionality included in MySQL. .

4.14.1 Strings

A string is a sequence of characters, surrounded by either single quote (`'') or double quote (`"') characters (only the single quote if you run in ANSI mode). Examples: 'a string' "another string"

Within a string, certain sequences have special meaning. Each of these sequences begins with a backslash (`\'), known as the escape character. MySQL recognizes the following escape sequences:

\0 An ASCII 0 (NUL) character.
\' A single quote (`'') character.
\" A double quote (`"') character.
\b A backspace character.
\n A newline character.
\r A carriage return character.
\t A tab character.
\z ASCII(26) (Control-Z). This character can be encoded to allow you to go around the problem that ASCII(26) stands for END-OF-FILE on Windows. (ASCII(26) will cause problems if you try to use mysql database < filename.)
\\ A backslash (`\') character.

\% A `%' character. This is used to search for literal instances of `%' in contexts where `%' would otherwise be interpreted as a wild-card character.

_ A `_' character. This is used to search for literal instances of `_' in contexts
where `_' would otherwise be interpreted as a wild-card character.

Note that if you use `\%' or `_' in some string contexts, these will return the strings `\%'
and `\_' and not `%' and `_'.

There are several ways to include quotes within a string:

A `'' inside a string quoted with `'' may be written as `'''.

A `"' inside a string quoted with `"' may be written as `""'.

You can precede the quote character with an escape character (`\').

A `'' inside a string quoted with `"' needs no special treatment and need not be doubled
or escaped. In the same way, `"' inside a string quoted with `'' needs no special treatment.

The SELECT statements shown below demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+
mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+
mysql> SELECT "This\nIs\nFour\nlines";
+--------------------+
| This
Is
Four
lines |
+--------------------+
```

If you want to insert binary data into a BLOB column, the following characters must be represented by escape sequences:

NUL ASCII 0. You should represent this by `\0' (a backslash and an ASCII `0' character).

\ ASCII 92, backslash. Represent this by `\\'.

' ASCII 39, single quote. Represent this by `\''.

" ASCII 34, double quote. Represent this by `\"'.

If you write C code, you can use the C API function mysql_escape_string() to escape
characters for the INSERT statement.

4.14.2 Numbers

Integers are represented as a sequence of digits. Floats use `.' as a decimal separator. Either type of number may be preceded by `-' to indicate a negative value.

An integer may be used in floating-point context; it is interpreted as the equivalent
floating-point number.

4.14.3 Hexadecimal Values

MySQL supports hexadecimal values. In number context these act like an integer (64-bit precision). In string context these act like a binary string where each pair of hex digits is converted to a character:

```
mysql> SELECT x'FF'
-> 255
mysql> SELECT 0xa+0;
-> 10
mysql> select 0x5061756c;
-> Paul
```

4.14.4 NULL Values

The NULL value means \no data" and is different from values such as 0 for numeric types or the empty string for string types.

NULL may be represented by \N when using the text file import or export formats

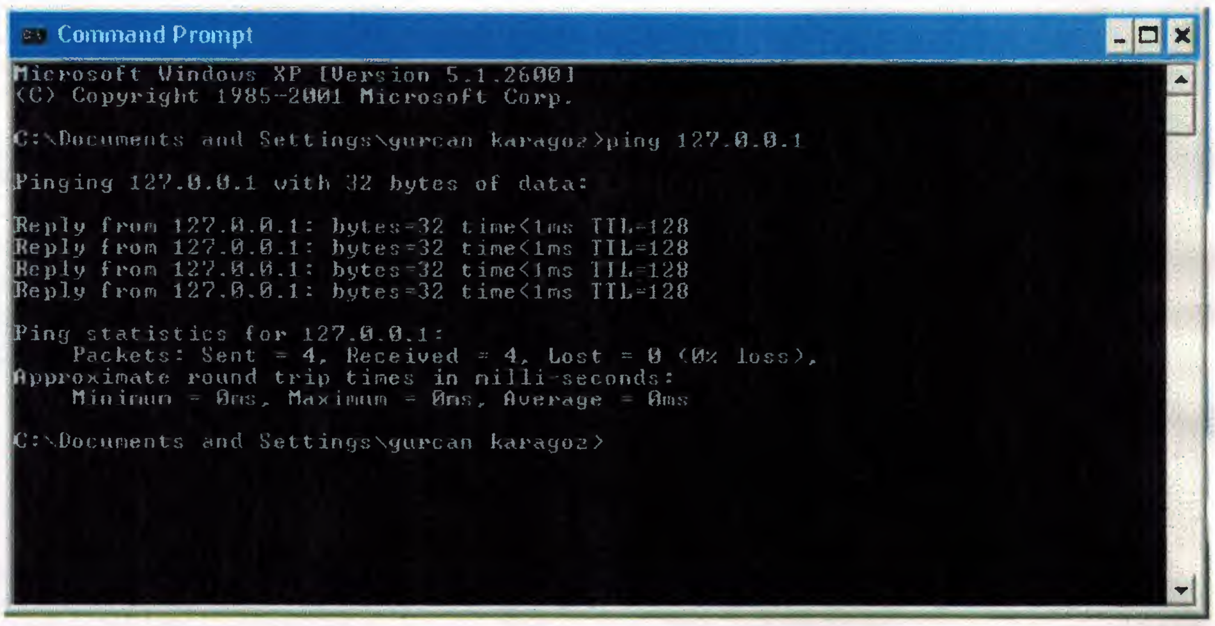
(LOAD DATA INFILE, SELECT ... INTO OUTFILE).

CHAPTER 5: CLIENT SERVER MODELLING

5.1 Structure of System

The project is designed for transferring and receiving customer and account information from server in a Local Area Network or in Internet. The sytem works with the logic of TCP/IP protocol.In server part of the program,Mysql server is running.The databases are created in server by the program administrator and the connection to the server are made by the help of Delphi6 components.The Client computer port is listening the 3306. port of the server.All transferrings are occured by the help of this single port.While constructing the program,the server must have a constant ip address like 127.0.0.1 and the client must take an ip address from the server.

As shown in Figure 5.1,to test the connnection the client is pinging to the server and the connection is ready to run but the user never see this ping statement,this will be done automaticly by the program.

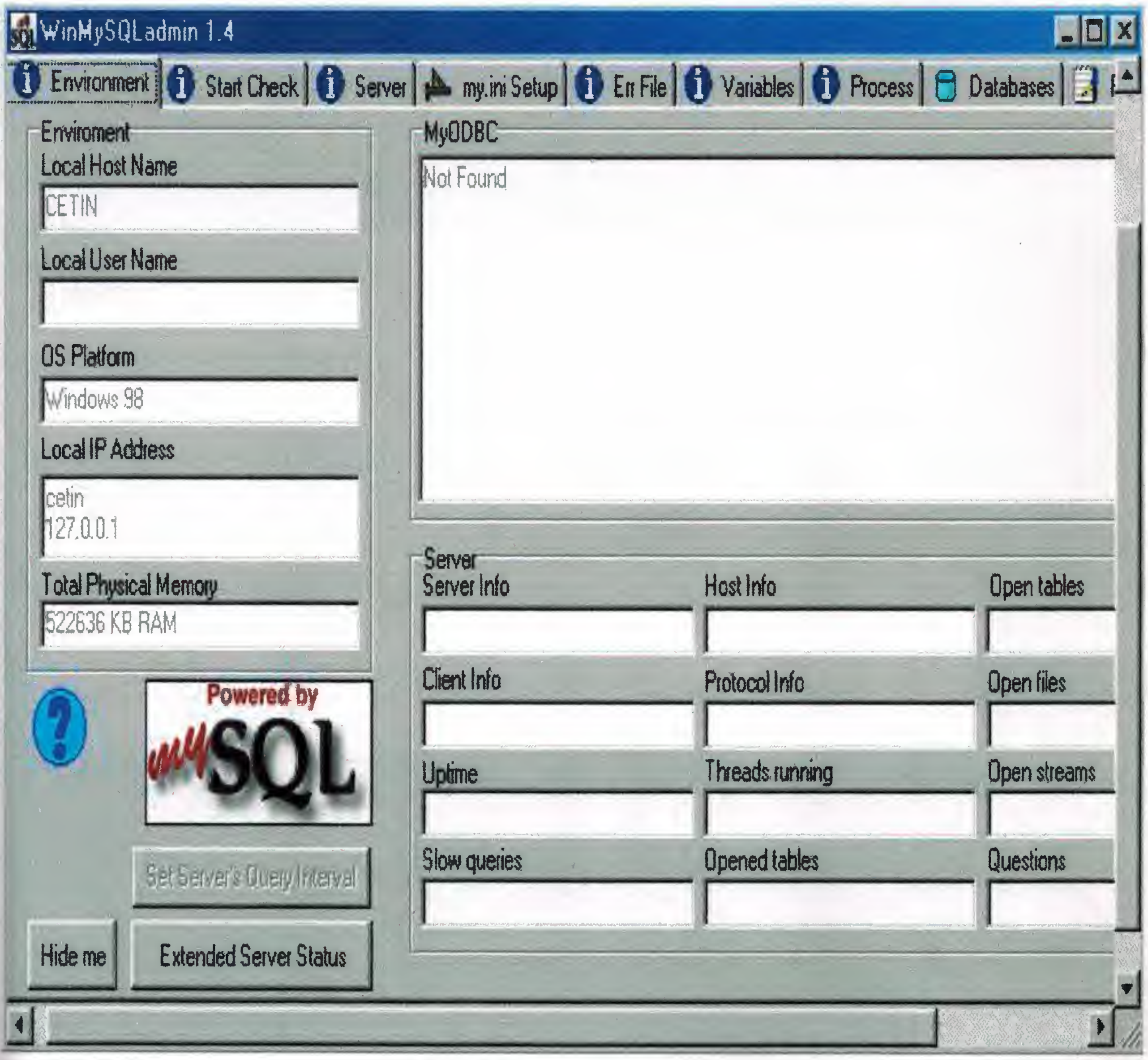


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\gurcan karagoz>ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\gurcan karagoz>
```

Figure 5.1 Command Prompt

With the logic of distributed systems,There can be more than one server or also clients may have their own databases,the aim is to reduce the clients work. All heavy information will be stored in the server and it can be called by the client in any part of the program.

As shown in figure 5.2,WinMySql Admin is running with the windows and all databases,server connections,ip configuration can be seen easily in this part.


