NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Student Registry system using WEB

Graduation Project
COM- 400

Student: Ozan DURMAZ (980450)

Supervisor: Ümit İLHAN

Nicosia-2003

ACKNOWLEDGEMENT

All the thanks are to Almighty Allah who made me able to do and complete this project successfully. It is the result of the continuous support and help of my parents and other family members who took care of me and encouraged in every way. I am thankful to my respectable teacher Mr. Umit Ilhan to guide me towards the completion of this project. In the end I am thankful to all of my friends and especially to Tariq Javed who helped me to complete my program and documentation.

ABSTRACT

An Active Server Page is a standard HTML file whose functionality is extended with additional features such as server-side scripts, objects and components. Additionally, an Active Server Page contains HTML tags that can interpreted and displayed by any web browser. Since an Active Server Page contains server-side scripts, web pages can be created with dynamic content.

ASP technology provides several built-in objects. Scripts can be made very powerful, by using these built-in objects accessible to an Active Server Page. These built-in ASP objects retrieve information from and send information to browsers. From the perspective of the Web Server, an Active Server Page is very different from a normal HTML page.

When a request is made for an Active Server Page, the browser receives a normal HTML page, which is the output of the asp file being processed. This enables an Active Server Page to be compatible with all browsers.

CONTENTS

INTRODUCTION

The frantic pace of progress on the Internet has slowed considerably in recent years as the medium has started to mature. Email and Web URLs are becoming as common as business cards and brochures. As part of the Internet, the World Wide Web is the predominant force in the growth of the global computer network. Its language, much richer than a few years ago, is still quite simple. The Web interface is attractive and friendly and is adaptable to many uses. There are Web sites for selling products, selling ideas, keeping up appearances, informing publics, distributing policy, delivering education, not to mention the vast array of sites devoted just to killing time.

HyperText Markup Language (HTML) is the language that puts the face on the Web. It consists of a variety of elements called tags, which are used for everything from defining a title to outlining a frame, from creating headings to inserting line breaks, from inserting an image to including a custom Java applet.

ASP enables the inclusion of executable scripts directly in HTML files. HTML development and scripting development becomes the same process, freeing the programmer to focus directly on the look and feel of a Web site, weaving dynamic elements into our pages as appropriate.

CHAPTER 1:

WHAT IS THE WORLD WIDE WEB

The World Wide Web (Web) is a network of information resources. The Web relies on three mechanisms to make these resources readily available to the widest possible audience:

1. A uniform naming scheme for locating resources on the Web
2. Protocols, for access to named resources over the Web
3. Hypertext, for easy navigation among resources

The ties between the three mechanisms are apparent throughout this specification.

Introduction to URIs

Every resource available on the Web -- HTML document, image, video clip, program, etc. -- has an address that may be encoded by a *Universal Resource Identifier*, or "URI".

URIs typically consist of three pieces:

1. The naming scheme of the mechanism used to access the resource.
2. The name of the machine hosting the resource.
3. The name of the resource itself, given as a path.

Consider the URI that designates the W3C Technical Reports page:
http://www.w3.org/TR

This URI may be read as follows: There is a document available via the HTTP protocol (see [RFC2616]), residing on the machine www.w3.org, accessible via the path "/TR". Other schemes you may see in HTML documents include "mailto" for email and "ftp" for FTP.

Here is another example of a URI. This one refers to a user's mailbox:

```
...this is text...
For all comments, please send email to
<A href="mailto:joe@someplace.com">Joe Cool</A>.
```

Fragment identifiers

Some URIs refer to a location within a resource. This kind of URI ends with "#" followed by an anchor identifier (called the *fragment identifier*). For instance, here is a URI pointing to an anchor named section_2:

```
http://somesite.com/html/top.html#section_2
```

Relative URIs

A *relative URI* doesn't contain any naming scheme information. Its path generally refers to a resource on the same machine as the current document. Relative URIs may contain relative path components (e.g., ".." means one level up in the hierarchy defined by the path), and may contain fragment identifiers.

Relative URIs are resolved to full URIs using a base URI. As an example of relative URI resolution, assume we have the base URI "http://www.acme.com/support/intro.html". The relative URI in the following markup for a hypertext link:

```
<A href="suppliers.html">Suppliers</A>
```

"http://www.acme.com/support/suppliers.html", while the relative URI in the following markup for an image

```
<IMG src="../icons/logo.gif" alt="logo">
```

would expand to the full URI "http://www.acme.com/icons/logo.gif".

In HTML, URIs are used to:

- Link to another document or resource
- Link to an external style sheet or script
- Include an image, object, or applet in a page,

- Create an image map
- Submit a form
- Create a frame document
- Cite an external reference
- Refer to metadata conventions describing a document
- Please consult the section on the URI type for more information about URIs.

What is HTML?

To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML (from HyperText Markup Language).

HTML gives authors the means to:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links, at the click of a button.
- Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
- Include spread-sheets, video clips, sound clips, and other applications directly in their documents.

A brief history of HTML

HTML was originally developed by Tim Berners-Lee while at CERN, and popularized by the Mosaic browser developed at NCSA. During the course of the 1990s it has blossomed with the explosive growth of the Web.

During this time, HTML has been extended in a number of ways. The Web depends on Web page authors and vendors sharing the same conventions for HTML. This has motivated joint work on specifications for HTML.

HTML 2.0 (November 1995 was developed under the aegis of the Internet Engineering Task Force (IETF) to codify common practice in late 1994. HTML+ (1993) and HTML 3.0 (1995 proposed much richer versions of

HTML. Despite never receiving consensus in standards discussions, these drafts led to the adoption of a range of new features. The efforts of the World Wide Web Consortium's HTML Working Group to codify common practice in 1996 resulted in HTML 3.2 (January 1997 Changes from HTML 3.2 are summarized in Appendix A

Most people agree that HTML documents should work well across different browsers and platforms. Achieving interoperability lowers costs to content providers since they must develop only one version of a document. If the effort is not made, there is much greater risk that the Web will devolve into a proprietary world of incompatible formats, ultimately reducing the Web's commercial potential for all participants.

Each version of HTML has attempted to reflect greater consensus among industry players so that the investment made by content providers will not be wasted and that their documents will not become unreadable in a short period of time.

HTML has been developed with the vision that all manner of devices should be able to use information on the Web: PCs with graphics displays of varying resolution and color depths, cellular telephones, hand held devices, devices for speech for output and input, computers with high or low bandwidth, and so on.

Internationalization

This version of HTML has been designed with the help of experts in the field of internationalization, so that documents may be written in every language and be transported easily around the world. This has been accomplished by incorporating which deals with the internationalization of HTML.

One important step has been the adoption of the ISO/IEC:10646 standard as the document character set for HTML. This is the world's most inclusive standard dealing with issues of the representation of international characters, text direction, punctuation, and other world language issues.

HTML now offers greater support for diverse human languages within a document. This allows for more effective indexing of documents for search engines, higher-quality typography, better text-to-speech conversion, better hyphenation, etc.

Separate structure and presentation

HTML has its roots in SGML which has always been a language for the specification of structural markup. As HTML matures, more and more of its presentational elements and attributes are being replaced by other mechanisms, in particular style sheets. Experience has shown that separating the structure of a document from its presentational aspects reduces the cost of serving a wide range of platforms, media, etc., and facilitates document revisions.

Consider universal accessibility to the Web

To make the Web more accessible to everyone, notably those with disabilities, authors should consider how their documents may be rendered on a variety of platforms: speech-based browsers, braille-readers, etc. We do not recommend that authors limit their creativity, only that they consider alternate renderings in their design.

Furthermore, authors should keep in mind that their documents may be reaching a far-off audience with different computer configurations. In order for documents to be interpreted correctly, authors should include in their documents information about the natural language and direction of the text, how the document is encoded, and other issues related to internationalization.

The HEAD element

```
<!-- %head.misc; defined earlier on as "SCRIPT|STYLE|META|LINK|OBJECT" -->
<!ENTITY % head.content "TITLE & BASE?">

<!ELEMENT HEAD O O (%head.content;) +(%head.misc;) -- document head -->
<!ATTLIST HEAD
 %i18n;                    -- lang, dir --
 profile    %URI;       #IMPLIED  -- named dictionary of meta info --
 >
```

profile = *uri*[CT]

This attribute specifies the location of one or more meta data profiles, separated by white space. For future extensions, user agents should consider the value to be a list even though this specification only considers the first URI to be significant. Profiles are discussed below in the section on meta data.

Attributes defined elsewhere

The HEAD element contains information about the current document, such as its title, keywords that may be useful to search engines, and other data that is not considered document content. User agents do not generally render elements that appear in the HEAD as content. They may, however, make information in the HEAD available to users through other mechanisms.

The TITLE element

```
<!-- The TITLE element is not considered part of the flow of text.
     It should be displayed, for example as the page header or
     window title. Exactly one title is required per document.
  -->
<!ELEMENT TITLE - - (#PCDATA) -(%head.misc;) -- document title -->
<!ATTLIST TITLE %i18n>
```

Every HTML document **must** have a TITLE element in the HEAD section.

Authors should use the TITLE element to identify the contents of a document. Since users often consult documents out of context, authors should provide context-rich titles. Thus, instead of a title such as "Introduction", which doesn't provide much contextual background, authors should supply a title such as "Introduction to Medieval Bee-Keeping" instead.

For reasons of accessibility, user agents must always make the content of the TITLE element available to users (including TITLE elements that occur in frames). The mechanism for doing so depends on the user agent (e.g., as a caption, spoken).

Titles may contain character entities (for accented characters, special characters, etc.), but may not contain other markup (including comments). Here is a sample document title;

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A study of population dynamics</TITLE>
... other head elements...
</HEAD>
<BODY>
... document body...
</BODY>
</HTML>
```

The title attribute

title = *text* [CS]

This attribute offers advisory information about the element for which it is set.

Unlike the TITLE element, which provides information about an entire document and may only appear once, the title attribute may annotate any number of elements. Please consult an element's definition to verify that it supports this attribute.

Values of the title attribute may be rendered by user agents in a variety of ways. For instance, visual browsers frequently display the title as a "tool tip" (a short message that appears when the pointing device pauses over an object). Audio user agents may speak the title information in a similar context. For example, setting the attribute on a link allows user agents (visual and non-visual) to tell users about the nature of the linked resource:

```
...some text...
Here's a photo of
<A href="http://someplace.com/neatstuff.gif" title="Me scuba diving">
  me scuba diving last summer
</A>
...some more text...
```

The title attribute has an additional role when used with the LINK element to designate an external style sheet. Please consult the section on links and style sheets for details.

The document body

The BODY element

```
<!ELEMENT BODY O O (%block;|SCRIPT)+ +(INS|DEL) -- document body -->
<!ATTLIST BODY
 %attrs;                          -- %coreattrs, %i18n, %events --
 onload        %Script;  #IMPLIED  -- the document has been loaded --
 onunload        %Script;  #IMPLIED  -- the document has been removed --
 >
```

background = *uri*[CT]

Deprecated. The value of this attribute is a URI that designates an image resource. The image generally tiles the background (for visual browsers).

text = *color*[CI]

Deprecated. This attribute sets the foreground color for text (for visual browsers).

link = *color*[CI]

Deprecated. This attribute sets the color of text marking unvisited hypertext links (for visual browsers).

vlink = *color*[CI]

Deprecated. This attribute sets the color of text marking visited hypertext links (for visual browsers).

alink = *color*[CI]

Deprecated. This attribute sets the color of text marking hypertext links when selected by the user (for visual browsers).

The body of a document contains the document's content. The content may be presented by a user agent in a variety of ways. For example, for visual browsers, you can think of the body as a canvas where the content appears: text, images, colors, graphics, etc. For audio user agents, the same content may be spoken. Since style sheets are now the preferred way to specify a document's presentation, the presentational attributes of BODY have been deprecated.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>A study of population dynamics</TITLE>
</HEAD>
<BODY bgcolor="white" text="black"
  link="red" alink="fuchsia" vlink="maroon">
  ... document body...
</BODY>
</HTML>
```

Using style sheets, the same effect could be accomplished as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
 <TITLE>A study of population dynamics</TITLE>
 <STYLE type="text/css">
  BODY { background: white; color: black}

A:link { color: red }
  A:visited { color: maroon }
  A:active { color: fuchsia }
 </STYLE>
</HEAD>
<BODY>
  ... document body...
</BODY>
</HTML>
```

Using external (linked) style sheets gives you the flexibility to change the presentation without revising the source HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

```
<HTML>
<HEAD>
 <TITLE>A study of population dynamics</TITLE>
 <LINK rel="stylesheet" type="text/css" href="smartstyle.css">
</HEAD>
<BODY>
 ... document body...
</BODY>
</HTML>
```

Introduction to style sheets

Style sheets represent a major breakthrough for Web page designers, expanding their ability to improve the appearance of their pages. In the scientific environments in which the Web was conceived, people are more concerned with the content of their documents than the presentation. As people from wider walks of life discovered the Web, the limitations of HTML became a source of continuing frustration and authors were forced to sidestep HTML's stylistic limitations. While the intentions have been good -- to improve the presentation of Web pages -- the techniques for doing so have had unfortunate side effects. These techniques work for some of the people, some of the time, but not for all of the people, all of the time. They include:

- Using proprietary HTML extensions
- Converting text into images
- Using images for white space control
- Use of tables for page layout
- Writing a program instead of using HTML

These techniques considerably increase the complexity of Web pages, offer limited flexibility, suffer from interoperability problems, and create hardships for people with disabilities.

Style sheets solve these problems at the same time they supersede the limited range of presentation mechanisms in HTML. Style sheets make it easy to specify the amount of white space between text lines, the amount lines are indented, the colors used for the text and the backgrounds, the font size and style, and a host of other details.

For example, the following short CSS style sheet (stored in the file "special.css"), sets the text color of a paragraph to green and surrounds it with a solid red border:

```
P.special {
color: green;
border: solid red;
}
```

Authors may link this style sheet to their source HTML document with the LINK element:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">

<HTML>
 <HEAD>
  <LINK href="special.css" rel="stylesheet" type="text/css">
 </HEAD>
 <BODY>
  <P class="special">This paragraph should have special green text.
 </BODY>
</HTML>
```

CHAPTER 2:

OVERVIEW OF SQL * PLUS

SQL*Plus is a program of working with an ORACLE database. With it you can (among other things):

- *create* tables in the database
- *store* information in the tables
- *change* information in the tables
- *retrieve* information in a form you choose, performing calculations on it and combining it in new ways
- *maintain* the database itself.

This Guide describes a command language called SQL*Plus, pronounced "Sequel Plus." ("SQL" stands for "Structured Query Language.") The SQL*Plus program accepts SQL*Plus commands from your keyboard, executes them through the ORACLE RDBMS, and formats the results as you specify.

WHO CAN USE SQL Plus

The SQL*Plus command language is designed to be easy to write and read. For example, to display the name, job, and salary of each employee in the EMP table, you might enter the following command:

```
SELECT  ENAME , JOB ,SAL
FROM  EMP;
```

Similarly, to remove employees in Department 10 from the EMP table, you might enter this command:

```
DELETE  FROM   EMP
WHERE  DEPTNO=10;
```

The SQL*Plus command language is powerful enough to serve the needs of users with some database experience, yet easy enough for new users who are just learning to use the ORACLE RDBMS.

Other Ways of Working with the ORACLE RDBMS

Since one must learn the SQL*Plus command language to use SQL*Plus, some users may prefer to work with the ORACLE RDBMS through Easy*SQL. Like SQL*Plus, Easy*SQL is an additional option that run's with the ORACLE RDBMS. New users may find the system of menus, panels, and boxes in Easy*SQL easier to learn

.

Although Easy*SQL may be "easier" to use than SQL*Plus, it is also more limited. For example, while SQL*Plus allows you to join any reasonable number of tables in a query, Easy*SQL allows you to join only three. Similarly, you can grant access to your tables to other users with SQL*Plus but not with Easy*SQL.

Your choice of SQL*Plus or Easy*SQL will be depend on your need of the extra functions of SQL*Plus, versus your desire for the simplicity of Easy*SQL.

WHAT YOU NEED TO RUN SQL*Phis

Equipment and soft ware:

The computer on whirh you run ORACLE and SQL/'Plus is called your host computer ORACLE and SQL*Plus can run on many different kinds of host computers.

Your host computer must have an operating system to manage the computer's resources and to mediate between the computer hardware and programs such as SQL^Plus. Different computers use different operating systems. For information about your host computer's operating system, see the user's manuals provided with the computer

Before you can begin using SQL "Plus, both ORACLE and SQL*Plus must be installed on your computer

Information:

A few aspects of ORACLES and SQL '"Plus differ from one type of host computer and operating system to another. These topics are discussed m/ another document, the ORACLE Installation and User's Guide, which is[3] published in a separate version for each host computer and operating-system that SQL^Plus supports. You should have a copy of the ORACLE Installation and User's Guide available for reference as you work through' this Guide. You will be referred to appropriate parts of the ORACLE Installation and User's Guide when you need information that is kept I there.
.

You will need a user name that identifies you as an authorized ORACLE i user, and a password that proves you are the legitimate owner of your user name.

If your host computer's operating system is meant to be shared by several people (if it is a multi-user system), you will need a user name and password to gain admittance to the operating system. These,may or may not be the same as the ones you use with ORACLE. If you need to distinguish between them, speak of the "system user name and password" and the "ORACLE user name and password".

.Each table in the database is "owned" by a specific ORACLE user name. In order to use the sample tables, you must haveyour own copies of them associated with your user name.

Getting Resources from the Database Administrator:

If your host computer runs a timesharing system,, your organization should have a person called a database administrator (a DBA for short) who supervises use of the ORACLE RDBMS. The DBA should be able to give you your system and ORACLE user name and password, and prepare the system so that you can perform the exercises in this Guide.

Doing It Yourself:

If your host computer is meant to be used by one person at a time, you may be expected to perform the DBA's functions for yourself. This section t outlines the process of getting ready to work with SQL "'Plus.

Instructions for installing the ORACLE RDBMS and SQL*Plus may be found in the ORACLE Installation and User's Guide for your host computer and operating system.

You may use the ORACLE user name SCOTT and the password TIGER. If you want to define your own user name and password, you may do so at any time; see the discussion of the GRANT riommand.

Since some of the exercises in this Guide change the contents of the sample tables, you must make your own copies of these tables to use when you work through the exercises. You can do this by running a pre-planned procedure file named DEMOBLD. For instructions on how to run DEMOBLD.

just you run:

SQL> @C:\orawin\dbs\demobld.sql;

When You're Done Working with This Guide:

When you finish using this Guide, and so have no more use for the sample tables, you may remove them by running a pre-planned procedure named DEMODROP. DEMODROP is explained in the ORACLE Installation and User's Guide for your specific computer system, along with DEMOBLD.

