

NEAR EAST UNIVERSITY

**GRADUATE SCHOOL OF APPLIED
AND SOCIAL SCIENCES**

**Client Server Application For Hospital Management
System Using Centralized Database**

VOLKAN BABAOĞLU

Master Thesis

Department Of Computer Engineering

Nicosia - 2004

NEU

JURY REPORT

**DEPARTMENT OF
COMPUTER ENGINEERING**

Academic Year: 2003-2004

STUDENT INFORMATION

Full Name	Volkan Babaoğlu		
Undergraduate degree	BSc.	Date Received	Spring 1999-2000
University	Cumhuriyet University	CGPA	2.58



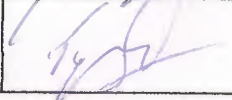
THESIS

Title	Client\Server Application for Hospital Management System using Centralized Database		
Description	The aim of this thesis is to develop a client\server based software for implementation of Hospital activities		
Supervisor	Assist.Prof.Dr. Firudin Muradov	Department	Computer Engineering

DECISION OF EXAMINING COMMITTEE

The jury has decided to accept / reject the student's thesis.
The decision was taken unanimously / by majority.

COMMITTEE MEMBERS

Number Attending	3	Date	01/04/2004
Name		Signature	
Assist. Prof. Dr. Doğan Haktanır, Chairman of the jury			
Assoc. Prof. Dr. Rahib Abiyev, Member			
Assoc. Prof. Dr. İlham Huseynov, Member			

APPROVALS

Date 01/04/2004	Chairman of Department Assoc. Prof. Dr. Doğan İbrahim 
---------------------------	--

DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENTAL DECISION

Date:01/04/2004

Subject: Completion of M.Sc. Thesis

Participants: Assist. Prof. Dr. Doğan Haktanır, Assoc. Prof. Dr. Rahib Abiyev,
Assoc.Prof. Dr. İlham Huseynov, Assist. Prof. Dr. Firudin Muradov.

DECISION

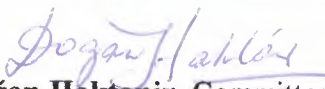
We certify that the student whose number and name are given below, has fulfilled all the requirements for a M .S. degree in Computer Engineering.

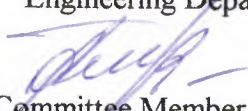
CGPA

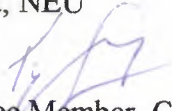
20002041


Volkan Babaoğlu

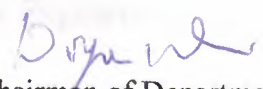
3.5


Assist. Prof. Dr. Doğan Haktanır, Committee Chairman, Electrical and Electronic
Engineering Department, NEU


Assoc. Prof. Dr. Rahib Abiyev, Committee Member, Computer Engineering
Department, NEU


Assoc. Prof. Dr. İlham Huseynov, Committee Member, Computer Information System
Department, NEU


Assist. Prof. Dr. Firudin Muradov, Supervisor, Computer Engineering
Department, NEU


Chairman of Department
Assoc. Prof. Dr. Doğan İbrahim

Volkan Babaoğlu : Client/Server Application for Hospital Management System
using Centralized Database

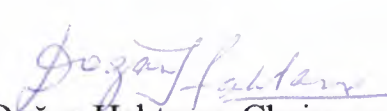
**Approval of the Graduate School of Applied and
Social Sciences**


Prof. Dr. Fakhraddin Mamedov
Director

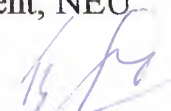


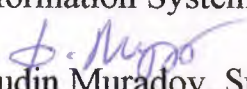
**We certify this thesis is satisfactory for the award of the
Degree of Master of Science in Computer Engineering**

Examining Committee in charge:


Assist. Prof. Dr. Doğan Haktanır, Chairman , Electrical and
Electronic Engineering Department, NEU


Assoc. Prof. Dr. Rahib Abiyev, Member , Computer Engineering
Department, NEU


Assoc. Prof. Dr. İlham Hüseynov, Member, Computer
Information System Department, NEU


Assist. Prof. Dr. Firudin Muradov, Supervisor, Computer Engineering
Department, NEU

ACKNOWLEDGEMENTS

First of all, I would like to extend my sincere and humble gratitude to Almighty Allah who endowed me potential and ability to make solid contribution to the already existing ocean of knowledge.

I wish to express my gratitude and sincere appreciation to my supervisor Assist. Prof. Dr. Firudin Muradov, Near East University for his kind patronage, guidance and concern not only during this thesis but always. I am really thankful to him for being there whenever I needed his advice. Also I wish to thank Assist. Prof. Dr. Doğan Haktanır, Assoc. Prof. Dr. Rahib Abiyev and Assoc. Prof. Dr. İlham Hüseynov.

With great respect and gratitude, I wish to thank Dr. Fakhraddin Mamedov, Dean of Engineering faculty, Near East University for providing me necessary atmosphere in the completion of MSc Computer Engineering.

My special note of thanks to all my friends, who made my stay in Near East University possible. I am really thankful to them for bearing with me and my moods during my studies.

No words could express the love and understanding my parents have extended to me during the whole period of my education and throughout my life. The same goes for the rest of my family who always had a word of appreciation and always stood beside me.

ABSTRACT

Hospital Management System is powerful, flexible and easy to use and has designed and developed to deliver real believable benefits to hospitals and clinics. Through its electronic patient information system, Hospital Management System ensures optimum use of the medical records of patients. Being patient-centric, the system makes it possible for all disparate files on a patient to be housed in a centrally located master file, ensuring retrieval and management of files effortless. Plus, easy integration ensures a faster start up.

Hospital Management System has been so designed as to make its usage intuitive, which makes adoption of Hospital Management System by the hospital personnel quick and easy. Its high level of scalability ensures quick upgrades when change in the hospital's business practices so demands. The developed system can make searching not only by standard, typical queries of user and non typical queries of user.

Hospital Management System is designed for multi-specialty hospitals, to cover a wide range of Hospital administration and management processes. It is an integrated client server application uses Delphi as the front-end for Graphical User Interface and MS SQL as the database. The main focus of this system is, how to use client-server computing concept with centralized database.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
CONTENTS	iii
INTRODUCTION	vii
INFORMATION FOR CLIENT SERVER SYSTEM	1
1 OVERVIEW	1
1.1 Origin of Term Client/Server	1
1.2 What is a Client process?	1
1.3 What is a Server process?	1
1.4 What is Client/Server System?	2
1.5 Design Structures for Client/Server	2
1.6 Two-Tier Design Structure	3
1.7 Three-Tier or N-Tier Design Structure	3
1.8 Distributed Design Structure	3
1.9 Summary	4
CLIENT SERVER COMPUTING	5
2 OVERVIEW	5
2.1 Overview of Client Server	5
2.2 Introduction to Client Server Systems	6
2.3 Definition of Client and Server	7
2.4 The Evolution of Clients and Servers	7
2.5 Client/Server Examples	8
2.6 Putting It All Together	8
2.7 Client Server Architecture	9
2.7.1 Two-tier Architecture	9
2.7.2 Three tier or N-tier Architecture	12
2.8 Advantages of Client Server Computing	15
2.9 Summary	15
DATABASE SYSTEM	16
3 OVERVIEW	16
3.1 Data Modeling Overview	16
3.1.1 Methodology	17
3.1.2 Data Modeling in the Context of Database Design	17
3.1.3 Components of a Data Model	17
3.1.4 Why is Data Modeling Important?	18
3.1.5 The Entity-Relationship Model	18

3.1.6	Basic Constructs of E-R Modeling	18
3.2	Data Modeling As Part of Database Design	23
3.2.1	Requirements Analysis	23
3.2.2	Steps in Building the Data Model	25
3.2.3	Normalization	33
3.3	Architectures of Database System	42
3.3.1	Centralized Database System	42
3.3.2	Distributed Database System	43
3.4	Summary	44
HOSPITAL MANAGEMENT SYSTEM AND DATABASE STRUCTURE OF HOSPITAL		45
4	OVERVIEW	45
4.1	Hospital Management System Solution for Hospitals	45
4.2	Hospital Management System – Registration	45
4.3	Hospital Management System - Doctor	45
4.4	Hospital Management System - Wards	46
4.5	Hospital Management System – Rooms	46
4.6	Hospital Management System - Pharmacy	46
4.7	Hospital Management System - User Manager	47
4.8	Benefit of Hospital Network	47
4.9	Database Design for Hospital Management System	48
Table 4.1	Patient Table Structure	49
Table 4.2	Outpatient Table Structure	49
Table 4.3	Inpatient Table Structure	49
Table 4.4	Room Table Structure	50
Table 4.5	Ward Table Structure	50
Table 4.6	Bed Availability Table Structure	50
Table 4.7	Doctor Table Structure	50
Table 4.8	Department Table Structure	51
Table 4.9	Inpatient Table Structure	51
Table 4.10	Pharmacy Table Structure	51
4.10	Adding Patient Algorithm	52
4.11	Adding Outpatient Algorithm	53
4.12	Adding Inpatient Algorithm	54
4.13	Adding Inpatient Disease Detail Algorithm	55
4.14	Relationship of Tables (MS SQL / hosdata1)	56
4.15	Summary	57
IMPLEMENTATION OF HOSPITAL MANAGEMENT SYSTEM IN DELPHI AND USER INTERFACE		58
5	OVERVIEW	58
5.1	Administrator Application	58

5.1.1	Administrator Application Main Form	58
5.1.2	Administrator Application \ Patient Menu	59
5.1.3	Administrator Application \ Outpatient Menu	59
5.1.4	Administrator Application \ Inpatient Menu	60
5.1.5	Administrator Application \ Room Menu	60
5.1.6	Administrator Application \ Ward Menu	61
5.1.7	Administrator Application \ Doctor Menu	61
5.1.8	Administrator Application \ Department Menu	62
5.1.9	Administrator Application \ Inpatient Disease Detail Menu	63
5.1.10	Administrator Application \ Pharmacy Menu	63
5.2	Reception Application	64
5.2.1	Reception Application Main Menu	64
5.2.2	Reception Application \ Add Menu	64
5.2.3	Reception Application \ Search Record	65
5.2.4	Reception Application \ Modify Record	65
5.2.5	Reception Application \ Show All Patient	66
5.3	Doctor Application	66
5.3.1	Doctor Application main form	66
5.3.2	Doctor Application \ Patient \ Search Record	67
5.3.3	Doctor Application \ Outpatient Menu	68
5.3.4	Doctor Application \ Outpatient Menu \ Add Record	68
5.3.5	Doctor Application \ Outpatient Menu \ Search Record	69
5.3.6	Doctor Application \ Outpatient Menu \ Modify Record	69
5.3.7	Doctor Application \ Inpatient Menu	70
5.3.8	Doctor Application \ Inpatient Menu \ Inpatient \ Search Record	70
5.3.9	Doctor Application \ Inpatient Menu \ Inpatient Disease Detail \ Search Record	71
5.3.10	Doctor Application \ Inpatient Menu \ Advanced Search	72
5.4	Room Application	72
5.4.1	Room Application main form	72
5.4.2	Room Application \ Inpatient Menu	73
5.4.3	Room Application \ Inpatient Menu \ Add Record	73
5.4.4	Room Application \ Inpatient Menu \ Search Record	74
5.4.5	Room Application \ Inpatient Menu \ Modify Record	74
5.4.6	Room Application \ Inpatient Disease Detail Menu	75
5.4.7	Room Application \ Inpatient Disease Detail Menu \ Add Record	75
5.4.8	Room Application \ Inpatient Disease Detail Menu \ Search Record	76
5.4.9	Room Application \ Inpatient Disease Detail Menu \ Modify Record	76
5.5	Ward Application	77
5.5.1	Ward Application main form	77
5.5.2	Ward Application \ Inpatient Menu	77
5.5.3	Ward Application \ Inpatient Menu \ Add Record	78
5.5.4	Ward Application \ Inpatient Menu \ Search Record	78
5.5.5	Ward Application \ Inpatient Menu \ Modify Record	79
5.5.6	Ward Application \ Inpatient Disease Detail Menu	79
5.5.7	Ward Application \ Inpatient Disease Detail Menu \ Add Record	80

5.5.8	Ward Application \ Inpatient Disease Detail Menu \ Search Record	80
5.5.9	Ward Application \ Inpatient Disease Detail Menu \ Modify Record	81
5.6	Pharmacy Application	81
5.6.1	Pharmacy Application main form	81
5.6.2	Pharmacy Application \ Add Menu	82
5.6.3	Pharmacy Application \ Stock In Menu	82
5.6.4	Pharmacy Application \ Stock Out Menu	83
5.6.5	Pharmacy Application \ Search Menu	83
5.6.6	Pharmacy Application \ Modify Record	84
5.7	Definition of Terms and Connection	84
5.7.1	Microsoft Data Access Components (MDAC)	84
5.7.2	How to Create Open Database Connectivity (ODBC)	85
5.7.3	How to Create ADO Connection for Delphi 7	91
	CONCLUSION	94
	REFERENCES	95
	APPENDIX A	96
	APPENDIX B	143

INTRODUCTION

Hospital Management System is designed to be used with a wide range of Hospital administration and management processes. It is a client server application which is build in Delphi as the front-end for Graphical User Interface, using MS SQL as the database. The main focus of this system is, how to use client-server computing concept with centralized database. This project investigates the concept of Client/Server system, the database and shows how is can be applied to solve the real world problems.

Many varieties of modern software use a client\server architecture, in which by one process (the client) are sent to another process (the server) for execution. Database systems are no exception, and it has become increasingly common to divide the work of a DBMS into a server process and one or more client processes.

In the simplest client\server architecture, the entire DBMS is a server, except for the query interfaces that interact with the user and send queries or other commands across to the server. For example, relational systems generally use the SQL language for representing requests from the client to the server. The DB server then sends the answer, in the form of a table or relation back to the client. The first important applications of DBMS's were Airline Reservations Systems, Banking Systems and Corporate Records.

In Airline Reservations Systems the items of data include :

- Reservations by a single customer on a single flight
- Information about flights
- Information about ticket prices

In banking systems data items include names and addresses of customers, accounts, loans and their balances and the connection between customers and their accounts and loans. Many early applications concerned corporate records, such as a record of each sale, information about accounts payable and receivable or information about employees. Recently in NEU developed DB system that covers many activities of the

university, such as enrollment of students in courses, payment of fees, automatic calculation of GPA, etc.

Comparison and the details of Hospital Management System Solutions are inside of Appendix B.

The aim of my thesis is to develop the easiest and most friendly user interface program for a hospital management system. This program contains all the gathered and required information to form a highly effective program.

This thesis includes five chapters;

In chapter 1 we focus our attention on Client/Server process. Most of the operative applications and information systems are nowadays built using Client/Server architecture. The applications are meant to support the main functions of organizations such as insurance, banking, manufacturing, administration, etc. Also this chapter includes definition of Design Structure for Client/Server.

In chapter 2 includes the Client/Server Computing and Client/Server architecture. There are 2 main Client/Server architectures; Two-tier and N-tier architecture.

Chapter 3 describes the Database System; Data Modeling, Database Design and Architecture of Database System.

Chapter 4 gives information about Hospital Management System (HMS) and Database Design for Hospital Management System. It is integrated Client/Server application, which uses Delphi as the front-end for Graphical User Interface and MS SQL as the database. This chapter also describes the structure of tables and relationship between tables.

All applications of this thesis (Administrator, Reception, Doctor, Room, Ward, Pharmacy) interface and detail of Microsoft Data Access Components (MDAC), ActiveX Data Objects (ADO), Object Linking and Embedding (OLE), Component Object Model (COM), Using Data Sources (ODBC) and ADO Connection for Delphi 7 are described in Chapter 5.

CHAPTER 1

INFORMATION FOR CLIENT SERVER SYSTEM

1 Overview

First chapter refers to the definition of client/server system and design structure of client/server system. There are 3 main client/server design structures. This chapter also includes all approach for design structure.

1.1 Origin of Term Client/Server

The term client/server was first used in the 1980s in reference to personal computers (PCs) on a network. The actual client/server model started gaining acceptance in the late 1980s. The client/server software architecture is a multipurpose, message-based and modular infrastructure that is intended to improve usability, flexibility, interoperability and scalability as compared to centralized, mainframe, time sharing computing.

1.2 What is a Client process?

The client is a process (program) that sends a message to a server process (program), requesting that the server perform a task (service). Client programs usually manage the user-interface portion of the application, validate data entered by the user, dispatch requests to server programs, and sometimes execute business logic. The client-based process is the front- end of the application that the user sees and interacts with. The client process contains solution specific logic and provides the interface between the user and the rest of the application system. The client process also manages the local resources that the user interacts with such as the monitor, keyboard, workstation CPU and peripherals. One of the key elements of a client workstation is the graphical user interface (GUI). Normally a part of operating system i.e. the window manager detects user actions, manages the windows on the display and displays the data in the windows.

1.3 What is a Server process?

A server process (program) fulfills the client request by performing the task requested. Server programs generally receive requests from client programs, execute

database retrieval and updates, manage data integrity and dispatch responses to client requests. Sometimes server programs execute common or complex business logic. The server-based process "may" run on another machine on the network. This server could be the host operating system or network file server; the server is then provided both file system services and application services. Or in some cases, another desktop machine provides the application services. The server process acts as a software engine that manages shared resources such as databases, printers, communication links, or high powered-processors. The server process performs the back-end tasks that are common to similar applications.

1.4 What is Client/Server System?

Client/server is a distributed computing model in which *client* applications request services from *server* processes. Clients and servers typically run on different computers interconnected by a computer network. A *client* application is a process or program that sends messages to a server via the network. Those messages request the server to perform a specific task, such as looking up a customer record in a database or returning a portion of a file on the server's hard disk. The client manages local resources such as a display, keyboard, local disks and other peripherals. The *server* process or program listens for client requests that are transmitted via the network. Servers receive those requests and perform actions such as database queries and reading files. Server processes typically run on powerful PCs, workstations or on mainframe computers.

1.5 Design Structures for Client/Server

In a client-server information collection and retrieval environment, the client program usually accepts user requests and provides screen displays. Server programs generally reside on more powerful machines and are used to process information generally stored in databases. When client-server programs are on different machines, the client and server machines are linked together by networks. Thus, client-server applications are naturally modularized into client, server and networking components.

The design structures for client server are as follow:

- Two-tier
- Three-tier
- Distributed

Other client-server design structures can be developed using a combination or variation of three. In choosing any design structure, application developers are advised to consider the trade-offs among development time, development effort, application complexity and software distribution efforts.

1.6 Two-Tier Design Structure

A two-tier client-server application design structure consists of a client program residing on a client machine communicating through a network with a database server program that resides on a server machine. In this case, the server program is usually a database server program that performs the read/write operations to the actual database itself. The database server program usually resides on the same server machine as the database itself, and is written and provided by the client-server tool provider. The tool provider also offers for inclusion in the client program an application program interface (API) that communicates with the database server program on the server machine.

1.7 Three-Tier or N-Tier Design Structure

Three-tier refers to the client program, the server program and the database server program on the server machine.

For this design structure, application developers must develop the client program and the server program. After that, it is necessary also to include the database server program as part of the server program in order to manage access and retrieval of database information.

1.8 Distributed Design Structure

The distributed client-server design structure differs from the three-tier structure in the location of the database server program and of the database itself. When the database server program used by the application system resides on the same machine as the server program, the structure becomes a three-tier structure. When the

database server program and the actual database used by the application reside on different machines, the design structure becomes a distributed client-server structure. In the distributed structure, the server program and the database server program are logically linked by the network.

This design approach requires about the same development time and effort as the three-tier structure, in which the server program and the database server program reside on the same server. It offers the developer a design structure of distributed database application systems, providing more flexibility for server location management and data accessibility than all designs mentioned here.

In most cases in which a developer chooses to prototype an application, it is advantageous to choose a client-server design structure that provides for fast development and requires only a small amount of effort to develop the application system. When the application system is reviewed and accepted by users, the design structure can then be modified to achieve other goals, such as enhanced capabilities and increased performance. When this methodology is used, the two-tier design structure is the best candidate. I chose the two-tier design structure for my theses because this architecture is fitted with the application which I choose for this thesis.

1.9 Summary

In conclusion, in choosing a structure, developers must view the choice as a business decision by considering priorities and trade-offs. The design structure for each application must be evaluated individually to take advantage of differences in capability, development effort, development time, degree of flexibility, and performance.

CHAPTER 2

CLIENT SERVER COMPUTING

2 Overview

Chapter two includes an introduction to client/server systems and client/server architectures. This chapter focuses on the most popular forms of implementation of two-tier and three-tier (or n-tier) client/server computing systems.

2.1 Overview of Client Server

CLIENT/SERVER concepts have roots in early computer systems, so it is useful to briefly review the history of modern computing. Early computers (mainframes) were typically in a "glass room," with special power and air conditioning, and attended to by a priesthood of system programmers.

Users typically shared a pool of "dumb" terminals and had to rely on centralized printing and storage resources. In this environment, the mainframe did all the processing, and users had no local computing horsepower.

In the late 1970s and early 1980s, smaller systems (minicomputers) were developed that required less power and air conditioning. Individual business unit owners wanted their own systems, more suited to their needs. Once a substantial number of applications were developed on these computers, inevitably other departments wanted access to them.

Somewhat concurrently, Apple, IBM and others developed the personal computer. Initially it seemed to have little application beyond that of the hobbyist. However, as powerful, shrink-wrapped applications were developed, and arcane operating systems such as MS-DOS were replaced by more user-friendly systems (e.g., Macintosh, Windows), users began to want to use PCs in the corporate computing environment.

This allowed for a new army of computer users who cared little and understood even less about computer "systems." All they wanted was the latest software applications at their fingertips, but they needed the backup and redundant architecture that previously resided only in the glass room.

Concurrently, systems that were more scalable were introduced using common chip sets and operating systems, with features such as redundant power and multiple processors. They were far more powerful than "personal" computers although they were based largely on the same architecture but they were considerably smaller and less costly than mainframes or even minicomputers.

One last piece of technology was needed to make client/server architecture a reality. In the late 1970s, Xerox developed the standards and technology that we know today as Ethernet. This provided a standard means of linking together computers from different manufacturers and formed the basis for modern local area networks (LANs) and wide area networks (WANs).

At this point, all the pieces were in place to develop client/server systems:

- A strong business need for decentralized computing horsepower
- Standard, powerful computers with user-friendly interfaces
- Mature, shrink-wrapped user applications with widespread acceptance
- Inexpensive, modular systems designed with enterprise-class

2.2 Introduction to Client Server Systems

Most of the initial client/server success stories involve small-scale applications that provide direct or indirect access to transactional data in legacy systems. The business need to provide data access to decision makers, the relative immaturity of client/server tools and technology, the evolving use of wide area networks and the lack of client/server expertise make these attractive yet low risk pilot ventures. As organizations move up the learning curve from these small-scale projects towards mission-critical applications, there is a corresponding increase in performance expectations, uptime requirements and in the need to remain both flexible and scalable. In such a demanding scenario, the choice and implementation of

appropriate architecture becomes critical. In fact one of the fundamental questions that practitioners have to contend with at the start of every client/server project is Which architecture is more suitable for this project Two Tier or Three Tier architecture.

Architecture affects all aspects of software design and engineering. The architect considers the complexity of the application, the level of integration and interfacing required the number of users, their geographical dispersion, the nature of networks and the overall transactional needs of the application before deciding on the type of architecture. An inappropriate architectural design or a flawed implementation could result in horrendous response times. The choice of architecture also affects the development time and the future flexibility and maintenance of the application. This report defines the basic concepts of client/server architecture, describes the two tier and three tier or N-tier architectures.

2.3 Definition of Client and Server

In the simplest sense, the client and server can be defined as follows:

A **client** is an individual user's computer or a user application that does a certain amount of processing on its own. It also sends and receives requests to and from one or more servers for other processing and/or data.

A **server** consists of one or more computers that receive and process requests from one or more client machines. A server is typically designed with some redundancy in power, network, and computing and file storage. However, a machine with dual processors is not necessarily a server. An individual workstation can function as a server.

Sometimes the term "server" or "client" may refer to the software rather than the computer. Thus, a "mail client" may refer to the mail software that resides on a client machine, rather than the machine itself.

2.4 The Evolution of Clients and Servers

The earliest versions of client/server were merely shared files; when a user needed a file, it was copied from the server to the local machine. When finished, it was

returned to the server. It was necessary to develop certain rules to handle conflict and synchronization issues. So as client/server systems evolved, they contained built-in synchronization and sharing engines. Client/server also embodies the concepts of user accounts and sharing of resources the system must separate and keep track of different users files and applications during a user session, then free up those resources for another user session. As client/server applications evolved, the functionality of the application was separated logically into two parts: the processes requiring the majority of the computing horsepower were put on the server, and the user interface and less processor intensive processes were put on the client.

2.5 Client/Server Examples

A common client/server example is a print server. Most people have probably noticed a temporary lockup or slowdown when a document is printed on a stand alone PC, especially if the document is complex. One can attach a printer to a PC and then share it with other users across the network. However, if everyone on the network simultaneously prints to that shared printer, it would likely lock up or even crash. Therefore, many times a machine is dedicated solely to handle printing a print server. It "serves" print requests to all users, and off-loads this task from local machines. Another example is a mail server which functions much like a post office, receiving mail centrally and delivering individual messages to individual clients.

2.6 Putting It All Together

In a small organization, a single server machine may serve more than one function, if the functions are simple enough. One or more applications may reside on a single server machine, with the server being divided into different "logical" partitions. In a large corporate environment, there may be many servers for separate tasks.

There is typically a primary domain controller (PDC), which authenticates users and controls access and log in to the computer system itself. There may be a mail server, which processes e-mail. There may also be a file server typically containing large disk drives and individual user directories to store user files in a uniform way. And there may be separate application servers for accounting, billing, customer care,

Web, e-commerce, database, transaction, manufacturing, inventory, etc. They are typically linked together using integration software (frequently called middleware) so that one can access many server applications from a single (client) machine, through a common interface, typically a browser.

Although client/server in its simplest form is two-tier (server and client), there are newer, more powerful architectures that are three-tier (where application logic lives in the middle-tier and it is separated from the data and user interface) or even n-tier (where there are several middle-tier components within a single business transaction) in nature. Sometimes client/server is referred to as distributed computing; they have the same basic concepts.

2.7 Client Server Architecture

When considering a move to client/server computing, whether it is to replace existing systems or introduce entirely new systems, practitioners must determine which type of architecture they intend to use. The vast majority of end user applications consist of three components: presentation, processing, and data. The client/server architectures can be defined by how these components are split up among software entities and distributed on a network. There are a variety of ways for dividing these resources and implementing client/server architectures. This paper will focus on the most popular forms of implementation of two-tier and three-tier client/server computing systems.

2.7.1 Two-tier Architecture

Although there are several ways to architect a two-tier client/server system, we will focus on examining what is overwhelmingly the most common implementation. In this implementation, the three components of an application (presentation, processing, and data) are divided among two software entities (tiers): client application code and database server (Figure 2.1). A robust client application development language and a versatile mechanism for transmitting client requests to the server are essential for a two-tier implementation. Presentation is handled exclusively by the client, processing is split between client and server, and data is stored on and accessed via the server. The PC client assumes the bulk of responsibility for application (functionality) logic with respect to the processing

component, while the database engine with its attendant integrity checks, query capabilities and central repository functions handles data intensive tasks. In a data access topology, a data engine would process requests sent from the clients. Currently, the language used in these requests is most typically a form of SQL. Sending SQL from client to server requires a tight linkage between the two layers. To send the SQL the client must know the syntax of the server or have this translated via an API (Application Program Interface). It must also know the location of the server, how the data is organized, and how the data is named. The request may take advantage of logic stored and processed on the server which would centralize global tasks such as validation, data integrity, and security. Data returned to the client can be manipulated at the client level for further sub selection, business modeling, "what if" analysis, reporting, etc.

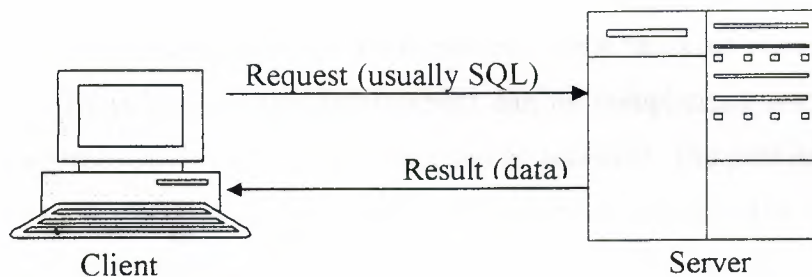


Figure 2.1 Data Access Topology for two-tier architecture. Majority of functional logic exists at the client level

The most compelling advantage of a two-tier environment is application development speed. In most cases a two-tier system can be developed in a small fraction of the time it would take to code a comparable but less flexible legacy system. Using any one of a growing number of PC-based tools, a single developer can model data and populate a database on a remote server, paint a user interface, create a client with application logic, and include data access routines. Most two-tier tools are also extremely robust. These environments support a variety of data structures, including a number of built in procedures and functions, and insulate developers from many of the more mundane aspects of programming such as

memory management. Finally these tools also lend themselves well to iterative prototyping and rapid application development (RAD) techniques, which can be used to ensure that the requirements of the users are accurately and completely met.

Two-tier architectures work well in relatively homogeneous environments with fairly static business rules. This architecture is less suited for dispersed, heterogeneous environments with rapidly changing rules.

Since the bulk of application logic exists on the PC client, the two-tier architecture faces a number of potential version control and application re-distribution problems. A change in business rules would require a change to the client logic in each application in a corporation's portfolio which is affected by the change. Modified clients would have to be re-distributed through the network a potentially difficult task given the current lack of robust PC version control software and problems associated with upgrading PCs that are turned off or not "docked" to the network.

System security in the two-tier environment can be complicated since a user may require a separate password for each SQL server accessed. The proliferation of end-user query tools can also compromise database server security. The overwhelming majority of client/server applications developed today is designed without sophisticated middleware technologies which offer increased security. Instead, end-users are provided a password which gives them access to a database. In many cases this same password can be used to access the database with data-access tools available in most commercial PC spreadsheet and database packages. Using such a tool, a user may be able to access otherwise hidden fields or tables and possibly corrupt data.

Client tools and the SQL middleware used in two-tier environments are also highly proprietary and the PC tools market is extremely volatile. The client/server tools market seems to be changing at an increasingly unstable rate. In 1994, the leading client/server tool developer was purchased by a large database firm, raising concern about the manufacturer's ability to continue to work cooperatively with RDBMS vendors which compete with the parent company's products. The number two

toolmaker lost millions and has been labeled as a takeover target. The tool which has received some of the brightest accolades in early 1995 is supplied by a firm also in the midst of severe financial difficulties and management transition. This kind of volatility raises questions about the long term viability of any proprietary tool an organization may commit to. All of this complicates implementation of two-tier systems migration from one proprietary technology to another would require a firm to scrap much of its investment in application code since none of this code is portable from one tool to the next.

2.7.2 Three tier or N-tier Architecture

The tree tier architecture (Figure 2.2) attempts to overcome some of the limitations of the two-tier scheme by separating presentation, processing, and data into separate, distinct software entities (tiers). The same types of tools can be used for presentation as were used in a two-tier environment; however these tools are now dedicated to handling just the presentation. When calculations or data access is required by the presentation client, a call is made to a middle tier functionality server. This tier can perform calculations or can make requests as a client to additional servers. The middle tier servers are typically coded in a highly portable, non-proprietary language such as C. Middle-tier functionality servers may be multi-threaded and can be accessed by multiple clients, even those from separate applications.

Although three-tier systems can be implemented using a variety of technologies, the calling mechanism from client to server in such as system is most typically the (remote procedure call or) RPC. Since the bulk of two-tier implementations involve SQL messaging and most three-tier systems utilize RPCs, it is reasonable to examine the merits of these respective request/response mechanisms in a discussion of architectures. RPC calls from presentation client to middle-tier server provide greater overall system flexibility than the SQL calls made by clients in the two-tier architecture. This is because in an RPC, the requesting client simply passes parameters needed for the request and specifies a data structure to accept returned values (if any). Unlike most two-tier implementations, the three-tier presentation client is not required to "speak" SQL. As such, the organization, names, or even the overall structure of the back-end data can be changed without requiring changes to PC-based presentation clients. Since SQL is no longer required, data can be

organized hierarchically, relationally, or in object format. This added flexibility can allow a firm to access legacy data and simplifies the introduction of new database technologies.