WinMySQLadmin 1.4

Environment | Start Check | Server | my.ini Setup | Err File | Variables | Process | Databases

Enviroment

Local Host Name: CETIN

Local User Name:

OS Platform: Windows 98

Local IP Address: cetin 127.0.0.1

Total Physical Memory: 522636 KB RAM

MyODBC: Not Found

Server

Server Info	Host Info	Open tables
Client Info	Protocol Info	Open files
Uptime	Threads running	Open streams
Slow queries	Opened tables	Questions

Figure 5.2 WinMySql Admin

If WinMySql Admin works,The database administrator is ready to run MySql Manager,In this part the administrator can create Databases,Tables,Queries easily,shown in figure 5.3

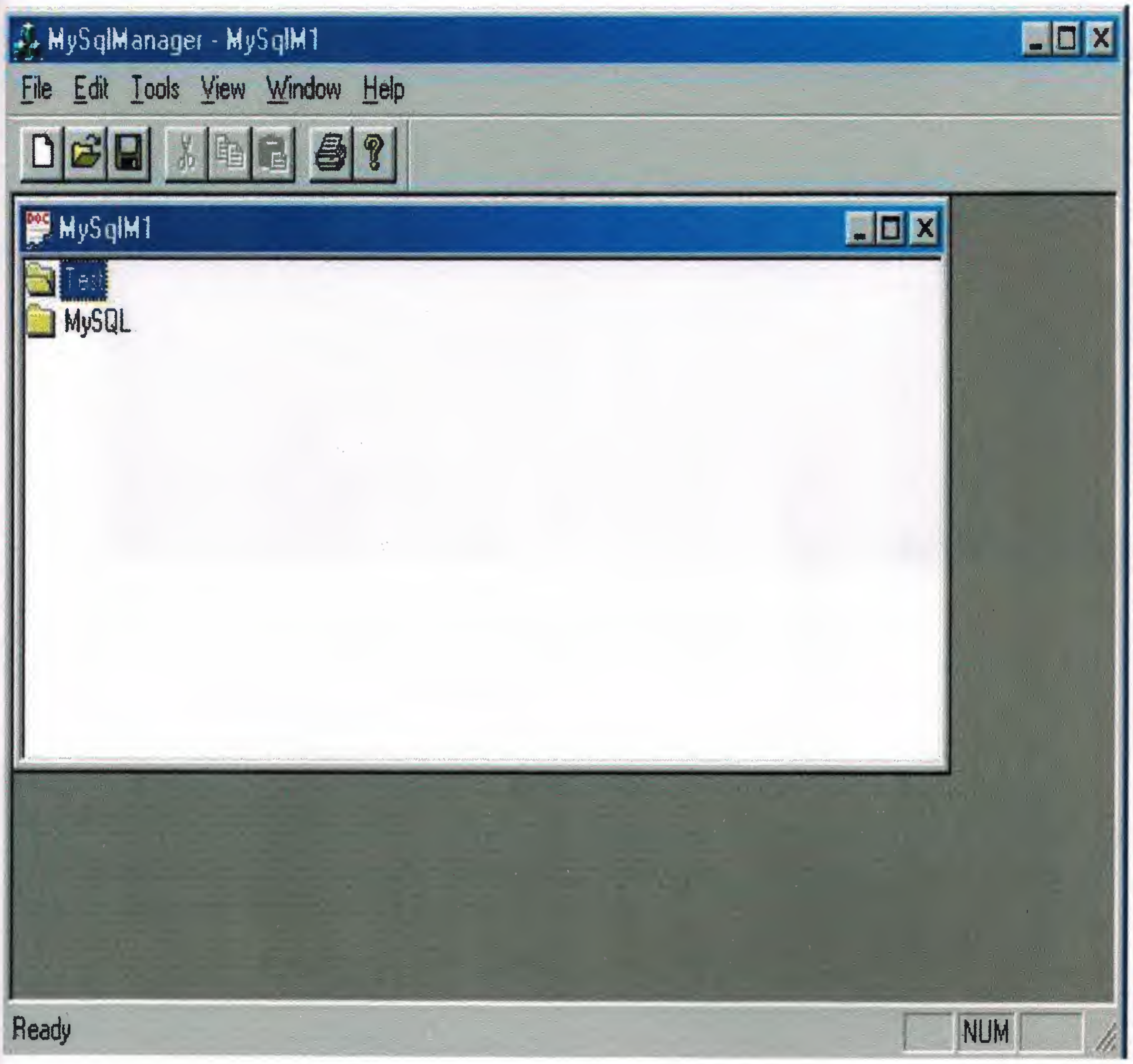
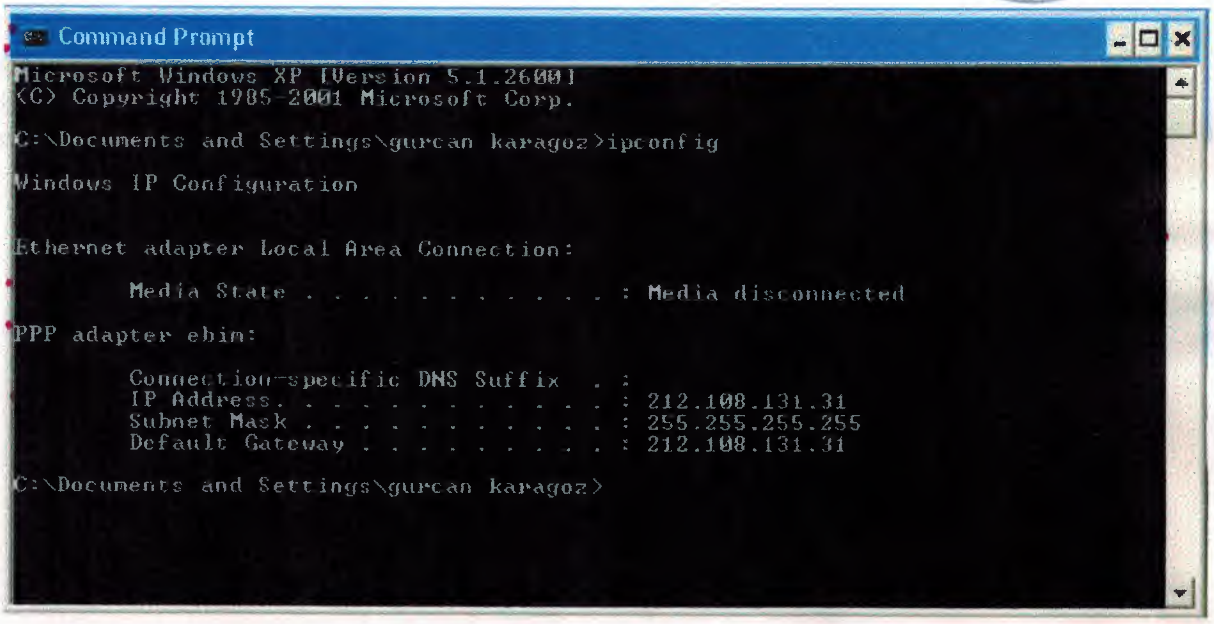


Figure 5.3 MySql Manager

Now the program is running,but it is the Local Area Network connection of the program.If the client wants to connect to the server from internet,The logic is again same,it has to known the ip adress of the server as shown in figure 5.4.The IP Address of server is 212.108.131.31 , so the program can run on internet.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\gurcan karagoz>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Media State . . . . . . . . . . . : Media disconnected

PPP adapter ebim:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 212.108.131.31
        Subnet Mask . . . . . . . . . . . : 255.255.255.255
        Default Gateway . . . . . . . . . : 212.108.131.31