About the LOGIN.SQL file:

If the ORACLE RDBMS has just been installed on your single-user computer, or if you are a new user of the ORACLE RDBMS on a multiuser system, you may skip this section. If the ORACLE RDBMS has been used before on your single-user computer, or if you have been using the ORACLE RDBMS on your multi-user system, this section contains information that may be important to you.

Just as the ORACLE RDBMS stores data in tables, the operating system stores data in *flies;* and if the operating system is a timesharing system, each file is "owned" by a specific system user name. One such file, named LOGIN.SQL, contains commands that SQL*Plus will execute automatically when you start it. For the exercises in this guide perform correctly, your copy of LOGIN.SQL must be set up in a certain in way.

DEMOBLD creates a copy of LOGIN.SQL that contains the appropriate commands. If you already have a file named LOGIN.SQL, DEMOBLD preserves that file by changing its name to LOGIN.OLD.

If you run an ORACLE apphcation that depends on the contents of your original LOGIN.SQL file, that application may not run correctly with the LOGIN.SQL file created by DEMOBLD. You can run such an application by changing the name of LOGIN.SQL to something else (e.g., LOGIN.NEW), then changing the name of LOGIN.OLD back to

LOGIN.SQL. For instructions on renaming files, see the user's manual for your host computer's operating system.

Remember to change the name of DEMOBLD's LOGIN file back to LOGIN.SQL before you resume working with the exercises in this Guide.

DEMODROP deletes LOGIN.SQL. If a file named LOGIN.OLD exists when DEMODROP is run, DEMODROP changes its name to LOGIN.SQL.

THE COMMAND LINE

The SQL-Plus prompt (SQL>) means that SQL*Plus is ready for you to enter a command.

The blinking underline or rectangular block after the SQL> is your computer's cursor or pointer. The cursor indicates the place where the next input you enter will appear on your screen

The line on which the SQL> prompt appears is called the command line. To tell SQL*Plus what to do, you enter commands to the right of the SQL>. At the end of each line/press [Return]. If you have finished the command, SQL*Plus will process it, then display a new SQL> prompt, indicating that it is ready for another command.

If you press [Return] before finishing a command, SQL*Plus will move the cursor to the next line and prompt you with a line number 2, 3, 4, and so on. Continue entering the command on the new lines.

There are two kinds of commands you can enter on the command line:

- SQL commands, for working with information in the database
- SQL*Plus commands, for formatting results, setting options, and editing and storing SQL commands

The rest of this chapter discusses general rules for using these commands.

ENTERING SQL COMMANDS

The commands you use to work with our database are part of the SQL command language. In Exercise 2-2, you will enter a SQL command to display all the information in the sample table EMP

To Display the EMP Table

1. On the command line, enter the first lineof the command:

```
SQL> SELECT *;
```

2. SQL^Plus will display a '2', its prompt for the second line. Enter the second line of the command:

```
2 FROM EMP;
```

The semicolon means that this is the end of the command. Press [Return].

3. SQL*Plus will process the command and display the results on the screen.

```
SQL> SELECT •
 2 FROM EMP;
```

4. After displaying the results, SQL*Plus will display its prompt SQL >, again

The SELECT command is the most important single SQL command because it is your chief means of requesting a display of information from tables. A SELECT command is often called a query.

SQL Command Syntax:

Just as a spoken language'has syntax rules that govern the way words are assembled into sentences, SQL*Plus has syntax rules that govern how words are assembled into Commands. There are only a few such rules, but you must follow them to make your commands be understood.

Ending a Command:

All SQL commands must end with a semicolon (;). The semicolon is a signal that you want SQL*Plus to process the command. After entering the semicolon, press [Return].

If you mistakenly press [Return] before entering the semicolon, SQL-'Tlus will prompt you for the next line of your command. Enter the semicolon and press [Return] again to end the command.

Continuing a Command Across Multiple Line:

SQL does not care how the words that compose a command are divided into lines. You may enter your command on a single line or on as many lines as you want.

Thus, the query you entered in Exercise

```
SOL> SELECT*

 2 FROM EMP ;
```

may be entered on one line:

```
SQL> SELECT* FROM EMP;
```

or on several lines:

```
    SQL> SELECT
    2*
   3 FROM
    4 EMP;
```

as long as individual words are not split between lines.

In this Guide, you will find most SQL commands divided into clauses, one clause on each line. In Exercise 2-2, for example, the SELECT and FROM clauses were placed on separate lines. Many users find this most convenient. But you may choose whatever line division makes your query most readably to you.

Spelling and spacing:

The words in a SQL command must be spelled correctly. They must also be separated from each other by a space or tab.

You may use *additional* spaces or tabs between-words, if you wish, to make your commands more readable. You will see examples of spacing and indentation throughout this Guide. When you enter the commands in the exercises, you do not have to space them as shown, but you may find them clearer if you do.

When case is significant:

The case (capitals or lowercase) of the words in a command is usually not significant in SQL*Plus. A few special situations where case is significant will be pointed out when we reach them.

For the sake of clarity, all table names, column names, and commands in this Guide appear in capital letters.

The SQL Buffer:

When you enter a SQL command, it is stored in a part of memory called the SQL buffer. It remains there until you enter a new command. This means that if you want to edit or re-run the current command, you may do so without re-entering it. See Chapter 4 for details about editing or rerunning a command.

The semicolon, which you must enter to indicate the end of a SQL command, is not stored in the buffer as part of the command.

Introduction to TCP/IP

Summary: TCP and IP were developed by a Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). It was initially successful because it delivered a few basic services that everyone needs (file transfer, electronic mail, remote logon) across a very large number of client and server systems. Several computers in a small department can use TCP/IP (along with other protocols) on a single LAN.

The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. On the battlefield a communications network will sustain damage, so the DOD designed TCP/IP to be robust and automatically recover from any node or phone line failure. This design allows the construction of very large networks with less central management. However, because of the automatic recovery, network problems can go undiagnosed and uncorrected for long periods of time.

As with all other communications protocol, TCP/IP is composed of layers:

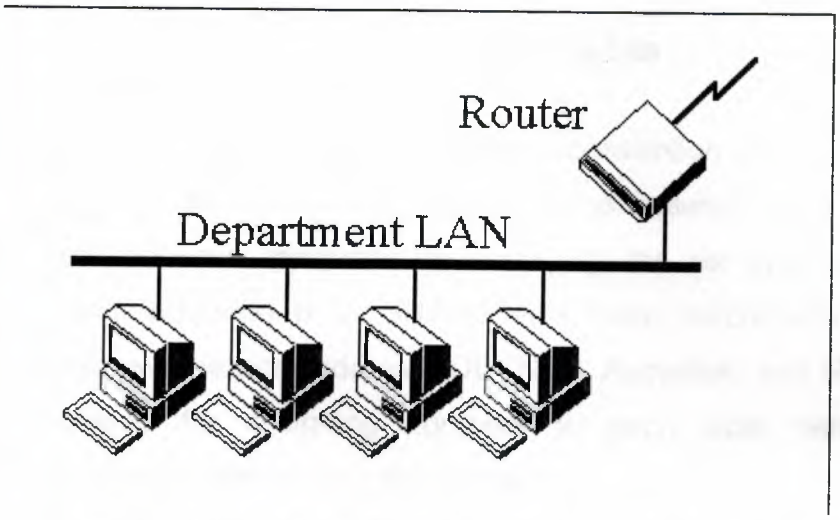
.IP - is responsible for moving packet of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.

.TCP - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.

.Sockets - is a name given to the package of subroutines that provide access to TCP/IP on most systems.

Network of Lowest Bidders

The Army puts out a bid on a computer and DEC wins the bid. The Air Force puts out a bid and IBM wins. The Navy bid is won by Unisys. Then the President decides to invade Grenada and the armed forces discover that their computers cannot talk to each other. The DOD must build a "network" out of systems each of which, by law, was delivered by the lowest bidder on a single contract.



The Internet Protocol was developed to create a Network of Networks (the "Internet"). Individual machines are first connected to a LAN (Ethernet or Token Ring). TCP/IP shares the LAN with other uses (a Novell file server, Windows for Workgroups peer systems). One device provides the TCP/IP connection between the LAN and the rest of the world.

To insure that all types of systems from all vendors can communicate, TCP/IP is absolutely standardized on the LAN. However, larger networks based on long distances and phone lines are more volatile. In the US, many large corporations would wish to reuse large internal networks based on IBM's SNA. In Europe, the national phone companies traditionally standardize on X.25. However, the sudden explosion of high speed microprocessors, fiber optics, and digital phone systems has created a burst of new options: ISDN, frame relay, FDDI, Asynchronous Transfer Mode (ATM). New technologies arise and become obsolete within a few years. With cable TV and phone companies competing to build the National Information Superhighway, no single standard can govern citywide, nationwide, or worldwide communications.

The original design of TCP/IP as a Network of Networks fits nicely within the current technological uncertainty. TCP/IP data can be sent across a LAN, or it can be carried within an internal corporate SNA network, or it can piggyback on the cable TV service. Furthermore, machines connected to any of these networks can communicate to any other network through gateways supplied by the network vendor.

Addresses

Each technology has its own convention for transmitting messages between two machines within the same network. On a LAN, messages are sent between machines by supplying the six byte unique identifier (the "MAC" address). In an SNA network, every machine has Logical Units with their own network address. DECNET, Appletalk, and Novell IPX all have a scheme for assigning numbers to each local network and to each workstation attached to the network.

On top of these local or vendor specific network addresses, TCP/IP assigns a unique number to every workstation in the world. This "IP number" is a four byte value that, by convention, is expressed by converting each byte into a decimal number (0 to 255) and separating the bytes with a period. For example, the PC Lube and Tune server is 130.132.59.234.

An organization begins by sending electronic mail to Hostmaster@INTERNIC.NET requesting assignment of a network number. It is still possible for almost anyone to get assignment of a number for a small "Class C" network in which the first three bytes identify the network and the last byte identifies the individual computer. The author followed this procedure and was assigned the numbers 192.35.91.* for a network of computers at his house. Larger organizations can get a "Class B" network where the first two bytes identify the network and the last two bytes identify each of up to 64 thousand individual workstations. Yale's Class B network is 130.132, so all computers with IP address 130.132.*.* are connected through Yale.

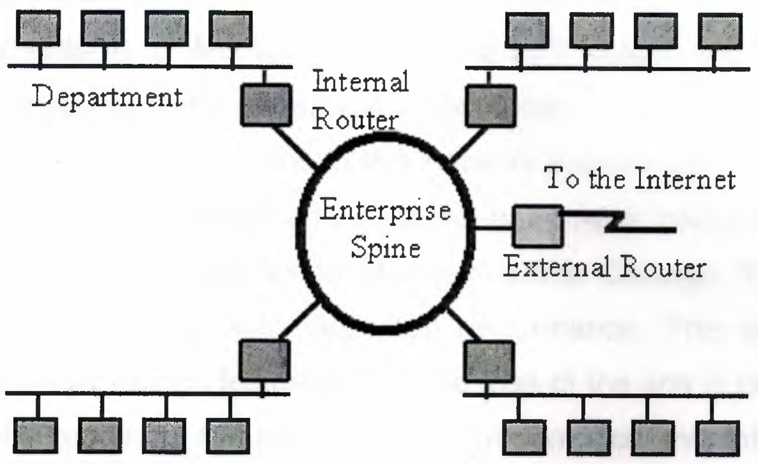
The organization then connects to the Internet through one of a dozen regional or specialized network suppliers. The network vendor is given the subscriber network number and adds it to the routing configuration in its own machines and those of the other major network suppliers.

There is no mathematical formula that translates the numbers 192.35.91 or 130.132 into "Yale University" or "New Haven, CT." The machines that manage large regional networks or the central Internet routers managed by the National Science Foundation can only locate these networks by looking each network number up in a table. There are potentially thousands of Class B networks, and millions of Class C networks, but computer memory costs are low, so the tables are reasonable. Customers that connect to the Internet, even customers as large as IBM, do not need to maintain any information on other networks. They send all external data to the regional carrier to which they subscribe, and the regional carrier maintains the tables and does the appropriate routing.

New Haven is in a border state, split 50-50 between the Yankees and the Red Sox. In this spirit, Yale recently switched its connection from the Middle Atlantic regional network to the New England carrier. When the switch occurred, tables in the other regional areas and in the national spine had to be updated, so that traffic for 130.132 was routed through Boston instead of New Jersey. The large network carriers handle the paperwork and can perform such a switch given sufficient notice. During a conversion period, the university was connected to both networks so that messages could arrive through either path.

Subnets:

Although the individual subscribers do not need to tabulate network numbers or provide explicit routing, it is convenient for most Class B networks to be internally managed as a much smaller and simpler version of the larger network organizations. It is common to subdivide the two bytes available for internal assignment into a one byte department number and a one byte workstation ID.



The enterprise network is built using commercially available TCP/IP router boxes. Each router has small tables with 255 entries to translate the one byte department number into selection of a destination Ethernet connected to one of the routers. Messages to the PC Lube and Tune server (130.132.59.234) are sent through the national and New England regional networks based on the 130.132 part of the number.

Arriving at Yale, the 59 department ID selects an Ethernet connector in the C& IS building. The 234 selects a particular workstation on that LAN. The Yale network must be updated as new Ethernets and departments are added, but it is not effected by changes outside the university or the movement of machines within the department.

A Uncertain Path

Every time a message arrives at an IP router, it makes an individual decision about where to send it next. There is concept of a session with a preselected path for all traffic. Consider a company with facilities in New York, Los Angeles, Chicago and Atlanta. It could build a network from four phone lines forming a loop (NY to Chicago to LA to Atlanta to NY). A message arriving at the NY router could go to LA via either Chicago or Atlanta. The reply could come back the other way.

How does the router make a decision between routes? There is no correct answer. Traffic could be routed by the "clockwise" algorithm (go NY to Atlanta, LA to Chicago). The routers could alternate, sending one message to Atlanta and the next to Chicago. More sophisticated routing measures traffic patterns and sends data through the least busy link.

If one phone line in this network breaks down, traffic can still reach its destination through a roundabout path. After losing the NY to Chicago line, data can be sent NY to Atlanta to LA to Chicago. This provides continued service though with degraded performance. This kind of recovery is the primary design feature of IP. The loss of the line is immediately detected by the routers in NY and Chicago, but somehow this information must be sent to the other nodes. Otherwise, LA could continue to send NY messages through Chicago, where they arrive at a "dead end." Each network adopts some Router Protocol which periodically updates the routing tables throughout the network with information about changes in route status.

If the size of the network grows, then the complexity of the routing updates will increase as will the cost of transmitting them. Building a single network that covers the entire US would be unreasonably complicated. Fortunately, the Internet is designed as a Network of Networks. This means that loops and redundancy are built into each regional carrier. The regional network handles its own problems and reroutes messages internally.

Its Router Protocol updates the tables in its own routers, but no routing updates need to propagate from a regional carrier to the NSF spine or to the other regions (unless, of course, a subscriber switches permanently from one region to another).

Undiagnosed Problems

IBM designs its SNA networks to be centrally managed. If any error occurs, it is reported to the network authorities. By design, any error is a problem that should be corrected or repaired. IP networks, however, were designed to be robust. In battlefield conditions, the loss of a node or line is a normal circumstance. Casualties can be sorted out later on, but the network must stay up. So IP networks are robust. They automatically (and silently) reconfigure themselves when something goes wrong. If there is enough redundancy built into the system, then communication is maintained.

In 1975 when SNA was designed, such redundancy would be prohibitively expensive, or it might have been argued that only the Defense Department could afford it. Today, however, simple routers cost no more than a PC. However, the TCP/IP design that, "Errors are normal and can be largely ignored," produces problems of its own.

Data traffic is frequently organized around "hubs," much like airline traffic. One could imagine an IP router in Atlanta routing messages for smaller cities throughout the Southeast. The problem is that data arrives without a reservation. Airline companies experience the problem around major events, like the Super Bowl. Just before the game, everyone wants to fly into the city. After the game, everyone wants to fly out. Imbalance occurs on the network when something new gets advertised. Adam Curry announced the server at "mtv.com" and his regional carrier was swamped with traffic the next day. The problem is that messages come in from the entire world over high speed lines, but they go out to mtv.com over what was then a slow speed phone line.

Occasionally a snow storm cancels flights and airports fill up with stranded passengers. Many go off to hotels in town. When data arrives at a congested router, there is no place to send the overflow. Excess packets are simply discarded. It becomes the responsibility of the sender to retry the data a few seconds later and to persist until it finally gets through. This recovery is provided by the TCP component of the Internet protocol.

TCP was designed to recover from node or line failures where the network propagates routing table changes to all router nodes. Since the update takes some time, TCP is slow to initiate recovery. The TCP algorithms are not tuned to optimally handle packet loss due to traffic congestion. Instead, the traditional Internet response to traffic problems has been to increase the speed of lines and equipment in order to say ahead of growth in demand.

TCP treats the data as a stream of bytes. It logically assigns a sequence number to each byte. The TCP packet has a header that says, in effect, "This packet starts with byte 379642 and contains 200 bytes of data." The receiver can detect missing or incorrectly sequenced packets. TCP acknowledges data that has been received and retransmits data that has been lost. The TCP design means that error recovery is done end-to-end between the Client and Server machine. There is no formal standard for tracking problems in the middle of the network, though each network has adopted some ad hoc tools.

Need to Know

There are three levels of TCP/IP knowledge. Those who administer a regional or national network must design a system of long distance phone lines, dedicated routing devices, and very large configuration files. They must know the IP numbers and physical locations of thousands of subscriber networks. They must also have a formal network monitor strategy to detect problems and respond quickly.

Each large company or university that subscribes to the Internet must have an intermediate level of network organization and expertise. A half dozen routers might be configured to connect several dozen departmental LANs in several buildings. All traffic outside the organization would typically be routed to a single connection to a regional network provider.

However, the end user can install TCP/IP on a personal computer without any knowledge of either the corporate or regional network. Three pieces of information are required:

1. The IP address assigned to this personal computer
2. The part of the IP address (the subnet mask) that distinguishes other machines on the same LAN (messages can be sent to them directly) from machines in other departments or elsewhere in the world (which are sent to a router machine)
3. The IP address of the router machine that connects this LAN to the rest of the world.