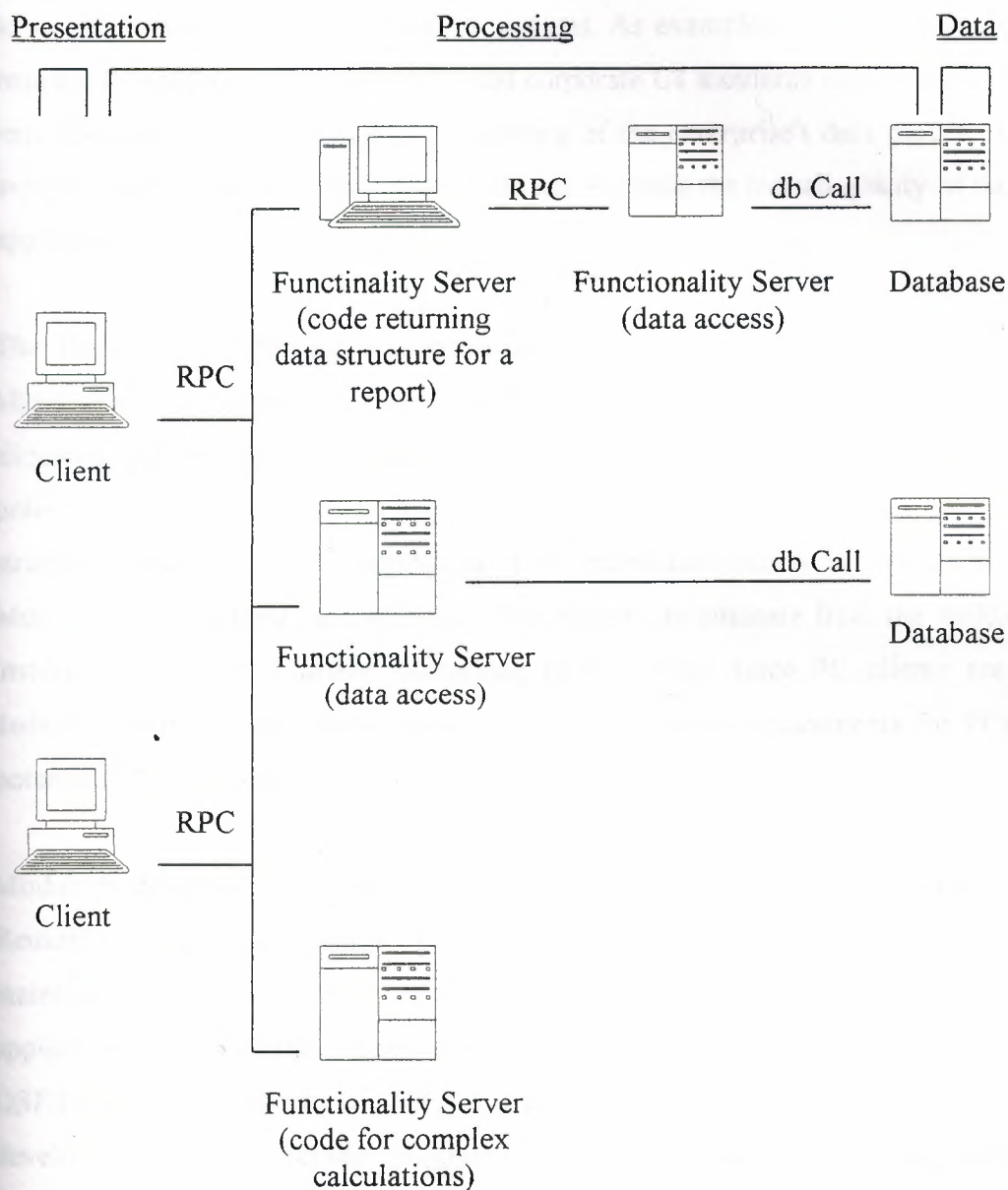


Figure 2.2 Three Tier Architecture. Most of the logic processing is handled by functionality servers. Middle-tier code can be accessed and utilized by multiple clients

In addition to the openness stated above, several other advantages are presented by this architecture. Having separate software entities can allow for the parallel development of individual tiers by application specialists. It should be noted that the skill sets required to develop c/s applications differ significantly from those needed to develop mainframe-based character systems. As examples, user interface creation requires an appreciation for platform and corporate UI standards and database design requires a commitment to and understanding of the enterprise's data model. Having experts focus on each of these three layers can increase the overall quality of the final application.

The three-tier architecture also provides for more flexible resource allocation. Middle-tier functionality servers are highly portable and can be dynamically allocated and shifted as the needs of the organization change. Network traffic can potentially be reduced by having functionality servers strip data to the precise structure required before distributing it to individual clients at the LAN level. Multiple server requests and complex data access can emanate from the middle tier instead of the client, further decreasing traffic. Also, since PC clients are now dedicated to just presentation, memory and disk storage requirements for PCs will potentially be reduced.

Modularly designed middle tier code modules can be re-used by several applications. Reusable logic can reduce subsequent development efforts, minimize the maintenance workload, and decrease migration costs when switching client applications. In addition, implementation platforms for three tier systems such as OSF/DCE offer a variety of additional features to support distributed application development. These include integrated security, directory and naming services, server monitoring and boot capabilities for supporting dynamic fault-tolerance, and distributed time management for synchronizing systems across networks and separate time zones.

There are of course drawbacks associated with three-tier architecture. More code in more places also increases the likelihood that a system failure will effect an

application so detailed planning with an emphasis on the reduction/elimination of critical paths is essential. Three tier brings with it an increased need for network traffic management, server load balancing, and fault tolerance.

2.8 Advantages of Client Server Computing

There are many advantages to client/server architecture including that subsystems can be optimized for a particular set of applications. Systems can grow modularly, as different applications grow. Then more powerful subsystems can be installed without wasting resources on other applications. "Forklift upgrades," where an entire system is replaced, are theoretically kept to a minimum.

With most of the crucial applications and data residing on centralized machines, or clusters of machines, systems can be engineered to high standards of reliability and availability.

2.9 Summary

In conclusion, client/server architecture has become the dominant structure for corporate computing in both small and large organizations. It combines the best concepts of centralized, robust infrastructure with decentralized capability and control in other words, it gives both IT managers and end users what they want and need. If implemented properly, client/server architecture achieves the best balance between complexity, cost and ease of use, with excellent scalability and reliability.

CHAPTER 3

DATABASE SYSTEM

3 Overview

This chapter is an informal introduction to data modeling using the Entity-Relationship (ER) approach. The basic techniques described are applicable to the development of relational database applications as well as those who use relational database servers such as MS SQL Server or Oracle. There are two main architectures of database, Distributed Database System and Centralized Database System. For implementation of these concepts, database should be normalized. I choose the Centralized Database architectures for my application. The detail for these techniques is mentioned in the coming section of this chapter. Normalization is a design technique to design a database but for this we must perform the analysis of database. Data Modeling is the way we can perform the analysis for a given problem.

3.1 Data Modeling Overview

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects and the rules which govern operations on the objects. As the name implies, the data model focuses on what data is required and how it should be organized rather than what operations will be performed on the data.

A data model is independent of hardware or software constraints. Rather than try to represent the data as a database would see it, the data model focuses on representing the data as the user sees it in the "real world". It serves as a bridge between the concepts that make up real-world events and processes and the physical representation of those concepts in a database.

3.1.1 Methodology

There are two major methodologies used to create a data model: the Entity-Relationship (ER) approach and the Object Model. I used the Entity-Relationship approach.

3.1.2 Data Modeling in the Context of Database Design

Database design is defined as: "design the logical and physical structure of one or more databases to accommodate the information needs of the users in an organization for a defined set of applications". The design process roughly follows five steps:

- Planning and analysis
- Conceptual design
- Logical design
- Physical design
- Implementation

The data model is one part of the conceptual design process. The other, typically is the functional model. The data model focuses on what data should be stored in the database while the functional model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries which will access and perform operations on those tables.

3.1.3 Components of a Data Model

The data model gets its inputs from the planning and analysis stage. Here the modeler, along with analysts, collects information about the requirements of the database by reviewing existing documentation and interviewing end-users.

The data model has two outputs. The first is an entity-relationship diagram which represents the data structures in a pictorial form. Because the diagram is easily learned, it is valuable tool to communicate the model to the end-user. The second component is a data document. This document that describes in details the data

objects, relationships, and rules required by the database. The dictionary provides the detail required by the database developer to construct the physical database.

3.1.4 Why is Data Modeling Important?

The goal of the data model is to make sure that the all data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language, it can be reviewed and verified as correct by the end-users. A data model is a plan for building a database. To be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database design to use to create the physical structure. The Entity-Relation Model (ER) is the most common method used to build data models for relational databases.

3.1.5 The Entity-Relationship Model

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. The process of designing a database begins with an analysis of what information the database must hold and what are the relationships among components of that information. Often, the structure of the database, called the database schema, is specified in one of several languages or notations suitable for expressing designs. After due consideration, the design is committed to a form in which it can be input to a DBMS, and the database takes on physical existence.

3.1.6 Basic Constructs of E-R Modeling

The ER model views the real world as a construct of entities and association between entities.

3.1.6.1 Entities

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person,

places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, and INVOICES. An entity is analogous to a table in the relational model. Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An *independent entity* is one that does not rely on another for identification. A *dependent entity* is one that relies on another for identification. An *entity occurrence* (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

3.1.6.2 Special Entity Types

Associative entities (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship. Subtypes entities are used in generalization hierarchies to represent a subset of instances of their parent entity, called the super type, but which have attributes or relationships that apply only to the subset. Associative entities and generalization hierarchies are discussed in more detail below.

3.1.6.3 Relationships

A Relationship represents an association between two or more entities. An example of a relationship would be:

- Employees are assigned to projects
- Projects have subtasks
- Departments manage one or more projects

Relationships are classified in terms of degree, connectivity, cardinality, and existence. These concepts will be discussed below.

3.1.6.4 Attributes

Attributes describe the entity of which they are associated. A particular instance of an attribute is a value. For example, "Jane R. Hathaway" is one value of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string. Attributes can be classified as

identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

3.1.6.5 Classifying Relationships

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

3.1.6.6 Degree of a Relationship

The degree of a relationship is the number of entities associated with the relationship. The n relationship is the general form for degree n . Special cases are the binary, and ternary, where the degree is 2 and 3 respectively. A binary relationship, the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees". A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n relationships are decomposed into two or more binary relationships.

3.1.6.7 Connectivity and Cardinality

The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many and many-to-many.

A *one-to-one* (1:1) relationship is when at most one instance of an entity A is associated with one instance of entity B. For example, "employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.

A *one-to-many* (1:N) relationships is when for one instance of entity A, there are zero, one or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

- A department has many employees

- Each employee is assigned to one department

A *many-to-many* (M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one or many instances of entity B and for one instance of entity B there are zero, one or many instances of entity A. An example is:

- Employees can be assigned to no more than two projects at the same time;
- Projects must have assigned at least three employees

A single employee can be assigned too many projects; conversely, a single project can have assigned to it many employees. Here the cardinality for the relationship between employees and projects is two and the cardinality between project and employee is three. Many-to-many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

3.1.6.8 Direction

The direction of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is the *parent entity*; the entity where the relationship terminates is the *child entity*.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

3.1.6.9 Type

An identifying relationship is one in which one of the child entities is also a dependent entity. A *non-identifying relationship* is one in which both entities are independent.

3.1.6.10 Existence

Existence denotes whether the existence of an entity instance is dependent upon the existence of another related entity instance. The existence of an entity in a

relationship is defined as either *mandatory* or *optional*. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory. An example of mandatory existence is the statement "every project must be managed by a single department". If the instance of the entity is not required, it is optional. An example of optional existence is the statement, "employees may be assigned to work on projects".

Examples of these symbols (Rectangular) are shown in Figure 3.1 below:

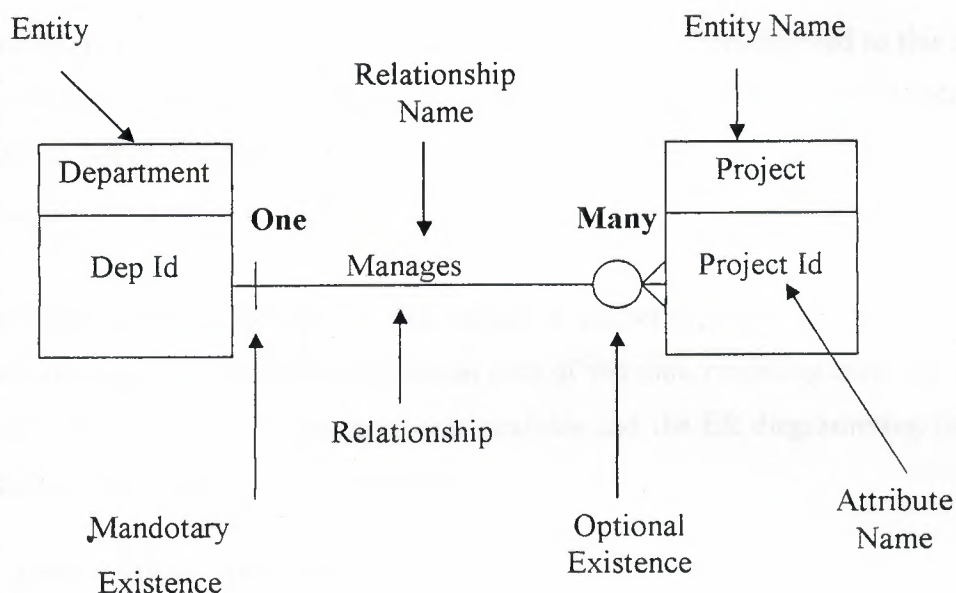


Figure 3.1 ER Notation

The Entity-Relationship Model is a conceptual data model that views the real world as consisting of entities and relationships. The model visually represents these concepts by the Entity-Relationship diagram. The basic constructs of the ER model are entities, relationships, and attributes. Entities are concepts, real or abstract about which information is collected. Relationships are associations between the entities. Attributes are properties which describe the entities. Next, we will look at the role of data modeling in the overall database design process and a method for building the data model.

3.2 Data Modeling As Part of Database Design

The data model is one part of the conceptual design process. The other is the function model. The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries that will access and perform operations on those tables.

Data modeling is preceded by planning and analysis. The effort devoted to this stage is proportional to the scope of the database. The planning and analysis of a database intended to serve the needs of an enterprise will require more effort than one intended to serve a small workgroup.

The information needed to build a data model is gathered during the requirements analysis. Although not formally considered part of the data modeling stage by some methodologies, in reality the requirements analysis and the ER diagramming part of the data model are done at the same time.

3.2.1 Requirements Analysis

The goals of the requirements analysis are:

- To determine the data requirements of the database in terms of primitive objects
- To classify and describe the information about these objects
- To identify and classify the relationships among the objects
- To determine the types of transactions that will be executed on the database and the interactions between the data and the transactions
- To identify rules governing the integrity of the data

The modeler or modelers, works with the end users of an organization to determine the data requirements of the database. Information needed for the requirements analysis can be gathered in several ways:

Review of existing documents such documents include existing forms and reports, written guidelines, job descriptions, personal narratives and memoranda. Paper documentation is a good way to become familiar with the organization or activity you need to model.

Interviews with end users these can be a combination of individual or group meetings. Try to keep group sessions to under five or six people. If possible, try to have everyone with the same function in one meeting. Use a blackboard, flip charts or overhead transparencies to record information gathered from the interviews.

Review of existing automated systems if the organization already has an automated system, review the system design specifications and documentation.

The requirements analysis is usually done at the same time as the data modeling. As information is collected, data objects are identified and classified as entities, attributes or relationship; assigned names; and defined using terms familiar to the end-users. The objects are then modeled and analyzed using an ER diagram. The diagram can be reviewed by the modeler and the end-users to determine its completeness and accuracy. If the model is not correct, it is modified, which sometimes requires additional information to be collected. The review and edit cycle continues until the model is certified as correct.

Three points to keep in mind during the requirements analysis are:

Talk to the end users about their data in "real-world" terms. Users do not think in terms of entities, attributes and relationships but about the actual people, things and activities they deal with daily.

Take the time to learn the basics about the organization and its activities that you want to model. Having an understanding about the processes will make it easier to build the model.

End-users typically think about and view data in different ways according to their function within an organization. Therefore, it is important to interview the largest number of people that time permits.

3.2.2 Steps in Building the Data Model

While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies specify a bottom-up development process where the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes and then the model is finished by adding non-key attributes. Other experts argue that in practice, using a phased approach is impractical because it requires too many meetings with the end-users. The sequence used for this document is:

- Identification of data objects and relationships
- Drafting the initial ER diagram with entities and relationships
- Refining the ER diagram
- Add key attributes to the diagram
- Adding non-key attributes
- Diagramming Generalization Hierarchies
- Validating the model through normalization
- Adding business and integrity rules to the Model

In practice, model building is not a strict linear process. As noted above, the requirements analysis and the draft of the initial ER diagram often occur simultaneously. Refining and validating the diagram may uncover problems or missing information which require more information gathering and analysis. Data modeling must be preceded by planning and analysis. Planning defines the goals of the database, explains why the goals are important and sets out the path by which the goals will be reached. Analysis involves determining the requirements of the

database. This is typically done by examining existing documentation and interviewing users.

An effective data model completely and accurately represents the data requirements of the end users. It is simple enough to be understood by the end user yet detailed enough to be used by a database designer to build the database. The model eliminates redundant data, it is independent of any hardware and software constraints and can be adapted to changing requirements with a minimum of effort.

Data modeling is a bottom up process. A basic model, representing entities and relationships, is developed first. Then detail is added to the model by including information about attributes and business rules.

3.2.2.1 Identifying Data Objects and Relationships

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirements analysis for the purpose of:

- Classifying data objects as either entities or attributes
- Identifying and defining relationships between entities
- Naming and defining identified entities, attributes and relationships

Documenting this information in the data document to accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents and design documents from the current information system.

While the definitions of the constructs in the ER Model are simple, the model does not address the fundamental issue of how to identify them. Some commonly given guidelines are:

- Entities contain descriptive information
- Attributes either identify or describe entities
- Relationships are associations between entities

These guidelines are discussed below.

- Entities
- Attributes
 - Validating Attributes
 - Derived Attributes and Code Values
- Relationships
- Naming Data Objects

3.2.2.1.1 Entities

There are various definitions of an entity:

"Any distinguishable person, place, thing, event or concept about which information is kept"

"A thing which can be distinctly identified"

"Any distinguishable object that is to be represented in a database"

"...anything about which we store information (e.g. supplier, machine tool, employee, utility pole, airline seat, etc.). For each entity type, certain attributes are stored".

3.2.2.1.2 Attributes

Attributes are data objects that either identify or describe entities. Attributes that identify entities are called *key attributes*. Attributes that describe an entity are called non-key attributes. Key attributes will be discussed in detail in a latter section.

The process for identifying attributes is similar except now you want to look for and extract those names that appear to be descriptive noun phrases. Validating Attributes values should be *atomic*, that is, present a single fact. Having disaggregated data allows simpler programming, greater reusability of data, and easier implementation of changes. Normalization also depends upon the "single fact" rule being followed.

3.2.2.1.3 Relationships

Relationships are associations between entities. Typically, a relationship is indicated by a verb connecting two or more entities.

3.2.2.1.4 Naming Data Objects

The names should have the following properties:

- Unique
- Have meaning to the end-user
- Contain the minimum number of words needed to uniquely and accurately describe the object

For entities and attributes, names are singular nouns while relationship names are typically verbs. Some authors advise against using abbreviations or acronyms because they might lead to confusion about what they mean. Other believes using abbreviations or acronyms are acceptable provided that they are universally used and understood within the organization. The first step in creating the data model is to analyze the information gathered during the requirements analysis with the goal of identifying and classifying data objects and relationships. The next step is developing the Basic Schema.

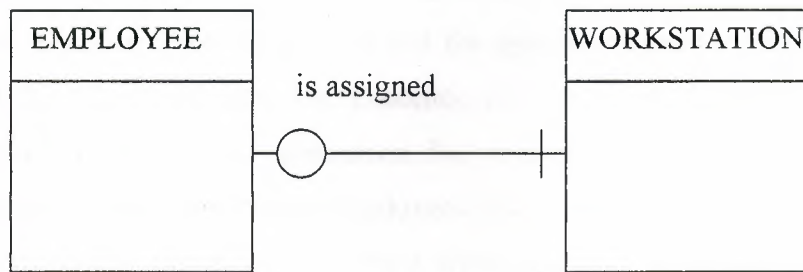
3.2.2.2 Developing the Basic Schema

Once entities and relationships have been identified and defined, the first draft of the entity relationship diagram can be created. This section introduces the ER diagram by demonstrating how to diagram binary relationships. Recursive relationships are also shown.

Binary Relationships

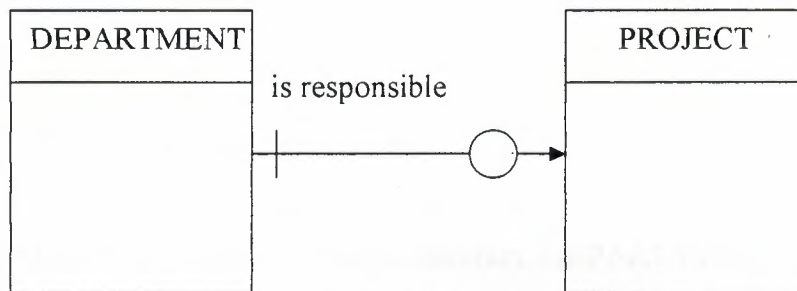
Figure 3.2 shows examples of how to diagram one-to-one, one-to-many and many-to-many relationships.

A. ONE - TO - ONE



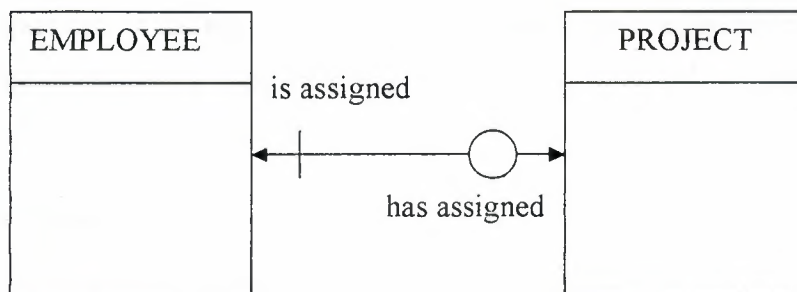
Every employee is assigned one workstation ; not all workstations are assigned to employees.

B. ONE - TO - MANY



A department may be responsible for many projects out each project is the responsibility of one department.

C. MANY - TO - MANY



Employees may be assigned to many projects; every project has assigned at least one employee.

Figure 3.2 Example of Binary Relationships

3.2.2.2.1 One-To-One

Figure 3.2.A shows an example of a one-to-one diagram. Reading the diagram from left to right represents the relationship every employee is assigned a workstation. Because every employee must have a workstation, the symbol for mandatory existence in this case the crossbar is placed next to the WORKSTATION entity.

Reading from right to left, the diagram shows that not all workstation are assigned to employees. This condition may reflect that some workstations are kept for spares or for loans. Therefore, we use the symbol for optional existence, the circle, next to EMPLOYEE. The cardinality and existence of a relationship must be derived from the "business rules" of the organization. For example, if all workstations owned by an organization were assigned to employees, then the circle would be replaced by a crossbar to indicate mandatory existence. One-to-one relationships are rarely seen in "real-world" data models. Some parishioners advise that most one-to-one relationships should be collapsed into a single entity or converted to a generalization hierarchy.

3.2.2.2.2 One-To-Many

Figure 3.2.B shows an example of a one-to-many relationship between DEPARTMENT and PROJECT. In this diagram, DEPARTMENT is considered the parent entity while PROJECT is the child. Reading from left to right, the diagram represents departments may be responsible for many projects. The optionality of the relationship reflects the "business rule" that not all departments in the organization will be responsible for managing projects. Reading from right to left, the diagram tells us that every project must be the responsibility of exactly one department.

3.2.2.2.3 Many-To-Many

Figure 3.2.C shows a many-to-many relationship between EMPLOYEE and PROJECT. An employee may be assigned to many projects; each project must have many employee. Note that the association between EMPLOYEE and PROJECT is optional because, at a given time, an employee may not be assigned to a project. However, the relationship between PROJECT and EMPLOYEE is mandatory because a project must have at least two employees assigned. Many-To-Many relationships can be used in the initial drafting of the model but eventually must be transformed into two one-to-many relationships. The transformation is required because many-to-many relationships cannot be represented by the relational model.

The process for resolving many-to-many relationships is discussed in the next section.

3.2.2.3 Define Keys

3.2.2.3.1 Primary and Foreign Keys

Primary and foreign keys are the most basic components on which relational theory is based. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an association between two entities. The next step in building the basic data model to

- Identify and define the primary key attributes for each entity
- Validate primary keys and relationships
- Migrate the primary keys to establish foreign keys

3.2.2.3.2 Define Primary Key Attributes

Attributes are data items that describe an entity. An *attribute instance* is a single value of an attribute for an instance of an entity. For example, Name and hire date are attributes of the entity EMPLOYEE. "Jane Hathaway" and "3 March 1989" are instances of the attributes name and hire date. The *primary key* is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.

To qualify as a primary key for an entity, an attribute must have the following properties:

- it must have a non-null value for each instance of the entity
- the value must be unique for each instance of an entity
- the values must not change or become null during the life of each entity instance

In some instances, an entity will have more than one attribute that can serve as a primary key. Any key or minimum set of keys that could be a primary key is called a *candidate key*. Once candidate keys are identified, choose one and only one primary key for each entity. Choose the identifier most commonly used by the user as long as

it conforms to the properties listed above. Candidate keys which are not chosen as the primary key are known as alternate keys. An example of an entity that could have several possible primary keys is Employee. Let's assume that for each employee in an organization there are three candidate keys: Employee ID, Social Security Number, and Name.

Name is the least desirable candidate. While it might work for a small department where it would be unlikely that two people would have exactly the same name, it would not work for a large organization that had hundreds or thousands of employees. Moreover, there is the possibility that an employee's name could change because of marriage. Employee ID would be a good candidate as long as each employee was assigned a unique identifier at the time of hire. Social Security would work best since every employee is required to have one before being hired.

3.2.2.3.3 Composite Keys

Sometimes it requires more than one attribute to uniquely identify an entity. A primary key that made up of more than one attribute is known as a *composite key*. Figure 3.3 shows an example of a composite key. Each instance of the entity Work can be uniquely identified only by a composite key composed of Employee ID and Project ID.

WORK		
<u>Employee ID</u>	<u>Project ID</u>	<u>Hours Worked</u>
01	01	200
01	02	120
02	01	50
02	03	120
03	03	100
03	04	200

Figure 3.3 Example of Composite Key

Next step is normalization which simply stated, normalization is the process of removing redundant data from relational tables by decomposing (splitting) a relational table into smaller tables by projection.

3.2.3 Normalization

3.2.3.1 Overview

Normalization is a design technique that is widely used as a guide in designing relational databases. Normalization is essentially a two step process that puts data into tabular form by removing repeating groups and then removes duplicated data from the relational tables. Normalization theory is based on the concepts of normal forms. A relational table is said to be a particular normal form if it satisfied a certain set of constraints. There are currently five normal forms that have been defined. In this section, I will cover the first three normal forms that were defined by E. F. Codd.

Normalization which simply stated the process of removing redundant data from relational tables by decomposing (splitting) a relational table into smaller tables by projection. The goal is to have only primary keys on the left hand side of a functional dependency. In order to be correct, decomposition must be loss less. That is, the new tables can be recombined by a natural join to recreate the original table without creating any spurious or redundant data.

3.2.3.2 Basic Concepts

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the third normal form (3NF). A relational table is in 3NF if and only if all non-key columns are (a) mutually independent and (b) fully dependent upon the primary key. Mutual independence means that no non-key column is dependent upon any combination of the other columns. The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF. In order to better understand the 2NF and higher forms, it is necessary to understand the concepts of functional dependencies and loss less decomposition.

3.2.3.3 Sample Data

Data taken from Date [Date90] is used to illustrate the process of normalization. A company obtains parts from a number of suppliers. Each supplier is located in one city. A city can have more than one supplier located there and each city has a status code associated with it. Each supplier may provide many parts. The company creates a simple relational table to store this information that can be expressed in relational notation as:

FIRST (s#, status, city, p#, qty) where

s# supplier identification number (this is the primary key)

status status code assigned to city

city name of city where supplier is located

p# part number of part supplied

qty> quantity of parts supplied to date

In order to uniquely associate quantity supplied (qty) with part (p#) and supplier (s#), a composite primary key composed of s# and p# is used.

3.2.3.4 Functional Dependencies

The concept of functional dependencies is the basis for the first three normal forms. A column, Y, of the relational table R is said to be functionally dependent upon column X of R if and only if each value of X in R is associated with precisely one value of Y at any given time. X and Y may be composite. Saying that column Y is functionally dependent upon X is the same as saying the values of column X identify the values of column Y. If column X is a primary key, then all columns in the relational table R must be functionally dependent upon X.

A shorthand notation for describing a functional dependency is:

$R.x \longrightarrow R.y$

which can be read as in the relational table named R, column x functionally determines (identifies) column y.

Full functional dependence applies to tables with composite keys. Column Y in relational table R is fully functional on X of R if it is functionally dependent on X and not functionally dependent upon any subset of X. Full functional dependence means that when a primary key is composite, made of two or more columns, then the other columns must be identified by the entire key and not just some of the columns that make up the key.

3.2.3.5 First Normal Form

A relational table, by definition, is in first normal form. All values of the columns are atomic. That is, they contain no repeating values. Figure 3.4 shows the table FIRST in 1NF.

FIRST

s#	status	city	q#	qty
s1	20	London	p1	300
s1	20	London	p2	200
s1	20	London	p3	400
s1	20	London	p4	200
s1	20	London	p5	100
s1	20	London	p6	100
s2	10	Paris	p1	300
s2	10	Paris	p2	400
s3	10	Paris	p2	200
s4	20	London	p2	200
s4	20	London	p4	300
s4	20	London	p5	400

Figure 3.4 Table in 1NF

Although the table FIRST is in 1NF it contains redundant data. For example, information about the supplier's location and the location's status have to be repeated for every part supplied. Redundancy causes what are called *update anomalies*. Update anomalies are problems that arise when information is inserted, deleted or updated. For example, the following anomalies could occur in FIRST:

INSERT. The fact that a certain supplier (s5) is located in a particular city (Athens) cannot be added until they supplied a part.

DELETE. If a row is deleted, then not only is the information about quantity and part lost but also information about the supplier.

UPDATE. If supplier s1 moved from London to New York, then six rows would have to be updated with this new information.

3.2.3.6 Second Normal Form

The definition of second normal form states that only tables with composite primary keys can be in 1NF but not in 2NF.

A relational table is in second normal form 2NF if it is in 1NF and every non-key column is fully dependent upon the primary key.

That is, every non-key column must be dependent upon the entire primary key.

FIRST is in 1NF but not in 2NF because status and city are functionally dependent upon only on the column s# of the composite key (s#, p#). This can be illustrated by listing the functional dependencies in the table:

s# \longrightarrow city, status

city \longrightarrow status

(s#,p#) \longrightarrow qty

The process for transforming a 1NF table to 2NF is:

Identify any determinants other than the composite key, and the columns they determine.

Create and name a new table for each determinant and the unique columns it determines.

Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.

Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.

The original table may be renamed to maintain semantic meaning.

To transform FIRST into 2NF we move the columns s#, status, and city to a new table called SECOND. The column s# becomes the primary key of this new table.

The results are shown below in Figure 3.5.