C:\Documents and Settings\gurcan karagoz>
```

Figure 5.4 Internet Connection

In our program,We have three different Tables;Customer,Account,Personel.Using this system,all transaction occurs easily,fastly,reliably.The Client can transfer the tables which are allowed by the program administrator.

MySql was not found in the standard package of Delphi 5,but in Delphi 6 The Borland company starts using Mysql in their programs.

5.2 User Interface

Figure5.5 shows the screen when the program runs. As you see there are some menus. These are customer, accounts, search, money operations, personel, about and exit. At the begining of the program these menus, a digital clock and a calender which shows whole month when clicked, are appeared on screen..

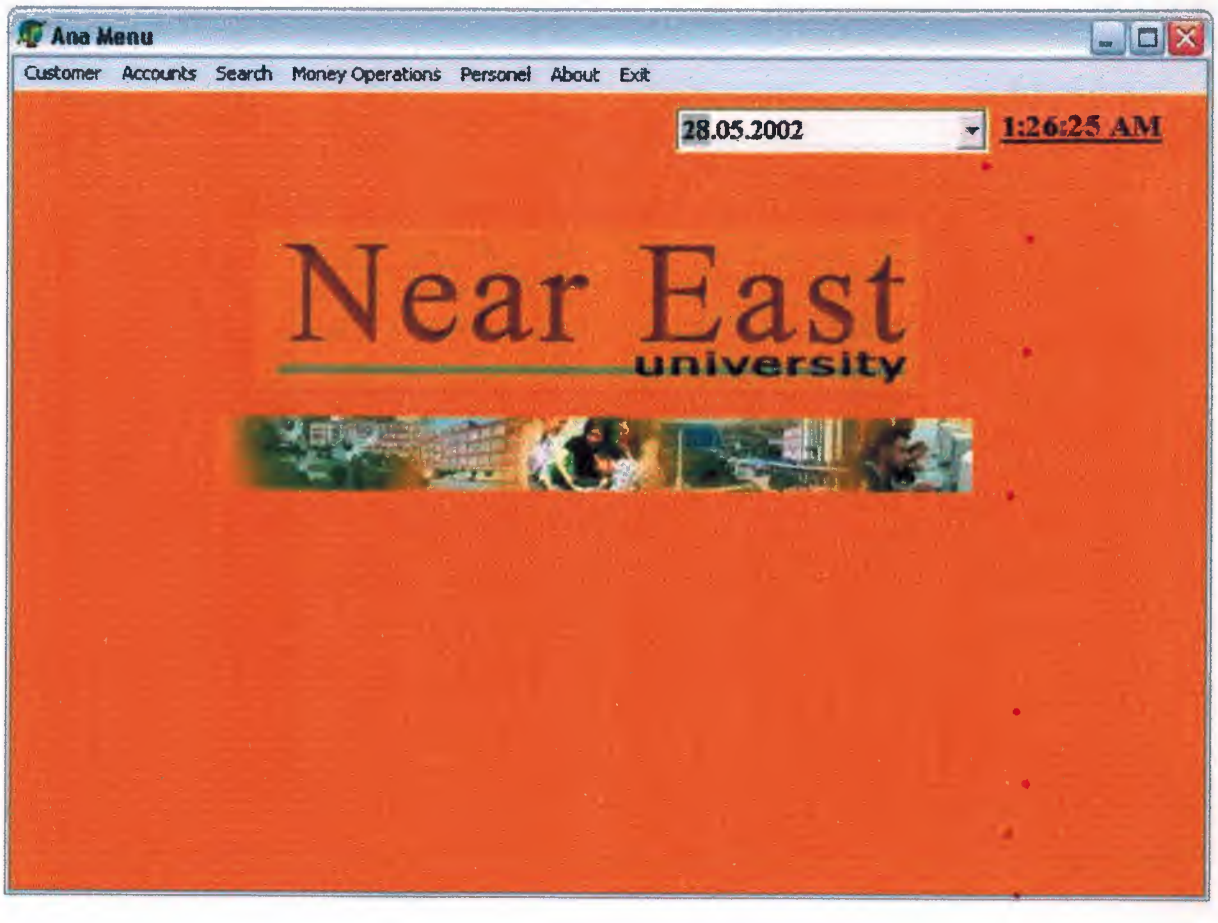
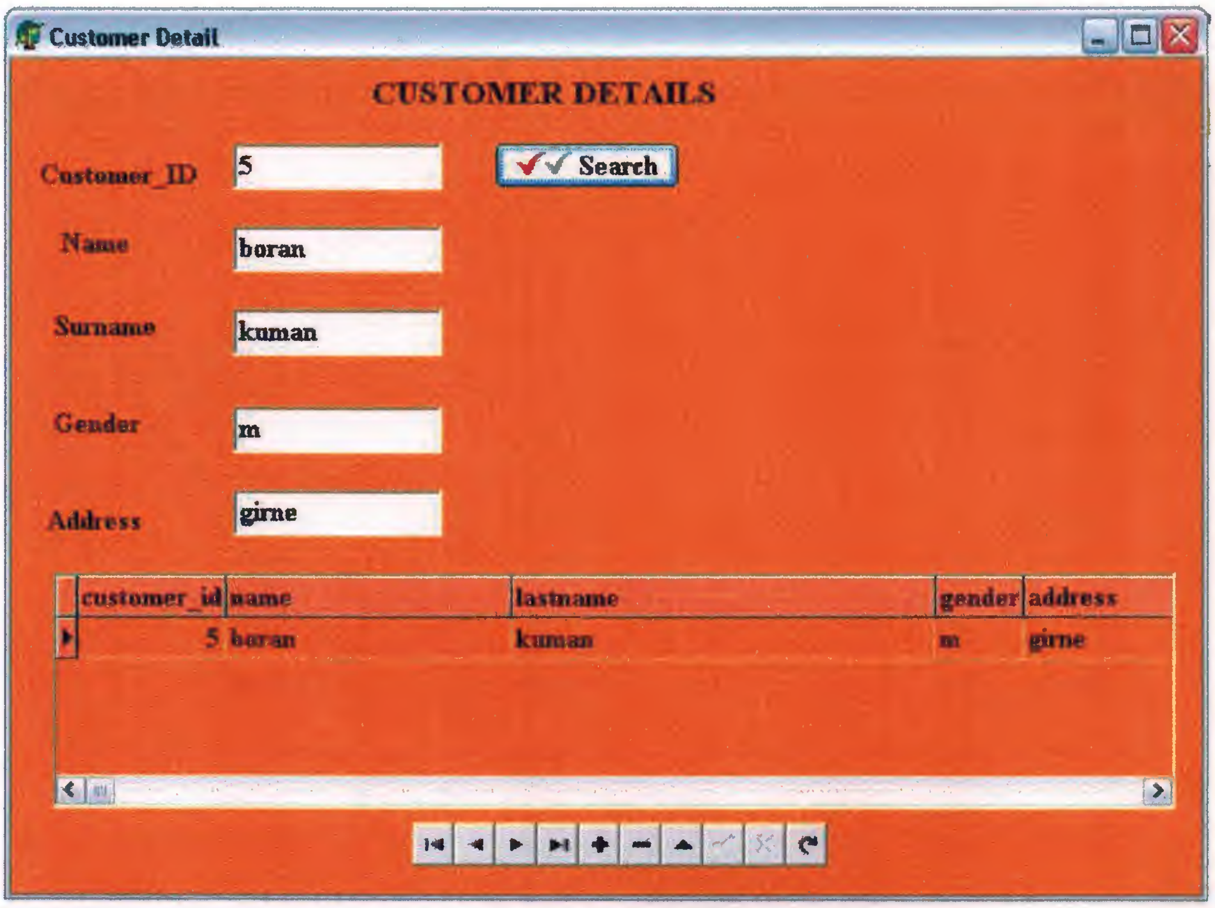


Figure 5.5 Main Form.

Figure 5.7 shows the screen when user clicks on the customer details. To monitor details of a customer you have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) details of customer comes to screen both to dbgrid and dbedit.

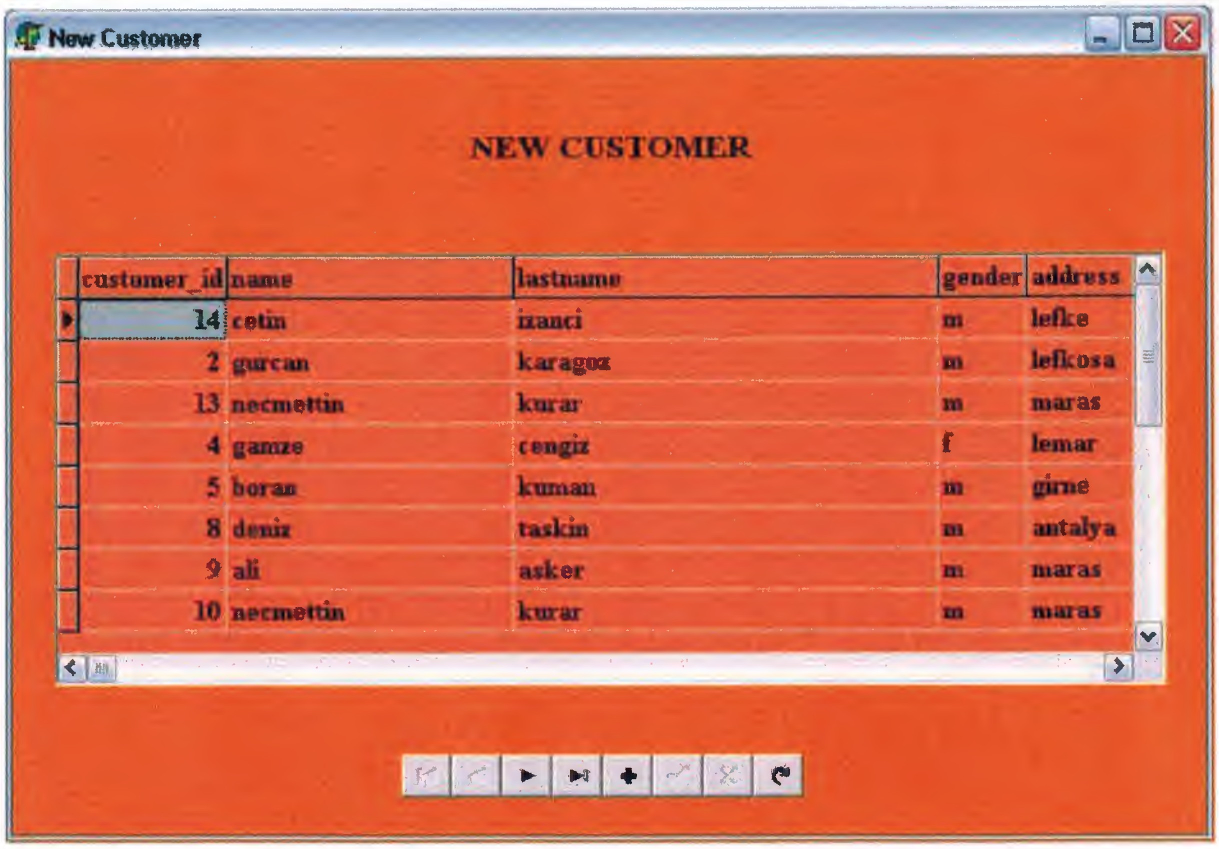


Customer_ID 5
Name boran
Surname kuman
Gender m
Address girne

customer_id	name	lastname	gender	address
5	boran	kuman	m	girne

Figure 5.7 Customer Details.

Figure 5.8 shows the screen when user clicks on new customer. User enters the information to necessary fields. Customer_id comes automaticly. İnorder to prevent customers having same id number, while we were creating our database we used an sql code that gives id automaticly and one after another. İf the last customer we entered has id=4 the next customer will have id=5 automaticly. After user finishes entering information, post button, which appears as check sign, must be clicked to save it into database.

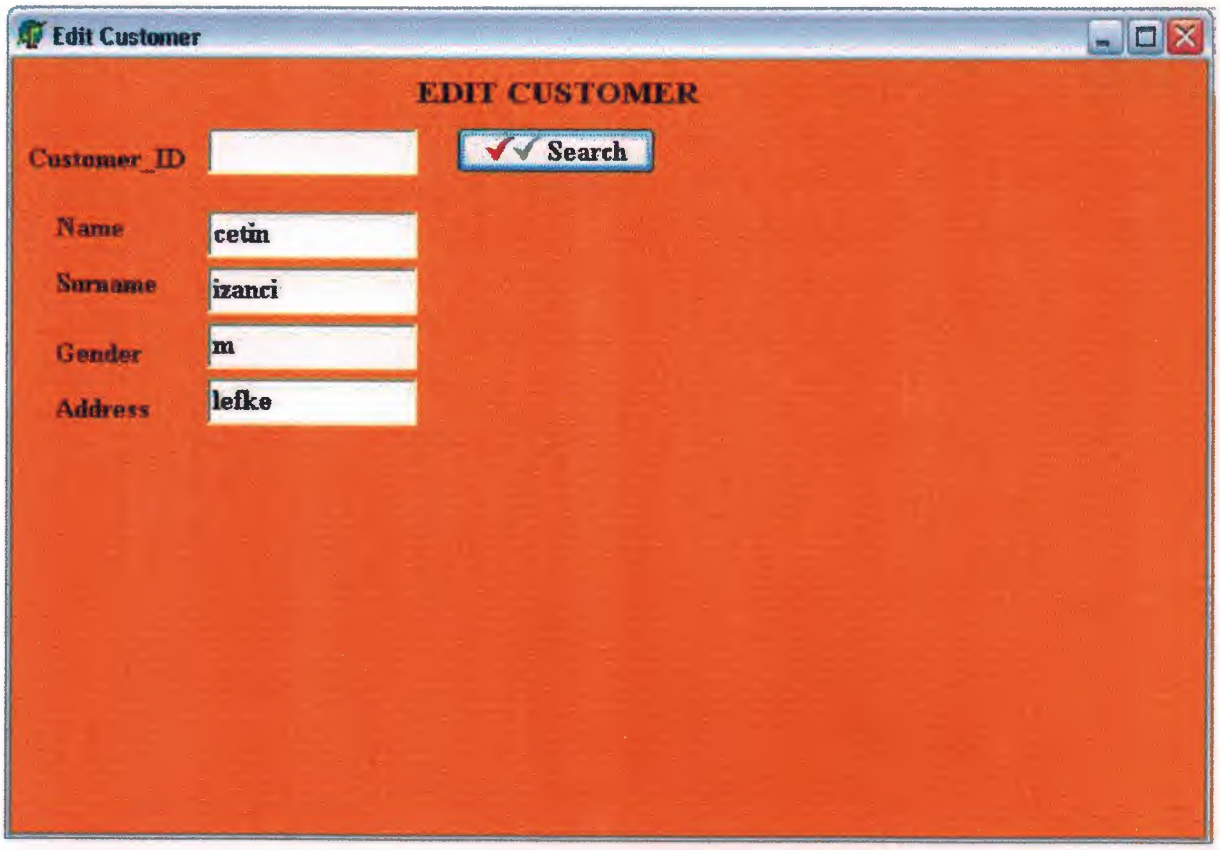


NEW CUSTOMER

customer_id	name	lastname	gender	address
14	cetin	izanci	m	lefke
2	gurcan	karagoz	m	lefkosa
13	necmettin	kurar	m	maras
4	gamze	cengiz	f	lemar
5	boran	kuman	m	girne
8	deniz	taskin	m	antalya
9	ali	asker	m	maras
10	necmettin	kurar	m	maras

Figure 5.8 New Customer.

Figure 5.9 shows screen when user clicks on edit customer. You have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) details of customer comes to screen. User makes necessary changes and then click post button which appears as a check sign. After post button is pressed changes are done and sent back to database with new(changed) information.