In the case of the PCLT server, the IP address is 130.132.59.234. Since the first three bytes designate this department, a "subnet mask" is defined as 255.255.255.0 (255 is the largest byte value and represents the number with all bits turned on). It is a Yale convention (which we recommend to everyone) that the router for each department have station number 1 within the department network. Thus the PCLT router is 130.132.59.1. Thus the PCLT server is configured with the values:

- My IP address: 130.132.59.234
- Subnet mask: 255.255.255.0
- Default router: 130.132.59.1

The subnet mask tells the server that any other machine with an IP address beginning 130.132.59.* is on the same department LAN, so messages are sent to it directly. Any IP address beginning with a different value is accessed indirectly by sending the message through the router at 130.132.59.1 (which is on the departmental LAN).

Additional information is available in self-study courses from SRA (1-800-SRA1277

CHAPTER 3:

WHY PROGRAM ACCESS ?

As the premiere database management system, Microsoft Access is used by millions all over the world. If it's so good at storing and retrieving information, why would anyone want to go to the trouble of programming an application created by Access? Although Access, with all its wizards and other versatile tools, can produce an amazingly complete and detailed finished application, it cannot provide all necessary capabilities and services to all users without some help.

By design. Access is meant to provide the most universally useful database features, which it does with remarkable thoroughness. However, each user or organization is bound to have special needs and processes that require enhancing the Access database tables, forms, reports, data access pages, and queries. As a bonus, programming Access can create a foolproof user interface and significant error-trapping procedures that can ensure vital database validity. Here are some examples of tasks you can perform with Access:

• Suppose your database is quite large and you need to make the same change to a lot of records at once. With Visual Basic procedures in an Access application, you can run through the entire recordset very efficiently in one operation rather than change the values one by one in a form.

• Suppose you're operating a mail-order book business and need to add state tax to the orders within your own state. You would need to look at the value in the State field of each customer's address. You can create an Access program that can do this for you. It can also add the tax if the customer is in your home state or omit it, if not.

• The validity of your database is very important. You can add procedures that catch errors in data entry and display meaningful messages to the user. If the error is not significant, you can display a reminder to return to and complete the data entry at a later time. You can even add a procedure that doesn't allow the user to move on until the error is corrected.

• Suppose you want to extract certain records, such as those for customers from a particular state, but each time you run the program you want to see the list of customers from a different state. Using Visual Basic code in Access, you can add a prompt to enter the desired value when you run the program.

These are only a few examples of how you can benefit by programming Access.

Access as a Front-End Development Tool

Back in the old days, if a non-computer person wanted a system to manage an important database, he or she had to hire an expensive programmer/consultant. With the advent of tools such as Access and Excel, someone with little programming expertise, but a clear picture of the requirements, can create a sophisticated application in a short amount of time. Access replaces the high-priced programmer who sat between the end user and the computer.

Technical Note: A Short History Lesson

The growth of computer hardware and software has evolved through several generations since the early 1950s, when Honeywelt introduced the first truly electronic computers. The transition from one generation to the next was clearly defined in the early years. For example, the step from vacuum tubes to transistors and then the next step to integrated circuits were quite distinct and dramatic.

Programming technology was no less evolutionary. In the first generation, programmers were forced to write Instructions in machine code consisting solely of numbers. At least they were permitted to use decimal numbers rather than the binary code the computer understands. The second generation opened up the world of assembly languages which were different for each model of computer. Numeric machine code had been replaced by three-tetter codes that were a little easier for humans to keep track of.

In the early 1950s, the third generation of programmino languages was born. Thanks to the late Dr. Grace Hopper, the concept of a language translator made programming possible in plain language. FORTRANT(FORmula TRANslation),COBOL (Commmon business oriented language),BASIC (Beginners all-purpose Symbolic instruction Code), and many others became quite popular in the 1950s and 1960s. The compilers andinterpreters translated the high-level code into binary for machine consumptiori.

The early BASIC, developed at Dartmouth College in the early 1966s for instructional use, was a conversational procedural language.That is, each instruction the student entered was immediately irtterpreted and executed. The object-orianted Visual Basic of today is quite different. Programs written in VB are first compiled intomachine language and thenexecuted as a package.

Thefourth-generation languages, often referred to as *4Gt.s,*brought forth many front -end applications that served as intermediaries between the user and the program generator. Access is an example of a fourth-gerteraticin language. ıt creates code in the background to carry out what you tell it, interactively, to do that's producedis event driven; that is, nothing will happen while the program is ruraiing unless something else happens first—the user clicks a button, presses a key, moves the mouse pointer, or takes some other action.

The fifth generation is a bit fuzzier. It includes expert and knowledge-based systems, artificial intelligence, and language-translation machines. Many expert systems, such as programs for medical diagnosis and oil exploration, were quite successful because their scope was extremely limited.

Much research has been poured into creating computers and programs that can think and reason as well as humans. In 1981, Japan announced Its plan to capture and store all human knowledge by developing the fifth generation of computers. These intelligent supercomputers would be able to learn, reason, and make decisions. They also would be able to converse with humans in natural language and understand pictures. Their natural language capabilities would even accurately translate idiomatic phrases into other languages. Naysayers and skeptics had a good time making jokes about trying to translate English to Russian and back.One such joke had the computer translate the saying "out of sight,out of mind" to Russian and back. When returned to English, it became "an invisible maniac."

It seems unlikely that anyone will ever be able to store all human knowledge into one machine, as the Japanese planned, and expect it to be able to reason using that body of knowledge. But, who knows?

As versatile as Access is, building a well-suited, totally responsive and integrated application for a specific use still requires some customization through additional programming. Access employs three programming languages to enable you to add the fine-tuning to an application. The Structured Query Language (SQL) is the language Access uses behind the scenes in queries that can extract, manipulate, and relate data from one or more tables. Macros consist of lists of actions that execute in response to an event such as a button click or when data in a form changes. Visual Basic code is a highly flexible and comprehensive language you can use to develop complete user-interactive applications. Each of these language components is discussed in detail in later chapters. In addition, other universal languages such as Hypertext Markup Language (HTML) and Visual Basic Scripting can be used with Access applications.

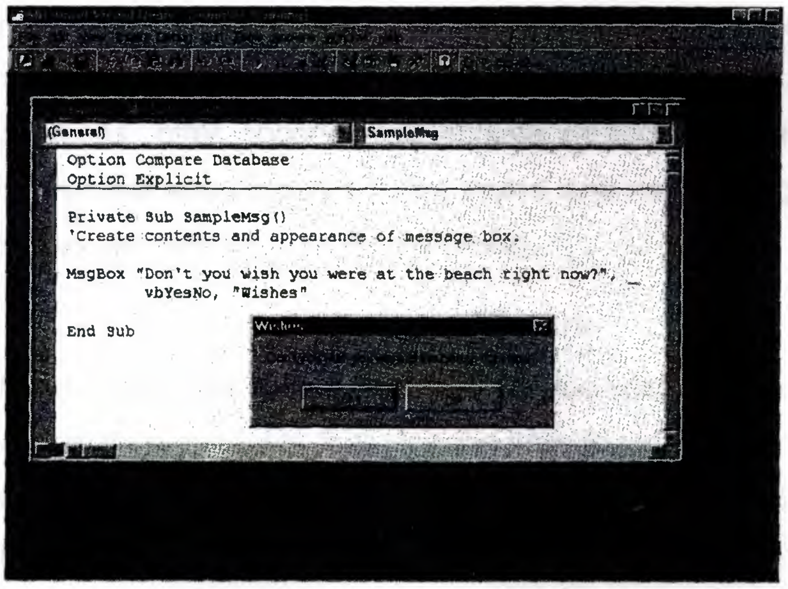
Creating End-User Applications

End user is a term often used to describe someone who has the need to use the computer for database management or other intricate efforts but lacks the time or training to leam all the nuances and complexities of the system. Therefore, a *developer,* the one who does have the time and training, creates a system that is simple on the outside but quite complex on the inside.

An end-user application provides smooth movement through all the computerized activities of the supported organization and responds quickly and appropriately to user actions. The developer also attempts to program responses to all foreseeable error conditions, whether caused by the user, invalid data, or the system itself.

Displaying Information

Message boxes, such as the one shown in Figure 1.1, are an important part of any application. Although the message in the figure is just for fun, message boxes provide relevant and helpful information about current activity or explain the reason for an error condition. In cases in which the user tries to delete a record or a value, a message might ask him or her to confirm the deletion. Message boxes do not require input from the user, only that the box be closed before proceeding.



```
Option Compare Database
Option Explicit

Private Sub SampleMsg()
'Create contents and appearance of message box.

MsgBox "Don't you wish you were at the beach right now?", _
       vbYesNo, "Wishes"

End Sub
```

A message box might contain up to three command buttons with various labels such as OK, Cancel, and Retry. Message boxes can display a choice of icons that indicate the type of message, such as information or warning, and also have a custom caption in the title bar.

Responding to User Actions and Input

Access provides many special tools called *wizards* that ask you for information about what you want to build and create the complete package behind the scenes using the specifications you enter. One of them is the

Access Command Button Wizard, which attaches actions to a command button's OnClick event property.

This results in a procedure that's executed when the user clicks the button. An example is a command button on a form that closes the form and returns to the previous window.

Another important aspect of user interaction is requesting information from the user (for example, a query that extracts records meeting a certain condition, such as customers whose addresses are in California). The procedure displays a dialog box and asks the user to enter the state code, which is then used as the selection criterion in the query.Dialog boxes differ from message boxes in that they require a user response. Dialog boxes ask questions, offer options, and acquire additional information. You're certainly familiar with Access's dialog boxes, and now you'll get a chance to create your own with Access programming.

Trapping Errors

In addition to the data validation rules you specify in the table definition, you can create procedures that alert the user to data errors such as conflicts between two values. For example, the user checks the Paid check box on a customer form and then enters a value in the Amount Due field. The procedure could test the value in the Paid field (Yes or No) and, if Yes, set the GotFocus property of the Amount Due text box to No. This would prevent the user from even reaching the Amount Due field. The procedure also could set the Amount Due to zero when the Paid field is checked.

The user cannot always interpret the cryptic error messages that Access displays. In trying to make the application foolproof, it helps to intercept these error messages and substitute a more helpful message. Access assigns a code number to each type of trappable error, which your procedure can decode to select a more appropriate custom message. For example, user errors on data entry such as entering data in one field that is not compatible with data entered in another field can cause form errors. Forms have an OnError event property to which you can attach the substitute message procedure.

Frustrating system and runtime errors are less predictable, but just as disabling for the user. With programming, you can create error-handling procedures to help solve the problem that caused the error, or at least give the user some information about why the error occurred and how to respond to it.

Peter's Principle: Building Error Traps

You and your users will be a lot happier if you get in the habit of building error traps throughout your application. As you create the application, make mental notes about all the places in the system where the user or the system could possibly make an error. Then add code that executes in the event of an error, even if it's just to stop the procedure and display an error message to the puzzled user. You'll notice when you examine wizard-created procedures that they abound in error-contingency plans.

Returning Results of Calculations and Comparisons

With the Access Expression Builder, you can specify some rather complex expressions as the source of the data for a text box via the text box's Control Source property. Using a function is a shortcut to creating the expression. Many commonly-used functions are included with Access, such as financial functions that compute interest rates or payment amounts. Other functions return mathematical values such as average, standard deviation, or trigonometric value. You can create your own functions using macros or Visual Basic code that will return any type of value you need. After you've built the function, you can refer to it by name from other forms and reports in the database.

Most functions also will accept arguments in place of actual values, so they can be used to calculate the result based on a value specified by the user. For example, suppose you've built a function to compute the annual income from a bond after the user has entered the annual rate. The annual rate is the argument of the function. When the user enters a rate, the function calculates the resulting income.

Following Conditional Branching and Loops

Conditional branching and looping procedures are a very important part of programming an application. Figure 1.2 illustrates three popular branching techniques. Conditional branching refers to determining the next course of action based on the outcome of a comparison. For example, if the Balance Due field contains a positive value greater than zero, go to the procedure that prints invoices; if not, go on to the next step in the regular path.

Another case of conditional branching is accomplished with the Select Case statement. When you have a list of several alternatives, the Select Case statement specifies which action to take with each of the different values. For example, salesmen have a graduated scale of commission depending on their sales volume. Each sales volume interval has its own Case statement to compute the amount of commission. When computing the amount of commission a salesman has earned, the procedure reads the total sales and jumps to the matching Case statement.

Looping procedures enable you to perform a series of operations on an entire recordset. For example, to update the current inventory in a retail store, you process the transaction against the master list, one record at a time. Items sold are subtracted from the number in stock, and items received are added. After the loop is begun, it continues until the end of the recordset is reached, updating where necessary and skipping the records with no matching transactions.

Sharing Data with Other Applications and the Web

The boundaries between Microsoft Office 2000 applications have dimmed almost to the point of being invisible. For example. Access can easily import charts and graphs from Excel, and Word can use an Access database as a data source for creating form letters with mail merge.

New features enable you to send a database via email as well as to create hyperiinks that you can click to jump to another location. The destination might be in another file in your system or as far away as the World Wide Web.

"Linking with Other Office Applications," describes exchanging data with other Microsoft applications. Chapter 18, "Working in a Multiuser

Environment," discusses sharing and controlling databases in a workgroup settings. Chapter 20, "Posting Your Database on the Web," gives more information about HTML and publishing dynamic Access data on the Web.

Understanding the Access Programming Languages

Access speaks three programming languages fluently: Structured Query Language (SQL), macros, and Visual Basic (VB). SQL is the language Access uses behind the scenes in query design. Macros are lists of actions that are to be followed when the user clicks a button or some other specific event occurs.

Visual Basic is a much more comprehensive and complex language that can be used to do almost anything while the application is running. It consists of objects, collections, events, methods, procedures, statements, and properties.

"Writing Visual Basic Procedures," presents more details about Visual Basic program structure and syntax. Appendix A, "What's New in Programming Access 2000?," highlights the new programming and application development features and capabilities that appear in Access 2000.

Structured Query Language (SQL)

The queries you create in the Access Design View grid are implemented in SQL code. While you're building a query, you can look at the SQL code any time by switching to SQL view. To switch the view, select it from the View menu or click the View toolbar button and then choose SQL View from the pull-down menu.

The query concerns the decision to reorder certain products. When the In Stock amount falls below the reorder level plus 1, the product is placed on order. The query is based on two related tables: the list of products in the Products table and the supplier information in the Supplier table. The tables are linked by the Supplier ID field.

The complete SQL statement is as follows:

```
SELECT Products.[Product ID], Products.Description,
Suppliers.[Supplier Name], Products.[Cat Code], Products.[In Stock],
 Products.[Re-order], Products.Supplier, Products.[Unit Cost],
Suppliers.[Contact Name], Suppliers.City, Suppliers.State,
Suppliers.PostalCode, Suppliers.Address, Suppliers.[Supplier ID],
Suppliers.PhoneNumber
FROM Suppliers INNER JOIN Products ON [Suppliers].
[Supplier ID]=[Products].[Supplier]
WHERE ((([Products].[Cat Code]) Not Like "1*") And
(([Products].[In Stock])<[Re-order]+1))
ORDER BY [Products].[Supplier], [Products].[Unit Cost] DESC;
```

In the first section of the SQL code, the SELECT operator lists which fields are to be included in the query result. The fields are named using both the table and field names with the dot operator between them. In the expression SELECT Products. [Product ID], the operator tells Access to look in the Products table for the Product ID field. Field names that contain a space or a dash must be enclosed in square brackets.

"Reviewing Access Database Elements," for more information about the naming conventions used in Access.

The FROM and INNER JOIN operators specify the relationship between the two tables:

Suppliers is the parent table, which has a one-to-many relationship with the Products table. The linking fields are specified by the ON operator: The Supplier ID field in the Suppliers table is linked to the Supplier field in the Products table. In other words, you can look up supplier information for a product by finding a record in the Suppliers table that has the same supplier code as the record in the Products table.

The WHERE operator lists the selection criteria specified in the criteria row of the Design View grid. This shows that the answer table should include records for all the products that need to be reordered (except fish products).

The final operator, ORDER BY, specifies the sort order: in ascending order by supplier and, secondarily, in descending order of unit cost. "Programming with SQL," takes a closer look at the SQL language, its uses, and its components.

Macro Coding

Unlike macros found in other applications, Access macros are not merely recordings of keystrokes. An Access macro consists of a list of actions that are carried out, step-by-step, in response to an event. For example, a macro can run when the user presses a command button, when a form closes, or when a text box control in a form gets focus. A macro also can execute when a specific condition occurs. Such a conditional macro might display a message box when data entered into a field has a certain value.

Macros are created in a special macro grid in the Macro Builder window. To begin a new macro, use one of the following methods:

• Choose New in the Macros page of the Database window.

• Click Build (...) next to one of the properties on the property sheet and then choose Macro Builder in the Choose Builder dialog box.

If you start a macro from the Macro Builder, you're asked to name the macro before proceeding. The macro grid displays at least two columns with two optional columns available, as needed (see Figure 1.5). The default columns are Action and Comment. To add the Macro Name and Condition columns, choose Macro Names or Conditions from the View menu.

To enter the action that's to take place when the macro runs, choose from the Action pull-down menu. ApplyFilter, Beep, CancelEvent, GoToPage, Maximize, and ShowAllRecords are just a few of the more than

50 macro actions in the list. Many of the actions have additional arguments that can be specified in a pane that opens below the grid when an action is selected.

The active window (the message box in the upper-right comer) is displayed when the word *Dog is* entered in the Type field of the MacroEx form shown in the upper-left window and Tab is pressed to move on. The macro condition tells Access to carry out the specified action if the value "Dog" is entered in the Type field. The large bottom window shows the macro code in the macro design window. The macro is named DogMacro, and the action is the command MsgBox, which displays a message box. The action arguments below the grid pane specify the text of the massege box and whether to beep when the message appears.

The last two lines in the Action Arguments pane. Type and Title, specify what kind of icon you want displayed in the message box and enable you to add a custom title, respectively. You can choose to display no icon or the Critical, Warning?, Warning!, or Information icon. The default for Type is None, and the default for Title is Microsoft Access.

After you've built the macro, you attach it to the event property of the button, text box, or other control. To attach a macro, set the event property the macro is meant to respond to, such as Onupdate, OnClick, or OnGotFocus, to the macro. The names of all the macros in the database appear in the control's property pull-down menu. Conditional macros can be attached to a text box in a form and respond to one or more specific values.

Macros are useful for straightforward responses to events, but they do have some limitations. For example, you cannot use a macro to set up error handling or to loop through a recordset to process transactions. A macro does not return a value; therefore, it cannot be used to retrieve user input or to return a calculated value or the result of a comparison.