SECOND

s#	status	city
s1	20	London
s2	10	Paris
s3	10	Paris
s4	20	London
s5	30	Athens

PARTS

s#	p#	qty
s1	P1	300
s1	P2	200
s1	P3	400
s1	P4	200
s1	P5	100
s1	P6	100
s2	P1	300
s2	P2	400
s3	P2	200
s4	P2	200
s4	P4	300
s4	P5	400

Figure 3.5 Tables in 2NF

Tables in 2NF but not in 3NF still contain modification anomalies. In the example of SECOND, they are:

INSERT. The fact that a particular city has a certain status (Rome has a status of 50) cannot be inserted until there is a supplier in the city.

DELETE. Deleting any row in SUPPLIER destroys the status information about the city as well as the association between supplier and city.

3.2.3.7 Third Normal Form

The third normal form requires that all columns in a relational table are dependent only upon the primary key. A more formal definition is:

A relational table is in third normal form (3NF) if it is already in 2NF and every non-key column is none transitively dependent upon its primary key. In other words, all non key attributes are functionally dependent only upon the primary key.

Table PARTS is already in 3NF. The non-key column, qty, is fully dependent upon the primary key (s#, p#). SUPPLIER is in 2NF but not in 3NF because it contains a *transitive dependency*. A transitive dependency occurs when a non-key column that is a determinant of the primary key is the determinate of other columns. The concept of a transitive dependency can be illustrated by showing the functional dependencies in SUPPLIER:

SUPPLIER.s# —> SUPPLIER.status
 SUPPLIER.s# —> SUPPLIER.city
 SUPPLIER.city —> SUPPLIER.status

Note that SUPPLIER.status is determined both by the primary key s# and the non-key column city. The process of transforming a table into 3NF is:

Identify any determinants, other the primary key, and the columns they determine.

Create and name a new table for each determinant and the unique columns it determines.

Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.

Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.

The original table may be renamed to maintain semantic meaning.

To transform SUPPLIER into 3NF, we create a new table called CITY_STATUS and move the columns city and status into it. Status is deleted from the original table, city is left behind to serve as a foreign key to CITY_STATUS, and the original table is renamed to SUPPLIER_CITY to reflect its semantic meaning. The results are shown in Figure 3.6 below.

SUPPLIER_CITY		CITY_STATUS	
s#	city	city	status
s1	London	London	20
s2	Paris	Paris	10
s3	Paris	Athens	30
s4	London	Rome	50
s5	Athens		

Figure 3.6 Tables in 3NF

The results of putting the original table into 3NF have created three tables. These can be represented in SQL as:

PARTS (#s, p#, qty)

Primary Key (s#,#p)

Foreign Key (s#) references SUPPLIER_CITY.s#

SUPPLIER_CITY(s#, city)

Primary Key (s#)

Foreign Key (city) references CITY_STATUS.city

CITY_STATUS (city, status)

Primary Key (city)

The advantage of having relational tables in 3NF is that it eliminates redundant data which in turn saves space and reduces manipulation anomalies. For example, the improvements to our sample database are:

After 3NF, all normalization problems involve only tables which have three or more columns and all the columns are keys. Many practitioners argue that placing entities in 3NF is generally sufficient because it is rare that entities that are in 3NF are not also in 4NF and 5NF. They further argue that the benefits gained from transforming entities into 4NF and 5NF are so slight that it is not worth the effort. However, advanced normal forms are presented because there are cases where they are required.

3.2.3.8 Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) is a more rigorous version of the 3NF deal with relational tables that had (a) multiple candidate keys, (b) composite candidate keys, and (c) candidate keys that overlapped.

BCNF is based on the concept of determinants. A determinant column is one on which some of the columns are fully functionally dependent.

A relational table is in BCNF if and only if every determinant is a candidate key.

3.2.3.9 Fourth Normal Form

A relational table is in the *fourth normal form* (4NF) if it is in BCNF and all multi valued dependencies are also functional dependencies.

Fourth normal form (4NF) is based on the concept of *multi valued dependencies* (MVD). A Multi valued dependency occurs when in a relational table containing at least three columns, one column has multiple rows whose values match a value of a single row of one of the other columns. A more formal definition given by Date is:

Given a relational table R with columns A, B, and C then

$R.A \twoheadrightarrow R.B$ (column A multi determines column B) is true if and only if the set of B-values matching a given pair of A-values and C-values in R depends only on the A-value and is independent of the C-value.

MVD always occur in pairs. That is $R.A \twoheadrightarrow R.B$ holds if and only if $R.A \twoheadrightarrow R.C$ also holds.

Suppose that employees can be assigned to multiple projects. Also suppose that employees can have multiple job skills. If we record this information in a single table, all three attributes must be used as the key since no single attribute can uniquely identify an instance.

The relationship between emp# and prj# is a multi valued dependency because for each pair of emp#/skill values in the table, the associated set of prj# values is determined only by emp# and is independent of skill. The relationship between emp# and skill is also a multi valued dependency, since the set of Skill values for an emp#/prj# pair is always dependent upon emp# only.

To transform a table with multi valued dependencies into the 4NF move each MVD pair to a new table. The result is shown in Figure 3.7.

EMPLOYEE_PROJECT

emp#	prj#
1211	1
1211	5

EMPLOYEE_SKILL

emp#	skill
1211	Analysis
1211	Design
1211	Program

Figure 3.7 Tables in 4NF

3.2.3.10 Fifth Normal Form

A table is in the *fifth normal form* (5NF) if it cannot have a loss less decomposition into any number of smaller tables.

While the first four normal forms are based on the concept of functional dependence, the fifth normal form is based on the concept of join dependence. Join dependency means that an table, after it has been decomposed into three or more smaller tables, must be capable of being joined again on common keys to form the original table. Stated another way, 5NF indicates when an entity cannot be further decomposed.

5NF is complex and not intuitive. Most experts agree that tables that are in the 4NF are also in 5NF except for "pathological" cases. Theory suggests that true many-to-many ternary relations are one such case.

Adding an instance to a table that is not in 5NF creates spurious results when the tables are decomposed and then rejoined. For example, let's suppose that we have an employee who uses design skills on one project and programming skills on another.

This information is shown below.

emp#	prj#	skill
1211	11	Design
1211	28	Program

Next we add an employee (1544) who uses programming skills on Project 11.

emp#	prj#	skill
1211	11	Design
1211	28	Program
1544	11	Program

Next, we project this information into three tables as we did above. However, when we rejoin the tables, the recombined table contains spurious results.

emp#	prj#	skill	
1211	11	Design	
1211	11	Program	<<—spurious data
1211	28	Program	
1544	11	Design	<<—spurious data
1544	11	Program	

By adding one new instance to a table not in 5NF, two false assertions were stated:

Assertion 1

Employee 1211 has been assigned to Project 11.

Project 11 requires programming skills.

Therefore, Employee 1211 must use programming skills while assigned to Project 11.

Assertion 2

Employee 1544 has been assigned to project 11.

Project 11 needs Design skills.

Therefore, Employee 1544 must use Design skills in Project 11.

3.3 Architectures of Database System

There are two main architecture of database system

- Centralized Database System
- Distributed Database System

3.3.1 Centralized Database System

A centralized database system is a system that keeps the data in one single database at one single location. In a centralized database system, a single machine called a database server hosts the DBMS and the database.

Multiple users or client workstations can work simultaneously on a centralized database system using the Client/Server configuration, or the Intranet configuration if

- An underlying LAN (Local Area Network) is available (LANs can span one or few adjacent buildings)
- An underlying WAN (Wide Area Network) is available (WANs can span all Lebanon)

The client/server architecture is a very successful and popular one as it balances the processing load between the client machine and the server machine.

The ongoing growth of Internet and intranet applications has refocused attention on centralized databases. In such configuration, the bulk of the processing does not lie on the client machine, but rather on the machine hosting the Application Server and the database server machine.

The main disadvantage of centralized database systems is that of single point of failure. When the database fails, work of all users is interrupted. Also in the case where WANs are used, failure of part of the network means the interruption of work at the remote location.

Therefore, centralized databases are easier to manage, maintain and control for security purposes. They should be the selection of choice if there is no need for a more complex architecture.

3.3.2 Distributed Database System

The main difference between centralized and distributed database systems is that, in the former, the data resides in one single location, whereas in the latter, the data resides in several locations or on multiple servers at the same location.

The distribution of the data across locations should be transparent to the user who continues to use the software application interface from his/her computer.

Distributed database systems involve many complex issues such as transparency, transaction management, optimization, data fragmentation and replication. Their design requires a high level of sophistication and competence from the supplier and their management requires an experienced Database Administrator.

The issues summarized below must be assessed during the selection process.

It is recommended that distributed architectures be used strictly on a per need basis because of the complexity of their design and maintenance.

3.4 Summary

In conclusion, data model is a conceptual representation of the data structure that are required by a database. There are two major methodologies used to create a data model: the Entity-Relationship(ER) approach and the Object Model. I used the Entity-Relationship approach. There are also two main architectures of database system: Centralized Database System and Distributed Database System. I used the Centralized Database System.

CHAPTER 4

HOSPITAL MANAGEMENT SYSTEM AND DATABASE STRUCTURE OF HOSPITAL

4 Overview

Hospital Management System (HMS) is powerful, flexible and easy to use and has designed and developed to deliver real conceivable benefits to hospitals and clinics. And more importantly it is backed by reliable and dependable support.

4.1 Hospital Management System Solution for Hospitals

HMS designed for multi specialty hospitals, to cover a wide range of Hospital administration and management processes. It is an integrated client server application uses Delphi as the front-end for Graphical User Interface and MS SQL as the database.

4.2 Hospital Management System – Registration

The Registration module is an integrated patient management system, which captures complete and relevant patient information. The system automates the patient administration functions to have better and efficient patient care process.

- Register Patient (Inpatient and Outpatient) Detail

It provides for all enquiries about the patient.

4.3 Hospital Management System - Doctor

This module supports doctors to take better and timely consultation decisions by providing instant access to comprehensive patient information.

- Medical Details

- Diagnosis Details
- Doctor's Diagnosis Details

Further more, Confidentiality of Doctors Observation, Previous History of Patients, Request for Investigations and so on, are the special features in Doctors Observation screen.

4.4 Hospital Management System - Wards

Wards module deals with the automation for inpatient in wards in hospital. It includes the following things:

- Manage the patient in wards
- Organization for beds
- Organization for patients according to their disease

4.5 Hospital Management System – Rooms

Rooms module deals with the automation for inpatient in rooms in hospital. It includes the following things:

- Manage the patient in rooms
- Organizing rooms
- Organizing for patients according to their disease

4.6 Hospital Management System - Pharmacy

Pharmacy module deals with the automation of general workflow and administration management process of a pharmacy. The pharmacy module is equipped with bar coding facility, which makes the delivery of medical items to the patient more efficient. This module deals with the activities such as:

- Stock management
- Medicine Details

4.7 Hospital Management System - User Manager

The User Manager module basically deals with administrative controlling the access to the information available in the application. It also deals with the System Related Activity like modifying, creating, deleting and other management stuff. It includes the all information about registration, doctor, wards, rooms and pharmacy modules.

4.8 Benefit of Hospital Network

In hospital, new patient should register own information to the reception. Then patient can visit to doctors. In doctors room patient illness become perceptible. That time patient becomes inpatient or outpatient. If patient is outpatient then patient can go after examine. If patient is inpatient then patient should visit related department. In hospital, inpatient place is consisting of patient's illness. If patient is reside in room then first patient visit rooms receptionist. If patient is reside in ward then first patient visit wards receptionist. Doctor visit patients with some period. It is depend upon patient diseases kind. When doctor finish examine then all inpatient information register to the inpatient record by rooms or wards receptionist. Rooms and Wards receptionist duty is just organize place for inpatient and register doctor report to the inpatient record.

The below figure 4.1 describe the general structure of Hospital Network

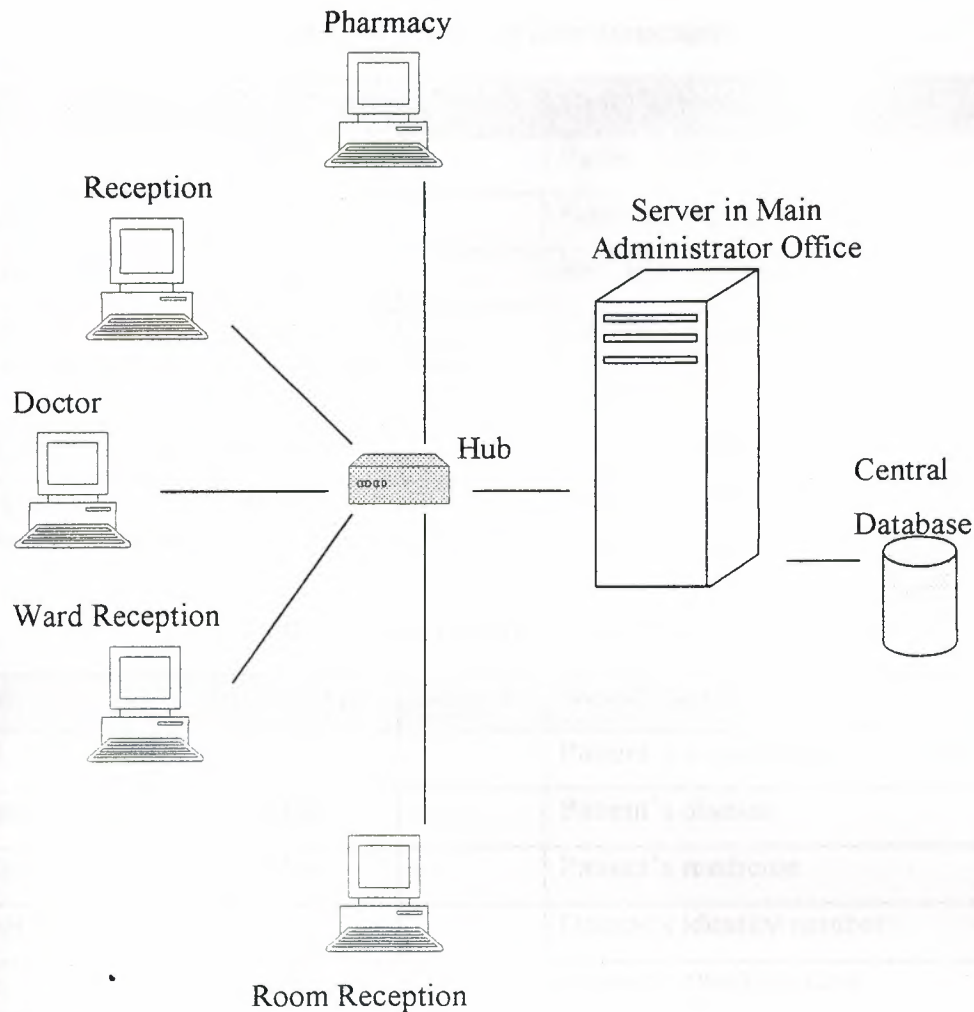


Figure 4.1 The general structure of Hospital Network

4.9 Database Design for Hospital Management System

For Hospital Management System Database, I used MS SQL. Database contain following tables: Patient, Outpatient, Inpatient, Room, Ward, Bed Availability, Doctor, Department, Inpatient Disease Detail and Pharmacy. All tables reside on server machine. The structures of the database are given in tables 4.1 ÷ 4.10.

Table 4.1 Patient Table Structure

Field	Data Type	Default	Description
SSN	varchar	-	Patient's social security number.
l_name	varchar	-	Patient's last name.
f_name	varchar	-	Patient's first name.
Phone	varchar	-	Patient's phone number.
Sex	char	-	Patient's sex.
d_of_birth	datetime	-	Patient's date of birth.
address	varchar	-	Patient's address.

Table 4.2 Outpatient Table Structure

Field	Data Type	Default	Description
SSN	varchar	-	Patient's social security number.
disease	varchar	-	Patient's disease.
medicine	varchar	-	Patient's medicine.
doctor_id	varchar	-	Doctor's identity number.
date	datetime	-	Patient's checking date.

Table 4.3 Inpatient Table Structure

Field	Data Type	Default	Description
SSN	varchar	-	Patient's social security number.
patient_no	numeric	-	Patient's registration number.
admission_date	datetime	-	Patient's admission date.
discharge_date	datetime	-	Patent's discharge date.
ward_id	varchar	-	Patient's ward id.
Bed_no	integer	-	Patient's bed number.
room_no	varchar	-	Patient's room number.
Dep_id	varchar	-	Patient's department id.

Table 4.4 Room Table Structure

Field	Data Type	Default	Description
room_no	varchar	-	Room number.
Dep_id	varchar	-	Department id.
status	char	-	Status of room.

Table 4.5 Ward Table Structure

Field	Data Type	Default	Description
ward_id	varchar	-	Ward id.
Dep_id	varchar	-	Department id.
ward_name	varchar	-	Ward name.
total_bed	varchar	-	Total bed.

Table 4.6 Bed Availability Table Structure

Field	Data Type	Default	Description
ward_id	varchar	-	Ward id.
Bed_no	integer	-	Bed number.
Dep_id	varchar	-	Department id.
status	char	-	Status of bed.

Table 4.7 Doctor Table Structure

Field	Data Type	Default	Description
doctor_id	varchar	-	Doctor id.
l_name	varchar	-	Doctor's last name.
f_name	varchar	-	Doctor's first name.
phone	varchar	-	Doctor's phone.
Sex	char	-	Doctor's sex.
Dep_id	varchar	-	Doctor's department id.

Table 4.8 Department Table Structure

Field	Data Type	Default	Description
dep_id	varchar	-	Department id.
dep_name	varchar	-	Department name.
dep_des	varchar	-	Department description.

Table 4.9 Inpatient Table Structure

Field	Data Type	Default	Description
SSN	varchar	-	Inpatient's social security number.
patient_no	numeric	-	Inpatient's number.
date_of_check	datetime	-	Inpatient's date of check.
time_of_check	varchar	-	Inpatient's time of check.
doctor_id	varchar	-	Doctor id.
disease	varchar	-	Inpatient's disease.
medicine	varchar	-	Inpatient's medicine.
response	varchar	-	Inpatient's response.

Table 4.10 Pharmacy Table Structure

Field	Data Type	Default	Description
medicine_id	varchar	-	Medicine id.
medicine_name	varchar	-	Medicine name.
price	varchar	-	Medicine price.
quantity	numeric	-	Medicine quantity.

4.10 Adding Patient Algorithm

The below diagram show in Figure 4.2 show the operation or step of Adding Patient.

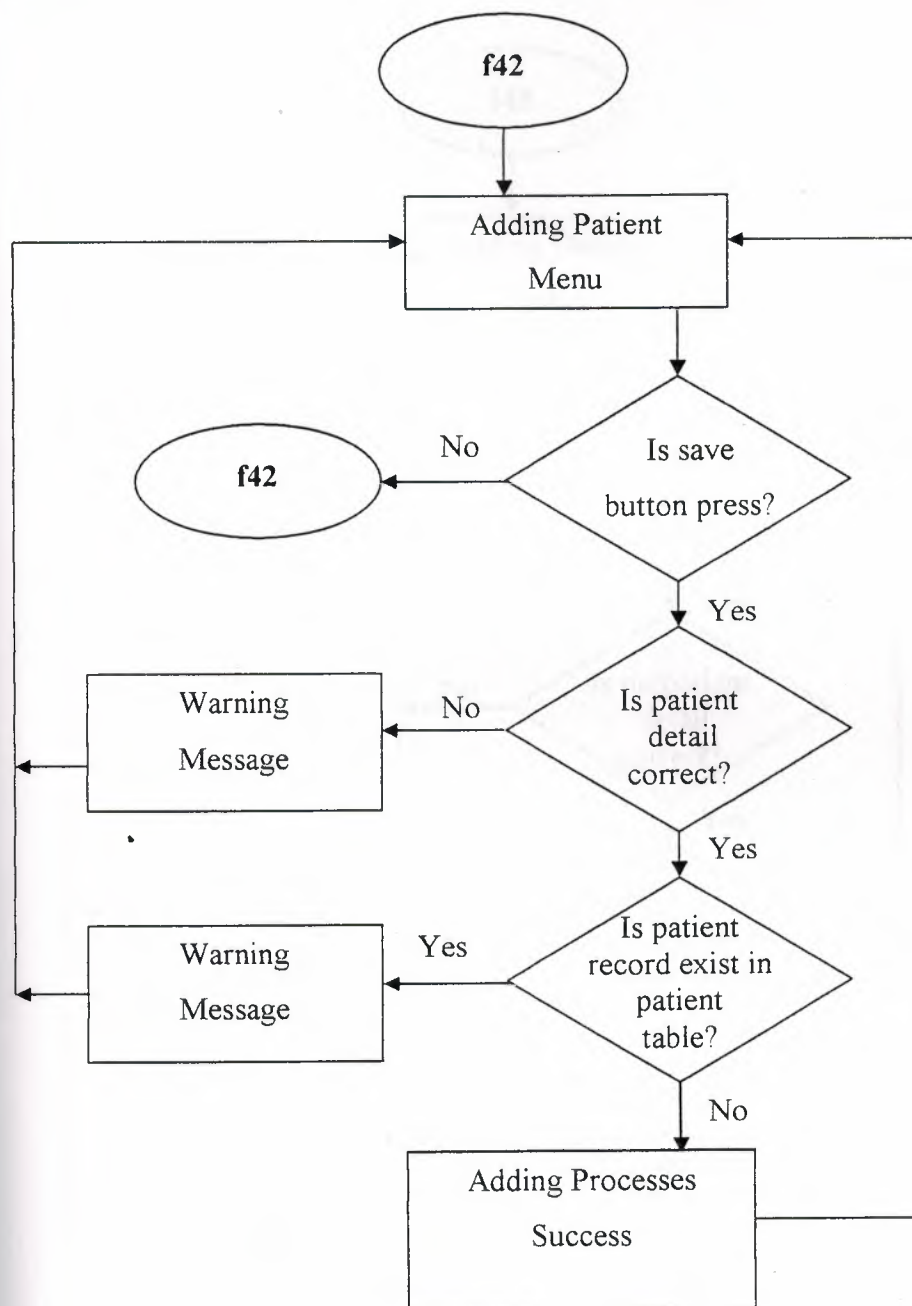


Figure 4.2 Adding Patient flow diagram

4.11 Adding Outpatient Algorithm

The below diagram show in Figure 4.3 show the operation or step of Adding Outpatient.

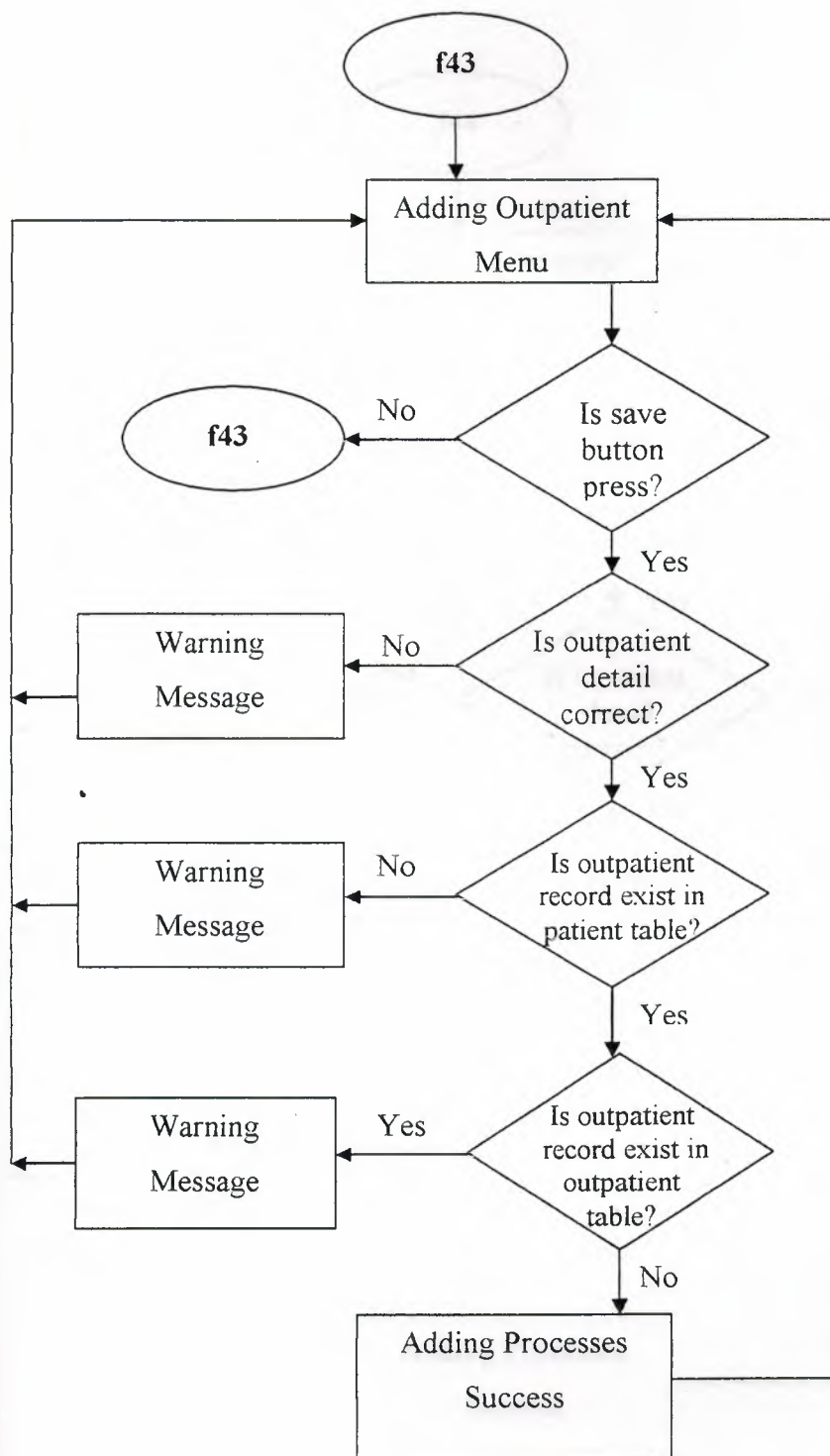


Figure 4.3 Adding Outpatient flow diagram

4.12 Adding Inpatient Algorithm

The below diagram show in Figure 4.4 show the operation or step of Adding Inpatient.

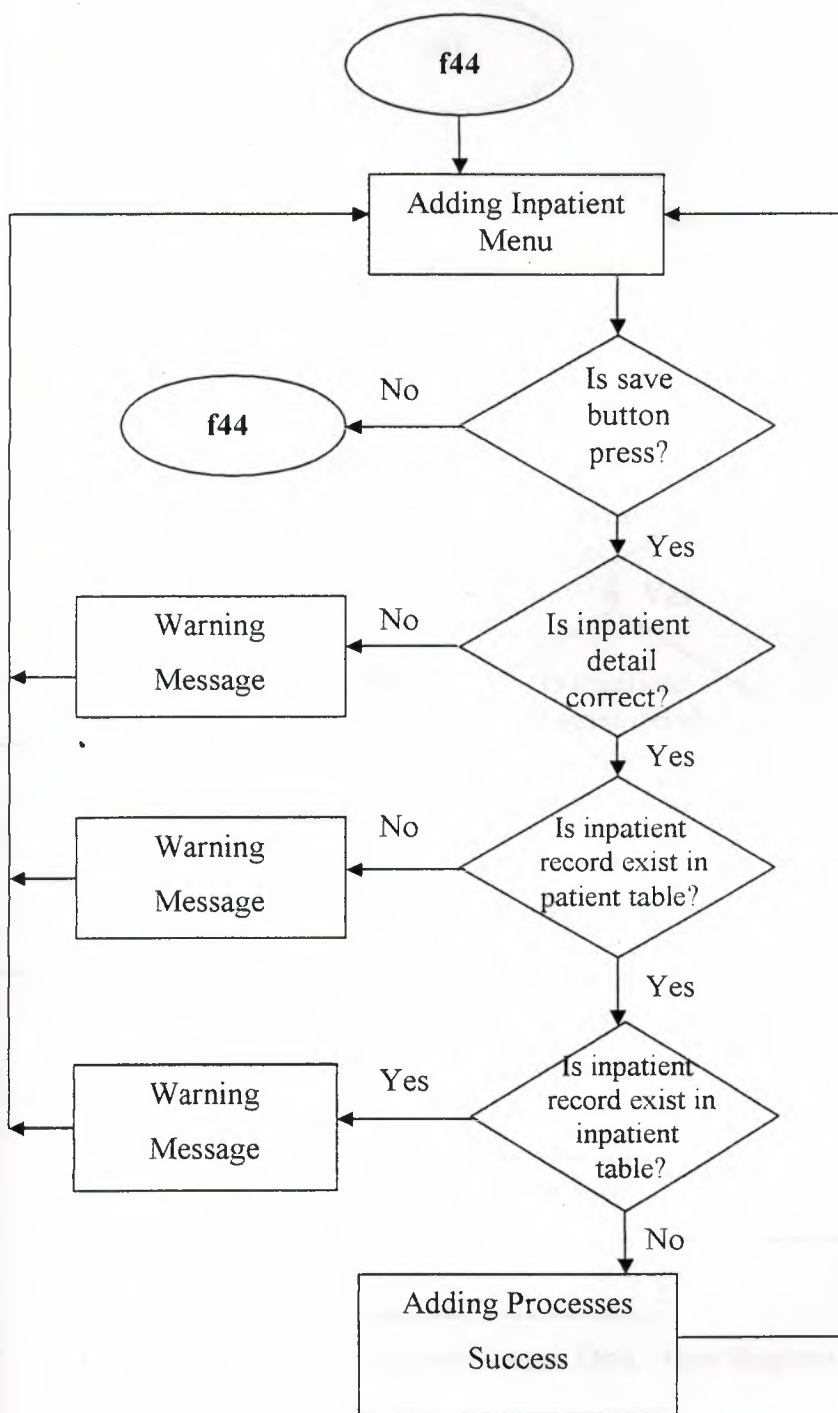


Figure 4.4 Adding Inpatient flow diagram

4.13 Adding Inpatient Disease Detail Algorithm

The below diagram show in Figure 4.5 show the operation or step of Adding Inpatient Disease Detail.

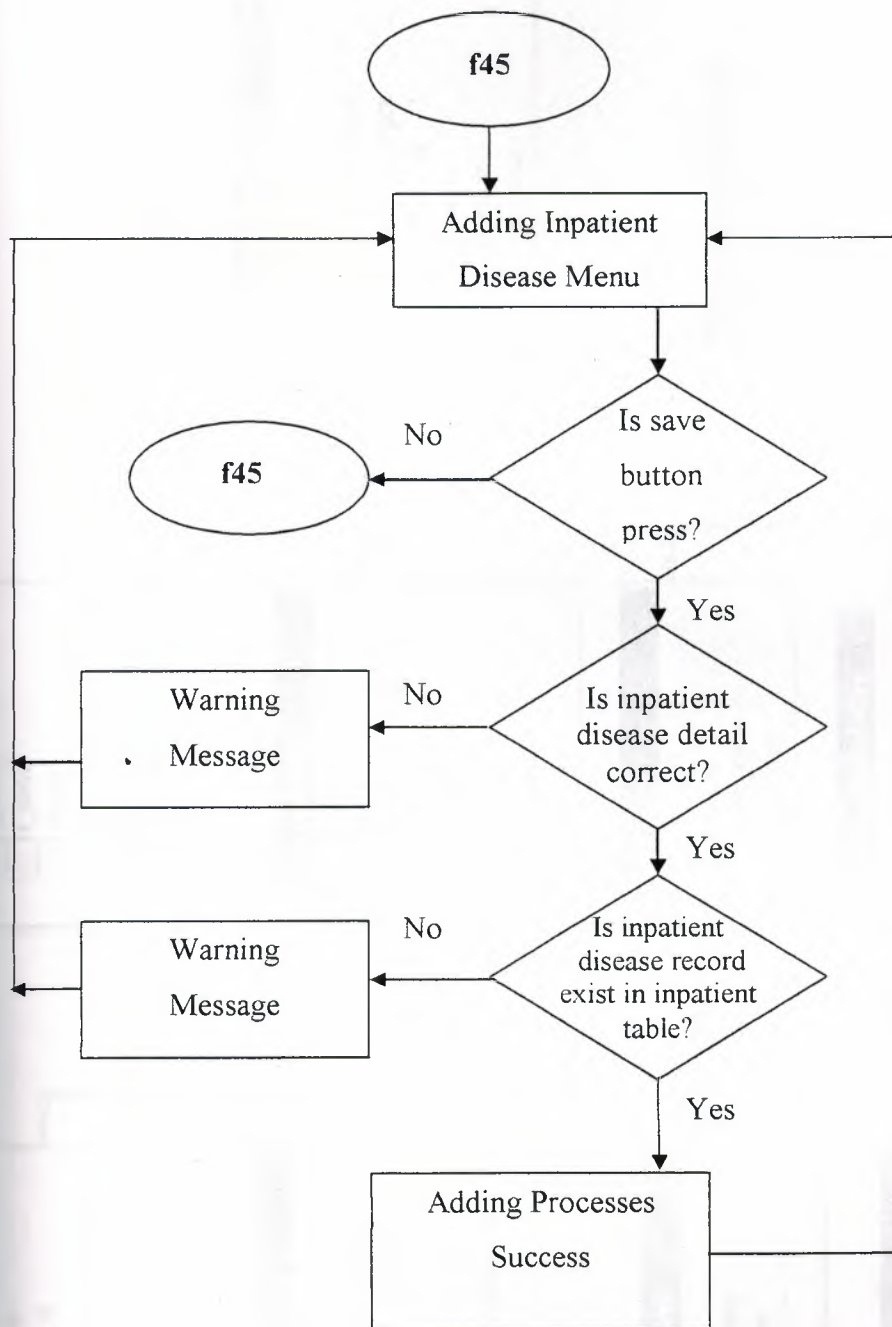


Figure 4.5 Adding Inpatient Disease Detail flow diagram

4.14 Relationship of Tables (MS SQL / hosdata1)

The below diagram show in Figure 4.6 show the relationship of tables.