Customer_ID

Name cetin

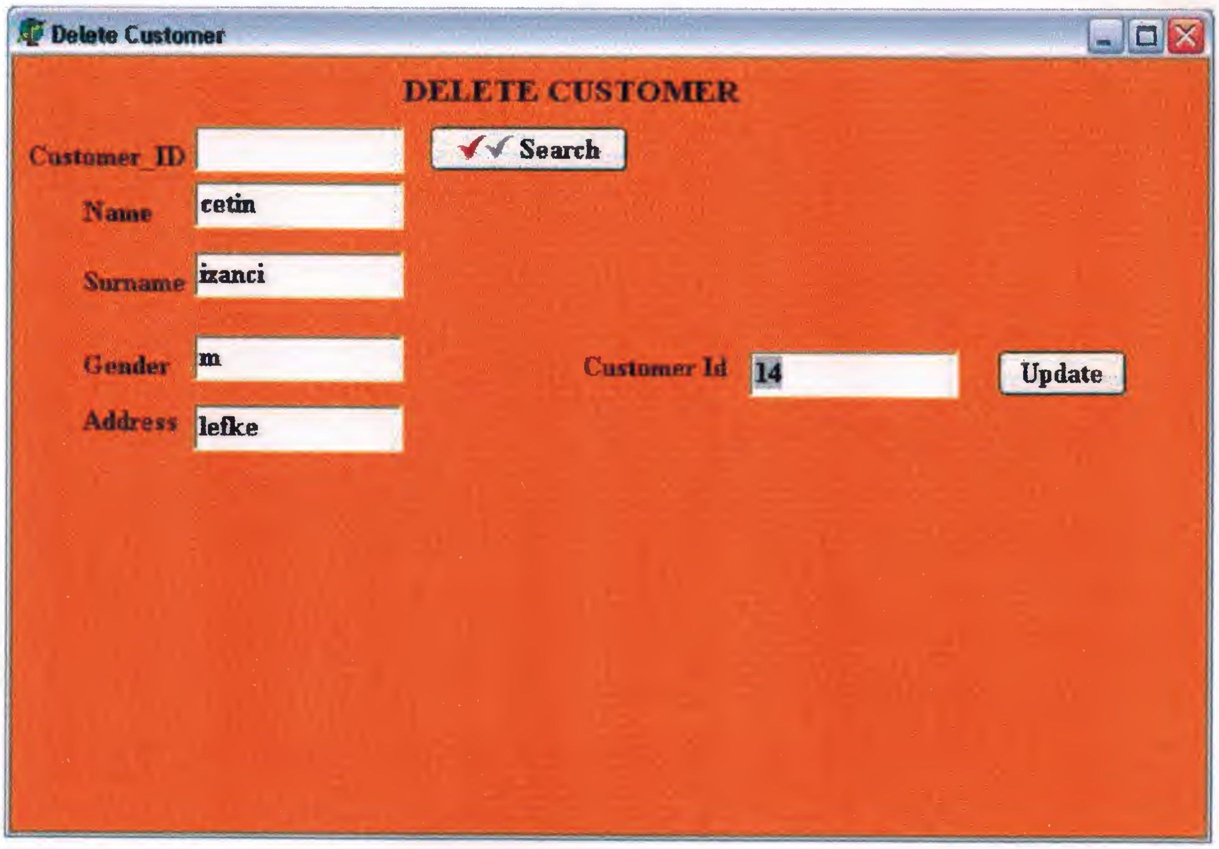
Surname izanci

Gender m

Address lefke

Figure 5.9 Edit Customer.

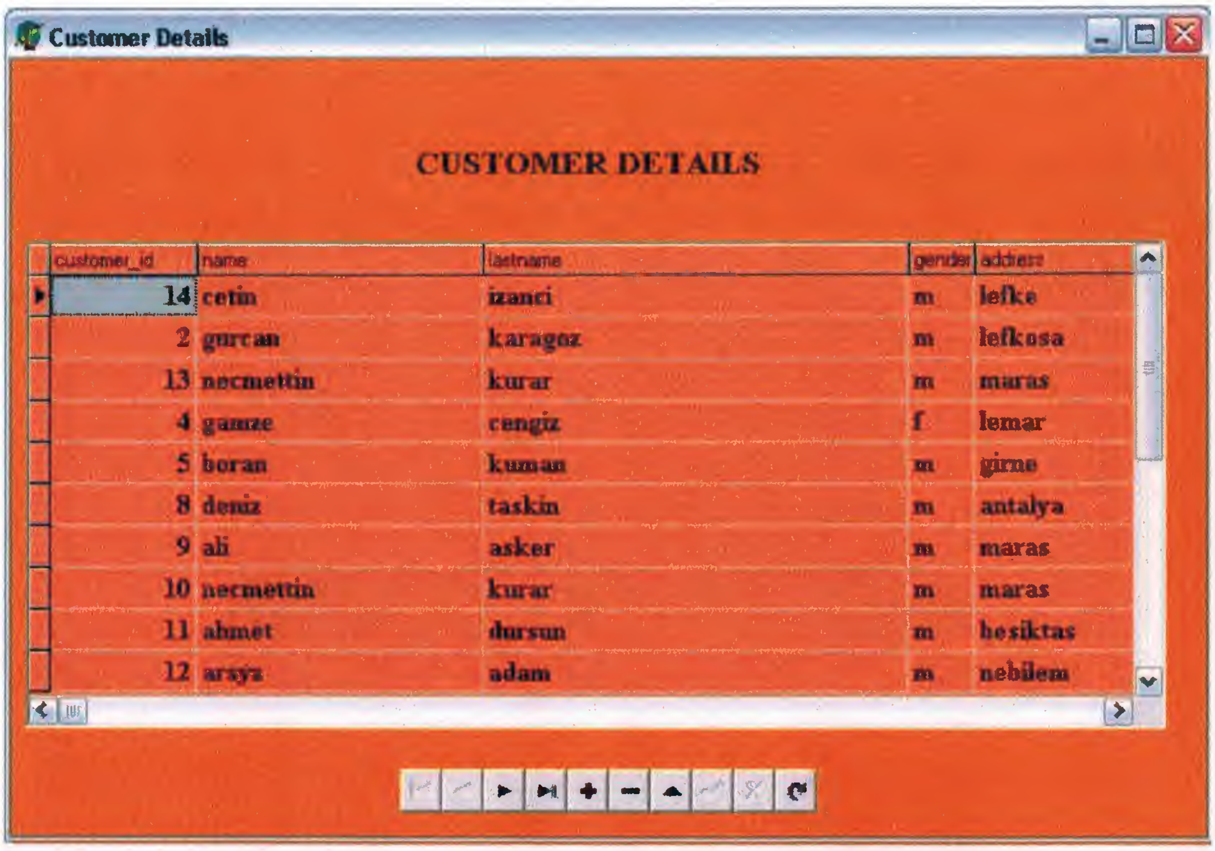
Figure 5.10 shows screen when user clicks on delete customer. You have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) details of customer comes to screen. İf the details belong to customer that you are going to delete write customer id to box which is next to update button and then press update button customer(details) and customer's accounts are going to be erased. İf a customer doesn't exist, how an account that belongs to him/her exists?



Customer_ID	
Name	cetin
Surname	izanci
Gender	m
Address	lefke
Customer Id	14

Figure 5.10 Delete Customer.

Figure 5.11 shows the screen when user clicks on customer list. Program connects to computer which has the database and takes information from the database and lists customers and details of the customers.



CUSTOMER DETAILS

customer_id	name	lastname	gender	address
14	cetin	izanci	m	lefke
2	gurcan	karagoz	m	lefkosa
13	necmettin	kurar	m	maras
4	gamze	cengiz	f	lemar
5	boran	kuman	m	girne
8	deniz	taskin	m	antalya
9	ali	asker	m	maras
10	necmettin	kurar	m	maras
11	ahmet	dursun	m	besiktas
12	arsya	adam	m	nebilem

Figure 5.11 Customer list.

Figure 5.12 shows the screen when user clicks on the accounts. User can perform some operations on accounts. These operations are, monitoring details of a specific account from database, adding a new account to database, editting(changing) details of a specific account which exists in database, deleting an account from the database, listing details of accounts that exists in our database. We will examine these operations in the following figures.

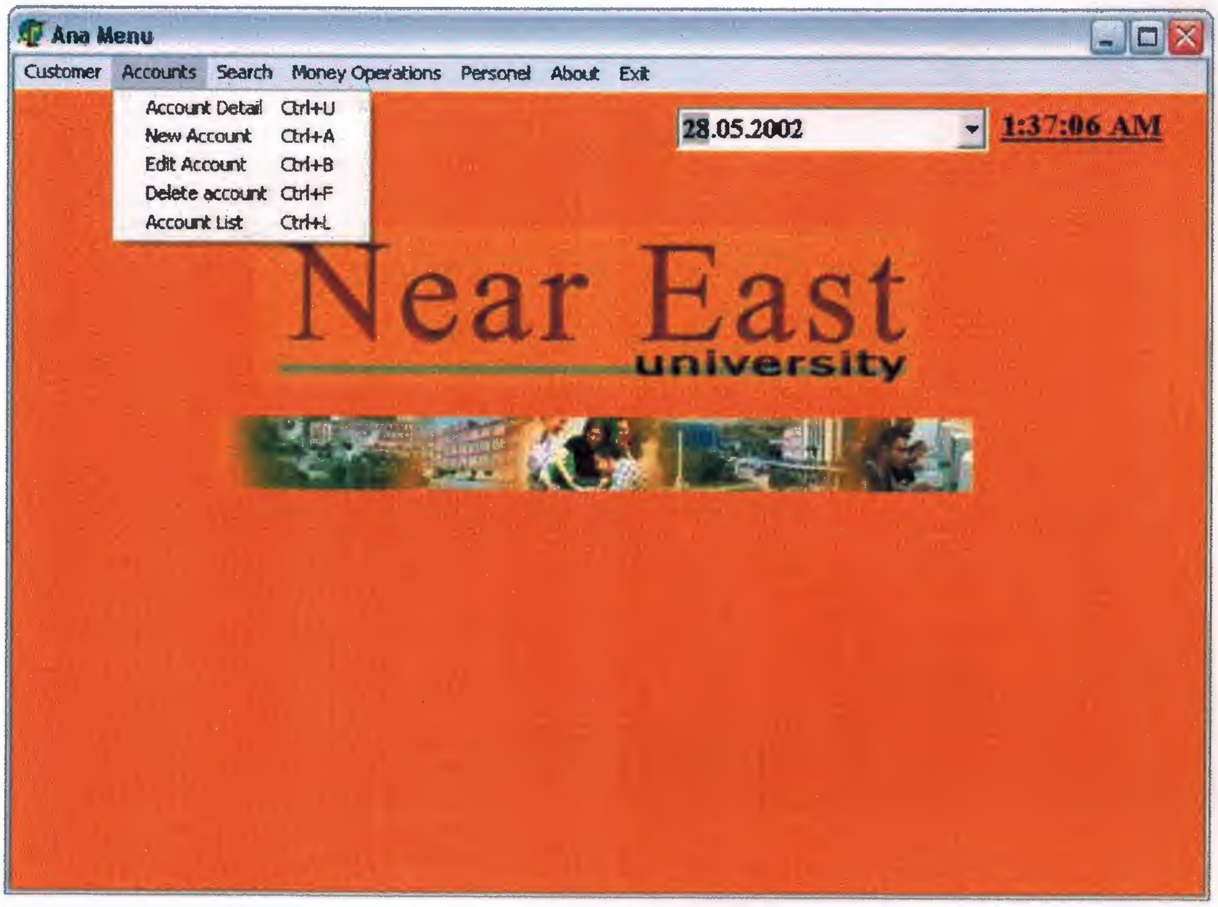
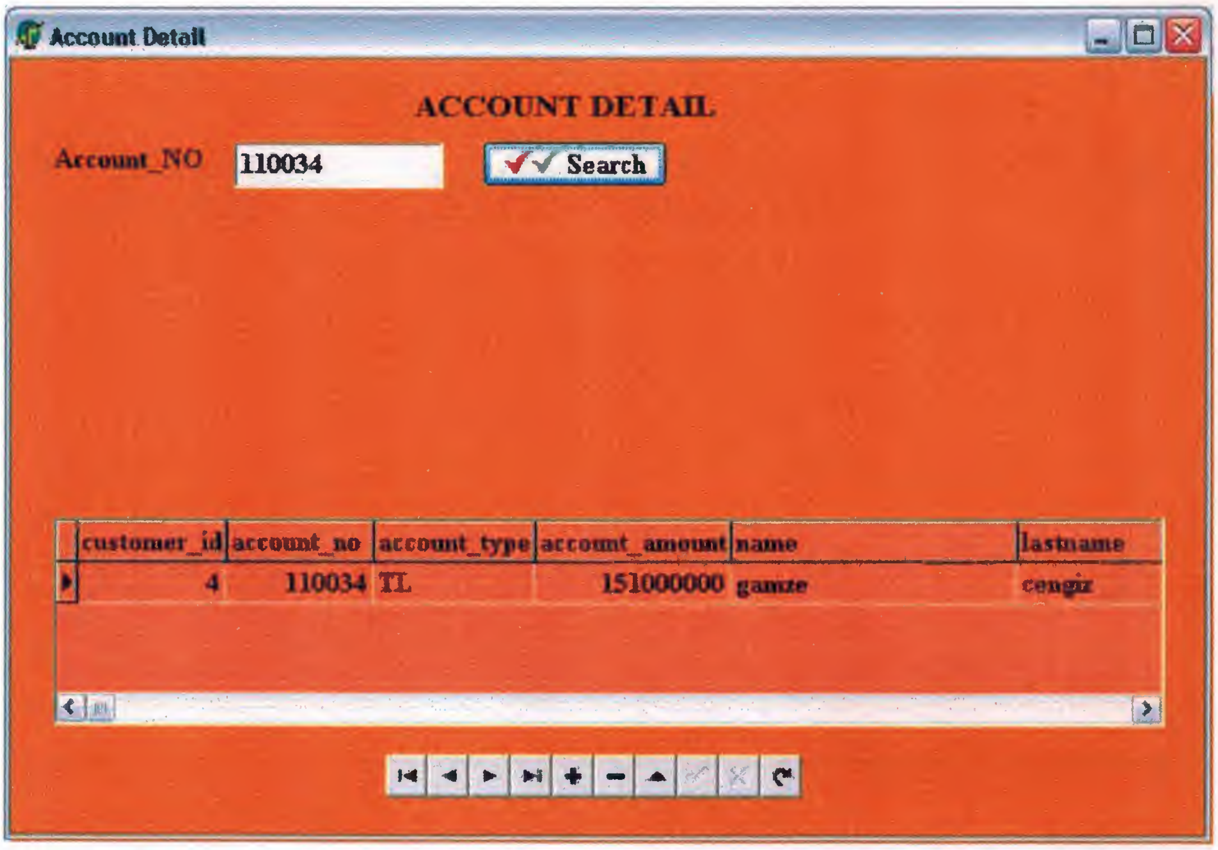


Figure 5.12 Accounts Menu.

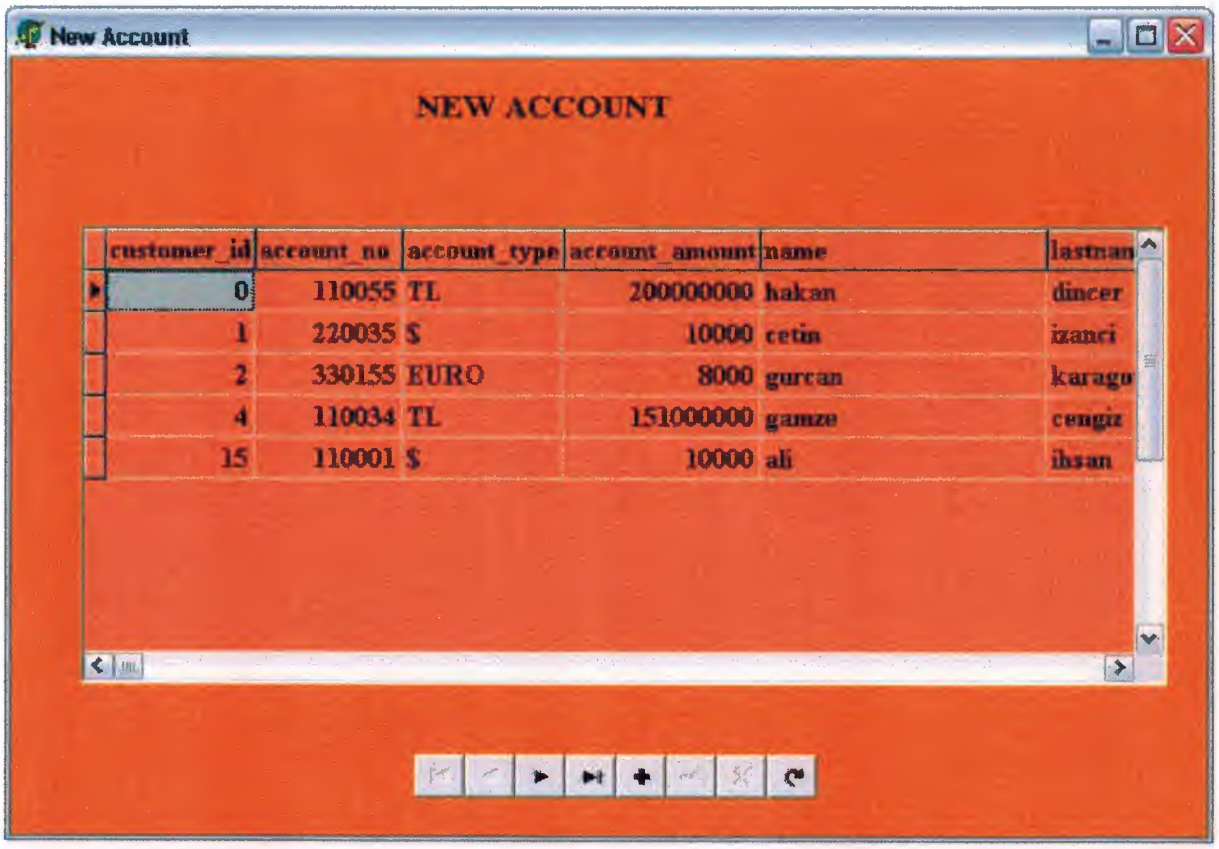
Figure 5.13 shows the screen when user clicks on the account detail. To monitor details of an account you have to enter account number of an account and then press search button. Program will connect to remote database and search for that number. İf search will be successful(if number found in database) details of account comes to screen both to dbgrid and dbedit.