Visual Basic (VB)

Everything you can do with macros you can do with Visual Basic (and a whole lot more). VB code takes the form of a *procedure,* which is a block of code that performs a specific operation or calculates and returns a value. There are two kinds of procedures:

subprocedures and functions.

Subprocedures carry out one or more operations but do not return any values, whereas *functions* not only carry out the operations but also return values. Access provides many event procedure examples in the VB Help topics that you can copy and paste into a control event property. Then you can make changes in the code to fit your application, such as changing variable names. You can also use the Access Code Builder to create your own custom procedures that will perform any actions you want.

The following is a pair of simple subprocedures that move focus between the two pages of a form when you click the command buttons named Pagel and Page2:

```
 Private Sub Page1_Click()
    Me.GoToPagel
End Sub
Private Sub  Page2_Click()
    Me.GoToPage End Sub
```

The GoToPage method moves the focus to the specified page in the current form. Access provides many built-in functions for all types of operations: Math, Text, Error Handling, Program Flow, Database, and several other categories. You can view the list of built-in functions in the Expression Builder dialog box ,Functions return values to the program: character strings, numeric values, and true/false values. For example, the function IsNull() is often used to find out whether a field has a value. IsNull() returns True if the field is empty or False if it has a value.

custom functions can be created in the Expression Builder dialog box or in the Code Builder window. The following custom function displays the initials extracted from the first and last names in a form:

```
 Private Function Initials() AsString
 ' Display initials.
Initials = Left(FirstName, 1) & Left(LastName, 1)
      End Function
```

The Initials function uses the built-in Left () function to extract the first character of each name. The & symbol concatenates the two letters to form the returned value, Initials. After building and naming the function, you can use it as the control source property instead of the whole expression by entering =Initials() in the property sheet.

Macros can be converted into VB functions very easily. Select the macro in the database window and choose ToolslMacro. Then choose Convert Macros to Visual Basic.

Notice the underline "continuation character" that can be used when a long VB statement does not fit on the screen. Access ignores the character and treats the multiple lines as a single statement.

This has been a very brief introduction to VB procedures. The remaining chapters go into much more detail about how to construct and use subprocedures and functions.

Deciding Language to Use

Each of Access's programming languages has special capabilities as well as some common functionality. For example, a query constructed in the Design View grid is coded in SQL but could just as well be programmed as an object in VB. As shown earlier, a macro easily can be converted to a VB procedure.

Queries are the primary means of viewing, changing, and analyzing data from one or more tables in a database. The three major types of queries are select, action, and SQL. *Select queries* retrieve information, calculate totals, and build crosstabs but do not change the data in the tables. *Action queries* affect data in tables. Both select and action queries can be used as control sources in form, report, and data access page designs. There are four types of action queries:

• The make-table query creates a new table from data already existing in one or more tables.

• The delete query deletes entire records from one or more tables based on the selection criteria.

• The append query adds complete records or only specific fields to one or more tables.

• The update query changes data in existing tables based on information in the Design View grid.

SQL queries accomplish more complex tasks and are coded using SQL statements rather than created in the query Design View grid. These include the following types:

• The *union query* combines corresponding fields from one or more tables or

queries into a single field. For example, you have retail pet supply stores in several shopping centers around town and want to combine the sales data from all of them into one table.

• The *pass-through query* works directly with tables in an ODBC (Open Database Connectivity) database by dealing with the server rather than linking to the tables from Access.

• The data-definition query makes changes to a table definition, such as creating and deleting tables, adding fields, and creating indexes.

• The *subquery* builds a SELECT statement within an existing select or action query. The subquery selects a subset of the records already extracted by the main query.

If you want to perform any of these tasks for working with tables, you must use the SQL language. The first three are built into the SQL view of the query window. *A subquery is built by entering the SQL SELECT statement in* the Criteria row of the Design View grid.

The choice between macros and Visual Basic code depends on what you want to do. Macros can perform simple tasks such as previewing a report and hiding a toolbar. Also, macros are easy to build. In the Macro Builder window, you're coached in creating the syntax and the argument definition. Some tasks can be done only with a macro (for example, performing some startup action when the database first opens or assigning an action to a keystroke or a key combination).

Programming in VB has many advantages, and for some tasks, you must use VB instead of a macro. Here are some examples:

• One advantage is that VB procedures are contained within the form or report definition, whereas macros are separate objects, listed in the Macros tab of the database window. If you move or copy a form or report to another database, the VB procedures automatically move with it, but not the macros. They must be moved or copied separately.

• If none of the built-in functions do exactly what you want, you can build custom function procedures with VB. Custom functions can also take the place of complicated expressions wherever they're used. Once created, custom functions are available from the Expression Builder.

• Macros process an entire set of records at once, giving you no opportunity to control individual transactions. By using VB procedures, you can step through the records and process them one at a time, varying the action depending on the values encountered.

- The arguments set for a macro cannot be changed while the macro is running. While a VB procedure is running, you can pass arguments to it or specify variables as the arguments.
- As mentioned earlier, a VB procedure can detect an error, intercept the error message, and replace it with a more meaningful message to the user.
- VB is extremely flexible when creating and manipulating database object definitions. You can change properties as well as add and delete controls.

In summary, the choice between building queries using the Design View grid and coding in SQL depends on the type of query you want to build. The line between the two techniques is clear. On the other hand, it's not so clear whether to use a macro or VB to carry out actions. Macros are simple to create and easy to use, but there are many advantages to using VB almost exclusively.

Access 2000 has migrated from the Visual Basic for Applications (VBA) language used in Access 97 to the more standard Visual Basic. While modules created with Access 97 are successfully converted to VB, many of the objects, properties, and methods have been replaced by new language elements. For purposes of backward compatibility, most of these replaced elements have been hidden rather than removed. By default, they do not appear in the Object Browser lists, but you can view them by setting one of the Object Browser options. Appendix B, "Converting from Earlier Versions of Access," contains more information about conversion considerations and potential problems.

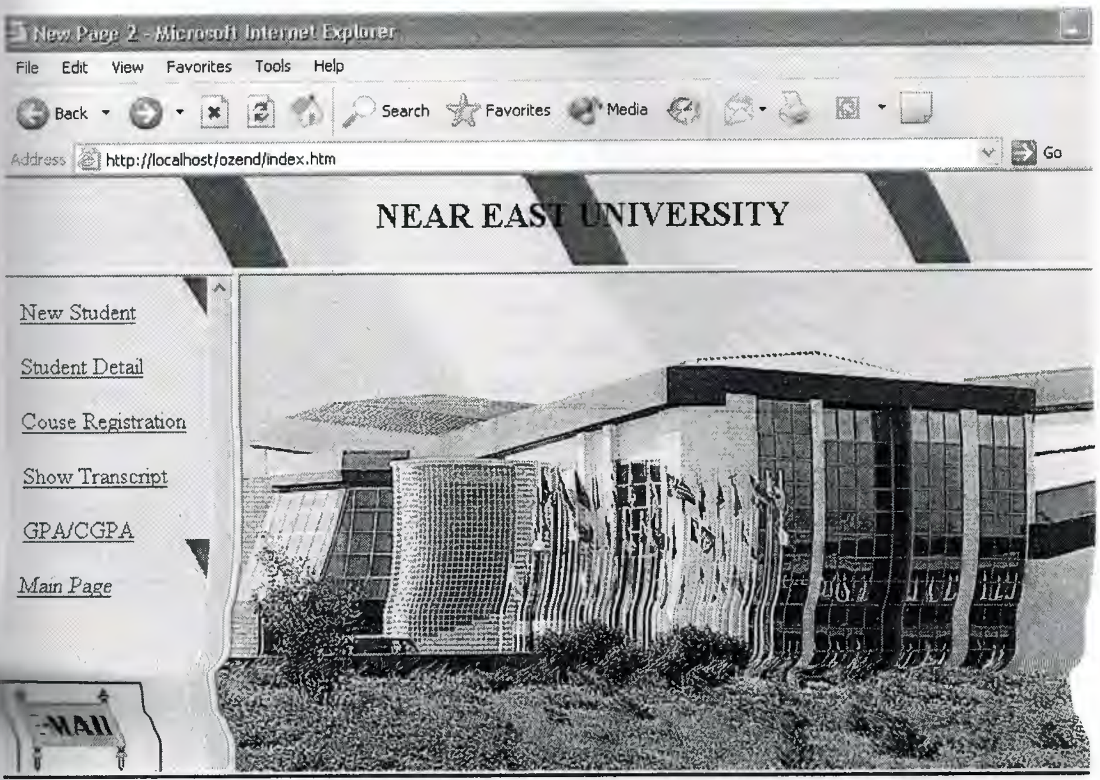
How Do the Wizards Fit In?

The talented wizards provided by Access give you an additional step up toward creating custom databases. Access, itself, is an application front-end development environment, and with the added expertise of the wizards, database development becomes quick and easy.

The wizards all create VB code in the background that you can edit and augment in order to complete or add fine-tuning to the database. Chapter 4, "Creating an Application with a Wizard," shows how to create a new database beginning with one of the Database Wizard's templates. "Examining and Modifying the Wizard's Code," examines the code *generated by the* wizard and makes some changes that modify the objects in the database.

CHAPTER 4:

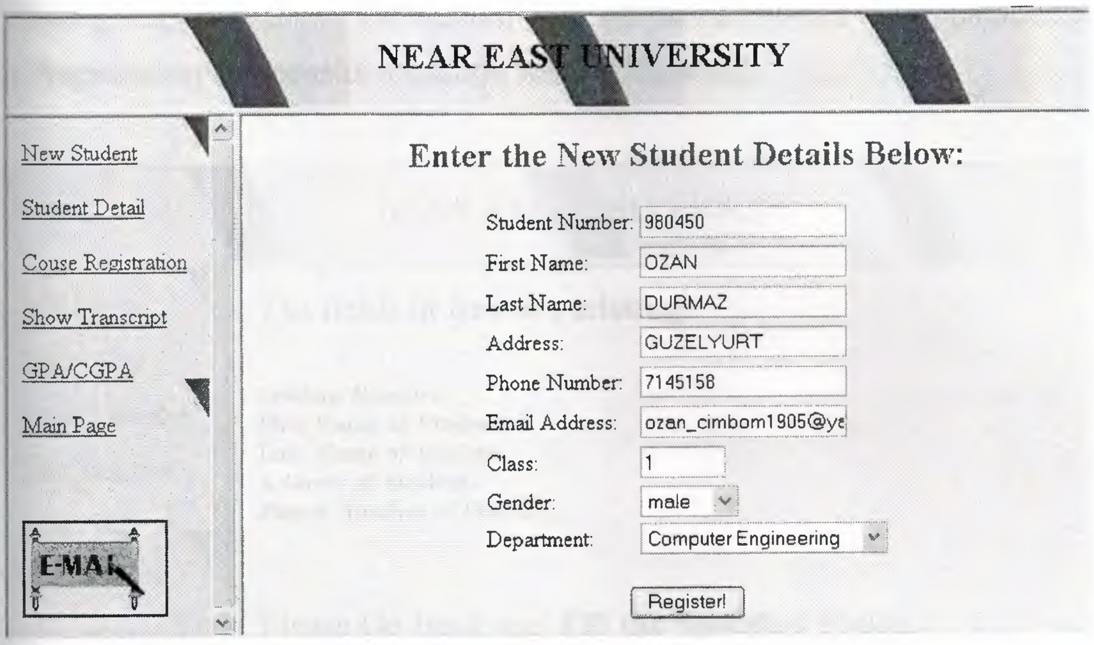
STEP 1: MAIN MENU


New Page 2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ozend/index.htm
Go
NEAR EAST UNIVERSITY
New Student
Student Detail
Couse Registration
Show Transcript
GPA/CGPA
Main Page
E-MAIL

Main Menu

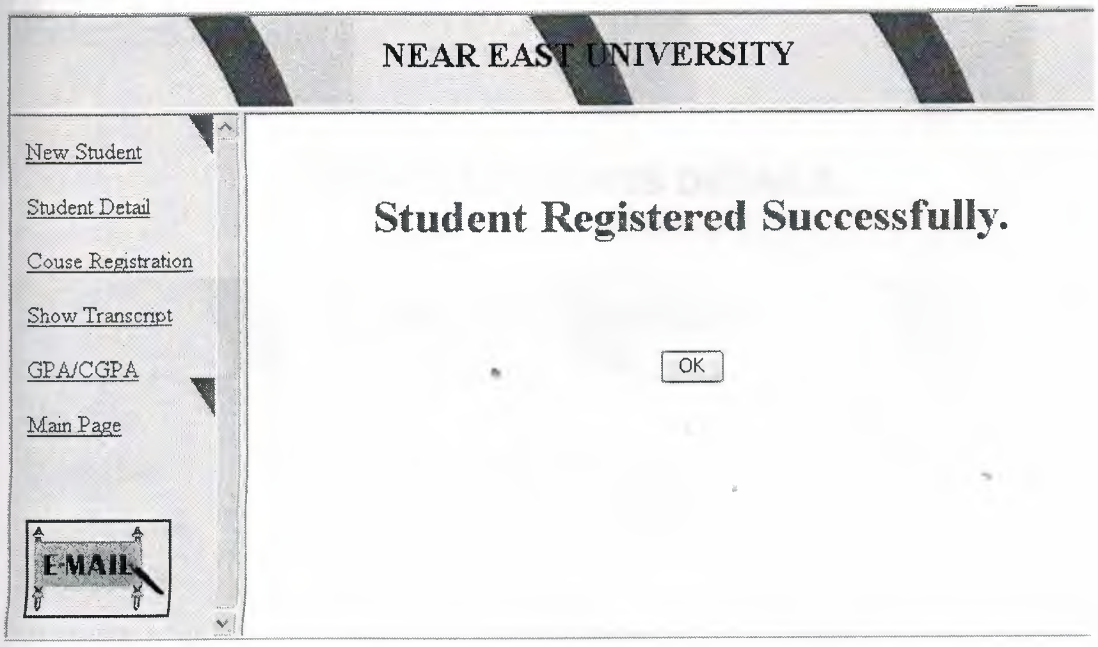
This is the index page of our website. It consist of three frames. In the upper frame, university name is being displayed, the left frame contains all the links to other pages. The right frame is our main frame where all the pages will open after clicking a link from left pane. The left pane has the following links to other pages of the website.

STEP 2:NEW STUDENTS.



Enter the New Student Details Below:

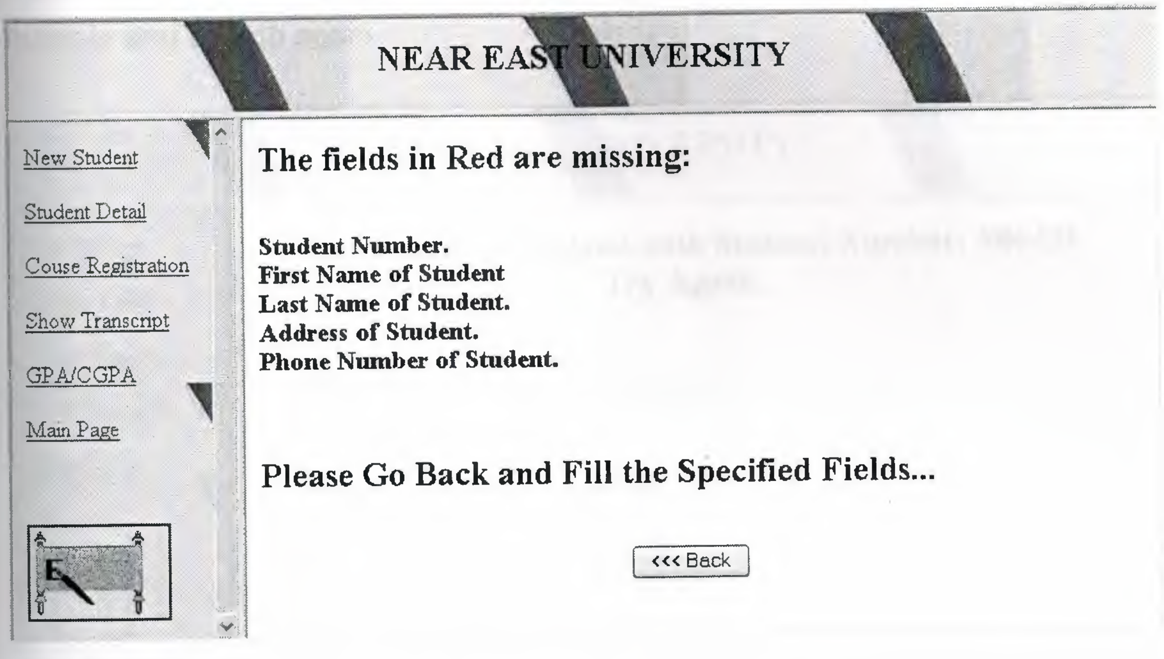
Student Number:	980450
First Name:	OZAN
Last Name:	DURMAZ
Address:	GUZELYURT
Phone Number:	7145158
Email Address:	ozan_cimbom1905@ye
Class:	1
Gender:	male
Department:	Computer Engineering



Student Registered Successfully.

New Student Link

By clicking this link, a page will be opened in the main frame with blank fields to register a new student. The fields will accept student number, first name, last name, address, phone number, email address, class, Gender and department values. By clicking the Register button, the fields will be checked if they have been filled or not. If any information is missing, a page with error message and missing fields will be displayed. If all the information have been added then by clicking Register button, the student data will be transferred to the database and a Registration Successful message will be displayed.