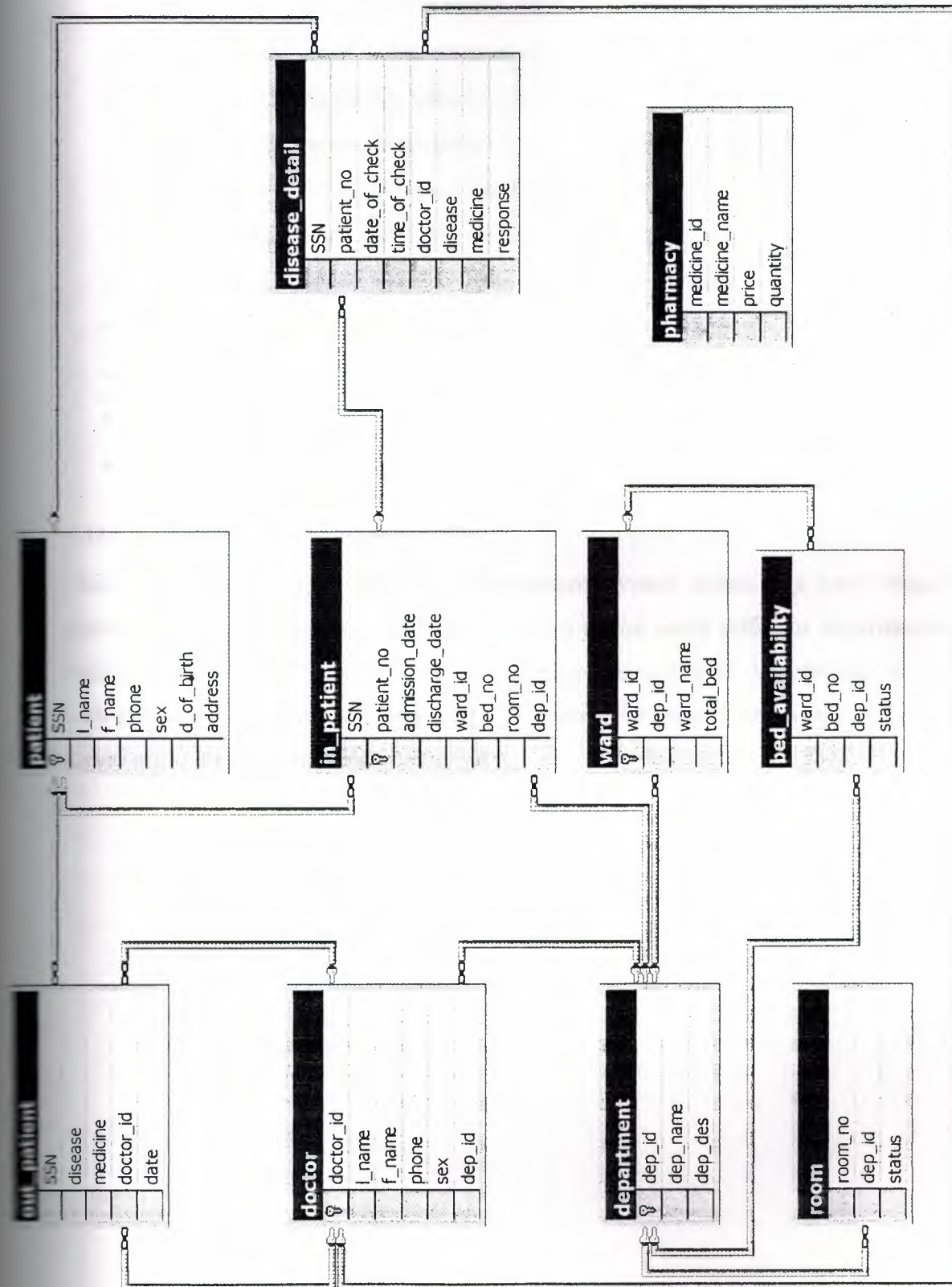


Figure 4.6 Relationship of Tables

The Relationship of Tables are given below;

- Outpatient and Patient tables relationship is one to one.
- Inpatient and Patient tables relationship is one to one.
- Disease detail and Patient tables relationship is one to one.
- Inpatient and Disease detail tables relationship is one to many.
- Outpatient and Doctor tables relationship is one to many.
- Inpatient and Department tables relationship is one to many.
- Doctor and Department tables relationship is one to many.
- Room and Department tables relationship is one to many.
- Bed availability and Department tables relationship is one to many.
- Ward and Bed availability tables relationship is one to many.
- Disease detail and Doctor tables relationship is one to many.

4.15 Summary

This chapter describes all Hospital Management System which is a very simple system used in a variety of ways that can help in the many different departments available at a hospital. I have created ten tables representing all departments linked with each other according to their relations. Creating the tables with Microsoft's SQL server made it more reliable and accurate.

CHAPTER 5

IMPLEMENTATION OF HOSPITAL MANAGEMENT SYSTEM IN DELPHI AND USER INTERFACE

5 Overview

This chapter focus on client applications and server application. The chapter describes how to connect client and server via network. This chapter would also be called the user manual where it describes thoroughly how to use all applications within the user interface.

5.1 Administrator Application

The Administrator application runs on the server machine. This application has the ability to manage everything related to the Hospital Management Database. Administrators can delete, modify and search any record related to any application in the Hospital Management Database. The administrator has the rights and authority to make changes whenever and wherever needed.

5.1.1 Administrator Application Main Form

As we can see in the main form interface the list of links lined up with labels representing each task to their form when pressed. If no choice could be reached there would always be the exit button to leave or exit the application. The below figure show in Figure 5.1 show the Administrator Application Main Form.

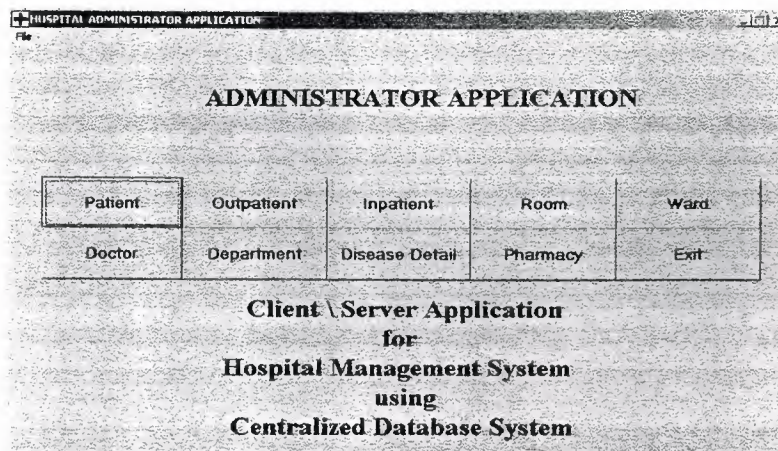


Figure 5.1 Administrator Application Main Form

5.1.2 Administrator Application \ Patient Menu

From here we can add and modify and delete all patients records, and even if we were to search for a record that would be possible by using the search button. The only difference between the administrator's interface and the rest is that this includes all transactions while other menus differ according to their users. For example in the doctor's menu you can find the search button. The below figure show in Figure 5.2 show the Administrator Application \ Patient Menu.

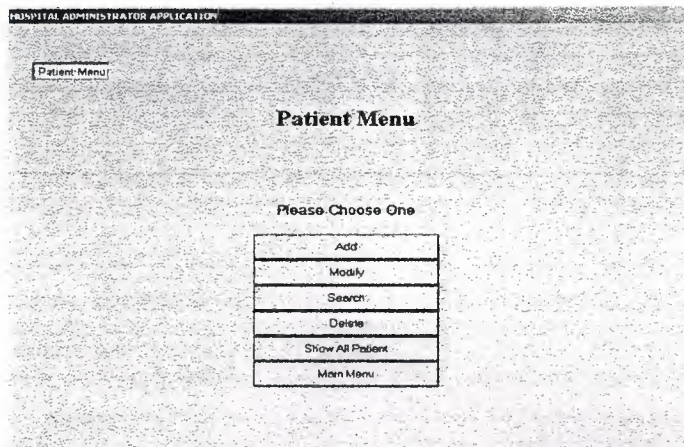


Figure 5.2 Administrator Application \ Patient Menu

5.1.3 Administrator Application \ Outpatient Menu

From here we can add and modify and delete all outpatients records, and even if we were to search for a record that would be possible by using the search button. Since they are all related and linked with each other under the administrator's authorization they therefore look alike in their user interface. The below figure show in Figure 5.3 show the Administrator Application \ Outpatient Menu.

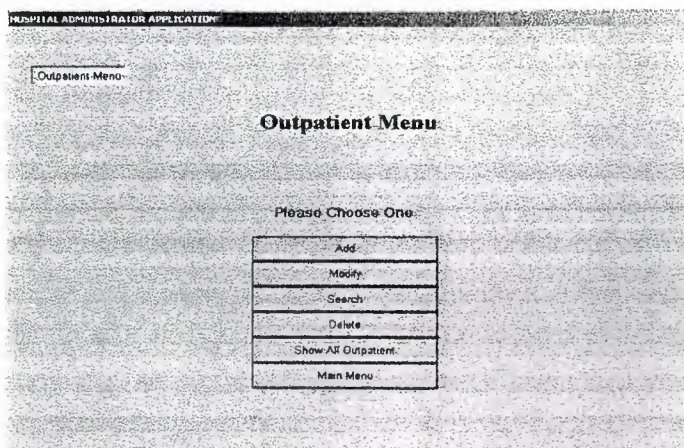


Figure 5.3 Administrator Application \ Outpatient Menu

5.1.4 Administrator Application \ Inpatient Menu

From here we can add and modify and delete all inpatients records, and even if we were to search for a record that would be possible by using the search button. Like mentioned in the previous definitions of interface, again the administrator has all the rights to make his changes in all user interfaces. The below figure show in Figure 5.4 show the Administrator Application \ Inpatient Menu.

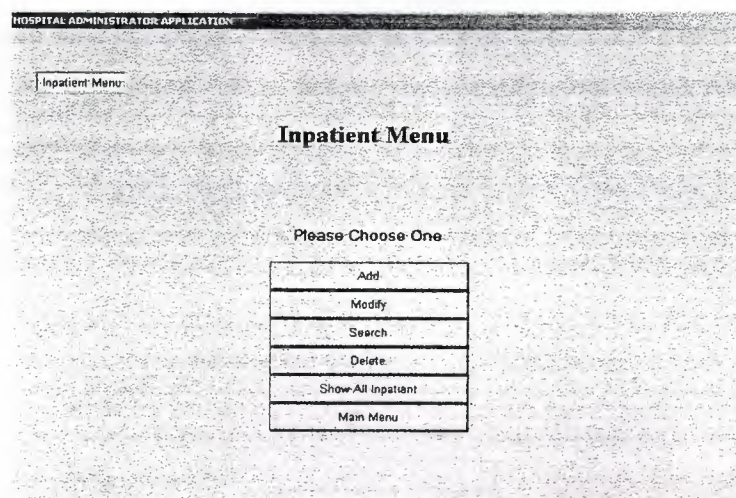


Figure 5.4 Administrator Application \ Inpatient Menu

5.1.5 Administrator Application \ Room Menu

In the room menu interface as you can see below we can add, modify, search, delete and show all rooms. It is vital for the administrator to control room from this point of view for security reasons, in other words not even the doctor can delete or make any changes in the room menu. The below figure show in Figure 5.5 show the Administrator Application \ Room Menu.

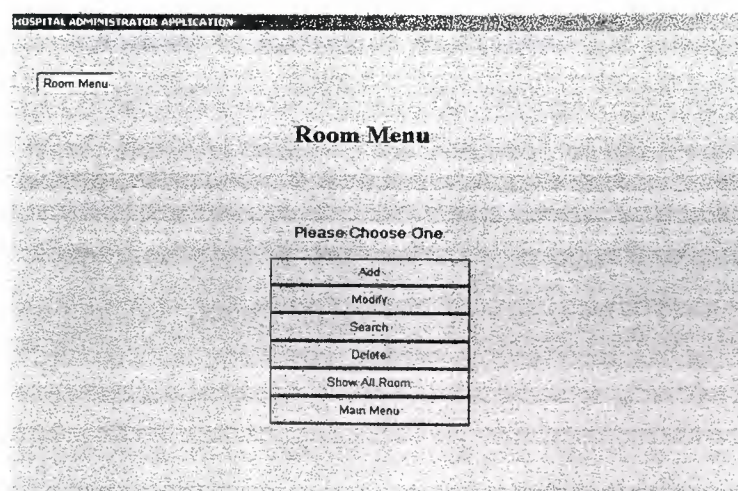


Figure 5.5 Administrator Application \ Room Menu

5.1.6 Administrator Application \ Ward Menu

In the ward menu interface as you can see below we can add, modify, search, delete and show all rooms. There is a major difference between the room menu and the ward menu and that is the number of beds present in each ward, therefore this will automatically lead to an addition in the ward menu, which is the bed availability. This makes it a lot easier for us to understand the previous recorded data from the add menu which will pin point which bed is vacant and which is not after the list of beds have been displayed. The below figure show in Figure 5.6 show the Administrator Application \ Ward Menu.

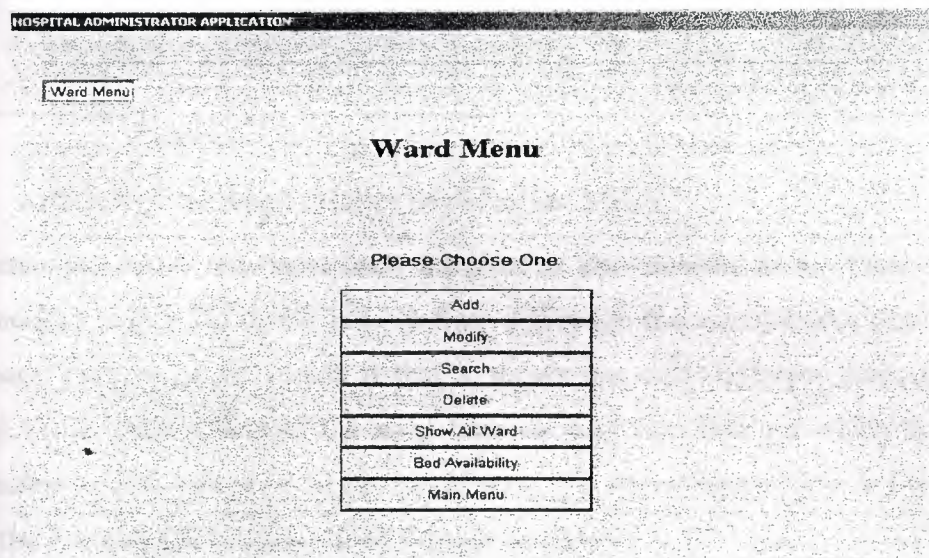


Figure 5.6 Administrator Application \ Ward Menu

5.1.7 Administrator Application \ Doctor Menu

As for the doctor it is generally the same interface as the above and since the administrator has full access to all records he therefore has the ability to add and modify and delete in the doctor's records. Very basic and brief information can be found on this menu, which is linked to the definition of his department, which is represented by a numeric value. In the department menu we will get to understand what field the doctor specializes in. The below figure show in Figure 5.7 show the Administrator Application \ Doctor Menu.

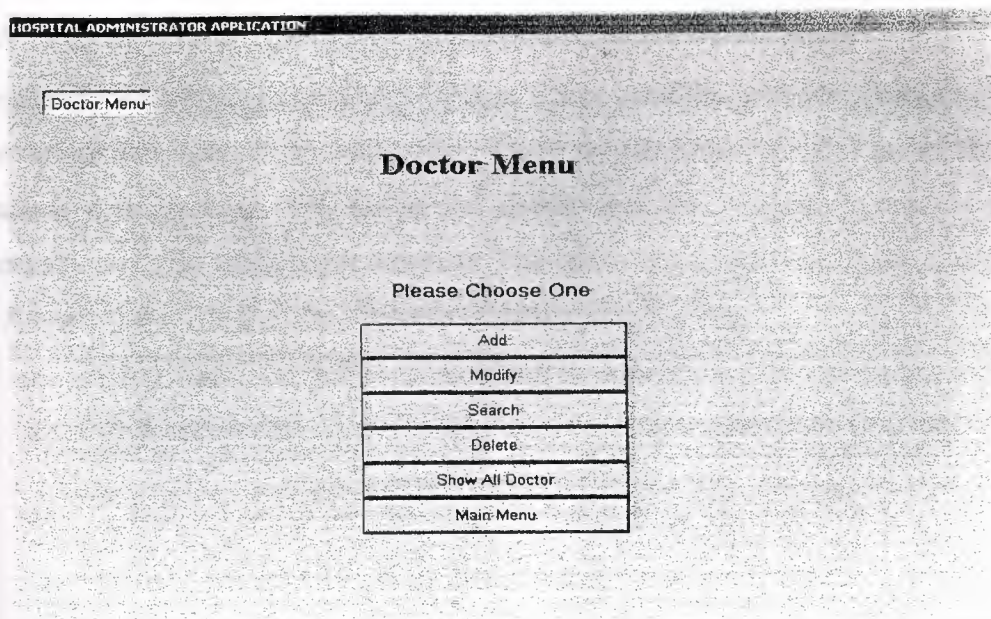


Figure 5.7 Administrator Application \ Doctor Menu

5.1.8 Administrator Application \ Department Menu

As mentioned above the department interface is also modifiable in others we can add, modify, search and delete records from it through the administrator application. The basic purpose of this menu is that it shows the many different fields that are related to the patients and at the same time we may also find out which doctor is responsible or is in charge of which department. The below figure show in Figure 5.8 show the Administrator Application \ Department Menu.

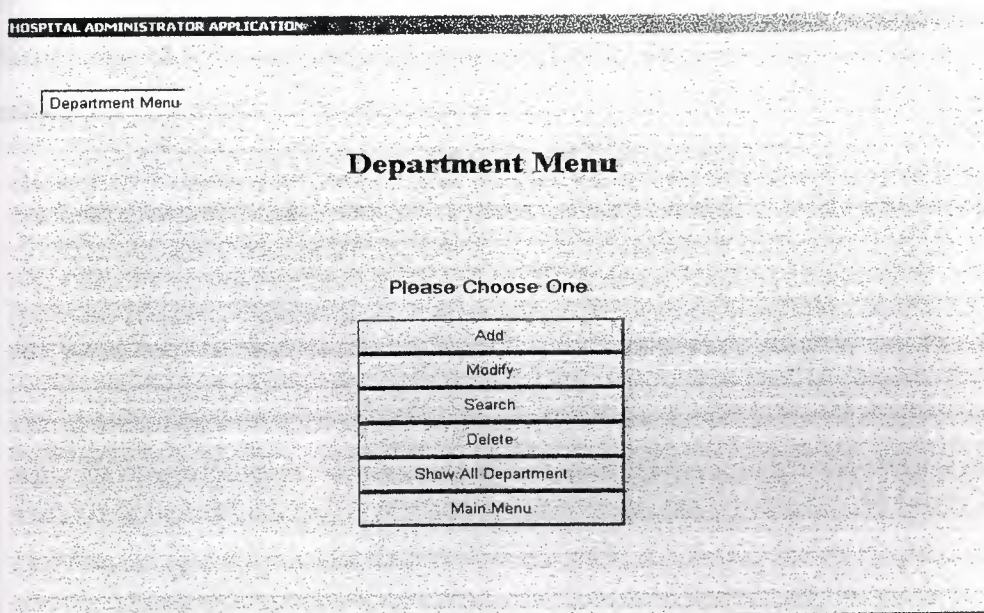


Figure 5.8 Administrator Application \ Department Menu

5.1.9 Administrator Application \ Inpatient Disease Detail Menu

The same goes with Inpatient disease menu where modifications and changes can be done under administrator authentication. There would obviously be a list of diseases matching each inpatient with his or her details included making it easier to make comments and give medical prescription. The below figure show in Figure 5.9 show the Administrator Application \ Inpatient Disease Detail Menu.

The screenshot shows a window titled "HOSPITAL ADMINISTRATOR APPLICATION". Inside, there is a sub-header "Inpatient-Disease-Detail-Menu". The main title is "Inpatient Disease Detail Menu". Below it, the text "Please Choose One:" is displayed. A vertical menu contains the following options: "Add", "Modify", "Search", "Delete", "Show All Disease Detail", and "Main Menu".

Figure 5.9 Administrator Application \ Inpatient Disease Detail Menu

5.1.10 Administrator Application \ Pharmacy Menu

The pharmacy menu as shown below shows the ability to add, modify and delete records. It also shows the availability of stock within the pharmacy. A display of all medicines can also be accessed. The below figure show in Figure 5.10 show the Administrator Application \ Pharmacy Menu.

The screenshot shows a window titled "HOSPITAL ADMINISTRATOR APPLICATION". Inside, there is a sub-header "Pharmacy Menu". The main title is "Pharmacy Menu". Below it, the text "Please Choose One:" is displayed. A vertical menu contains the following options: "Stock In", "Stock Out", "Add", "Modify", "Search", "Delete", "Show All Pharmacy", and "Main Menu".

Figure 5.10 Administrator Application \ Pharmacy Menu

5.2 Reception Application

In the reception application menu the field of area work is very restricted because this where the receptionist will be using to add patients. If mistakes occur the modify button can help correct them.

5.2.1 Reception Application Main Menu

The below figure show in Figure 5.11 show the Reception Application \ Main Menu.

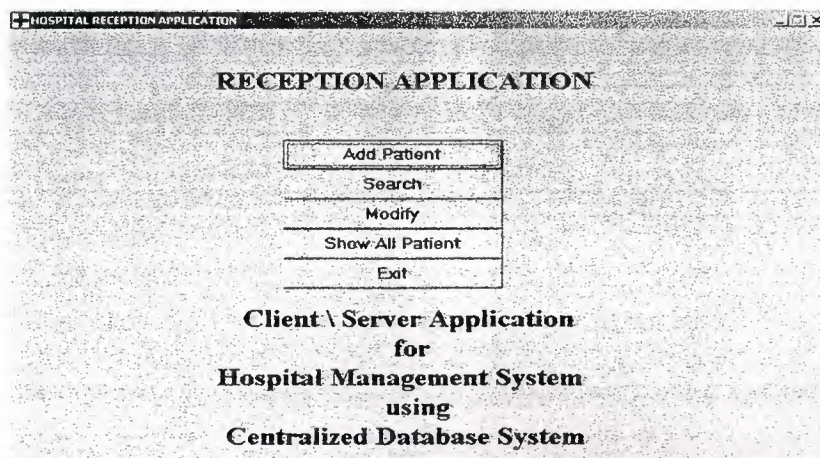


Figure 5.11 Reception Application \ Main Menu

5.2.2 Reception Application \ Add Menu

When adding a patient I am subjected to fill out all details regarding his status, including his name, address, phone number, social security number and sex. This makes it easier to track down and keep in touch with all our patients in times of sickness. The below figure show in Figure 5.12 show the Reception Application \ Add Menu.

A screenshot of a software application window titled "HOSPITAL RECEPTION APPLICATION". The window shows a form titled "Patient Menu \ Add Record:". The form has several input fields: "SSN", "First Name", "Surname", "Phone", "Sex" (with a dropdown arrow), "Date of Birth", and "Address". To the right of these fields are two buttons: "Save" and "Back".

Figure 5.12 Reception Application \ Add Menu

5.2.3 Reception Application \ Search Record

The search record menu is a simple way of pin pointing a patient by using three information fields where two of them are unique when searched for, leaving the date of birth, which can be repeated or shared among some patients. That's why it would be better to make your search by using SSN since it is unique. The below figure show in Figure 5.13 show the Reception Application \ Search Record.

The screenshot shows the 'HOSPITAL RECEPTION APPLICATION' window with the 'Patient Menu \ Search Record' sub-menu. On the left, there is a form with the following fields: SSN (43756865), First Name (demet), Surname (yanar), Phone (5432236894), Sex (F), Date of Birth (22.04.1972), and Address (narli sok. no : 3/1 di). On the right, there is a 'Search' section with three radio buttons: 'SSN' (selected), 'Phone', and 'Date of Birth'. Below these are three empty input fields and a 'Search' button. At the bottom right, there is a 'Back' button.

Figure 5.13 Reception Application \ Search Record

5.2.4 Reception Application \ Modify Record

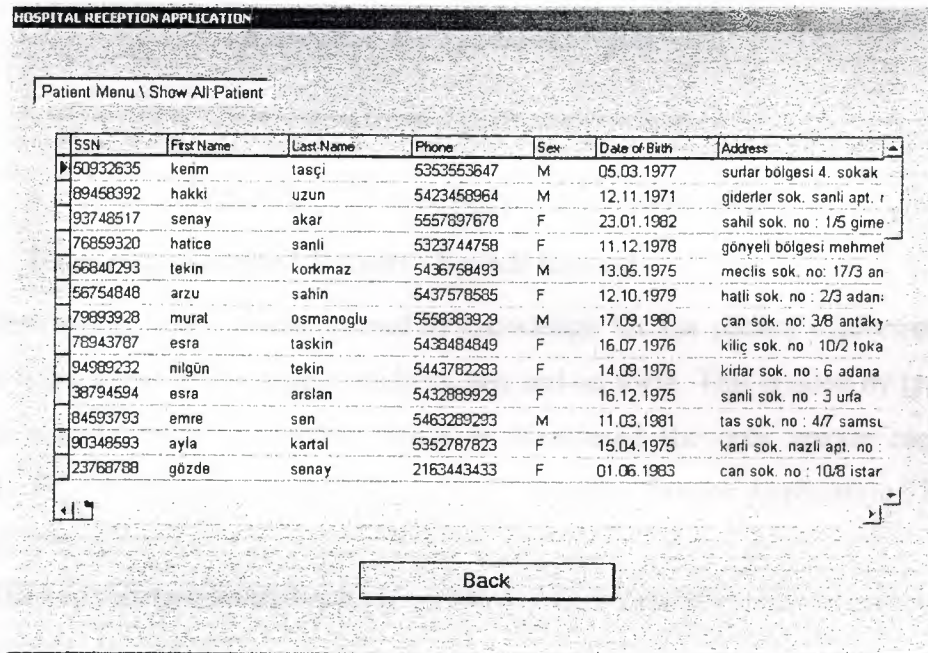
In the modify record interface we can make changes ONLY when a record is found and when it's found a save button appears on the bottom of the interface incase of any changes that have been made to the record. The below figure show in Figure 5.14 show the Reception Application \ Modify Record.

The screenshot shows the 'HOSPITAL RECEPTION APPLICATION' window with the 'Patient Menu \ Modify Record' sub-menu. On the left, there is a form with the following fields: SSN (43756865), First Name (demet), Surname (yanar), Phone (5432236894), Sex (F), Date of Birth (22.04.1972), and Address (narli sok. no : 3/1 di). On the right, there is a 'Search' section with three radio buttons: 'SSN' (selected), 'Phone', and 'Date of Birth'. Below these are three empty input fields and a 'Search' button. At the bottom right, there are two buttons: 'Save' and 'Back'.

Figure 5.14 Reception Application \ Modify Record

5.2.5 Reception Application \ Show All Patient

In the show all patient interface the properties and information of all patients are displayed. The display may be sorted out by right-clicking on any field where a pop-up menu will appear requesting the user in which form would you want to list the patients. This is done within the fields that are listed accordingly. For example you may want to arrange the names of the patients via alphabetical order or according to their phone numbers or social security numbers and so forth. The below figure show in Figure 5.15 show the Reception Application \ Show All Patient.



SSN	First Name	Last Name	Phone	Sex	Date of Birth	Address
50932635	kenim	tasçi	5353553647	M	05.03.1977	surlar bölgesi 4. sokak
69458392	hakki	uzun	5423458964	M	12.11.1971	giderler sok. sanli apt. r
93748517	senay	akar	5557897678	F	23.01.1982	sahil sok. no : 1/5 gime
76859320	hatice	sanli	5323744758	F	11.12.1978	gonyeli bölgesi mehmet
56840293	tekin	korkmaz	5436758493	M	13.05.1975	meclis sok. no: 17/3 an
56754848	arzu	sahin	5437578585	F	12.10.1979	hatli sok. no : 2/3 adan
79893928	murat	osmanoglu	5558383929	M	17.09.1980	can sok. no: 3/8 antaky
78943787	esra	taskin	5438484849	F	16.07.1976	kilic sok. no : 10/2 toka
94989232	nilgün	tekin	5443782283	F	14.09.1976	kirtlar sok. no : 6 adana
38794594	esra	arslan	5432889929	F	16.12.1975	sanli sok. no : 3 urfa
84593793	emre	sen	5463289293	M	11.03.1981	tas sok. no : 4/7 samsi
90348593	ayla	kartal	5352787823	F	15.04.1975	karli sok. nazli apt. no :
23768788	gözde	senay	2163443433	F	01.06.1983	can sok. no : 10/8 istar

Back

Figure 5.15 Reception Application \ Show All Patient

5.3 Doctor Application

The doctor application which gives general information about patients and whether or not they are inpatients or out patients to simplify records at great easy access.

5.3.1 Doctor Application main form

As we can see in the interface below the doctor application consists of a four menu interface. The below figure show in Figure 5.16 show the Doctor Application \ Main Menu.