customer_id	account_no	account_type	account_amount	name	lastname
4	110034	TL	151000000	gamze	cengiz

Figure 5.13 Account Details.

Figure 5.14 shows the screen when user clicks on new account. User enters the information to necessary fields. After user finishes entering information, post button, which appears as check sign, must be clicked to save it into database on remote computer.

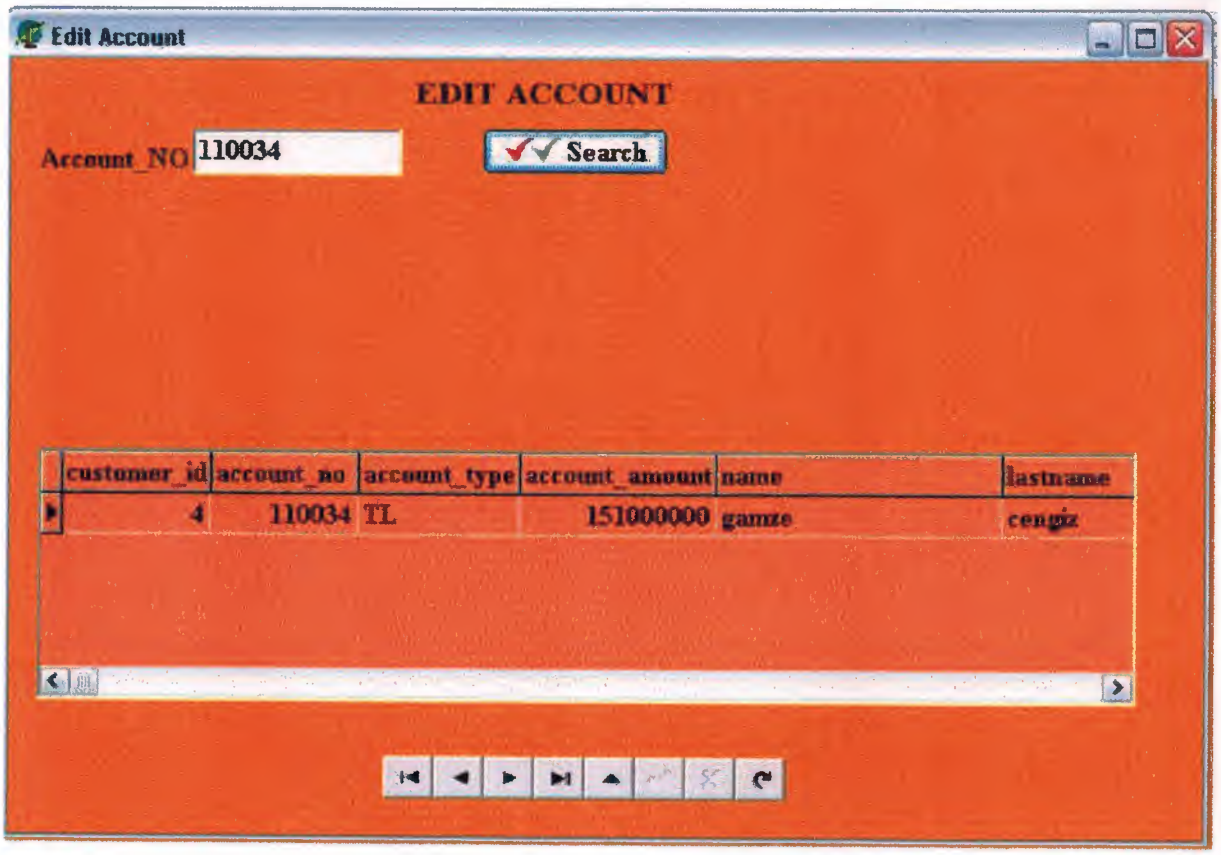


NEW ACCOUNT

customer_id	account_no	account_type	account_amount	name	lastnan
0	110055	TL	200000000	hakan	dincer
1	220035	$	10000	cetin	izanci
2	330155	EURO	8000	gurcan	karago
4	110034	TL	151000000	gamze	cengiz
15	110001	$	10000	ali	ihsan

Figure 5.14 New Account.

Figure 5.15 shows screen when user clicks on edit account. You have to enter number of an account and then press search button. Program will connect to remote database and search for that number. If search will be successful(if number found in database) details of account comes to screen. User makes necessary changes and then click post button which appears as a check sign. After post button is pressed changes are done and sent back to database with new(changed) information.

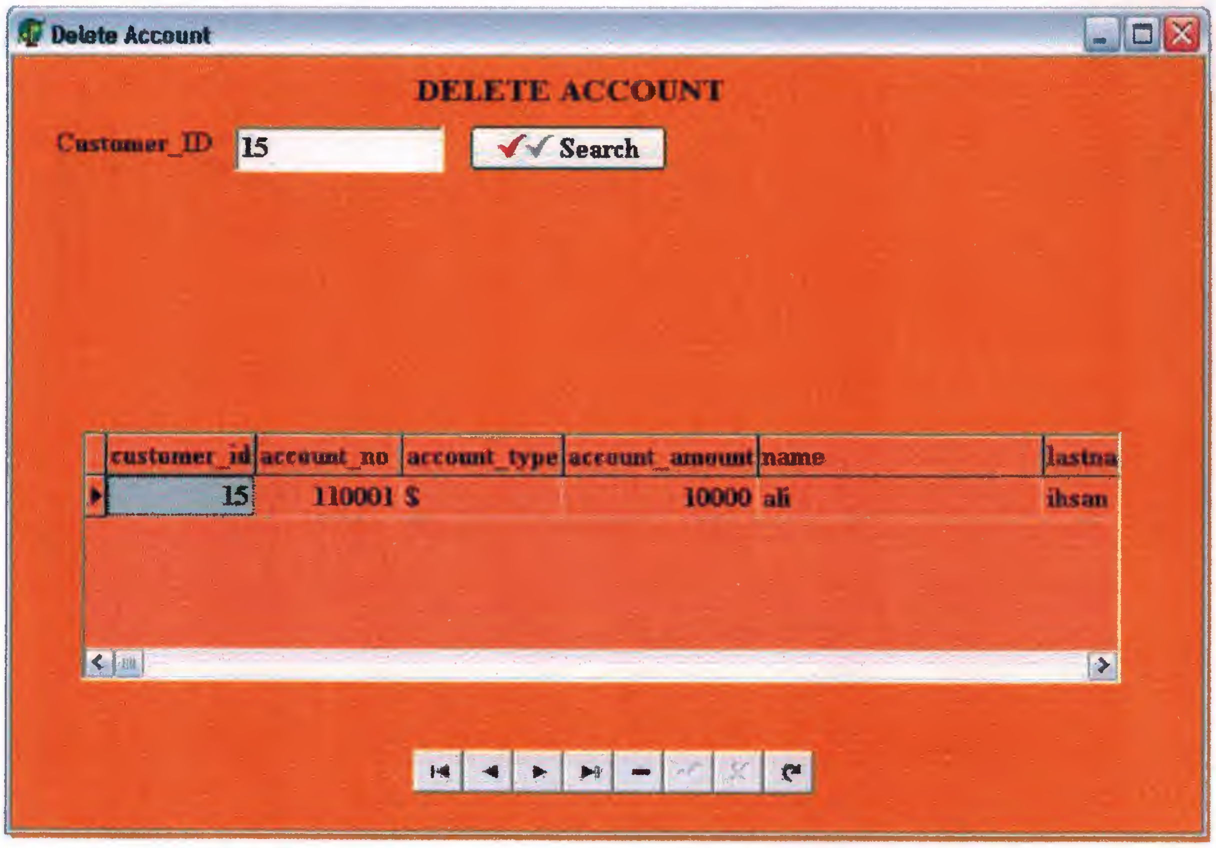


Account_NO 110034

customer_id	account_no	account_type	account_amount	name	lastname
4	110034	TL	151000000	gamze	cengiz

Figure 5.15 Edit Account.

Figure 5.16 shows screen when user clicks on delete account. You have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) account details of customer comes to screen. İf the details belong to account that you are going to delete just click delete button, which appears as minus sign below the details of account.

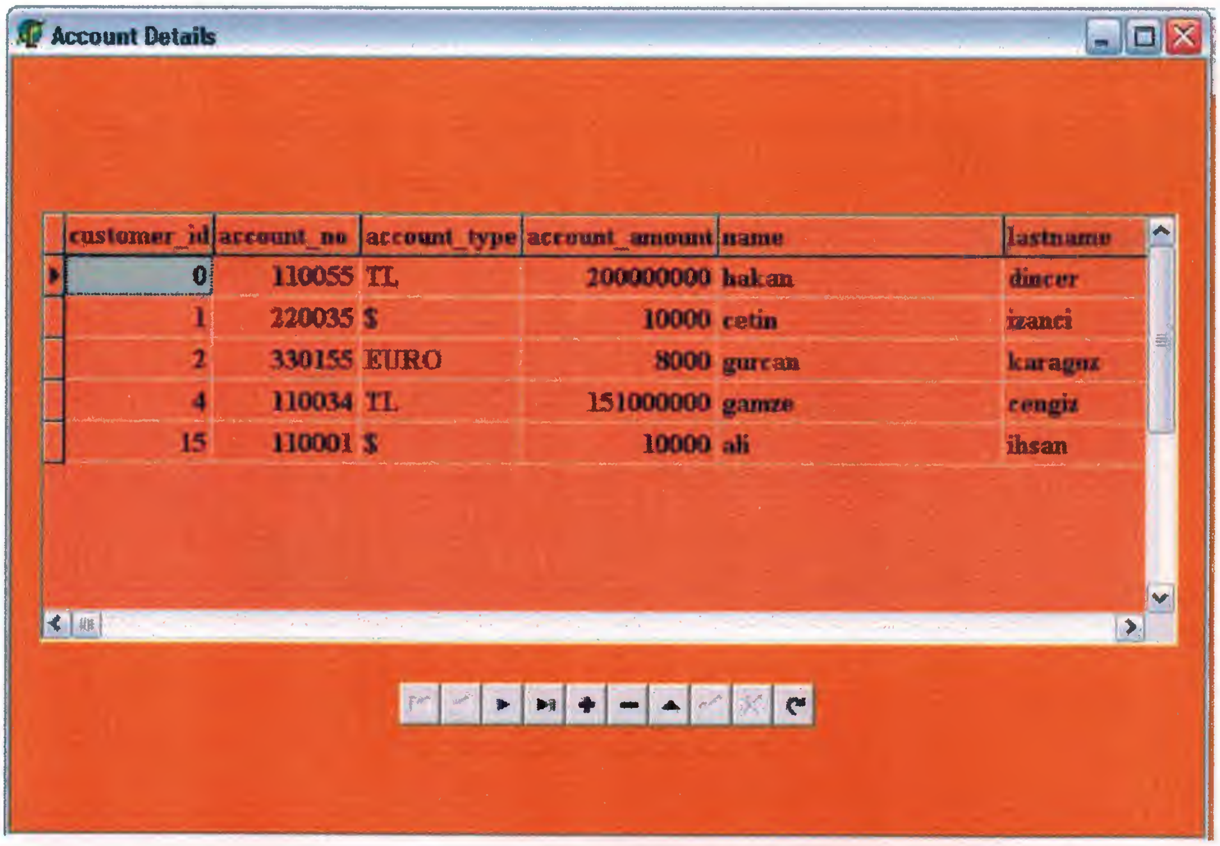


Customer_ID: 15

customer_id	account_no	account_type	account_amount	name	lastna
15	110001	$	10000	ali	ihsan

Figure 5.16 Delete Account.

Figure 5.17 shows the screen when user clicks on account list. Program connects to computer which has the database and takes information from the database and lists accounts and details of the accounts.



customer_id	account_no	account_type	account_amount	name	lastname
0	110055	TL	200000000	hakan	dincer
1	220035	$	10000	cetin	izanci
2	330155	EURO	8000	gurcan	karaguz
4	110034	TL	151000000	gamze	cengiz
15	110001	$	10000	ali	ihsan