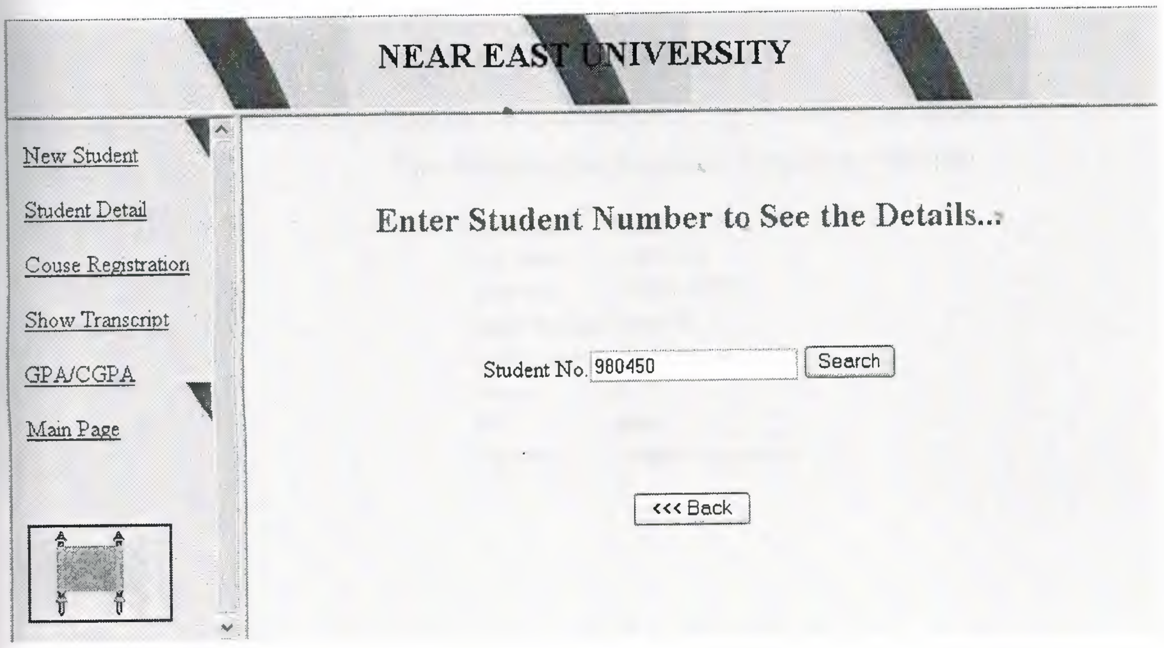


The fields in Red are missing:

Student Number.
First Name of Student
Last Name of Student.
Address of Student.
Phone Number of Student.

Please Go Back and Fill the Specified Fields...

STEP 3: STUDENTS DETAILS.

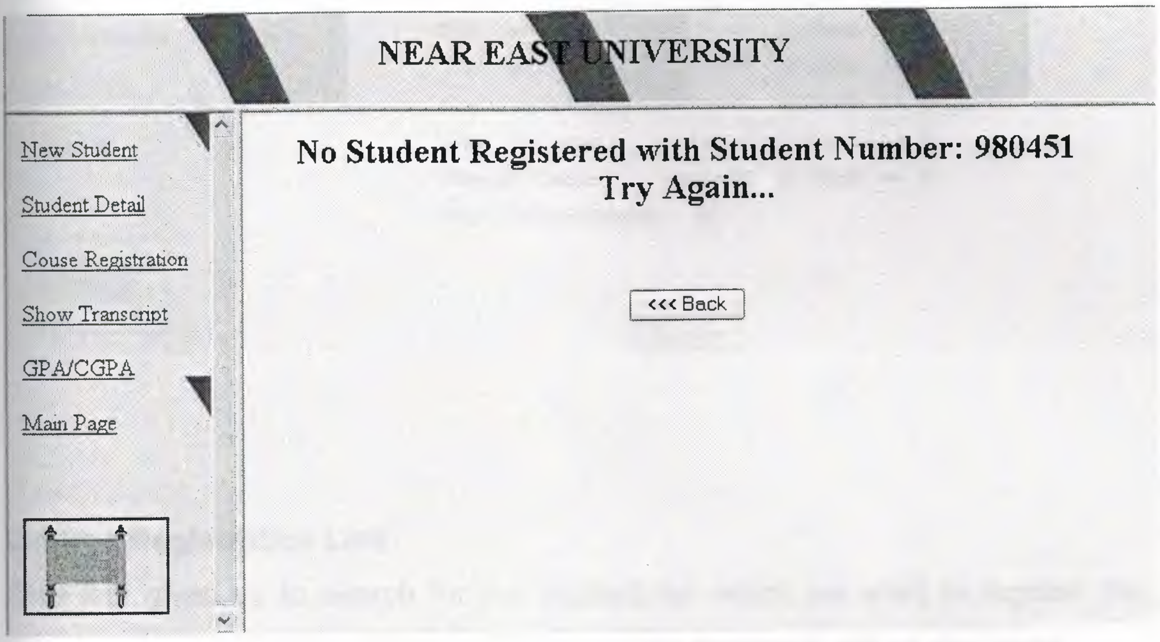


Enter Student Number to See the Details...

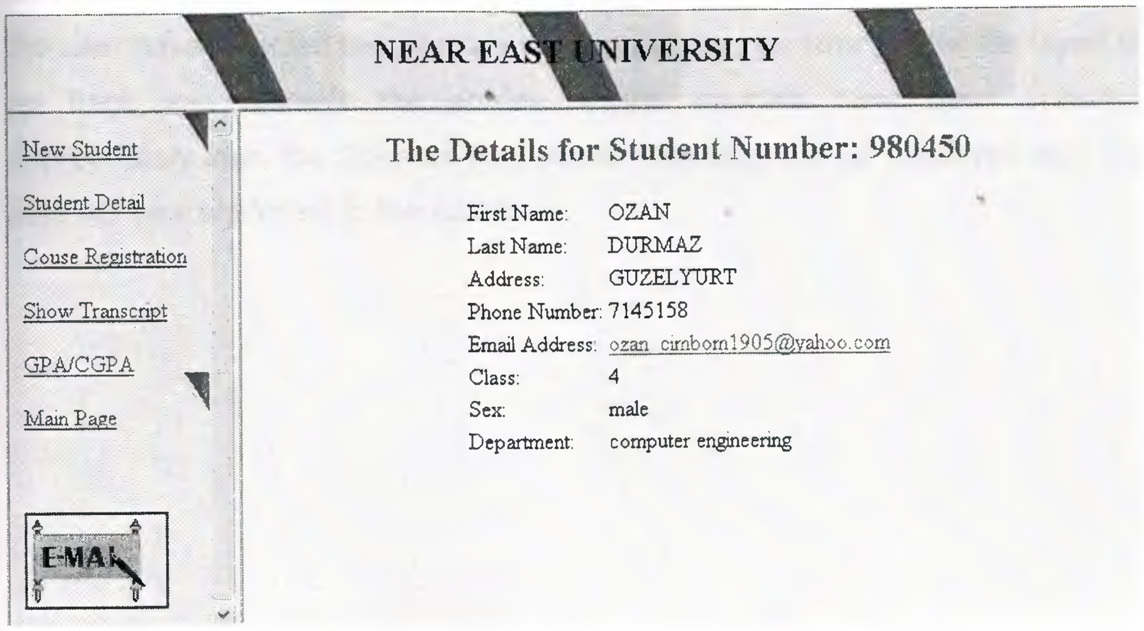
Student No. 980450

Student Detail Link

By clicking this link, a page with a blank field to accept student number will be displayed. The user will enter the student number. Click Search will search for the student data with specified student number. If the data is found then it will be displayed in a suitable manner and if related data not found then an error page will be displayed saying that there is no student registered with specified student number. We will have the option to go back and re-enter the student number and search again.



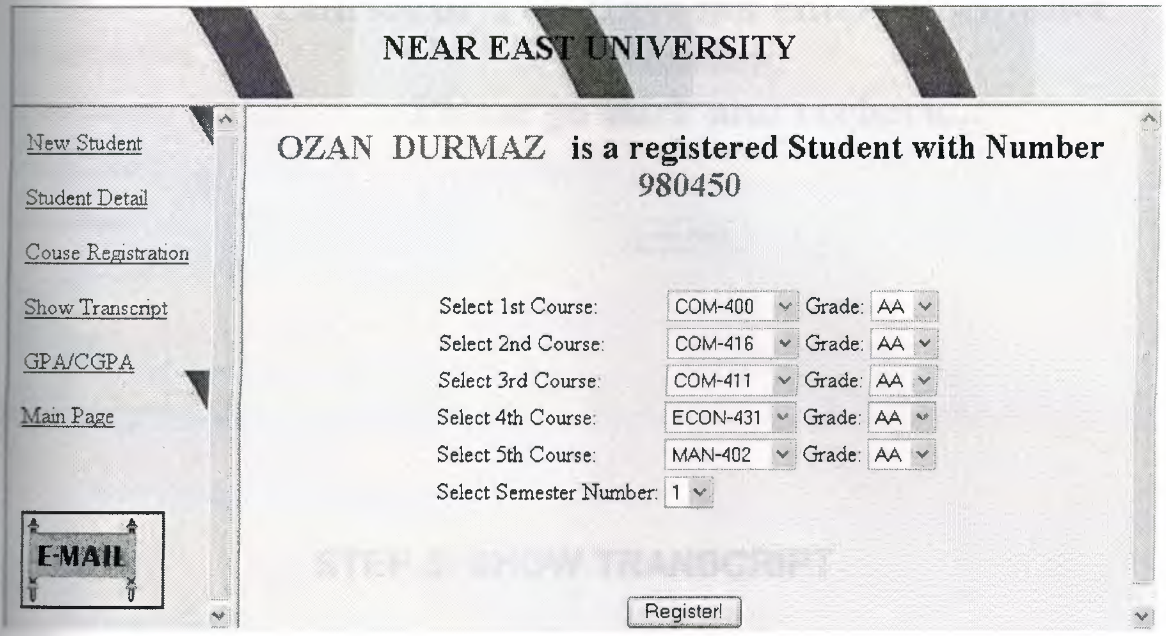
No Student Registered with Student Number: 980451
Try Again...



The Details for Student Number: 980450

First Name:	OZAN
Last Name:	DURMAZ
Address:	GUZELYURT
Phone Number:	7145158
Email Address:	ozan_cimbom1905@yahoo.com
Class:	4
Sex:	male
Department:	computer engineering

STEP 4:COURSE REGISTRATION



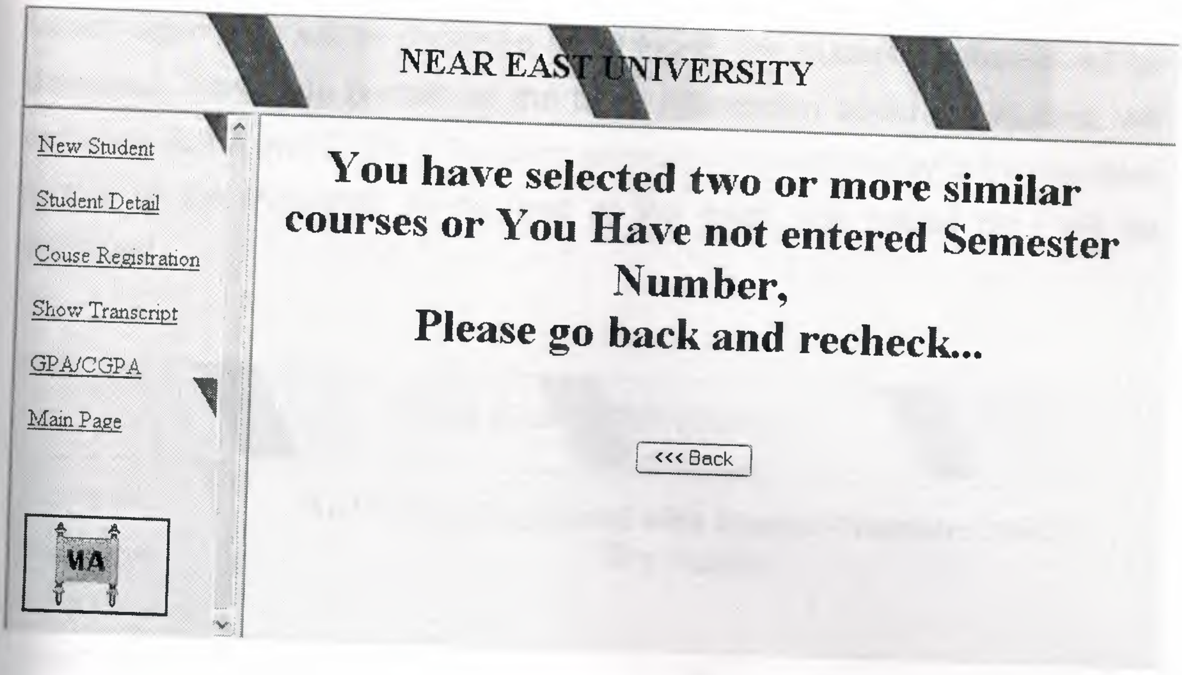
OZAN DURMAZ is a registered Student with Number 980450

Select 1st Course:	COM-400	Grade:	AA
Select 2nd Course:	COM-416	Grade:	AA
Select 3rd Course:	COM-411	Grade:	AA
Select 4th Course:	ECON-431	Grade:	AA
Select 5th Course:	MAN-402	Grade:	AA
Select Semester Number:	1		

Course Registration Link

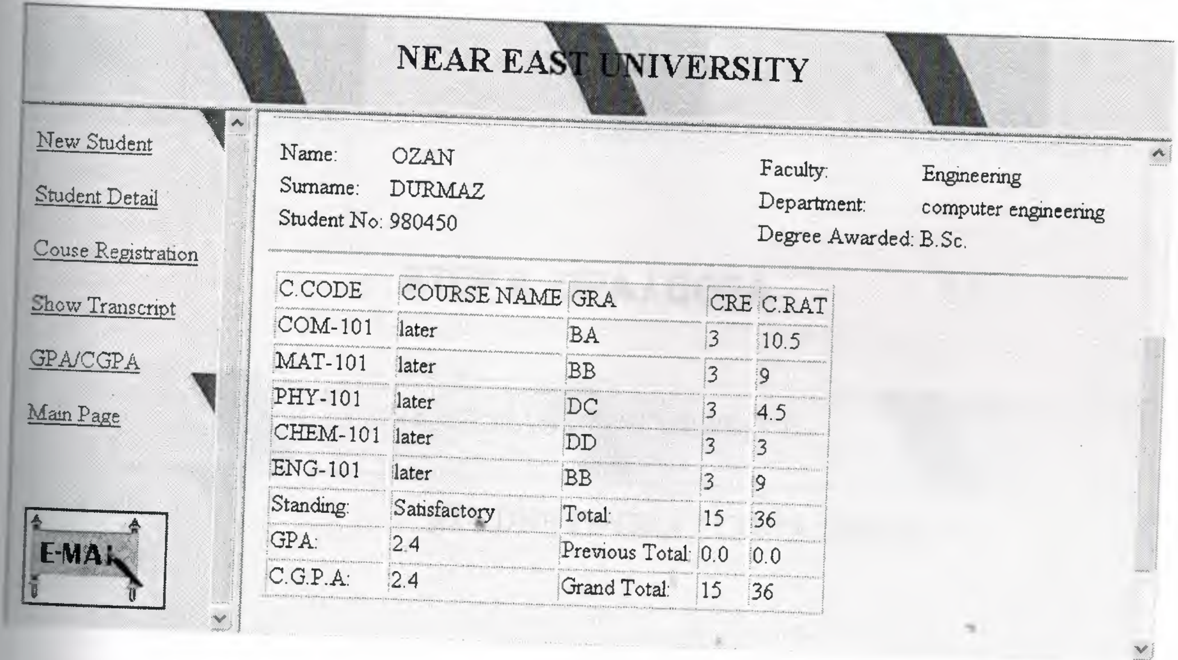
This link gives us to search for the student for which we want to register the courses. If the student is not found, appropriate message will be displayed.

If the student found, another page will appear with the available courses. We can select the courses and their grades from the combo boxes and click OK. If the user have selected two or more similar courses, an error will be displayed to go back and recheck the entries. If the courses have been selected appropriately then the Courses Registered message will be displayed and the data will be transferred to the database.



You have selected two or more similar courses or You Have not entered Semester Number,
Please go back and recheck...

STEP 5: SHOW TRANSCRIPT



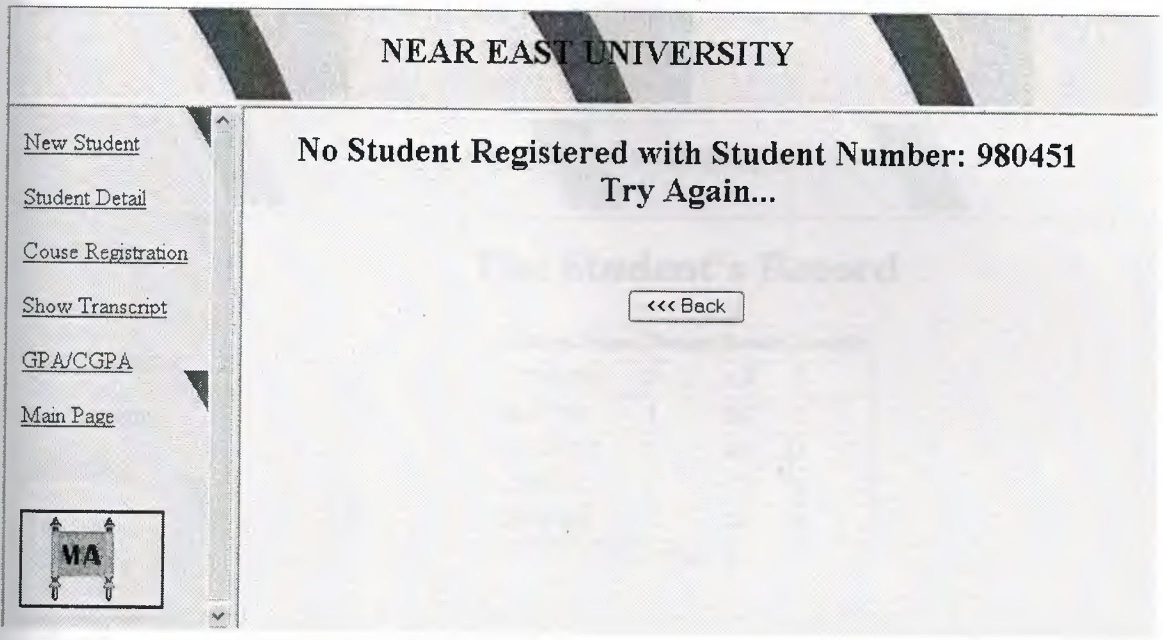
Name: OZAN
Surname: DURMAZ
Student No: 980450

Faculty: Engineering
Department: computer engineering
Degree Awarded: B.Sc.