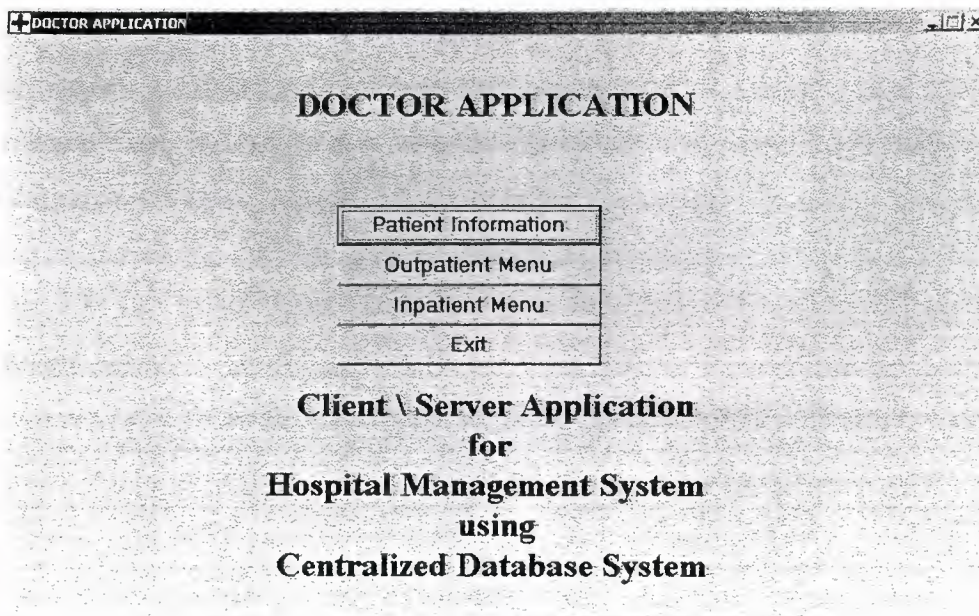


Figure 5.16 Doctor Application \ Main Menu

5.3.2 Doctor Application \ Patient \ Search Record

The first would be the doctor's need of knowledge for the patient's information for example the name of the patient, address, sex and so forth. This is done by typing the patient's social security number, which is considered the only unique method of search. The below figure show in Figure 5.17 show the Doctor Application \ Patient \ Search Record.

DOCTOR APPLICATION

Patient \ Search Record

SSN	43756865
First Name	demet
Last Name	yanar
Phone	5432236894
Sex	F
Date of Birth	22.04.1972
Address	narli sok. no : 3/1 di

Search

SSN

Search

Back

Figure 5.17 Doctor Application \ Patient \ Search Record

5.3.3 Doctor Application \ Outpatient Menu

In the outpatient menu the doctor can add, search, modify, but not delete any of the information displayed before him, because the records are created at the reservation application and even the reception can not delete any records for security purposes so this again leaves the administrator in charge of all deletions and major changes in any record. The below figure show in Figure 5.18 show the Doctor Application \ Outpatient Menu.

The screenshot shows a window titled "DOCTOR APPLICATION". Inside, there is a sub-header "Outpatient Menu:". Below this, the text "Outpatient Menu" is centered. Underneath, it says "Please Choose One:". A vertical menu box contains four options: "Add", "Search", "Modify", and "Main Menu".

Figure 5.18 Doctor Application \ Outpatient Menu

5.3.4 Doctor Application \ Outpatient Menu \ Add Record

When adding a patient in the outpatient menu we need to fill out some information regarding this matter. The information could be the social security number or the type of disease and medicine and the doctor ID, which represented by a numeric value and the date of the patient's check in. Then and only then can we modify and save the changes by clicking on the save button that appears below. The below figure show in Figure 5.19 show the Doctor Application \ Outpatient Menu \ Add Record.

The screenshot shows a window titled "DOCTOR APPLICATION". Inside, there is a sub-header "Outpatient Menu \ Add Outpatient". Below this, there are five labels with corresponding input fields: "SSN", "Disease", "Medicine", "Doctor id", and "Date". To the right of these fields is a vertical menu box containing three options: "Save", "Back", and "Main Menu".

Figure 5.19 Doctor Application \ Outpatient Menu \ Add Record

5.3.5 Doctor Application \ Outpatient Menu \ Search Record

The search record is for searching the outpatient records and to double check for any mistakes or confirmation for the doctor's purposes. It will also include any late visits that the outpatient has had, in other words more of a log file or history file for the doctor's knowledge. The below figure show in Figure 5.20 show the Doctor Application \ Outpatient Menu \ Search Record.

The screenshot shows a window titled "DOCTOR APPLICATION" with a sub-header "Outpatient Menu \ Search Record". On the left, there is a label "SSN:" followed by a text box containing "90348593". On the right, there is a "Search:" section with an "SSN:" label and an empty text box, and a "Search" button below it. Below these elements is a table with four columns: "Date", "Disease", "Medicine", and "Doctor id". The table contains two rows of data. At the bottom of the window, there are two buttons: "Back" and "Main Menu".

Date	Disease	Medicine	Doctor id
14.05.2003	kidney	apranax fort	15
12.02.2003	stomach	aspirin	7

Figure 5.20 Doctor Application \ Outpatient Menu \ Search Record

5.3.6 Doctor Application \ Outpatient Menu \ Modify Record

The outpatient record also like other records may need to be modified and corrected that's why we need the modification interface incase of mistakes. It also shows the history of the outpatient when social security number is entered. The below figure show in Figure 5.21 show the Doctor Application \ Outpatient Menu \ Modify Record.

The screenshot shows a window titled "DOCTOR APPLICATION" with a sub-header "Outpatient Menu \ Modify Record". On the left, there are five labels with corresponding text boxes: "SSN:" (90348593), "Disease:" (stomach), "Medicine:" (aspirin), "Doctor id:" (7), and "Date:" (12.02.2003). On the right, there is a "Search:" section with an "SSN:" label and an empty text box, and a "Search" button below it. At the bottom of the window, there are three buttons: "Save", "Back", and "Main Menu".

Figure 5.21 Doctor Application \ Outpatient Menu \ Modify Record

5.3.7 Doctor Application \ Inpatient Menu

In the inpatient menu the aim is to search for records of inpatients that have previously been to the hospital. And on the lower button you may very well search for the inpatient disease details. The below figure show in Figure 5.22 show the Doctor Application \ Inpatient Menu.

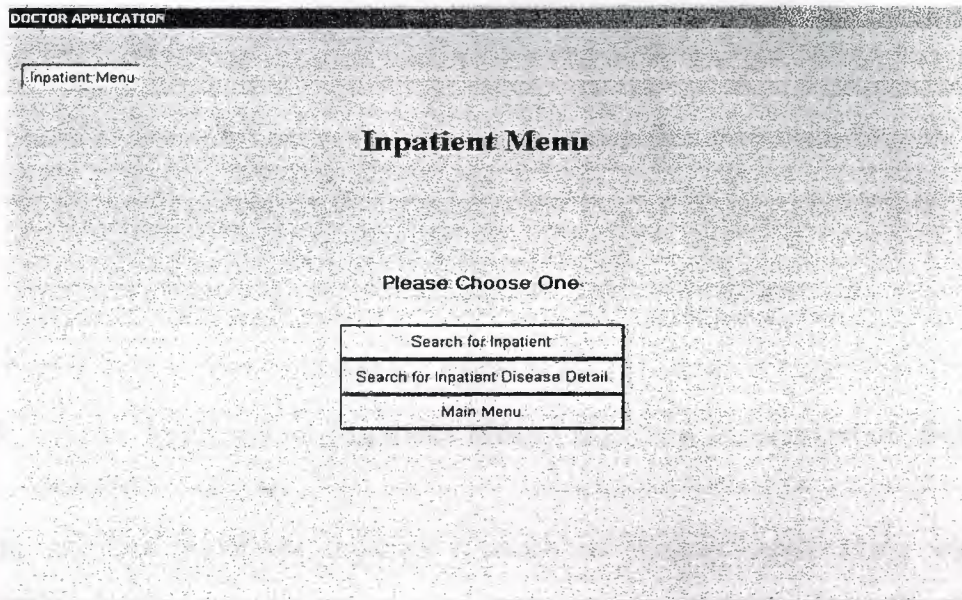


Figure 5.22 Doctor Application \ Inpatient Menu

5.3.8 Doctor Application \ Inpatient Menu \ Inpatient \ Search Record

In this menu like the previous outpatient menu the only easy unique way of searching is by the social security number. where it will ease the record and previous log files like when the last time the inpatient came to the hospital and for what cause and in which department he was sent to. The first would be the doctor's need of knowledge for the inpatient's information. This is done by typing the inpatient's social security number, which is considered the only unique method of search. In order to overcome confusion I have managed to assign a command where an inpatient would be staying in a room and the ward and bed number would automatically be nulled. The below figure show in Figure 5.23 show the Doctor Application \ Inpatient Menu \ Inpatient \ Search Record.

DOCTOR APPLICATION

Inpatient \ Search Record

Patient No: 10
 SSN: 76859320
 Admission Date: 13.12.2003
 Discharge Date: 14.12.2003
 Department id: 2
 Ward id:
 Bed No:
 Room No: 2

Search

SSN:
 Search

Back
 Main Menu

Figure 5.23 Doctor Application \ Inpatient Menu \ Inpatient \ Search Record

5.3.9 Doctor Application \ Inpatient Menu \ Inpatient Disease Detail \ Search Record

In the interface below the inpatient's search of diseases details along with the inpatient's history. The below figure show in Figure 5.24 show the Doctor Application \ Inpatient Menu \ Inpatient Disease Detail \ Search Record.

DOCTOR APPLICATION

Inpatient Disease Detail \ Search Record

SSN: 94989232

Search

SSN:
 Search

Patient No	Date of Check	Time of Check	Doctor id	Disease	medicine
5	06.02.2003	11:00	1	nephritis	antibiotic
5	06.02.2003	13:00	1	nephritis	antibiotic

Back
 Main Menu

Figure 5.24 Doctor Application \ Inpatient Menu \ Inpatient Disease Detail \ Search Record

5.3.10 Doctor Application \ Inpatient Menu \ Advanced Search

One of the main characteristic of system is that it can answer the query not only typically and non typically questions. Example of this query searching for by the type of disease, searching for by the type of used medicine or searching for by the doctor id. Also for this screen report is available.

SSN	Pno	Date of check	Time of check	DId	Disease	Medicine	Response
78787878	25	31.12.2003	12:21	1	headach	aspirin	
94999232	5	06.02.2003	11:00	1	nephritic	antibiotic	
94999232	5	06.02.2003	14:00	1	cancer	kamoteraphi	
94999232	5	06.02.2003	13:00	1	nephritis	antibiotic	

Figure 5.25 Doctor Application \ Inpatient Menu \ Advanced Search

5.4 Room Application

Room application is an application where the room receptionist has all records concerning the rooms.

5.4.1 Room Application main form

The room application consists of two menus as shown in the interface below. The below figure show in Figure 5.25 show the Room Application \ Main Menu.

ROOM APPLICATION

Inpatient Menu
Inpatient Disease Detail Menu
Exit

Client \ Server Application
for
Hospital Management System
using
Centralized Database System

Figure 5.26 Room Application \ Main Menu

5.4.2 Room Application \ Inpatient Menu

The first menu is the inpatient menu where the room receptionist can add, search and modify the records. The below figure show in Figure 5.26 show the Room Application \ Inpatient Menu.

The screenshot shows a window titled "ROOM APPLICATION". Inside, there is a sub-header "Inpatient Menu". Below this, the text "Please Choose One" is displayed. A vertical menu box contains four options: "Add", "Search", "Modify", and "Main Menu".

Figure 5.27 Room Application \ Inpatient Menu

5.4.3 Room Application \ Inpatient Menu \ Add Record

When clicking on the add record the patient number is displayed automatically where a fields regarding this matter are to be filled and saved for future usage. The fields consist of Dept ID and Social security number, admission and discharge date and the room number itself. After filling the details we may save our work by clicking on the save button. The below figure show in Figure 5.27 show the Room Application \ Inpatient Menu \ Add Record.

The screenshot shows a window titled "ROOM APPLICATION". Inside, there is a sub-header "Inpatient Menu \ Add Record". Below this, there are several input fields: "Patient No" (with the value "23" entered), "SSN", "Admission Date", "Discharge Date", "Department id", and "Room No". To the right of these fields is a vertical menu box containing three options: "Save", "Back", and "Main Menu".

Figure 5.28 Room Application \ Inpatient Menu \ Add Record

5.4.4 Room Application \ Inpatient Menu \ Search Record

Again the searching method is done by social security number since that could be the easiest and most unique way of identification. The below figure show in Figure 5.28 show the Room Application \ Inpatient Menu \ Search Record.

ROOM APPLICATION

Inpatient Menu \ Search Record

Patient No	15
SSN	90348593
Admission Date	11.11.2003
Discharge Date	15.11.2003
Department id	7
Room No	7

Search

SSN

Search

Back

Main Menu

Figure 5.29 Room Application \ Inpatient Menu \ Search Record

5.4.5 Room Application \ Inpatient Menu \ Modify Record

The same here when modifying a record the social security number is required as a vital field. The below figure show in Figure 5.29 show the Room Application \ Inpatient Menu \ Modify Record.

ROOM APPLICATION

Inpatient Menu \ Modify Record

Patient No	15
SSN	90348593
Admission Date	11.11.2003
Discharge Date	15.11.2003
Department id	7
Room No	7

Search

SSN

Search

Save

Back

Main Menu

Figure 5.30 Room Application \ Inpatient Menu \ Modify Record

5.4.6 Room Application \ Inpatient Disease Detail Menu

In the disease details menu the receptionist can make changes and save them for future usage. The below figure show in Figure 5.30 show the Room Application \ Inpatient Disease Detail Menu.

The screenshot shows a window titled "ROOM APPLICATION" with a sub-header "Inpatient Disease Detail Menu". Below the header, the text "Please Choose One" is displayed. A vertical list of buttons is shown: "Add", "Search", "Modify", and "Main Menu".

Figure 5.31 Room Application \ Inpatient Disease Detail Menu

5.4.7 Room Application \ Inpatient Disease Detail Menu \ Add Record

When adding the information the fields apply above just like they apply here. The fields are given and include the SSN, patient number, date of check, time of check and doctor ID. And here again we may save our records by clicking on the save button. The below figure show in Figure 5.31 show the Room Application \ Inpatient Disease Detail Menu \ Add Record.

The screenshot shows a window titled "ROOM APPLICATION" with a sub-header "Inpatient Disease Detail Menu \ Add Record". On the left, there are labels for "SSN", "Patient No", "Date of Check", "Time of Check", "Doctor id", "Disease", "Medicine", and "Response", each followed by a text input field. On the right, there is a vertical list of buttons: "Save", "Back", and "Main Menu".

Figure 5.32 Room Application \ Inpatient Disease Detail Menu \ Add Record

5.4.8 Room Application \ Inpatient Disease Detail Menu \ Search Record

Search record menu again like mentioned above use the social security number as a key search. The below figure show in Figure 5.32 show the Room Application \ Inpatient Disease Detail Menu \ Search Record.

ROOM APPLICATION

Inpatient Disease Detail Menu \ Search Record

SSN: 94989232

Patient No: 5

Date of Check: 06.02.2003

Time of Check: 11.00

Doctor id: 1

Disease: nephritis

Medicine: antibiotic

Response:

Search:

SSN:

Search

Back

Main Menu

Figure 5.33 Room Application \ Inpatient Disease Detail Menu \ Search Record

5.4.9 Room Application \ Inpatient Disease Detail Menu \ Modify Record

The room application also has a modification record where the social security number is required for any modifications needed. The below figure show in Figure 5.33 show the Room Application \ Inpatient Disease Detail Menu \ Modify Record.

ROOM APPLICATION

Inpatient Disease Detail Menu \ Modify Record

SSN: 94989232

Patient No: 5

Date of Check: 06.02.2003

Time of Check: 11.00

Doctor id: 1

Disease: nephritis

Medicine: antibiotic

Response:

Search:

SSN:

Search

Save

Back

Main Menu

Figure 5.34 Room Application \ Inpatient Disease Detail Menu \ Modify Record

5.5 Ward Application

The ward application is very simple just like the previous room application where would find easy ways of recording adding and modifying the records.

5.5.1 Ward Application main form

In the ward application main form as shown below we can take a look at the links to the buttons and we will discuss them further below. The below figure show in Figure 5.34 show the Ward Application \ Main Menu.

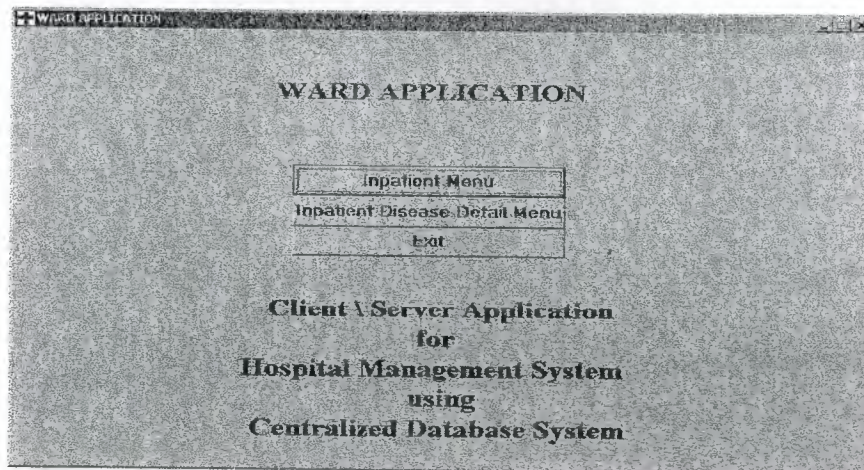


Figure 5.35 Ward Application \ Main Menu

5.5.2 Ward Application \ Inpatient Menu

Like room application the ward inpatient menu has the add button and the search and modify buttons. The below figure show in Figure 5.35 show the Ward Application \ Inpatient Menu.

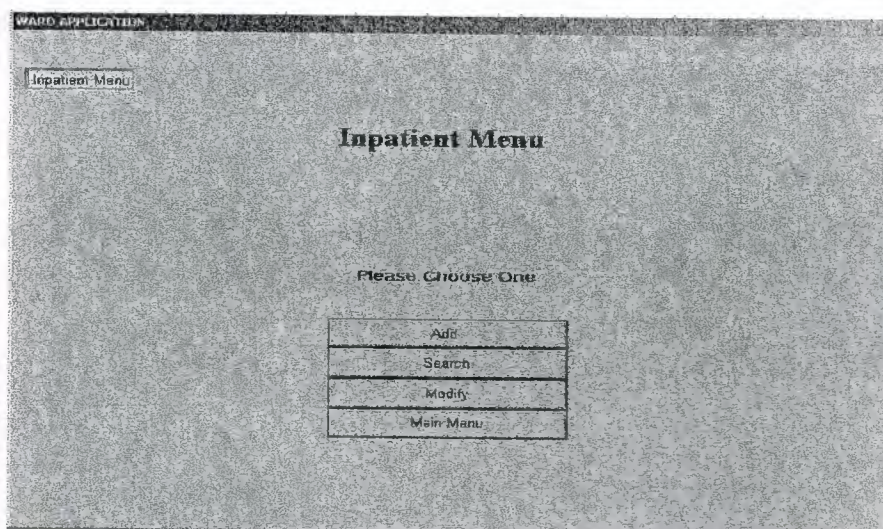


Figure 5.36 Ward Application \ Inpatient Menu

5.5.3 Ward Application \ Inpatient Menu \ Add Record

The inpatient add record gives an automatic patient number and has pretty much the same information under the room inpatient menu except for the ward id and the bed no which are included in the actual building. We can also save our information by clicking on the save button. The below figure show in Figure 5.36 show the Ward Application \ Inpatient Menu \ Add Record.

WARD APPLICATION

Inpatient Menu \ Add Record

Patient No. 23

SSN

Admission Date

Discharge Date

Department id

Ward id

Bed No.

Save

Back

Main Menu

Figure 5.37 Ward Application \ Inpatient Menu \ Add Record

5.5.4 Ward Application \ Inpatient Menu \ Search Record

The inpatient menu also has a search record where again the social security number is required for the fields. And this helps the ward receptionist pinpoint and locate an inpatient. The below figure show in Figure 5.37 show the Ward Application \ Inpatient Menu \ Search Record.

WARD APPLICATION

Inpatient Menu \ Search Record

Patient No. 24

SSN 76859320

Admission Date 05.05.2003

Discharge Date 05.06.2003

Department id 1

Ward id 1

Bed No. 2

Search

SSN

Search

Back

Main Menu

Figure 5.38 Ward Application \ Inpatient Menu \ Search Record

5.5.5 Ward Application \ Inpatient Menu \ Modify Record

The inpatient modification record is also done via social security number, where we can locate and make changes needed, and then save them. The below figure show in Figure 5.38 show the Ward Application \ Inpatient Menu \ Modify Record.

WARD APPLICATION

Inpatient Menu \ Modify Record

Patient No	24	Search
SSN	76859320	SSN
Admission Date	05.05.2003	Search
Discharge Date	05.06.2003	
Department id	1	Save
Ward id	1	Back
Bed No	2	Main Menu

Figure 5.39 Ward Application \ Inpatient Menu \ Modify Record

5.5.6 Ward Application \ Inpatient Disease Detail Menu

The inpatient disease detail menu also consists of a menu where we may add, search and modify our records. The below figure show in Figure 5.39 show the Ward Application \ Inpatient Disease Detail Menu.

WARD APPLICATION

Inpatient Disease Detail Menu

Inpatient Disease Detail Menu

Please Choose One

Add
Search
Modify
Main Menu

Figure 5.40 Ward Application \ Inpatient Disease Detail Menu

5.5.7 Ward Application \ Inpatient Disease Detail Menu \ Add Record

When adding a record all fields are specified by the doctor for the receptionist to benefit from at times of the doctor's absence. The menu consists of patient no, social security number, date of check and time of check, doctor id, disease, medicine and response. The below figure show in Figure 5.40 show the Ward Application \ Inpatient Disease Detail Menu \ Add Record.

The screenshot shows a window titled 'WARD APPLICATION' with a sub-header 'Inpatient Disease Detail Menu \ Add Record'. It contains several input fields for data entry: SSN, Patient No, Date of Check, Time of Check, Doctor id, Disease, Medicine, and Response. To the right of these fields is a vertical stack of three buttons: 'Save', 'Back', and 'Main Menu'.

Figure 5.41 Ward Application \ Inpatient Disease Detail Menu \ Add Record

5.5.8 Ward Application \ Inpatient Disease Detail Menu \ Search Record

The inpatient diseases are all the same since they are all using the social security number as there key search. The below figure show in Figure 5.41 show the Ward Application \ Inpatient Disease Detail Menu \ Search Record.

The screenshot shows a window titled 'WARD APPLICATION' with a sub-header 'Inpatient Disease Detail Menu \ Search Record'. It features input fields for SSN, Patient No, Date of Check, Time of Check, Doctor id, Disease, Medicine, and Response. Some fields are pre-filled with example data: SSN (34989232), Patient No (5), Date of Check (06.02.2003), Time of Check (11.00), Doctor id (1), Disease (nephritis), and Medicine (antibiotic). To the right, there is a 'Search' section with an SSN input field and a 'Search' button. Below this are 'Back' and 'Main Menu' buttons.

Figure 5.42 Ward Application \ Inpatient Disease Detail Menu \ Search Record

5.5.9 Ward Application \ Inpatient Disease Detail Menu \ Modify Record

Before making any modification in this field we need to search using the social security number and at the same time locating our inpatient. The need for modification in this field is very important since not all medicines heal or work from the first time so the response may change from time to time according to the inpatient's recovery. The below figure show in Figure 5.42 show the Ward Application \ Inpatient Disease Detail Menu \ Modify Record.

WARD APPLICATION

Inpatient Disease Detail Menu \ Modify Record

SSN: 94989232

Patient No: 5

Date of Check: 06.02.2003

Time of Check: 11:00

Doctor Id: 1

Disease: nephritis

Medicine: antibiotic

Response:

Search

SSN:

Search

Save

Back

Main Menu

Figure 5.43 Ward Application \ Inpatient Disease Detail Menu \ Modify Record

5.6 Pharmacy Application

The pharmacy application is another application simplified for the knowledge of medicines available and more.

5.6.1 Pharmacy Application main form

The main form consists a group of buttons with links each one leading to a different field. As shown on the interface below. The below figure show in Figure 5.43 show the Pharmacy Application \ Main Menu.

PHARMACY APPLICATION

Add

Stock In

Stock Out

Search

Modify

Exit

Client \ Server Application
for
Hospital Management System
using
Centralized Database System

Figure 5.44 Pharmacy Application \ Main Menu

5.6.2 Pharmacy Application \ Add Menu

The add menu consists of the medicine id which is represented by a numeric value which is the bar code id here again no confusion can be made since each id is unique. The medicine name is to be added and so is the quantity in stock along with the price tag, which represents the price per medicine. We may very well save all the information by using the save button. The below figure show in Figure 5.44 show the Pharmacy Application \ Add Menu.

The screenshot shows a window titled "PHARMACY APPLICATION" with a menu bar containing "Pharmacy Menu \ Add Record". Below the menu bar, there are four input fields labeled "Medicine id", "Medicine Name", "Quantity", and "Price". To the right of these fields is a button labeled "Save" and a button labeled "Back".

Figure 5.45 Pharmacy Application \ Add Menu

5.6.3 Pharmacy Application \ Stock In Menu

The stock in menu can search all medicines by typing down their id, which will show their amount added to the amount available in the stock. In other words an addition to the existing medicine we have in stock. The below figure show in Figure 5.45 show the Pharmacy Application \ Stock In Menu.

The screenshot shows a window titled "PHARMACY APPLICATION" with a menu bar containing "Pharmacy Menu \ Stock in". Below the menu bar, there are four input fields labeled "Medicine id", "Medicine Name", "Quantity", and "Price". To the right of these fields is a button labeled "Search". Below the "Search" button is a button labeled "Save" and a button labeled "Back".

Figure 5.46 Pharmacy Application \ Stock In Menu

5.6.4 Pharmacy Application \ Stock Out Menu

The stock out menu is the amount of medicine that is taken or used which will cause the stock or quantity of that same medicine to decrease. The below figure show in Figure 5.46 show the Pharmacy Application \ Stock Out Menu.

PHARMACY APPLICATION

Pharmacy Menu \ Stok Out

Medicine id: 1234567891234

Medicine Name: apranax fort

Quantity: 43

Price: 67000

Search

Medicine id:

Search

Save

Back

Figure 5.46 Pharmacy Application \ Stock Out Menu

5.6.5 Pharmacy Application \ Search Menu

The search menu can give us a summary of what we have in our stock and this includes prices and quantity of that medicine we have in stock. The below figure show in Figure 5.47 show the Pharmacy Application \ Search Menu.

PHARMACY APPLICATION

Pharmacy Menu \ Search Record

Medicine id: 1234567891234

Medicine Name: apranax fort

Quantity: 43

Price: 67000

Search

Medicine id:

Search

Back

Figure 5.47 Pharmacy Application \ Search Menu

5.6.6 Pharmacy Application \ Modify Record

We may very well modify our records in the records due to an increase of prices for example or an expiry date and so forth, so this makes it important to make all these changes through the modify record menu. The below figure show in Figure 5.48 show the Pharmacy Application \ Modify Record.

PHARMACY APPLICATION

Pharmacy Menu \ Modify Record

Medicine id: 1234567891234

Medicine Name: apranax fort

Quantity: 43

Price: 67000

Search

Medicine id:

Search

Save

Back

Figure 5.48 Pharmacy Application \ Modify Menu

5.7 Definition of Terms and Connection

5.7.1 Microsoft Data Access Components (MDAC)

MDAC is an umbrella for Microsoft's database technologies and includes ADO, OLE DB, ODBC, and RDS (Remote Data Services). Often you will hear people use the terms MDAC and ADO interchangeably (but incorrectly) because their version numbers and releases are now aligned. As ADO is only distributed as part of MDAC, we talk in terms of MDAC releases. The major releases of MDAC have been versions 1.5, 2.0, 2.1, 2.5 and 2.6. MDAC is also distributed with most Microsoft products that have some kind of database content. This includes Windows 98, Windows 2000, Windows Millennium Edition, Microsoft Office, Internet Explorer and SQL Server.

5.7.1.1 ActiveX Data Objects (ADO)

ADO is part of a bigger picture called Microsoft Data Access Components (MDAC). ADO, which stands for ActiveX Data Objects, is Microsoft's high-level interface for database access. ADO is implemented on Microsoft's data-access OLE DB technology, which provides access to relational and non-relational databases as well as e-mail and file systems and custom business objects.

5.7.1.2 Object Linking and Embedding (OLE)

OLE DB is a Component Object Model (COM) based data access interface. It supports applications written to use OLE DB or data object interfaces that use OLE DB. OLE DB is designed to work with relational databases (such as those in SQL Server).

OLE DB uses a provider to gain access to a particular data source. Providers for SQL Server, Oracle, Jet (Microsoft Access databases), ODBC supplied with SQL Server. Using the OLE DB provider for ODBC, OLE DB can be used to gain access to any ODBC data source.

5.7.1.3 Component Object Model (COM)

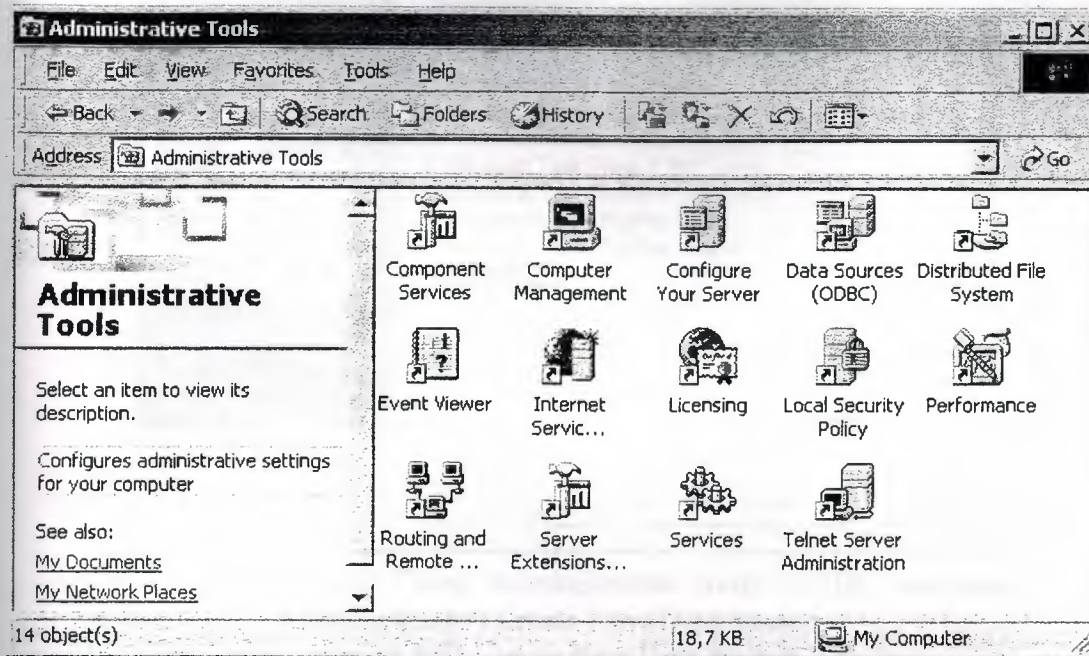
The programming model on which several SQL Server and database application programming interfaces (APIs), such as SQL-DMO, OLE DB, and ADO, are based.

5.7.1.4 Using Data Sources (ODBC)

You can use Data Sources Open Database Connectivity (ODBC) to access data from a variety of database management systems. For example, if you have a program that accesses data in a SQL database. To do this, you must add software components called drivers to your system. Data Sources (ODBC) helps you add and configure these drivers. To open Data Sources (ODBC), click Start, point to Settings, and then click Control Panel. Double-click Administrative Tools, and then double-click Data Sources (ODBC).