Figure 5.17 Account List.

Figure 5.18 shows screen when user clicks search. There are 2 ways of searching. This figure shows the one that is done by using name or surname or address. You click one of the search property that you want to perform search operation. Write the key information(name,surname or address) to the box that will come to screen then click search button. Program will connect to remote database and search for that information. İf search will be successful(if information found in database) details of customer comes to screen.

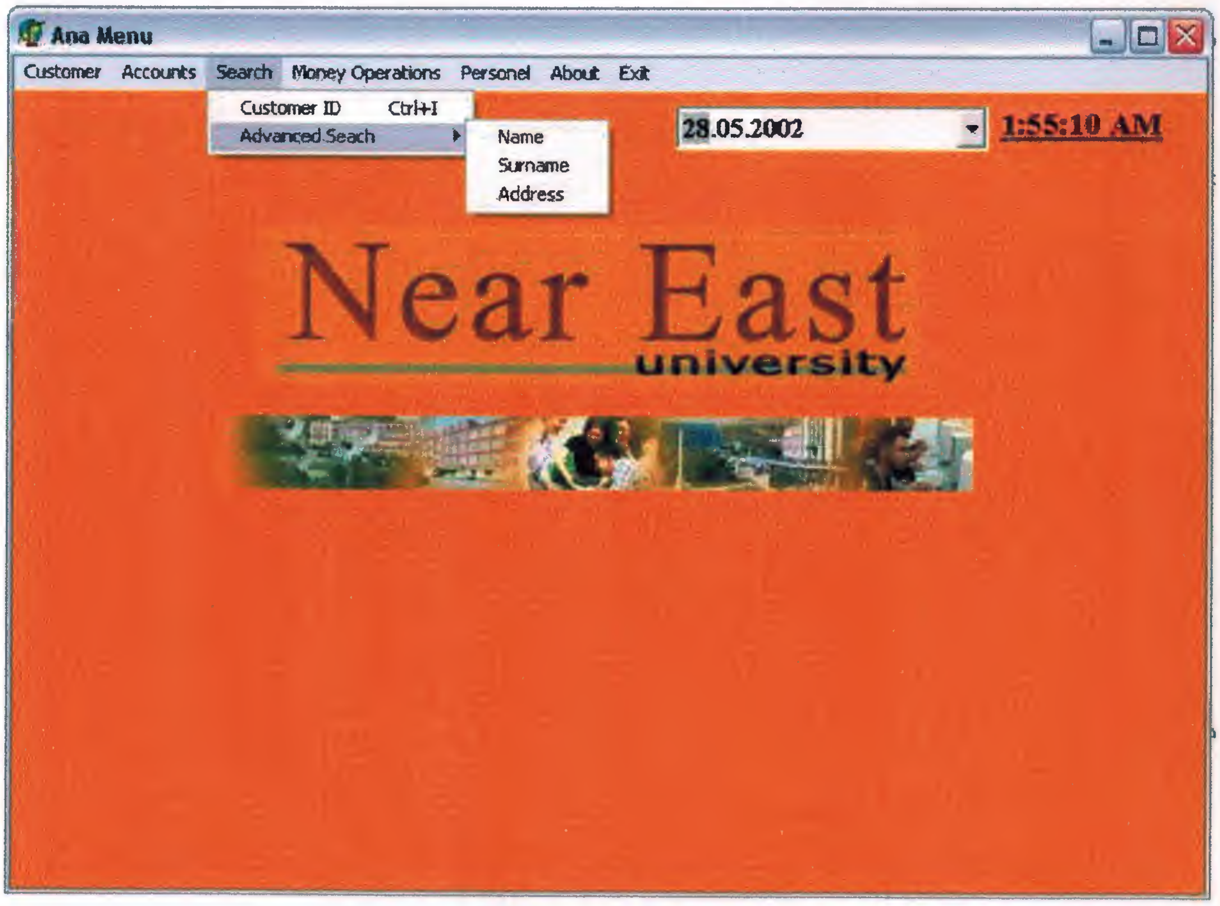
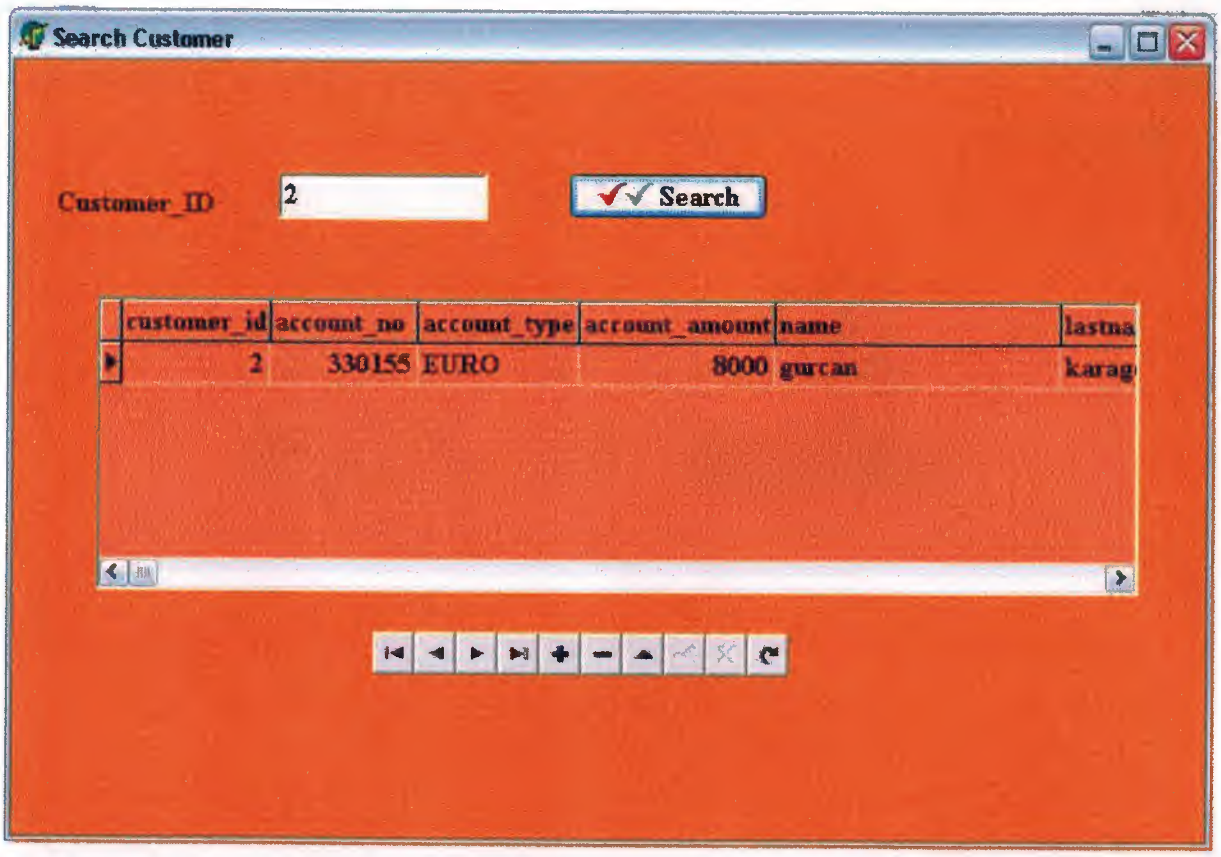


Figure 5.18 Search Menu.

Figure 5.19 shows the screen when user clicks on the customer id that pereforms a search by using customer id. To search a customer you have to enter id of customer and then press search button. Program will connect to remote database and search for that id. If search will be successful(if id found in database) details of customer comes to screen to dbgrid.



Customer_ID: 2

customer_id	account_no	account_type	account_amount	name	lastna
2	330155	EURO	8000	gurcan	karag

Figure 5.19 Search With customer_id.

Figure 5.20 shows the screen when user clicks on the money operations. User can perform some operations on customers' accounts. These operations are, drawing money, depositing money and money order that means transfering money. We will examine these operations in the following figures.

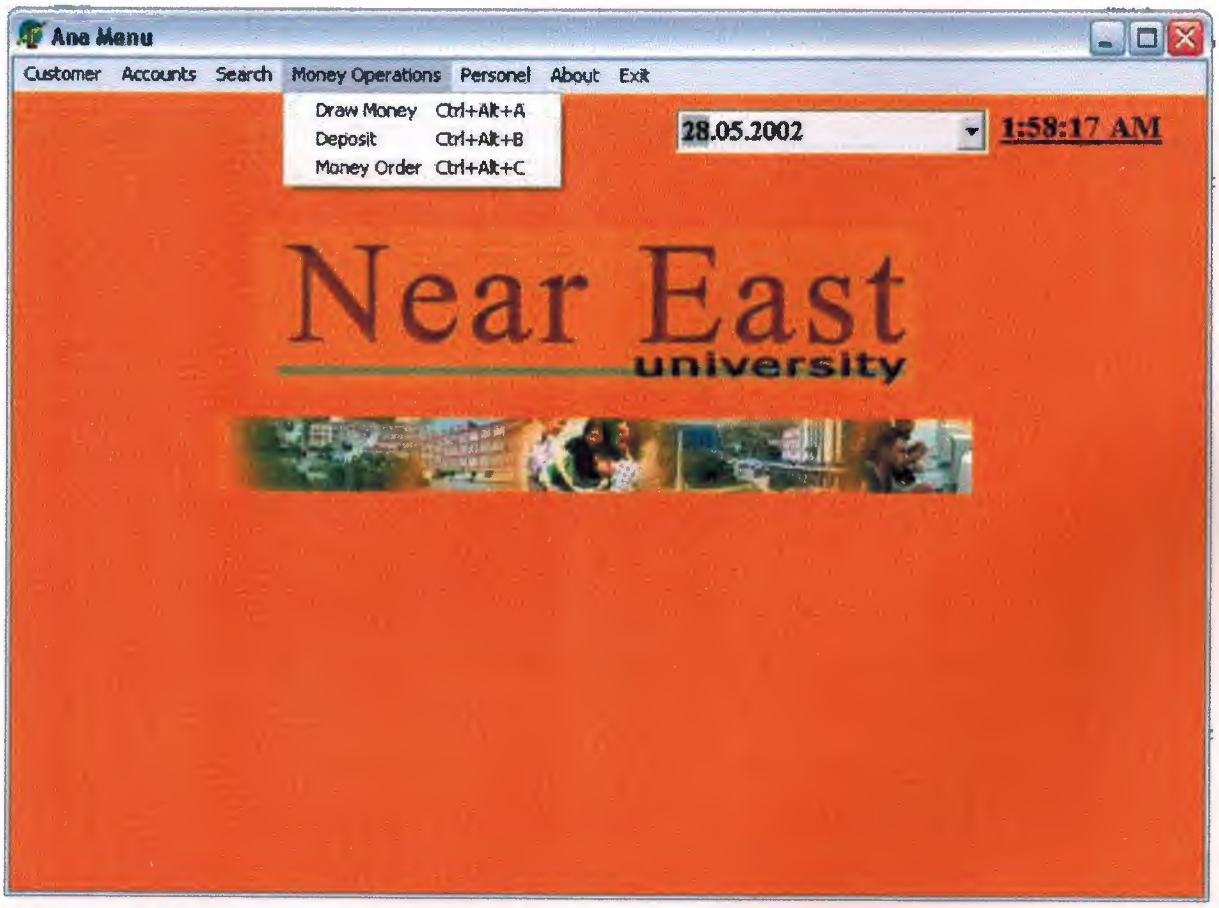
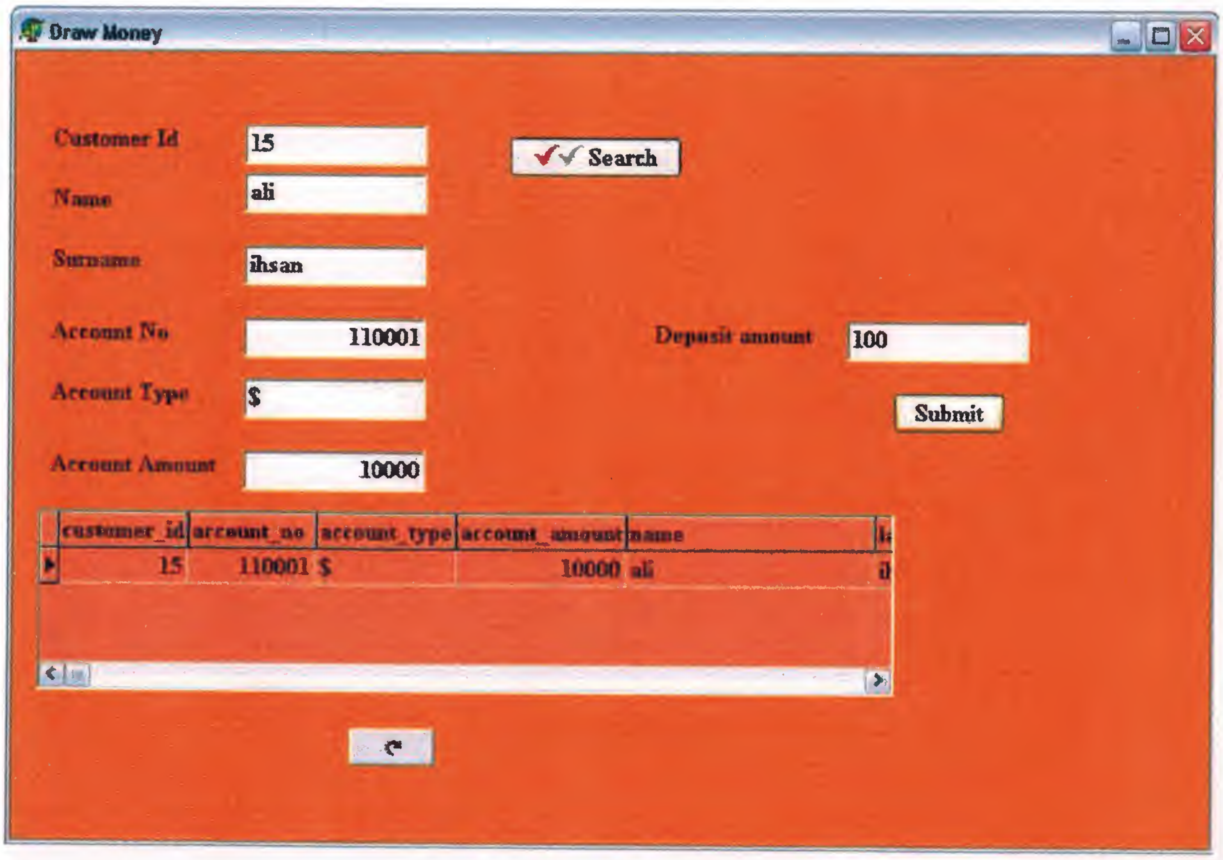


Figure 5.20 Money Operations.

Figure 5.21 shows the screen when user clicks on draw money. To draw money to an account of a customer you must perform a search first. You have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) details of customer account comes to screen both to dbgrid and dbedit. İf details belong to customer that is going to draw money, amount to be drawed is written to box at right side of label 'deposit amount' and then submit button is clicked. İf refresh button is clicked, change in amount can be seen while program is running...

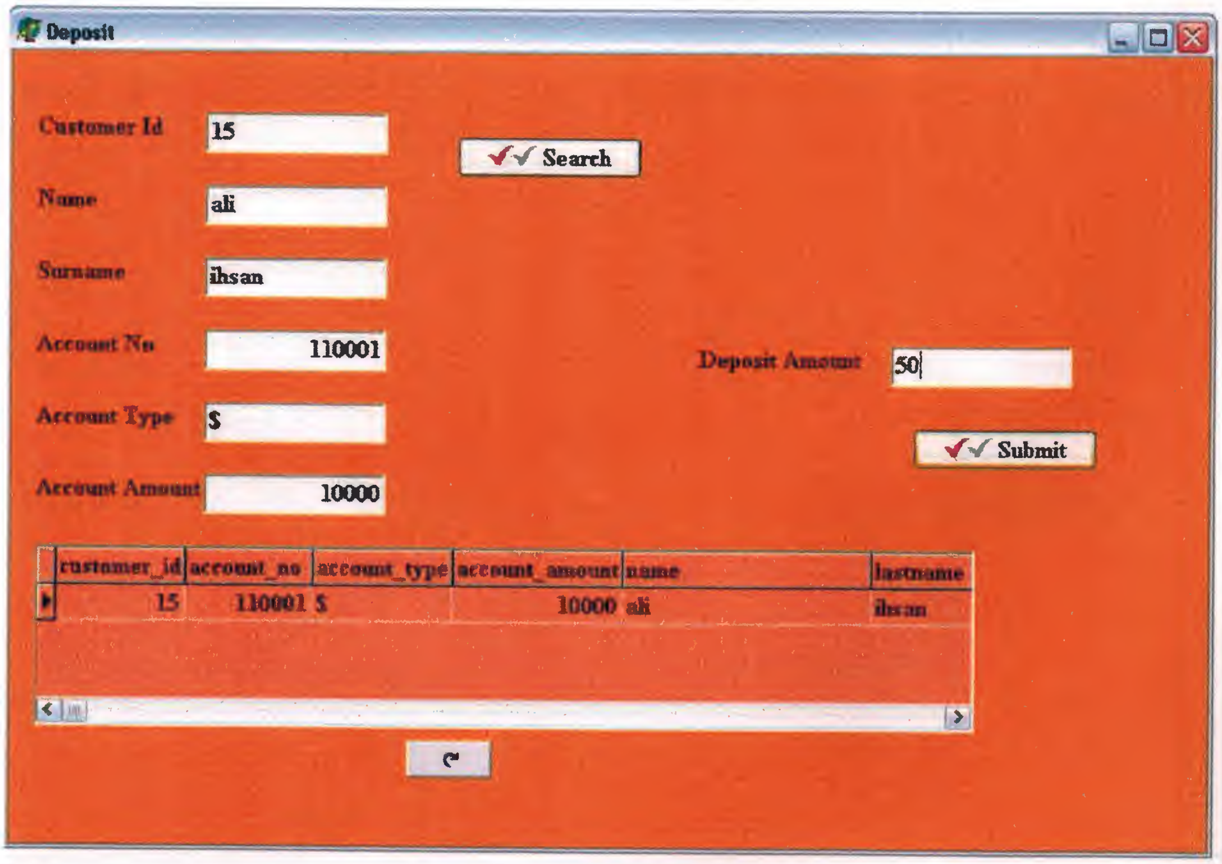


Customer Id	15
Name	ali
Surname	ihsan
Account No	110001
Account Type	$
Account Amount	10000
Deposit amount	100

customer_id	account_no	account_type	account_amount	name
15	110001	$	10000	ali