C.CODE	COURSE NAME	GRA	CRE	C.RAT
COM-101	later	BA	3	10.5
MAT-101	later	BB	3	9
PHY-101	later	DC	3	4.5
CHEM-101	later	DD	3	3
ENG-101	later	BB	3	9
Standing:	Satisfactory	Total:	15	36
GPA:	2.4	Previous Total:	0.0	0.0
C.G.P.A:	2.4	Grand Total:	15	36

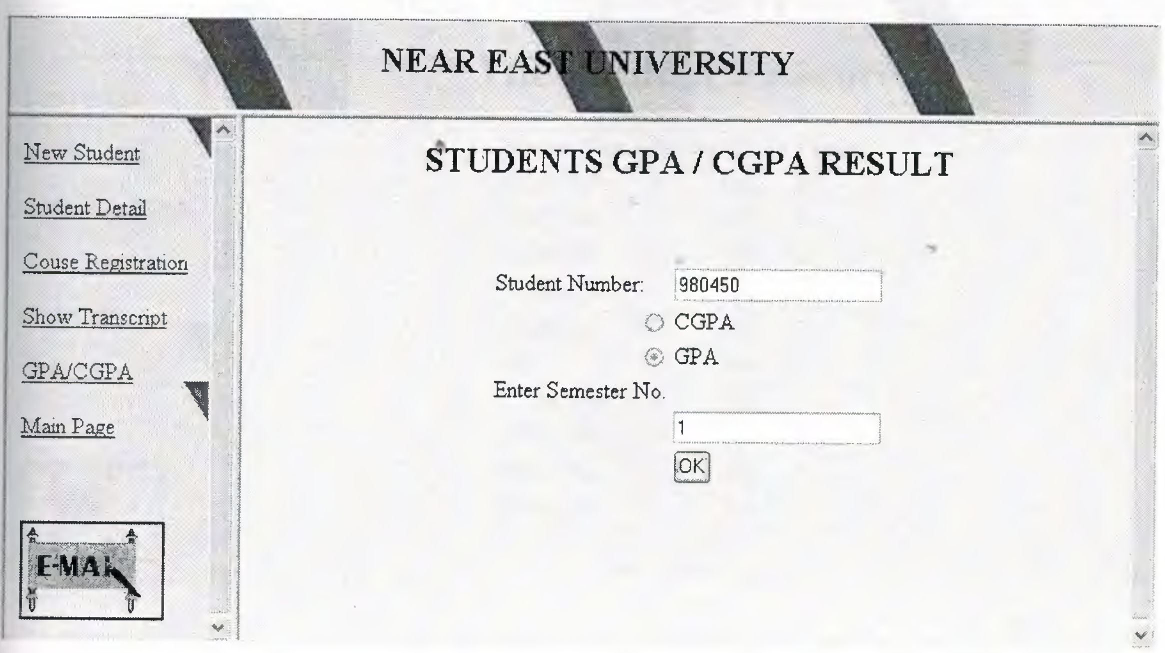
Show Transcript Link

To see the transcript, again we have to search with the student number. If not found an error page will be displayed saying that there is no student registered with this student number. If the student is registered then the program will check if the specified student has registered courses or not. If no registered course found, again error will be displayed and if found, the student's transcript will be displayed. This page contain all the basic information about the student, will calculate G.P.A and C.G.P.A for each semester and will display in the standard format of the transcript. In the end of the page, the issued date will be displayed.



No Student Registered with Student Number: 980451
Try Again...

STEP 6: GPA / CGPA

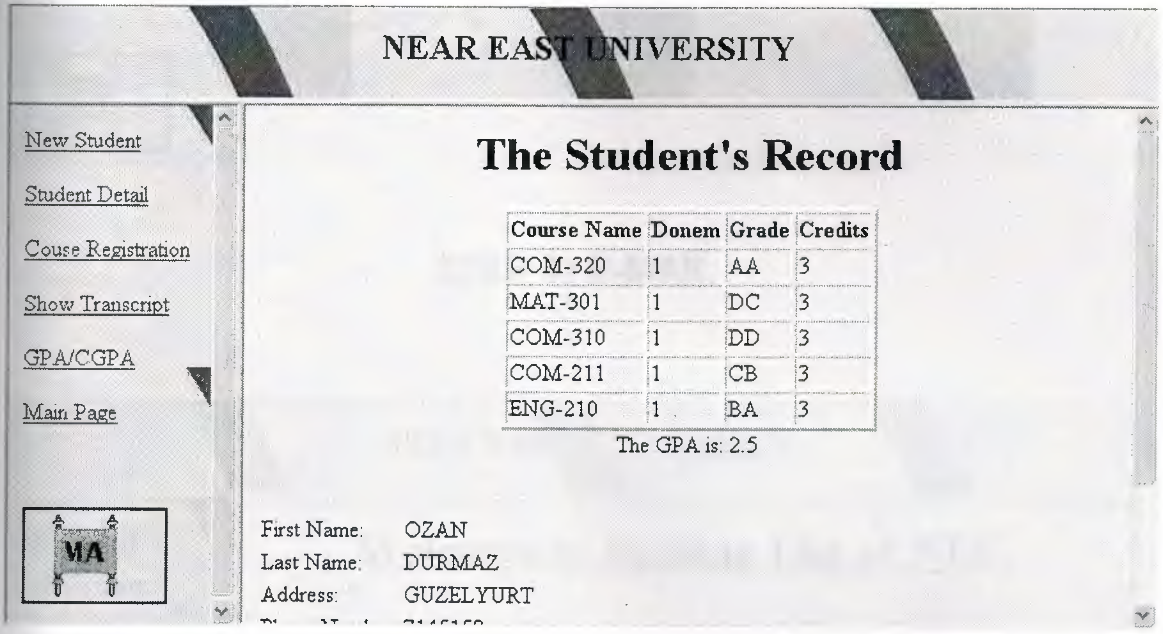


STUDENTS GPA / CGPA RESULT

Student Number: 980450
CGPA [✓ GPA]
Enter Semester No. 1

GPA/CGPA Link

This link will display page with a blank field to accept student number and two option button to select either CGPA or GPA. If GPA is selected the user must enter semester number for which he want to see the transcript. Click OK button to GPA or CGPA as defined earlier. If the semester number is not specified then the CGPA will be displayed by default. If GPA is selected and the semester number is entered then the courses for semester and the GPA will be displayed.

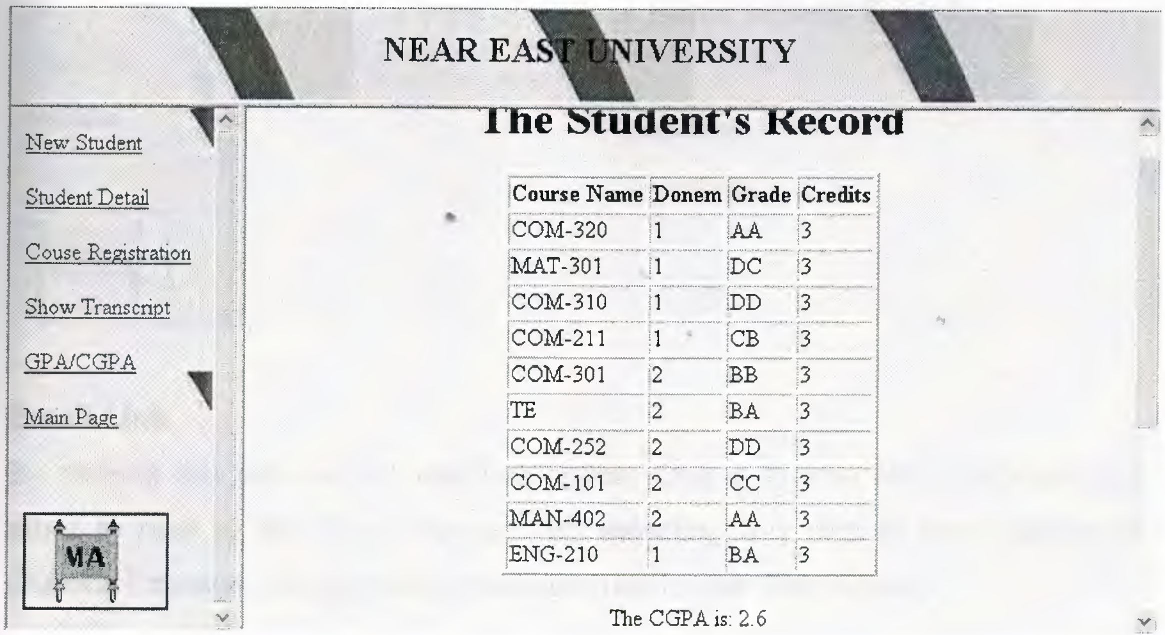


The Student's Record

Course Name	Donem	Grade	Credits
COM-320	1	AA	3
MAT-301	1	DC	3
COM-310	1	DD	3
COM-211	1	CB	3
ENG-210	1	BA	3

The GPA is: 2.5

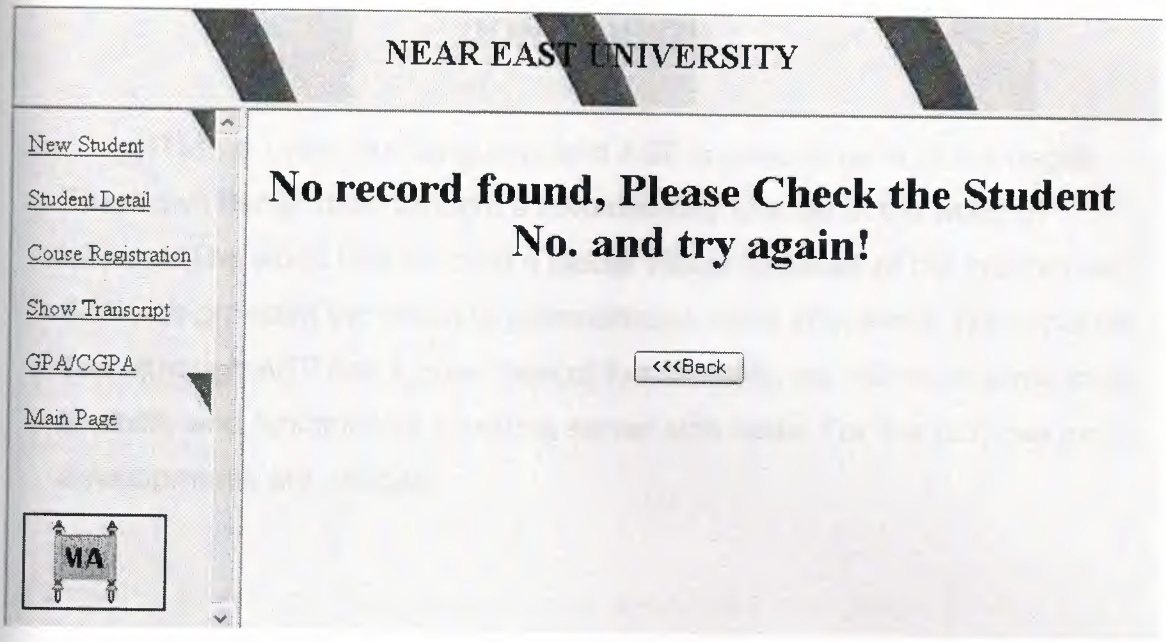
First Name:	OZAN
Last Name:	DURMAZ
Address:	GUZELYURT



The Student's Record

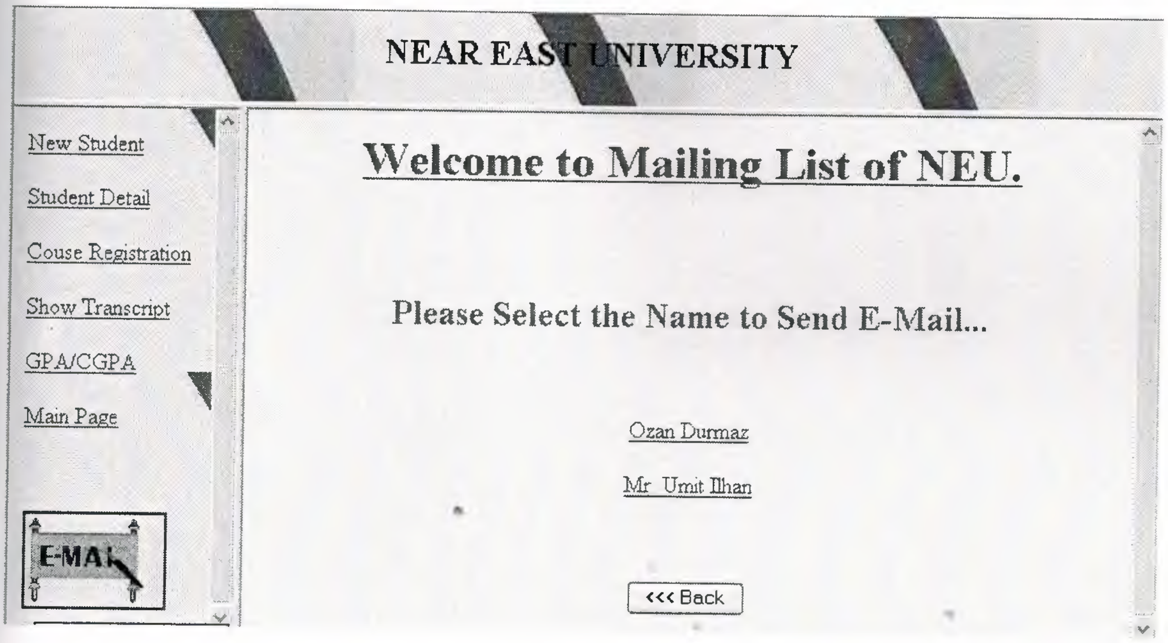
Course Name	Donem	Grade	Credits
COM-320	1	AA	3
MAT-301	1	DC	3
COM-310	1	DD	3
COM-211	1	CB	3
COM-301	2	BB	3
TE	2	BA	3
COM-252	2	DD	3
COM-101	2	CC	3
MAN-402	2	AA	3
ENG-210	1	BA	3

The CGPA is: 2.6


NEAR EAST UNIVERSITY

No record found, Please Check the Student No. and try again!

STEP 7: E-MAIL


NEAR EAST UNIVERSITY

Welcome to Mailing List of NEU.

Please Select the Name to Send E-Mail...

Ozan Durmaz

Mr Umit Ilhan

Email Link

By clicking this link, we will see two option. One is to mail Mr. Umit Ilhan and other to mail to Mr. Ozan Durmaz. By selecting any one of them, Microsoft Outlook Express will open where we can mail to specified name.

CONCLUSION

HTML is hyper text language and ASP is used to build active pages. *These two things have brought a revolutionary change in the world of* internet. The world has become a *global village* because of the internet and ASP has provided the mean to communicate more effectively. We conclude that although ASP has a great deal of functionality, we still need some more flexibility and functionality in writing server side code. For this purpose more developments are needed.

REFERENCE

(1) Lecture Notes of COM 340 given by Mr Umit ILHAN,2003

(2) www.farukcubukcu.com ASP program and exerces programs.

(3) www.msdn.microsoft.com/library,Microsofts official library site,2002

(4) ASP.NET,Web Developers Guide,Syngress,USA,2002

(5) Microsoft ASP Developers site,Retrieved 2002 from World Wide Web:http://www.microsoft.com/ASP/.

(6) www.asptoday.com,forum pages, Retrieved 2002 from web.

(7) aspnedir.com, web pages, Retrieved 2002 from world web.

(8) aspindir.com, web pages, Retrieved 2002 from world web.

APENDIX A:

CODING:

STEP 1:MEAN MENU

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 2</title>
<base target="contents">
</head>
<body background="images/renk1.jpg">
<p align=center>
<h2;&nbsp; NEAR EAST UNIVERSITY</h2>
</P>
&nbsp;</body></html>

<html>
<head>
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 2</title>
<base target="main">
<script language="JavaScript" fptype="dynamicanimation">
<!--
function dynAnimation() {}
function clickSwapImg() {}
//-->
</script>
```

```
<script language="JavaScript1.2" fptype="dynamicanimation"
src="file:///D:/Program%20Files/Microsoft%20Office/Office10/fpclass/animate.js
>
</script>
</head>
<body background="images/renk1.jpg" onload="dynAnimation()">
<p>
<a title="Click here to Register New Student..." target="main"
href="NewStudent.htm" dynamicanimation="fpAnimformatRolloverFP1"
fprolloverstyle="color: #008000" onmouseover="rollIn(this)"
onmouseout="rollOut(this)" language="Javascript1.2">
New Student</a></p>
<p>
<a title="Click here to Check the Student Details..." target="main"
href="CheckDetail.htm" dynamicanimation="fpAnimformatRolloverFP1"
fprolloverstyle="color: #008000" onmouseover="rollIn(this)"
onmouseout="rollOut(this)" language="Javascript1.2">
Student Detail</a></p>
<p>
<a title="Click here to Register Courses..." target="main" href="Department.htm"
dynamicanimation="fpAnimformatRolloverFP1" fprolloverstyle="color: #008000"
onmouseover="rollIn(this)" onmouseout="rollOut(this)"
language="Javascript1.2">
Couse Registration</a></p>
<p>
<a title="Click here to see the Transcript..." target="main"
href="CheckTranscript.htm" dynamicanimation="fpAnimformatRolloverFP1"
fprolloverstyle="color: #008000" onmouseover="rollIn(this)"
onmouseout="rollOut(this)" language="Javascript1.2">
Show Transcript</a>
</p>
```

```
<p>
<a target="main" title="Click here to Check GPA/CGPA..." href="st_gpa.htm"
dynamicanimation="fpAnimformatRolloverFP1" fprolloverstyle="color: #008000"
onmouseover="rollIn(this)" onmouseout="rollOut(this)"
language="Javascript1.2">
GPA/CGPA</a></p>
<p>
<a target="main" title="Click here to Assign grades..." href="CheckAssign.htm"
dynamicanimation="fpAnimformatRolloverFP1" fprolloverstyle="color: #008000"
onmouseover="rollIn(this)" onmouseout="rollOut(this)"
language="Javascript1.2">
Assign Grades</a></p>
<br><br>
<p>
<a target="main" title="Click here to Send E-mail..." href="SendMail.htm">
<img src="images\em.gif" width="96" height="64"></a></p>
</body>
</html>

<html>
<head>
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 2</title>
<base target="_self">
</head>
<body bgproperties="fixed" background="images/AKM2.jpg">
</body>
</html>
```

STEP 2:NEW STUDENTS //HTML

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
<TITLE></TITLE>
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<h2><font color=green>Enter the New Student Details Below:</font></h2>
<form method="post" action="NewStudent.asp">

<table border=0>
<tr>
        <td>Student Number:</td>
        <td><input type=text name="StNo" size="20"></td>
</tr>
<tr>
        <td>First Name:</td>
        <td><input type=text name="FirstName" size="20"></td>
</tr>
<tr>
        <td>Last Name:</td>
        <td><input type=text name="LastName" size="20"></td>
</tr>
<tr>
        <td>Address:</td>
        <td><input type=text name="Address" size="20"></td>
</tr>
<tr>
        <td>Phone Number:</td>
        <td><input type=text name="PhoneNumber" size="20"></td>
</tr>
```

```
<tr>
        <td>Email Address:</td>
        <td><input type=text name="EmailAddress" size="20"></td>
</tr>
<tr>
        <td>Class:</td>
        <td><input type=text value=1 size=6 name="ClassID"></td>
</tr>
<tr>
        <td>Gender:</td>
        <td>
                <select name="Sex">
                <option value="male">male
                <option value="female">female
        </td>
</tr>
<tr>
        <td>Department:</td>
        <td>
                <select name="Department">
                <option value="Computer">Computer Engineering
                <option value="Electrical">Electrical Engineering
                <option value="Mechanical">Mechanical Engineering
                <option value="Civil">Civil Engineering
        </td>
</tr>
</table>
<br>
<input type=submit value="Register!">
</form>
</center>
</BODY>
</HTML>
```