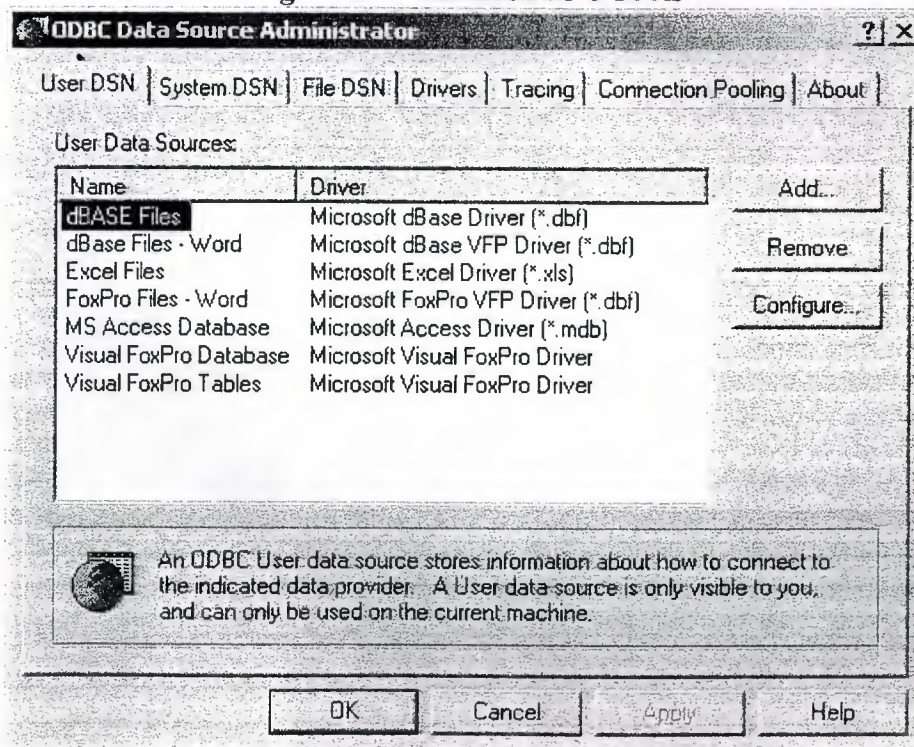
5.7.2 How to Create Open Database Connectivity (ODBC)

By following figures we can create Open Database Connectivity (ODBC). First of all choose Data Sources (ODBC) component.



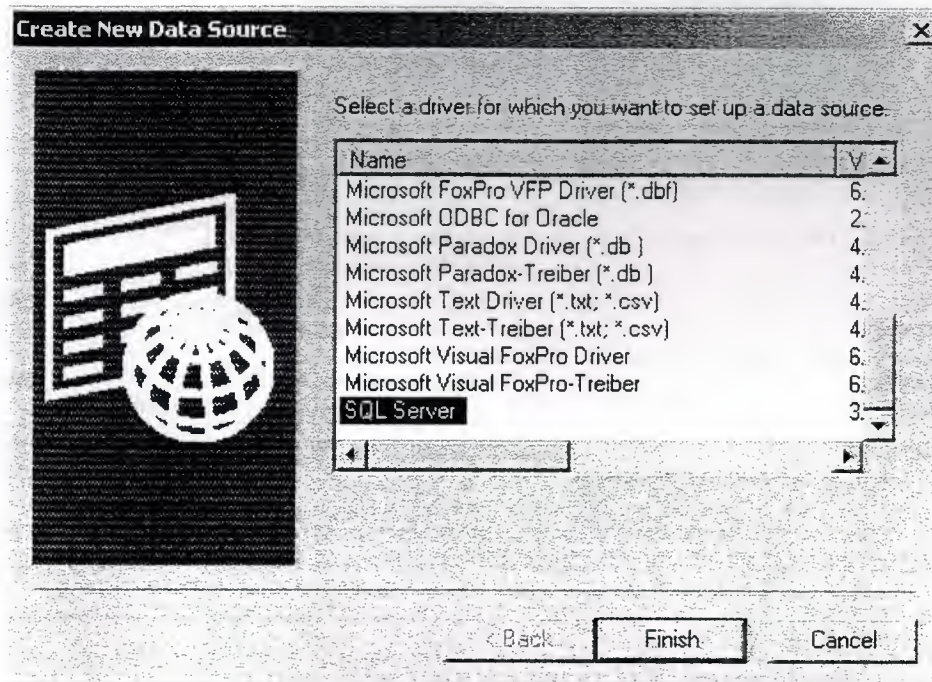
Start \ Settings \ Control Panel \ Administrative Tools

Figure 5.49 Administrative Tools

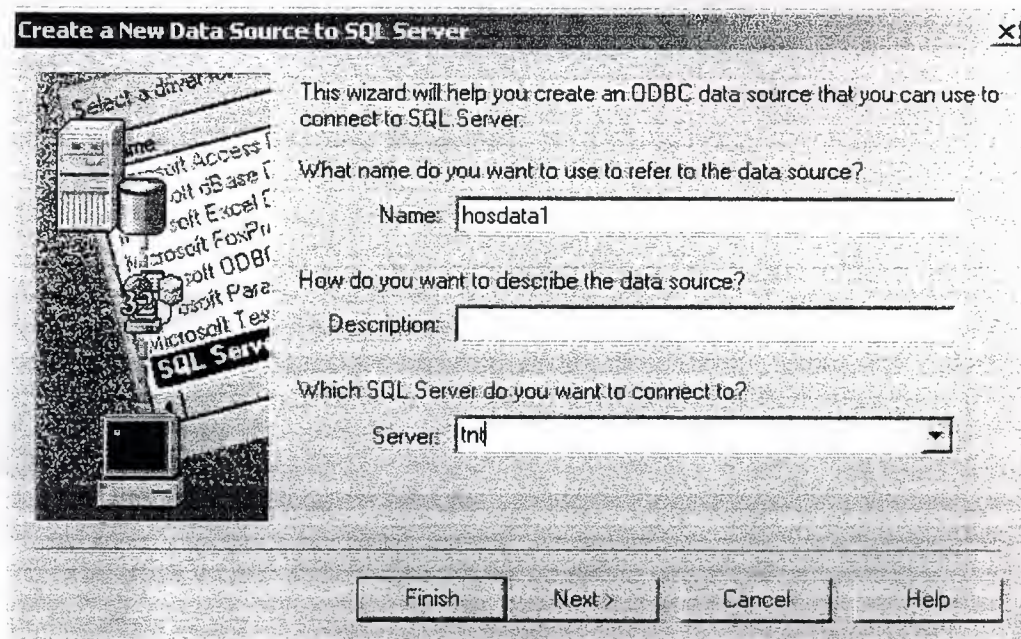


Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator

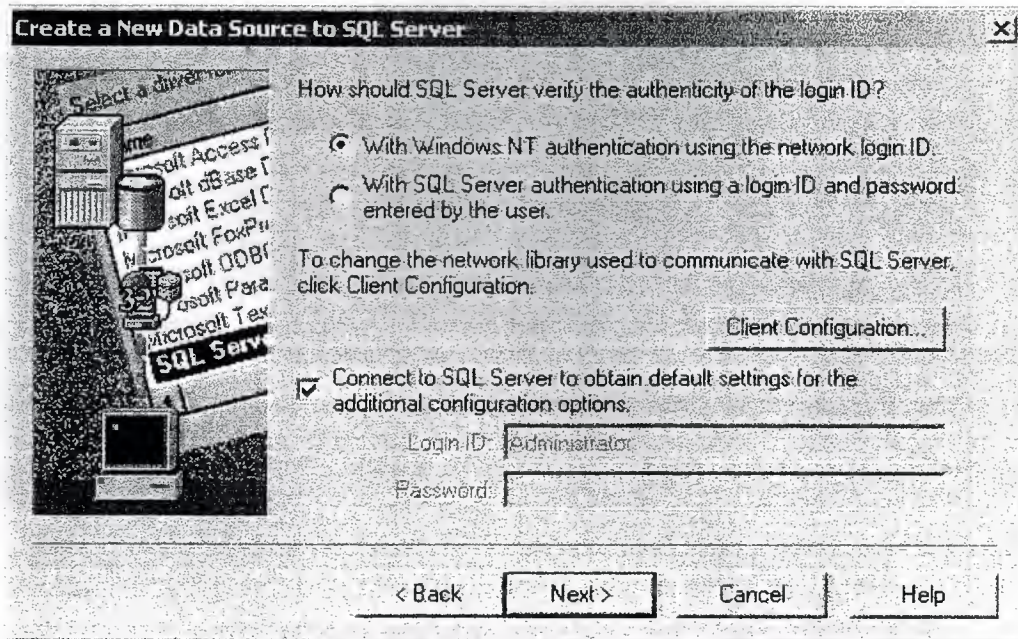
Figure 5.50 ODBC Data Source Administrator



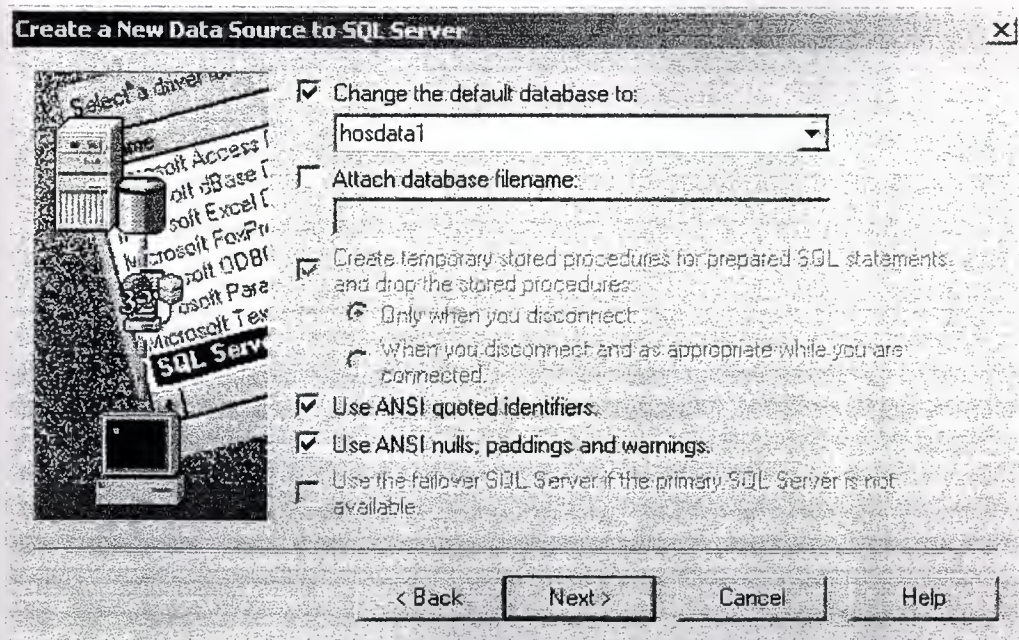
Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ Create New Data Source
Figure 5.51 Create New Data Source



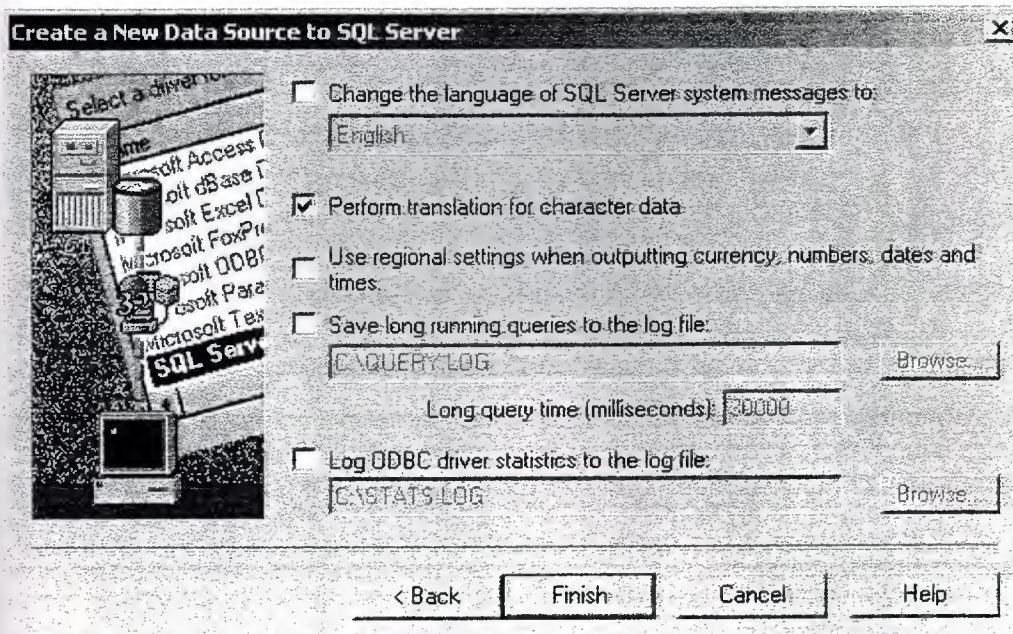
Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ Create New Data Source to SQL Server \ Page1
Figure 5.52 Create a New Data Source to SQL Server



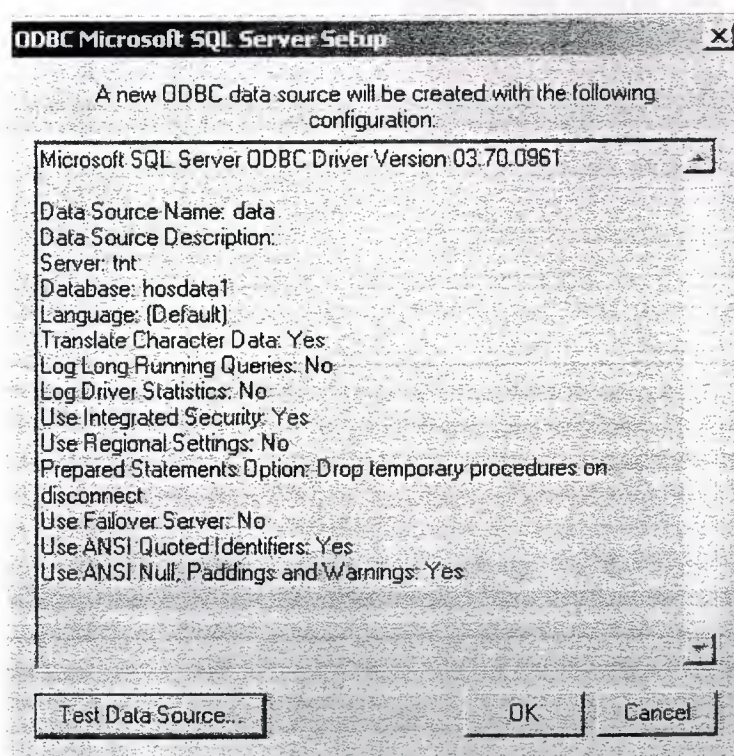
Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ Create New Data Source to SQL Server \ Page2
Figure 5.53 Create a New Data Source to SQL Server



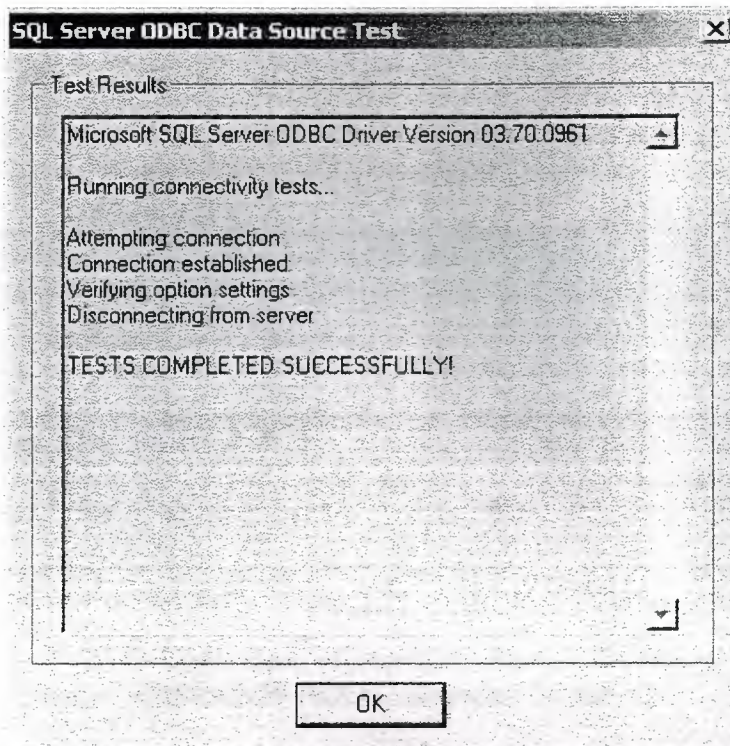
Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ Create New Data Source to SQL Server \ Page3
Figure 5.54 Create a New Data Source to SQL Server



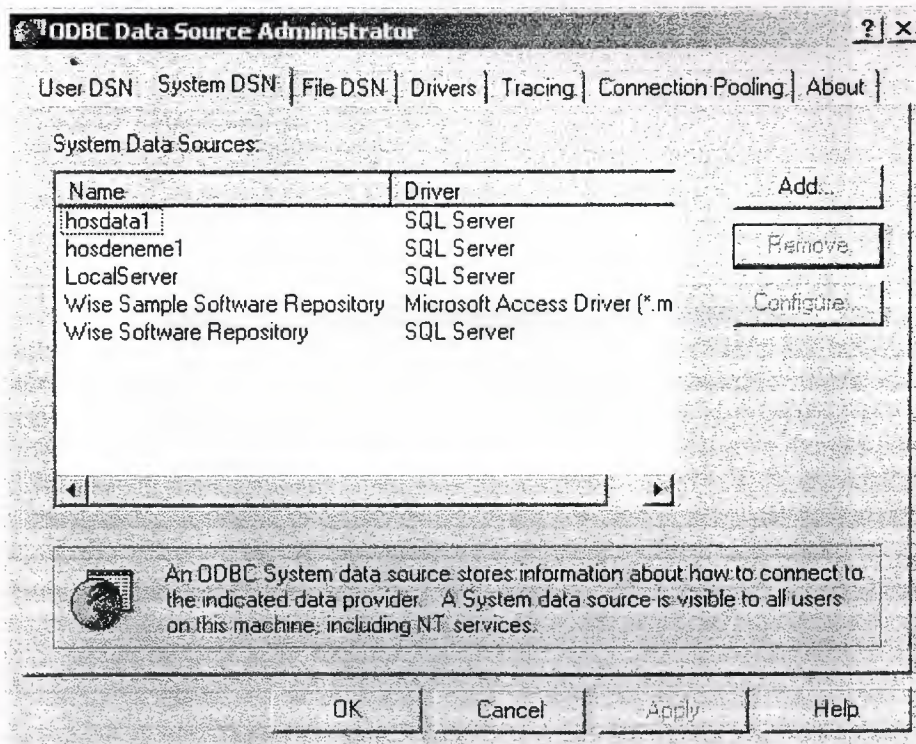
Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ Create New Data Source to SQL Server \ Page4
Figure 5.55 Create a New Data Source to SQL Server



Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ ODBC Microsoft SQL Server Setup
Figure 5.56 ODBC Microsoft SQL Server Setup



Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ SQL Server ODBC Data Source Test
Figure 5.57 SQL Server ODBC Data Source Test

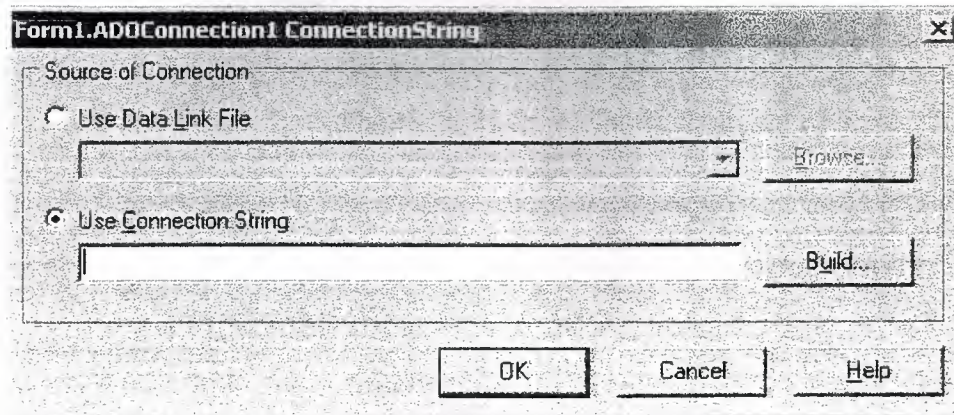


Start \ Settings \ Control Panel \ Administrative Tools \ ODBC Data Source Administrator \ Create New Data Source to SQL Server \ Page2
Figure 5.58 ODBC Data Source Administrator

At the result of these operation Open Database Connection (ODBC) is created.

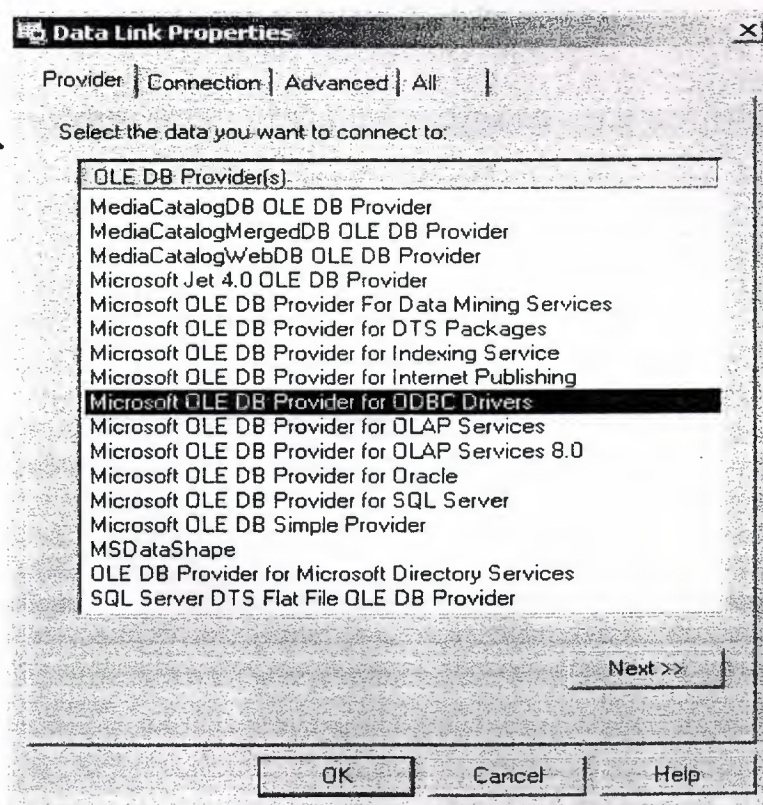
5.7.3 How to Create ADO Connection for Delphi 7

By following figures we can create ADO Connection for Delphi 7.



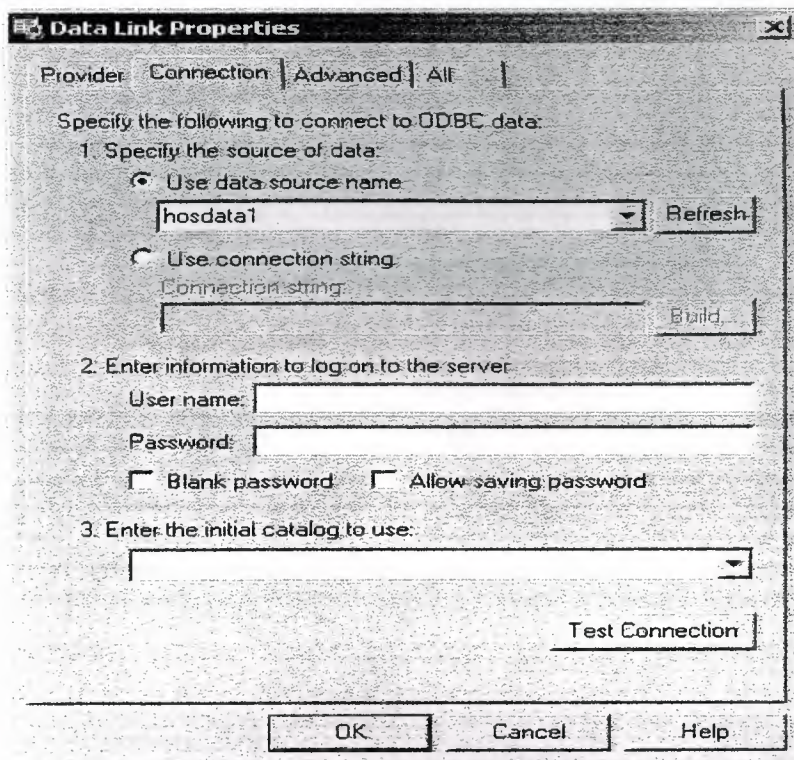
ADO Connection \ Connection String \ Build

Figure 5.59 ADO Connection String



ADO Connection \ Connection String \ Data Link Properties \ Provider

Figure 5.60 Data Link Properties



Data Link Properties

Provider: Connection | **Advanced** | All

Specify the following to connect to ODBC data:

1. Specify the source of data:

☒ Use data source name

hosdata1 Refresh

☐ Use connection string

Connection string: Build

2. Enter information to log on to the server:

User name: Password:

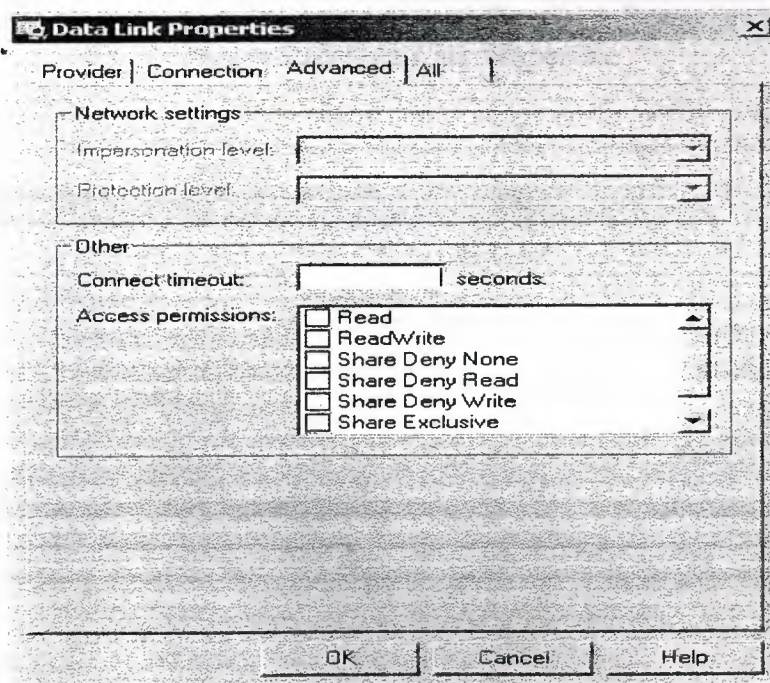
☐ Blank password ☐ Allow saving password

3. Enter the initial catalog to use:

Test Connection

OK Cancel Help

ADO Connection \ Connection String \ Data Link Properties \ Connection
Figure 5.61 Data Link Properties



Data Link Properties

Provider | Connection | **Advanced** | All

Network settings

Impersonation level: Protection level:

Other

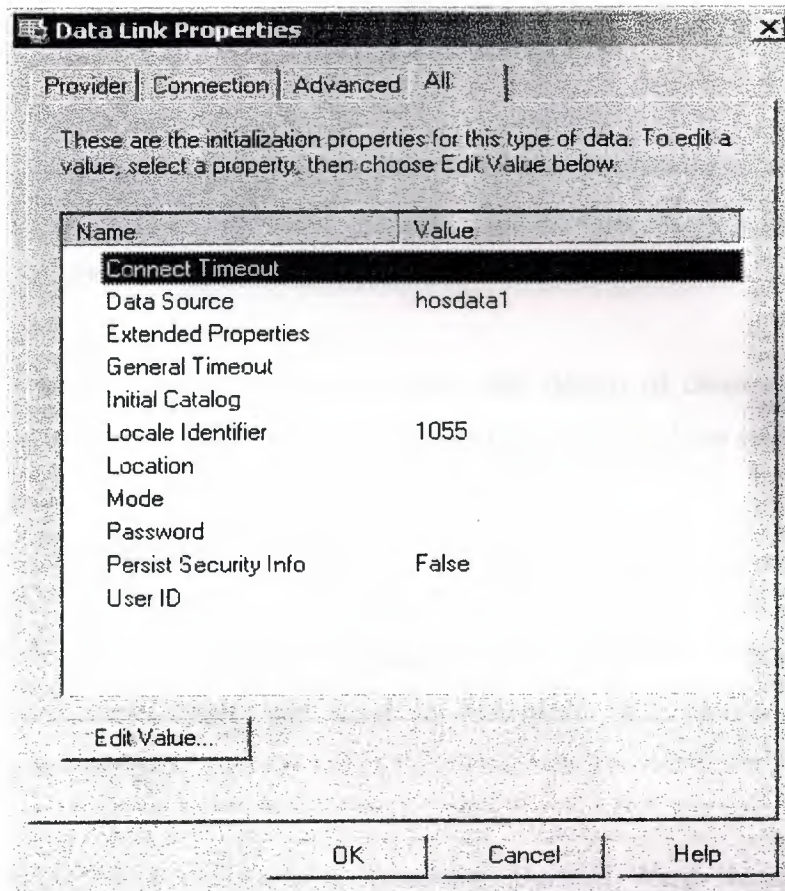
Connect timeout: seconds

Access permissions:

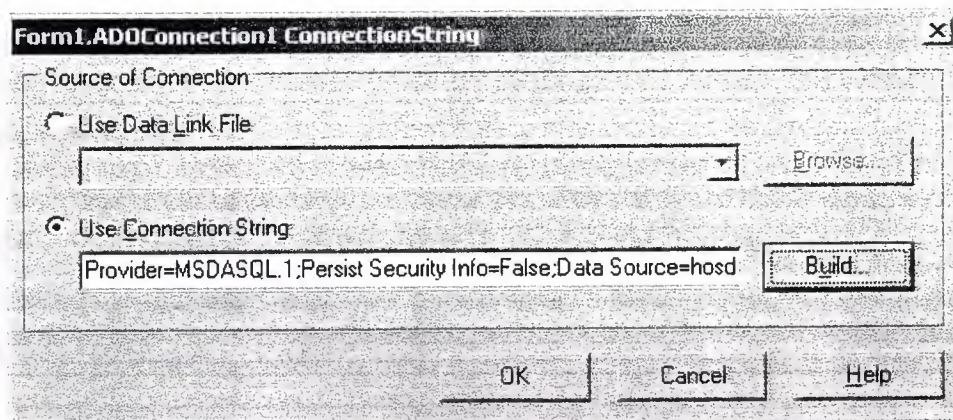
- ☐ Read
- ☐ ReadWrite
- ☐ Share Deny None
- ☐ Share Deny Read
- ☐ Share Deny Write
- ☐ Share Exclusive

OK Cancel Help

ADO Connection \ Connection String \ Data Link Properties \ Advanced
Figure 5.62 Data Link Properties



ADO Connection \ Connection String \ Data Link Properties \ All
Figure 5.63 Data Link Properties



ADO Connection \ Connection String \ OK
Figure 5.64 Ado Connection String

At the result of these operation ADO Connection for Delphi 7 is created.

CONCLUSION

Networking of computers allows some tasks to be executed on a server system, and some tasks to be executed on client systems. This division of work has led to the development of client-server database systems.

The thesis includes detailed information about the design of client-server system, client-server architecture, data modeling and architecture of database systems.

We develop the client-server application for Hospital Management System using centralized database system in this master thesis starting creating necessary tables such as patient, outpatient, inpatient, inpatient disease detail, doctor, department, bed availability, pharmacy, room and ward in Microsoft SQL Server. Application programs are in Delphi 7.

We have developed Administration, Reception, Doctor, Ward Reception, Room Reception and Pharmacy applications in the thesis, that makes it easier for a user to access and retrieve data for Hospital activities. These applications are working with centralized database. The centralized database provides the security of the system.

The user can easily display and print out different forms concerning patient's disease information.

The developed system can make searching not only by standard, typical queries of user and non typical queries of user. By advanced search options, we can query data by SSN(Social Security Number), type of Disease, Doctor id and type of Medicine.

REFERENCES

Books

1. Batini, Ceri, S. Kant, and B. Navathe; Conceptual Database Design; (1991). The Benjamin/Cummings Publishing Company.
2. Date; An Introduction to Database Systems, 5th ed.; (1990). Addison-Wesley.
3. Fleming, Candace C. and Barbara von Halle; Handbook of Relational Database Design; (1989). Addison-Wesley.
4. Kroenke, David; Database Processing, 2nd ed.; (1983). Science Research Associates.
5. Martin, James; Information Engineering; (1989). Prentice-Hall.
6. Reingruber, Michael C. and William W. Gregory; The Data Modeling Handbook: A Best-Practice Approach to Building Quality Data Models; (1994). John Wiley & Sons, Inc.
7. Simsion, Graeme; Data Modeling Essentials: Analysis, Design, and Innovation; (1994). International Thompson Computer Press.
8. Toby J.; Database Modeling & Design: The Basic Principles, 2nd ed.; (1994). Morgan Kaufmann Publishers, Inc.
9. R. Orfali, D. Harkey, and J. Edwards; Essential Client/Server Survival Guide; (1994). John Wiley & Sons.; ISBN 0-471-13119-9.
10. Sean Nolan; Microsoft SQL Server 7.0 Database Implementation Training Kit \ Microsoft Corporation; (1999). Microsoft Press.; ISBN 1-57231-826-0
11. Marco Cantu; Mastering Delphi 7; (2002). SYBEX Inc.; ISBN : 0-7821-2874-2.