Figure 5.21 Draw Money.

Figure 5.22 shows the screen when user clicks on deposit money. To deposit money to an account of a customer you must perform a search first. You have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) details of customer account comes to screen both to dbgrid and dbedit. İf details belong to customer that is going to deposit money, amount to be deposited is written to box at right side of label 'deposit amount' and then submit button is clicked. İf refresh button is clicked, change in amount can be seen while program is running...

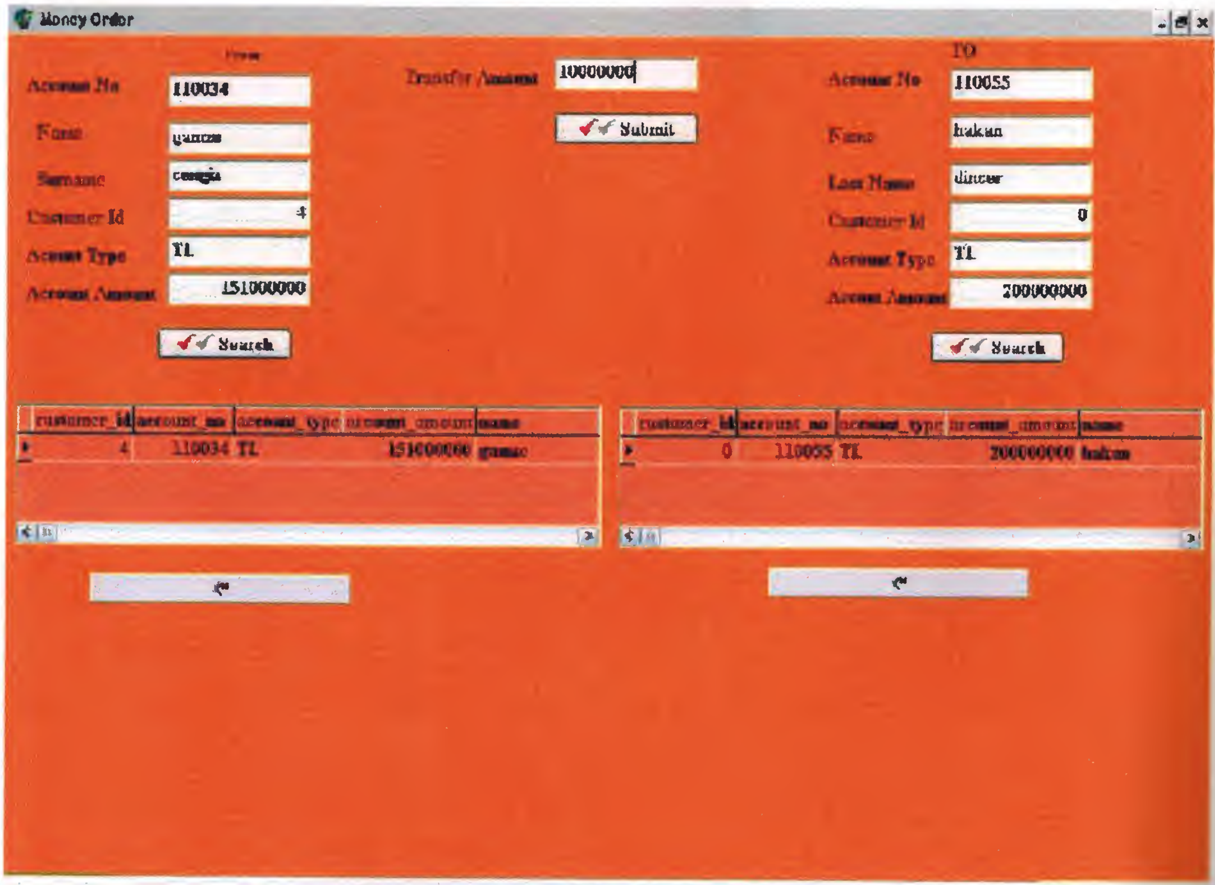


Customer Id	15
Name	ali
Surname	ihsan
Account No	110001
Account Type	$
Account Amount	10000
Deposit Amount	50

customer_id	account_no	account_type	account_amount	name	lastname
15	110001	$	10000	ali	ihsan

Figure 5.22 Deposit Money.

Figure 5.23 shows the screen when money order is clicked. Two seperate searches are performed to see accounts's details to transfer and receive money. You have to enter id of customer and then press search button. Program will connect to remote database and search for that id. İf search will be successful(if id found in database) details of customer account comes to screen both to dbgrid and dbedit. Details at left belongs to sender of the money and details at right belongs to receiver of the money. İf correct accounts are found, user writes amount to be transferred and presses submit button. İf monetary units are same transfer will occur and amount will be subtracted from sender and will be added to receiver. After refresh buttons are pressed changes will be seen in list while program is running. İf monetary units are not same transfer will not occur..



Transfer Amount	10000000

From

Account No	110034
Name	gamze
Surname	cengiz
Customer Id	4
Account Type	TL
Account Amount	151000000

TO

Account No	110055
Name	hakan
Last Name	dincer
Customer Id	0
Account Type	TL
Account Amount	200000000

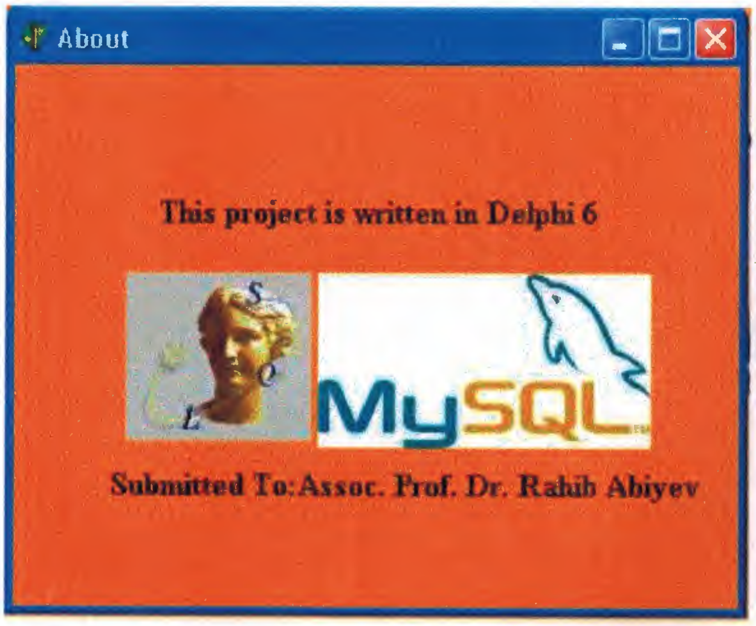
customer_id	account_no	account_type	account_amount	name
4	110034	TL	151000000	gamze

customer_id	account_no	account_type	account_amount	name
0	110055	TL	200000000	hakan

Figure 5.23 Money Transfer.

Figure 5.24 shows screen when user clicks about project. Language we used to write this project(delphi), symbols of delphi and MySQL and name of instructor that we are going to submit project come on screen.