STEP 2.1 NEW STUDENTS//ASP

```
<%@ Language=VBScript %>
<%StNo=Request.Form ("StNo")
FirstName=Request.Form ("FirstName")
LastName=Request.Form ("LastName")
Address=Request.Form ("Address")
ClassID=Request.Form ("ClassID")
Department=Request.Form ("Department")
Sex=Request.form("Sex")
PhoneNumber=Request.Form ("PhoneNumber")
Email=Request.Form ("Email")
%>
<%flag=true%>
<%if StNo="" or FirstName="" or LastName="" or Address="" or
PhoneNumber="" then
flag=false
end if%>
<%if flag=false then%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
</HEAD>
<BODY bgcolor="#CCFFFF">
<h2>The fields in <font color=red>Red</font> are missing:</h2><br>
<font color=red><h3>
<%if StNo="" then%>Student Number.<%end if%><br>
<%if FirstName="" then%>First Name of Student<%end if%><br>
<%if LastName="" then%>Last Name of Student.<%end if%><br>
<%if Address="" then%>Address of Student.<%end if%><br>
<%if PhoneNumber="" then%>Phone Number of Student.<%end if%><br>
</h3></font>
<br><br>
```

```
<h2>Please Go Back and Fill the Specified Fields...</h2>
<br>
<center>
<input type=button value="<<< Back" onClick="{history.back()}">
</center>
</BODY>
</HTML>

<%end if%>
<%if flag=true then%>
<HTML>
<HEAD>
<%set db=server.createobject("adodb.connection")
db.Open "provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\PROJE-
400\StudentList.MDB;Persist Security Info=False"
sql="insert into Students (StNo, FirstName, LastName,
Address,PhoneNumber,Email, ClassID, Sex, Department)values ('"&StNo&"',
'"&FirstName&"','"&LastName&"','"&Address&"','"&PhoneNumber&"','"&Email&"',
'"&ClassID&"','"&Sex&"','"&Department&"')"
db.execute sql
%>
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<br><br>
<font color=green>
<h1>Student Registered Successfully.</h1>
</font>
<br><br><br>
<form action=NewStudent.htm>
<input type=submit value="  OK  ">
</form>
```

```
</BODY>
</HTML>
<%end if%>
```

STEP 3: STUDENTS DETAILS//HTML

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
<TITLE></TITLE>
</HEAD>
<BODY bgcolor="#CCFFFF">
<form method="post" action="StDetail.asp">
<center>
<font color=blue><h2>&nbsp;</h2><h2>Enter Student Number to See the
Details...</h2></font>
&nbsp;<p><br>
Student No.<input type=text name="number" size="20">
<input type=submit value="Search"> </p>
<br><br><br>
<input type=button value="<<< Back" onClick="{history.back()}">
</center>
</form>
</BODY>
</HTML>
```

STEP 3.1: STUDENTS DETAILS//ASP

```
<%@ Language=VBScript %>
<%
number=request.form("number")
set db=server.createobject("adodb.connection")
```

```
db.Open "provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\PROJE-
400\StudentList.MDB;Persist Security Info=False"
sql="select * from Students where StNo = '"&number&"'"
set rs1=db.execute(sql)
%>
<%if rs1.eof and rs1.bof then%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<h2>No Student Registered with Student Number: <%=number%><br>
Try Again...</h2>
<br><br>
<input type=button value="<<< Back" onClick="{history.back()}">
</center>
</BODY>
</HTML>
<%end if%>
<%if not rs1.eof and not rs1.bof then%>

<html>
<head></head>
<body bgcolor="#CCFFFF">
<center>
<font color="blue">
<h2>The Details for Student Number: <%=number%></h2>

<table border=0>
<tr>
        <td>First Name:</td>
        <td><%Response.Write rs1.fields("FirstName")%></td>
</tr>
```

```
 <tr>
          <td>Last Name:</td>
          <td><%Response.Write rs1.fields("LastName")%></td>
 </tr>
 <tr>
          <td>Address:</td>
          <td><%Response.Write rs1.fields("Address")%></td>
 </tr>
 <tr>
          <td>Phone Number:</td>
          <td><%Response.Write rs1.fields("PhoneNumber")%></td>
 </tr>
 <tr>
          <td>Email Address:</td>
          <td><a href="mail to:<%Response.Write
rs1.fields("Email")%>"><%Response.Write rs1.fields("Email")%></td>
</tr>
 <tr>
          <td>Class:</td>
          <td><%Response.Write rs1.fields("ClassID")%></td>
 </tr>
 <tr>
          <td>Sex:</td>
          <td><%Response.Write rs1.fields("Sex")%></td>
 </tr>
 <tr>
          <td>Department:</td>
          <td><%Response.Write rs1.fields("Department")%></td>
</tr>
</table>
</center>
</font>  </body>
</html>
<%end if%>
```

STEP 4: COURSE REGISTRATION // HTML

```
<html>
<head>
<title>"new page.asp"</title>
</head>
<body link="#339933" vlink="#996600" alink="#CC9900" bgcolor="#CCFFFF">
<font face="Arial"><br><br><br>
<FORM action="StRegistration.asp" Method="post">

<CENTER>
<H2><font color=blue>Enter Student Number to Register
Courses...</font></H2>
<br><br>
<P>Student Number:<INPUT Name="stno" size="20" >
<input type="submit" value="Search" name="ok"></P>
<br><br><br>
<input type="button" value="<<< Back" onClick="{history.back()}">

</FORM>
</CENTER>
</html>
```

STEP 4.1: COURSE REGISTRATION//ASP-1

```
<%@ Language=VBScript %>
<%name=request.form("name")
number=request.form("stno")
set db=server.createobject("adodb.connection")
db.Open "provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\PROJE-
400\StudentList.MDB;Persist Security Info=False"

sql="select * from Grade where StNo = '"&number&"' "
sql1="select * from Students where StNo = '"&number&"'"
sql2="select Code from Course"
```

```
set rs=db.execute(sql)
set rs1=db.Execute(sql1)
set rs2=db.Execute (sql2)%>

<%if rs1.eof and rs1.bof then%>
<HTML>
<HEAD>
<title>No Found</title>
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<h1><font color=red>
The Student is not registered , please try again.
</font></h1>
<br><br>
<input type=button onClick="{history.back()}" value="<<< Back">
</center>
</BODY>
</HTML>
<%end if%>
<%if not rs1.eof and not rs1.bof then%>

<html>
<head><title>Student Record</title></head>
<body bgcolor="#CCFFFF">
<h2><center><font color=blue>
<%Response.Write rs1.fields("FirstName")%>&nbsp
<%Response.Write rs1.fields("LastName")%></font>
&nbsp is a registered Student with Number
<font color=blue><%Response.Write rs1.fields("StNo")%>
</font></h2></center>
```

```
<form action="Register.asp" method="post">
<!--for test only-->
<input type=hidden name="number" value=<%=number%>>
<br><br>
<!--combo boxes start here-->
<center>
<table border=0>
<tr>
<td>Select 1st Course:</td>
<td>
<SELECT name="course1">
<%while not rs2.eof%>
<OPTION value="<%=rs2.fields("Code")%>"><%=rs2.fields("Code")%>
<%rs2.movenext
wend%>
</SELECT>
</td>
<td>Grade:</td>
<td><select name="grade1">
        <option value="AA">AA
        <option value="BA">BA
        <option value="BB">BB
        <option value="CB">CB
        <option value="CC">CC
        <option value="DC">DC
        <option value="DD">DD
        <option value="FD">FD
        <option value="FF">FF
        </select></td>
</tr>
<tr>
<td>Select 2nd Course:</td>
<td>
<%rs2.movefirst%>
```

```
 <SELECT name="course2">
 <%while not rs2.eof%>
 <OPTION value="<%=rs2.fields("Code")%>"><%=rs2.fields("Code")%>
 <%rs2.movenext
 wend%>
 </SELECT>
 </td>
 <td>Grade:</td>
 <td><select name="grade2">
         <option value="AA">AA
         <option value="BA">BA
         <option value="BB">BB
         <option value="CB">CB
         <option value="CC">CC
         <option value="DC">DC
         <option value="DD">DD
         <option value="FD">FD
         <option value="FF">FF
         </select></td>
</tr>

<tr>
<td>Select 3rd Course:</td>
<td>
<%rs2.movefirst%>
<SELECT name="course3">
<%while not rs2.eof%>
<OPTION value="<%=rs2.fields("Code")%>"><%=rs2.fields("Code")%>
<%rs2.movenext
wend%>
</SELECT>
</td>
<td>Grade:</td>
```

```
<td><select name="grade3">
        <option value="AA">AA
        <option value="BA">BA
        <option value="BB">BB
        <option value="CB">CB
        <option value="CC">CC
        <option value="DC">DC
        <option value="DD">DD
        <option value="FD">FD
        <option value="FF">FF
        </select></td>
</tr>
<tr>
<td>Select 4th Course:</td>
<td>
<%rs2.movefirst%>
<SELECT name="course4">
<%while not rs2.eof%>
<OPTION value="<%=rs2.fields("Code")%>"><%=rs2.fields("Code")%>
<%rs2.movenext
wend%>
</SELECT>
</td>
<td>Grade:</td>
<td><select name="grade4">
        <option value="AA">AA
        <option value="BA">BA
        <option value="BB">BB
        <option value="CB">CB
        <option value="CC">CC
        <option value="DC">DC
        <option value="DD">DD
        <option value="FD">FD
        <option value="FF">FF
```

```
            </select></td>
    </tr>
    <tr>
    <td>Select 5th Course:</td>
    <td>
   <%rs2.movefirst%>
   <SELECT name="course5">
   <%while not rs2.eof%>
   <OPTION value="<%=rs2.fields("Code")%>"><%=rs2.fields("Code")%>
   <%rs2.movenext
   wend%>
   </SELECT>
   </td>

   <td>Grade:</td>
   <td><select name="grade5">
            <option value="AA">AA
            <option value="BA">BA
            <option value="BB">BB
            <option value="CB">CB
            <option value="CC">CC
            <option value="DC">DC
            <option value="DD">DD
            <option value="FD">FD
            <option value="FF">FF
            </select></td>
  </tr>
  <tr>
  <td>Select Semester Number:</td>
  <td>
  <select name="Semester">
  <option value="1">1
  <option value="2">2
  <option value="3">3
```

```
<option value="4">4
<option value="5">5
<option value="6">6
<option value="7">7
<option value="8">8
</select>
</td>
</tr>
</table>
<BR><br><br>
<input type=submit value="Register!">
</center>
</form>
</body>
</html>
<%end if%>
```

STEP 4.2: COURSE REGISTRATION//ASP-2

```
<%@ Language=VBScript %>
<%course1=Request.Form ("course1")
course2=Request.Form("course2")
course3=Request.Form ("course3")
course4=Request.Form ("course4")
course5=Request.Form ("course5")
semester=Request.Form ("Semester")
number=Request.Form ("number")

grade1=Request.Form ("grade1")
grade2=Request.form("grade2")
grade3=Request.Form ("grade3")
grade4=Request.Form ("grade4")
grade5=Request.Form ("grade5")
%>
```

```
<%flag=false%>
<%if course1=course2 or course1=course3 or course1=course4 or
course1=course5 or course2=course3 or course2=course4 or course2=course5
or course3=course4 or course3=course5 or course4=course5 or semester=""
then
flag=true
end if%>
<%if flag=true then%>

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<h1><font color=red>
You have selected two or more similar courses or You Have not entered
Semester Number, <br>
Please go back and recheck...</font></h1>
<br><br>
<input type=button value="<<< Back" onClick="{history.back()}">
</center>
</BODY>
</HTML>
<%end if%>
<%if flag=false then%>
<html>
<head><title>Registration</title>

<%set db=server.createobject("adodb.connection")
db.Open "provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\PROJE-
400\StudentList.MDB;Persist Security Info=False"
```

```
sql1="insert into Grade (StNo, Course, Donem, Grade,CreditHr)values
('"&number&"', '"&course1&"','"&semester&"','"&grade1&"','3')"
sql2="insert into Grade (StNo, Course, Donem, Grade,CreditHr)values
('"&number&"', '"&course2&"','"&semester&"','"&grade2&"','3')"
sql3="insert into Grade (StNo, Course, Donem, Grade,CreditHr)values
('"&number&"', '"&course3&"','"&semester&"','"&grade3&"','3')"
sql4="insert into Grade (StNo, Course, Donem, Grade,CreditHr)values
('"&number&"', '"&course4&"','"&semester&"','"&grade4&"','3')"
sql5="insert into Grade (StNo, Course, Donem, Grade,CreditHr)values
('"&number&"', '"&course5&"','"&semester&"','"&grade5&"','3')"

db.execute sql1
db.execute sql2
db.execute sql3
db.execute sql4
db.execute sql5%>
</head>
<body bgcolor="#CCFFFF">
<center>
<h3><font color=blue>The following course have been registered...</font></h3>
<br>
<%=course1%>: <%=grade1%><br>
<%=course2%>: <%=grade2%><br>
<%=course3%>: <%=grade3%><br>
<%=course4%>: <%=grade4%><br>
<%=course5%>: <%=grade5%><br>

<form action="Department.htm">
<input type=submit value="  OK  ">
</form>
</center>
</body>
</html>
<%end if%>
```

STEP 5: SHOW TRANSCRIPT//HTML

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
<TITLE></TITLE>
</HEAD>
<BODY bgcolor="#CCFFFF">
<form method="post" action="StTranscript.asp" id=form1 name=form1>
<center>
<font color=blue><h2>Enter Student Number to See the
Transcript...</h2></font>
<br><br>
Student No.<input type=text name="number" size="20">
<input type=submit value="Search" id=submit1 name=submit1>
<br><br><br>
<input type=button value="<<< Back" onClick="{history.back()}">
</center>
</form>
</BODY>
</HTML>
```

STEP 5.1.1: SHOW TRANSCRIPT//ASP

```
<%@ Language=VBScript %>
<%
number=request.form("number")
set db=server.createobject("adodb.connection")
db.Open "provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\PROJE-
400\StudentList.MDB;Persist Security Info=False"
sql="select * from Students where StNo = '"&number&"'"
sql0="select * from Grade where StNo = '"&number&"'"
```

```
set rs=db.execute(sql)
set rs0=db.Execute (sql0)
%>

<%if rs.eof and rs.bof then%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<h2>No Student Registered with Student Number: <%=number%><br>
Try Again...</h2>
<br><br>
<input type=button value="<<< Back" onClick="{history.back()}" id=button1
name=button1>
</center>
</BODY>
</HTML>
<%end if%>
<%if (not rs.eof and not rs.bof) and (rs0.eof and rs0.bof) then%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>
<h2>The Student with Student Number:<%=number%> has no registered
Courses...</h2>
<br><br>
<input type=button value="<<< Back" onClick="{history.back()}" id=button1
name=button1>
</center>
```

```
</BODY>
</HTML>
<%end if%>

<%if not rs0.eof and not rs0.bof then%>
<html>
<head>
<%
sql1="select * from Grade where StNo = '"&number&"' and Donem='1'"
sql2="select * from Grade where StNo = '"&number&"' and Donem='2'"
sql3="select * from Grade where StNo = '"&number&"' and Donem='3'"
sql4="select * from Grade where StNo = '"&number&"' and Donem='4'"
sql5="select * from Grade where StNo = '"&number&"' and Donem='5'"
sql6="select * from Grade where StNo = '"&number&"' and Donem='6'"
sql7="select * from Grade where StNo = '"&number&"' and Donem='7'"
sql8="select * from Grade where StNo = '"&number&"' and Donem='8'"

set rs1=db.execute(sql1)
set rs2=db.execute(sql2)
set rs3=db.execute(sql3)
set rs4=db.execute(sql4)
set rs5=db.execute(sql5)
set rs6=db.execute(sql6)
set rs7=db.execute(sql7)
set rs8=db.execute(sql8)
%>

</head>
<body bgcolor="#CCFFFF">
<center>
<h1>NEAR EAST UNIVERSITY</h1><br>
<h2>Transcript of Academic Record</h2><br><br>
<h3>LEFKOSA-MERSIN 10-TURKEY Tel:(90)(392)2236464
Fax:(90)(392)2236461 <br>E-mail:info@neu.edu.tr www.neu.edu.tr</h3>
```

```
<hr>
</center>
<!--table for name, surname, no etc-->
<table border=0><!--outer table-->
<tr><td>
<table border=0>
<tr>
        <td>Name:</td>
        <td><%Response.Write rs.fields("FirstName")%></td>
</tr>
<tr>
        <td>Surname:</td>
        <td><%Response.Write rs.fields("LastName")%></td>
</tr>
<tr>
        <td>Student No:</td>
        <td><%response.write rs.fields("StNo")%></td>
</tr>
</table>
</td>
<td &nbsp</td>
<td>
<table border=0>
<tr>
        <td>Faculty:</td>
        <td>Engineering</td>
</tr>
<tr>
        <td>Department:</td>
        <td><%Response.Write rs.fields("Department")%>
</tr>
```

```
<tr>
        <td>Degree Awarded:</td>
        <td>B.Sc.</td>
</tr>
</table>
</td></tr></table><!--outer table-->
<hr>
<!--outer table starts here-->
<table>
<tr>
<td>
<!--first table-->
<%if not rs1.eof and not rs1.bof then%>
<table border=1>
<tr>
        <td>C.CODE</td>
        <td>COURSE NAME</td>
        <td>GRA</td>
        <td>CRE</td>
        <td>C.RAT</td>
</tr>
<%while not rs1.eof%>
<tr>
        <td><%Response.Write RS1.Fields ("Course")%>&nbsp;</td>
        <td>later</td>
        <td><%Response.Write rs1.fields("Grade")%>&nbsp;</td>
        <td><%Response.Write rs1.fields("CreditHr")%>&nbsp;</td>
        <td><%
                cr1=RS1.Fields("Grade")
                if cr1="AA" then     credit=4.0 end if
                if cr1="BA" then     credit=3.5 end if
                if cr1="BB" then     credit=3.0 end if
                if cr1="CB" then     credit=2.5 end if
                if cr1="CC" then     credit=2.0 end if
```

```
            if cr1="DC" then    credit=1.5 end if
            if cr1="DD" then    credit=1.0 end if
            if cr1="FD" then    credit=0.5 end if
            if cr1="FF" then    credit=0.0 end if
            CRAT1=credit*rs1.fields("CreditHr")
            CRATTotal1=CRATTotal1+CRAT1
            %>
            <%=CRAT1%> &nbsp;</td>
</tr>
<%RS1.MoveNext
Wend
%>
<tr>
        <td>Standing:</td>
        <td>Satisfactory</td>
        <td>Total:</td>
        <td><%
            rs1.movefirst
            while not rs1.eof
            cr1=RS1.Fields("Grade")
            if cr1="AA" then    credit=4.0 end if
            if cr1="BA" then    credit=3.5 end if
            if cr1="BB" then    credit=3.0 end if
            if cr1="CB" then    credit=2.5 end if
            if cr1="CC" then    credit=2.0 end if
            if cr1="DC" then    credit=1.5 end if
            if cr1="DD" then    credit=1.0 end if
            if cr1="FD" then    credit=0.5 end if
            if cr1="FF" then    credit=0.0 end if

            CreditTotal1=CreditTotal1+RS1.Fields("CreditHr")
            rs1.movenext
            wend%>
            <%=CreditTotal1%> &nbsp;</td>
            <td><%=CRATTotal1%>&nbsp;</td>
    </tr>
    <tr>
            <td>GPA:</td>
            <td>
                    <%GPA1=CRATTotal1/CreditTotal1%>
                    <%=GPA1%> &nbsp;</td>
            <td>Previous Total:</td>
            <td>0.0</td>
            <td>0.0</td>
    </tr>
    <tr>
            <td>C.G.P.A:</td>
            <td>
                    <%CGPA1=GPA1%>
                    <%=CGPA1%> &nbsp;</td>
            <td>Grand Total:</td>
            <td>
                    <%GrandTotalCRE=CreditTotal1+0%>
                    <%=GrandTotalCRE%> &nbsp;</td>
            <td>
                    <%GrandTotalCRAT=CRATTotal1+0%>
                    <%=GrandTotalCRAT%> &nbsp;</td>
    </tr>
    </table>
    <%end if%>
```