Online Articles

1. Hospital Management System (HMS): www.Hospital Management System.htm
2. The Basic Concepts of Client/server Architecture: www.Basic Concepts of Client-server Architecture
3. What is Client / Server: www.Client-Server Information1.htm
4. Understanding Client/Server Computing: www.Understanding Client-Server Architecture.htm
5. Hospital Information System: www.Hospital Information System - HIS - Medinous.htm

APPENDIX A

Client/Server Application for Hospital Management System Using Centralized Database System – Program Listing

Administrator \ Patient \ Add Record

```
procedure TForm2.Button6Click(Sender: TObject);
type
patientng1=set of '0'..'9';
patientcg1=set of 'a'..'z';
var
patientfor:integer;
patientv1:integer;
patientng:patientng1;
patientcg:patientcg1;
begin
patientng:=['0'..'9'];
patientcg:=['a'..'z'];
adotable1.Close;
adotable1.Open;
if (adotable1.Locate('SSN',edit1.Text,[locaseinsensitive])=true) then
begin
application.messagebox('This record already exist in Patient table. Please check
again.','Check Record',MB_OK+48+MB_SYSTEMMODAL);
edit1.SetFocus;
exit;
end;
patientv1:=edit1.GetTextLen;
if patientv1=0 then
begin
application.MessageBox('Please enter numeric value on SSN field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit1.setfocus;
exit;
end
else
begin
if (patientv1<8) or (patientv1>8) then
begin
application.MessageBox('Please enter 8 character value on SSN field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit1.setfocus;
exit;
end;
for patientfor:=0 to patientv1 do
```



```

begin
  if (edit1.Text[patientfor] in patientcg) then
    begin
      application.messagebox('Please enter numeric value on SSN field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
      edit1.setfocus;
      exit;
    end;
  end;
end;
patientv1:=edit2.gettextlen;
if patientv1=0 then
  begin
    application.MessageBox('Please enter character value on First Name field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit2.setfocus;
    exit;
  end
else
  begin
    for patientfor:=0 to patientv1 do
      begin
        if (edit2.text[patientfor] in patientng) then
          begin
            application.messagebox('Please enter character value on First Name
field','Check Value',MB_OK+48+MB_SYSTEMMODAL);
            edit2.setfocus;
            exit;
          end;
        end;
      end;
    patientv1:=edit3.gettextlen;
    if patientv1=0 then
      begin
        application.MessageBox('Please enter character value on Surname field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit3.setfocus;
        exit;
      end
    else
      begin
        for patientfor:=0 to patientv1 do
          begin
            if (edit3.text[patientfor] in patientng) then
              begin
                application.messagebox('Please enter character value on Surname
field','Check Value',MB_OK+48+MB_SYSTEMMODAL);
                edit3.setfocus;

```

```

        exit;
    end;
end;
end;
patientv1:=edit4.gettextlen;
if patientv1=0 then
    begin
        application.MessageBox('Please enter numeric value on Phone field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit4.setfocus;
        exit;
    end
else
    begin
        if (patientv1<10) or (patientv1>12) then
            begin
                application.MessageBox('Please enter right value on Phone field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                edit4.setfocus;
                exit;
            end;
        for patientfor:=0 to patientv1 do
            begin
                if (edit4.text[patientfor] in patientcg) then
                    begin
                        application.messagebox('Please enter numeric value on Phone field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                        edit4.setfocus;
                        exit;
                    end;
            end;
        end;
    end;
if combobox1.itemindex=-1 then
    begin
        application.MessageBox('Please choose Sex (Male\Female)','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        combobox1.setfocus;
        exit;
    end;
if (maskedit1.text[1] in patientng) and (maskedit1.text[2] in patientng) then
    begin
        maskedit1.SelStart:=0;
        maskedit1.SelLength:=2;
        if (maskedit1.seltext='00') then
            begin
                application.messagebox('Please check day on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                maskedit1.setfocus;
            end;
        end;
    end;

```



```

    exit;
end;
if (strtoint(maskedit1.seltext)>31) then
begin
    application.messagebox('Please check day on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit1.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check day on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit1.setfocus;
    exit;
end;
if (maskedit1.text[4] in patientng) and (maskedit1.text[5] in patientng) then
begin
    maskedit1.SelStart:=3;
    maskedit1.SelLength:=2;
    if (maskedit1.seltext='00') then
        begin
            application.messagebox('Please check month on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit1.setfocus;
            exit;
        end;
        if (strtoint(maskedit1.seltext)>12) then
            begin
                application.messagebox('Please check month on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                maskedit1.setfocus;
                exit;
            end;
        end
    else
        begin
            application.MessageBox('Please check month on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit1.setfocus;
            exit;
        end;
    if (maskedit1.text[7] in patientng) and (maskedit1.text[8] in patientng)
    and (maskedit1.text[9] in patientng) and (maskedit1.text[10] in patientng) then
        begin
            maskedit1.SelStart:=6;
            maskedit1.SelLength:=4;

```

```

if (maskedit1.seltext='0000') then
begin
    application.messagebox('Please check year on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit1.setfocus;
    exit;
end;
edit83.SelStart:=6;
edit83.SelLength:=4;
if (strtoint(maskedit1.seltext)>strtoint(edit83.seltext)) then
begin
    application.MessageBox('Please check year on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit1.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check year on Date of Birth field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit1.setfocus;
    exit;
end;
patientv1:=edit8.GetTextLen;
if patientv1<2 then
begin
    application.MessageBox('Please check Address field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit8.setfocus;
    exit;
end;
adotable1.append;
adotable1.FieldValues['SSN']:=edit1.Text;
adotable1.FieldValues['f_name']:=edit2.Text;
adotable1.FieldValues['l_name']:=edit3.Text;
adotable1.FieldValues['phone']:=edit4.Text;
if combobox1.itemindex=0 then
begin
    adotable1.FieldValues['sex']:= 'M';
end;
if combobox1.itemindex=1 then
begin
    adotable1.FieldValues['sex']:= 'F';
end;
adotable1.FieldValues['d_of_birth']:=maskedit1.Text;
adotable1.FieldValues['address']:=edit8.Text;
adotable1.Post;

```



```

edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
combobox1.itemindex:=-1;
maskedit1.Text:="";
edit8.Text:="";
form2.Label12.Visible:=false;
form2.label13.Visible:=false;
form2.Label14.Visible:=false;
form2.label15.Visible:=false;
form2.label16.Visible:=false;
form2.label17.Visible:=false;
form2.label19.Visible:=false;
form2.edit1.Visible:=false;
form2.edit2.Visible:=false;
form2.edit3.Visible:=false;
form2.edit4.Visible:=false;
form2.ComboBox1.Visible:=false;
form2.MaskEdit1.Visible:=false;
form2.Edit8.Visible:=false;
form2.Button6.Visible:=false;
form2.Button59.Visible:=false;
form2.Button84.Visible:=false;
if (application.MessageBox('This record saved successfully. Do you want add
another record ?','Add
Record',MB_YESNO+32+256+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=true;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;
    form2.tabsheet16.tabvisible:=false;
    form2.tabsheet17.tabvisible:=false;
    form2.tabsheet18.tabvisible:=false;
    form2.tabsheet19.tabvisible:=false;
    form2.tabsheet20.tabvisible:=false;

```

```

form2.tabsheet21.tabvisible:=false;
form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;
form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.TabVisible:=false;
form2.TabSheet36.TabVisible:=false;
form2.TabSheet37.TabVisible:=false;
form2.TabSheet38.TabVisible:=false;
form2.TabSheet39.TabVisible:=false;
form2.TabSheet40.TabVisible:=false;
end
else
begin
form2.Label12.Visible:=true;
form2.label13.Visible:=true;
form2.Label14.Visible:=true;
form2.label15.Visible:=true;
form2.label16.Visible:=true;
form2.label17.Visible:=true;
form2.label19.Visible:=true;
form2.edit1.Visible:=true;
form2.edit2.Visible:=true;
form2.edit3.Visible:=true;
form2.edit4.Visible:=true;
form2.ComboBox1.Visible:=true;
form2.MaskEdit1.Visible:=true;
form2.Edit8.Visible:=true;
form2.Button6.Visible:=true;
form2.Button59.Visible:=true;
form2.Button84.Visible:=true;
edit1.setfocus;
end;
end;

```

Administrator \ Outpatient \ Add Record

```

procedure TForm2.Button66Click(Sender: TObject);
type

```



```

outpatientng1=set of '0'..'9';
var
outpatientng:outpatientng1;
begin
outpatientng:=['0'..'9'];
adotable1.Close;
adotable1.Open;
if (adotable1.Locate('SSN',edit20.Text,[locaseinsensitive])=false) then
begin
application.messagebox('This SSN not included in Patient Table. Please check
again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit20.text:="";
edit20.setfocus;
exit;
end;
if (edit21.gettextlen<3) then
begin
application.MessageBox('Less character value on Disease field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit21.setfocus;
exit;
end;
if (edit22.gettextlen<3) then
begin
application.MessageBox('Less character value on Medicine field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit22.setfocus;
exit;
end;
adotable7.Close;
adotable7.open;
if (adotable7.locate('doctor_id',edit23.Text,[locaseinsensitive])=false) then
begin
application.MessageBox('This Doctor id not included in Doctor Table. Please
check again','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit23.SetFocus;
exit;
end;
if (maskedit12.text[1] in outpatientng) and (maskedit12.text[2] in outpatientng) then
begin
maskedit12.SelStart:=0;
maskedit12.SelLength:=2;
if (maskedit12.seltext='00') then
begin
application.messagebox('Please check day on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
maskedit12.setfocus;
exit;

```

```

end;
if (strtoint(maskedit12.seltext)>31) then
begin
    application.messagebox('Please check day on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit12.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check day on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit12.setfocus;
    exit;
end;
if (maskedit12.text[4] in outpatientng) and (maskedit12.text[5] in outpatientng) then
begin
    maskedit12.SelStart:=3;
    maskedit12.SelLength:=2;
    if (maskedit12.seltext='00') then
    begin
        application.messagebox('Please check month on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit12.setfocus;
        exit;
    end;
    if (strtoint(maskedit12.seltext)>12) then
    begin
        application.messagebox('Please check month on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit12.setfocus;
        exit;
    end;
end
else
begin
    application.MessageBox('Please check month on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit12.setfocus;
    exit;
end;
if (maskedit12.text[7] in outpatientng) and (maskedit12.text[8] in outpatientng)
and (maskedit12.text[9] in outpatientng) and (maskedit12.text[10] in outpatientng)
then
begin
    maskedit12.SelStart:=6;
    maskedit12.SelLength:=4;

```



```

if (maskedit12.seltext='0000') then
begin
    application.messagebox('Please check year on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit12.setfocus;
    exit;
end;
edit83.SelStart:=6;
edit83.SelLength:=4;
if (strtoint(maskedit12.seltext)>strtoint(edit83.seltext)) then
begin
    application.MessageBox('Please check year on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit12.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check year on Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit12.setfocus;
    exit;
end;
adotable2.Open;
adotable2.Append;
adotable2.FieldValues['SSN']:=edit20.Text;
adotable2.FieldValues['disease']:=edit21.Text;
adotable2.FieldValues['medicine']:=edit22.Text;
adotable2.FieldValues['doctor_id']:=edit23.Text;
adotable2.FieldValues['date']:=maskedit12.Text;
adotable2.Post;
edit20.Text:="";
edit21.Text:="";
edit22.Text:="";
edit23.Text:="";
maskedit12.Text:="";
form2.label31.Visible:=false;
form2.label32.Visible:=false;
form2.Label33.Visible:=false;
form2.label34.Visible:=false;
form2.label114.visible:=false;
form2.edit20.Visible:=false;
form2.edit21.Visible:=false;
form2.Edit22.Visible:=false;
form2.Edit23.Visible:=false;
form2.MaskEdit12.Visible:=false;
form2.Button66.Visible:=false;

```

```

form2.Button68.visible:=false;
form2.Button67.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?','Add Record',MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=true;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;
    form2.tabsheet16.tabvisible:=false;
    form2.tabsheet17.tabvisible:=false;
    form2.tabsheet18.tabvisible:=false;
    form2.tabsheet19.tabvisible:=false;
    form2.tabsheet20.tabvisible:=false;
    form2.tabsheet21.tabvisible:=false;
    form2.tabsheet22.tabvisible:=false;
    form2.tabsheet23.tabvisible:=false;
    form2.tabsheet24.tabvisible:=false;
    form2.tabsheet25.tabvisible:=false;
    form2.tabsheet26.tabvisible:=false;
    form2.tabsheet27.tabvisible:=false;
    form2.tabsheet28.tabvisible:=false;
    form2.tabsheet29.tabvisible:=false;
    form2.tabsheet30.tabvisible:=false;
    form2.tabsheet31.tabvisible:=false;
    form2.tabsheet32.tabvisible:=false;
    form2.tabsheet33.tabvisible:=false;
    form2.tabsheet34.tabvisible:=false;
    form2.TabSheet35.Tab Visible:=false;
    form2.TabSheet36.Tab Visible:=false;
    form2.TabSheet37.Tab Visible:=false;
    form2.TabSheet38.Tab Visible:=false;
    form2.TabSheet39.Tab Visible:=false;
    form2.TabSheet40.Tab Visible:=false;
end
else
begin

```



```

form2.label31.Visible:=true;
form2.label32.Visible:=true;
form2.Label33.Visible:=true;
form2.label34.Visible:=true;
form2.label114.Visible:=true;
form2.edit20.Visible:=true;
form2.edit21.Visible:=true;
form2.Edit22.Visible:=true;
form2.Edit23.Visible:=true;
form2.MaskEdit12.Visible:=true;
form2.Button66.Visible:=true;
form2.Button68.visible:=true;
form2.Button67.Visible:=true;
edit20.SetFocus;
end;
end;

```

Administrator \ Inpatient \ Add Record

```

procedure TForm2.Button75Click(Sender: TObject);
type
inpatientng1=set of '0'..'9';
var
inpatientng:inpatientng1;
inpatientfor:integer;
inpatientfor1:integer;
inpatientfor2:integer;
inpatientv1:integer;
inpatientno3:integer;
begin
inpatientng=['0'..'9'];
adotable1.close;
adotable1.Open;
if(adotable1.Locate('SSN',edit30.Text,[locaseinsensitive])=false) then
begin
application.MessageBox('This SSN not included in Patient Table. Please check
again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit30.text:="";
edit30.setfocus;
exit;
end;
if (maskedit4.text[1] in inpatientng) and (maskedit4.text[2] in inpatientng) then
begin
maskedit4.SelStart:=0;
maskedit4.SelLength:=2;
if (maskedit4.seltext='00') then
begin

```

```

    application.messagebox('Please check day on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit4.setfocus;
    exit;
end;
if (strtoint(maskedit4.seltext)>31) then
begin
    application.messagebox('Please check day on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit4.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check day on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit4.setfocus;
    exit;
end;
if (maskedit4.text[4] in inpatientng) and (maskedit4.text[5] in inpatientng) then
begin
    maskedit4.SelStart:=3;
    maskedit4.SelLength:=2;
    if (maskedit4.seltext='00') then
begin
        application.messagebox('Please check month on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit4.setfocus;
        exit;
    end;
    if (strtoint(maskedit4.seltext)>12) then
begin
        application.messagebox('Please check month on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit4.setfocus;
        exit;
    end;
end
end
else
begin
    application.MessageBox('Please check month on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit4.setfocus;
    exit;
end;
if (maskedit4.text[7] in inpatientng) and (maskedit4.text[8] in inpatientng)
and (maskedit4.text[9] in inpatientng) and (maskedit4.text[10] in inpatientng) then

```



```

begin
maskedit4.SelStart:=6;
maskedit4.SelLength:=4;
if (maskedit4.seltext='0000') then
begin
application.messagebox('Please check year on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
maskedit4.setfocus;
exit;
end;
edit83.SelStart:=6;
edit83.SelLength:=4;
if (strtoint(maskedit4.seltext)>strtoint(edit83.seltext)) then
begin
application.MessageBox('Please check year on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
maskedit4.setfocus;
exit;
end;
end
else
begin
application.MessageBox('Please check year on Admission Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
maskedit4.setfocus;
exit;
end;
if (maskedit5.text[1] in inpatientng) and (maskedit5.text[2] in inpatientng) then
begin
maskedit5.SelStart:=0;
maskedit5.SelLength:=2;
if (maskedit5.seltext='00') then
begin
application.messagebox('Please check day on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
maskedit5.setfocus;
exit;
end;
if (strtoint(maskedit5.seltext)>31) then
begin
application.messagebox('Please check day on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
maskedit5.setfocus;
exit;
end;
end
else
begin

```

```

    application.MessageBox('Please check day on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit5.setfocus;
    exit;
end;
if (maskedit5.text[4] in inpatientng) and (maskedit5.text[5] in inpatientng) then
begin
    maskedit5.SelStart:=3;
    maskedit5.SelLength:=2;
    if (maskedit5.seltext='00') then
        begin
            application.messagebox('Please check month on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit5.setfocus;
            exit;
        end;
    if (strtoint(maskedit5.seltext)>12) then
        begin
            application.messagebox('Please check month on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit5.setfocus;
            exit;
        end;
    end
else
begin
    application.MessageBox('Please check month on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit5.setfocus;
    exit;
end;
if (maskedit5.text[7] in inpatientng) and (maskedit5.text[8] in inpatientng)
and (maskedit5.text[9] in inpatientng) and (maskedit5.text[10] in inpatientng) then
begin
    maskedit5.SelStart:=6;
    maskedit5.SelLength:=4;
    if (maskedit5.seltext='0000') then
        begin
            application.messagebox('Please check year on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit5.setfocus;
            exit;
        end;
    edit83.SelStart:=6;
    edit83.SelLength:=4;
    if (strtoint(maskedit5.seltext)>strtoint(edit83.seltext)) then
        begin

```



```

    application.MessageBox('Please check year on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit5.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check year on Discharge Date field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit5.setfocus;
    exit;
end;
maskedit4.SelStart:=6;
maskedit4.sellength:=4;
maskedit5.selstart:=6;
maskedit5.sellength:=4;
if (maskedit5.seltext<maskedit4.seltext) then
begin
    application.MessageBox('Please check Discharge year field','Warning for
Discharge field',MB_OK+48+MB_SYSTEMMODAL);
    maskedit5.setfocus;
    exit;
end;
if (maskedit5.seltext=maskedit4.seltext) then
begin
    maskedit4.selstart:=3;
    maskedit4.SelLength:=2;
    maskedit5.selstart:=3;
    maskedit5.sellength:=2;
    if (maskedit5.seltext<maskedit4.seltext) then
begin
        application.MessageBox('Please check Discharge month field','Warning for
Discharge field',MB_OK+48+MB_SYSTEMMODAL);
        maskedit5.setfocus;
        exit;
end;
if (maskedit5.seltext=maskedit4.seltext) then
begin
        maskedit4.selstart:=0;
        maskedit4.sellength:=2;
        maskedit5.SelStart:=0;
        maskedit5.sellength:=2;
        if (maskedit5.seltext<maskedit4.SelText) then
begin
            application.MessageBox('Please check Discharge day field','Warning for
Discharge',MB_OK+48+MB_SYSTEMMODAL);
            maskedit5.setfocus;

```

```

        exit;
    end;
end;
end;
if (edit33.gettextlen=0) then
begin
    application.MessageBox('Please check value on Department id field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit33.setfocus;
    exit;
end;
adotable8.close;
adotable8.Open;
if (adotable8.Locate('dep_id',edit33.Text,[locaseinsensitive])=false) then
begin
    application.MessageBox('This Department id not included in Department Table.
Please check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
    edit33.SetFocus;
    exit;
end;
if (radiobutton4.checked=true) then
begin
    if (edit34.gettextlen=0) then
begin
        application.MessageBox('Please check value on Ward id field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit34.SetFocus;
        exit;
    end
else
begin
        adotable5.close;
        adotable5.Open;
        if (adotable5.Locate('ward_id',edit34.Text,[locaseinsensitive])=false) then
begin
            application.MessageBox('This Ward id not included in Ward Table. Please
check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
            edit34.setfocus;
            exit;
        end
    else
begin
        if (strtoint(edit33.text)<>adotable5.FieldValues['dep_id']) then
begin
            application.MessageBox('This Ward id not belong to that Department.
Please check again','Check Value',MB_OK+48+MB_SYSTEMMODAL);
            edit34.SetFocus;
            exit;
        end
    end
end
end

```



```

        end;
    end;
end;
if (edit35.gettextlen=0) then
begin
    application.MessageBox('Please check value on Bed No field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit35.SetFocus;
    exit;
end
else
begin
    adotable6.close;
    adotable6.Open;
    if (adotable6.Locate('bed_no',edit35.Text,[locaseinsensitive])=false) then
        begin
            application.MessageBox('This Bed No not included in Bed Availability
Table. Please check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
            edit35.setfocus;
            exit;
        end
    else
        begin
            inpatientv1:=adotable6.RecordCount;
            adotable6.First;
            for inpatientfor:=0 to inpatientv1 do
                begin
                    if ((adotable6.FieldValues['dep_id']=edit33.text) and
                        (adotable6.FieldValues['ward_id']=edit34.Text) and
                        (adotable6.FieldValues['bed_no']=edit35.Text) and
                        (adotable6.FieldValues['status']='E')) then
                        begin
                            adotable6.RecNo:=inpatientfor+1;
                            adotable6.Edit;
                            adotable6.FieldValues['status']:='F';
                            adotable6.Post;
                            adotable3.Open;
                            adotable3.Append;
                            adotable3.edit;
                            adotable3.FieldValues['patient_no']:=edit29.Text;
                            adotable3.FieldValues['SSN']:=edit30.Text;
                            adotable3.FieldValues['admission_date']:=maskedit4.Text;
                            adotable3.FieldValues['discharge_date']:=maskedit5.Text;
                            adotable3.FieldValues['ward_id']:=edit34.Text;
                            adotable3.FieldValues['bed_no']:=edit35.Text;
                            adotable3.FieldValues['room_no']:=null;
                            adotable3.FieldValues['dep_id']:=edit33.Text;
                            adotable3.Post;

```

```

form2.Edit29.text:="";
form2.Edit30.text:="";
form2.MaskEdit4.text:="";
form2.MaskEdit5.text:="";
form2.Edit33.text:="";
form2.Edit34.text:="";
form2.Edit35.text:="";
form2.Edit36.text:="";
form2.Label40.Visible:=false;
form2.Label41.Visible:=false;
form2.Label42.Visible:=false;
form2.Label43.Visible:=false;
form2.Panel1.visible:=false;
form2.Label47.Visible:=false;
form2.Edit29.Visible:=false;
form2.Edit30.Visible:=false;
form2.MaskEdit4.Visible:=false;
form2.MaskEdit5.Visible:=false;
form2.Edit33.Visible:=false;
form2.Edit34.Visible:=false;
form2.Edit35.Visible:=false;
form2.Edit36.Visible:=false;
form2.Button75.Visible:=false;
form2.Button77.Visible:=false;
form2.Button76.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want
add another record ?','Add
Record',MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=true;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;
    form2.tabsheet16.tabvisible:=false;
    form2.tabsheet17.tabvisible:=false;
    form2.tabsheet18.tabvisible:=false;
    form2.tabsheet19.tabvisible:=false;

```



```

form2.tabsheet20.tabvisible:=false;
form2.tabsheet21.tabvisible:=false;
form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;
form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.TabVisible:=false;
form2.TabSheet36.TabVisible:=false;
form2.TabSheet37.TabVisible:=false;
form2.TabSheet38.TabVisible:=false;
form2.TabSheet39.TabVisible:=false;
form2.TabSheet40.TabVisible:=false;
exit;
end
else
begin
form2.Label40.Visible:=true;
form2.Label41.Visible:=true;
form2.Label42.Visible:=true;
form2.Label43.Visible:=true;
form2.Panel1.visible:=true;
form2.Label47.Visible:=true;
form2.Edit29.Visible:=true;
form2.Edit30.Visible:=true;
form2.MaskEdit4.Visible:=true;
form2.MaskEdit5.Visible:=true;
form2.Edit33.Visible:=true;
form2.Edit34.Visible:=true;
form2.Edit35.Visible:=true;
form2.Edit36.Visible:=true;
inpatientno3:=0;
inpatientv1:=adotable3.RecordCount;
adotable3.First;
for inpatientfor1:=0 to inpatientv1 do
begin
if (inpatientno3<adotable3.FieldValues['patient_no']) then
begin
inpatientno3:=adotable3.FieldValues['patient_no'];
adotable3.Next;

```

```

        end;
        if (inpatientno3>adotable3.FieldValues['patient_no']) then
            begin
                adotable3.Next;
            end;
        end;
        edit29.Text:=inttostr(inpatientno3+1);
        form2.Button75.Visible:=true;
        form2.Button77.Visible:=true;
        form2.Button76.Visible:=true;
        edit30.SetFocus;
        exit;
    end;
    end;
    adotable6.Next;
    end;
    inpatientv1:=adotable6.RecordCount;
    adotable6.first;
    for inpatientfor2:=0 to inpatientv1 do
        begin
            if ((adotable6.FieldValues['dep_id']=edit33.text) and
                (adotable6.FieldValues['ward_id']=edit34.Text) and
                (adotable6.FieldValues['bed_no']=edit35.Text) and
                (adotable6.FieldValues['status']='F')) then
                begin
                    application.messagebox('This bed is full. Please choose another
one.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
                    edit35.SetFocus;
                    exit;
                end;
            end;
            adotable6.Next;
        end;
    end;
    end;
    end;
    if (radiobutton5.checked=true) then
        begin
            if (edit36.gettextlen=0) then
                begin
                    application.MessageBox('Please fill room number field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                    edit36.SetFocus;
                    exit;
                end;
            end;
            adotable4.close;
            adotable4.Open;
            if (adotable4.Locate('room_no',edit36.Text,[locaseinsensitive])=false) then
                begin

```



```

    application.MessageBox('This room number not included in Hospital table !
Please check again','Check Value',MB_OK+48+MB_SYSTEMMODAL);
    edit36.setfocus;
    exit;
end
else
begin
    if (edit33.text<>adotable4.FieldValues['dep_id']) then
        begin
            application.MessageBox('This Room number not belong to that
Department','Check Value',MB_OK+48+MB_SYSTEMMODAL);
            edit33.setfocus;
            exit;
        end;
        if (adotable4.FieldValues['status']='F') then
            begin
                application.MessageBox('This Room is full, please choose another
one','Check Value',MB_OK+48+MB_SYSTEMMODAL);
                edit36.SetFocus;
                exit;
            end;
            if (adotable4.FieldValues['status']='E') then
                begin
                    adotable4.edit;
                    adotable4.FieldValues['status']:='F';
                    adotable4.Post;
                    adotable3.Open;
                    adotable3.Append;
                    adotable3.edit;
                    adotable3.FieldValues['patient_no']:=edit29.Text;
                    adotable3.FieldValues['SSN']:=edit30.Text;
                    adotable3.FieldValues['admission_date']:=maskedit4.Text;
                    adotable3.FieldValues['discharge_date']:=maskedit5.Text;
                    adotable3.FieldValues['ward_id']:=null;
                    adotable3.FieldValues['bed_no']:=null;
                    adotable3.FieldValues['room_no']:=edit36.Text;
                    adotable3.FieldValues['dep_id']:=edit33.Text;
                    adotable3.Post;
                    form2.Edit29.text:="";
                    form2.Edit30.text:="";
                    form2.MaskEdit4.text:="";
                    form2.MaskEdit5.text:="";
                    form2.Edit33.text:="";
                    form2.Edit34.text:="";
                    form2.Edit35.text:="";
                    form2.Edit36.text:="";
                    form2.Label40.Visible:=false;
                    form2.Label41.Visible:=false;

```

```

form2.Label42.Visible:=false;
form2.Label43.Visible:=false;
form2.Panel1.Visible:=false;
form2.Label47.Visible:=false;
form2.Edit29.Visible:=false;
form2.Edit30.Visible:=false;
form2.MaskEdit4.Visible:=false;
form2.MaskEdit5.Visible:=false;
form2.Edit33.Visible:=false;
form2.Edit34.Visible:=false;
form2.Edit35.Visible:=false;
form2.Edit36.Visible:=false;
form2.Button75.Visible:=false;
form2.Button77.Visible:=false;
form2.Button76.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add
another record ?', 'Add Record', MB_YESNO+32+MB_SYSTEMMODAL)=IDNO)
then

```

```

begin
form2.tabsheet1.tabvisible:=false;
form2.tabsheet2.tabvisible:=false;
form2.tabsheet3.tabvisible:=true;
form2.tabsheet4.tabvisible:=false;
form2.tabsheet5.tabvisible:=false;
form2.tabsheet6.tabvisible:=false;
form2.tabsheet7.tabvisible:=false;
form2.tabsheet8.tabvisible:=false;
form2.tabsheet9.tabvisible:=false;
form2.tabsheet10.tabvisible:=false;
form2.tabsheet11.tabvisible:=false;
form2.tabsheet12.tabvisible:=false;
form2.tabsheet13.tabvisible:=false;
form2.tabsheet14.tabvisible:=false;
form2.tabsheet15.tabvisible:=false;
form2.tabsheet16.tabvisible:=false;
form2.tabsheet17.tabvisible:=false;
form2.tabsheet18.tabvisible:=false;
form2.tabsheet19.tabvisible:=false;
form2.tabsheet20.tabvisible:=false;
form2.tabsheet21.tabvisible:=false;
form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;

```



```

form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.TabVisible:=false;
form2.TabSheet36.TabVisible:=false;
form2.TabSheet37.TabVisible:=false;
form2.TabSheet38.TabVisible:=false;
form2.TabSheet39.TabVisible:=false;
form2.TabSheet40.TabVisible:=false;
exit;
end
else
begin
form2.Label40.Visible:=true;
form2.Label41.Visible:=true;
form2.Label42.Visible:=true;
form2.Label43.Visible:=true;
form2.Panel1.visible:=true;
form2.Label47.Visible:=true;
form2.Edit29.Visible:=true;
form2.Edit30.Visible:=true;
form2.MaskEdit4.Visible:=true;
form2.MaskEdit5.Visible:=true;
form2.Edit33.Visible:=true;
form2.Edit34.Visible:=true;
form2.Edit35.Visible:=true;
form2.Edit36.Visible:=true;
inpatientno3:=0;
inpatientv1:=adotable3.RecordCount;
adotable3.First;
for inpatientfor1:=0 to inpatientv1 do
begin
if (inpatientno3<adotable3.FieldValues['patient_no']) then
begin
inpatientno3:=adotable3.FieldValues['patient_no'];
adotable3.Next;
end;
if (inpatientno3>adotable3.FieldValues['patient_no']) then
begin
adotable3.Next;
end;
end;
edit29.Text:=inttostr(inpatientno3+1);
form2.Button75.Visible:=true;
form2.Button77.Visible:=true;
form2.Button76.Visible:=true;

```

```

        edit30.SetFocus;
    exit;
end;
end;
end;
end;
end;