Figure 5.24 About.



This project is written in Delphi 6

Submitted To:Assoc. Prof. Dr. Rahib Abiyev

CONCLUSION

The most important question for the database programmers "How can tables carry large amount of data and How fast the transaction occurs?". Nowadays by the fast development of technologies, the data amount becomes larger and inversely proportional the transaction becomes slower, To overcome this problem programmers develop new and fast database servers; Nowadays one of the fastest database program is MySql server.

We develop our project by using MySql so that it is very fast that it can transfer data one megabyte per second from server to client and can hold large amount of data up to eight million terabytes, it is very reliable and can operate in any operating system such as Linux, Solaris, Windows.

The project is developed in Borland Delphi 6 and components are used to link database to the Delphi. The program can run on two pc's up to sixteen pc's reliably and fastly over Internet, Local Area Network(LAN) and Wide Area Network(WAN).

REFERENCES

1.Marco Cantu,"Mastering Delphi 6",Sybex 2001.

2. Mark Maslakowski, "Sam's Teach Yourself MySQL in 21 Days",

3. Steve Teixeira and Xavier Pacheco," Delphi™ 6 Developer's Guide, Copyright © 2002 by Sams Publishing

4.MySql Reference Manual,Copyright 1997-2001.

5. T,Ozsu, P.Valduriez, "Distributed and Parallel Database Systems", ACM Computing Surveys, vol.28, no.1, pp 125-128, March 1996.

6. T,Ozsu, P.Valduriez, "Distributed Data Management: Unsolved Problems and New Issues",Readings in Distributed Computing Systems, IEEE Computer Society Press, pp 512-544,1994.

7. A.Silberschatz, S.Zdonik, et.al., "Strategic Directions in Database Systems - Breaking Out of the Box", ACM Computing Surveys, vol.28, no.4, pp.764-778, Dec. 1996.

8. V.Kumar, M.Hsu, *Recovery Mechanisms in Database Systems*, Prentice Hall PTR, Prentice-Hall, Inc. 1998.

9. M. Stonebraker, "Future Trends in Database Systems", in *Multidatabase Systems*, IEEE Computer Society Press, pp 339-350, 1994.

10. T,Ozsu, P.Valduriez, *Principles of Distributed Database Systems*, Prentice-Hall, Englewod Cliffs, N.J., 1991.

11.Kevin Hough, Sql Server Database Design Study Guide, Microsoft Certified Professional Approved Study Guide.1997.