STEP 6:GPA/CGPA //HTML

```
<html>
<head>
<title>STUDENTS GPA</title>
</head>
<body bgcolor="#CCFFFF" text="#000000" link="#339933" vlink="#996600"
alink="#CC9900">
```

```
<CENTER>
<H2>STUDENTS GPA / CGPA RESULT</H2>
<p align="center">&nbsp;</p>
<form method="post" action="cgpResult.asp">
<table border=0>
<tr>
        <td>Student Number:</td>
        <td><input name="number" size="20" ></td>
</tr>
<tr>
        <td align=right><input type="radio" name="point" value="CGPA" ></td>
        <td align=left>CGPA</td>
</tr>
<tr>
        <td align=right><input type="radio" name="point" value="GPA"
CHECKED></td>
        <td align=left>GPA</td>
</tr>
<tr>
        <td>Enter Semester No.</td>
</tr>
<tr>
        <td></td>
        <td align=right><input type="text" name="Donem" value=0
size="20"></td>
</tr>
<tr>
        <td></td>
        <td><input type="submit" value="OK"></td>
</tr>
</table>
</form>
</CENTER>
```

```
<br><p align="right">&nbsp;</p></FORM>
<p>&nbsp;</p></body>
</html>
```

STEP 6.1: GPA/CGPA//ASP

```
<%@ Language=VBScript %>
<%name=request.form("name")
number=request.form("number")
point=Request.Form ("point")
        if point="GPA" then
        dd=Request.Form("Donem")
        end if
set db=server.createobject("adodb.connection")
db.Open "provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\PROJE-
400\StudentList.MDB;Persist Security Info=False"
if dd=0 then
sql="select * from Grade where StNo = '"&number&"'"
sql1="select * from Students where StNo = '"&number&"'"
end if
if dd<>0 then
sql="select * from Grade where StNo='"&number&"' and Donem='"&dd&"' "
sql1="select * from Students where StNo='"&number&"'"
end if
set rs=db.execute(sql)
set rs1=db.Execute(sql1)%>
<%if rs.eof and rs.bof then%>
<html>
<head><title>No Record</title></head>
<body bgcolor="#CCFFFF">
        <br>
        <center>
        <h1>No record found, Please Check the Student No. and try again!</h1>
        <br>
        <br>
        <input type=button value="<<<Back" onClick="{history.back()}">
        </center>
    </body>
    </html>


    <%end if%>
    <%if not rs.eof and not rs.bof then%>
    <HTML>
    <HEAD>
    <META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
    </HEAD>
    <BODY bgcolor="#CCFFFF">
    <center>
    <h1>The Student's Record</h1>
    <table border=2 align=center>
    <tr>
    <th>Course Name</th><th>Donem</th><th>Grade</th><th>Credits</th>
    </tr>
    <%while not rs.eof%>
    <tr>
    <td><%Response.Write RS.Fields ("Course")%>&nbsp;</td>
    <td><%Response.Write RS.Fields ("Donem")%>&nbsp;</td>
    <td><%Response.Write Rs.fields ("Grade")%>&nbsp;</td>
    <td><%Response.Write Rs.fields ("CreditHr")%>&nbsp;</td>
    </tr>
    <%RS.MoveNext
    Wend
    %>
    </table>
    </center>
```

```
<%rs.MoveFirst%>
<%while not rs.eof
cr=RS.Fields("Grade")
        if cr="AA" then       credit=4.0 end if
        if cr="BA" then       credit=3.5 end if
        if cr="BB" then       credit=3.0 end if
        if cr="CB" then       credit=2.5 end if
        if cr="CC" then       credit=2.0 end if
        if cr="DC" then       credit=1.5 end if
        if cr="DD" then       credit=1.0 end if
        if cr="FD" then       credit=0.5 end if
        if cr="FF" then       credit=0.0 end if

Toplam1=Toplam1+RS.Fields("CreditHr")
Toplam2=Toplam2+credit*Rs.fields("CreditHr")
rs.movenext
wend%>
<%ave=Toplam2/Toplam1%>
<center>
<%if dd=0 then%>
<p>The CGPA is:
<%end if%>
<%if dd<>0 then%>
The GPA is:
<%end if%>
<%Response.Write ave%><br>
</center>
<br><br>
<table border=0 align=left>

<tr>
        <td>First Name:</td>
        <td><%Response.Write rs1.fields("FirstName")%></td>
</tr>
```

```
<tr>
        <td>Last Name:</td>
        <td><%Response.Write rs1.fields("LastName")%></td>
</tr>
<tr>
        <td>Address:</td>
        <td><%Response.Write rs1.fields("Address")%></td>
</tr>
<tr>
        <td>Phone Number:</td>
        <td><%Response.Write rs1.fields("PhoneNumber")%></td>
</tr>
<tr>
        <td>Email Address:</td>
        <td><a href="mail to:<%Response.Write
rs1.fields("Email")%>"><%Response.Write rs1.fields("Email")%></td>
</tr>
<tr>
        <td>Class:</td>
        <td><%Response.Write rs1.fields("ClassID")%></td>
</tr>
<tr>
        <td>Sex:</td>
        <td><%Response.Write rs1.fields("Sex")%></td>
</tr>
<tr>
        <td>Department:</td>
        <td><%Response.Write rs1.fields("Department")%></td>
</tr>
</table>

</BODY>
</HTML>
<%end if%>
```

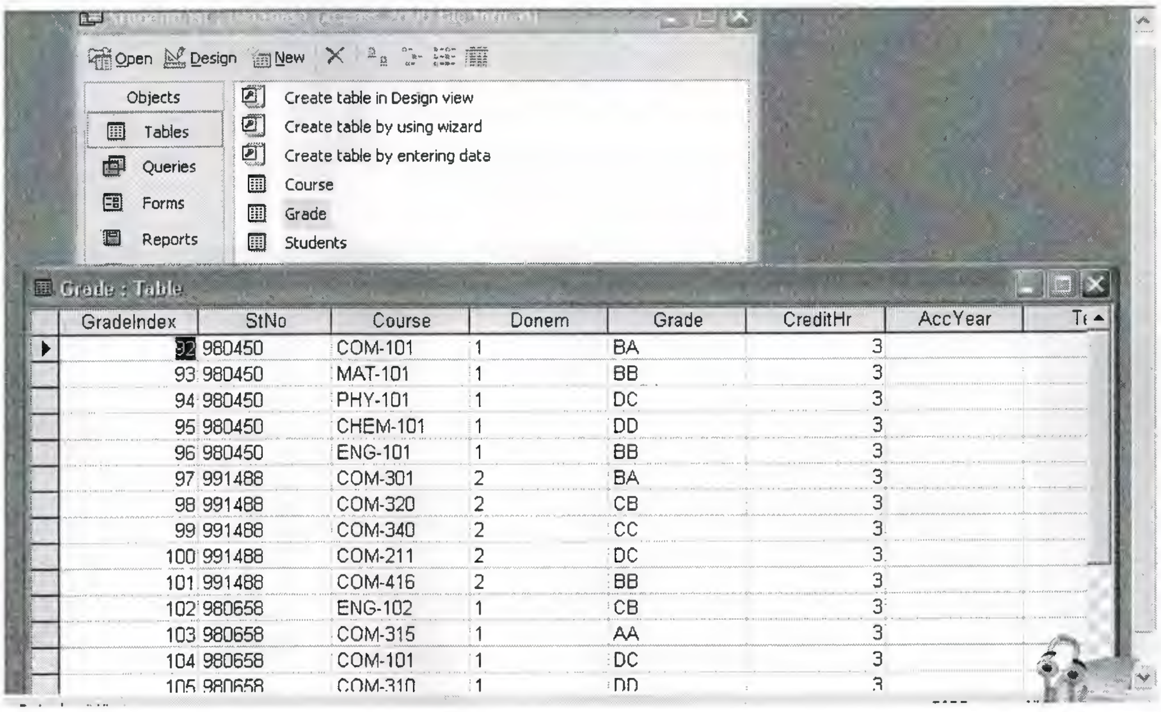
STEP 7: E- MEAL // HTML

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
<TITLE></TITLE>
</HEAD>
<BODY bgcolor="#CCFFFF">
<center>

<h1><u><font color=green>
Welcome to Mailing List of NEU.</font></u></h1>
<br><br><br>
<h2><font color=blue>Please Select the Name to Send E-Mail...
</h2></font>
<br><br>
<a href=mailto:"ozan_sdn@yahoo.com">Ozan Durmaz</a><br><br>
<a href=mailto:"uilhan@hotmail.com">Mr. Umit Ilhan</a>
<br><br><br><br>
<input type=button value="<<< Back" onClick="{history.back()}">
</center>

</BODY>
</HTML>
```

APENDIX B:

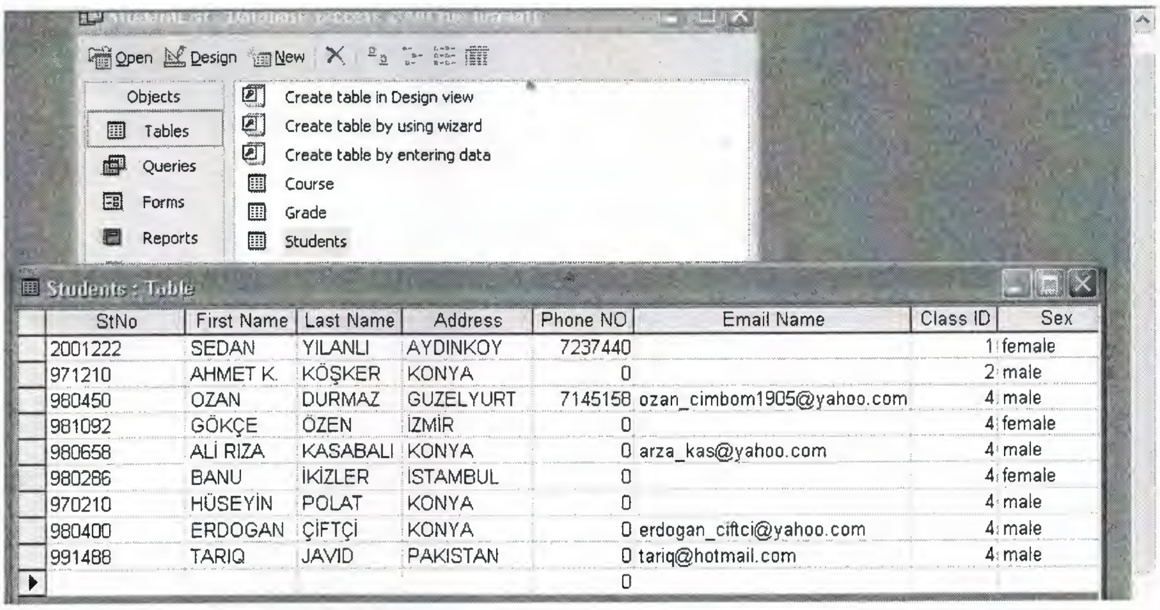
TEST RESULTS:

DATABASE 1:COURSE REGISTRATION.


Open
Design
New
Objects
Tables
Queries
Forms
Reports
Create table in Design view
Create table by using wizard
Create table by entering data
Course
Grade
Students
Grade : Table

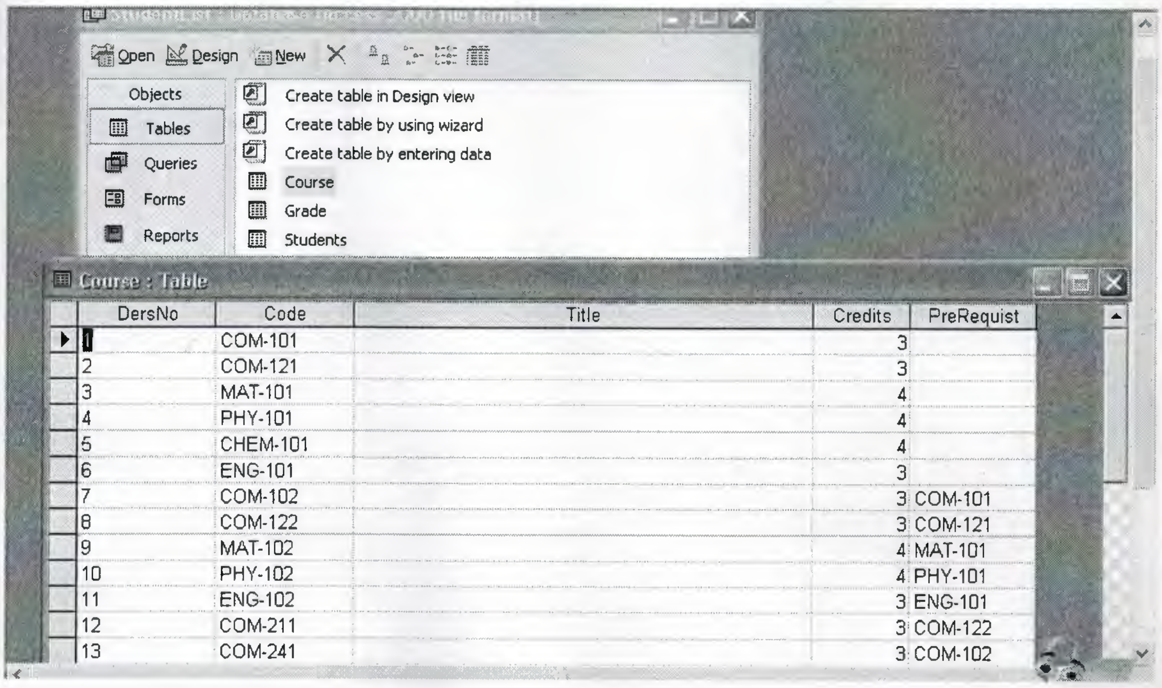
GradeIndex	StNo	Course	Donem	Grade	CreditHr	AccYear	T
92	980450	COM-101	1	BA	3		
93	980450	MAT-101	1	BB	3		
94	980450	PHY-101	1	DC	3		
95	980450	CHEM-101	1	DD	3		
96	980450	ENG-101	1	BB	3		
97	991488	COM-301	2	BA	3		
98	991488	COM-320	2	CB	3		
99	991488	COM-340	2	CC	3		
100	991488	COM-211	2	DC	3		
101	991488	COM-416	2	BB	3		
102	980658	ENG-102	1	CB	3		
103	980658	COM-315	1	AA	3		
104	980658	COM-101	1	DC	3		
105	980658	COM-310	1	DD	3		

DATABASE 2: NEW STUDENTS LIST.


Open
Design
New
Objects
Tables
Queries
Forms
Reports
Create table in Design view
Create table by using wizard
Create table by entering data
Course
Grade
Students
Students : Table

StNo	First Name	Last Name	Address	Phone NO	Email Name	Class ID	Sex
2001222	SEDAN	YILANLI	AYDINKOY	7237440		1	female
971210	AHMET K.	KÖŞKER	KONYA	0		2	male
980450	OZAN	DURMAZ	GUZELYURT	7145158	ozan_cimbom1905@yahoo.com	4	male
981092	GÖKÇE	ÖZEN	İZMİR	0		4	female
980658	ALİ RIZA	KASABALI	KONYA	0	arza_kas@yahoo.com	4	male
980286	BANU	İKİZLER	İSTAMBUL	0		4	female
970210	HÜSEYİN	POLAT	KONYA	0		4	male
980400	ERDOGAN	ÇİFTÇİ	KONYA	0	erdogan_ciftci@yahoo.com	4	male
991488	TARIQ	JAVID	PAKISTAN	0	tariq@hotmail.com	4	male
				0			

DATABASE 3: COURSE LIST.


Open
Design
New
Objects
Tables
Queries
Forms
Reports
Create table in Design view
Create table by using wizard
Create table by entering data
Course
Grade
Students
Course : Table

DersNo	Code	Title	Credits	PreRequist
1	COM-101		3	
2	COM-121		3	
3	MAT-101		4	
4	PHY-101		4	
5	CHEM-101		4	
6	ENG-101		3	
7	COM-102		3	COM-101
8	COM-122		3	COM-121
9	MAT-102		4	MAT-101
10	PHY-102		4	PHY-101
11	ENG-102		3	ENG-101
12	COM-211		3	COM-122
13	COM-241		3	COM-102