```

Administrator \ Room \ Add Record

```

procedure TForm2.Button88Click(Sender: TObject);
var
    roomv1:integer;
    roomfor:integer;
    roomno1:integer;
begin
    if edit6.gettextlen=0 then
    begin
        application.messagebox('Please fill Department id field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit6.setfocus;
        exit;
    end;
    adotable8.close;
    adotable8.Open;
    if (adotable8.Locate('dep_id',edit6.Text,[locaseinsensitive])=false) then
    begin
        application.messagebox('This Department id not included in Department Table.
Please check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
        edit6.SetFocus;
        exit;
    end;
    adotable4.close;
    adotable4.Open;
    adotable4.append;
    adotable4.fieldvalues['room_no']:=edit5.Text;
    adotable4.FieldValues['dep_id']:=edit6.Text;
    adotable4.FieldValues['status']:= 'E';
    adotable4.Post;
    edit5.Text:='';
    edit6.Text:='';
    label18.Visible:=false;
    label20.Visible:=false;
    label21.Visible:=false;
    edit5.Visible:=false;
    edit6.Visible:=false;
    combobox3.Visible:=false;
    button88.Visible:=false;

```



```

button89.Visible:=false;
button90.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?','Add Record',MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=true;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;
    form2.tabsheet16.tabvisible:=false;
    form2.tabsheet17.tabvisible:=false;
    form2.tabsheet18.tabvisible:=false;
    form2.tabsheet19.tabvisible:=false;
    form2.tabsheet20.tabvisible:=false;
    form2.tabsheet21.tabvisible:=false;
    form2.tabsheet22.tabvisible:=false;
    form2.tabsheet23.tabvisible:=false;
    form2.tabsheet24.tabvisible:=false;
    form2.tabsheet25.tabvisible:=false;
    form2.tabsheet26.tabvisible:=false;
    form2.tabsheet27.tabvisible:=false;
    form2.tabsheet28.tabvisible:=false;
    form2.tabsheet29.tabvisible:=false;
    form2.tabsheet30.tabvisible:=false;
    form2.tabsheet31.tabvisible:=false;
    form2.tabsheet32.tabvisible:=false;
    form2.tabsheet33.tabvisible:=false;
    form2.tabsheet34.tabvisible:=false;
    form2.TabSheet35.Tab Visible:=false;
    form2.TabSheet36.Tab Visible:=false;
    form2.TabSheet37.Tab Visible:=false;
    form2.TabSheet38.Tab Visible:=false;
    form2.TabSheet39.Tab Visible:=false;
    form2.TabSheet40.Tab Visible:=false;
end
else
begin

```

```

form2.Label18.Visible:=true;
form2.Label20.Visible:=true;
form2.Label21.Visible:=true;
form2.edit5.Visible:=true;
form2.Edit6.Visible:=true;
form2.ComboBox3.Visible:=true;
form2.Button88.Visible:=true;
form2.Button89.Visible:=true;
form2.button90.Visible:=true;
adotable4.close;
adotable4.Open;
roomno1:=0;
roomv1:=adotable4.RecordCount;
adotable4.First;
for roomfor:=0 to roomv1 do
begin
  if (roomno1<adotable4.FieldValues['room_no']) then
  begin
    roomno1:=adotable4.fieldvalues['room_no'];
    adotable4.Next;
  end;
  if (roomno1>adotable4.FieldValues['room_no']) then
  begin
    adotable4.Next;
  end;
end;
edit5.Text:=inttostr(roomno1+1);
form2.Edit6.SetFocus;
end;
end;

```

Administrator \ Ward \ Add Record

```

procedure TForm2.Button98Click(Sender: TObject);
type
wardcg1=set of 'a'..'z';
var
wardcg:wardcg1;
wardfor:integer;
wardv1:integer;
wardno1:integer;
begin
wardcg:=['a'..'z'];
adotable8.close;
adotable8.Open;
if (adotable8.Locate('dep_id',edit17.Text,[locaseinsensitive])=false) then
begin

```



```

    application.messagebox('This Department id not included in Department Table.
Please check again','Check Value',MB_OK+48+MB_SYSTEMMODAL);
    edit17.SetFocus;
    exit;
end;
if (edit19.gettextlen=0) then
begin
    application.messagebox('Please fill Ward Name field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit19.SetFocus;
    exit;
end;
if (edit31.gettextlen=0) then
begin
    application.messagebox('Please fill Total Bed field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit31.SetFocus;
    exit;
end;
wardv1:=edit31.GetTextLen;
for wardfor:=0 to wardv1 do
begin
    if (edit31.Text[wardfor] in wardcg) then
    begin
        application.messagebox('Please enter numeric value on Total Bed field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit31.SetFocus;
        exit;
    end;
end;
if (strtoint(edit31.Text)>=10) then
begin
    application.messagebox('Please enter value, less than 10 on Total Bed
field.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
    edit31.SetFocus;
    exit;
end;
adotable5.close;
adotable5.Open;
adotable5.Append;
adotable5.FieldValues['ward_id']:=edit16.Text;
adotable5.FieldValues['dep_id']:=edit17.Text;
adotable5.FieldValues['ward_name']:=edit19.Text;
adotable5.FieldValues['total_bed']:=edit31.Text;
adotable5.post;
wardv1:=strtoint(edit31.text);
adotable6.close;
for wardfor:=1 to wardv1 do

```

```

begin
    adotable6.open;
    adotable6.Append;
    adotable6.FieldValues['ward_id']:=edit16.Text;
    adotable6.FieldValues['bed_no']:=wardfor;
    adotable6.FieldValues['dep_id']:=edit17.Text;
    adotable6.FieldValues['status']:= 'E';
    adotable6.post;
end;
edit16.Text:="";
edit17.Text:="";
edit19.Text:="";
edit31.Text:="";
label59.Visible:=false;
label60.Visible:=false;
label61.Visible:=false;
label62.Visible:=false;
edit16.Visible:=false;
edit17.Visible:=false;
edit19.Visible:=false;
edit31.Visible:=false;
button98.Visible:=false;
button99.Visible:=false;
button100.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?', 'Add Record', MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=true;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;
    form2.tabsheet16.tabvisible:=false;
    form2.tabsheet17.tabvisible:=false;
    form2.tabsheet18.tabvisible:=false;
    form2.tabsheet19.tabvisible:=false;
    form2.tabsheet20.tabvisible:=false;
    form2.tabsheet21.tabvisible:=false;

```



```

form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;
form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.TabVisible:=false;
form2.TabSheet36.TabVisible:=false;
form2.TabSheet37.TabVisible:=false;
form2.TabSheet38.TabVisible:=false;
form2.TabSheet39.TabVisible:=false;
form2.TabSheet40.TabVisible:=false;
end
else
begin
form2.Label59.Visible:=true;
form2.Label60.Visible:=true;
form2.Label61.Visible:=true;
form2.Label62.Visible:=true;
form2.Edit16.Visible:=true;
form2.Edit17.Visible:=true;
form2.Edit19.Visible:=true;
form2.Edit31.Visible:=true;
form2.Button98.Visible:=true;
form2.Button99.Visible:=true;
form2.Button100.Visible:=true;
adotable5.close;
adotable5.Open;
wardno1:=0;
wardv1:=adotable5.RecordCount;
adotable5.First;
for wardfor:=0 to wardv1 do
begin
if (wardno1<adotable5.FieldValues['ward_id']) then
begin
wardno1:=adotable5.fieldvalues['ward_id'];
adotable5.Next;
end;
if (wardno1>adotable5.FieldValues['ward_id']) then
begin
adotable5.Next;

```

```

    end;
    end;
    edit16.Text:=inttostr(wardno1+1);
    edit17.SetFocus;
    end;
end;

```

Administrator \ Doctor \ Add Record

```

procedure TForm2.Button110Click(Sender: TObject);
type
doctornrg1=set of '0'..'9';
doctorcrg1=set of 'a'..'z';
var
doctornrg:doctornrg1;
doctorcrg:doctorcrg1;
doctorfor:integer;
doctorv1:integer;
doctorno1:integer;
begin
doctornrg=['0'..'9'];
doctorcrg=['a'..'z'];
doctorv1:=edit50.GetTextLen;
if (doctorv1=0) then
begin
application.messagebox('Please enter value on First Name field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit50.SetFocus;
exit;
end;
for doctorfor:=0 to doctorv1 do
begin
if (edit50.Text[doctorfor] in doctornrg) then
begin
application.messagebox('Please enter character value on First Name
field.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit50.SetFocus;
exit;
end;
end;
doctorv1:=edit51.GetTextLen;
if (doctorv1=0) then
begin
application.messagebox('Please enter value on Surname field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit51.SetFocus;
exit;
end;
end;

```



```

for doctorfor:=0 to doctorv1 do
begin
  if (edit51.Text[doctorfor] in doctornrg) then
    begin
      application.messagebox('Please enter character value on Surname field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
      edit51.SetFocus;
      exit;
    end;
  end;
doctorv1:=edit52.GetTextLen;
if (doctorv1=0) then
begin
  application.messagebox('Please enter value on Phone field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
  edit52.SetFocus;
  exit;
end;
for doctorfor:=0 to doctorv1 do
begin
  if (edit52.Text[doctorfor] in doctorcrg) then
    begin
      application.messagebox('Please enter numeric value on Phone field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
      edit52.SetFocus;
      exit;
    end;
  end;
end;
if (combobox5.itemindex=-1) then
begin
  application.messagebox('Please choose Sex(Male\Female)','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
  combobox5.SetFocus;
  exit;
end;
adotable8.close;
adotable8.Open;
if (adotable8.Locate('dep_id',edit53.Text,[locaseinsensitive])=false) then
begin
  application.messagebox('This Department id not included in Department Table.
Please check again','Check Value',MB_OK+48+MB_SYSTEMMODAL);
  edit53.SetFocus;
  exit;
end;
adotable7.close;
adotable7.Open;
adotable7.append;
adotable7.FieldValues['doctor_id']:=edit49.Text;

```

```

adotable7.FieldValues['l_name']:=edit51.Text;
adotable7.FieldValues['f_name']:=edit50.Text;
adotable7.fieldvalues['phone']:=edit52.Text;
if (combobox5.itemindex=0) then
begin
    adotable7.FieldValues['sex']:= 'M';
end
else
begin
    adotable7.FieldValues['sex']:= 'F';
end;
adotable7.FieldValues['dep_id']:=edit53.Text;
adotable7.Post;
edit49.Text:="";
edit50.Text:="";
edit51.Text:="";
edit52.Text:="";
combobox5.itemindex:=-1;
edit53.Text:="";
label68.Visible:=false;
label69.Visible:=false;
label70.Visible:=false;
label71.Visible:=false;
label72.Visible:=false;
label73.Visible:=false;
edit49.Visible:=false;
edit50.Visible:=false;
edit51.Visible:=false;
edit52.Visible:=false;
combobox5.Visible:=false;
edit53.Visible:=false;
button110.Visible:=false;
button111.Visible:=false;
button112.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?','Add Record',MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=true;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;

```



```

form2.tabsheet12.tabvisible:=false;
form2.tabsheet13.tabvisible:=false;
form2.tabsheet14.tabvisible:=false;
form2.tabsheet15.tabvisible:=false;
form2.tabsheet16.tabvisible:=false;
form2.tabsheet17.tabvisible:=false;
form2.tabsheet18.tabvisible:=false;
form2.tabsheet19.tabvisible:=false;
form2.tabsheet20.tabvisible:=false;
form2.tabsheet21.tabvisible:=false;
form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;
form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.TabVisible:=false;
form2.TabSheet36.TabVisible:=false;
form2.TabSheet37.TabVisible:=false;
form2.TabSheet38.TabVisible:=false;
form2.TabSheet39.TabVisible:=false;
form2.TabSheet40.TabVisible:=false;
end
else
begin
label68.Visible:=true;
label69.Visible:=true;
label70.Visible:=true;
label71.Visible:=true;
label72.Visible:=true;
label73.Visible:=true;
edit49.Visible:=true;
edit50.Visible:=true;
edit51.Visible:=true;
edit52.Visible:=true;
combobox5.Visible:=true;
edit53.Visible:=true;
button110.Visible:=true;
button111.Visible:=true;
button112.Visible:=true;
adotable7.close;

```

```

adotable7.Open;
doctorno1:=0;
doctorv1:=adotable7.RecordCount;
adotable7.First;
for doctorfor:=0 to doctorv1 do
begin
if (doctorno1<adotable7.FieldValues['doctor_id']) then
begin
doctorno1:=adotable7.fieldvalues['doctor_id'];
adotable7.Next;
end;
if (doctorno1>adotable7.FieldValues['doctor_id']) then
begin
adotable7.Next;
end;
end;
edit49.Text:=inttostr(doctorno1+1);
edit50.setfocus;
end;
end;

```

Administrator \ Department \ Add Record

```

procedure TForm2.Button120Click(Sender: TObject);
var
depv1:integer;
depno1:integer;
depfor:integer;
begin
if (edit61.gettextlen=0) then
begin
application.messagebox('Please enter value on Department Name field.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit61.SetFocus;
exit;
end;
if (edit62.gettextlen=0) then
begin
application.messagebox('Please enter value on Department Description field.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit62.SetFocus;
exit;
end;
adotable8.close;
adotable8.Open;
adotable8.Append;
adotable8.FieldValues['dep_id']:=edit60.Text;
adotable8.FieldValues['dep_name']:=edit61.Text;

```



```

adotable8.FieldValues['dep_des']:=edit62.Text;
adotable8.Post;
form2.edit60.Text:="";
form2.Edit61.Text:="";
form2.edit62.Text:="";
form2.label81.Visible:=false;
form2.Label82.Visible:=false;
form2.Label83.Visible:=false;
form2.Edit60.Visible:=false;
form2.edit61.Visible:=false;
form2.edit62.Visible:=false;
button120.Visible:=false;
button121.Visible:=false;
button122.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?', 'Add Record', MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=true;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=false;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;
    form2.tabsheet16.tabvisible:=false;
    form2.tabsheet17.tabvisible:=false;
    form2.tabsheet18.tabvisible:=false;
    form2.tabsheet19.tabvisible:=false;
    form2.tabsheet20.tabvisible:=false;
    form2.tabsheet21.tabvisible:=false;
    form2.tabsheet22.tabvisible:=false;
    form2.tabsheet23.tabvisible:=false;
    form2.tabsheet24.tabvisible:=false;
    form2.tabsheet25.tabvisible:=false;
    form2.tabsheet26.tabvisible:=false;
    form2.tabsheet27.tabvisible:=false;
    form2.tabsheet28.tabvisible:=false;
    form2.tabsheet29.tabvisible:=false;
    form2.tabsheet30.tabvisible:=false;
    form2.tabsheet31.tabvisible:=false;

```

```

form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.Tab Visible:=false;
form2.TabSheet36.Tab Visible:=false;
form2.TabSheet37.Tab Visible:=false;
form2.TabSheet38.Tab Visible:=false;
form2.TabSheet39.Tab Visible:=false;
form2.TabSheet40.Tab Visible:=false;
end
else
begin
form2.label81.Visible:=true;
form2.Label82.Visible:=true;
form2.Label83.Visible:=true;
form2.Edit60.Visible:=true;
form2.edit61.Visible:=true;
form2.edit62.Visible:=true;
button120.Visible:=true;
button121.Visible:=true;
button122.Visible:=true;
adotable8.Close;
adotable8.Open;
depno1:=0;
depv1:=adotable8.RecordCount;
adotable8.First;
for depfor:=0 to depv1 do
begin
if (depno1<adotable8.FieldValues['dep_id']) then
begin
depno1:=adotable8.fieldvalues['dep_id'];
adotable8.Next;
end;
if (depno1>adotable8.FieldValues['dep_id']) then
begin
adotable8.Next;
end;
end;
edit60.Text:=inttostr(depno1+1);
edit61.setfocus;
end;
end;

```

Administrator \ Inpatient Disease Detail \ Add Record

```

procedure TForm2.Button136Click(Sender: TObject);
type
detailng1=set of '0'..'9';

```



```

var
detailng:detailng1;
detailv1:integer;
detailfor:integer;
find:boolean;
begin
detailng:=['0'..'9'];
if (edit67.gettextlen=0) then
begin
application.MessageBox('Please fill SSN field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit67.setfocus;
exit;
end;
if (edit68.gettextlen=0) then
begin
application.MessageBox('Please fill Patient No field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit68.setfocus;
exit;
end;
find:=false;
adotable3.Close;
adotable3.Open;
detailv1:=adotable3.RecordCount;
adotable3.first;
for detailfor:=0 to detailv1 do
begin
if (adotable3.fieldvalues['SSN']=edit67.Text) and
(adotable3.fieldvalues['patient_no']=edit68.Text) then
begin
find:=true;
end;
adotable3.Next;
end;
if (find=false) then
begin
application.MessageBox('Please check SSN and Patient No.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit67.setfocus;
exit;
end;
if (maskedit8.text[1] in detailng) and (maskedit8.text[2] in detailng) then
begin
maskedit8.SelStart:=0;
maskedit8.SelLength:=2;
if (maskedit8.seltext='00') then
begin

```

```

    application.messagebox('Please check day on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit8.setfocus;
    exit;
end;
if (strtoint(maskedit8.seltext)>31) then
begin
    application.messagebox('Please check day on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit8.setfocus;
    exit;
end;
end
else
begin
    application.MessageBox('Please check day on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit8.setfocus;
    exit;
end;
if (maskedit8.text[4] in detailng) and (maskedit8.text[5] in detailng) then
begin
    maskedit8.SelStart:=3;
    maskedit8.SelLength:=2;
    if (maskedit8.seltext='00') then
    begin
        application.messagebox('Please check month on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit8.setfocus;
        exit;
    end;
    if (strtoint(maskedit8.seltext)>12) then
    begin
        application.messagebox('Please check month on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit8.setfocus;
        exit;
    end;
end
else
begin
    application.MessageBox('Please check month on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit8.setfocus;
    exit;
end;
if (maskedit8.text[7] in detailng) and (maskedit8.text[8] in detailng)
and (maskedit8.text[9] in detailng) and (maskedit8.text[10] in detailng) then

```



```

begin
    maskedit8.SelStart:=6;
    maskedit8.SelLength:=4;
    if (maskedit8.seltext='0000') then
        begin
            application.messagebox('Please check year on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit8.setfocus;
            exit;
        end;
    edit83.SelStart:=6;
    edit83.SelLength:=4;
    if (strtoint(maskedit8.seltext)>strtoint(edit83.seltext)) then
        begin
            application.MessageBox('Please check year on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
            maskedit8.setfocus;
            exit;
        end;
    end
else
    begin
        application.MessageBox('Please check year on Date of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit8.setfocus;
        exit;
    end;
if (maskedit9.text[1] in detailing) and (maskedit9.text[2] in detailing) then
    begin
        maskedit9.SelStart:=0;
        maskedit9.SelLength:=2;
        if (strtoint(maskedit9.seltext)>23) then
            begin
                application.messagebox('Please check hour on Time of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                maskedit9.setfocus;
                exit;
            end;
        end
    end
else
    begin
        application.MessageBox('Please check hour on Time of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        maskedit9.setfocus;
        exit;
    end;
if (maskedit9.text[4] in detailing) and (maskedit9.text[5] in detailing) then
    begin

```

```

maskedit9.SelStart:=3;
maskedit9.SelLength:=2;
if (strtoint(maskedit9.seltext)>59) then
begin
    application.messagebox('Please check minute on Time of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit9.setfocus;
    exit;
end;
else
begin
    application.MessageBox('Please check minute on Time of Check field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    maskedit9.setfocus;
    exit;
end;
adotable9.close;
adotable9.Open;
detailv1:=adotable9.RecordCount;
adotable9.first;
for detailfor:=0 to detailv1 do
begin
    if (adotable9.fieldvalues['SSN']=edit67.Text) and
        (adotable9.fieldvalues['patient_no']=edit68.Text) and
        (adotable9.fieldvalues['date_of_check']=maskedit8.text) and
        (adotable9.fieldvalues['time_of_check']=maskedit9.text) then
    begin
        application.messagebox('This record already exist in Inpatient Disease Detail
Table. Please check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
        edit67.setfocus;
        exit;
    end;
    adotable9.Next;
end;
adotable7.close;
adotable7.Open;
if (adotable7.locate('doctor_id',edit69.Text,[locaseinsensitive])=false) then
begin
    application.MessageBox('This Doctor id not included in Doctor Table. Please
check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
    edit69.setfocus;
    exit;
end;
if (memo1.gettextlen=0) then
begin
    application.MessageBox('Please fill Disease field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);

```



```

memo1.setfocus;
exit;
end;
adotable9.close;
adotable9.Open;
adotable9.Append;
adotable9.FieldValues['SSN']:=edit67.Text;
adotable9.FieldValues['patient_no']:=edit68.Text;
adotable9.FieldValues['date_of_check']:=maskedit8.Text;
adotable9.FieldValues['time_of_check']:=maskedit9.Text;
adotable9.FieldValues['doctor_id']:=edit69.Text;
adotable9.FieldValues['disease']:=memo1.Text;
adotable9.FieldValues['medicine']:=memo2.Text;
adotable9.FieldValues['response']:=memo3.Text;
adotable9.post;
form2.edit67.Text:="";
form2.edit68.Text:="";
form2.maskedit8.text:="";
form2.maskedit9.Text:="";
form2.Edit69.text:="";
form2.Memo1.Text:="";
form2.memo2.Text:="";
form2.Memo3.Text:="";
form2.Label88.Visible:=false;
form2.Label89.visible:=false;
form2.label90.Visible:=false;
form2.label91.visible:=false;
form2.label92.Visible:=false;
form2.label93.Visible:=false;
form2.label94.visible:=false;
form2.label95.Visible:=false;
form2.edit67.Visible:=false;
form2.edit68.Visible:=false;
form2.MaskEdit8.Visible:=false;
form2.MaskEdit9.Visible:=false;
form2.edit69.visible:=false;
form2.memo1.Visible:=false;
form2.Memo2.visible:=false;
form2.memo3.Visible:=false;
button136.Visible:=false;
button137.Visible:=false;
button138.Visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?','Add Record',MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
form2.tabsheet1.tabvisible:=false;
form2.tabsheet2.tabvisible:=false;
form2.tabsheet3.tabvisible:=false;

```

```

form2.tabsheet4.tabvisible:=false;
form2.tabsheet5.tabvisible:=false;
form2.tabsheet6.tabvisible:=false;
form2.tabsheet7.tabvisible:=false;
form2.tabsheet8.tabvisible:=false;
form2.tabsheet9.tabvisible:=true;
form2.tabsheet10.tabvisible:=false;
form2.tabsheet11.tabvisible:=false;
form2.tabsheet12.tabvisible:=false;
form2.tabsheet13.tabvisible:=false;
form2.tabsheet14.tabvisible:=false;
form2.tabsheet15.tabvisible:=false;
form2.tabsheet16.tabvisible:=false;
form2.tabsheet17.tabvisible:=false;
form2.tabsheet18.tabvisible:=false;
form2.tabsheet19.tabvisible:=false;
form2.tabsheet20.tabvisible:=false;
form2.tabsheet21.tabvisible:=false;
form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;
form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.Tab Visible:=false;
form2.TabSheet36.Tab Visible:=false;
form2.TabSheet37.Tab Visible:=false;
form2.TabSheet38.Tab Visible:=false;
form2.TabSheet39.Tab Visible:=false;
form2.TabSheet40.Tab Visible:=false;
end
else
begin
form2.Label88.Visible:=true;
form2.Label89.visible:=true;
form2.label90.Visible:=true;
form2.label91.visible:=true;
form2.label92.Visible:=true;
form2.label93.Visible:=true;
form2.label94.visible:=true;
form2.label95.Visible:=true;

```



```

form2.edit67.Visible:=true;
form2.edit68.Visible:=true;
form2.MaskEdit8.Visible:=true;
form2.MaskEdit9.Visible:=true;
form2.edit69.visible:=true;
form2.memo1.Visible:=true;
form2.Memo2.visible:=true;
form2.memo3.Visible:=true;
button136.Visible:=true;
button137.Visible:=true;
button138.Visible:=true;
edit67.SetFocus;
end;
end;

```

Administrator \ Pharmacy \ Add Record

```

procedure TForm2.Button148Click(Sender: TObject);
type
pharmacycg1=set of 'a'..'z';
var
pharmacycg:pharmacycg1;
pharmacyfor:integer;
pharmacyv1:integer;
begin
pharmacycg:=['a'..'z'];
if (edit78.gettextlen=0) then
begin
application.MessageBox('Please fill Medicine id field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit78.setfocus;
exit;
end;
if (edit78.gettextlen<13) or (edit78.gettextlen>13) then
begin
application.MessageBox('Please enter 13 character on Medicine id field.','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
edit78.setfocus;
exit;
end;
pharmacyv1:=edit78.GetTextLen;
for pharmacyfor:=0 to pharmacyv1 do
begin
if (edit78.Text[pharmacyfor] in pharmacycg) then
begin
application.messagebox('Please enter numeric value on Medicine id
field','Check Value',MB_OK+48+MB_SYSTEMMODAL);
edit78.setfocus;

```

```

        exit;
    end;
end;
adotable10.close;
adotable10.Open;
if (adotable10.Locate('medicine_id',edit78.Text,[locaseinsensitive])=true) then
    begin
        application.messagebox('This record already exist in Pharmacy Table. Please
check again.','Check Value',MB_OK+48+MB_SYSTEMMODAL);
        edit78.setfocus;
        exit;
    end;
if (edit79.gettextlen<=3) then
    begin
        application.MessageBox('Please fill Medicine Name field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit79.setfocus;
        exit;
    end;
if (edit80.gettextlen=0) then
    begin
        application.MessageBox('Please fill Quantity field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit80.setfocus;
        exit;
    end;
pharmacyv1:=edit80.GetTextLen;
for pharmacyfor:=0 to pharmacyv1 do
    begin
        if (edit80.Text[pharmacyfor] in pharmacycg) then
            begin
                application.messagebox('Please enter numeric value on Quantity field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
                edit80.setfocus;
                exit;
            end;
        end;
    end;
if (edit81.gettextlen=0) then
    begin
        application.MessageBox('Please fill Price field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
        edit81.setfocus;
        exit;
    end;
pharmacyv1:=edit81.GetTextLen;
for pharmacyfor:=0 to pharmacyv1 do
    begin
        if (edit81.Text[pharmacyfor] in pharmacycg) then

```



```

begin
    application.messagebox('Please enter numeric value on Price field','Check
Value',MB_OK+48+MB_SYSTEMMODAL);
    edit81.setfocus;
    exit;
end;
end;
adotable10.close;
adotable10.Open;
adotable10.append;
adotable10.FieldValues['medicine_id']:=edit78.Text;
adotable10.fieldvalues['medicine_name']:=edit79.Text;
adotable10.fieldvalues['quantity']:=edit80.Text;
adotable10.FieldValues['price']:=edit81.Text;
adotable10.Post;
form2.edit78.Text:="";
form2.edit79.Text:="";
form2.edit80.Text:="";
form2.edit81.Text:="";
form2.label109.visible:=false;
form2.label110.visible:=false;
form2.label111.visible:=false;
form2.label112.visible:=false;
form2.edit78.Visible:=false;
form2.edit79.Visible:=false;
form2.edit80.Visible:=false;
form2.edit81.Visible:=false;
form2.button148.visible:=false;
form2.button149.visible:=false;
form2.button150.visible:=false;
if (application.messagebox('This record saved successfully. Do you want add another
record ?', 'Add Record',MB_YESNO+32+MB_SYSTEMMODAL)=IDNO) then
begin
    form2.tabsheet1.tabvisible:=false;
    form2.tabsheet2.tabvisible:=false;
    form2.tabsheet3.tabvisible:=false;
    form2.tabsheet4.tabvisible:=false;
    form2.tabsheet5.tabvisible:=false;
    form2.tabsheet6.tabvisible:=false;
    form2.tabsheet7.tabvisible:=false;
    form2.tabsheet8.tabvisible:=false;
    form2.tabsheet9.tabvisible:=false;
    form2.tabsheet10.tabvisible:=true;
    form2.tabsheet11.tabvisible:=false;
    form2.tabsheet12.tabvisible:=false;
    form2.tabsheet13.tabvisible:=false;
    form2.tabsheet14.tabvisible:=false;
    form2.tabsheet15.tabvisible:=false;

```

```

form2.tabsheet16.tabvisible:=false;
form2.tabsheet17.tabvisible:=false;
form2.tabsheet18.tabvisible:=false;
form2.tabsheet19.tabvisible:=false;
form2.tabsheet20.tabvisible:=false;
form2.tabsheet21.tabvisible:=false;
form2.tabsheet22.tabvisible:=false;
form2.tabsheet23.tabvisible:=false;
form2.tabsheet24.tabvisible:=false;
form2.tabsheet25.tabvisible:=false;
form2.tabsheet26.tabvisible:=false;
form2.tabsheet27.tabvisible:=false;
form2.tabsheet28.tabvisible:=false;
form2.tabsheet29.tabvisible:=false;
form2.tabsheet30.tabvisible:=false;
form2.tabsheet31.tabvisible:=false;
form2.tabsheet32.tabvisible:=false;
form2.tabsheet33.tabvisible:=false;
form2.tabsheet34.tabvisible:=false;
form2.TabSheet35.TabVisible:=false;
form2.TabSheet36.TabVisible:=false;
form2.TabSheet37.TabVisible:=false;
form2.TabSheet38.TabVisible:=false;
form2.TabSheet39.TabVisible:=false;
form2.TabSheet40.TabVisible:=false;
end
else
begin
form2.label109.visible:=true;
form2.label110.visible:=true;
form2.label111.visible:=true;
form2.label112.visible:=true;
form2.edit78.Visible:=true;
form2.edit79.Visible:=true;
form2.edit80.Visible:=true;
form2.edit81.Visible:=true;
form2.button148.visible:=true;
form2.button149.visible:=true;
form2.button150.visible:=true;
edit78.setfocus;
end;
end;

```


APPENDIX B

Hospital Management System Solutions Comparision

Many varieties of modern software use a client\server architecture, in which by one process (the client) are sent to another process (the server) for execution. Database systems are no exception, and it has become increasingly common to divide the work of a DBMS into a server process and one or more client processes.

In the simplest client\server architecture, the entire DBMS is a server, except for the query interfaces that interact with the user and send queries or other commands across to the server. For example, relational systems generally use the SQL language for representing requests from the client to the server. The DB server then sends the answer, in the form of a table or relation back to the client.

The first important applications of DBMS's were Airline Reservations Systems, Banking Systems and Corporate Records.

In Airline Reservations Systems the items of data include :

- Reservations by a single customer on a single flight
- Information about flights
- Information about ticket prices

In banking systems data items include names and addresses of customers, accounts, loans and their balances and the connection between customers and their accounts and loans.

Many early applications concerned corporate records, such as a record of each sale, information about accounts payable and receivable or information about employees. Recently in NEU developed DB system that covers many activities of the university, such as enrollment of students in courses, payment of fees, automatic calculation of GPA, etc.

1. Comparision with local market solutions :

- Dos base
- Less reporting features

- Not scalable
- Difficult to learn

2. Comparison with international market :

(Medinious Hospital Management System, The Medical Office Suite for Hospital Management System and Nestech Hospital Management System)

- Can handle accounts perfectly
- Can do billing
- Advanced graphical user interface, Difficult to learn
- Extensive reporting features
- Efficient web uploading features for information
- Very expensive
- Needs high configuration system
- Advance search options

3. My solution :

- Simple and easy to use graphical user interface
- Less expensive
- Customized reporting features
- Replaces excel, word and DOS based solutions
- Specific features for different level of operations (e.g. Reception, Doctor, Pharmacy,...)
- Easy to change for different Hospital requirements
- Scalable on any level
- Data is meeting the standards of RDMS
- ODBC (Open Database Connectivity) used

Hospital Management System Solution Comparisons by Table

Property of products	Soft Aid Solution	Medinous Solution	Nestech Solution	My Solution	Local Market Solution
For large Healthcare organization	√	√	√	X	X
Specific features for different level of operations	X	X	X	√	X
Advanced search options	√	√	√	√	X
Reporting features	√	√	√	√	X
MS SQL Server used	√	X	X	√	X
Complex Graphical User Interface	√	√	√	X	√
Easy to change for different Hospital requirements	√	√	√	√	X
Easy to learn and use	X	X	X	√	X
Expensive	√	√	√	X	X
Data is meeting the standards RDMS	√	√	√	√	X
DOS based	X	X	X	X	√
ODBC (Open Database Connectivity) used	√	√	√	√	X
Input data control	√	√	√	√	X